# A Novel Approach For Component Classifications And Adaptation Using JALTREE Algorithm

**B.Jalender**
**Associate Professor**
**VNRVJIET,Hyderabad**

**Dr.A.Govardhan**
**Professor&Rector**
**JNTU,Hyderabad**

**Summary**
Component adaptation is widely recognized as one of the main problems of the components, used in component based software engineering (CBSE). We developed methods to adjust the components classified by the keywords. Three main methods are discussed in this article those methods are combined with several domain component interfaces, high level simple notation for the adapter design patterns. The automated process for classifying high-level components are using adaptation is novel to software engineering domain. All Specifications and many technologies for re-using software, CBD and further developments have been emerged in recent years. The effects of these technologies on program quality or software costs must be analyzed. The risk concerns a single technology and must identify its combinations. In this paper, we are going to discuss the methods to adapt components of different technologies

***Key words:***
*Adaptive, domain, component, reuse, technologies*

## 1. Introduction

The key benefits of Component based software Engineering is to support for the design methods that are used for adaptive reuse .Reuse construction of the new system need not start from scratch, but the modification of the integration and the description of the existing ones. CBSE was used to support evolution of the components of the different technologies [1]. However, it is sufficient to keep the store recently. The modification of the components is widely accepted as one of the main problems of the CBSE. The ability for 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 in general [2].

A platform that focuses on components (such as CORBA, COM, JavaBeans,React,.net) that address common collaboration using the Interface Definition Language (IDL) to identify the proposed functions (and requests) from different programs. 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 behaviour of the associated components. In addition to tests based on the case of component compatibility, more stringent techniques are required to maximize their integration of craft activities into engineering. For example, the system developer must verify that the integration of third-party components may suggest a new technique in the application being developed. To determine the properties of the system that contain a large number of interactive elements, the official description of the interaction characteristics of the component is required [4].

Software Reuse enables us to make improve the reuse components of software and reduces the cost of software. Reuse allows us to make the characteristics of the software artifacts available from not to building a computer system from scratch. Reuse of program means reusing the inputs techniques, and outputs after the software development effort. Industry control of the operations of a number of reusable components they require to build software quickly and cost effective. If not, reuse resources effectively becomes inefficient and takes time. The source of reusable components to make repository to use whenever necessary of that particular components [5].

Development of repository provides the components to select when it required. With the assurance that at one time the relationship will protect to adapt, ensure that the corresponding application is built using the components and those should not be changed because of changes in the components. A new and added functionality we have the components to make it easier to complete the application and use of this components and no changes of downtime [6].

## 2. Related Work

Developing software is a concept that creates in the system. The principles focus on the creation and evolution of software. The CBSE focus reuse of an software. It is going to adapt the new techniques such a way that component classification becomes easier. The CBSE focus in on the releases new development process of the

Manuscript received February 5, 2022
Manuscript revised February 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.2.15

components. In the same manner, allowed for a time for completion of work, the development of software projects quickly [7].component needs to change or adapting to a new system, just make code change only. If you need to patch a program, someone is the adaptability of the system means that the system can easily adapt to a diverse environment. Object-oriented software can easily adapt to new requirements because of the high level of abstraction.

It model problems with set of types or classes from which objects are created. It has focused the creation and rapid evolution of the system. There is no step icon organize in this process. A similar pattern of development of the software and development speed. Adaptability grew up in rapid application development. Adaptive Software Development overall focused on the problem of establishing the self-absorbed, sharing ideas among 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 at different stages of the life cycle of different generations of the same project. Sometimes uses the term "vertical reuse" in a more limited way. Successful vertical reuse requires detailed information about the application domain [9].

The good software recovery process facilitates increased productivity, reliability, quality, performance and cost reduction, effort, risk and implementation time. The primary investment is necessary to start the software recovery process, but this investment is amortized over some recovery. It creates a repository and recovery process of knowledge base, which improves the quality after each recovery, reduces the required development of work for the future projects, 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. Domain and application environments are relatively stable, so reuse is considered appropriately [11][12].

Software reengineering 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 indicate that the existing system needs to be modified. Reengineering is associated to the reuse of software, as systems must understand in part or in full before they can be converted or reused. However, reengineering usually requires more changes than the ones who 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 candidate 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 heavy burden 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].

