International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

Towards Constructive Cost Analysis for demand based Reusable Domain Specific Components

[1]N Md Jubair Basha, [2]Dr. Gopinath Ganapathy, [3]Dr. Mohammed Moulana
[1]Research Scholar, Department of Computer Science & Engineeirng,
Bharathidasan University, Tiruchirapalli, TN, INDIA.
e-mail: jubairbasha@gmail.com
[2]Senior Professor, Department of Computer Science & Engineeirng,
Bharathidasan University, Tiruchirapalli, TN, INDIA.
e-mail: gganapathy@gmail.com
[3]Professor, Department of Computer Science & Engineeirng,
Koneru Lakhsmiah Educational Foundation, Vaddeswaram, AP, INDIA.
e-mail: moulanaphd@gmail.com

Abstract*— The prevailing software development methodology embraced by a majority of organisations is characterised by its agility. The neglect of normal analysis and design procedures is a consequence of the significant pressures associated with designing a product within specified time and budget constraints. This phenomenon could potentially result in a death of software of superior quality, while simultaneously impeding the constructive reuse of components. In the majority of component approaches, the demand of domain specific sofwatre components occurs during the later stages. In this paper, various components can be identified as demand based reusbale domain specific software components, which might also help in reusing these components in the subsequent increments. The strategy for extraction of components & procedure for reusing the existing components is described and a sample case to realize the same is presented.Still there is a dire need to early identify the demand based domain specific sofwatre components and perform the constructive cost analysis for the reusable domain specific software components. The issues related to the estimation of cost reuse measures are still challenging. This paper presents the constructuve cost analysis for the demand based reusable domain specific sofwtare components and proposes reuse measures for the family of applications with the quantized values. By analyzing these cost measures, the budget and effort in the development can be reduced. The results are estimated from the HR Portal domain specific softwrae application as a case study and its respective scenario has been explored in a better manner.*

Keywords-Estimation, Component Extraction, Reusability, demand based domain specific scomponent, cost concern matrix, victim components

I. INTRODUCTION

Software reuse refers to the activity of implementing or modifying software systems by utilizing pre-existing software assets [2]. The concept of software reuse has garnered significant attention among the software community due to its perceived advantages, such as enhanced product quality and reduced product cost and schedule. The objective is to establish and sustain a collection of reusable components that serve as a foundation for future products within a specific functional domain.

The utilization of reusable components is progressively supplanting the utilization of monolithic and proprietary technology [1]. The rationale behind this transition is driven by the imperative to minimize life cycle expenses, improve software excellence, and optimize the resources required for system development and testing.

An effective software reuse process enables enhanced productivity, quality, and reliability, while simultaneously reducing costs and implementation time. The initiation of a software reuse process necessitates an initial investment, which, however, proves to be cost-effective within a limited number of reuses. In summary, the establishment of a reuse process and repository generates a knowledge foundation that progressively enhances in quality with each instance of reuse. This, in turn, diminishes the extent of development efforts necessary for forthcoming projects and ultimately mitigates the risk associated with new projects that rely on repository knowledge.

There are several significant benefits associated with the utilization of domain-specific components.

The utilization of component reuse results in cost and schedule reductions, as it eliminates the requirement for developing the component from scratch. If deemed necessary, the component has the potential to undergo modifications.

The term "reduced" refers to a state or condition in which something is diminished or the allocation of resources to testing activities accounts for a significant portion, specifically more than 60%, of the overall effort expended in software development. The utilization of domain-specific components leads to a reduction in testing effort.

The certification process for the developed component has already been finished. The component is expected to exhibit high quality.

Numerous organizations have devised domain-specific components that serve as valuable resources, enabling their future reuse. Although the component may not be utilized again as a mirror component, it has the potential to undergo modifications. The level of effort needed to modify a

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

component is lower in comparison to that necessary for developing it from the beginning. Nevertheless, it is imperative to establish a methodology for the identification and cultivation of domain-specific components.

The subsequent section of this paper is structured in the following manner. Section 2 delineates the pertinent literature in the field, whereas section 3 elucidates the incorporation of software reuse. Section 4 provides an overview of the domain engineering process. Section 5 contains a detailed analysis of the HR Portal application. Section 6 focuses on the assessment of cost metrics associated with the reuse of resources. Section 7 provides a detailed analysis of a hypothetical scenario that aims to demonstrate the financial implications associated with the acquisition and implementation of domain-specific components. The paper is concluded in Section 8.

