# A Novel Black Box Approach For Component Adaptation Technique

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

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

**Summary**
There are several ways to improve software performance by using existing software. So, the developments of some programs are the most promising ways. However, traditional part programming studies usually assume that the components are recycled "as is". Existing models of component objects only provide limited support for partial adjustments, namely white box technologies ( copy-paste & inheritance) and the black-box methods (such as mixing and encapsulation). These technologies have problems related to recovery, efficiency, implementation of indirect costs, or their own problems. This paper suggests as JALTREE, The Black Box adaptation technology, which allows us for the implementation of previous components, but we need configurable the interface types, for measuring the adaptability. In this article we discussed the types of adjustments including component interfaces and component composition. An example of customizing JALTREE and component can be illustrated in several examples

***Keywords:***
*Adaptation, Software Reuse, Component, White box, Black box*

## 1. Introduction

CBSE is more and more focused on software engineering in the world. The main goal of CBSE is to create a set of reusable software components may be used to develop component based (CBD) applications [1].

In this article, we are going to discuss the techniques described of CBSE not sufficient to cope with all the necessary adjustments without showing potentially significant problems. Examples of these problems are the failure to reuse components because they cannot be adapted and the specifications themselves are adapted. In addition, software engineers can make significant efforts to understand the components before they can adapt. In addition, non-transparent adaptive techniques can cause self-problems. Finally, adjusting components can require a lot of code and very simple behaviors, such as messaging messages [2].

To answer these questions, we had introduced the term JALTREE, which allows software engineers to set up previous but configurable types of functional components at one place. For example, switching behavior is an interface to adjust the design pattern of the adapter and the design pattern adapter is used by customer specific interfaces. JALTREE allows software designers to customize components by identifying new types of adaptive behavior; software engineers can define a new type of adaptation. Finally, software engineers can create different types of behavior for individual components [3].

We believe that the contribution of this document is to determine the requirements that must be met through component adaptation technologies and to identify the problems related to the technology of adaptation to traditional components. In addition, JALTREE offers a new, additional technology. Finally, we propose a number of useful behavioral modes of Java, C++, Python and C languages that can improve recovery components and support direct language on JALTREE [4].

### 1.1 Systematic Reuse

Systematic reuse of software has become a promising approach to improving the productivity and quality of software development. Many large companies have launched systematic re-use programs and have developed a number of reuse frameworks to help organizations carry out these activities. However, in practice, it is difficult to achieve system recovery. In this paper, we believe that the algorithm will give the incentive conflict associated with traditional reuse procedures [5].

### 1.2 Software Reuse

Software Engineering describes a technology collection that implements the engineering approach of building and supporting software products. The technology is used here in a broad sense, i.e. concepts, principles, development approaches, methods, tools, techniques, and even software processes. During the limited period of the project's organizational effort, people and endowed with other resources needed to produce a certain result [6][7].

Software engineering involves the development of software systems. As software systems have grown in size, new software development methods have emerged. These methods include object-oriented programming, component oriented programming, appearance-based programming, and more. These methods provide better insight into the presentation and development of software systems.
These methods are inspired by the real world and provide a rapid development method for the system [8]. In this article we are dealing with the principle of reuse, which can be applied to all types of methods. In this case,

Manuscript received February 5, 2022
Manuscript revised February 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.2.11

methods and object elements are considered to be the main concepts of software recovery. Code recovery means using existing code or components in a software system. This existing code can exist in the form of libraries, add-ons, or software developed by other users[9].

There are also different recovery methods and those type of interface has its own advantages and costs. A software system is a collection of various software modules or components that are integrated as a whole system. After adding software components, the entire software lifecycle has changed. It is now necessary to test and evaluate each component of the software separately, and if these components are already in the execution mode of some other application, it is sufficient to connect to the current application of these software components [10].

It is now necessary to experiment and evaluate the interconnection code and software components. The initial investment is necessary to initiate the recovery process, but the investment is not utilized after a certain recovery period [11].

