IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

# An In-Depth Exploration Of Reusability Characteristics In Modern Software Engineering Practices

Mithila Chavan, Assistant Professor, Vidyalankar School of Information Technology, Mumbai

## Abstract:

One of the most important aspects of software quality is reusability. The concept of software reuse has been put forth for more than a couple of decades. Because it can shorten delivery time to market, improve quality, and cut costs, software reuse is regarded as a very important factor for software development. Due to the demand, software companies began to consider reusability of software assets that might also be used to address persistent issues. As software companies discovered that reusing source code alone does not result in cost savings, they expanded the idea of software reuse to include other life cycle elements. The maintenance aspect is another crucial one for software reuse. Review for reusability in software engineering is done in the paper by reading several test cases, going over real-world instances, and consulting many research articles.

**Key words**: Metrics, Models, Methods, Approaches, Software, Reuse, Reusable Assets, Value, Measuring, Maintenance.

## Introduction:

Software engineering method known as reuse-based software engineering "is one where the development process is focused on reusing existing software." Reusability has a wide range of benefits, including decreased time and cost, improved software quality, and lower implementation costs[3].

Software reuse is the practice of building a new system using an existing one rather than starting from scratch. In other words, it involves leveraging software assets or artefacts that already exist to create a new system. In 1968, McIlroy developed the idea of software reuse as a means of resolving the issue of developing sizable, trustworthy software systems in a manageable and cost-effective manner[5].

A measure of a feature of software artefacts or their specifications is referred to as a software metric. Software metrics can be used to measure software quality, assist project managers in assessing and managing the development lifecycle, and support software engineers in assessing the software's quality. Measuring reusability in object-oriented (OO) paradigms is an example of how metrics are used[3].

The cost of reusable software directly affects maintenance costs for software development. Reusability is a highly important requirement in software engineering; hence the topic is taken in to consideration for research [5].

Three basic principles make up software reuse rules[1]:

(1) The object first needs to be reusable,
(2) next it needs to be valuable for the reusable object,
(3) and last the user needs to be fully aware of how to utilise it.

This sector is still in its initial stage. According to several sources, research into software reuse began in 1968, but most of the questions are not yet addressed [5].

# Methods:

To obtain an answer to the research questions, search methodology for a systematic mapping investigation has been used. This entails examining, locating, assessing, and interpreting each research study that is pertinent to the specific research question, topic, or phenomenon of interest.

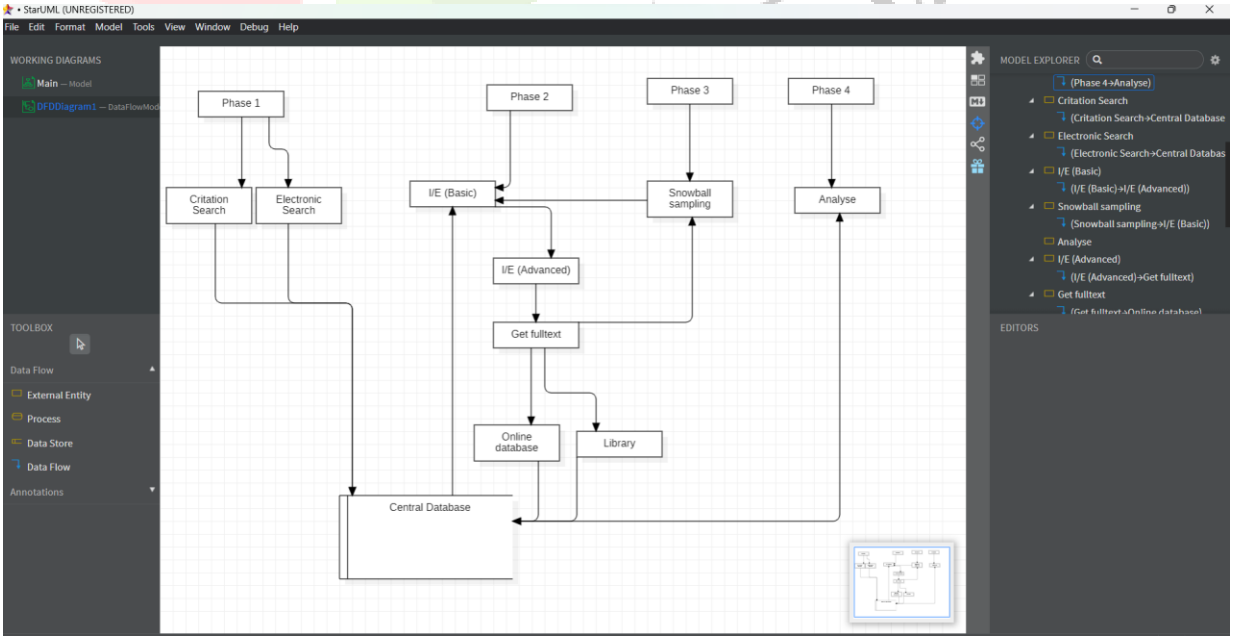
The search methodology used for our systematic mapping study is shown in figure. There are four stages to the search strategy[5]:

Phase 1: The first step is to conduct an electronic and citation-based search. Choosing search phrases and search questions is part of this process. The fact that the search results, including the papers that were accepted and refused, are stored, and documented makes the procedure simple. The study questions, the issue area, and the phenomena are used to frame the search phrases and search questions.