II. RELATED WORK

In the context of reuse-driven development, the reusability of software assets becomes feasible when organizations possess a substantial number of applications and the development team possesses a comprehensive understanding of the value inherent in rendering these artifacts reusable. Moreover, the existing patterns that commonly address a shared problem fail to consider the extraction of reusable components from the requirements statements. The presentation focused on platform-specific patterns, such as Java design patterns and J2EE patterns. The subsequent methodologies are now employed in the field of re-engineering to facilitate the design and development of component-based systems.

A. CORUM II

The CORUM II framework organizes the requirements from several perspectives in order to facilitate the integration of architecture-based reengineering tools with code-based reengineering tools. Nevertheless, this approach fails to provide the necessary workflow for the implementation of a reengineering project.

B. MORALE

Mission Oriented Architecture (MOA) is a conceptual framework that emphasizes the alignment of an organization's architecture with its mission objectives. MOA is a strategic approach that aims the concept of Legacy Evolution pertains to the challenge of developing and adapting intricate software systems. The objectives of this system are as follows: purpose-driven: The process of enhancing the legacy system should be guided by the purpose to be achieved, rather than solely relying on technical criteria. The adjustments to software that have the greatest impact in terms of time and cost are those that significantly modify the design, structure, and behavior of the system.

C. L2CBD

The Legacy to Component Based Development (L2CBD) methodology offers the capability to convert existing legacy systems into modern component-based systems, resulting in enhanced software architecture. The characteristics that are supported by L2CBD are as follows:

This paper proposes an architectural methodology for developing novel application structures.

This approach involves utilizing reverse engineering methods to derive architectural information from both the source code and domain knowledge.

• The proposed methodology for generating component systems allows for the reuse of architectural components in subsequent iterations.

D. CBD96

The CBD96 methodology employs a business component identification approach that organizes objects that are closely connected into groups. The approach employed fails to consider the incorporation of reusable system components and is afflicted by dependence concerns.

E. Cheesman and Daniels (2001)

The approach being referred to is an expanded iteration of the CBD96 method. The proposed approach encompasses a methodology for discerning constituent elements through the utilization of use cases and business type models. In this methodology, the authors employ inter-class relationships as the primary criterion for finding components. The central element of each clustering is represented by the core class, and the process is guided by the responsibility obtained from use cases.

F. S. D. Kim & S. H. Chang (Kim, 2004)

The methodology titled "A Systematic Method to Identify Software Components" places emphasis on the principles of strong cohesion and low coupling during the process of discovering reusable software components. This methodology employs clustering algorithms, measurements, decision rules, and a collection of heuristics. This approach presupposes the presence of an object-oriented model for a certain domain, encompassing a use case model, object model, and dynamic model. By leveraging these artifacts, the method seamlessly converts them into components. This strategy primarily emphasizes use case dependency rather than focusing on the structural links between classes and their message call information.

The concept of reusability metrics encompasses a methodology for quantitatively evaluating the effectiveness of reusable components. Numerous metrics pertaining to reusability have been proposed in academic literature, with a notable focus on qualitative rather than quantitative measures. The metrics for measuring reusability, as discussed in reference [19], are founded on four key attributes: self-descriptiveness, modularity, portability, and platform independence. However, the weights assigned to them are dependent on subjective assumptions, which are qualitative in nature. A collection of metrics pertaining to reusability is proposed in reference [20]. While this strategy is more efficient compared to non-automatable techniques, the objective is solely to reuse the interfaces of the components. The current technique does not incorporate measures for reusability throughout the design phase. In their study, Wang et al. [21] put out the proposition that it is necessary to redesign components that are not suited for reuse due to their shortcomings. Nevertheless, a

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

comprehensive strategy for the entire system is not provided. The proposed study by [22] focuses solely on measuring the impact on victim components, while neglecting to present any measures for the broader family of applications. The concept of reusability encompasses not only the reuse of code, but also the reuse of various parts throughout the software development process [23].There is an urgent requirement to evaluate a comprehensive cost analysis approach for domain-specific reusable software components.