In short, the repository and recovery process will create a knowledge base that improves quality after each recovery.Reusing the software allows programmers to take advantage of their past performance and help significantly increase the productivity and quality of the software. The contribution of this document is an indicative process designed to effectively implement software reuse. A key issue with current reuse software is the lack of a standard process model that describes the details needed to support recovery-based software development. Since the beginning of programming, the software has been restored after the application has been developed. However, the practice of re-use is mainly temporary, and any reuse is not fully realized[12]
.
Implementation of Reusable Software Component Design Software Engineering Institute will develop a reuse-based approach. This JALTREE algorithm will discusses the current direction and progress of methodological work.

### 1.3 Component Classification

The reusable software element is generally known as a component. The Components may consist of ideas, projects, source code, link libraries and testing strategies, but they are not needed. In the classification, developers must specify which components or what type of requirement components they need. These requirement components should be brought into the library, assess eligibility and, where necessary to develop the component. If the developer is satisfied with taking back the appropriate component, he or she may add it to the current system development. The component recovery system is designed to identify the exact component that is required or closest to the match in the minimum time frame using a valid query. The restored components should be available for choice.

Classification of components for recovery is more complex than the classification of books in the library. The current classification process is categorized by software components are divided into the following categories: These categories are free text, counted, property value and versatile. The selection of each method is assessed on the basis of the results described in the system of good recovery [13].

The gap between the growing demand for complex software systems and the ability to deliver high-quality software at the right time and at low cost will grow. This has led to tremendous pressure to increase the productivity and efficiency of software development. Software engineering research has been driving the challenge of accelerating software development and reducing costs for many years [14].

Opportunities are arise an effective interface for restoring the repository is necessary for the success of each component of the programming system. However, these interfaces must overcome several issues in order to achieve the goal of supporting the use of software reuse. The main problem is the recovery barrier. Assuming that the developer initiates the recovery process when needed, most current reusable repository systems are designed to be independent of the current process and programming tools [15].

The reuse system described here is based on the principles of classification and thresholds. It allows software designers to define components and restore components that are similar to the required components. An algorithm describing the component behavior to be recovered. When the reuse component is selected for the necessary operation, the recovery system enumerates the package list of the operation group. A linear search algorithm was also reported to specify a reuse list for the package. If the recommended packages do not match, the user can try to find a slightly different set of tasks for other components. This approach makes it easier for users to navigate similar components to identify the best reuse component. The proposed classification algorithm will apply the techniques that are used for developing the adaptive component. This classification uses attribute values for different parts of the component. The value of the attribute pattern is initially used to determine the classification for components, platform of operating

systems, and programming languages[16]. This classification is discussed in the below section .

An element is software that is compatible with the component model and can be used and created independently, without the need to modify it according to the combined criteria. Typical examples of specific areas components are interfaces, computer components, memory, administrators, controllers, and network services. Components can come from many disciplines, in multiple languages, or from project-style records. The component version may also exist. Due to the large number of components, we believe that the component management system is required to monitor all available component properties. To add reusable components to the system, developers must be able to find and understand them. If this process fails, recovery is not possible [17].

Sorting software allows users to organize collections of structural components so that they can be easily searched. A number of tests have been conducted to classify reusable components using a variety of methods. These methods are usually used with caution. All four main methods described (free text, attribute values, calculations, and comprehensive classifications) have advantages and disadvantages associated with them [18].

The proposed classification algorithm will apply the techniques that are used for developing the adaptive component. This classification uses façade design pattern for classify the values for different parts of the component. The architecture of the design pattern is initially used to determine the classification for components and different platformof the operating systems, and different types of programming languages [19].

The façade design pattern method allows to search for regular domains or specific domains when necessary. Classify the functionality of components using a common approach. In addition to the functional code, there is also a version of the component. A component version is directly related to its overall functionality, how it works, and how the component adapted [20].

The system also saves the specifications for each component loaded into the repository. Therefore, the system can also support keyword-based searches. If the system stores most of the functionality of a component, the system works better and can be used in a variety of ways. Systematic reuse of software is considered to solve short-term development problems without affecting performance. Research is under way to develop more user-friendly and efficient recycling system. Plan to use many tools and mechanisms to support reuse in software development [21].