Foundation, the lack of basic planning steps allows the development to software quickly. Now in some cases on the internet and do not serve the necessary detail, not a problem. The cycle of software in this process is very short for a new version with additional equipment could come quickly. Method or the 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 end point. As with the development of the software, there is no truth, only the endpoint that did not require software or code is ported to a request generation. On the other hand, rapid development application that allows for the end of a job, free from the problems that have software which meets the requirements of the end user [18][19].

Developing software with the three steps, each evolving around the coding of a program. The first step is speculation. In this section, try the coders 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. There is no available report; the project uses the basic requirements set by the end user [20].

The brainstorm is the ensure the individual projects they are doing and how to combine their own place. The project does not need any additional information or outside contributions and determine the part of the software [21].

During the training phase, the release of new version of the software to use. These assets will be improved and use the reports used in the first part of the project, and the cycle repeated itself [22].

An additional analysis of the software component before adding it to the recovery library can lead to significant savings on part of the life cycle, especially if component is reused so many times, thus the reusable component is obtained in combination with domain analysis [23]. And re-use library assessment requires further investigation before the system is included. This additional 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. This 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 from the efficiency of the documents used in the feature [26].

Consider the source code of the module, which is used for adding the recovery library. Good software engineering practices require a description of the module interface and the maintenance cost of that software. However, this document does not apply to reusable components for a variety of reasons [27].

Many organizations use programming practices and require each source code to be tested against organizational standards. This may mean that every decision in the Tree module must perform some test results during the test. If there are no errors, the module is assumed to be correct. Now suppose that 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, potential 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 in accuracies [24].

The classification of reusable software components should be based on at least 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's say that the source code is satisfactorily tested by the development of organization and is at the present considered as candidate for recovery [30].

The metrics should be about test ability, easy pairing with other module, and the probability that the source code of the module is complete if it is placed in the reuse of the library. Portability is considered to be an ideal feature of most software. In the era of ubiquitous computer 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].

## 3. Proposed Work

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

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

Recovery and modification of individual components it 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 are spaces and solution problems. The description and the satisfaction criteria are defined in the research objectives. When the problems and requirements of the official description component, the satisfaction criteria can be defined by mathematical techniques [39]. The evaluation of the correction is to determine if a component meets the specifications of a problem and how does it. Because of problems and components, the evaluation of the correction is the process to determine, if a component can be used to solve the problem. Evaluation of the correction is best simulation for satisfaction model of formal specifications and 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 and the success achieved. Successful CBD is widely accepted as a promising method of software re-use. Recently, the CBD has been presented as a complex and adaptable solution for the corporate IT system of buildings. Another new programming paradigm, feature-oriented programming is designed to be a modular feature. Provides the mechanism for implementation and execution the characteristics that make up the concept of representative fields [41].

Most existing studies only analyze a subset of the object-oriented concepts evaluate the design quality of reusable components. Almost all studies are taken into account but the important features of the object-oriented paradigm excessive focus on implementing languages C++ and Java. Deep observation has been made in this study consider all the basic concepts of the sample and measure it at design time as common as possible (independent of any implementation language). Measurement trend analyze during the evolution of industrial strength software components over a period of time. Software metrics help measure the properties of a program. The metrics have two types of reuse-oriented paradigms have been studied one is component-based software development (CBSD), and another one is object-oriented software development (OOSD)[42].

Component-based software metrics discuss 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 C++ and Java are object oriented programming. 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 cohesive, which helps to develop an object-oriented program easy 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 [42].

### 3.1 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.

### 3.2 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 must be reusable. Again, automatic recovery improvement is essential. Eventually, it produces components that are potentially reusable [33].

### 3.3 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. More accurate, the larger the component, the less likely it is to be reused across multiple applications. In many cases, it's hard to find the perfect fit [34].

### 3.4 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 unaltered parts of the component 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.

A logical reuse repository in the component library that stores reusable components and 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 [41]. However, finding and re-using 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 the components. This applies not only to the comprises code, but also is manufactured in such a way that the products define the system's life cycle, in the form of specifications, requirements and design.