III. SOFTWARE REUSE

Software reuse refers to the technique of utilizing existing software components or leveraging software expertise to develop novel software solutions. Reusable assets encompass two main categories: reusable software and software knowledge. The concept of reusability pertains to the likelihood of a software asset being reused [3]. Software reuse refers to the practice of utilizing pre-existing software components, known as "designed software for reuse," several times throughout the development process [4]. The practice of software reuse offers several benefits to organizations, including the effective management of software development complexity, enhanced product quality, and improved production efficiency. In contemporary times, there has been a surge in the use of design reuse practices, particularly in relation to object-oriented class libraries, application frameworks, design patterns, and accompanying source code [5]. Jianli et al. introduced a pair of complimentary approaches aimed at the reuse of pre-existing components. One of the features enables the evolution of components themselves, which is accomplished by binary class level inheritance across modules of components. One approach involves organizing the entities based on their semantic definitions, allowing for their compilation-time assembly or runtime binding. Component containment remains the primary technique for achieving software product line development [6]. In order to facilitate the retrieval of components, a substantial amount of information must be gathered, preserved, and analyzed. Maurizio has developed an approach for the automated construction of a software catalogue, which includes tools for preserving and retrieving information [7]. Software reuse can be categorized into two main divisions, namely product reuse and process reuse. Product reuse encompasses the practice of reusing a software component, whereby a new component is generated through the integration and assembly of modules. The concept of process reuse refers to the practice of reusing old components obtained from a repository. These components have the potential to be reused either in their current form or with slight modifications. The archival of the updated software component can be achieved by the process of versioning these components. The classification and selection of these components can then be based on the specific domain requirements [8].

The structure of a component plays a crucial role in determining its functionality and usability. The occurrence of reuse is not incidental. In order to ensure the feasibility of reuse, it is imperative to undertake specification, building, and testing processes. The development of new software is rendered more costly, perhaps by a factor of up to ten, as a result of this factor.

Numerous distinct criteria have been proposed for evaluating the quality of a component. The aforementioned conditions can be succinctly summarized as follows:

The component ought to embody an abstraction. The software system should exhibit a high level of cohesion and provide only the necessary operations required to ensure its use in an effective manner. The software should provide a clearly delineated interface, encompassing both syntactic and semantic aspects. In the event that two operations within distinct components possess identical names, it is expected that they exhibit comparable behavior. However, it is crucial that their writing style bears resemblance to academic discourse in order to enhance comprehension.

The component should possess independence from its surrounding entities, exhibiting loose connections and thereby maintaining low coupling with other components. The adoption of an object-oriented mindset promotes individual autonomy. The component should possess a general abstraction that may be effectively applied across multiple applications, hence minimizing the need for additional modifications.

The concept of understandability encompasses both internal and exterior dimensions. Due to their extended lifespan, high-quality components are likely to undergo prolonged maintenance. The component system encompasses the processes of selecting, classifying, and managing the components contained inside the repository, as well as the creation of novel components. It is recommended that the component repository be distributed across the development organization to ensure accessibility of the components. It is preferable for the component repository to be shared throughout multiple distinct products. This implies that the component system should be capable of supporting multiple projects simultaneously. In the event that new projects are to be undertaken, it is imperative to acquire the necessary components that are vital to the development process. The project proposals ought to undergo evaluation by a committee comprising seasoned designers as well as a representative from the component department, thereby establishing a software component committee. The evaluation of whether the proposed components should be developed or not should be conducted. Once the decision to proceed with the building of the component has been made, it is then forwarded to the component construction phase, with a specified date. Once prepared, the component is incorporated into the repository, resulting in an updated version state as depicted in Figure 1. The analysis of the value of the software component group should be conducted as the component is being utilized. Which component is utilized most frequently? Which items are completely unused? What is the extent of the benefits derived from the components? This analysis facilitates the advancement of the component system.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

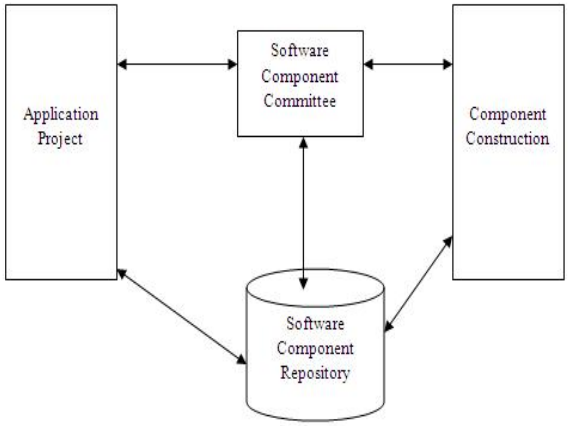

Figure 1: Organization for Component Management

Component-Based Development (CBD) encompasses similar characteristics as the Spiral Model. The software components, referred to as Classes, that are collected as applications can be characterized as a paradigm of Component Based Development (CBD) [9].

The construction of the component model begins by placing emphasis on the potential components. This objective can be accomplished by applying appropriate algorithms and programmes to manipulate the data. The components required for the software projects are stored within the repository. After identifying the candidate components, the repository is examined to determine if the necessary components are there. If appropriate components are identified, they are dug and then repurposed. If the component is not discovered in the repository, it may be created first using the object-oriented methodology. The initial repeat of the application is to build a level-headed of components mined from the repository and new components to locate the creative demand of the unique application. The Process Flow is integrated into the spiral model and serves to extend the component assembly curves as the component life cycle progresses in subsequent iterations.
Software reusability can be achieved through the utilization of the Component Based Development Model, which has proven to be highly advantageous for software engineers.