Phase 2: In this phase, inclusion and exclusion criteria are put into practise. These criteria are described in parts.

Step 3: The third phase differs differently from the first and second phases. The third phase is distinct from the first and second phases in some ways. In the third step, snowball sampling was conducted, which was a little different from the Kitchenham's recommendations.

Step 4: In this phase, the studies gathered in the previous three phases are analysed. The studies for each phase are stored and retrieved using a central database.




**Search Strategy**

## The following are some steps for estimating the costs of reuse[23]:

1. Cno-reuse = development cost without reuse
2. Reuse Level, R = total size of reused components / size of application
3. F_use = relative cost for the reuse of a component
4. typically, 0.1 - 0.25 of development cost is considered.
5. Cpart-with-reuse = Cno-reuse * (R * Fuse)
6. Cpart-with-no-reuse = Cno-reuse * (1 - R)
7. Cwith-reuse = Cpart-with-reuse + Cpart-with-no-reuse
8. Cwith-reuse = Cno-reuse * (R * Fuse + (1 - R))

## Another model of estimating the cost of reuse is that we can categorize the type of reuse in the context of cost estimation as follows[24]:

A) Component Reuse without Modification

B) Component Reuse with modification

(1) In the case of component reuse without modification, the average cost of developing using reusable components can be formulated as follows:

Cost search + (1-p) * Development no-reuse

(2) Where Cost search is the cost of performing a search operation, Development no-reuse is the cost of developing without reuse (i.e., the cost of developing the component from scratch) and p is the probability that the component is found in the component library. It is observed that the reuse option would be preferable only if:

Cost search + (1-p) * Development no-reuse < Development no-reuse

(3) In the case of component reuse with modification, the average cost of developing using reusable components can be formulated as follows:

Cost search + Cost adapt + (1-p) * Development no-reuse

## Methods to ensure Quality of reusable code:

Ways to measure security of a software system are:

**Software Quality Indicators & Test Metrics**

| | |
|---|---|
| Reliability | Number of failures, calendar time |
| Performance efficiency | Load testing, stress testing, response time |
| Security | Time to fix failures, number of error messages |
| Maintainability | Lines of code |
| Rate of Delivery | Number of "stories" |
| Testability | Number of technologies needed to test, quality of documentation |
| Usability | Completion rate, satisfaction level |

[17]

The quality of software cannot be checked and improved alone through testing. When assessing software, it's crucial to employ top-notch test metrics. The effectiveness of any software testing effort is gauged by test metrics. Errors are more likely to creep into production without the proper test measurements.

When the primary factors affecting software quality are thoroughly tested, together with applying test metrics to make sure the testing effort is successful, high-quality software is the result.

Large enterprises find it difficult to maintain track of both automated and manual tests; in today's development teams, a central dashboard that coordinates all software analysis efforts and pertinent test metrics is crucial for producing high-quality software[3].

### How can we measure and assure reliability?

Think about the many sorts of proof that might back up reliability claims made prior to operations. In practise, it will be required to use a variety of types of data to back up reliability claims, especially where high levels of reliability need to be guaranteed. It is a challenging challenge in and of itself to combine such divergent facts to support decision-making, and this is a current study area[19].

## Literature Review:

The ideas of reusability, maintainability, security, quality, risk assessment, dependability, reliability, and availability are the key topics of our literature review, and they have been extended in a question-and-answer format while reading numerous publications.

### What is reusability ?

One of the buzzwords used in software development the most frequently is "reusability." Recently, many technologies—including React—have begun to outfit themselves with this feature. Component reusability is now among the most often used features in UI design[26].

The capacity of a component to be reused in a different environment is a crucial trait to utilise in evaluating the quality of components.

### How does software quality impact reusability?

1) Security: Before reusing code, it's crucial to make sure it doesn't have any internal vulnerabilities. Reusing internal code is often safer than reusing code from outside sources[17].

