

# An Analysis of Software Reusability implementation based on Matrix Approach using Machine Learning

<sup>1</sup>. Sandhya

Ph.D Research Scholar  
Department of Computer Science & Applications  
NIILM University Kaithal, India

<sup>2</sup>.Dr.Mukesh Kumar

Professor, Department of Computer Science & Engineering  
NIILM University Kaithal, India

**Abstract:** Software engineering is an application of engineering which is more focused on original development, but reusability plays a very significant role in order to produce good quality, error free, and less maintainable software. Software reusability is an attribute of quality which helps in selecting beforehand acquired notions in new statuses. Software reusability not only advances productivity, but it also provides good quality software and has also optimistic effect on maintainability. Software reusability is advantageous in the manner that it provides high reliability, low cost of maintenance, and reduction in development time. In this paper, we have discussed and analyzed various machine learning techniques used for estimation of software reusability. It is found that machine learning techniques are competitive in nature with other reusability estimation techniques and can be used for estimation of reusability. This study will help software developers and information industry to elucidate that how software reusability can assist them in selecting advanced quality of software.

Date of Submission: 18-09-2024

Date of acceptance: 02-10-2024

## I. INTRODUCTION

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. This simple yet powerful vision was introduced in 1968. Software reuse has, however, failed to become a standard software engineering practices . The basic purpose of software reuse practice is to design repositories which help in accurate and efficient way of finding and retrieving software components/artifacts. The success of component-based software requires potential customers reusing from some knowledge space . Because of certain technical, organizational, and psychological software engineering research issues and trends,Software reuse is still evolving as a major research discipline..Generally reuse is confused with redesign or recoding but there is a difference between them.Reuse in software engineering was proposed at the same conference as the term software engineering was introduced . So reuse and SE have co-existed for thirty years. But even before this conference Wilkes recognized the need to build a library of mathematical subroutines in 1949 to avoid having to rewrite them . In 1968 McIlroy introduced the idea of reusable components. McIlroy's components were comparable to standard, off-the-shelf components that are used in computer manufacturing. McIlroy's components would have been built by dedicated component factories. His components were source code components and were to be used in their original form in other programs. Unlike in other engineering disciplines, SE has only succeeded moderately, at the best, in reuse .

### 1.2 TYPES OF REUSABLE ASSETS

- Code
- Requirements
- Architecture/design documents
- Test Plans
- Manuals
- Templates for any asset

- Design decisions

## **II. SOFTWARE REUSE TECHNIQUES**

### **2.1 Black-Box Reuse**

Reusing a component as a black box means using it without seeing, knowing or modifying any of its internals. The component provides an interface that contains all the information necessary for its utilization. The implementation is hidden and cannot be modified by the user. Thus reusers get the information about what a component is doing, but they do not have to worry about how this is achieved. The implementation can be changed without any effects on users.

Usually a black box is reused as-is. Object-oriented techniques allow modifications of black boxes by making modifications and extensions to a component without knowing its internals.

### **2.2 White-Box Reuse**

White-box reuse means reuse of components of which internals are changed for purpose of reuse. White box are typically not reused as is, but by adaptation. They create more opportunities for reusers due to the ease of making arbitrary changes. On the negative side of white box reuse, it requires additional testing and costlier maintenance. Unlike black boxes, a component and thoroughly tested. Additionally, the new component requires separate maintenance. If many copies of a component exist with slight modifications, it becomes burdensome to fix errors that affect all of them. If the changes made to a component are only minor, e.g., a few variable renaming or changes in procedure calls, the term Grey-box reuse is also used.

### **2.3 Glass-Box reuse**

The term glass-box reuse is used when components are used as-is like black boxes, but their internals can be seen from outside. This gives the reuser information about how the component works without the ability to change it. But this information may be crucial for understanding how certain tasks are carried out. It may also give the reusers some confidence from being able to see inside the component and capture how it works. Additionally, getting internal information provides some kind of knowledge transfer and, for example, can help in building new components.