The study conducted by Yourdon.E. [10] presents findings on the successful implementation of software reusability by QSM Associates Inc. The research highlights the advancements in component assembly, resulting in a significant reduction in the development life cycle. Notably, the study reports an impressive 84% decrease in project cost and a productivity index of 26.2, surpassing the industry median of 16.9. The aforementioned findings indicate that the incorporation of roughness in the component repository and CBD Model yields numerous advantages for software engineers.

In their study, Singh et al. [11] examined the many implications of reusability in the context of a component-based approach, as well as the metrics and models associated with software reuse. This paper is a study that examines the empirical validation of the metrics given for component-based systems.

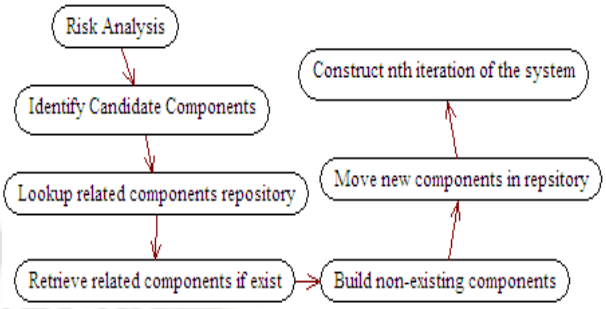

Figure 2: Component Based Development Model [9]

Component interface metrics possess the potential to enhance the reusability aspect of components. The eminence of these entities arises from the fact that alternative sources of information pertaining to reusability are often utilized in the form of third-party components, which tend to be opaque in nature. Furthermore, the utilization of automation by these entities facilitates a more impartial, meticulous, and proficient analysis of component reusability. The set of interface metrics introduced in this study has demonstrated that measuring component interfaces can provide more accurate and relevant information for analyzing component reusability. Metrics have the capacity to provide a significant amount of valuable information via interfaces, surpassing the effectiveness of non-automatable methodologies. These metrics provide a deeper comprehension of the assets associated with the interfaces of components. The lesson pertaining to metrics involved doing a reusability analysis on the tested components, which relied on expert knowledge of these components. The present analysis pertains to the utilization of reusability analysis in relation to components, the understanding of which remains elusive to metrics practitioners. The user's text does not contain any information to rewrite [12]. AlOmara, Eman Abdullah, et al. [25] presented insights regarding how developers discuss software reuse by analyzing Stack Overflow. These findings can be used to guide future research and to assess the relevancy of software reuse nowadays.

IV. DOMAIN ENGINEERING

Domain Engineering (DE) is an essential process wherein reusable components are created and effectively managed to ensure that the architectural design adequately meets the specific needs of the designated domain [13]. The term "domain" pertains to the functional regions encompassed by a collection of application systems that share comparable software needs [14].

The process of Domain Engineering [15] is depicted in Figure 3. Domain engineering (DE) encompasses several essential aspects, namely domain analysis, domain design, and domain implementation. The DARE-COTS tool, which is referenced as [16], is utilized for the purpose of Domain Analysis. To achieve the generic variable qualities of a group of systems, it is necessary to have a relevant domain in the

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

pre-phase. A domain analysis model can be constructed by concealing the properties. Based on this framework, the software architecture specific to the domain can be devised, followed by the creation and management of reusable components.

When embarking on the development of a novel system in an unexplored domain, it becomes imperative to accurately capture the system's needs and specifications in accordance with the domain model. Subsequently, the design of the new system should be refined in alignment with the principles of Domain Specific Software Architecture (DSSA). Finally, the appropriate components should be selected and organized to effectively govern and administer the newly developed system. The term "Application Engineering" refers to the process of designing and creating a distinctive system for applications.

The process of domain engineering, as described in reference [15], provides a comprehensive overview of the Decision Support System for Product Quality Tracking System. This analysis elucidates the process of creating a product quality tracking system that is both open and reusable, based on the principles of domain engineering. The research reported in this article highlights the importance of reusing the primary functionality of a system when developing an application in the same domain or when the necessary components are readily available. There is still a significant amount of work that has to be undertaken in order to develop a comprehensive product quality tracking system utilizing assertive techniques and establishing a robust repository. In their study, Massimo et al. [17] conducted an evaluation of the use of domain analysis in the field of production management. They measured the outcomes and identified areas for improvement by grouping the domain analysis approach inside the approved development process.

