Journal of Theoretical and Applied Information Technology
15th May 2022. Vol.100. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

NOVEL TECHNIQUES FOR COMPONENTS CLASSIFICATION AND ADAPTATION

[1]Dr.SAMPATH KORRA, [2]Dr.V.BIKSHAM

[1]Associate Professor, Department of CSE, Sri Indu College of Engineering &Technology (A), Hyderabad

[2]Associate Professor, Department of CSE, Sreyas Institute of Engineering & Technology, Hyderabad

E-mail: [1]sampath_korra@yahoo.co.in,[2]vbm2k2@gmail.com

ABSTRACT

More than thirty years have passed since the software introduced the idea of reuse. There are many successful cases that have been reported, but people believe that the programs are still in the re-development phase and are not reaching its full potential. The software requires us to anticipate the future needs of software systems so that new units can be built, some functions, features are fragmented and reused easily by the engineers. These tools are designed to achieve this aspect of software reuse through component adaptation and enhance existing applications. Use software information to reuse a large number of platforms and tools in the presence. This work does not think of the classification based on the naming services and components, but this work tries to use the most relevant features of components such as software developers and it explains the functional requirements to make a definite decision. In the way, the proposed algorithm is very generic and widely available to all technologies. It is only for a short period of time for the selection of software components and all types of services.

Keywords: *Software Reuse, Architecture, Domain Engineering, Indicators, Components.*

1. INTRODUCTION

This article debates the summary of some important aspects of component classification technique. Because the main purpose of reusable components is to increase efficiency and reduce costs, which ultimately affects the economy of the software industries and summarized in three ways to reuse the software[1].

The reuse of software is considered to be the ability to increase productivity in software development and quality of software. The key benefits of newly introduced software are the support for the design method and the important event is not the construction of the new system from scratch, but the modification of the integration and the description of the existing ones. CBSE was used to support evolution [2] of the components of the different technologies. However, it is sufficient to keep the repository empty outstanding, types of software reuse: One is the software component management that consists of the specification, classification, and extracts of the existing components, and the other is the integration part which includes integration of the reusable component into the application. Several approaches are developed in recent years to address the issues of reuse. However, the lack of approaches to smooth integration constitutes a significant obstacle to effective recovery. Reusability tools that can be used to reactivate low functionality changes or modify the source code [3].

Reusability is a basic concept of software engineering. The new software engineering research and practices are dealing with reducing the cost as well as reduces time with better quality and apart according to the development of reusability is a matter about creating a library element, thus allowing the development of new programs, applications of available components. Reusability software is the use of engineering knowledge or software component objects that are available. Creating a new system reusability is the main paradigm in increasing the quality of software development. This is an important area of technical research software that tends to improve software significantly for production and quality. The main advantage CBSD is a cheap and quality solution. Higher productivity, Flexibility, and quality applications changing capacity, efficient storage and resizing are some of the additional benefits CBSD. If there are many components available, it is necessary to develop certain software metrics for different feature components. It is

necessary to measure components in order to realize the reuse of the benefit. It can also measure reusability of a component by indirectly, the complexity, arrangement, and monitoring can be also measured to reusable of a component indirectly [4].

A software component is a standalone entity that provides services and interacts with the environment through all defined interfaces that require the functionality of other components. One of the main drivers behind the Plug-in technology is reused. A series of reusable software components can create the application by grouping the existing component together. Other drivers of component technology are the independent development of application components, increased flexibility, adaptation, and maintenance of software systems. To successfully connect components together, each component's interface must match the requirements of other components [5]. In this way, the "nodes" are defined among the components are good. Therefore, the development of component-based applications depends on the constrained components and compliance with a compatible interface with standard interactive protocols [6]. Due to fierce competition in the market for software components, many types of research have focused on the software components over the past few years, how to find them, how to choose them, and how to make them work as integrated software [7].

Available methodology and techniques are used for the classification of the components [8]. However, what happens when changing the existing dependencies in the evolution process is largely ignored. By re-using contracts, we can make this change and assess its impact by registering these dependencies and using the operators again. In addition, the scope of reuse agreements is wider than managing changes in a continuously changing system: it exposes the architecture of the system and can be used as a structural document, often in the system of software engineers to adapt to their needs [9].

To improve the software system's adaptability mainly refers to the software system's ability to adapt to the needs changes in the external environment.

The software system update is mostly possible from the following ways: [10] Improving software compatibility: The design software is based on the business requirements that cannot be responded at this time, but should be commonality, which means that different changes may be the account that the system design process should be carried out. If the condition changes outside, the system should be able to improve and maintain stable operation. The software updates itself-description: The system has a certain capacity, a self-explanatory, and an external condition change can be explained by using as many parameters as possible. If the update conditions are changed, the related parameters are needed, but the large area of the software is not required [11].

Software update tool used for a large component which requires no changes or is often called the modules; they will be seen as the tool that handles what is in the design process, which increases the flexibility and performance of the system.

Software update module is going to create a powerful and independent module requires different functions, that should be done through the modules to do all the information to simplify the process.

Software updates try to make free software design platform unlimited software and hardware platforms [12].

