Journal of Theoretical and Applied Information Technology
30th April 2024. Vol.102. No 8
© Little Lion Scientific

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

A HEURISTIC-BASED METHOD FOR DEMAND-BASED REUSABLE COMPONENT DESIGN CONFIGURE: AN EMPIRICAL ANALYSIS

N MD JUBAIR BASHA[1*],Dr GOPINATH GANAPATHY[2], Dr MOHAMMED MOULANA[3]

[1]Research Scholar, Department of Computer Science & Engineering, Bharathidasan University, Tiruchirappalli, Tamil Nadu, INDIA
[2]Senior Professor, Department of Computer Science & Engineering, Bharathidasan University, Tiruchirappalli, Tamil Nadu, INDIA.

[3]Professor, Department of Computer Science & Engineering, Koneru Lakshmaiah Educational Foundation, Vaddeswaram, Guntur, Andhra Pradesh, *INDIA*

Email: jubairbasha@gmail.com*

ABSTRACT

During software development, the developer must select components from the repository that meet the needs of the customer. If the repository has legacy components that fit the client's needs, they may easily be removed and delivered to the customer. If the developer is unable to locate the specific components required, the developer must configure the equivalent components from legacy components before delivering the solution. In other circumstances, when the developer is unable to locate components in the repository that do not satisfy the customer's needs, the components must be developed from scratch. The linked work now addresses the situation in which the developer discovers components that only partially satisfy the customer's needs. The identification of components, their reusability, and their ability to be grouped with other components are all investigated in this proposed work. The use of a heuristic method in the creation of configurable reusable components is discussed in this article. The major focus of this work is on the situations in the components, with the characteristics from the scenarios being identified. Furthermore, in a facade, these functions are organized as configurable reusable components. An empirical analysis was also conducted. This makes it possible for developers to locate the configured reusable components.

Keywords: *Software Components, Configured Reusable Components, Façade, Feature Point, Lack Of Cohesion In Methods, Heuristic Function*

1. INTRODUCTION

Programmers have been reusing algorithms, subroutines, and chunks of code from previously produced programmes since the beginning of programming. Mcllory [1], who stressed the importance of component-based software systems, was the first to codify the concept of software reuse. The application of Mcllory's concept resulted in thoughts for developing software systems in the same way that hardware systems are developed (e.g. electronic circuits). Later study focused on reuse and its potential orientations, as well as the value of reuse [2, 3]. Most of the largest software development vendors, such as IBM, HP, and Motorola, have adopted reuse as one of their regular paradigms on their production lines, and many others have claimed positive results with reuse in their software development projects [4]. Software reuse is a method of generating, organizing, and locating reusable components for future development. The two major strategies for recycling software are generally recognized: developing with reuse and creating for reuse. The former strategy includes features such as classifying and searching for software components, whereas the later technique focuses on creating and producing reusable components. In reality, reuse development is a prerequisite for reuse development since you can't reuse a component that isn't even available. Until date, it appears that there has been no widely acknowledged standard for the design of reusable software components [12]. With (object-oriented) class libraries, application frameworks, and design patterns, as well as source code [7], the reuse of design is becoming more common. Two complimentary strategies of reusing legacy components were presented by Jianli.et al[10]. Many techniques, as well as design flaws in reusable domain-specific components, have been examined in earlier work. Until date, it appears that there has been no widely acknowledged standard for the design of reusable software components [12]. This encourages utilizing a heuristic-based approach to develop configurable reusable components. Looking at sequence diagrams and identifying situations from the use case diagram is how the heuristic-based method is done. For modifying the behavior of the components, feature points (FP) can be used. Developers may quickly detect the common behavior of components based on client needs by utilizing feature points (FP). Compared to the many ways outlined above [12, 13, 14, 15, 17], this can help to identify the components and make them more reusable. This paper discusses about the heuristics-based approach for demand based reusable component design configure which is performed by empirical analysis.

This paper is structured as follows. Section 2 discusses the related work with extensive literature on component reuse. Section 3 discusses the need for the proposed work and highlights the drawbacks of the earlier approaches. This section provides the motivation for this paper. Section 4 implements the proposed Heuristic-Based Method for Demand-Based Reusable Component Design Configure. Section 5 examines the mathematical results for the proposed heuristic-based approach and compares the existing approaches by conducting empirical analysis. Section 6 concludes the work with a future proposal.

