Indonesian Journal of Electrical Engineering and Computer Science
Vol. 36, No. 2, November 2024, pp. 1092~1104
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v36.i2.pp1092-1104

Improved search method for classified reusable components on cloud computing

Adnan Rawashdeh[1], Mouhammd Alkasassbeh[2], Radwan Dwairi[1], Hani Abu-Salem[3], Hashem Al-Mattarneh[4]
[1]Department of Information Technology, Faculty of IT and CSs, Yarmouk University, Irbid, Jordan
[2]Department of Computer Science, Princess Summaya University for Technology, Amman, Jordan
[3]Department of Computer Science, University of South Carolina, Aiken, USA
[4]Department of Civil Engineering, Hijjawi Faculty for Engineering Technology, Yarmouk University, Irbid, Jordan

Article Info

Article history:

Received Dec 5, 2023
Revised Jul 21, 2024
Accepted Jul 29, 2024

Keywords:

Classification
Cloud computing
Clustering
Component-based development
Reusable components
Software reuse

ABSTRACT

Expanding development environments to accommodate huge amounts of reusable components along with associated maintenance and evolution responsibilities has become difficult and costly for software organizations to cope with, while benefits are limited to owner organizations. The challenge of organizing reusable assets so that finding the right component needed has always been a big challenge. The literature of software reuse lacks a comprehensive search method that is efficient and covers the entire system development lifecycle (SDLC). This research work attempts to make an efficient use of the cloud computing advantages and thus, encourages the migration of reusable components to the clouds. The maintenance, the search process and cost-related problems encountered with traditional in-house development environments can be resolved conclusively on the cloud. This research work proposes a multi-classification and clusters approach to migrate reusable components to the cloud. Accordingly, it applies indexing process to classified reusable components achieving efficient search. In addition, the proposed approach adopts a comprehensive SDLC-based classification to organize reusable components so that searching and finding an appropriate component becomes an easy task due to the fact it is bound to the particular undergoing phase. Cloud computing provides more storage and resources with low cost, compared to traditional in-house development environments.

Corresponding Author:

Adnan Rawashdeh
Department of Information Technology, Faculty of IT and CSs, Yarmouk University
P O Box 566, Irbid 21163, Jordan
Email: adnan.r@yu.edu.jo

1. INTRODUCTION

Software reusability means reusing an existing segment of source code in the development of a new system in order to add new functionality with little or no changes. In common engineering disciplines, systems are developed by assembling existing chunks of code that have been used in other systems [1]. Software engineering has traditionally concentrated on system development life-cycle itself, but more recently, software engineers have observed that to achieve quality software, faster and at lower cost, a systematic software reuse design process has to be adapted [2]. The principle of systematic software reuse is considered an essential strategy to achieve the saught long-term benefits of software reuse, this is supported by nemours articles in the literature including: [3]-[13]. On the other hand, Guha and Al-Dabass [14] described that cloud computing refers to the delivery of computing resources as a service over a network. The term originates from the cloud-shaped symbol used in system diagrams to represent the intricate infrastructure it encompasses.

Reusable components are typically kept in a repository as supported and described in many articles, including: [8], [10], [11], [15]-[18] and since the focus of this research work is on component-based reuse, it is worth describing a value-based central repository. Khan and Khan [19] summarized sixteen components and their interactions in such a setup, four of which are related to this research work. These include: (i) the audit system or access procedure to the central repository. It ensures that data is added and referenced into the repository correctly to facilitate retrieval systematic retrieval. (ii) A searching and report generation process to inform the project manager about the number of reusable components in order to communicate with clients regarding product delivery. (iii) In the central repository, tagging is done for each component to help in the classification of components with similar attributes. (iv) The legacy directory contains all components supported by the legacy systems. Legacy components are sub-categorized according to various third-party tools, codes, test cases, or design documents.

Furthermore, a set of rules associated with a value-based repository system has been identified in [19]. The related ones to this research work include: classification: the components should be classified before storing into the repository. Sub classification categories can be identified based on functionality and type of reusable components. Tagging: appropriate keywords can be associated with each component. The keyword tagging helps other users to easily find a specific component for reusability purpose. Indexing: a keyword indexing can be performed for search optimization of the reusable components stored in the repository.