It is not easy to include recovery software during the programming process. When the investment is restored, but even after the expected learning curve is not bothered, we can re-enter the recovery option later. In other words, although we are ready to use managed software, it is important to know as much as possible how to develop recovery procedures in the right direction. Ensure that there are sufficient resources and means to reuse the software, including technical knowledge (domain analysis, reusable resources, storage resources, and identification creation), resources and incentives to create and use reusable objects. Resources with any long-term investment, we should be able to ensure that investors things are pending in accordance with the plan, and this requires an effective feedback loop. In addition, to reuse manage, we should also invest in refining judgment, measure and review the project development process. Obligations must be determined specifically to acquire and maintain the reusable components of the project[13].

Profitability is the main objective of all software organizations. In recent years, software development paradigm has become tremendously important due to the rapid changes in company needs. Customers are now demanding that their desired products are delivered in a minimal period of time. To meet these needs, various rapid development techniques are called agile development and reuse is introduced in the software industry. With the help of reusable components, the development and implementation of software products can be much

simpler and cheaper. Such approaches may help to save costs and working hours so that the developing country can use its terrible resources in other projects[14] [15].

Appropriate archiving installation of reusable components is necessary if an organization can store important data and, if necessary, bring many standard repositories which are used by different organizations. For this purpose, as important data is stored and downloaded. However, in fact, each repository contains a vast amount of data and generally, the adequate commercial standards for the storage mechanism do not apply. While some repository maintenance software is available on the market, it provides only a mechanism for archiving artifacts. The retention of a large amount of data requires the opening of an extensive research and the procedure for the submission of recommendations to precise reusable components; therefore, researchers must investigate a large amount of data in the repository in order to find the desired component. In this paper, we discussed the new concept of central values-based software repository(CBVSR). The proposed approach helps users find the most appropriate artifact because it gives the best match possible to the user's query. To capture, delete, modify and destroy the CBVSR data, appropriate standards have been complied with. Software reuse was taken of the quality, safety, and integrity of the data, and submitted for developers' technical assistance indices, marking and classification [16].

The software repository stores various software components or objects for future reuse. It is placed through a local or global network based on organizational requirements. The user can access the software repository directly or indirectly without physically moving to its location [17]. Software archives are designed to be malware-free and maintain reusable and valuable artifacts.

Software companies develop products to gain a higher market share by attracting customers or stakeholders. The entire software development process has been modified with the introduction of agile techniques. The developers now aim to provide the required products for a minimum period of time. Due to this, reuse of different software components is ideal in these situations. However, the recovery has some limitations. For example, incorrect integration of the reusable code may damage the whole system architecture. It may also increase the complexity of detecting defects and its elimination during the test phase. Injecting an inappropriate component into the software design can also compress the entire system, and so on. The problems mentioned in the earlier reuse of software may cost organizations a lot of time and money. To resolve these problems, the software repository is required based on the value that the various software components can be stored in the repository. Categorize them into different categories; Indicating the sets and then at different intervals so that the recovery process designed for the reused element is quick and easy. The Securities-Based repository may contribute to a key role of software engineering to make the reuse more accurate [18].

Reuse is the possibility of reusing source code to add new features with little or no change. Reusable modules and classes are reduced during deployment, increasing the likelihood that past tests and applications will eliminate errors and, if necessary, identifying changes in the code to change of implementations. All routines or functions are the simplest forms of reuse. Code reusable components are periodically arranged using modules or namespaces in the layer [19].

Software Engineers believes that software objects and components provide a more advanced form of reuse, although it is difficult to measure and objectively define levels or reuse results.

Reuse means direct management of problems with drafting, packaging, distribution, installation, configuration, implementation, maintenance, and updating. If these problems are not taken into account, the software may be useful for the project, but it will no longer be used in practice.

The reuse of software is a method that has been practiced for a long time. Programmers are copied and pasted code long from the early days of programming. Although it can accelerate the development process [20].

The word reuse is very limited and does not apply to larger projects. The full benefits of software reuse can only be achieved through system reuse, and system reuse is an integral part of the software development lifecycle. Some of the key aspects of software recovery research and presents a rough for a reusable software repository. The next step is to further going into depth of the concept and implement the prototype to ensure its effectiveness. We examine the history of software reliability engineering, current trends and problems, and specific difficulties. Future trends and promising research issues for reliability engineer software have also been taken into account. We created current and future potential trends for software reliability engineering based on industry and customer needs [21].

While the software process proposals are continually being displayed, it is difficult to adapt

them to a particular business. Therefore, certain types of customization are always necessary. Although process customization is a mandatory activity in most software process proposals, it is usually carried out after an ad hoc approach and a lot of research has been done so far reuse can be considered to be minimal. We provide a systematic overview of the adaptation of the software processes, analyzing existing approaches, discussing the main problems of the operation, and establishing a new and comprehensive new research framework [22].

Reusing software is the process of creating a software system for existing software rather than creating a software system from scratch. What was originally written in another project and the implementation of universally recognized recovery? Code recovery means that a partial or complete computer program written at the same time may use or may not be used in another program[23]. Code reprogramming is a common technique that saves time and effort by reducing overwork. Resources or software elements include all software products, including requirements and recommendations, descriptions and designs, advanced projects, data formats, algorithms, user manuals, and test suites. Things created by the development software can be reused. The software has been developed and reused by the same person, and the use and recovery of operating systems, database management systems and other system tools are different in the same project and implementation, product maintenance and new product versions. Software Engineering is focused on more initial development, but now it is recognized to get better, faster and cheaper software, the design process based on systematic software reuse is required [24][25].