The components in question are intended to be re-used in the visualization system and include code, documentation, design, requirements, architecture, etc. Creating repositories of reusable software implementation of a classification scheme to create a library and provide components of the scanning and recovery interface. The main requirement is the compositional classification mechanism. The system must fulfils three functions: load components, download components and search for software components [40].

Object-oriented programming languages provide another form of reuse. C, C++,Java, python and React framework used for classification of components . 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 changes, 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 to create new subclasses in super classes. The inheritance process encourages specific methods for reusing previously defined data attributes and processes [41].

## 4. Experimental Results and Discussions

In this article we are proposing a novel approach for component classification and adaptation using JALTREE algorithm .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, 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. We had developed a tool that will use the input as source code and will provide a series of components that will be adjusted according to the requirements [41].

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.

### 4.1 Adapting components through JALTREE

JALTREE as a new technique and very suitable technology for adjusting the components of a reusable component system. The JALTREE principle is that the functions of the components and reusable components of the domain are two independent units on the one hand and on the other must be closely integrated.

Based on the above observations, we found that the partially based software engineer requires multiple types of customization of the reusable addition, along with a variety of reusable components. This type of control must be configured and can be combined to allow the adaptation of complex components [41].

JALTREE tool provide a classification of connected components and technologies. Underlining the components which imports and expresses the source code. Software components are more than just functions and classes having group together to get the classification [42].

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 [41].

In this Article we used JALTREE algorithm for developing tool

```
Pseudo-code: C
k: Component  itemset of size k L
k : frequent itemset of size k L 1 = {frequent items};

for ( k = 1; L k != Ø; k++)
{
do
begin
Ck+1 = components  generated from L k;
for each transaction t in database
end do
}
do
increment the count of all components in Ck+1 that are contained in t
Lk+1 = candidates in Ck+1 with min_support end
end do
return U k L k
```

In the above algorithm we had taken Component item set size if K L from that one we are going to extract the keywords and frequent items .frequent items are generated from Ck+1 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.
Sample code for Banking application in **C:**

```
struct acc_type
{
   char bank_name[20];
   char bank_branch[20];
   char acc_holder_name[30];
   int acc_number;
   char acc_holder_address[100];
   float available_balance;
};
struct acc_type account[20];
```

Sample code for Banking application in **C++:**

```
class bank {
int ac;
static reset;
float balance, amount, short a;

public:
void deposit();
void withdraw();
void chkbalance();
int menu();
};
```

Sample code for Banking application in **Java:**

```
public class Bank {
   public static void main(String[] args)
   {
      bankInternal myObj = new bankInternal();
      myObj.deposit();
      myObj.withdraw();
   }
}
```

