# Apriori Algorithm Based Component Classifications and Adaptation for Software Reuse

*Sampath Korra*[1]

## Abstract

With the growth in software development, the software user expects high-quality software services with lesser time complexity. Therefore, the demand for software component based development is also increasing. In the recent past, a good number of research attempts have tried to build an automated framework for generating the recommendations for component equivalence. In software engineering and program building, reusability is the utilization of existing resources in some structure inside the product items. Adaptive software reuse products improve the life cycle of software and enhance the planning, coding, and documentation in different approaches of reusability. On the other hand, reusability can be used to meet with new software demands. Thus, the proposed novel approach to Apriori based component classifications and adaptation is used to software component reusability based on the recommendation of rule-based analysis.

*Keywords :* Adaptive, component, domain, reuse, technologies

## I. INTRODUCTION

The key benefit of Componen Based Software Engineering(CBSE) is supporting the design methods that are used for adaptive reuse. Reuse construction of the new system need not start from scratch, but it can be done with the modification of the integration and the description of the existing ones. CBSE was used to support the evolution of the components of different technologies[1]. However, nowadays it is sufficient to keep the store. Modification of the components is widely accepted as one of the main problems of the CBSE. The ability of application developers to easily adjust closed software components to function properly within their programs is necessary for the market creation of real components and the implementation of components [2]. A platform that focuses on components (such as CORBA, COM, JavaBeans, and .Net) that address common collaboration using linguistic Interface Definition Language (IDL) to identify the proposed functions (and requests) from different programs is needed.

The components of the IDL interface are important for software integration as they highlight the signature between components in the view of their specifications. However, all solutions to signature problems do not guarantee that the component will work properly. Obviously, it can happen exactly at the protocol level due to the order of the exchange messages, and it also blocks the conditions [3], i.e. due to the corresponding behavior of the associated components. In addition to tests based on the case of component compatibility, more stringent techniques are required to maximize integration of craft activities into engineering. For example, the system developer must verify that the integration of third-party components may suggest new techniques in the application being developed. To determine the properties

Manuscript Received : February 18, 2022 ; Revised : March 6, 2022 ; Accepted : March 14, 2022. Date of Publication : April 5, 2022.

S. Korra[1], *Associate Professor,* Department of CSE, Sri Indu College of Engineering and Technology (Autonomous), Sheriguda, Ibrahimpatnam, Hyderabad - 501 510, Telangana. Email : sampath_korra@yahoo.co.in ;
ORCID iD : https://orcid.org/0000-0002-3345-5293

**DOI : https//doi.org/10.17010/ijcs/2022/v7/i2/169683**

of the system that contain a large number of interactive elements, the reusable description of the interaction characteristics of the component is required [4].

Software reuse enables us to improve the reuse components of software and reduces its cost. Reuse allows us to make the characteristics of the software artifacts available, from not to build a computer system from scratch. Reuse of program means reusing the inputs techniques and outputs after the software development effort. Software industry uses different components for operations, it requires a number of reusable components to build software quickly and cost-effectively. If not, ineffective reuse of resources become unacceptable. It takes time to develop software and building repository of reusable components for use [5]. Development of repository provides the components to select when required with the assurance that at one time the relationship will protect to adapt and ensure that the corresponding application is built using the components and those should not be changed because of changes in components. A new and added functionality is that we have the ingredients to make it easier to complete the application and use of these components, and there are no changes in downtime [6].

## II. LITERATURE REVIEW

The principles of software development focus on the creation and evolution of software. The CBSE advantage is the reuse of software. It is going to adopt new techniques in a way that component classification becomes easier. The CBSE focuses on new development process of the components in a release. In the same manner, allowing for time for completion of work, the development of software proceeds quickly [7]. The part needs to change or adapt to a new system, only code changes are needed. Adaptability of the system means that the system can easily adapt to a diverse environment. The object-oriented software can easily adapt to new requirements because of the high level of abstraction. It models problems with a set of types or classes from which objects are created. It focuses on the creation and rapid evolution of the system. There is no particular step to organize in this process. There is a similar pattern of development of software and development speed. Adaptability grew up in rapid application development. In Adaptive Software Development, the overall team focuses on the problem of establishing the self-absorbed, sharing ideas, individuals, and teams online [8].

Reuse is typically categorized according to the nature of the environment in which the software components are restored. Vertical reuse occurs when software components are reused for different projects in the same domain as the application. An example of vertical reuse may be a boundary detection process used in various image processing programs. It should be remembered that the recovery of software components is done at different stages of the life cycle of different generations of the same project. Sometimes, the term vertical reuse is employed in a more limited way. Successful vertical reuse requires detailed information about the application domain [9].

The software recovery process facilitates increased productivity, reliability, quality, performance, cost reduction, effort, risk, and implementation time. Primary study is necessary to start the software recovery process, but this study is disbursement over some recovery. It creates a repository and recovery process of the knowledge base, which improves the quality after each recovery, reduces the required development work for future software development, and ultimately reduces the risks of new projects based on knowledge repository [10].

The term *reuse of software* means that there are reusable software component files that can be used as part of new system development. The most popular software for reusing software components can be advanced, such as requirements or lower-level designs, such as source code modules or related components such as test plans or documentation. Regional and application environments are relatively stable, so reuse is considered appropriate to develop projects [11][12].

Software re-engineering is a successful system as a stand-alone system but must communicate with other applications due to changes in requirements. This can happen if the control system now uses a commercially available database package to store data instead of using it as a patented package. The new interface requirements clearly indicates that the existing system needs to be modified. Re-engineering is associated with reuse of software as systems must understand in part or in full before they can be converted or reused. However, re-engineering usually requires more changes than the ones that wish to reuse the software [13][15].