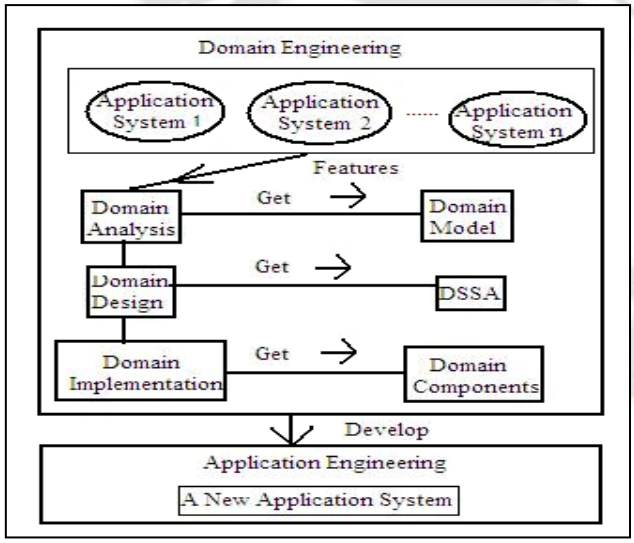

Figure 3: Process of Domain Engineering [15]

A. DOMAIN SPECIFIC COMPONENT FRAMEWORKS

Given the significant progress in software system development across several domains, there arises an imperative requirement for the creation and advancement of Domain Specific Component Frameworks (DCSF). Many software development techniques incorporate agile principles in their development methodologies. The evolution of Domain Specific Component Frameworks has been seen through the identification of patterns. In their study, Frederic et al. (2018) introduced the concept of Domain Components and conducted an analysis of patterns to create a comprehensive framework. This framework offers a unified way to implementing the semantics of Domain Components by examining the Domain Specific services. The research offered by the authors [18] examines many case studies that span across multiple areas. The architectural patterns suggested in this study will be integrated with the utilized generative programming techniques, which encompass the challenges associated with implementing domain-specific considerations. The article discusses a research problem that pertains to the creation of containers. Specifically, it focuses on the need for a symmetric approach that involves the establishment of policies to effectively manage a wide range of domain-specific services.

V. ANALYSIS OF HR PORTAL APPLICATION

The HR Portal Application is a software system designed to facilitate human resources management within an organization.

The system has been developed to facilitate client interaction with both the web tier and business tier, as well as establish a connection to the Data Access Object (DAO) component. The web-tier component is comprised of Java Server Pages (JSPs) and Servlets.The Business tier encompasses the Enterprise JavaBeans (EJBs).The DAO's composition comprises. The classes interact with their respective objects in order to establish communication with the database. The web-tier components consist of the HttpServlet, HRProcessServlet, Login Servlet, InterviewResultServlet, and RegistrationServlet classes.The three stateless bean classes in the Business-tier components are EmployeeBean, InterviewResultsBean, and HRProcessBean.The components of the DAO (Data Access Object) include the BaseDAO, EmployeeDAO, InterviewDAO, HRDAO, and ProcessDAO classes.

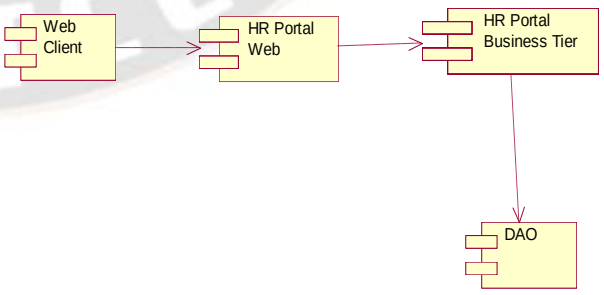

Figure 4: Components of HR Portal Domain Application System

Many of the systems that prioritize reuse typically include the establishment and maintenance of a repository containing reusable components. However, it is necessary to

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

develop a methodology for identifying components that are reused or have potentially been utilized extensively. These components are commonly referred to as Non-Victim components.

If the designer wishes to determine whether element of the system is not being effectively reused at a given moment, they must conduct a lookup on the component management relation. A centralized repository is responsible for maintaining a table that facilitates the management of component reuse. The table is comprised of two distinct fields. The nomenclature of a particular component is indicated by its name, while the count denotes the frequency with which the component has been utilized across multiple systems.

Table 1 presents a comprehensive inventory of the components of the HR portal system that were utilized in several applications. Components that are not utilized regularly are referred to as victim components. Given that the Businesstier component has been utilized a mere 10 times, it can be considered a potential candidate for the victim component. In order to enhance the potential for future reuse, it is necessary to re-organize the victim component by separating it into many segments.