Glass box reuse has its negative sides. It may lead to dependencies on certain implementation details that become fatal when the internals of the component are changed. Unfortunately, giving reusers detailed information about a component's internals often serves as compensation of nonexistent or insufficient documentation.

### **2.4 Generative Reuse**

Generative reuse is itself a reuse technique, but it can be seen as kind of black-box reuse. Instead of picking of several existing black boxes, a component's specification is created and its implementation automatically generated by a program generator. The program generator is a black box; its internals are of no interest to the reuser. Also, the generated implementation will not be modified. If changes are necessary, they will be made in the specification and the implementation is recreated. (In practice, however, modifications to the generated implementation are sometimes made due to shortcomings in the generator.)

## **III. SOURCES OF RESUABLE SOFTWARE COMPONENTS**

Licensing and ownership becomes confusing when software components are sold in a market. Software Assets can be acquired through any of the following methods:

- (i) With the utility asset library supplied with the operating system, compiler or application development environment
- (ii) Through purchase from a company or broker, usually without source.
- (iii) With source code, from an Open Source community
- (iv) From an internal company repository.
- (v) Through "mining" of legacy code or previous versions of products in the product-line.
- (vi) Through a swap, barter, or purchase agreement or contract with another internal group.
- (vii) Created as part of the regular product or application development process, then packaged and published to a corporate or local repository.
- (viii) Proactively defined to meet the needs of new projects, then developed prior to an application development project, or as a sub-project.
- (ix) Through the evolution, reengineering, or wrapping of existing assets obtained from another source, in order to be more appropriate to current and future projects.
- (x) Created by a distinct component development and maintenance team that develops components and Web services only for release to others.

## **IV. REUSE APPROACHES OF SOFTWARE**

There are different ways or approaches a reuse process is executed in organizations.

### **4.1 Proactive and Reactive Approach**

There are two basic investment approaches for software reuse:

Proactive investment for software reuse is like the waterfall approach. This approach tends to require a large initial investment, and returns on investment can only be seen when products are developed and maintained. Reactive investment is an incremental approach to asset building. One develops reusable assets as reuse opportunities arise while developing products.

The fundamental obstacle preventing spread of controlled vocabularies based reuse systems is that the extent of digital libraries, open sources libraries and the web precedes the ability of any one authority to use traditional methods of metadata creation and indexing. While metadata creation is valuable and indispensable within a particular domain or project, it can be costly to implement and can present significant scaling difficulties. Advances in research of automatic metadata generation applications are increasing. It is found by some researchers that such gains in efficiency, were they to be achieved, would allow information professionals to dedicate their efforts on those resources and intellectually demanding metadata activities (i.e. assigning controlled index terms). Until such time automatic applications are fully realized, describing or indexing the amount of information available on the web, Intranet or previous repositories will remain beyond the scope of any one authority.

### **4.2. Centralized and Distributed Approach**

There are basically two different process approaches for a reuse programme in an organization:-

The centralized approach typically has a separate department or organizational unit that is dedicated to developing, distributing, maintaining and providing training about reusable assets. With the distributed approach a reuse program is implemented collaboratively by projects. But most of the times the projects with same product line fall in this category.

#### **Advantages of distributed approach**

- (i) There is less overhead cost, as there is no need to create a separate department.
- (ii) Asset development costs are distributed among projects. No large up-front investment is necessary.

#### **Disadvantages of distributed approach**

- (i) It may be difficult to coordinate asset development responsibilities if there is no common for the reuse program.
- (ii) Even if there is a shared vision among projects, it may not be easy for a given project to provide a component that meets the needs of other projects
- (iii) There must be a convincing cost/benefit model to solicit active participation.

## **V. MISCONCEPTIONS RELATED TO SOFTWARE REUSE**

But even with these advantages, software reuse has not taken an integral place in enterprise software engineering policy. There are still many misconceptions related with reuse :-

5.1 Software Reuse is a Technical Problem-Research work in reuse has been focused on technical issues keeping aside the human issues and motivation for management to introduce and sustain reuse program in companies.