In fact, classification and indexing are the two identified rules with most relevance to our research work in this paper. Malik [20] proposed an approach of two indexes to classify software components for reuse, while in this paper a new system development lifecycle (SDLC)-based approach is used and supported with three indexes to search reusable components during the implementation phase (coding) of a given software project.

Problem statement: many software organizations recognized that developing software with reusable components could significantly reduce development effort and cost while accelerating delivery. However, the lack of a standard process model in this field led to the overall failure of several approaches, including the Jasmine and Vasantha methods [21]. Reinhartz-Berger [22] stated that while software reuse reduces costs and effort, improves quality, and enhances productivity, it also presents challenges in retrieving existing artifacts and adapting them to the specific context.

Another aspect of the problem has been pointed out in, [6], [23], [24] there is no explicit integration of knowledge with software development projects, which forces developers to repeatedly search for similar and recurring solutions to tasks, instead of reusing this knowledge. Giordano *et al.* [25], identifies two limitations in the current literature: (i) the extent to which developers actually use code reuse mechanisms over time is unknown, and (ii) it is unclear how these mechanisms may contribute to explaining defect-proneness and maintenance effort during software evolution. Khan and Khan [19] outlined several drawbacks of traditional existing repository systems. These include: (i) un-standardized procedures for keeping reusable components in a repository led to the fact that developers would create folders of their own choice. Usually, they use their names for the folders instead of meaningful names that reflect the contents of folders. Thus, other developers would be unable to recognize them or utilize them. (ii) No control or any sort of management of developers accessing the repository. As a result, unnecessary data will be accumulated and decrease the efficiency of the system. (iii) Unlike locals, remote developers could not reach out to the repository asset. (iv) The lack of management policy to categorize and keep items with similar characteristics in one location makes the utilization of the repository more difficult. (v) Absence of indexing of the components residing in the repository makes tracking of the software components difficult.

Furthermore, in the literature [26]-[28] there are calls for innovative approaches that avoid re-designing and re-implementing software solutions, features, patterns, components, designs, tests, and so forth. Therefore, the problems to be handled in this research work include inefficiency of search method, disorganization, and difficulty of finding the appropriate reusable components. Literature review: software reuse is not yet a mature discipline, and offers many research opportunities [29]. One research attempt to show that fuzzy retrieval has an improved retrieval performance over typical boolean retrieval [30]. Other attempts went to develop a conceptual framework to investigate software reuse practices [28], [31] or achieving reuse through an enterprise architecture-based software capability profile [32]. While others, such as [26], [33] adapted opportunistic design, they argue that challenges associated with such a development model are quite different from traditional software development and reuse. Frakes and Kang [34], component-based software engineering (CBSE) has emerged directly from advancements in software reuse. An important area in software engineering has been designing software components for future reuse. The characteristics, desired properties, and design principles for CBSE have been extensively studied and analyzed. A software system developed using reusable components follows a ‘with’ reuse process, whereas a component designed for reuse in other systems follows a ‘for’ reuse process. In the ‘for’ reuse process, the key focus is to examine how components are built for reuse and how this process impacts the quality of the components.

Frakes and Kang [34] identified two sets of hypotheses. The first set of hypotheses is associated with ‘Design for Reuse’, these include, a reusable component is larger than its equivalent one-use component. A reusable component requires more development effort than its equivalent one-use component. When designing and building a reusable component, a developer is more productive compared to designing and building a one-use component. A reusable component typically has more parameters than its one-use counterpart. The second set of hypotheses is associated with ‘Design with Reuse’, and these include, the smaller the component, the easier it is to reuse, with size measured in source lines of code. A component designed and built following a specific reuse design principle is easier to reuse than one not built with that principle. Additionally, the more experience a programmer has, the easier it is to reuse a component. A component that has been tested by the user before reuse is easier to reuse than an untested one.