Table 1. Component Management Relation

Component	Count of Reuse
DAO	36
Web tier	10
Business tier	24

The act of achieving the count of reuse is accomplished by putting the HR Portal application onto the Net Beans Integrated Development Environment (IDE). The Netbeans Profile feature enables the tracking of the frequency at which a component is invoked. The application provides information on the number of times the components are triggered. Based on the data shown in Figure 5, it is possible to determine the measures of reuse cost.

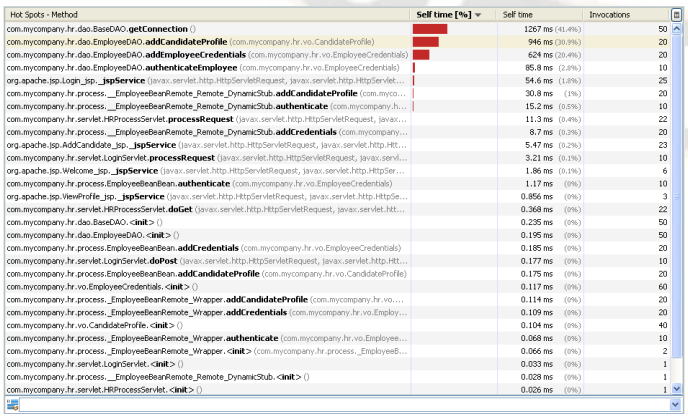

Figure 5.Invocations occurred for different components of HR Portal Application

VI. ESTIMATION OF REUSE COST MEASURES

Each identified concern necessitates an assessment of the associated cost and a specific plan for its implementation.

The concern cost matrix undergoes periodic updates to incorporate new components or modifications to current components. Additional concerns are incorporated into the repository through the process of entering data into the concern cost matrix (CCM), which adheres to the prescribed structure outlined below.

Concern Cost Matrix

	C1	C2	…	Cm
Cost				

```
For (i=1 to n do)
   Begin
        For (j=1 to m do)
         Begin
              If (RMF[i][j] = 0) then
                   Cnt=Cnt+1;
         End;
      CSV[i]= cnt*CCM[i];
   End;
```

The Vector CSV[i] indicates the cost saved in implementation of concern Ci. The idea is if the concern is already implemented (i.e RMF[i][j]=1), The cost associated with the implementation of the aforementioned concern, denoted as CCM[i][j], is conserved due to the reuse of the identical component. If a component is modified (say comp1.0), its alteration may produce another component (say comp 1.1), this will also be recorded in repository and the cost of maintenance is retained in CCM. In a similar vein, the calculation of schedule utilization can also be undertaken.

VII. A SCENARIO

Let us examine a specific scenario pertaining to a Distribution Processing System. Within this operational framework, the customer initiates an order placement, subsequently followed by the storekeeper's assessment of the condition of the items. Ultimately, the accountant assumes the responsibility of generating an invoice.

The use case diagram is constructed to represent any interdependencies that exist between use cases.

The use case diagram pertaining to a certain iteration of the system is presented below.

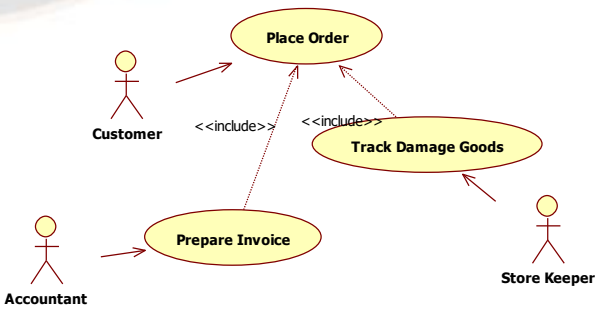

Figure 6. Use case diagram for Distribution Processing System

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

The Trace Damage Goods use case and Prepare Invoice use case can't be realized till the Place Order use case is realizes successfully as shown in the Figure 6.

As per the proposed component identification strategy, since place order behavior becomes the mandatory pre-condition of other use cases, it is a candidate for a component. The Place order use case is realized and then stored in the repository with the initial version number. The effort & schedule required to implement the place order use cases was evaluated and stored in the repository. This parameter helps in knowing the trade off benefits of the reusability.

In the above scenario it was assumed that the component repository doesn't contain any of the above identified behavior components.

Considering another inventory system, that needs the similar functionality to be implemented as described by Place order use case. The proposed strategy mines the components for realization of the identified behavior.

If the place order component provides the similar functionality it can be directly re-used. If some modification to the place order use case is needed, it is modified. The effort & schedule is evaluated and the same is stored along with new version number in the repository.