Software components plays an important role in the field of software development process of the software industry. So many articles through the entire lifecycle of the process, each one has its own highlighted to achieve the work of the products. Every working recovery helps reduce costs, reduces time and reduces efforts as a benefit. If different components are developed in a variety of environments in a repository, it is imperative to design reusable components [22].

The literature addresses different approaches which have consequences. Many of them have been proposed and only the code has been addressed as part of an isolated environment. The problem of incompatibility with inherited components motivates the offer approach to production of reusable design components for the legacy components and reuse in different environments. This type of approach helps software developers develop the software faster and easier [23].

The software industry requires re-use to reduce budget and reduce efforts. This can be achieved with attractively reusable components. A number of approaches are given and only a recovery code only. However, the issue of incompatibility with legacy components was discussed with the design of reusable components of legacy components in different environments. In this article explains generate the reusable projects to determine the inherited and recovered components classification according to domain knowledge. This lead to the implementation of the methodology, which can be tested with reuse levels corresponding to different environments [24].

## 2. Component Adaptation Techniques

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 several 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 [25].

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.An object 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. Objects can grew up in rapid application development in

software engineering. Adaptive Software Development is overall team focused on the problem of establishing the self-absorbed, sharing ideas, and individuals and teams online [26][27].

Software components can be deployed and distributed in the collection class, and then adapted to the needs of accountability in our sub-class. Unfortunately, by Sub-class a class collection interface may require redeployment of the different class just because they use crumbled, parent of a subcomponent that is referred to the sub classification.

The anomaly of the sub -class is a problem as it can be completely negated the benefits based on defending the program. We propose a code called adapting language and system designed to anomaly of sub-class. Classification not leads to a new class and stand alone as sub-class, but it gives existing classes and update the collection interfaces. If approved, the new language for programs based on the similaritywords. Judging by words like C # and Java, classification can save the important conservation of the evolving relationship with the class and improve adaptability of system software [28].

When the external interface of a component is explicitly defined to reduce the subordination share that is made between the component and the encapsulation component, it is created in order to use only through the internal interface. Outside 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 between each component, they do not affect the changes in the execution of other components because they use 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 [29].

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 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 abstracted [30].

### 2.1 Adaptation Techniques

If we use a traditional object-oriented programming language, software engineers have three parts of customization techniques that can be used to modify components for reuse, namely copy and paste, inheritance, and packaging. The following paragraphs describe each technology and then evaluate the requirements identified [31].

Sub classing from a collection of dependent classes may require making use of all class component classification and its class of dependent on parent node. In order to use this if not valid class is referred to as a special sub class. The sub classes a special involvement since it can take away the benefits of inheritance. Then we suggest eliminating the anomaly of sub class with class.

An object oriented programming language accompaniment for sub classing. In diversity to sub classing the class, not to creates new component, but, give and make a class. The class is not limited to a single class and re produce across a collection of related classes, new classes all collections lead to skipped class. Therefore, the class must maintain the collection of classes in assurance that replace a program update class translation of all in class [32].

#### 2.1.1 Copy and Paste

By using this copy paste technique the component is going to provides a little resemblance to the components required by the software engineer, the most efficient method can copy the code that corresponds to the segment of the component that is going reused as the development component.

Once the code is copied, the software engineer usually makes it convenient to adapt to the context of the new component, and the other functions are defined or copied from different sources called as code cleaning technique [33].

**Combine**: The code aggregation will use adaptive functions more efficiently and use of components is very low. In those situations where it is necessary to write different types of behavior, software engineers must manually connect to the entire code [34].

**Configure**: Customizing components by copying and inserting is not a first class installation behavior and therefore cannot be configured.

**Reuse**: Adaptive reuse is not a extravagant representation and is an interrelated code that restores the component, components can be used or adapted to integrate the same behavior and replication.

### 2.1.2 Inheritance

The second method of adjusting and reusing the white box is fulfilled by inheritance. For example the inherited collection and C++ execute the condition and behavior of the component to be reuse. Another example, in the case of Python, all methods and estimates defined in super-classes are available in subclasses.

**Transparent**: transparent inheritance, because subclasses are implicitly transmitted through a super-class. If reuse components, there is no difference super classes and customer object notifications using examples of imported subclasses [36].