Reuse is a great opportunity to improve software quality while reducing costs. It is based on the concept of reusable components and is used in the same way that electrical engineers choose system components.

Software reuse can occur at many levels of the software lifecycle [14]. Most reuse researchers believe that domain analysis is a prerequisite for successful program reuse. Domain analysis is a common system analysis whose primary purpose is to identify the operations and elements needed to determine the information used to process a particular application in a domain. In addition, domain analysis can accurately identify domains and software components in areas that are beneficial for user reuse. Ideally, anybody wants to be able to create domain-specific languages that allow writing descriptions based on important domains [16].

There are many options for high-quality experimental work that is reused in the library. The problem is that it is difficult to use parallel methods for parallel attempts to make comparisons. System reuse is a typical work and it is difficult to get more actions that are not directly related to a particular project. However, there are some research opportunities that can bring profits [17].

The lack of basic planning steps allows the development of software quickly for lifecycle of the software. The cycle of software in this process is very short for a new version. It can come with additional equipment quickly. The method or quick prototyping is the cornerstone of the development of both the software and the development of the application where the difference between the two methods is the endpoint. In the development of the software only the endpoint that did not require software code is ported to a request generation. On the other hand, rapid development application allows for the end of a job, free from the problems that have software which meets the requirements of the end-user [18][19].

Software is developed with three steps, each evolving around the coding of a program. The first step is speculation when code developers try to understand the nature of software and the needs of those who use it. This is dependent on improvements and user reports to guide the work. If there is no available report, the project uses the basic requirements set by the end-user [20]. It ensures that individual projects team members are developing and communicating via intra network for the software developments. They keep entire data at one system, then they combine the modules, it is tested and uploaded or handed over to the client. The project does not need any additional information or outside contributions and to determine the part of the software [21] during the training phase, the release of a new version of the software to use. These assets are improved and the reports are used in the first part of the project and the cycle repeats itself [22].

An additional analysis of the software component before adding it to the recovery library can lead to significant savings in the lifecycle, especially if the component is reused many times. Thus, the reusable component is obtained in combination with domain analysis [23]. Reuse library assessment requires further study before the system is included. This results in a classification of the reusable component before it is placed into the recovery library [25].

The classification of reusable software components is a logical step in the reuse process. It means a description of the component. This description is generally more complete than the description typically used for documented components that have been developed for reuse resulting in efficiency of the documents used in the feature [26].

Consider the source code of the module which is used for adding to the recovery library. Software Engineering practices require a description of the module interface and the maintenance cost of the software [27]. Many organizations use programming practices and require each source code to be tested against organizational standards. This means that every decision in the Tree module must show some results during the test. If there are no errors, the module is assumed to be correct. Suppose the same module is reused, and the application is usually different from the application being created. If the new application system is required in real-time, due to the new real-time restrictions, it is not immediately clear that the module can be used in the new system [28].

In addition to the source code, potentially adaptive reusable software components must be authenticated before being added to the reuse library. For example, all documents must be read by an independent team before reusing the libraries in the new system. The goal is to provide an independent review of the documents and avoid key inaccuracies for reuse projects. Cognitive software reuse is used to provide challenges that are needed for the success of significant management of components. The framework is used for component normalization for the platform-independent by using adaptive software reuse [24].

The classification of reusable software components should be based on atleast two factors: perceptual component corrections and some metrics describing the likelihood of component reuse. Indicators should show the number of other software systems. It is planned to use a section, difficulties include a part of other software systems and quality assessment of certain types of equipment [29]. Source code classification is the most common complementary evaluation of software components before they are placed again in libraries. Let us say that the source code is satisfactorily tested by the development of the organization and is at present considered a candidate for recovery [30].

The metrics should be about testability, easy pairing with another module, and the probability that the source code of the module is complete if it is placed in the reuse library. Portability is considered to be an ideal feature of most software. In the era of ubiquitous computing technology and rapid development, few software products are unable to implement many environments throughout their lifecycle. Storage products must use their own cost to implement as many platforms as possible [31].

Any type or scale of software can take advantage of the ability to migrate when new systems and better systems are available. Older software systems developed as part of agile software development are often poorly designed and documented but still work well in organizations of critical applications. Poor design and documentation make it difficult to reclaim the functionality of legacy systems in future projects.

## III. MATHEMATICAL MODEL OF THE PROPOSED FRAMEWORK

The first step of this proposed work is the pre-processing framework for determining and extracting the influential parameters. The most appropriate algorithm for feature or attribute selection is a Genetic algorithm, where initially all the attributes are considered as individual subsets and the final combination of the attributes or features are noted as optimal best feature subset. The framework proposed is as follows:

### A. Equations

**Step 1:** Calculate and collect the list of attributes to be ordered in terms of significance:

$$M[] \leftarrow \forall\, P \,\exists\, \left(\sum_{i=1}^{n} p_i\right) \tag{1}$$

Where,

$M$: initial set of attributes,

$P$: The total available list of attributes.

**Step 2 :** Assign the selection vector as S[L], where L is the size of the initial attribute list. Initially, the vector is filled with zeros to denote that no optimal subset is selected.

$$\forall S, S[i] \leftarrow 0 \tag{2}$$

Step 3: In this step of the algorithm, the fitness for all the attributes is calculated so that the ranking can be provided for the features against all other attributes.

$$\begin{aligned}&\text{Info}_{\text{Gain}}(S_i)\\ &= 1 + \text{Info}_{\text{Gain}}(\text{Class}_{i,}S_i),\\ &\text{if } \text{Info}_{\text{Gain}}(\text{Class}_{i,}S_i) \geq \text{Info}_{\text{Gain}}(\text{Class}_{i-1,}S_{i-1})\\ &1 + \text{Info}_{\text{Gain}}(\text{Class}_{i,}S_i), \textit{Else}\end{aligned} \tag{3}$$

Step 4: Once the attributes are been ranked, the final selected subset is produced.

$$S[m] \leftarrow \text{Highest}(\text{Info}_{\text{Gain}}(M)) \tag{4}$$

## IV. PROPOSED AUTOMATED FRAMEWORK

Reuse of components is a process of recovery and processing. Adjust the components in the component database. Resolve a specific problem. To solve these problems, tool users must extract and compare the requirements, and formats implemented in the search process and details of adaptation. Although the traditional component representation language completely removes copies of implementation details, it does not work during the search [36].