2. RELATED WORK

The use of existing software or the creation of new software based on software knowledge is referred to as software reuse. Software or software expertise are both examples of reusable assets. A software asset's reusability is indicated by its reusability feature [5]. The process of designing software and reusing it [6] is referred to as software reuse. We can reduce software development complexity, improve product quality, and speed up production in the organization by reusing software. During software development, the developer must select components from the repository that meet the criteria of the client. If the repository has legacy components that fit the customer's needs, they can be removed and the product provided to them. If the developer cannot discover exactly the correct components, the product must be delivered after configuring the corresponding components from legacy components. In other circumstances, if the developer is unable to locate components in the repository that do not satisfy the customer's needs, the components must be developed from scratch. The linked work now addresses the situation in which the developer discovers components that only partially fulfill the customer's criteria. The identification of components, their reusability, and their ability to be grouped with other components are all examined in this work. With (object-oriented) class libraries, application frameworks, and design patterns, as well as source code [7], design reuse has become increasingly prevalent. Two complimentary strategies of reusing legacy components were presented by Jianli et al [10]. Component evolution is permitted among them, and this is accomplished by binary class inheritance between component modules. Semantic entities define the other, which can be built at compile time or bound at runtime. Despite the fact that component confinement remains the primary reuse mechanism that leads to the establishment of software product lines [8]. In order to locate the components again, a large amount of data must be collected, preserved, and analyzed. Maurizio et.al.[11] presented a way for automatically generating a software catalogue that includes tools for preserving and retrieving data [9]. Product reuse and Process reuse are the two major kinds of software reuse. Through module integration and design, product reuse entails reusing a software component while also creating a new component. Reusing legacy components from the repository is referred to as process reuse. These parts can be reused without modification or with minimal changes. By versioning these components, the updated software components may be preserved. Depending on the desired domain, components may be categorized and selected [10]. The use of existing software or the creation of new software using software expertise is referred to as software reuse. Software or software expertise are both examples of reusable assets. The feature of reusability reflects the likelihood of a software asset being reused [5]. The process of using "designed software for reuse" over and over again is referred to as "software reuse" [6]. We can control software development complexity, improve product quality, and speed up production in the company by reusing software.

In software development, as per the customer's requirements, the developer needs to identify the components from the repository. If the repository contains the legacy components which matches with the customer requirements can be easily extracted and deliver the product to the customer. If the developer doesn't find the exactly matching components, then it is necessary to configure the relevant components from the legacy components and will deliver the product. In other case, if the developer does not find the components which not at all matching the customer's requirements from the repository, it is necessary to develop the components from the scratch. Now the related work is concerned with the case when the developer identifies the components which are partly matching with the customer's requirements. This work explores the identification of components up to what extent they can be reusable and what extent they can be grouped with the other components?

With (object-oriented) class libraries, application frameworks, and design patterns, as well as the source code, the reuse of design has established itself [7]. Jianli et al. offered two strategies for reusing legacy components that are complimentary to one other. One of them permits component evolution through binary class level inheritance across component modules. The other is by specified semantic entity, which can be constructed or bound at runtime. Although component confinement remains the primary reuse paradigm that contributes to the creation of software product lines [8]. For the retrieval of the components, a large amount of data must be gathered, preserved, and analyzed. Maurizio has developed an approach for automatically creating a software catalogue [9], which includes tools for preserving and retrieving information. Product reuse and Process reuse are the two major kinds of software reuse. The term "product reuse" refers to the recycling of software components as well as the creation of new components as a consequence of module integration and building. The reuse of old components from the repository is referred to as process reuse. These components can be reused without modification or with minor changes. Versioning these components allows the updated software component to be archived. Depending on the necessary domain, the components can be categorized and selected [10]. Identifying objects and processes for a class of related systems for a certain domain can help enhance software reuse. Domains are areas of application in software engineering [11]. So, the domain specific components are considered as a part of this research work to carry out.