**Configuration**: As explained above, although adaptive behavior is described by a extravagant unit i.e. subclass and inheritance cannot be configured to accommodate a certain portion of behavior.

### 2.2 Integrated software development process

The phases in software development life cycle are analysis, specification, design, and application and all these are use on the same for developing integrated software. Well integrated, means that the design by using modified can be linked back to the model [37].

### 2.2.1 Encapsulation (Information Hiding)

Ability to hide the device and other components of the system. Components cannot be determined because the quality of the product does not depend on the method used to communicate [38].

### 2.2.2 Polymorphism

With the development of technology, different types of objects make special facts. Objects that are close to objects that represent the similar entities of the object oriented programming [39].

### 2.2.3 Wrapping

The package declares one or more than one components as part of the packaged component, but the component had only composition options and can only send a small change to the packaged component of the client's requirements. There is no clear line between the packaging and the collection, but the packaging is used to adjust the behavior of the components that are closing, and adding aggregation, can combine new features with existing components that provide related functionality. The significant drawback of packaging is that it can lead to significant overall costs, since a complete packaging of reusable components has to be a classified, including the non-personalized interface elements [41]. The required grouping is estimated as follows.

**Transparent**: The package is fully adapted to an adaptive component, the component client cannot send messages directly to the part, but must always pass the package. This needs the container to process all components that can be directed to the component, including those is need not adapted.

**Configuration**: Although adaptive behavior is provided by the primary drive, i.e. the container, it is generally not possible to configure some parts of adaptive behavior. For example, if the container requires to change the operation name of the adapter component, usually cannot configure the container with a new action name because it must be in a reusable container [42].

### 2.2.4 Differences between adapting components and classes

Object that complies with the rule (OOD) is the principle of justice for change in the system model and encourage Software Engineers to modularize code verifier to reduce the impact of changes in the future. OOD is the process in two ways to serve above purpose. First in its class with a corresponding public and private use, support a secret message class. Second, the inheritance is a mechanism which it acquires a property of one or more other objects. The programmer associated with the tools to learn of the hierarchical inheritance, do good for reusing the existing code, leading to poor design [43].

Table.1 Component adaptation techniques Vs different technologies

| Technologies | Adaptation Techniques | | |
|---|---|---|---|
| | Copy paste | Inheritance | Wrapping |
| C | Yes | No | No |
| C++ | No | Yes | Yes |
| Java | No | Yes | Yes |
| Python | Yes | No | No |
| .Net | No | No | No |

Software Architecture and inheritance are the different modeling to be used in a variety of architectural description languages (ADLs), such as ACME Architectural model[46] to specify that when the interface inheritance now notably, the use of objects associated with the particular type [44].

However, there is lot of difference between Component Based Software Engineering (CBSE) and the Object Oriented Programming (OOP) is to wish the engineers to adapt an existing component that complies with the software needed to perform the difficult task of understanding the class hierarchies. Especially, the adapter must determine the portion of the class to do better to make the change when the first commitment is not broken. Often, additional classes were added to the page in order not to change the structure of the class was better for modifications to the existing classes.

Therefore, there is a tacit warning and the objects associated with the designer's technology system and the maintainer adapter is same. We seek to find ways for a builder to use practice and protect the knowledge only and its related documentation [45].

## 3. PROPOSED ALGORITHM

Start

S1 and S2 are the 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

The above algorithm describes about the black box approach for component adaptation techniques.

S,S1,S2,S3,S4 are the components of different technologies. The compositions of two systems are represented by S[k1] and S[k2]. The final component must be configured and can be combined to allow the adaptation of complex components.

## 4. Black Box Approach For Adapting Components

JALTREE as a new technique and very suitable technology for adjusting the components of a component system. The JALTREE principle is that the functions of the components and components of the regulation 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. This section specifies the adaptation of different types of components and will be presented here [46].

Big component consists of components and other adjustments, this combination must be transparent. In other words, the party and the customer do not need to know of the existence of the corresponding entity. Furthermore, the adapted entities do not need to be aware of the existence of other adaptation measures.

### 4.1 Component Adaptation types

In our work to adjust the part, we identified three typical components, namely changes, component interfaces, component composition, and component monitoring.
In the below sections each category will be discussed in more detail.