The concept of ‘Design with Reuse’ is the focus here. The reusable components (previously designed ‘for reuse’) are available, but they need to be organized in a way that makes it easy and efficient to search and find the appropriate ones accordingly. Numerous articles can be found in the literature concerning searching and classification of reusable components, including semantic-based and natural language search [11], [15], [35]; using the intent and contextual meaning behind a search query to deliver more relevant results. Such techniques rely on the user’s intent to formulate the query, not the developer of the component, as a result the search may not retrieve the exact desired components. Another set of articles [36], [37] claimed software reuse technology based on common factor method, and in [38] construction and utilization of problem-solving knowledge-base for dealing with problems, and to recommend potential causes of a problem based on multiple symptoms reported.

Furthermore, Hudaib *et al.* [39] used the Chidamber and Kemerer (CK) metrics suite to identify the reuse level of OO classes. The self-organizing map was used to cluster datasets of CK metrics values obtained from Java-based systems. The objective was to identify the relationship between CK values and the reusability level of the class; categorized as high, medium and low reusable. The research work focused solely on object-oriented classes, while ignoring the procedural programming paradigm. Likewise, Ma *et al.* [40] only focused on unified modeling language (UML) class-diagram.

Manjhi and Chaturvedi [41] claimed that a functional paradigm may facilitate code reuse more effectively than object-oriented or procedural programming paradigms. Their research centers on identifying reusable components within the functional programming paradigm. They conducted experiments using Haskell to analyze various software metrics of components. The self-organizing map algorithm was employed to cluster functions from three packages into three distinct classes based on their reusability. Function clustering was determined by metrics thresholds and their average values. Other types of reusable components, such as documents, diagrams, charts, and forms were not considered.

Nguyen *et al.* [42] had formerly proposed AURORA as a machine learning classifier for meta model repositories, and they attempted an improvement of AURORA. In both cases, their work was limited to meta model repositories. In addition, Namitha and Kumar [43] they proposed an algorithm for identifying the recurring concepts in data stream clustering, limiting the service to data streams.

Some of the researchers attempt to deploy reusability approach in specific medical application, such as Liu *et al.* [44], they provide a reliable and reproducible approach for genomic data management within a software tool (R) environment to enhance the accessibility and reusability of genomic data. Such approach is only limited to medical-applications associating with DNA data. It can be found in the reuse literature research work focused on early stages of the SDLC, namely planning, analysis, and design, such as Guber and Reinhartz-Berger [8], avoiding the inclusion of an important phase the coding or the implementation phase.

Therefore, the contribution of this research work is to overcome limitations found in previous related work as described above and propose an efficient approach to easily find the appropriate reusable components for a given requirement. The organization of the remainder of this paper is as follows: section 2 presents the methodology along with rationale justification and contribution of this research work. Section 3 presents a critical discussion of the findings, and section 4 draws the conclusions of this research and outlines associated future research work.

2. METHOD

The methodology of this research work consists of the following steps:

- Step 1: justifying the need for migrating reusable components to the cloud, and thus, describing cloud-computing model, advantages, benefit and drawbacks.

– Step 2: evaluating existing classification methods of reusable components, and consequently selecting the most appropriate combination of such methods to be adopted in this research work.
– Step 3: describing the proposed multiple classification approach for reusable components, including the introduction of a new classification based on the SDLC.
– Step 4: indexing and clustering reusable components to facilitate an efficient search and find process.

Figure 1, visualize the steps of the executed methodology in this research work.

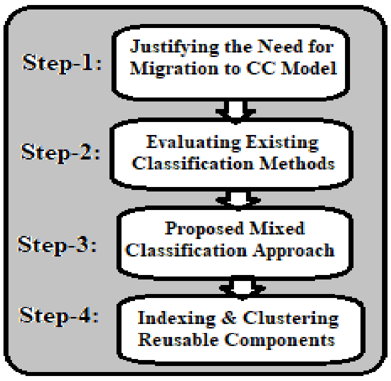
Step-1:
Justifying the Need for Migration to CC Model
Step-2:
Evaluating Existing Classification Methods
Step-3:
Proposed Mixed Classification Approach
Step-4:
Indexing & Clustering Reusable Components

Figure 1. Proposed methodology steps

2.1. Step 1: justification to migrate reusable components to the cloud

There are good reasons for business organizations to migrate their data and operations to the cloud. Numerous articles in the literature, including Brohi and Bamiah [45], and Millham [46], support this claim. In business organizations, users are required to pay only when they use resources. There are no maintenance costs associated with resource management across the system. Additionally, cloud computing models enhance business agility by allowing the system to scale up or down as needed to meet business demands. Thus, cloud computing represents an evolution of several technologies, including grid computing, distributed computing, and service-oriented computing.