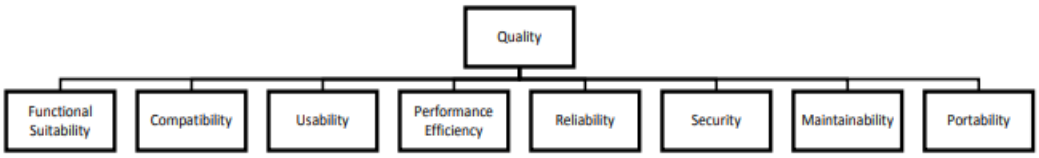
2) Reliability: Reusable code must be dependable. You may ensure trustworthy and dependable code by assuring availability, fault tolerance, and recoverability[17].

3) Performance effectiveness: Code reusability is only valuable if the implemented code is effective[17].

4) Maintainability: It's crucial to make sure that the code being reused is robust and simple to maintain[17].

### What role does security play in the quality of software reuse?

In many software-intensive systems, security is a crucial and challenging quality aspect. Unfortunately, throughout the requirements phase of the development life cycle, security is frequently overlooked. When security is added later, during design and implementation, it leads to insufficient analysis, cost overruns, and vulnerabilities that result in annual costs of billions of dollars. Even if security criteria are established, they are either too general to be useful or at an unsuitable level of abstraction. Excessive attention to design considerations. Security should be included into system development from the outset and handled with the same rigour as other system requirements for maximum effectiveness[15].

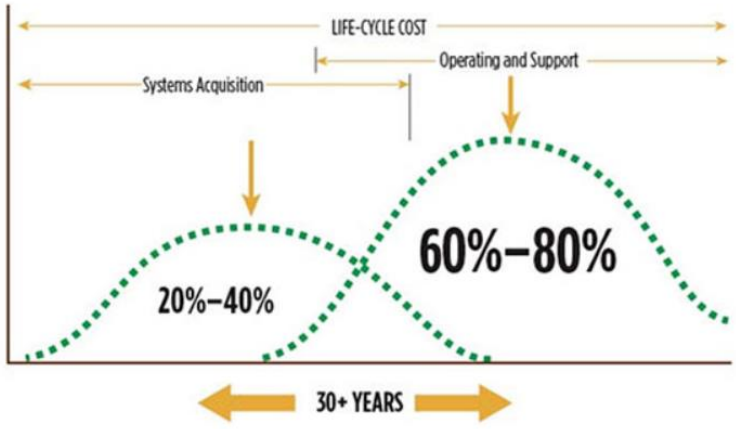



ISO 25010 Software Quality Characteristics[11]

## What role does maintainability play in the creation of software that is reused?

The convenience, precision, safety, and economy of maintenance chores within the building system are all taken into account during design in order to ensure maintainability. Enhancing the effectiveness and efficiency of maintenance is the goal of maintainability. Even after accounting for inflation, the expenses of sustaining a software system eventually surpass those of its initial construction[18].

According to some estimates, maintenance expenses make up 60% of a system's overall cost. Corrective, adaptive, perfective, and preventive maintenance are the four different types. According to estimates, the four classes make up 21%, 25%, 50%, and 4% of the maintenance expenditure, respectively. Perfective maintenance adds new features and enhances existing components of the system, while corrective maintenance removes faults, adaptive maintenance modifies software in response to changes in the processing environment or the application domain, and preventive maintenance restructures a software system to make future maintenance easier[20].




**Credit: Defense Acquisition University website**

Studies show that operation and maintenance (O&M) costs are greater than three times the cost of initial construction and that O&M costs can equate to 60% to 80% of all life cycle costs. [18]

**Cost estimation graph of Maintainability**

## What is the impact of dependability on the development of software reuse?

A design strategy based on engineering must include dependability issues. Although development in this area has been modest in the software industry, it is essential for society and individual actors to make more effective decisions. The costs of being unreliable or failing to match dependability to needs rise as software dependence increases. Some current trends, such the move towards employing more commercial off-the-shelf components, present opportunities as well as technical hurdles for this development[19].

In terms of user and developer education and improved technical community communication, there are non-technical challenges to be overcome. Learning more about the effects of the methods for establishing dependability and improving knowledge organisation to enhance judgement and decision-making are two research topics[19].

### How can software reusability risk be assessed?

In case the source code of a reused software system or component is not present then upholding cost may be increased as the reused elements of the system may become increasingly incompatible with system changes. A danger to software reusability arises when a legacy system has used software components that, over time, are no longer readily available or may be prohibitively expensive. We need to move away from our legacy system for a variety of reasons, including the fact that patching is infrequent because the system is "out of support" with the vendor, that the system is excessively dependent on a single person for management, and that there are other functional restrictions[8].

## Observations:

According to T. Capers Jones'[22], if current reusability trends continue, commercial programming systems may contain as much as 50% reused logic and code by the year 1990. Before reusable code becomes a reliable fundamental technology, however, significant progress will need to be made in the domains of reusable data, reusable structures, and reusable design.