#### 4.1.1 Component interfaces

A software component is a unit of composition with contractually specified interfaces. In C++ and java the interfaces used by the programmers using COM(Component Object Model).Using this component model we use the abbreviation of the abstract class of C++ and java technologies. These stubs inherit from the base interface abbreviation but are not inherited at the component level COM model. When you use the component again, the wrapper and position are irrelevant and you are forced to interact with the class in an abstract way. When you reuse a class for inheritance or decomposition, it can access its entire state (except for members of a private class) and does not follow the advantages of the defined interaction protocol. You are forced to learn to reuse a class implementation just as you develop it yourself. The discovery of the interface will allow the component to use different components without prior knowledge of the applied interactive protocol. When

you think of a component as a physical package that contains one or more logical components that are typically rendered through a set of interfaces.

**4.1.2. Component Composition**

Composition is not supported: there is no support for the composition at any level. In the operating system, applications are executed independently and do not need to interact with other applications. Data exchange between applications is possible due to the communication mechanism between processes. However, even if the application uses an operating system service, the application is not specified correctly.

**4.1.3 Component Monitoring**

The above discussed part of customization categories includes a change in the behavior of components to be reused, such as their interface. As the name suggests, this class focuses on tracking components, so other parts refer or rely on when a specific event occurs in the monitored component. In the following we examine the monitoring of three examples that will be mutually accessible for the reusable components, i.e. indirect calls, observer design patterns and status monitoring. The second is the first type of specialization [48].

Below figure.1shows that JALTREE is depicts as graphically. Shows a basic custom component with two types of customization. In our graphical JALTREE interface we have designed components of different technologies for C, C ++, Java and Python.


Figure.1 Component adaptation through JALTREE

Below figure.2 describes that JALTREE is represented graphically. The main components that use both types of components but the Adaptive components are encapsulated scalability, but according to the end user requirements the adaptation part is done [47].

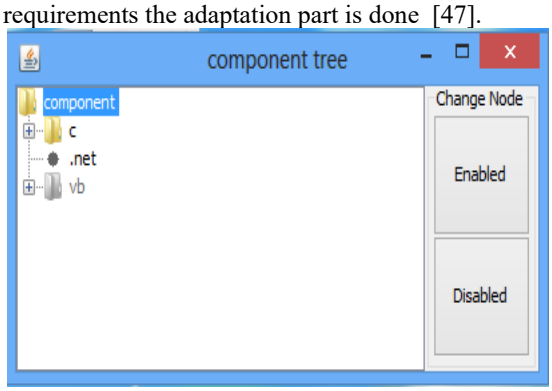


Figure 2. Component adaptation through JALTREE same components without cluster

Below figure.3 illustrates that graphically JALTREE is represented for the component adaptation. Adaptive components are encapsulated scalability and the main components that use both types of components. In below figure.3 the C, C++,and Java components are in the same subsets.And .net and vb are in the different subsets.

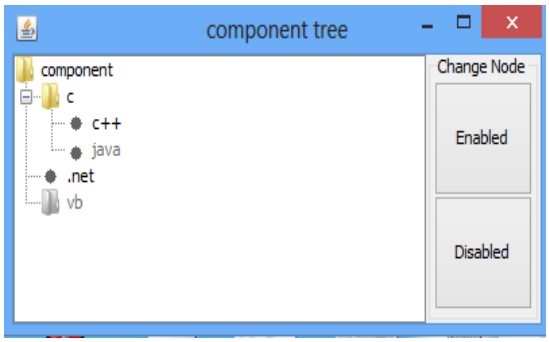


***Figure 3. Component adaptation through JALTREE same components with cluster***

**Implicit Call**: This type is actually a generic type of component monitoring. The definition of 'indirect reference' includes the direct notification of the delivery of the message or the implicit reporting of related components by generating the event, when certain conditions or transactions of a controlled component occur.