Software reuse provides the basis for components and interfaces about interoperability standards and can be defined metadata for component customization and composition. Component customization the ability to adapt component prior to a consumer's setup or use, components are usually treated in black box fashion, their application can be customized using only the clearly defined customization interfaces, exposing the components as little as possible. Customization and deployment tools, customizing functionality to modify simple properties or even complex behavior by providing instances of other components as parameters to customization interface enables [26].

Software Development using components (development components, CBD) [27]. By using the software system, plans, structures, and components of the new software that will lead to support the reuse of the software component. Reducing delivery time in essence this software development is based on two components the way reuse, and structure of the system. Therefore, some documents are called part of the system reuse core technology that supports the software for reuse as a software component of technology. This CBD is an important area in the field of reusable components has evolved very quickly in recent years. Given components, models, language descriptions, classification and extracts, complex assemblies, made as primary studies. The deep insight of introduced software, component concepts have been deleted for a more restrictive time to source codes, intermediate codes, but requirements extensions, software architecture, documents, testing, and other useful data development information. These messages contain a variety of new active components. In olden days the era of software architecture, but a special time is a long time, the lowest of the deployment has been ignored. Due to the complexity of the software system continues to grow, component customization, assembly, and collaboration technologies cannot create an easy to understand deployment mechanism, easy to assemble and the high automation. For then need to design according to the requirements of the software, select the system and adapt it to a high level of components that use the system of abstract architecture software[28].

Software components are a software unit that communicates with other independently developed components. Component-based software engineering focuses primarily on package software, independent units, to allow maximum re-usability. A component-based software development (CBSD) is the customization of object-oriented software development (OOSD) and share the goal of software recoverability [6]. Object-oriented software development is a method of implementation and software development is an interface methodology. In component-based software development, the importance of the standardization interface is the best components, without limitation, on how to implement achieved. Therefore, the component-based software development is closely linked to the module design in a separate interface and implementation [29].

For object-oriented software development, code reuse is the competent code for implementing the succession. Although object-oriented a language also allows for partitioning implementation and interface, class libraries and procedural libraries designed to equate with larger applications are a successful example of software reuse.

2. RELATED WORK

In this research article, we presented some basic concepts and principles of component models and component model applications. The component models define interfaces, naming, interoperability, customization, composition, evolution, packaging, and distribution standards. In addition, the specifications of the run-time environments and services are required to standardize the component models. Typically, there are component model implementations on top of an operating system, but some operating systems, such as Microsoft Windows, have already begun to include component model implementations. Finally, operating systems can serve as component model applications directly to the CBSE [30].

The 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 the set of types or classes from which objects are created.

In particular, Java is compiled with a low-level, machine-independent code called bytecode. This bytecode is then interpreted as a Java virtual machine running on a particular machine. This converts Java code to platform independence, which means that the same bytecode can be adapted to any machine that has a different operating system. Migrating Java programs to another machine does not even require recompiling[31].

We must define the component model independently of the distributable so that components can be packaged. If a component is installed and configured in the component infrastructure, it is deployed. The component manufacturer expects the component to be packaged with anything that does not exist in the infrastructure. This can include program code, configuration data, other components, and additional resources. Component-based systems require support for the development of the system. Components that act as a server for other components may need to be replaced by new versions that provide new or improved functionality. A new version can only be a different application but can provide modified or new interfaces. Existing clients of such components are not ideally affected or should be affected as little as possible. Also, older and new versions of a component may need to reside on the same system. The rules and standards for component development and versioning are therefore an extremely important part of a component model [32].

To improve the quality and productivity of the software, we need to modify the existing software. The software's quality and capacity improvement method, which changes the existing software changes, is called software recovery. It is more expedient to make changes to existing software instead of creating new software from scratch. Software reuse is the process of creating software systems for existing software instead of creating them from scratch. The following concept will be used for recovery. Reuse determines the extent of software reuse. It depends on the new features of the software, which become software and features of existing software [33].

3. PROPOSED WORK

The reuse of software components is becoming increasingly important in all aspects of software engineering. A software component can be any part of the software required to use the software, which can be a module or function. Components can be considered part of an identifiable and reusable software system. Components can be reusable functions such as statistical libraries, digital libraries or packages, modules, subsystems, and classes. The success of recovery depends on the quality of the components. If the project provides the behavior required in the recommended situation, the project can be reused. We want to achieve a high level of reuse, should consider the different situations in which the components are used. This software component can be used in many different applications in a variety of commercial and technical environments, with a variety of programmers using a variety of technologies and tools for various users of different organizations.

Many different products use many ideas and algorithms for any document generated by the software lifecycle. The source code is more common; therefore, many people misunderstand the reuse of software as the sole reuse of source code. Recently, the source code and reuse of projects have become popular in class libraries (object-oriented), application frameworks, and design drawings. Software components provide tools for regular and system recovery. Today, the term component is often used as a synonym for an object, but it also means a module or function. Recently, the concept of developing some software or components has become popular [34].

Reusing software components is critical to increasing productivity. However, in order to take full advantage of this potential, we need to focus on reuse development, a process that produces

potentially reusable components. Systematic reuse of software and reuse of components can affect the entire software development process. Software process models have been developed to provide instructions for creating high-quality software teams to achieve predictable costs. The original model is based on the belief that the system is built on stable scratch-based requirements. The software process model has a customized experience and many changes and improvements are recommended. As software is reused, new software engineering models are emerging. The new model is based on system reuse of defined components developed in various projects [35].