The formal interface specifications can simplify and make the assistant search process more precise than representative components and requirements of the problem directly. Also, support for formal specifications and perfect automated mathematical operations can take into account three forms of interface specification aspects of reusable components: (i) potential recovery solutions; (ii) evaluation of the correction; and (iii) architecture to make changes [37][38].

Recovery and modification of individual components is a process to find and modify components. Resolve the problem given a question and a set of components; recovery is one or more processes. The component focuses on more potential solutions. A trivial answer is the most appropriate model for research activity. Component sets solve problems. The description and satisfaction criteria are defined in the research objectives. When the problems and requirements of the particular description component are clear, the satisfaction criteria can be defined by mathematical techniques [39]. The evaluation of the correction is to determine whether a component meets the specifications of a problem and how. Because of problems and components, the evaluation of the correction is the process to determine whether a component can be used to solve the problem for evaluation of the correction. The best simulation is the satisfaction model of formal specifications and the use of formal satisfaction standard support for the formal evaluation of the component correction [40].

Component based development makes a lot of sense for the software engineering industry. Successful CBD is widely accepted as a promising method of software reuse. Recently, the CBD has been presented as a complex and adaptable solution for the corporate IT system for developing software. Another new programming paradigm, feature-oriented programming is designed to be a modular. It provides the mechanism for the implementation and execution of the characteristics that make up the concept of representative fields [41].

Existing studies only analyse a subset of the Object-oriented concepts and evaluate the design quality of reusable components. Almost all studies are taken into account but the important features of Object-oriented paradigm excessively focuses on implementing languages C++ and Java. Observation has been made in this study considering all the basic concepts of the sample and measuring it at design time as common as possible (independent of any implementation language). Measurement trend analysis during the evolution of industrial strength software components was done over a period of time. Software metrics help to measure the properties of a program. The metrics have two types of reuse-oriented paradigms: (i) Component-Based Software Development (CBSD); and (ii) Object-Oriented Software Development (OOSD). Strategies for developing adaptive reusable components using top-down domain analysis lead to good quality of the component [42].

Component-based software metrics are discussed at two levels: system level and component level. Component based research software indicators are not yet mature. The basic concepts of C language and Python are similar, and C++ and Java are similar because these are based on Object-oriented programming. There is lack of automation indicators, therefore, the number of empirical studies in this area is also very small. The Object-oriented paradigm has several concepts such as abstraction, inheritance, information hidden, polymorphic, coupled, and cohesiveness which help to develop an Object-oriented program easily to modify and expand, so it is easy to reuse. Object-Oriented metrics are discussed at different levels, such as systems, software packages, and course levels [43].

The importance of the genetic system cannot be reduced because some of its functions are too valuable to give up and its reproduction is too expensive. Developing agile software is the first choice for all types of small, medium, and large organizations. The problem facing the software industry is that the development of agile software produces specialized products that cannot be recycled. All these developments have been exhausted. Therefore, the biggest challenge is to reuse a flexible development environment [32].

## *A. Estimated Reuse*

Based on the number of criteria corresponding to the total number of current guidelines, assessment recovery is part of the assessment process and an assessment report is presented. Here, we have to automate this process. The result of this process is to ensure that the project being restored meets certain important features.

**Algorithm 1 : Dependency Based Reusable Component Identification (DBRCI)**

```
Step - 1.   Consider list of components as C[i]
Step - 2.   For each C[i] analysis the dependencies with
            other components
                a. If C[i].ExternalSystemCall = C[j]
                    i. Then increase the DependencyRanking{C[i]}
Step - 3.   For each DependencyRanking{C[i]}
                a. Search Max|DependencyRanking{C[i]}|
                b. Find the correlated C[i]
Step - 4.   Report Top|C[i]| as prime components for reuse
```

## *B. Improving Reuse*

Improved reuse is the process that converts and improves the reuse of components when adding attributes to reuse. This process is based on the assessment report drawn up in the previous step. The recyclable enhancer must know which abstract traits are reusable. Again, automatic recovery improvement is essential. Eventually, it produces components that are potentially reusable [33].

**Algorithm - 2: Subset Based Max Dependency Identification (SBMDI)**

```
Step - 1.    Consider the prime components as Top|C[i]|
Step - 2.    For each Top|C[i]|
                 a. Identify dependency subset as CS[j]
Step - 3.    For each CS[j]
                 a. Calculate Max|CS[j]| // Identify the
                    maximum dependency subset
Step - 4.    Report Max|CS[j]| as Improved Reusable Component Set
```

## *C. Find the Right Component*

The research process is more than just finding the perfect match. It is often necessary to locate similar components because even if the target component requires partial upgrades and is not so reusable, it can be close enough to the ideal components, reducing costs and eliminating many errors. Higher the accuracy, larger the component, the less likely it is to be reused across multiple applications. It is hard to find the perfect fit component in many cases [34].

## *D. Substitution*

As new components become more demanding, components can be created, modified, and developed. Suppose we can build a system that allows significant recovery of altered parts of the component that is unrealistic. The percentage change must be defined as the value of the input cost and quality model. Anybody can use some tools for modifying components.