Brohi and Bamiah [45] summarized the complete structure of the cloud-computing paradigm:
- Stakeholders (providers, users and adapters).
- Services (SaaS, PaaS, and IaaS). Both software as a service (SaaS), and platform as a service (PaaS) are considered appropriate examples for a setup that accommodates reusable components, as proposed in this research work. Developers can use such a cloud model, namely a compiler package, to build and test their software applications on the cloud itself; and that is PaaS. Although SaaS refers to running applications provided to clients as a service, simply downloading or copying the reusable components to be deployed on developers' own workspace is also considered SaaS. Infrastructure as a service (IaaS) is not of concern to this work. These applications can be accessed through a thin layer interface such as a web-browser. Consequently, system developers do not need to install and buy licensed applications or even heavy hardware. They simply pay per use.
- Development models (public, private, hybrid and community clouds).
- Challenges (trust and privacy, interoperability and reliability).
- Benefits (cost reduction, easy scalability and increased productivity).

Furthermore, Sinha *et al.* [47] in their book highlight the need for cloud computing and the real benefits derived by its application. This supports the claim in this research work that cloud computing paradigm is an excellent environment for reuse application.

2.2. Step 2: evaluation of existing classification methods

Classification of reusable components is the process of assigning one to a group of similar components. Classification helps in identifying what criteria may be used in the matching process. Reusable components should be organized in a way that permits an easy retrieval process.

Typical criteria for a classification plan of reusable components support the following:
– Continually expanding collections of reusable components;
– Finding components that are similar or approximate, not just exact matches;
– Finding functionally equivalent components across domains;
– Being precise and having high descriptive power;

– Being easily usable by end user, and agreeable to automation.

According to Malik [20], Rao and Niranjan [48], several attempts can be found in the literature to classify components. These include free text classification that is used with textual nature components, such as documents. Keywords are used in such a classification system. For a given number of searches used to retrieve appropriate information, each search with a different keyword returns different components. Consequently, all component-indexes are searched in order to find an entry. The drawback of free-text classification is that it requires high cost for the indexing process, while keywords used are uncertain in nature. In addition, the outcome of a search process may result in several irrelevant components. Furthermore, as claimed by Coullon *et al*. [49] the complexity of distributed software creates a necessity for a suitable methodology to ensure the correctness of setting up component-based systems.

Enumerated classification is hierarchical structure consisting of several levels of classes and subclasses, whereas the leaf-level classes are the actual components. The drawback of enumerated classification alone is that it does not provide a useful classification technique for reusable software components because it is difficult to expand. On the other hand, the enumerated classification is a very fast method.

Formal specification-based classification based on the description of the component's function. According to Houhamdi [50], each component is indexed with a formal specification that indicates its functional behavior. With software reuse and retrieval, formal specification-based classification can assist in discovering whether one component can be substituted for another, or how a component can be changed to meet certain requirements.

Signature matching, in this classification, each component is given a signature. For example, if C1 and C2 are the components then C_signature can be represented as follows: c_signature: match (c1, c2) = match (returns type and parameter of c1, c2) for instance, integer add and subtract operations have the same signature but opposite behavior; the c library routines strcpy and strcat have the same signature but completely different behavior. For repositories with huge numbers of reusable components, users will frequently encounter a situation where components have identical signatures but very different behaviors, which considered a potential problem.

Faceted classification, the idea of this classification is to identify descriptive features of components, such as functions, data manipulated by components or the context in which the component is used. These descriptive features are called facets. The next step is to prioritize the facets based on their importance and then connect them to a component. The faceted classification is useful to categorize the wide variety of reusable components in a repository. The collection of attributes that describe a component is referred to as the facet descriptor. This includes aspects such as function, object type, or system type. Additionally, keywords are assigned to these facets for each component stored in a reuse repository. Developers use a keyword to search for a possible component. Malik [20] claimed that faceted classification has proven to be an effective technique to create libraries of reusable components, irrespective of its limitations.