The focus should be on the development of reuse, not just the recovery process, which is the normal process of system development. The development of reuse software requires planning, developing and transferring documentation and recovery. The priority of documents in software projects is traditionally low. However, the right documentation is critical to system reuse components. We continue to skip the documentation, will not be able to use components to improve performance. Detailed information about the components is necessary [36]. The development of potentially reusable components depends to a large extent on the definition of functions such as functionality and language domains. These features can be clearly reflected in the reuse guide. Therefore, we must develop objective guidelines and reuse.

Although the software engineer life cycles may vary very much, they are all about the same breakdown analysis, requirements, design, implementation, testing and debugging. These same phases, though not necessarily in that order or with the same emphasis, are the stages of the framework. In this section, the use of concrete has been based on the findings described in the previous two sections.

It is possible to change the part of the product before making the change. We must apply tests before debugging starts. Before it can be tested, planned and/or implemented do so, etc. but it is necessary to have the requirements ready before reviewing the design phase. As people may be seen as an opportunistic process, and as always use past experience. It must be known whether they are applicable before the requirements are terminated. To make a selection of alternatives better reuse, ease of implementation, and risks one must look forward[37].

The issue can only be noted in the direction of a possible solution, one of them will be reconsidered. The changing insights and bugs one have to go back to previous stages and one has to look back to be able to learn. Because of all these reasons, the Yoyo approach is recommended when ordering the stages were going down and up will be controlled “is required" strategy.

The only component that is approaching through the interface is one of the solutions to lift the reusability of the components. The interface acts as to show the element into the unit of the external software by abstracting the function of the element, and it also acts as the media to receive external contacts to use the service of the component.

The interface simply displays the type of service that supplies the component, but it does not display the internal portion of the component, such as how the component supplies the service. This is a substance of abstraction as an agent of encapsulation, and a hidden data or encapsulation of the composition is a combination of data and operations as a component and also takes advantage of hidden data [38].

When the external interface of a component is explicitly defined to reduce the subordination share that is made better component and the encapsulation component, it is created in order to use only through the internal interface. Outside of the details should be hidden inside of the encapsulation components, and hidden data can protect the internal awareness of the components from the near, and the errors can be local in the internal language of encapsulation components and hidden data also reduce the number of interfaces. Each component and those components do not affect the changes in the execution of other components because uses components that are not related to other component operations. So it is good and easy to get not only maintenance but also extend to other new programs. Reuse of components is a method that will contain a system builder consisting of components such as blocks made. Reuse of black boxes without any changes in the detailed events that occur in the internal components generally. In the case of a black box for recovery, the most important idea is to hide the data. When using a black box, there is a hidden reason for the main concept is that it can extend the reuse of the component because it is not necessary to know the details of such components, because this hide is good enough to abstract [39].

In each stage, the understanding of the problem, focusing on reuse and learning is emphasized. Understanding is emphasized because the correct problem needs to be solved.

The components must be in the environment so that they can be reused. The components base is connected means to select potentially useful components.

4. EXPERIMENTAL RESULTS AND DISCUSSION

We developed a tool to use the association rules for component classification. Python, C, C++ and Java code in banking applications have evolved separately and are constantly evolving, as shown in the illustration below; each of these tools depends on or affects other tools [40]. It creates a multi-dimensional development and configuration management that can be difficult to make the cluster. In one case, we have realized that could not involve two different subgroups in the same cluster because they are based on the logic used in the different versions used in the programming languages [41]. All of these applications are developed for banking applications.

The algorithm is divided into steps:

1. Keywords Database used to find all frequently items in a text file.
2. frequent keywords are identified and subsets are formed

```
Apriori Algorithm Pseudocode procedure Apriori
(T, minSupport)

 {
L1= {frequent items};
for (k= 2; Lk-1 !=Ø; k++)

{ Ck= candidates generated from Lk-1
//that is cartesian product Lk-1 x Lk-1 and
eliminating any k-1 size itemset that is not
//frequent for each transaction t in database

 do
{  #increment the count of all components in Ck
that are contained in t Lk = candidates in Ck with
minSupport }

//end for each }
//end for return U
  ; }
```

In the above algorithm, we had taken Component item set size is T, and candidates are generated for Lk-1.In the above algorithm software, developers will take the function minSupport to extract the frequently appearing items. Frequent items are generated from Lk-1 using the Cartesian product and stored in the array Ck[i]. Further, we are taking the subsets of generated subsets to classify the components which are extracted from different technologies and finally, the algorithm will return the components which are ready to adapt.

Banking application data developed by each technology's input tools, and thus have the opportunity to know what components are typically adapted within one. This algorithm is going to figure out a new way to build a list of frequently used keyword pairs of these data [19]. After classification the database of components consist of the sets {2,3,4}, {1,2,3,}, {2,3,4}, {2,3,1}, {1,2,4}, {1,3,4}, {3,4,2}, {1,3,4,2}[20] . The first step of this tool is to find the most frequent items, called the keywords and, of each component separately and at the end result is how to form the component adaptation subsets that are shown in below figures.