Eleven different requirements reuse methodologies are documented in the literature, according to the analysis of 69 primary studies that were discovered using the snowball sampling method. The most popular method for improving the artifact's reusability is to structure the requirements. Other noted ways that have been studied include analogy, matching similar criteria, parametrizing the requirements, and combining various approaches. These methods are based on several sorts of reusable requirements, such as textual form, abstract needs, use cases, etc[14].

It was discovered that just a portion of the identified techniques had undergone industry-wide security evaluation. 22 studies—15 case studies and 7 experiments—were assessed in terms of data collection, measurement of variables, reuse and security strategy, and key takeaways[14].

It was discovered that industry evaluations of text-based requirement reuse techniques are common. It was noted that empirical research solely explored the benefits of requirement reuse and gave no consideration to the drawbacks of various methods. The most popular technique for the analysis of requirements is document examination. Since context and validity threats are rarely discussed in studies, it was challenging to generalise the findings of the examined investigations[14].

## Challenges faced:

1) Communication: If there is a significant communication gap, and it gets harder to achieve code reuse and communicate effectively with all the developers engaged as the project's scope grows. It becomes more challenging to discuss with the entire team to identify project areas where code may be reused and to properly communicate the characteristics and parameters for code reuse[25].

2) Legal considerations, Warranty, Open Source, Copyright and Reuse price estimation.

4) Increased dependability: The programme needs to be incredibly reliable[13].

5) Inadequate tool support : Every organisation must offer new tool assistance to the programmers and developers as well as specialised training on how to use these tools efficiently and effectively. These organisations must offer organised tool manuals to their developers[13].

6) Maintenance cost increase : As the systems become older their maintenance cost increases especially if the systems can't be updated[13].

# Conclusion :

Software reuse is the process of developing a software piece or system that uses previously produced software parts or components.

A system's or product's software components require time to develop. Utilizing code fragments created for earlier applications could thereby speed up development.

An organised and modular programming technique was probably employed to design software with reusability in mind.

This enables the elements that carry out a certain function to include a self-contained block of code. It takes in certain inputs and outputs of the required results. One type of software that may be incorporated into numerous apps and printing tools is a printer driver.

The following are the main findings of our paper on reuse in software development:

1. It boosts productivity.
2. It lowers expenses.
3. And it raises overall quality.
4. The practice of reusability in software development is very common and effective.
5. Reusability helps to save time, and development risks are decreased as the codes or components are already tried and tested.

# References:


(1) Software Reusability: Approaches and Challenges (rsisinternational.org)
(2) A survey on Software Reusability | IEEE Conference Publication | IEEE Xplore
(3) Software Reusability Classification and Predication Using Self-Organizing Map (SOM) (scirp.org)
(4) (PDF) Software Reuse: Research and Practice (researchgate.net)
(5) FULLTEXT01.pdf (diva-portal.org)
(6) Current Issues in Software Re-Usability: A Critical Review of the Methodological & Legal Issues (scirp.org)
(7) (PDF) Software Reuse: Research and Practice (researchgate.net)
(8) A Study on Risk Assesment in Software Design Reusability – IJERT
(9) Software Reuse: Issues and Research Directions by Yongbeom Kim, Edward A. Stohr :: SSRN
(10) Reusing software: issues and research directions | IEEE Journals & Magazine | IEEE Xplore
(11) Seang_Cau_200222569_MASc_SSE_Spring2017.pdf (uregina.ca)
(12) Phases of an SLR according to Kitchenham et al. (2016) | Download Scientific Diagram (researchgate.net)
(13) Software Reuse Considerations - Accendo Reliability
(14) (PDF) A Systematic Literature Review of Software Requirements Reuse Approaches (researchgate.net)
(15) Security Requirements Reusability and the SQUARE Methodology (cmu.edu)
(16) PDF_Writing_a_Literature_Review_Paper.pdf (sjsu.edu)

(17) Software Quality Metrics: How to Measure Testing Efforts | AltexSoft
(18) Design for Maintainability: The Importance of Operations and Maintenance Considerations During the Design Phase of Construction Projects | WBDG - Whole Building Design Guide
(19) LittlewoodStrigini.pdf (umass.edu)
(20) Maintainability and Reusability | SpringerLink
(21) Reusability in Software Development | by Bien VO | Dwarves Foundation | Medium
(22) Reusability in Programming: A Survey of the State of the Art | IEEE Journals & Magazine | IEEE Xplore
(23) 09-reuse.pdf
(24) (PDF) Cost Estimation Model For Reuse Based Software Products (researchgate.net)
(25) Requirements are slipping through the gaps — A case study on causes & effects of communication gaps in large-scale software development | IEEE Conference Publication | IEEE Xplore
(26) Software Component Reusability - Cuelogic An LTI Company