International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 10 Issue 5 ǁ 2022 ǁ PP. 139-142

Software Reuse: Extraction from Repository

Sheleshma Shukla[#1], Dr. Dhirendra Pandey[#2]
[#1,2]*Department of Information Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, Uttar Pradesh*
Corresponding Author: Sheleshma Shukla

Abstract
The best way to cut development cost of a software is to reuse some parts previously developed software by extracting the executable components to produce a more flexible software as compare to previous version. In this paper the best practices of searching and selecting components from repository has been analyzed.

Keywords: *Repository, Reuse, Extraction, Flexibility, Knowledge Collaboration.*

Date of Submission: 15-05-2022 Date of acceptance: 30-05-2022

I. INTRODUCTION

Setting up the criteria for reusing component which are already by some other developed software is the core part of selection from the software repository .We can set some boundary of choosing it in a best way, like by focusing on requirements, designing, code, test case and the knowledge (experience of a developer can be used as a key guideline).This paper is an aspect of software reuse by the core analysis of repository. Repository is like a directory consisting project files and folder having lots of used information, documentation and code. Repository can be public or private. Software repositories are a treasure of information and give a distinctive view of the actual evolution for a particular software system. Software engineering researchers have presumed number of approaches to extract this information. We can access repository directly or from a specific databases, files or documents are obtained for distribution in network. In empirical software engineering research repository analysis is very relevant for collecting data. Researchers choose repositories which fulfill the specific criteria, extract data from it and analyze the data for confirmation of research questions.

II. LITERATURE REVIEW

2.1 Meeting up the Requirement

Being a software Engineer, writing the code is not just a responsibility it also include large number of many other roles. Meeting up the customer satisfaction is one of the biggest task for an engineer. Just listening a term requirements, sounds very easy to gather it .But reality is not same. Starting practicing into the idea of software requirement, you will get that it comprise extensive understanding and skills. The tasks include the way of gathering the requirements, Keeping in mind the needs and priorities of an organization, Focusing the real requirement for users of the software, Consider the technical limitations required, You must know the duration when it will be done, last but not the least you also have to look the implementation should match with a set criterion. Organization's requirements process the requirement enterprise which is the responsibility of a software engineer. Gathering right requirement is necessary to verify the implementation. Area consist of:

- Elicitation – requirement gathering
- Classification – requirement categorization
- Validation – requirement confirmation
- Development & Implementation - building the software on basis of requirement
- Negotiation - dealing with conflicts due to grip
- Verification - software function evaluation to meet the requirement

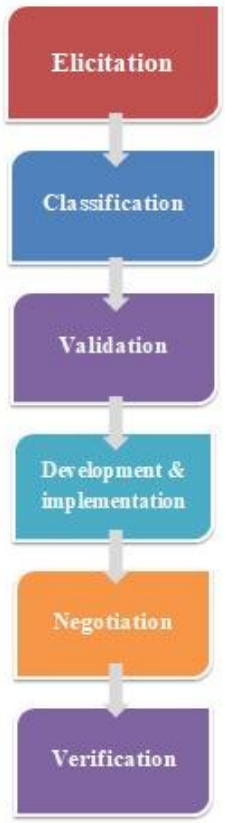

Figure 1: Requirement Analysis Process

2.2 Art in Designing

Art of designing is converting the requirements in some flexible form to provide ease in software coding and implementation. So that customer requirement based document is prepared. Coding, role of each module, scope of the classes and the functions goals are the main parts of software designing. All structures comprises the code parts, relations among them and characteristics possess by each parts and relations. System design states the system organization or structure and gives a demonstration about its behavior. Design selection series are the impression of quality, compliance, degree to which application enhanced and overall success of the system. A system shows the elements gathering which effectuate functions set. Code designing represents the essential features of run time components of a package. Components and data elements limits in their relationship to realize the desired set of properties.

2.3 Catching the code

One of the most active and innovative study fields in computing is code reuse. The use of existing software to generate new software is known as code reuse. The two most crucial variables in software development are reuse and reuse. Reuse necessitates the explicit handling of compilation, packaging, distribution, installation, and configuration issues. Implementation, upkeep, and revision. To keep up with expanding software needs, programmers must reuse existing code due to its pervasiveness and necessity. Researchers have begun to look at the psychological processes and programmer qualities that influence code repetition. Programmers were given 18 pieces of computer code incorporating transparency and reputation manipulations after completing personality surveys on predisposition to trust and skepticism. The findings showed that trustworthiness has little impact on the desire to reuse code. However, different aspects of suspicion and suspicious propensity have an impact on readiness to reuse. Programmers with lower trait mal-intent judgments and higher cognitive activity reported being more inclined to reuse code. The implications and uses of the findings are examined.

2.4 Follow the test

Software programs are complicated products with the potential to behave in unexpected and undesirable ways. Software testing refers to the procedures that must be followed to ensure that the software is working properly. That program is trustworthy. There are benefits and drawbacks to automation testing. According to studies, the key benefits of automation in software testing are reusability and efficiency. A smaller

number of test cases and a shorter test execution time Costs are one of its disadvantages. Engaged in the purchase of tools, personnel training, and test case design. In this paper, a test case is a collection of test steps that must be completed in a certain order. Stimulus and an expected reaction make up the test phase. A test suite consists of a collection of test cases. Software testing methodologies are divided according to the information source utilized to generate test cases. Software testing may be characterized in two ways using that criterion:

There are two types of testing: black-box testing and white-box testing [1]. The tester does not perform anything using the black-box method. It ignores the inherent logical structure and concentrates on the task at hand [2]. A formal specification or a well-defined collection of parameters can be used to describe its behavior. Circumstances before and after the tester decides if the test results are correct. Those in the standard are comparable. Because black-box testing only looks at software, Functional or specification-based testing [1] is a type of testing that focuses on behavior and functionality. White-box (or glass-box) testing, on the other hand, considers how software is used.