**Algorithm - 3 : Replicable Component Identification and Substitution (RCIS)**

```
Step - 1.    Consider the component Subset as Max|CS[j]|
Step - 2.    For each component C[i] in Max|CS[j]|
                 a. If C[i] == C[i+1] and Max|CS[C[i]]| == Max|CS[C[i+1]|
                     i. Then Replace C[i] with C[j]
Step - 3.    Report the Reduced Max|CS[j]|
```

## *E. Ingredients*

The composition process incorporates the most demanding requirements for the composition. It must be possible to express the combined structure of the independent advanced units with explicit calculations [35].

A logical reuse repository in the component library that stores reusable components has the characteristics of the resources it contains. In order to use the software repository effectively, the user again needs to know its content exactly to determine if the library can be satisfied. The repository is used as a mechanism for storing, searching, and retrieving components [44]. However, finding and reusing the right software components is often very difficult, especially when it comes to many components and documentation on how to use them. Development often extends the method used for software libraries. Reusable composers are defined for developing components. This applies not only to the code but is also manufactured in such a way that the products define the system's lifecycle in the form of specifications, requirements, and design. The components in question are intended to be reused in the visualization system and include code, documentation, design, requirements, architecture etc. Creating repositories of a reusable software implementation of a classification scheme is done to create a library and provide components of the search and recovery interface. The main requirement is the compositional classification mechanism. The system must fulfill three functions: load components, download components, and search for software components [45].

Object-oriented programming languages provide another form of reuse. C and Python contain a good discussion. Object-oriented linguistic attributes contribute to reuse, including information hiding, attribute inheritance, and polymorphism. Information hiding is a reusable mechanism because when a part of these systems change, it cannot see that the information that needs to be changed can be reused for the system. By absorbing variables and methods from the super class, property inheritance allows creating new subclasses in super classes. The inheritance process encourages specific methods for reusing previously defined data attributes and processes [46].

## V. EXPERIMENTAL RESULTS AND DISCUSSIONS

This paper proposed a novel approach for component classification and adaptation. Software developers may not know what artifacts are available to develop adaptive software, how the access will be understood and/or how to combine it, and modify to meet current requirements. These challenges are contained in each phase of the modified position [25]. First of all, we need to find some useful code (via an access mechanism or a delivery mechanism), understand the recovered information, and adapt it to current requirements. A tool was developed that uses the input as source code and provides a series of components that are adjusted according to the requirements [47] by using the Apriori algorithm for developing a tool on a PROMISE Software Engineering Repository dataset available publicly.

First, using the proposed framework, the initial dependency rulesets are extracted. The ruleset extraction results are furnished in Table I. The result is also visualized graphically (Fig. 1).

Next, the feature subset extraction is carried out and the results are given in Table II.

Here, merit is the entropy or the fitness function result for each subset based on the correlation of attributes and the scale denotes the acceptance of the generated subjects.

Although the reuse of software has been implemented in some way for many years, it is still a new discipline. It also covers non-technical issues such as law, economics, measurement, and organization. This research article focuses on the technical aspects of software reuse, in particular on the reuse of software components. However, it also contains information on other forms of reuse and distinguishes them [48].

The results are also visualized graphically (Fig. 2). Our tools provide a classification of connected components and technologies. This classification goes

TABLE I .

RULE SET EXTRACTION RESULTS

| Initial Feature | Dependent Feature | Rule Confidence | Extracted Rule |
|---|---|---|---|
| Independent Team | Repository | 1 | Independent Team (no) ==> Repository (yes) |
| Type of Software Production | Rewards Policy | 1 | Type of Software Production (product-family) ==> Rewards Policy (no) |
| Key Reuse Roles Introduced | Repository | 1 | Key Reuse Roles Introduced (yes) ==> Repository (yes) |
| Type of Software Production, Repository | Rewards Policy | 1 | Type of Software Production (product-family). Repository (yes) ==> Rewards Policy (no) |
| Rewards Policy | Type of Software Production | 0.95 | Rewards Policy (no) ==> Type of Software Production (product-family) |
| Rewards Policy | Repository | 0.95 | Rewards Policy (no) ==> Repository (yes) |
| Type of Software Production | Repository | 0.95 | Type of Software Production (product-family) ==> Repository (yes) |
| Top Management Commitment | Repository | 0.95 | Top Management Commitment (yes) ==> Repository (yes) |
| Repository, Rewards Policy | Type of Software Production | 0.95 | Repository (yes). Rewards Policy (no) ==> Type of Software Production (product-family) |
| Type of Software Production, Rewards Policy | Repository | 0.95 | Type of Software Production (product-family). Repository Rewards Policy (no) ==> Repository (yes) |