5.2 Artificial Intelligence will solve the Reuse Problem Artificial intelligence and latest genetic algorithms are believed to build good software retrieval system.

5.3 Reusing Code Results in Huge Increases in Productivity. There is no direct correlation between creating an efficient reuse system and increase in productivity, whatsoever except the indirect benefits

5.4 There is a myth that Building a repository of components will motivate people for reusing components.

5.5 Generally overgeneralization of software components when building software components in order to provide as much as possible functionality in a single component.

## **VI. Failures of Software Reuse**

There have been many reasons for the lack of progress in software reuse:

### **6.1 Non-Technical Reasons**

- (i) The reuse community concentrated its research to technical issue like building repositories, tools for search and retrieval of reusable artifacts. But now nontechnical factors like organization, process, business drivers and human involvement are becoming more important.
- (ii) Failures in the projects were due to not introducing reuse processes
- (iii) Lack of commitment by top management
- (iv) Many companies/teams/people are still in wrong belief that using OOP approach for setting up the repository will result in successful reuse program.
- (v) Lack of adequate tools to organize, search, and retrieve reusable modules has impeded reuse
- (vi) Value of software reuse refers to whether it is more cost effective, in terms of time, money, or personnel, to reuse software as opposed to developing it from scratch each time it is needed. Since software reuse has its own costs, e.g. location, classification, documentation, and storage, the reuse of software is not always valuable.
- (vii) The feasibility of software reuse relates to the ease with which software can be effectively reused. Feasibility is dependent on the facilities provided by a software development environment, and the ease with which reusable software candidates can be identified.

### **6.2 TECHNICAL REASONS**

In addition to this, technical issues have been observed :

1. Ability to scale the reuse for larger systems
2. How to do formal specification sufficiently to do automation of software reuse repositories
3. Reuse and Domain engineering specification need to be broadly defined .
4. Finding out the link between software reuse, domain engineering and corporate policy.
5. Role of software reuse in newer development environment
6. To quantitatively estimate the number of potential reuses
7. Reuse at run time-development of self-adaptive software

## **VII. METHODS SUGGESTED TO IMPROVE REUSE**

There are many ideas that has been proposed to improve software reuse :

- (i) Focus on achieving black-box reuse.
- (ii) Establish reuse oriented process and organization
- (iii) Adopt an incremental approach .
- (iv) Strong foundations for software component composition
- (v) Change adaptable components
- (vi) More effective reuse incentive structure

## **VIII. REUSE SCOPES**

The amount of possible software reuse depends on the degree of commonalty among applications that share software. A domain is an area of activity or knowledge. If the majority of applications are in a specific domain, a higher degree of reuse is probable than among applications across a broad range of different applications. Domain-specific reuse and general-purpose reuse are often called vertical and horizontal reuse, respectively. Developing quality software with less cost is difficult in today scenario. Software reuse seems to be one of the solutions of this ever-lasting problem. Using assets developed for one application program in another application program is the most simplistic view of reuse.

## **8. MACHINE LEARNING**

Nowadays, machine learning techniques are widely used in software reuse. Machine learning algorithms help in dealing with various problems of software engineering and can also be used in development and maintenance task. It makes software product adaptive and self-configuring as discussed by Zhang . Lounis and AitMehedine explored machine learning techniques for producing predictive models for three different quality characteristics and concluded that the accuracies obtained by machine learning models are comparable to other techniques. Artificial intelligence in SE application levels taxonomy, i.e., how AI can be applied in the field of software engineering is explained by Feldt et al. They categorize application into three facets. The

results showed the risks associated with distinct AI applications. It also helps companies in selecting particular AI technique for their software applications and also in creating strategies. Figure 3 represents the classification of various machine learning techniques used for estimation of software reusability.