COCOMO model was used to estimate the effort and schedule for the realization of the use cases initially. However for the modified component, maintenance effort was evaluated using two parameters i.e. requirements added and requirements modified.

In order to estimate the reuse cost measure, it is necessary to know about the number of components available in the related application. The HR Portal application consists of four components as described in Section IV. Initially, the cost of a developing typical system without reuse is considered. It can be represented as follows.

$C_{no\text{-}reuse}$=Cost of developing typical system without reuse

Whenever the reuse is applied to some portion of the system it can be designated as R, the software from a set of component systems.

The Reuse level 'R' can be estimated by considering the number of reused components to the total number of components in the system.

$$\text{Reuse level, } R = \frac{\text{Number of reused components}}{\text{Total Number of components in the system}}$$

$$= 2/4 = 0.5$$

$$= 50\%$$

The Reuse level 'R' usually costs less than developing the whole system from the scratch.

After analyzing the percentage level of reuse components in the system the relative cost to reuse a component has to be defined,

F_{use}= Relative cost to reuse a component

Let us assume that the relative cost to reuse a component is 0.2 as default.

With R=50% and F_{use}=0.2, the cost to develop with reuse is 60% of the cost of developing an application without reuse.

The cost to develop an application system with reuse has two parts. One is the (1-R) part, developed without reuse at the normal cost. The other is the R part, developed with reuse, at a lower cost. In order to do this, it is necessary to estimate the costs separately and add.

$C_{part\text{-}with\text{-}reuse}=C_{no\text{-}reuse} * (R * F_{use})$

$C_{part\text{-}with\text{-}no\text{-}reuse}=C_{no\text{-}reuse} * (1-R)$

$C_{with\text{-}reuse} = C_{part\text{-}with\text{-}reuse} + C_{part\text{-}with\text{-}no\text{-}reuse}$

$C_{with\text{-}reuse} = C_{no\text{-}reuse} * (R * F_{use} (1-R))$

The cost saved due to reuse

$C_{saved} = C_{no\text{-}reuse} * C_{with\text{-}reuse}$

$= C_{no\text{-}reuse} * (1- (R * F_{use} (1-R)))$

$= C_{no\text{-}reuse} * R *(1- F_{use})$

The relative development cost-benefit (ROI) is due to reuse of components is then estimated to be

$$ROI_{saved} = \frac{C_{saved}}{C_{no\text{-}reuse}}$$

$$=R * (1- F_{use})$$

The relative development cost-benefit(ROI) is 40%
When R=50% and F_{use} =0.2 , ROI_{saved} is 40%

It is intended to know how much cost is necessary to know about for creating a new reusable component and manage it. So, this can be denoted as $F_{create.}$

F_{create} = Relative cost to create and manage a reusable component system.

Here all the developed component systems are used to reuse part, R percent, of any application system.

Then the cost to develop the component system for R percent is designated as follows.
$C_{component\text{-}systems}=R * F_{create} * C_{no\text{-}reuse}$

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

Since F_{create} is much greater than F_{use}, it is necessary to reuse each component and component system several times in several application systems, to make this worthwhile from a cost perspective. It has proved from the literature that the different ranges for F_{create} and F_{use} values depends upon the specific languages, complexity of the problem area and the relevant process followed.

Polin J.S. [24] had suggested the default values of F_{create} and F_{use} are 1.5 and 0.2.

If there are 'n' application systems in the family, then the cost saving for the application system family is:

$$C_{family\text{-}saved} = n * C_{saved} - C_{component\text{-}systems}$$

$$= C_{no\text{-}reuse} * (n * R*(1- F_{use})- R * F_{create})$$

Finally the return on investment in creating the set of components can be considered as follows.

$$ROI = \frac{C_{family\text{-}saved}}{C_{component\text{-}systems}}$$

$$ROI = \frac{(n * R*(1- F_{use})- R * F_{create})}{R * F_{create}}$$

$$ROI = \frac{(n * (1- F_{use})- R * F_{create})}{F_{create}}$$

When F_{use}= 0.2 and F_{create}= 1.5 then

$$ROI= \frac{(n * 0.8 – 1.5)}{1.5}$$

With breakeven point of minimum value i.e. n > 2**.**

Hence, with the above analysis, the productivity in the organization can be easily improved by increasing the number of the application components which are much reusable.

VIII. CONCLUSION

The reusability of components contributes to the development of high-quality products, since it ensures that components stored in the repository have undergone successful testing. The majority of reusability measures discussed in academic literature are either qualitative in nature or focus solely on interface reusability measurements. This work endeavours to present a proposal for the utilization of reusable domain specific software components with its quantitative measures. Whenever a new application has to be developed its Concern Cost Matrix is maintained not only to identify the existing components but also to identify the effort saved. The constructive cost for the reusable demand based components and non reused components has been quantified. The measures for the family of applications are also estimated. With these constructive cost measure analysis, the budget and effort in the development will get reduced. In future, strategies to measure the generic domain specific software components may be quantified.