Sample 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 Withdrawn\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?'))
      if option == 1:
```

The basic concepts of C language and Python are similar and C++ and Java are similar because C++ and Java are object oriented programming. In this below figure we are showing the different domains in the Software Industry.

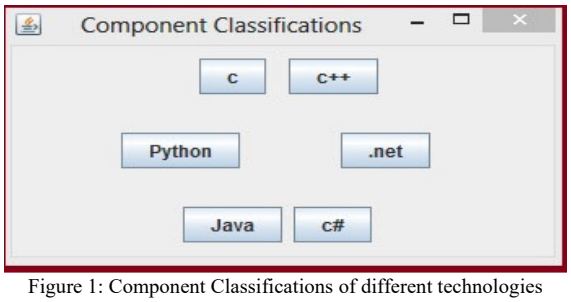



Figure 1: Component Classifications of different technologies

In this section 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 in shown in next figure.

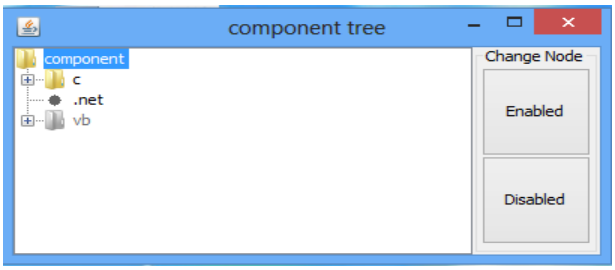



Figure 2: Component adaptation through JALTREE components without cluster

In the above section 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 below figure.bank.c and bank.cpp are ready for adaptation because they are in the same cluster.

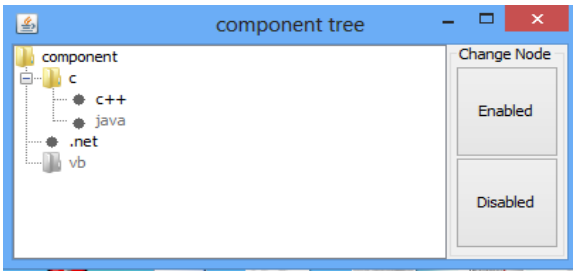


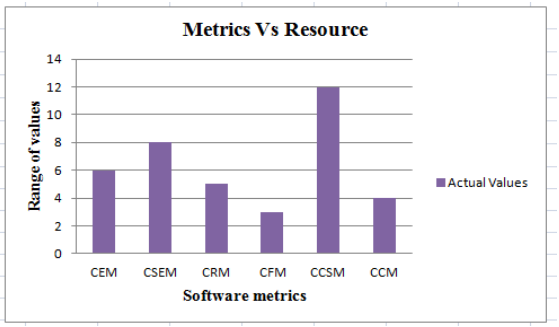

Figure 3: Component adaptation through JALTREE components with cluster

In the below table3 we supplied the values for component compatibility test metrics. In the second result we supplied the values within the range . Hence the components are compatible for adaptation.

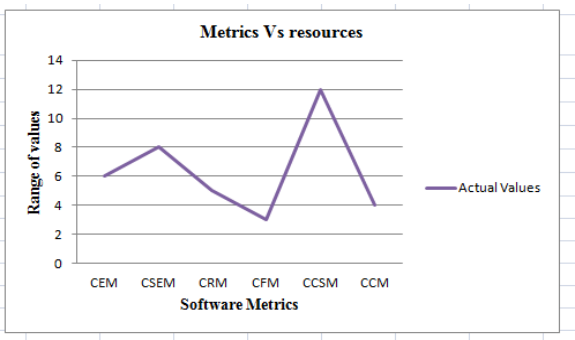
| Software Metric | Actual Values |
|---|---|
| CEM | 6.0 |
| CSEM | 8.0 |
| CRM | 5.0 |
| CFM | 3.0 |
| CCSM | 12.0 |
| CCM | 4.0 |

Table 3. Software metrics with actual values

In the below Graph we taken the values with in 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 with in the range of actual.




In the above table we had taken the values with in 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.




## 5. Conclusions

The main purpose of this article is to bring the meaning of the automatic development method of the adapter with the ability to solve the behavior between different interaction components. Our work is based on a good research flow that supports the application of official methods to explain the interaction features of the software system. More specifically, we perform a way to fill the interface components with behavioral explanations to facilitate the analysis of the system and, in general, the validation and detection behavior, particularly the reference, but slightly more closely.

A number of targeted academic studies are diagnosing problems. Different domains are found for classification of the components. Manual Adjustment of third-party components for resources in the context (can be serious) different (also the problem of software regulation is determined in particular by the work, which is considered the starting point for our work. We developed this tool which constitutes the group of software that is important in the context of classification theory. The connection between the components is made by the overlap that determines the polymorphism during operation in both components. Polymorphisms and inheritance is similar to our mapping, even if the type of adjustment is more restrictive: they can not remember the previous action or data, or change the different behavior at the protocol level, simply changing the type of translation of the name similar to that provided by the description of the signature.

Our approach will enhance the ability to adjust components by combining expressiveness and effectiveness and reliability. For the future work, this is recommended to expand the adaptation framework for different technologies.

## References

[1] 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.

[2] 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.

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

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

[5] 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.

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

[7] 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.

[8] Lientz, B.P., Swanson, E.B., Tompkins, G.E.: Characteristics of Application Software Maintenance. Communications of the ACM, 21(6): 466-471, June 1978.
[9] Malaiya, Y., Denton, J.: Requirements Volatility and Defect Density. Proc. 10th IEEE International Symposium on Software Reliability Engineering (ISSRE'99), 1999, pp. 285-294.