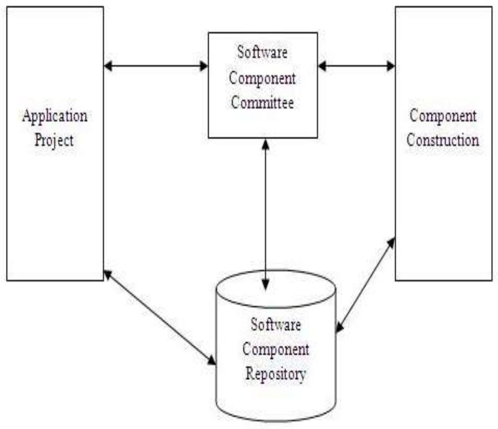

Figure 1. Organization for Component Management

A component's design is critical to its functionality. The ability to reuse something is not a byproduct. For reuse, there must be specifications, building, and testing. It costs up to ten times more to design new software for a component as a result of this. For a good component, many criteria have been proposed. The following are a few of the criteria: The component should be used to represent a concept. It should be extremely well-coordinated, with just the operations required for effective utilization. It also requires a well-defined interface in terms of both syntax and semantics. If the names of two operations in two distinct components are the same, they should function identically. Their writing style, on the other hand, should be identical for clarity. The component must be self-contained, loosely coupled, and hence have low coupling with other parts. Independence comes from an object-oriented perspective. The component should be a generic abstraction that may be used in a variety of applications without requiring any additional modifications. Internal and outward comprehension are both necessary. Good components are serviced for a long period since they have a long-life expectancy. The component system handles the selection, classification, and maintenance of components in the repository, as well as the creation of new ones. The component repository should be available to everyone in the development team. Ideally, the component repository will be shared by several products. As a result, the component system should be able to handle several projects. The creation of new projects necessitates the inclusion of the necessary components. The project proposals should be evaluated by a software component committee, which is made up of experienced designers and one representative from the components department. You should consider if the proposed components require further development. If the component's design is finalized, it is sent to the component design department with a deadline. It's then added to the component repository and given a new version status, as illustrated in Figure 1. The software component group should evaluate the value of the component when it is in use. Software systems are still grappling with the task of identifying reusable components. The domain-specific components [16] face the similar difficulty. Won Kim et al. [13] divide component-based reuse into two categories: without modification and with change. Simply selecting a component from a software component repository and inserting it into newly generated software is referred to as reusing without change. General functions in programming language libraries, such as the arithmetic functions in the C programming library, are one of the known mechanisms for this sort of reuse. The most significant cause for legacy components that cannot be reused in their original form is the functional difference. A functionally coordinated existing component and a new component to be built is rare. The new component under development may necessitate certain changes to the older components' equivalent functions, as well as the inclusion of new functions. Similar granular components, on the other hand, are unlikely to be reused without change across product lines (or business units). The described function reuse, however, poses a hurdle. Identification of most reusable software is still an issue [21,22].

2.1 MOTIVATION

During software development, the developer must identify the components from the repository according to customer requirements. If the repository contains the legacy components that meet the customer's requirements, they can simply be extracted and the product shipped to the customer. If the developer does not find exactly the right components, it is necessary to configure the corresponding components from the legacy components and to deliver the product. In other cases, when the developer cannot find the components from the repository that do not meet customer requirements at all, it is necessary to develop the components from scratch. The associated work now deals with the case that the developer identifies the components that partially meet the customer requirements The identification of components, their reusability, and their ability to be grouped with other components are all examined in this work. The developer must determine the common behavior of the components from historical components according to client requirements. Components with similar behavior can be reused. With (object-oriented) class libraries, application frameworks, and design patterns, as well as source code [7], design reuse has become increasingly prevalent. Two complimentary strategies of reusing legacy components were presented by Jianli. et al. Many techniques that do not develop reusable domain-specific components have been described in earlier work. Until date, it appears that there has been no widely acknowledged standard for the design of reusable software components [12]. This motivates designing special types of components that share a behavior that must be identified by the legacy components. Feature Oriented Software Development (FOSD) [14] uses object-oriented languages with special classes that define features and develop object-oriented systems by keeping feature objects together. AsmaaAlyaed et al. [15] has presented an approach to domain engineering that extends Feature Oriented Domain Analysis (FODA). This approach constructs the function model as an executable architecture and thus enables the definition of the reference architecture as such a function model with variability, as it is defined in the feature model. Even so, this approach of using state diagrams to aid decision-making in identifying the features is contemplated. The state diagram diagram may not identify all behaviors related to the domain-specific components. The greatest challenges in reuse are to find the right reusable device from the multitude of functions and to adapt the reusable device to current needs and licensing problems [16].Collin McMillan et.al.[17] created an approach named Portfolio for finding high relevant functions from the archive of C/C++ source code. To address the needs of programmers to reuse retrieved code as functional abstractions. However, this approach is subjected to only C/C++ programming code but not for all the object-oriented approaches. However, commonly accepted standard for designing reusable software components seems to be unrecognized widely till now [12].The above literature motivates to propose a heuristic based approach by considering sequence diagram and identifying scenarios from the use case diagram. The feature point (FP) can be considered for the change in the behavior of the components which was not discussed in any of the above approaches. This may help to identify and make the components more reusable rather than the previous discussed different approaches [12, 13, 14, 15,17]. Identification of most reusable software is still an issue [21,22,25]. Among them some of the issues are also highlighted in [23,24,26,27,28]. The proposed work fits better to the object oriented systems only.