REFERENCES

[1] H.K.Kim, Y.K.Chung, "Transforming a Legacy System into Components", *Springer-Verlag Berlin Heidelberg*, 2006.

[2] S. Mahmood, R.Lai and Y.S. Kim, "Survey of Component-based software development", *IET Softw, Vol. 1, No. 2*, April 2007.

[3] William B. Frakes, Kyo Kang: Software Reuse and Research: Status and Future, IEEE Transactions on Software Engineering", Vol. 31, No. 7, July 2005

[4] Xichen Wang, Luzi Wang: Software Reuse and Distributed Object Technology, IEEE Transactions on Software Engineering, 2011.

[5] Sametinger: Software Engineering with Reusable Components, Springer-Verlag, ISBN 3- 540-62695-6, 1997.

[6] Jianli He, Rong Chen, Weinan Gu: A New Method for Component Reuse, IEEE Transactions on Software Engineering, 2009.

[7] Maurizio Pighin: A New Methodology for Component Reuse and Maintenance, IEEE Transactions on Softwrae Engineering, 2001.

[8] Yong-liu, Aiguang-Yang: Research and Application of Software Reuse, ACIS International Conference on Software Engineeing, Artificial Intelligence, IEEE, 2007.

[9] Roger S. Pressman, "Software Engineering: A Practitioner's Approach", 5th Edition, McGraw Hill, 2001.

[10] Yourdon, E,"Software Reuse", Application Development Strategies,vol. 6, no.12, Dec 1994, pp.1-16.

[11] Sarbjeet Singh, Manjit Thapa, Sukhvinder Singh, Gurpeet Singh, "Software Engineering – Survey of Reusability Based on SoftwareComponent ", International Journal of Computer Applications (0975- 8887) Volume 8- No.12, October 2010.

[12] Marcus A.S. Boxall, Saeed Araban,"Interface Metrics for Reusability Analysis of Components", IEEE,ASWEC '04.

[13] N Md Jubair Basha, Salman Abdul Moiz, A.A Moiz Qyser, "Performance Analysis of HR Portal Domain Components Extraction ", International Journal of Computer Science & Information Technologies (IJCSIT), Vol2 (5), 2011, 2326-2331.

[14] Fuqing Yang, Bing Zhu, Hong Mei: "Reuse Oriented Requirements Modeling", Tsinghua University Press, Beijing, 2008.

[15] Youxin Meng, Xinli Wu, Yuzhong Ding," Research and Design on Product Quality Tracking System Based on Domain Engineering", IEEE, 2010.

[16] William Fakes, Ruben Prieto- Diaz, Christopher Fox, "DARE- COTS:A Domain Analysis Support Tool", IEEE, USA, 1997.

[17] Massimo Fenarlio, Andrea Valerio, "Standardizing Domain-Specific Components: A Case Study", ACM, Vol. 5, No.2, June, 1997

[18] Frederic Loiret, Ales Plesk, Phillipe Merle, Lionel Seinturier, Michl Malohlava, "Constructing Domain Specific Component Frameworks through Architecture Refinement", 35th Euromicro Conference on Software Engineering and Advanced Applications", IEEE, 2009.

[19] James F Peters, Witold Pedrycz, " Software Engineering, An Engineering Approach", Wiley India Private Limited, 2007.

[20] Marcus A.S.Boxall, Saeed Araban," Interface Metrics for Reusability Analysis of Components", Proceedings of 2004 IEEE Australian Software Engineering Conference (ASWEC' 04).

[21] Zhongjie Wang et.al, " A Survey of Business Component Methods and Related Technique", World Academy of Science, Engineering and Technology, pp.191-200, 2006.

[22] N Md Jubair Basha, Salman Abdul Moiz, " A Methodolgy to manage victim components using CBO measure ", IJSEA Vol.3(2) 2012, pp.87-96,2012.

[23] AlOmara, E. A., Peruma, A., Mkaouer, M. W., Newman, C., & Ouni, A. (2023). How is Software Reuse Discussed in Stack Overflow?. arXiv preprint arXiv:2311.00256.

[24] Poulin, Jeffrey S. *Measuring software reuse: principles, practices, and economic models*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[25] AlOmara, Eman Abdullah, et al. "How is Software Reuse Discussed in Stack Overflow?." *arXiv preprint arXiv:2311.00256* (2023).