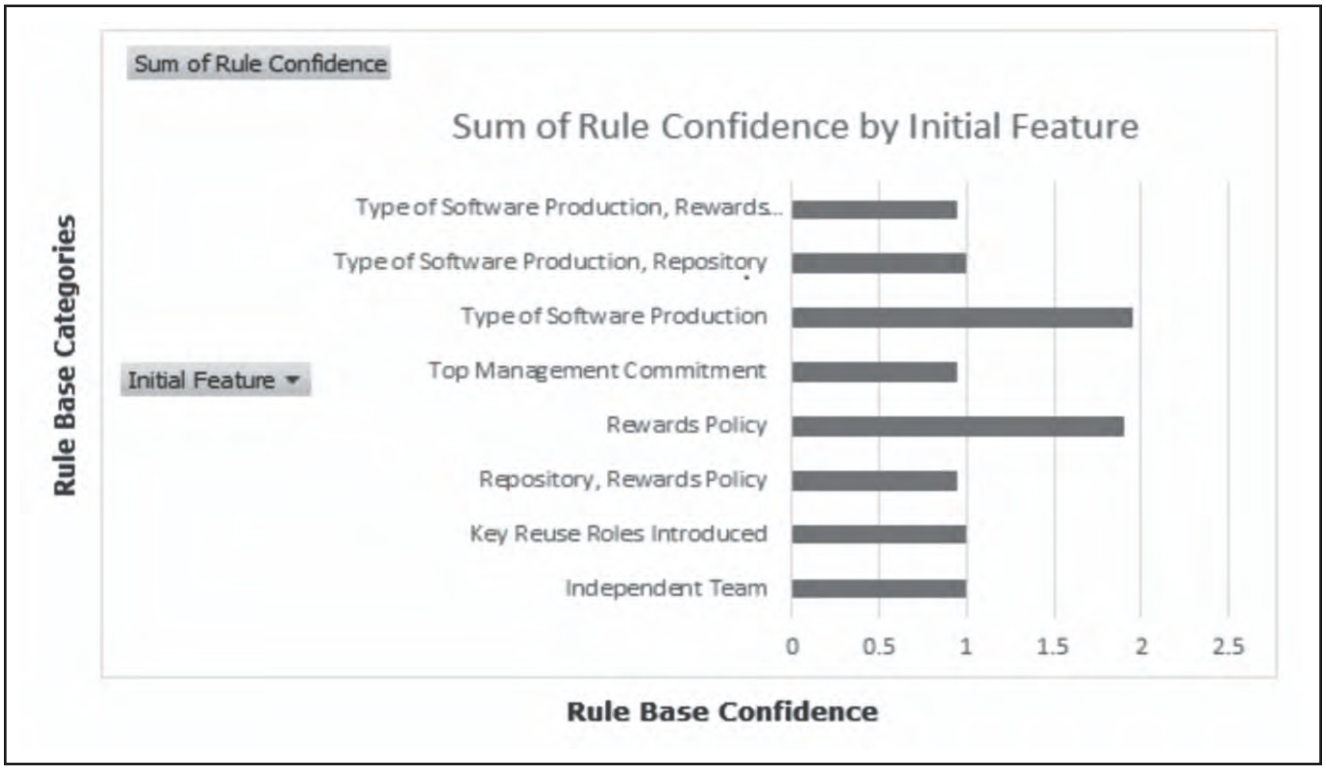

**Fig. 1. Rule-Based Component Reusability Analysis**

**TABLE II.**

**FEATURE SUBSET EXTRACTION RESULTS**

| Subset Name | Subset Description | Subset Count | Merit | Scale |
|---|---|---|---|---|
| Sub1 | 1 5 7 9 11 13 18 22 | 8 | 0.48393 | 0.77985 |
| Sub2 | 1 5 7 9 11 13 15 17 18 22 26 | 11 | 0.45797 | 0.47728 |
| Sub3 | 1 5 9 11 13 20 21 22 | 8 | 0.43825 | 0.24738 |
| Sub4 | 1 5 7 9 11 13 18 21 22 | 9 | 0.4565 | 0.46008 |
| Sub5 | 5 7 8 9 13 14 18 22 26 | 9 | 0.41703 | 0 |
| Sub6 | 1 5 7 9 11 13 18 22 | 8 | 0.48393 | 0.77985 |
| Sub7 | 1 3 5 7 9 11 13 18 22 25 26 | 11 | 0.45921 | 0.49168 |
| Sub8 | 1 5 7 9 11 13 18 22 | 8 | 0.48393 | 0.77985 |
| Sub9 | 1 3 5 7 9 10 11 13 18 22 | 10 | 0.45849 | 0.48329 |
| Sub10 | 1 5 9 11 13 16 18 20 22 | 9 | 0.44171 | 0.2877 |
| Sub11 | 1 3 5 7 9 10 11 13 17 19 22 25 | 12 | 0.43172 | 0.17122 |
| Sub12 | 1 5 7 9 11 13 18 22 | 8 | 0.48393 | 0.77985 |
| Sub13 | 1 5 7 8 9 11 13 18 22 | 9 | 0.4782 | 0.71306 |
| Sub14 | 1 5 7 8 9 11 12 13 18 20 26 | 11 | 0.44355 | 0.30911 |
| Sub15 | 1 4 5 7 11 15 16 22 | 8 | 0.41875 | 0.02004 |
| Sub16 | 1 3 5 8 9 11 12 13 17 18 22 24 26 | 13 | 0.42833 | 0.13172 |
| Sub17 | 1 5 7 11 15 22 | 6 | 0.46345 | 0.54117 |
| Sub18 | 1 4 5 7 9 11 13 18 22 26 28 | 11 | 0.4265 | 0.11039 |
| Sub19 | 1 5 7 9 11 13 18 22 | 8 | 0.48393 | 0.77985 |

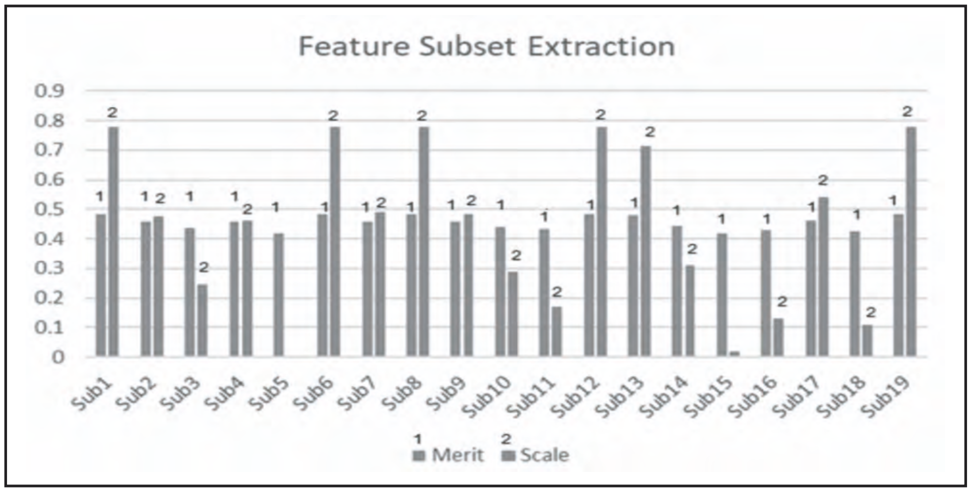

**Fig. 2. Feature Subset Analysis**

**TABLE III.**

**FEATURE SUBSET SUBSTITUION RESULT**

| Subset Name | Subset Description | Subset Count | Merit |
|---|---|---|---|
| Sub_Proposed | 1,5,7,9,22 | 5 | 0.492 |