MVC model is very useful, it is assumed that software engineers know that if they define an object they stick to other objects. Construction of component systems, components sometimes have to be reused, but usually the components are not ready for this purpose. Therefore, the observer mode function must be an integral part of the other component so that it can be used as a monitoring component. Adaptive behavior is communication that is responsible for the management and involvement of objects. Only the implementation of the notification is defined in the formal provisions. The classification level

code behavior is left to the exercise of the end user. The observer's design pattern is specialized for the type of indirect notification. The informal mapping function extends the mapping of interface three elements to all related objects as defined with the observer model, the modified method is called [49] and its shown in below MVC Architecture (figure.4) .

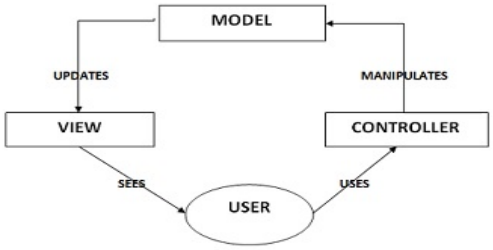


Figure 4.MVC Architecture

**Situation monitoring**: In some cases, the final end user is not satisfied the adaptive component. So component does not wish to report any changes in the relevant component, but only when the reuse levels exceed a certain limit. The traditional observation of the observer's model is not complete, but the use of it to achieve this behavior is very feasible. This allows software engineers to reuse components to determine which areas of the state should define related components.

### 4.2 Evaluation of Trees

JALTREE technique is a tree fashion technique which is used for adaptive reuse to meet the requirements in the order to overcome the traditional adaptation techniques do not meet the requirements.

**Transparent**: The Fit type, which can be cross-sectional, is completely transparent. The component identity is maintained and the adjustment type only affects those aspects of the component that need to be adjusted.

**Configurable**: The type of adjustment consists of the general part and the specific part. To select, and able to define specific sections for each instance of adjustment type. Furthermore, the general type of part adjustment is available in all cases and is therefore the most widely used level of reuse for adaptive component [50].

## 5. CONCLUSION

The Component Based Software Engineering (CBSE) is becoming increasingly important role in the field of software industries. It can effectively create applications that can reuse components. The most conventional methods accept that components in these applications that are reused "as is", but that the "rest of" reuse is unlikely and that most components need to be customized for the application. The JALTREE is a new method used for component reusability using black box technique and it is going to rectify the problems faced is the software development life cycle. Traditional component adjustment techniques are copying, pasting, packaging and inheritance. In this Article first we given the adaptation technique using white box, and the second example is given using black box adaptation technology. None of the traditional methods can meet one or more of the specified requirements.

In this paper we discussed the four parts of JALTREE that is ready to adapt the reusable components, in addition to the set of reusable components; a partially based software engineer also requires multiple types of customization of reusable components. For example, different types of adaptive behavior are proposed and defined, i.e. component adaptation, component assembly and component monitoring.

JALTREE implements the structure of tree fashion through the layered concept. JALTREE is an expanded Component Object Model that contains parts such as state and behavior of the component, Category level, including, variables and methods. Through JALTREE, we can provide software engineers with a powerful type of component customization. To illustrate this point, four types of technological adaptation have been introduced.

## REFERENCES:

[1] A. Kaur and K. S. Mann, “Component Selection for ComponentBased Software Engineering,” International Journal of Computer Applications, vol. 2, no. 1, 2015, pp. 109-114.

[2] A. Vescan, “Pareto Dominance - Based Approach for the Component Selection Problem,” Second UKSIM European Symposium on Compute, 2013, pp. 58-63.

[3] N. Haghpanah, S. Moaven, J. Habibi, M. Kargar, and S. H. Yeganeh, “Approximation Algorithms for Software Component Selection Problem,” in Proc. Asia Pacific Software Engineering Conference, 2007, pp. 159–166.

[4] A. Kumar, P. Tomar, N. S. Gill, and D. Panwar, “New Optimal Process for Selection of Software Components,” in Proc. 1st National Conference on Next Generation Computing and Information Security, jointly organized by Computer Society of India and IMS, Noida, U.P., India, 2010, pp. 376.

[5] IEEE Standards Board, “IEEE Standard Glossary of Software Engineering Terminology,” Computer Society of the IEEE, 1990.

[6] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, “Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty,” in

Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems, 2017, pp. 17-26.