2.5 Knowledge reuse

Software reuse can result in significant increases in software productivity and quality while lowering costs development expenses. However, expressing software reuse objectives might be difficult. A developer may want to reuse a software component but find it difficult to explain their reuse intentions in a way that is compatible with, for example, or the component retrieval system understands. There have been several intelligent retrieval algorithms created.

Aid a developer in efficiently identifying or discovering components these options have something in common shortcoming: the developer must be capable of predicting and initiating all reuse possibilities process. There is a need for a complete approach that can help with retrievals as well as other tasks like identify chances for reuse.

III. PROCESSING REPOSITORY

We need to extract the document that has reusable knowledge, what knowledge should be found out for reuse and should be extracted. The approach leads towards the most effective development of new software by reusable components. Experiences provide the clear vision of right path that results as a more flexible software with the benefits of cost reduction as well as quality improvement.

3.1 Extraction Tools:

Software reuse is becoming increasingly important in saving software development time and effort while also improving software quality and efficiency. Since the notion of reusing existing knowledge for software reuse was first proposed in 1968 [3], new vistas have emerged. The main concept is Domain engineering is the driving force behind software reuse (aka product line engineering). The degree of reusability is defined as the ability to reuse anything [4]. Reusability of software refers to the ability to reuse parts or all of it. Other systems [5, 6] are concerned with the packaging and scope of the functions that programmers are capable of [7]. According to [8] the US Department of Defense could save $300 million on its own per year by raising its reuse rate by only 1%. Furthermore, software reuse is intended to improve productivity, maintainability, and portability, and hence the finished product's overall quality [9].

3.2 A Reuse Repository

The use of a reuse repository to improve code reuse received mixed reviews. There was no clear understanding of the reuse repository. A reuse repository, according to respondents, would increase code reuse. There are appropriate approaches and procedures for using it. The fundamental premise of code repositories without the context supplied by a software product line.

3.3 Repository Reuse Issue:

A repository is a requirement for supporting software developers and other users in the development of software that may be reused. There are various publications in the literature that look into reuse repositories. However, they generally focus on reusable component search and retrieval challenges, leaving crucial elements of reuse repositories unexplored. However, certain problems addressed by businesses interested in adopting or creating a reuse repository remain unresolved. Frequently asked questions include: What are the primary functions and needs of a reuse repository? What are some viable alternatives? What should a reuse repository look like? In response to these concerns, this article proposes a systematic strategy for describing, creating, and testing with comparisons to existing tools and approaches. Compares to current tools and methodologies for specifying, planning, and implementing a reuse repository that was built and implemented successfully in actual Brazilian software factories. We also go through the major design decisions, issues that were discovered, and future research and development directions.

IV. CONCLUSION AND FUTURE WORK

Better representation techniques, including means for definition and verification, are required for all software assets. Researchers emphasize the need for assistance and support. The component behavioral contract requirements are strictly enforced. The capacity to foresee future asset variability is a critical component of reuse success. Since the late 1960s, researchers have been studying reuse. Although much has been accomplished, there is still plenty to be done before the deadline.

ACKNOWLEDGEMENT

I am grateful to my guide Dr. Dhirendra Pandey for their support in bringing out this paper successfully.

REFERENCES

[1]. H. V. Vliet, Software Engineering: Principles and Practice. John Wiley and Sons Ltd, Jul. 2008, 740 pp., isbn: 0470031468 (cit. on p. 6).
[2]. P. Ammann and J. Offutt, Introduction to Software Testing. Cambridge University Press, Jan. 2008, 322 pp., ISBN: 9780521880381 (cit. on p. 6).
[3]. Gomes. P and Bento.C, "A Case Similarity Metric for Software Reuse and Design," Artif.Intell.Eng. Des.Anal.Manuf., Vol. 15, pp. 21-35, 2001.
[4]. Frakes.W and C.Terry, Software Reuse: Metrics and Models, ACM Computing Surveys, Vol. 28, No. 2, June 1996.
[5]. Mccall,J.A et. al., "Factors in Software Quality," Griffiths Air Force Base, N.Y. Rome Air Development Center Air Force Systems Command, 1977.
[6]. Gill.N.S "Reusability Issues in Component-Based Development," SigsoftSoftw. Eng. Notes, Vol. 28, pp. 4-4, 2003.
[7]. Gaffney,J.J.E "Metrics in Software Quality Assurance," Presented at the proceedings of the ACM '81 Conference, 1981.
[8]. Poulin, J. S., 1997. Measuring Software Reuse–Principles, Practices and Economic Models, Addison-Wesley
[9]. Sharma.A, et al., "Reusability assessment for software components," SigsoftSoftw. Eng. Notes, Vol. 34, pp. 1-6, 2009