Source code for banking application in **C:**

```
struct account _type
{
    char bank_name[100];
    char bank_branch[200];
    char account_holder_name[30];
    long int account_number;
    char account_holder_address[100];
    double available_balance;
};
struct account_type account[20];
```

Source code for banking application in **C++:**

```
class banking  {
long int acc;
static res;
double balance, amount, short;

public:
void depositbank();
void withdrawbank();
void chkbalancebank();
int menu1();
};
```

Journal of Theoretical and Applied Information Technology
15th May 2022. Vol.100. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Source code for banking application in **Java:**

```
public class Banking  {
   public static void main(String[] args)
   {
      bankInternal myObject = new bankInternalobject();
      myObj.depositbank();
      myObj.withdrawbank();
   }
}
```

Source code for banking application in **Python:**

```
while restart not in ('n','NO','no','N'):
print('Please Press 1 For Your Balance\n')
print('Please Press 2 To Make a Withdraw\n')
print('Please Press 3 To Pay in\n')
print('Please Press 4 To Return Card\n')
option = int(input('What Would you like to choose?'))
```

Proposed Algorithm:

Start

S1 and S2 are subsets of ∑(S1, S2) S.
S3-> (S1^S2)
if S1 and S2 are the subsets
then
S4->S1 && S2
If S1 and S2 or S4 and S3 are the Subsets
then
S->∑(S1,S2) + (S3&&S4)
S1->Selective component 1
S2->Selective component 2
S3->Adaptive component
S4-> Adaptive component
∑->superset of S1, S2
for each s1 , s3 and s4
if(s4 >s2)
then
Sk-> s2^ (s4 % s3) + s1
else
Se->Sp+Sk*s2*s3;
for all s1,s2,s3,s4
Sa=Se/(Sk + Sp);
Sa->Code based self adaptive reuse.
Sk->Code based adaptive reuse
Sp->Selective product component

end

<< ->Inner to Outer loop and vice versa

% ->Modulus Operator for Reverse the loop

& ->Similarities between two logic of the code

Table 1.Software Metrics with range values

Software Metric	Range
CEM	1.5 to 12.0
CSEM	1.0 to 9.0
CRM	2.0 to 11.0
CFM	1.0 to 10.0
CCSM	1.0 to 15.0
CCM	3.0 to 11.0

In the below Figure.1 we had taken sample banking code of different technologies and the classification measurement is set to middle the source code is supplied as the input values. The result is shown in next figure 4.

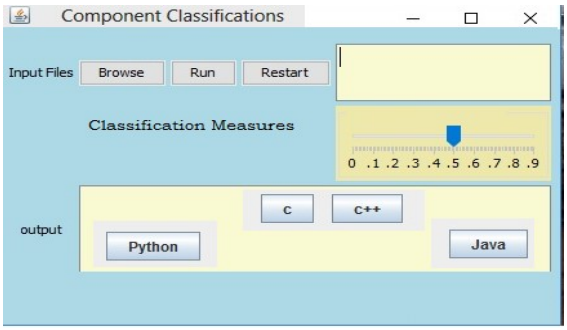

Figure 1: Illustration of Tool using Apriori Algorithm

In Figure.2, we had taken sample banking code of different technologies and the classification measurement is set to middle. The source code is supplied as the input values. And if the source code is executed with the classification we will get c and cpp are in the same cluster. The components are shown in the Figure.2 bank.c and bank.cpp are ready for adaptation because they are in the same cluster.

Journal of Theoretical and Applied Information Technology
15th May 2022. Vol.100. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

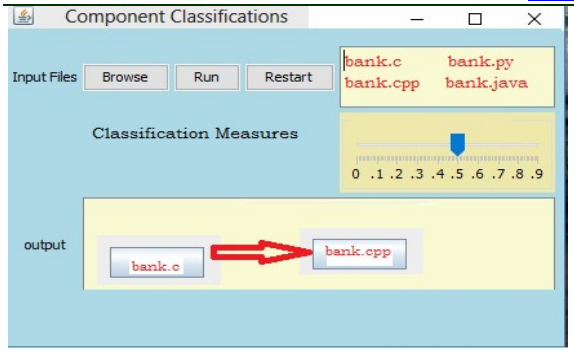

Figure 2: Illustration of selecting different technologies as input

In the above figure.2, we had taken a sample banking code of different technologies and the classification measurement is calculated using the efficiency metrics.

In Table.2, we allotted not accepted metric values for compatibility test metrics. We supplied here values out of bound. Hence none of the components is compatible with adaptation. CSEM metric, CCSM metrics are an upper bound, CRM and CCM are lower bound, CEM and CFM metrics are within the range

Table 2.Software metrics with actual values

Software Metric	Not accepted metric values
CEM	3.0
CSEM	30.0
CRM	0.0
CFM	2.0
CCSM	20.0
CCM	0.0

In Figure.5 given in the form of various high value for the different criteria and finally, none of its components are adaptive in nature this way because of our selection of metrics not satisfied the requirements. Hence none of the components for adaptive is selected.

In the below Figure.3, we had taken the values within the range of all metrics. The graph will provide the pictorial representation of metric values when the metrics are having actual values of the given range. This graph will show the metric vs. resource when the values are in within the range of actual.