3. A HEURISTIC-BASED METHOD FOR DEMAND-BASED REUSABLE COMPONENT DESIGN CONFIGURE

Fichman et al. [18] have suggested that heuristic approaches encourage reuse where it can be most efficiently practiced than the other approaches, as discussed earlier. The heuristics can be defined in such a way that a specific solution to a specific problem is achieved. The use of heuristics generally corresponds to the need of decision makers to efficiently generate satisfactory solutions. This motivates to propose a heuristically based approach versus other approaches discussed earlier.

3.1. FAÇADE

Facade is a packaged subset of components, or references to components, selected from the component system [19]. Each facade provides public access to only those parts of the component system that have been selected for reuse. Facade enables developers to extract the reusable components from the legacy systems.

3.2. FEATURE

A feature is a use case, part of a use case or a responsibility of a use case [19]. A feature point is a variability of features.

The proposed Heuristic-Based Method For Demand-Based Reusable Component Design Configure is as follows:

1. Step I: Consideration of use cases and actors from the use case diagram generated from the system.
2. Step II: Identify scenarios from the use case diagram.
3. Step III: The identified scenarios are implemented in the sequence diagram. The common scenarios, i.e.. Features that are the same in the domains are extracted. The functions are collected from the scenarios of the components.
4. Step IV: The reusability levels of the components can be identified by using a measure of the LCOM of the different scenario scenarios.
5. Step V: These functions are collected and placed in a configured reusable component C. grouped 'extracted from various domain-specific components in a facade.
6. Step VI: The configured reusable component consists of feature points (FP) that differ from feature to feature.
7. Step VII: The heuristic function is applied in identifying the feature points (FP) for component reuse.
8. Step VIII: Repeat Step IV for all components in the component-based systems.

The above approach can be realized using the different case studies as follows:

The point of sale case study is very popular and is used in many large supermarkets or department stores and is used to meet the needs of the sales system. This is an online system and is used to manage or control most large store activity. This case study manages and controls the inventory details and does the online accounting and generates various online reports. Figure 2, describes the use case diagram of the point of sale component. The actors are the customer and the cashier. Both actors interact with each use case. The list of use cases are Buy Product, Barcode Scanning, Paybill, Process

Sale, Complete Sale, Update Inventory, and Tax Calculation. Various scenarios were identified from this use case diagram.

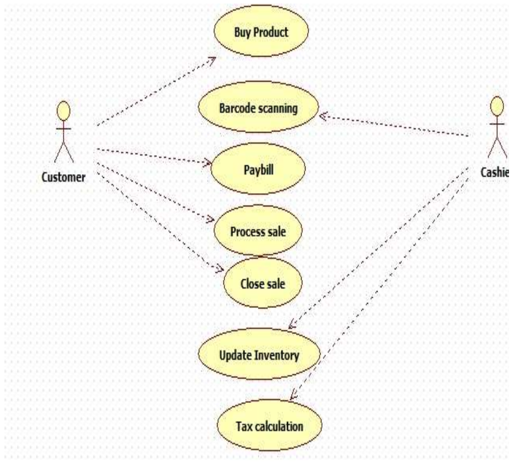

Figure 2. Use case Diagram of Point of Scale

Table 3. *Total scenarios identified in Point of sale*

Component Name	Total Number of Actors	Total Number of Usecases	Total Number of scenarios identified
Point of Sale	2	7	8