### 8.1 Supervised Learning

Supervised learning trains the model on the basis of both input and output data and makes future predictions. In order to develop predictive models, it uses classification and regression techniques. Classification Techniques. Classification techniques classify the available input into categories. If the available data can be categorized or can be separated into different class, use classification algorithms. For example, whether an email is spam or genuine, it also finds application in speech recognition, face recognition and medical imaging. Regression Techniques. Regression techniques predict the output of input data in continuous form or response. If the output is in the form of real numbers or we are working with ranges, then regression techniques are used. Applications of these Software Reusability Estimation Using Machine Learning 70 techniques include electricity load forecasting, handwriting recognition, forecasting stock prices, and for showing fluctuations in power demand. Srinivasan and Fisher [9] used regression and neural networks machine learning approaches for the estimation of efforts required for software development and compared CartX and backpropagation learning methods to traditional methods for estimating efforts of software. Proposed techniques are found to be competitive with SLIM, COCOMO, and function points.

**8.2 Unsupervised Learning** Unsupervised learning trains the model on the basis of input data only and helps in finding hidden patterns and intrinsic structures. It explores the data available and gives good internal representation of data and reduces dimensions of data. Clustering. Clustering is an unsupervised learning technique. It is used for exploratory analysis of data and for finding hidden patterns in data. It finds application in market research and object recognition. On the basis of similarity characteristics, data is portioned into groups. Clusters are formed so that objects in one cluster are highly similar to each other and objects in different clusters differ.

### 9. Matrix Operations:

Machine learning algorithms involve various matrix operations. Matrix multiplication is used in linear regression, neural networks, and support vector machines. Element-wise operations like addition, subtraction, and division are performed during normalization and regularization processes.

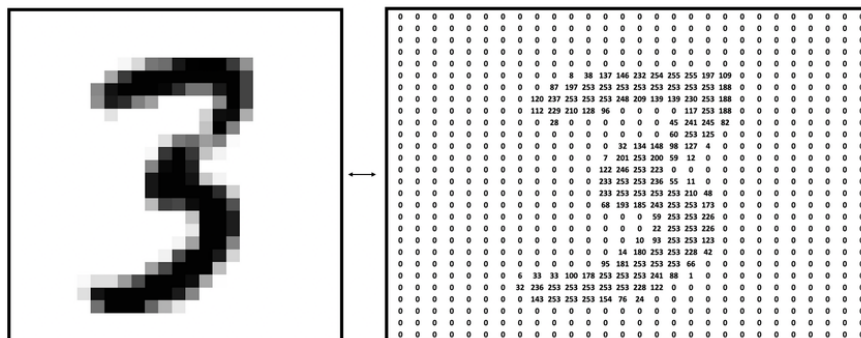
If I had to learn ML for the starting I would learn the Linear Algebra. Mathematics is the heart of Machine learning. We should have the intuitive idea how the things are happening in the background.

Matrices are an essential part of linear algebra. Linear algebra is the branch of mathematics that deals with vector spaces and linear transformations between them. Matrices provide a concise and powerful way to represent and manipulate linear transformations.

But why we should have knowledge about matrix operations in machine learning? See all the mathematical stuffs are covered by python libraries in machine learning. We don't have to explicitly calculate them. But we should have the intuition about the operations and how they are calculated.

How they are used in ML-

1. Data Representation:



In machine learning, from datasets to images all are often represented as matrices. Datasets are represented where each row corresponds to an instance or sample, and each column represents a feature or attribute of the data. This tabular representation allows efficient storage and processing of large datasets. Images are represented by their pixel density in matrix form. In grayscale images, each pixel is represented by a single value that indicates the intensity or brightness of that pixel. The matrix elements will contain these intensity values, ranging from 0 (black) to 255 (white).