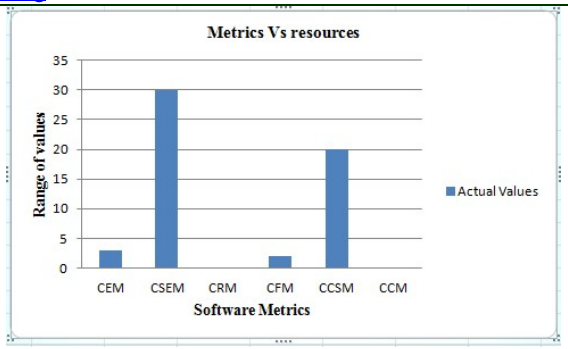

Figure.3 Metric Vs resources graph with not accepted metric values

In Figure.4 we had taken the values within the range of all metrics. The metrics we are provided are supplied as the input for the interface. The graph will provide the information about the software metric vs resources when the metrics are in the given range that is shown in Figure 4.

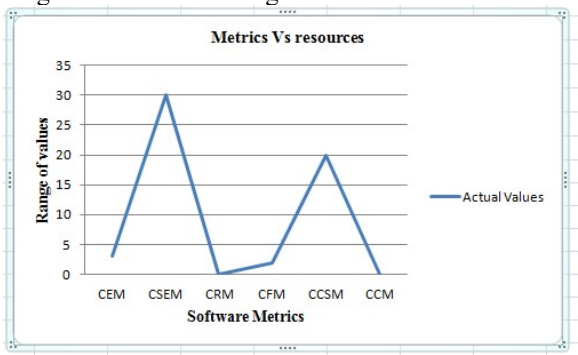

Figure.4Metrics Vs resource actual value graphs

In Figure.5, we have been assigned the values within the range of all metrics. The metrics we are provided are supplied as the input for the interface. The component classification interface will generate the components which are ready for adaptation.comp2 and comp3 are ready to adapt as they get classified in the same cluster.

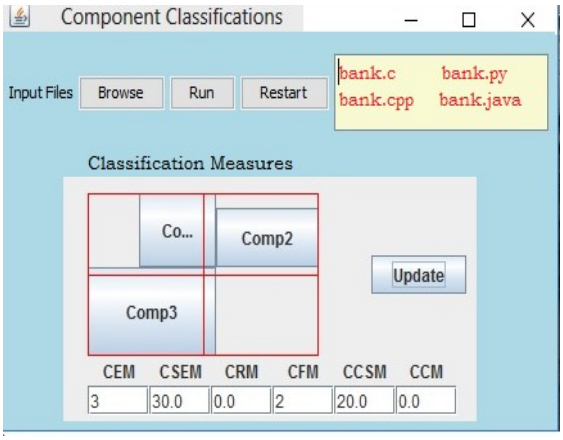

Figure 5: Illustration of Adaptive components are subsets

This is not always the case, that most popular components give better performance or better matches the needs of the first user [42]. Finally, the rating depends on the similarity of the code because it is a good choice [43]. Our algorithm uses only the keywords as the first user requirements (functional and non-functional), but also consider that the component is a value that is included in the text file pheromone and a specific impact on the performance of the components Field.

The resulting list of components not only meet the requirements of the first user because our algorithm provides better results, but also to provide an approach to the performance of the component, regardless of the popularity of the components.

A matter of security, isolation, securing, saving the time, core components, code in order to get the security applied in the form of the white box must be. The change is the most important quality of the white box applied. [44] This is the reason why the change is the principle of recovery. The white boxes are the reuse of existing components as much as cost, and the change attempts are becoming small. The component should support the user to change its properties or methods, which the components must suit their own purposes. There is a change in the components that require we to enter more time to understand the elements and therefore be able to understand it is necessary for the anxiety changes. The white Box brings their claims related to reuse and then reuse. Therefore, a very large majority of can be extended. Check the specific function of the security component. The main component is to reuse the appropriate white box.

Our functional and non-functional requirements to run the algorithm. So many tests re-conducted and the results are very similar. The results are very interesting because they show the logic to select the software components based on the similarities [45].

In addition to the general challenges associated with the development algorithm to create stochastic time, another challenge is that one algorithm is not always suitable and available to all possible situations besides this is also common for decision-making, to compare a number of algorithms to choose one that is a specific scenario or using different algorithms under different conditions. The architecture of the software is flexible and easy to iron, with these required solutions. The challenge is to solve possible conflicts between different algorithms, entering their data and their export requirements. Not an interesting task . have developed a system part of the design, their algorithms and procedures are independent components connected to an ill-defined interface as specified before every three steps. Where we can work freely and their results are meaningful, this may be interested in the end user. Only the last step is the time set[46].

There are also additional coupled components such as storage (for example, time series storage), User interface and Timing visualization The architecture is more flexible, such as the application can store information from different third programming languages Java, C, C++, Python. The user interface is switched better the specific platform components that are merged. Different visualization components can be developed by third parties to create other specific time sets using algorithms. Another feature of this platform is to improve Components are intended for this architecture but can be replaced by new components. This action is only used with storage components designed to handle the external I/O data files as soon as the new version of the application is built. In addition to the new emerging storage components, they can be replaced with old ones in the most easily composed components. Exportable each component is a stand-alone software that can be thoroughly tested as a separate module[47].