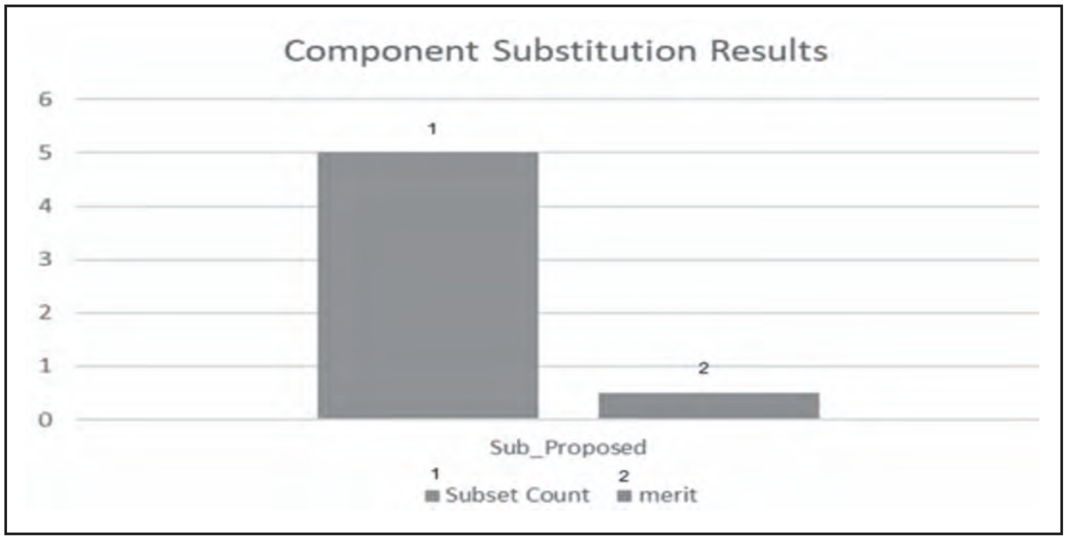

**Fig. 3. Component Replacement Results**

beyond the source code components and covers all aspects of distributed information underlining the components which import and express the source code. Software components are more than just functions and classes having a group together to get the classification [49].

Like software reuse, software components will go beyond the source code. The coverage of the components is wider than the structure and model. Our tools show the success of the reuse of components and evaluate using the proposed classification scheme [50].

Finally, the component replacement results are analysed in Table 3 and Fig. 3 on the standard Kaggle dataset [51].

Finally, with the discussion on the highly satisfiable obtained results, this work presents the conclusion in the next section.

## VI. CONCLUSION

Association rule learning is a standard AI based technique for finding fascinating relations between factors with regards to extensive databases. It is proposed to distinguish solid principles found in databases utilizing a few proportions of intriguing quality. This standards based methodology additionally creates new principles as it breaks down more information. A definitive objective expecting a sufficiently substantial dataset is to enable a machine to impersonate the human Cerebrum's element extraction and theoretical association capacities from new uncategorized information. This paper will enhance the ability to adjust components by combining expressiveness, effectiveness, and reliability.

## VII. CURRENT AND FUTURE DEVELOPMENT

The current research outcomes are challenged by the software development industry with the arguments of reliability as different types of applications from different domain demand unique specifications and standards to be met. Also, the components which are to be reused for other specifications must match the compliances of that specification. Considerably, identification of the reusable components is always a challenge and it is observed that the identification of the flexible components for which matching the domain specification is easy is also a highly complex process. Finally, finding the ideal coupling point of the identified modules is also a significant challenge. Future enhancement for this work is to develop a framework for the identification of the flexible components for reuse in the software development lifecycle and identifies the unique coupling points of these components.

## AUTHOR'S CONTRIBUTION

Dr. Sampath Korra conceived the original idea of the paper. He collected data, analysed, and interpreted the results. He drafted the manuscript and revised it critically for important intellectual content.

## CONFLICT OF INTEREST

He author certifies that he has no affiliation or involvement with any organization or entity in which he has any financial or non-financial interest in the subject matter or material discussed in this manuscript.

## FUNDING ACKNOWLEDGEMENT

The authors received no financial support for the research, authorship and/or publication of this article.