2) Dot product:

## Dot Product

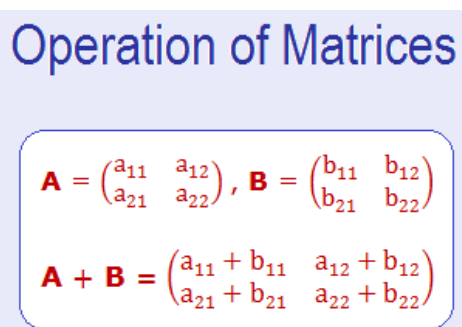
$$\begin{bmatrix} a & b \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = [ax + by]$$
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{bmatrix}$$

Dot products are extensively used in machine learning. In algorithms like gradient descent, the dot product is used to calculate the gradient of the loss function with respect to the model parameters. The dot product between the gradient and the input vector determines the step direction and magnitude in the parameter update. And the most important use is in neural networks, specifically in the computation of weighted sums in the hidden layers. Each neuron in a neural network layer receives input from the previous layer, and the dot product between the input and the neuron's weights is computed to determine the neuron's activation. You should have at least basic understanding of dot product and you will surely know what is going behind the scenes.

3) Feature Extraction:

Matrices are employed in feature extraction techniques such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD). These methods transform high-dimensional data into a lower-dimensional space using matrix operations, facilitating data compression and noise reduction. For this you should have knowledge of basis of vector.

4) Matrix Operations:



**Operation of Matrices**

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$
$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

Machine learning algorithms involve various matrix operations. Matrix multiplication is used in linear regression, neural networks, and support vector machines. Element-wise operations like addition, subtraction, and division are performed during normalization and regularization processes.



## Operation of Matrices

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

### IX. CONCLUSION

New standards are being introduced at international Level. Besides this the actual benefits of adopting reuse is generally observed after long time, therefore use of reuse is ignored by most of the organizations but using software reuse techniques is much beneficial for software industries. The management should encourage the use of reuse in their company and proper training may be made for the Software Engineers, Programmers.

- [1]. "Software Reuse Strategy: An Implementation." M.Tech Dissertation Submitted CSE Deptt., CDLU, Sirsa (2008)
- [2]. Frakes, W., Terry, C., Software Reuse: Metrics and Models, ACM Computing Surveys, Vol. 28, No. 2, June 1996, pp. 415-435.
- [3]. Boehm, B., Software Engineering Economics. Prentice-Hall Inc., Upper Saddle River, NJ, 1981.
- [4]. William B. Frakes, Kyo Kang, Software Reuse Research: Status and Future, IEEE transaction on software engineering ( July 2001), pp.: 529-536
- [5]. McClure, Software Reuse Techniques: Adding reuse to the software Development Process, Prentice Hall, 1997
- [6]. McClure, Software Reuse : A standard-Based Guide, Willey Computer society Press, 2001
- [7]. Johannes Samtinger, Software Engineering with Reusable Components, Spriner-Verlag, March 1997
- [8]. W. Frakes, R. Prieto-Diaz. A Domain Analysis support Tool, Computer Science Society, 1997, XVII International Conference of the Chilean, pp 73-77
- [9]. de Almedia, Alexandre. Information Reuse and Integration IEEE Conference 15-17 Aug 2005, Page 61-70
- [10]. Rosene, F., A Software Development Environment Called STEP, ACM Conference on Software Tools, New York, NY, April 1985.
- [11]. Lewis, T.G., Apple Macintosh Software, Software Reviews, IEEE Software, March 1985, pp. 89-92.
- [12]. Adelson, B. and Soloway, E., The Role of Domain Experience in Software Design, IEEE Transactions of Software Engineering, Volume SE-11, Number 11, 1985, pp. 233-242.
- [13]. Freeman, P., Gaudel, M.-C., Building a Foundation for the Future of Software Engineering. Communications of ACM, Vol. 34, No. 5, May 1991, pp. 31-33.
- [14]. Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Software. Addison-Wesley, 1995.
- [15]. Gilb, T., Principles of Software Engineering Management. Addison-Wesley Publishing Company, 1988.
- [16]. Abudawood, T. and Flach, P.A. (2009). Evaluation measures for multi-class subgroup discovery. In W.L., Buntine, M., Grobelnik, D., Mladenić and J., Shawe-Taylor (eds.), Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD 2009), Part I, LNCS, volume 5781, pp. 35-50.