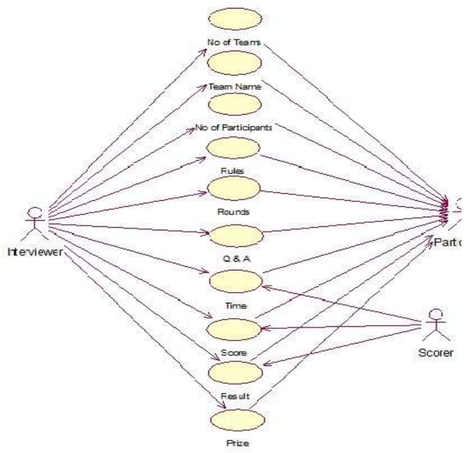

Figure 3: Usecase Diagram of Recruitment System

Figure 3 shows the use case diagram of the recruiting component. The use case diagram includes the three actors Interveiwer, Participant and Scorer. About 10 use cases interact with these three actors. Table 4 contains the list of scenarios identified from this use case diagram

Table 4. *Total Scenarios identified in Recruitment Systems*

Component Name	Total Number of Actors	Total Number of Usecases	Total Number of scenarios identified
Recruitment	3	10	22

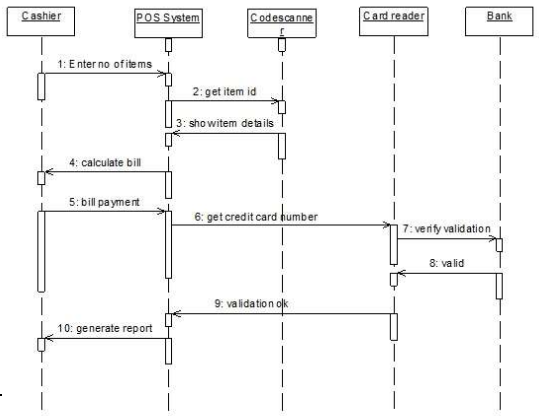

Figure 4. Sequence Diagram of Point of Sale

The sequence diagram for the point of sale component is shown in Figure 4. Cashier is the limit class that interacts with the point of sale. The scenarios identified in the use case diagram are implemented in the sequence diagram. Each scenario is illustrated one after the other. There are also some scenarios that in turn act as the component's response. Code scanners and code readers are the different control classes of this sequence diagram. Bank is the entity class. Different methods relate to the boundary, control and entity classes. These methods are the relevant behavior modeling of the system. The different scenarios are identified by the point of sale system, but the other component systems such as online quiz, point of sale, recruitment system, web interactive and education system are also taken into account, but not recognized. The point-of-sale component is only implemented. The features are considered from the scenarios obtained from the sequence diagram. The common and related features are extracted and grouped into a new component C1 'and C2'. These are known as configured reusable components. A configured reusable component contains the general and related functions. Such configured reusable components are grouped into a facade. The building owners can easily remove the reusable components as a facade. The overall presentation of the concept is shown in Figure 5. This applies to the component-based systems of the domain-specific components. The configured reusable components can also differ for different domain-specific components.

4.DEMAND BASED REUSABLE DOMAIN-SPECIFIC SOFTWARE COMPONENTS: AN EMPIRICAL EVALUATION

The reuse of software components can be assessed taking into account the heuristics discussed above. The results were evaluated taking into account the domain-specific components. The various domain components were realized and implemented according to the identified heuristics. The scenarios were identified in the use case diagram. These were implemented in the sequence diagram. The empirical evaluations of the domain-specific component reuse are discussed in detail, taking into account the lack of cohesion of the methods (LCOM). The metrics were identified to determine the level of component reuse. The measure "Lack of cohesion in methods" (LCOM) [20] focuses on the cohesion between methods of a class used in the case studies discussed. The LCOM measure can be defined as follows

$$\text{LCOM} = \begin{cases} |P| - |Q|, & \text{if } |P| > |Q| \\ 0 & \text{Otherwise} \end{cases}$$