[7] E. Fredericks. Machine learning and language syntax: The genetic language parser. M.s., Oakland University, 2010.

[8] Balzer, R., ``A 15 Year Perspective on Automatic Programming", IEEE Trans. on Software Engineering, vol. 11, no. 11, Nov. 1985, pp. 1257-1267.

[9] Biggerstaff, T. and A. Perlis (eds), Software Reusability (2 vols.), ACM Press / Addison-Wesley, 1989.

[10]Batory,D., etal., ``Scalable Software Libraries", Proc.ACM SIGSOFT '93: Symposium on the Foundations of Software Engineering, Los Angeles, CA, Dec. 1993.

[11] Booch, G., Software Components with Ada, Benjamin-Cummings, 1987.

[12] Dewar, R. B. K., The SETL Programming Language, manuscript, 1980.

[13] Efremidis, S. and Gries, D., ``An Algorithm for Processing Program Transformations", Tech. Report TR 93-1389, C.S. Dept., Cornell Univ.

[14] Gautier, R. and P. Wallis, Software Reuse with Ada, London: Peter Peregrinus Ltd., 1990.

[15] Goguen, J. A., ``Reusing and Interconnecting Software Components", IEEE Computer, Feb. 1986, pp. 16-28.

[16] Goguen,J.A., "Principles of Parameterized Programming", in software reusability:vol.1 concepts and models,pp.159-225.

[17] Andres J. Ramirez, Erik M. Fredericks, Adam C. Jensen, and Betty H.C. Cheng. Automatically relaxing a goal model to cope with uncertainty. In Gordon Fraser and Jerffeson Teixeira de Souza, editors, Search Based Software Engineering, volume 7515 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012., pages 198–212

[18] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, "Autorelax: automatically relaxing a goal model to address uncertainty," Empirical software engineering, 2014,pp. 1-36.

[19]Erik M. Fredericks, Andres J. Ramirez, and Betty H. C. Cheng. Towards run-time testing of dynamic adaptive systems. In Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '13, IEEE Press, 2013.pages 169–174.

[20]Erik M. Fredericks and Betty H.C. Cheng. Exploring automated software composition with genetic programming. In Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion, GECCO '13 Companion, Amsterdam, The Netherlands, 2013. ACM., pages 1733–1734

[21] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Department of Computer Science, King's College London, Tech. Rep. TR-09-03, 2009.

[22] Sandeep Neema, Ted Bapty, and Jason Scott. Development environment for dynamically re-configurable embedded systems. In Proc. of the International Conference on Signal Processing Applications and Technology. Orlando, FL, 1999.

[23] Nelly Bencomo and Amel Belaggoun. Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks. In Requirements Engineering: Foundation for Software Quality,. Springer, 2013.pages 221–236

[24]John H. Holland. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA, 1992.

[25]Conor Ryan, JJ Collins, and Michael O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Genetic Programming,. Springer, 1998.pages 83–96

[26] John R Koza. Genetic programming as a means for programming computers by natural selection. Statistics and Computing, 4(2): 2014,87–112.

[27] N. Bredeche, E. Haasdijk, and A.E. Eiben. On-line, on-board evolution of robot controllers. In Pierre Collet, Nicolas Monmarch´e, Pierrick Legrand, Marc Schoenauer, and Evelyne Lutton, editors, Artifical Evolution, volume 5975 of Lecture Notes in Computer Science,. Springer Berlin Heidelberg, 2010.pages 110–121

[28] Richard P Gabriel, Linda Northrop, Douglas C Schmidt, and Kevin Sullivan. Ultra-large-scale systems. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications,. ACM, 2006,pages 632–634.

[29] Gries, D., ``The Transform: a New Language Construct", lecture presented at the University of Texas, Feb. 18, 1991.

[30] Hille, R. F., Data Abstraction and Program Development using Modula-2, Prentice Hall, 1989.

[31] Kruchten, P., E. Schonberg, and J. Schwartz, ``Software Prototyping using the SETL Language", IEEE Software, vol. 1, no. 4 (Oct. 1984), pp. 66-75.