These changes occur in almost every aspect of society and everyone has their own needs and requirements. They need to adapt and implement change in a timely manner. Recently, the development of business processes and online social networks have become increasingly active. Software engineers participate in various regional cooperation and exchange ideas of research and expertise [48]. Software reuse requires us to predict the future needs of the software system, so the new synergies that can be built and some of their functions and features can be modularized so that engineers can easily reuse it.

5. CONCLUSION

The purpose of the clustering of components is to customize reusable software components. Components must be adjusted to reuse components. Customization should be done through clustering. Component-based system development is accomplished by using the reusable components of software systems, and the functionality supported by these components varies greatly according to quality

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and complexity. The application context of the component application is also very different.

The aim of this tool is to provide an open platform to use the adaptation of future researchers. Software reuse has made new developments in the use of software adaptation, business processes, and efficient technologies, encourage researchers to enrich potential researchers and adopt new technologies, which will provide new and improved technologies based on some areas.

REFERENCES:

[1] Andrew Begel , Jan Bosch , Margaret-Anne Storey, Bridging Software Communities through Social Networking, IEEE Software, v.30 n.1, p.26-28, January 2013.

[2] Andrew Begel , Yit Phang Khoo , Thomas Zimmermann, Codebook: discovering and exploiting relationships in software repositories, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, May 01-08, 2010, Cape Town, South Africa.

[3] Hans-Jörg Beyer , Dirk Hein , Clemens Schitter , Jens Knodel , Dirk Muthig , Matthias Naab, Introducing Architecture-Centric Reuse into a Small Development Organization, Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems, May 25-29, 2008, Beijing, China.

[4] Christian Bird , David Pattison , Raissa D'Souza , Vladimir Filkov , Premkumar Devanbu, Latent social structure in open source projects, Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, November 09-14, 2008, Atlanta, Georgia.

[5] Joel Brandt , Philip J. Guo , Joel Lenstein , Mira Dontcheva , Scott R. Klemmer, Opportunistic Programming: Writing Code to Prototype, Ideate, and Discover, IEEE Software, v.26 n.5, p.18-24, September 2009

[6] Constantinou, E., Naskos, A., Kakarontzas, G., Stamelos, I.: Extracting reusable components: A semi-automated approach for complex structures. Inf. Process. Lett. 1153, 414---417 2015.

[7] Laura Dabbish , Colleen Stuart , Jason Tsay James Herbsleb, Leveraging Transparency, IEEE Software, v.30 n.1, p.37-43, January 2016.

[8] David Garlan , Robert Allen , John Ockerbloom, Architectural Mismatch: Why Reuse Is So Hard, IEEE Software, v.12 n.6, p.17-26, November 1995.

[9] David Garlan , Robert Allen , John Ockerbloom, Architectural Mismatch: Why Reuse Is Still So Hard, IEEE Software, v.26 n.4, p.66-69, July 2009.

[10]Hans-Jörg Happel , Thomas Schuster , Peter Szulman, Leveraging Source Code Search for Reuse, Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems, May 25-29, 2008, Beijing, China.

[11]Reid Holmes , Robert J. Walker, Systematizing pragmatic software reuse, ACM Transactions on Software Engineering and Methodology (TOSEM), v.21 n.4, p.1-44, November 2012.

[12] Oliver Hummel , Colin Atkinson, Using the b as a reuse repository, Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components, June 12-15, 2006, Turin, Italy.

[13]Korra, Sampath, A. Vinaya Babu, and S. Viswanadha Raju. "The adaptive approach to software reuse." *Contemporary Computing and Informatics (IC3I), 2014 International Conference on*. IEEE, 2014.

[14]Andrew J. Ko , Robert DeLine , Gina Venolia, Information Needs in Collocated Software Development Teams, Proceedings of the 29th international conference on Software Engineering, p.344-353, May 20-26, 2007.

[15]Charles W. Krueger, Software reuse, ACM Computing Surveys (CSUR), v.24 n.2, p.131-183, June 1992.

[16]Otávio Augusto Lazzarini Lemos , Sushil Bajracharya , Joel Ossher , Paulo Cesar Masiero , Cristina Lopes, A test-driven approach to code search and its application to the reuse of auxiliary functionality, Information and Software Technology, v.53 n.4, p.294-306, April, 2011.

[17]Josip Maras , Maja Štula , Ivica Crnković, Towards specifying pragmatic software reuse, Proceedings of the 2015 European Conference on Software Architecture Workshops, September 07-11, 2015, Dubrovnik, Cavtat, Croatia.

[18]Nan Niu , Steve Easterbrook, Exploiting COTS-Based RE Methods: An Experience Report, Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems, May 25-29, 2008, Beijing, China.

[19]Niu, N., Jin, X., Niu, Z., Cheng, J.-R., Li, L., Kataev, M.: A clustering-based approach to enriching code foraging environment. IEEE Trans. Cybern. to appear.

[20]Nan Niu , Anas Mahmoud , Gary Bradshaw, Information foraging as a foundation for code navigation (NIER track), Proceedings of the 33rd

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

International Conference on Software Engineering, May 21-28, 2011, Waikiki, Honolulu, HI, USA.