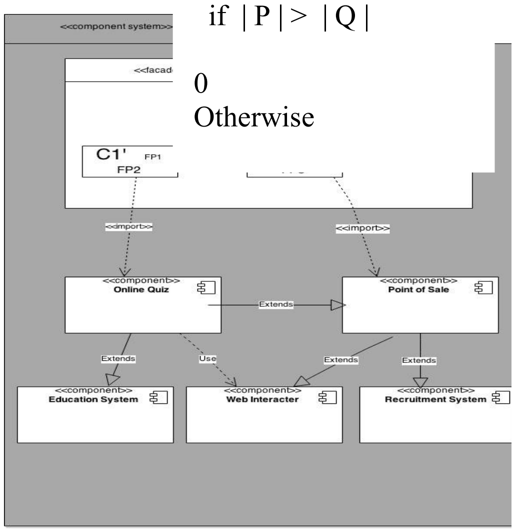

Figure 5: Component Based System representing a facade and Domain specific reusable components

$$P = \{ (Ci, Ci) \mid Ci \cap Cj = \emptyset \}$$

and

$$Q = \{ (Ci, Ci) \mid Ci \cap Cj \neq \emptyset \}$$

Where $\cap$ stands for stands for the operation of intersection of two sets of components and $\emptyset$ denotes an empty sets.

If all sets { C1, C2 , C3 , ------------Cn} are empty then let P = ∅.

C1={prepareinvoice(),buyproduct(),barcodescanning(),paybill(),processsale(),clearsale(),updateinventory(),calculate() }

C2={ score(), time(), calculate(), prize(),rules(),No.ofteams(),No.ofparticipants(), calculate(), conductexam() }

C3={createwebpage(),clearwebpage(), updatewebpage(), setpage(),stayonpage(), addcontent(), modifycontent(), clearcontent(), submitwebpage()}

C4={conductexam(),results(),internalexam(),timetable(),externalexam(),calculateresult(),calculateattendance(),generatenotice(),generatesalary(),updateattendance(),clearcontent()}

C5={checkqualifications(),submitCV(),shortlistcandidates(),announceresult(),conducttest(),conductinterview(), checkexperience(), shortlistcandidates()}

All similar method interactions from the behavioral scenarios obtained above apply mathematically. The common methods are identified using the above mathematical model. This results as follows:

$C1 \cap C3 = \{ \emptyset \}$
$C1 \cap C4 = \{ \emptyset \}$
$C1 \cap C5 = \{ \emptyset \}$
$C2 \cap C3 = \{ \emptyset \}$
$C3 \cap C5 = \{ \emptyset \}$
$C4 \cap C5 = \{ \emptyset \}$
$C1 \cap C2 =$ {calculate ()}
$C2 \cap C4 =$ {conductexam()}
$C2 \cap C5 =$ {announceresult()}
$C3 \cap C4 =$ {clearcontent()}

Next, the value of both P and Q from the above replacement of various methods can be analyzed from these empirical values. The value is assigned under the P and others are assigned under the Q.

P={(C1,C3),(C1,C4),(C1,C5),(C2,C3),(C3,C5), (C4,C5)}

Q= {(C1,C2),(C2,C4),(C2,C5),(C3,C4)}

and finally

LCOM = | P | - | Q | = 6 – 4 =2

The LCOM measure shows the desperation of the methods in the components. The higher the value of LCOM, the lower the similarity between the methods in the components, i.e. The LCOM measurement therefore makes it possible to know the degree of reusability of methods in the components. The LCOM measured value for the assigned components is reached at the value '2'. This means that the value of LCOM for these component systems is very low. The reusability of the components is very high. It is analyzed that the LCOM value of different components in different component-based systems can vary. Taking into account the above Q, related methods are grouped in a facade in the component systems. This facade makes it possible to identify reusable components in the component systems. The builders can easily extract the reusable components from this facade. Furthermore, the components in the facade are again separated according to the changed features with the feature points. A feature point (FP) is the variability of the features. The points of variation are reached depending on the differences in the characteristics of the other methods in the facade. The heuristic function is taken into account for the identification of the feature points as follows:

$H_{reuse}=$ LCOM { Ci } > | 2 |

The above equation describes the heuristic function. This feature helps identify variations in the configured reusable components. This heuristic function takes into account a value of approximately 2. The value for lack of cohesion in methods (LCOM) can be equal to or greater than 2, since at least 2 characteristics must be taken into account, which are almost identical in their behavioral implementation. These feature points are used to identify those features in the components that have different behavioral functions. The value of the heuristic function is determined by the behavior of the component system. This can vary from different component systems and the number of components used in the system. By considering the above heuristic function, the feature point can be identified. This helps in deciding the feature point in the configured reusable components. According to the requirements, the developers can easily recognize the common behavior of the components using the feature points (FP). In Figure 5, FP1 and FP2 are the feature points identified in the C1 'configured reusable component in the facade. FP3 is the feature point identified in the C2 'configured reusable component in the facade. FP1, FP2 and FP3 are identified using the heuristic function. The feature points were identified as a function of the features with their behavioral aspects of the components. It was found that FP1, FP2 and FP3 show at least 2 behavioral characteristics of the selected components. This is achieved through the lack of method cohesion (LCOM). Thus, feature points help identify the variations of configured reusable components in the features.