## REFERENCES

[1] H. Algestam, M. Offesson, and L. Lundberg, "Using components to increase maintainability in a large telecommunication system," in *9th Asia-Pacific Softw. Eng. Conf., 2002*, pp. 65–73, doi: 10.1109/APSEC.2002.1182976

[2] M. T. Baldassarre, A. Bianchi, D. Caivano, C. A. Visaggio, and M. Stefanizzi, "Towards a maintenance process that reduces software quality degradation thanks to full reuse," in *Proc. 8th IEEE Workshop Empirical Stud. Softw. Maintenance (WESS'02)*, 2002, p. 5

[3] V. R. Basili, "Viewing maintenance as reuse-oriented software development," *IEEE Softw*., vol. 7, no. 1, pp. 19–25, Jan. 1990, doi: 10.1109/52.43045

[4] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: A roadmap," in *Proc. Conf. Future Softw.Eng*., Assoc. Comput. Machinery, New York, NY, USA, 2000, pp. 73–87, doi: 10.1145/336512.336534

[5] D. Damian, J. Chisan, L. Vaidyanathsamy and Y. Pal, "An industrial case study of the impact of requirements engineering on downstream development," in *2003 Int. Symp. Empirical Softw. Eng. Proc*., pp. 40–49, doi: 10.1109/ISESE.2003.1237963

[6] M. Jørgensen, "The quality of questionnaire based software maintenance studies," in *ACM SIGSOFT Softw. Eng.* Notes, vol. 20, no. 1, 1995, pp. 71–73, doi: 10.1145/225907.225916

[7] M. M. Lehman, "Laws of software evolution revisited," In Montangero, C. (Eds.) Softw. Process Technol. EWSPT 1996. *Lecture Notes Comput. Sci.,* vol. 1149. Springer, Berlin, Heidelberg, doi: 10.1007/BFb0017737

[8] B. P. Lientz, E. B. Swanson, G. E. Tompkins,"Characteristics of application software maintenance," *Commun. ACM*, vol. 21, no. 6, pp. 466–471, 1978, doi: 10.1145/359511.359522

[9] Y. K. Malaiya and J. Denton, "Requirements volatility and defect density," in *Proc. 10th Int. Symp. Softw. Rel. Eng.* (Cat. No.PR00443), 1999, pp. 285–294, doi: 10.1109/ISSRE.1999.809334

[10] G. Basalla, *The evolution of technology.* New York, NY, USA: Cambridge Univ. Press, 1989, doi: 10.1017/CBO9781107049864

[11] J. S. Brown and P. Duguid, *The social life of inf.* Harvard Bus. School Press, Boston, MA.

[12] R. Dawkins, *The blind watchmaker*, W.W. Norton and Company, New York – London, 1987.

[13] G. Fischer, "Domain-oriented design environments," *Automated Softw. Eng.*, vol. 1, no. 2, pp. 177–203, 1994, doi: 10.1007/BF00872289

[14] J. Greenbaum and M. Kyng, "Knowledge-based design environments," Ph.D. Dissertation, *Dept. of Comput. Sci.,* Univ. of Colorado at Boulder, Boulder, Co. (Eds.), 2011, doi: 10.3727/108812897792458281

[15] J. Greenbaum and M. Kyng, *Des. work : Cooperative Des. Comput. Syst.,* Hillsdale, NJ, USA: Lawrence Erlbaum Associates Inc..

[16] A. Henderson and M. Kyng, "There's no place like home: continuing design in use," in J. Greenbaum & M. Kyng (Eds.), *Des. at Work: Cooperative Des. Comput. Syst.,* Hillsdale, New Jersey, NJ, USA: Lawrence Erlbaum Associates, 1991, pp. 219–240.

[17] S. R. Henninger, "Locating relevant examples for example-based software design," Ph. D. Dissertation, Dept. of Comput. Sci., Univ. of Colorado at Boulder, Boulder, CO, USA, 1993.

[18] W. Kintsch, *Comprehension: A paradigm for cognition,* Cambridge Univ. Press, Cambridge, England.

[19] B. A. Nardi, *A small matter of program, Cambridge,* MA, USA: MIT Press, 1993.

[20] B. H. Liskov and S.N. Zilles, "Specification techniques for data abstractions," *IEEE Trans. Softw.* Eng., vol. SE-1, no. 1, March 1975, pp. 7–19, doi: 10.1109/TSE.1975.6312816

[21] K. J. Sullivan and J. C. Knight, "Experience assessing an architectural approach to large-scale, systematic reuse," in *Proc. 18th Int. Conf. Softw. Eng.*, Berlin, May 1996, pp. 220–229.

[22] D. C. Schmidt, "Why software reuse has failed and how to make it work for you." [Online]. Available: http://www.dre.vanderbilt.edu/~schmidt/reuse-lessons.html Accessed: Aug. 18, 2002.

[23] D. E. Harms and B. W. Weide, "The influence of software reuse on Programming Language Design," *The Ohio State Univ.,* 1990.

[24] S. Korra, D. Vasumathi, and A. Vinayababu, "An approach for cognitive software reuse framework," in *2018 2nd Int. Conf. Intell. Comput, Control Syst.*, pp. 1–6. IEEE, June 2018, doi: 10.1109/ICCONS.2018.8662897

[25] D'Alessandro, M. Iachini, and P. L. Martelli, "A the generic reusable component: An approach to reuse hierarchical OO designs," *Softw. Rseusability,* 1993.

[26] E. M. Dusink, "Cognitive psychology, software psychology, reuse and software engineering, technical report," TU Delft, Delft, The Netherlands, 1991.

[27] E.M. Dusink, "Testing a Software Engineering method statistically. *Technical report," TWI,* TU Delft, Delft, the Netherlands, 1991.

[28] H. Gomaa, L. Kerschberg, V. Sugumaran, C. Bosch, I. Tavakoli, and L. O' Hara, *Automated Softw. Eng.*, vol. 3, pp. 285–307, 1996, doi: 10.1007/BF00132570

[29] A. Kumar, "Software reuse library based proposed classification for efficient retrieval of components," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, pp. 884–890, 2013.

[30] J.-M. Morel, "The REBOOT approach to software reuse," in Software Reuse: *The Future, The BCS Reuse SIG1995 Workshop,* 1995.

[31] J. Bosch, *Design and use of software architectures: Adopting and evolving a product-line approach,* New York, NY: Pearson Education, 2000.

[32] M. D. Jonge, *To reuse or to be reused techniques for component composition and construction,* pp. 57–58, 2003.