[10]Basalla, G. (1988) The Evolution of Technology, Cambridge University Press, New York. Brown, J. S. & Duguid, P. (2000)
[11]The Social Life of Information, Harvard Business School Press, Boston, MA. Curtis, B., Krasner, H., & Iscoe, N. (1988) "A Field Study of the Software Design Process for Large Systems," Communications of the ACM, 31(11), pp. 1268-1287.
[12]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.
[13]Fischer, G. (1994) "Domain-Oriented Design Environments," Automated Software Engineering, 1(2), pp. 177-203
[14]Knowledge-Based Design Environments, Ph.D. Dissertation, Department of Computer Science, University of Colorado at Boulder, Boulder, CO. Greenbaum, J. & Kyng, M. (Eds.) (2011)
[15] 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.
[16] 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.
[17]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)
[18]Comprehension: A Paradigm for Cognition, Cambridge University Press, Cambridge, England. Nakakoji, K. (1993) Increasing Shared Understanding of a Design Task Between Designers and Design Environments:
[19]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.
[20] 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.
[21]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. 1996, pp. 220–229
[22] Schmidt, D. C., Why Software Reuse has Failed and How to Make it Work for You [Online], Available: http://www.flashline.com/content/DCSchmidt/lesson_1.jsp
[23] Douglas Eugene Harms "The Influence of Software Reuse on Programming Language Design" The Ohio State University 1990.
[24] "Breaking Down the Barriers to Software Component Technology" by Chris Lamela IntellectMarket, Inc
[25] D'Alessandro, M. Iachini, P.L. Martelli, "A The generic reusable component: an approach to reuse hierarchical OO designs" appears in: software reusability,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] Gomaa, H., Kerschberg, L., Sugumaran, V. et al. Autom Software Eng (1996) 3: 285.
[29] A. Kumar, "Software Reuse Library Based Proposed Classification for Efficient Retrieval of Components," *International Journal of advanced research in computer science and software engineering," 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] Bosch, Jan. Design and use of software architectures: adopting and evolving a product-line approach. Pearson Education, 2000
[32] Merijn de Jonge, To Reuse or To Be Reused Techniques for Component Composition and Construction,2003 pages 57-58.
[33] Roberto A. Flores-Mendez, "Towards a Standardization of Multi-Agent System Frameworks"
[34] Grady H. Campbell, Jr. Adaptable Components (1999) { *Proc. 21st Intl. Conf. Soft. Eng.*, Association for Computing Machinery, 1999, pp. 685-6 }
[35]Kelly T.P. and Whittle B.R. (1995) *Applying lessons learnt from Software reuse to other domains*. The Seventh Annual Workshop on Software Reuse. 28-30 August 1995.St. Charles, Illinois, USA.
[36] Marius, Lucian-lonel, "Multi-criterion Analysis of Reference Architectures and Modeling Languages used in Production Systems Modeling, IEEE, 2005.
[37] Mugurel T. Ionital, Deiter K. Hammer, Henk Obbink, "ScenarioBased Software Architecture Evaluation Methods: An Overview", Technical University, Eindhoven, 2003
[38] Pragnesh Jay Modi, Spiros Mancoridis, William M. Mongan, William Regli, Israel Mayk, "Towards a Reference Model for AgentBased Systems", ACM, 2006 .
[39] Rem William Collier, "Agent Factory: A Framework for the Engineering of Agent-Oriented Applications", 2002.
[40] Richard N.Taylor, Will Tracz, Lou Coglianese, "Software Development Using Domain-Specific Software Architecture", ACM, 1995
[41] Oliver Hummel , Colin Atkinson, Using the web as a reuse repository, Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components, June 12-15, 2006, Turin, Italy .
[42] Shahanawaj "AhamadEvolutionary Computing Driven Extreme Learning Machine for Objected Oriented Software Aging Prediction" IJCSNS Vol. 22 No. 1 pp. 781—78-2022.
[43] Ch. Kishore Kumar , Dr. R. Durga "Estimation of Software Defects Use Data Mining-Techniques of Classification Algorithm" IJERT Vol. 10 Issue 12, December-2021.