[32] Lamb, D., ``Sharing Intermediate Representations: The Interface Description Language", Tech. Report CMU-CS-83 129, C.S. Dept., CarnegieAuthor No.1, Author No 2 Onward, "Paper Title Here", *Proceedings of xxx Conference orJournal (ABCD)*, Institution name (Country), February 21-23, year, pp. 626-632.

[33] Bosch, Jan. "Superimposition: A component adaptation technique." *Information and software technology* 41.5 (1999): 257-273.

[34] Geert Hofstede, Culture's Consequences: International Differences in Work-Related Values (Beverly Hills, CA:Sage Publications, 1980), p. 19.

[35] M. Rokeach, The Nature of Human Values, New York, The Free Press, 1973. • H.M. Trice and J.M. Beyer, The Culture of work Organisations, Englewoods Cliffs, NJ, Prentice Hall, 1993.

[36] J.P. Kotler and J.L. Herkett, Corporate Culture and Performance, New York, The Free Press, 1992.

[37]E.H. Schin, Organisational Culture and Leadership, 2nd ed., San Francisco, Jossey-Bass, 1992.

[38]. Burke W.J. ; Merrill H.M. ; Schweppe F.C. ; Lovell B.E. ; McCoy M.F. ; Monohon S.A. IEEE Transactions on Power Systems, 1988 vol: 3 issue: 3, 1284-1290

[39]. Advertising versus pay-per-view in electronic media, Prasad, A. ; Mahajan, V. ; Bronnenberg, B., International Journal of Research in Marketing year: 2003 vol: 20 issue: 1 pages: 13-30

[40]. Consumers' trade-off between relationship, service package and price: An empirical study in the car industry, Odekerken-Schroder Gaby ; Ouwersloot Hans ; Lemmink Jos ; Semeijn Janjaap European Journal of Marketing year: 2003 vol: 37 issue: 1-2 pages: 219-242

[41].AFFONSO, F. J.; NAKAGAWA, E. Y. A reference architecture based on reflection for self-adaptive software. In: Software Components, Architectures and Reuse (SBCARS), 2013 Seventh Brazilian Symposium on. [S.l.: s.n.], 2013. p. 129–138. [In press].

[42]ANDERSSON, J. et al. Reflecting on self-adaptive software systems. In: SEAMS/ICSE 2009. [S.l.: s.n.], 2009. p. 38 –47.

[43] SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst., ACM, New York, NY, USA, v. 4, n. 2, p. 1–42, maio 2009. ISSN 1556-4665.

[44] BENCOMO, N. et al. Requirements reflection: requirements as runtime entities. In: Software Engineering, 2010 ACM/IEEE 32nd International Conference on. [S.l.: s.n.], 2010. v. 2, p. 199–202. ISSN 0270-5257.

[45] JANIK, A.; ZIELINSKI, K. Aaop-based dynamically reconfigurable monitoring system. Inf. Softw. Technol., Butterworth-Heinemann, Newton, MA, USA, v. 52, n. 4, p. 380–396, abr. 2010. ISSN 0950-5849.

[46] PENG, Y. et al. A reflective information model for reusing software architecture. In: CCCM/ISECS 2008. [S.l.: s.n.], 2008. v. 1, p. 270 –275. [22] SHI, Y. et al. A reflection mechanism for reusing software architecture. In: QSIC 2006. [S.l.: s.n.], 2006. p. 235 –243. ISSN 1550-6002.

[47] ESFAHANI, N. A framework for managing uncertainty in self-adaptive software systems. In: Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on. [S.l.: s.n.], 2011. p. 646– 650. ISSN 1938-4300.

[48] SOUZA, V. E. S. et al. Awareness requirements for adaptive systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. New York, NY, USA: ACM, 2011. (SEAMS '11), p. 60–69. ISBN 978-1-4503-0575-4.

[49] Shahanawaj "AhamadEvolutionary Computing Driven Extreme Learning Machine for Objected Oriented Software Aging Prediction" IJCSNS Vol. 22 No. 1 pp. 781—78-2022.

[50] Ch. Kishore Kumar , Dr. R. Durga "Estimation of Software Defects Use Data Mining-Techniques of Classification Algorithm" IJERT Vol. 10 Issue 12, December-2021