[33] R. A. Flores-Mendez, *Towards a standardization of multi-agent system framework*, 1999.

[34] G. H. Campbell, Jr. "Adaptable Components," in *Proc. 21st Intl. Conf. Soft. Eng.,* Assoc. Comput. Machinery, 1999, pp. 685–686.

[35] T. P. Kelly and B. R. Whittle, "Applying lessons learnt from software reuse to other domains," in *7th Annu. Workshop Softw. Reuse,* Aug. 28–30, 1995, St. Charles, Illinois, USA.

[36] M. Cioca and L.-I.Cioca, "Multi-criterion analysis of reference architectures and modeling languages used in production systems modeling," in *INDIN '05. 2005 3rd IEEE Int. Conf. Ind. Inform,, 2005*, pp. 230–233, doi: 10.1109/INDIN.2005.1560381

[37] M. Ionita, D. Hammer, and J. Obbink, "Scenario-based software architecture evaluation methods: An overview," 2002.

[38] P. J. Modi, S. Mancoridis, W. M. Mongan, W. Regli, and I. Mayk, "Towards a reference model for agent based systems," in *Proc. 5th Int. Joint Conf. Auton. Multiagent Syst.*, May 2006, pp. 1475–1482, doi: 10.1145/1160633.1160922

[39] R. W. Collier, "Agent factory: A framework for the engineering of agent-oriented applications," Ph.D. Thesis, Univ. College, Dublin, Ireland, 2001.

[40] R. N. Taylor, W. Tracz, and L. Coglianese, "Software development using domain-specific software architectures: : CDRl A011—a curriculum module in the SEI style," ACM SIGSOFT Softw. *Eng. Notes,* vol. 20, no. 5, Dec. 1995, pp. 27–38, doi: 10.1145/217030.217034

[41] D. E. Harms, "The influence of software reuse on Programming Language Design," *The Ohio State Univ.,* 1990.

[42] S. Korra, D. Vasumathi, and A. Vinaybabu, "A novel approach for building adaptive components using top-down analysis," *Int. J. Eng. Technol., vol. 7*, no. 4.19, pp. 1036–1040, 2018, doi: 10.14419/ijet.v7i4.19.28282

[43] G. Kakarontzas, E. Constantinou, A. Ampatzoglou, and I. Stamelos, "Layer assessment of object-oriented software," *J. Syst. Softw.,* vol. 86, no. 2, pp. 349–366, Feb., 2013, doi: 10.1016/j.jss.2012.08.041

[44] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proc. 29th Int. Conf. Softw. Eng*., pp. 344–353, May 20–26, 2007, doi: 10.1109/ICSE.2007.45

[45] C. W. Krueger, "Software reuse," *ACM Comput. Surveys*, vol. 24, no. 2, pp.131–183, June 1992, doi: 10.1145/130844.130856

[46] O. A. L. Lemos, S. Bajracharya, J. Ossher, P. C. Masiero, and C. Lopes, "A test-driven approach to code search and its application to the reuse of auxiliary functionality," *Inf. Softw, Technol.*, vol. 53, no. 4, pp. 294–306, April 2011, doi: 10.1016/j.infsof.2010.11.009

[47] J. Maras, M. Štula, and I. Crnković, "Towards specifying pragmatic software reuse," in *Proc. 2015 Eur. Conf. Softw. Architecture Workshops*, Sep. 07–11, Dubrovnik, Cavtat, Croatia, pp. 1–4, 2015, Art. No. 54, doi: 10.1145/2797433.2797489

[48] N. Niu, J. Savolainen, Z. Niu, M. Jin and J. -R. C. Cheng, "A systems approach to product line requirements reuse.," *IEEE Syst. J.*, vol. 8, no. 3, pp. 827–836, 2014, doi: 10.1109/JSYST.2013.2260092

[49] N. Niu, X. Jin, Z. Niu, J. C. Cheng, L. Li, and M. Y. Kataev, "A Clustering-Based Approach to Enriching Code Foraging Environment," *IEEE Trans. Cybernetics*, vol. 46, no. 9, pp. 1962–1973, Sept. 2016, doi: 10.1109/TCYB.2015.2419811

[50] N. Niu, A. Mahmoud, and G. Bradshaw, "Information foraging as a foundation for code navigation: NIER track," in *2011 33rd Int. Conf. Softw. Eng.,* 2011, pp. 816–819, doi: 10.1145/1985793.1985911

[51] "RTTool Kaggle Dataset," [Online]. Available: https://github.com/aserg-ufmg/RTTool/tree/master/src/org/scitools/metrics

## About the Author

**Dr. Sampath Korra** is Associate Professor with Department of Computer Science and Engineering at Sri Indu College of Engineering and Technology(A), Sheriguda, Ibrahimpatnam, Hyderabad, India. He completed Ph.D. in Computer Science and Engineering from Jawaharlal Nehru Technological University, Kakinada (JNTUK), Andhra Pradesh, India in the year 2020. He completed M.Tech. in Software Engineering from Kakatiya University, Warangal in 2006 and B.Tech. in Computer Science and Information Technology from JNT University, Hyderabad in 2004. He qualified GATE-2003 and GATE 2004. He certified as Java Programmer (SCJP) and Python Programmer from Microsoft. He has 15 years of teaching experience in various Computer Science and Engineering subjects at Under-Graduate and Post-Graduate level. He has 12 years of research experience including teaching. He has published many research articles in reputed journals and conferences. He is the author of 3 textbooks and has 4 patents to his credit. He has participated in many conferences, seminars, workshops/FDPs, and webinars related to research. His current research interests are Software Engineering, Machine Learning, Artificial Intelligence, Data mining, Data Science, and Bio-Informatics.