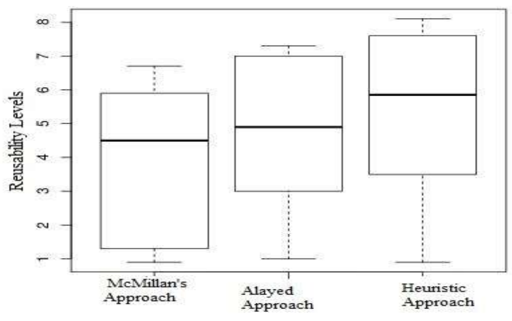

Figure 6. Comparison of heuristic based approach with both existing approaches

Figure 6 describes the comparison of the previous approaches and the proposed work. The previous approaches are named as McMillan et.al.[17] and Alayed et.al.[15] are considered for comparison with the proposed heuristics approach. The re usability level of the different approaches are shown using the box plotting. McMillan et.al.[17], Alayed et.al.[15] and heuristic based approach re usability levels are showed and compared effectively. It is identified that Alayedet.al[15] approach is better than the McMillan et.al.[17]. In comparison with both earlier approaches, heuristic based approach is better than these both approaches. The median value in the box plot shows the re usability level of the different approaches discussed in this paper. The median of the McMillan's approach [17] is 4.4 and Alayed [15] approach is 4.9, where as the proposed heuristics approach is around 5.8. The reusability index value may change for different component-based systems. As the behavioral aspects of the system will play a vital role in the software component reuse. This is achieved with the reusable configured components which are grouped in a facade. The heuristic function will also help in identification of feature points (FP) for extracting common features from the reusable configured components. A configured reusable component includes only those features that can be reusable a lot. This is achieved from the Lack of cohesion in methods (LCOM) measure. It is assumed that heuristic based approach gives the possible solution for any contemporary problems in computer science. The proposed heuristic based approach is well suited and can be applicable in the software industry may produce better results in the reuse of a software components.

5. CONCLUSION AND FUTURE WORK

Organizations benefit greatly from component-based systems built with demand based reusable domain specific software components. The most difficult aspect of reuse is selecting the appropriate reusable from a vast array of options and adapting reusability to current requirements. Different techniques and design flaws in reusable domain-specific components have been examined in earlier work. Until date, it appears that there has been no widely acknowledged standard for the design of demand based reusable domain specific software components. This encourages utilizing a heuristic-based approach to develop configurable reusable components. As mentioned in the preceding sections, the suggested strategy is realized and executed utilizing several domain specific components and implemented using various domain specific applications. The degree of reusability of domain-specific components is evaluated and compared to several ways, with the suggested heuristic-based approach being found to be superior to the others and extracted the demand-based domain specific components. The suggested heuristic method for Demand-Based Reusable Component Design Configure and empirical analysis was conducted which is well-suited, may be used in the software business can improve outcomes when extracting demand based reusable domain-specific software components. The reuse design guidelines for component quality characteristics can be applied in the future as part of the ongoing effort.

REFERENCES:

[1] McIlroy, M. Douglas, et al. "Mass-produced software components." Proceedings of the 1st International Conference on Software Engineering, GarmischPattenkirchen, Germany. sn, 1968.
[2] Mili, Hafedh, FatmaMili, and Ali Mili. "Reusing software: Issues and research directions." Software Engineering, IEEE Transactions on 21.6 (1995): 528-562.
[3] Frakes, William B., and Kyo Kang. "Software reuse research: Status and future " IEEE transactions on Software Engineering 31.7 (2005): 529-536.
[4] Almeida, Eduardo S., et al. "CRUISE: Component Reuse in Software Engineering." (2007).
[5] Zhongjie Wang et.al, " A Survey of Business Component Methods and Related Technique", World Academy of Science, Engineering and Technology, pp.191-200, 2006.
[6] N Md JubairBasha, Salman Abdul Moiz, "A Methodology to manage victim components using CBO measure", published in International Journal of Software Engineering & Applications(IJSEA), Vol3 (2) pp 87-96, March 2012, ISSN:0975-9018.
[7] William B. Frakes, Kyo Kang: Software Reuse and Research: Status and Future, IEEE Transactions on Software Engineering", Vol. 31, No. 7, July 2005.
[8] Xichen Wang, Luzi Wang: Software Reuse and Distributed Object Technology, IEEE Transactions on Software Engineering, 2011.
[9] Sametinger: Software Engineering with Reusable Components, Springer-Verlag, ISBN 3- 540-62695-6, 1997.
[10] Jianli He, Rong Chen, WeinanGu: A New Method for Component Reuse, IEEE Transactions on Software Engineering, 2009.
[11] Maurizio Pighin: A New Methodology for Component Reuse and Maintenance, IEEE Transactions on Software Engineering, 2001.
[12] Alkazemi, Basem Y., Mohammed K. Nour, and Pete A. Lee. "Characterizing Reusable Software Components" published in DReMeR'13, June, 2013, Pisa, Italy.
[13] Kim, Won. "On Issues with Component-Based Software Reuse." Journal of object technology 4.7 (2005): 45-50.
[14] Kästner, Christian, and Sven Apel. "Feature-Oriented Software Development." Generative and Transformational Techniques in Software Engineering IV. Springer Berlin Heidelberg, 2013. 346-382.
[15] Alayed, Asmaa, Kung-Kiu Lau, and Cuong Tran. "Towards Component-based Domain Engineering." Software Engineering and

Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on. IEEE, 2013.

[16] Bauer, Veronika, Jonas Eckhardt, Benedikt Hauptmann, and Manuel Klimek. "An exploratory study on reuse at google." In Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices, pp. 14-23. ACM, 2014.

[17] McMillan, Collin, et al. "Portfolio: finding relevant functions and their usage." Software Engineering (ICSE), 2011 33rd International Conference on. IEEE, 2011.

[18] Fichman, Robert G., and Chris F. Kemerer. "Incentive compatibility and systematic software reuse." Journal of Systems and Software 57.1 (2001): 45-60.

[19] Jacobson, Ivar, Martin Griss, and PatrikJonsson. Software reuse: architecture process and organization for business success. Vol. 285. New York: acm Press, 1997.

[20] Peters, James F., and WitoldPedrycz. Software engineering: an engineering approach. John Wiley & Sons, Inc., 1998.

[21] Aggarwal, Jyoti, and Manoj Kumar. "Software metrics for reusability of component based software system: a review." International Arab Journal of Information Technology 18.3 (2021): 319-325.

[22] Zulkurnain, F. I. N. M., et al. "A decade of software reuse: Research direction." International Journal of Advanced Science and Technology (2020).

[23] McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., & Fu, C. (2011, May). Portfolio: finding relevant functions and their usage. In Proceedings of the 33rd International Conference on Software Engineering (pp. 111-120).

[24] Basha, N. M. J., & Moiz, S. A. (2012, September). A methodology to reconfigure victim components using modularity. In 2012 CSI Sixth International Conference on Software Engineering (CONSEG) (pp. 1-6). IEEE.

[25] Basha, N. M. J., & Choudhury, S. (2016). Assessment of Reusability Levels on Domain-Specific Components Using Heuristic Function. In Innovations in Computer Science and Engineering: Proceedings of the Third ICICSE, 2015 (pp. 153-161). Springer Singapore.

[26] Basha, N., Ganapathy, G., & Moulana, M. (2022). A Preliminary Exploration on Component Based Software Engineering. International journal of computer science and network security: IJCSNS, 22(9), 143-148.

[27] Basha, N. M. J., Ganapathy, G., & Moulana, M. (2021, December). CREA-Components Reusability Evaluation and Assessment: An Algorithmic Perspective. In International Conference on Advanced Informatics for Computing Research (pp. 132-142). Cham: Springer International Publishing.

[28] N Md Jubair Basha, et al. (2023). Towards Constructive Cost Analysis for demand based Reusable Domain Specific Components. International Journal on Recent and Innovation Trends in Computing and Communication, 11(9), 494–502. https://doi.org/10.17762/ijritcc.v11i9.8835