[21]Niu, N., Savolainen, J., Niu, Z., Jin, M., Cheng, J.-R.: A systems approach to product line requirements reuse. IEEE Syst. J. 83, 827-836 2014

[22]Niu, N., Yang, F., Cheng, J.-R., Reddivari, S.: Conflict resolution support for parallel software development. IET Softw. 71, 1-11 2013

[23]M. Morisio, M. Ezran, and C. Tully, "Success and Failure Factors in Software Reuse," IEEE Transactions on Software Engineering, vol. 28, no. 4, pp. 340-357, April 2002

[24]Juha Savolainen , Nan Niu , Tommi Mikkonen , Thomas Fogdal, Long-Term Product Line Sustainability with Planned Staged Investments, IEEE Software, v.30 n.6, p.63-69, November 2013.

[25]Jonathan Sillito , Gail C. Murphy , Kris De Volder, Asking and Ansring Questions during a Programming Change Task, IEEE Transactions on Software Engineering, v.34 n.4,p.434-451, July 2008.

[26]Algestam, H., Offesson, M., Lundberg, L.: Using Components to Increase Maintainability in a Large Telecommunication System. Proc. 9th International AsiaPacific Software Engineering Conference (APSEC'02), 2002, pp. 65-73.

[27]Baldassarre, M.T., Bianchi, A., Caivano, D., Visaggio, C.A., Stefanizzi, M.: Towards a Maintenance Process that Reduces Software Quality Degradation Thanks to Full Reuse. Proc. 8th IEEE Workshop on Empirical Studies of Software Maintenance (WESS'02), 2002, 5 p

[28]Basili, V.R: Viewing Maintenance as Reuse-Oriented Software Development. IEEE Software, 7(1): 19-25, Jan. 1990.

[29] Bennett, K.H., Rajlich, V.: Software Maintenance and Evolution: a Roadmap. In ICSE'2000 - Future of Software Engineering, Limerick, 2000, pp. 73-87.

[30]Damian, D., Chisan, J., Vaidyanathasamy, L., Pal, Y.: An Industrial Case Study of the Impact of Requirements Engineering on Downstream Development. Proc. IEEE International Symposium on Empirical Software Engineering (ISESE'03), 2003, pp. 40-49.

[31]Jørgensen, M.: The Quality of Questionnaire Based Software Maintenance Studies, ACM SIGSOFT - Software Engineering Notes, 1995, 20(1): 71-73.

[32]Lehman, M.M.: Laws of Software Evolution Revisited. In Carlo Montangero (Ed.): Proc. European Workshop on Software Process Technology (EWSPT96), Springer LNCS 1149, 1996, pp. 108-124.

[33]Lientz, B.P., Swanson, E.B., Tompkins, G.E.: Characteristics of Application Software Maintenance. Communications of the ACM, 21(6): 466-471, June 1978.

[34]Malaiya, Y., Denton, J.: Requirements Volatility and Defect Density. Proc. 10th IEEE International Symposium on Software Reliability Engineering (ISSRE'99), 1999, pp. 285-294.

[35]Basalla, G. (1988) The Evolution of Technology, Cambridge University Press, New York. Brown, J. S. & Duguid, P. (2000)

[36]K.Venugopal Reddy, Sampath Korra,"Object-Oriented Analysis and Design Using UML", BS Publications, 2018.

[37]Dawkins, R. (1987) The Blind Watchmaker, W.W. Norton and Company, New York - London. Fischer, G. (1987) "Cognitive View of Reuse and Redesign," IEEE Software, Special Issue on Reusability, 4(4), pp. 60-72.

[38]Fischer, G. (1994) "Domain-Oriented Design Environments," Automated Software Engineering, 1(2), pp. 177-203.

[39]Knowledge-Based Design Environments, Ph.D. Dissertation, Department of Computer Science, University of Colorado at Boulder, Boulder, CO. Greenbaum, J. & Kyng, M. (Eds.) (2011)

[40]Design at Work: Cooperative Design of Computer Systems, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ. Grudin, J. (1994) "Groupware and social dynamics: Eight challenges for developers," Communications of the ACM, 37(1), pp. 92-105.

[41]Henderson, A. & Kyng, M. (1991) "There's No Place Like Home: Continuing Design in Use." In J. Greenbaum & M. Kyng (Eds.), Design at Work: Cooperative Design of Computer Systems, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, pp. 219-240.

[42]Henninger, S. R. (1993) Locating Relevant Examples for Example-Based Software Design, Ph. D Dissertation, Department of Computer Science, University of Colorado at Boulder, Boulder, CO. Kintsch, W. (1998).

[43]Comprehension: A Paradigm for Cognition, Cambridge University Press, Cambridge, England. Nakakoji, K. –July 2003.

[44]The Role of a Specification Component, Ph.D. Dissertation, Department of Computer Science, University of Colorado at Boulder, Boulder, CO. Nardi, B. A. (1993) A Small Matter of Programming, The MIT Press, Cambridge, MA.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[45]B.H. Liskov and S.N. Zilles, "Specification Techniques for Data Abstractions," IEEE Transactions on Software Engineering, vol. SE-1, no. 1, March 1975, pp. 7-19.

[46]Sullivan,K.J.;Knight,J.C.;"Experience assessing an architectural approach to large-scale, systematic reuse," in Proc. 18th Int'l Conf. Software Engineering, Berlin, Mar. 2006, pp. 220–229

[47] D'Alessandro, M. Iachini, P.L. Martelli, "A The generic reusable component: an approach to reuse hierarchical OO designs" appears in: software reusability,1993

[48] Pamela Samuelson, "Is copyright law steering the right course?," IEEE Software, September 2016, pp. 78-86.