Attribute value classification is like faceted classification, a set of attributes is defined for all components in a domain area, and such set is used to classify a component by assigning values to the identified attributes. Unlike faceted classification, there is no restriction on the number of attributes that can be utilized, nor is there prioritization among them, which is seen as advantageous. For instance, a book can have numerous attributes like the author, publisher, ISBN number, and classification code in the dewey decimal system. However, the drawback of attribute value classification is that it is slow.

Cluster-based classification is unlike typical classification, which is simply some sort of supervised learning. Clustering algorithms, a form of unsupervised learning, organize a set of reusable components into clusters. The goal is to identify clusters that are internally cohesive yet distinct from one another. Components within the same cluster should be highly similar, while also differing significantly from components in other clusters. There are two common types of clustering, distance-based and conceptual clustering. For example, reusable components would be grouped according to their descriptive concepts.

Folksonomy approach of classification (FTC) is a distributed classification system, typically established by a team of resource users. Users provide tags (can be thought of as keywords, category names or metadata) to online items, such as images, videos, and text. These tags are then shared and sometimes refined by users. The advantage of this method is its simplicity; users can use it without a need for training or background knowledge. On the other hand, ambiguity of the tags can emerge while users apply the same tag in different ways without guidelines. Thus, the major drawback of FTC is the imprecise tagging.

2.3. Step 3: proposed approach toward classification of reusable components

In this research, a multi-method classification approach is used. As mentioned in the above section, each individual classification method has some drawbacks, and each method may suit a certain descriptive feature, such as type, functionality or name. Therefore, choosing any single classification method would not solve the problem efficiently. Consequently, the choice of a multi-method classification came as an attempt to overcome the drawbacks of individual methods while maximizing the benefits of using more than one method.

The first classification method is dedicated to organizing major existing application domains. These include, banking, inventory, payroll, e-commerce, insurance, healthcare, airline reservation, user-interface, educational, communication and many others. More than one choice can be applicable to classify these application domains, for example, free text classification FTC. Both of these approaches use keywords or tags to identify certain application domains. Herein, FTC will be used to manage existing application domains due to the simplicity of choosing a suitable tag. No training is necessary, and most users will simply identify a commonsense tag.

Table 1 illustrates a possible outline of such tagging process. The first column from the left-hand-side represents high-level tags, each of which can be used to group multiple sets of systems in a single application domain. Obviously, Table 1 can accommodate any number of existing domains. The second column shows the associated sub-tags that belong to a higher-level domain. For example, forms, buttons, labels, data-fields, and titles are sub-tagged, and all belong to the high-level tag 'GUI-objects'. The third column describes the possible items that can be found under each sub-tag. For example, input/output forms and error messages are described as forms.

Table 1. Illustration of possible tagging for existing application domains

High-level tag	Sub-Tag	Description
1. Functional-code	Finance	Indicates all types of financial systems, including banking, ATMs, credit card, investment, and money exchange.
	Reserve	Indicates all types of reservation systems, including airlines, hotels, and car rentals.
	Inventory	Indicates all types of inventory systems, including typical inventory items for grocery stores, and pharmacies.
	Insurance	Indicates all types of insurance systems, including health, life, vehicles, homes, and assets.
	Payroll	Indicates all types of payroll systems, including typical based-salary employees, hourly-wage employees, consultants, etc.
	+more…	+description…
2. GUI-objects	Forms	Indicates all types of object-forms used in the user-interface systems, including input-forms, out-put, and error-messages.
	Buttons	Indicates all types of object-buttons used on forms.
	Labels	Indicates all types of object-labels, headers or titles used on forms.
	+more…	+description…
3. Data	Test-cases	Indicates different groups of test cases, including credit card numbers, social security-numbers, ISBN numbers, grades, etc.
	Datasets	Indicates different groups of datasets, machine-tweets data sets, and addresses.
	Database	Indicates different groups of databases, including inventory, and payroll.
	+more…	+description…
:	:	:
4. Documents	Requirements	Indicates different groups of requirements of analysis documentations, including preliminary-requirements, and specification-requirements.
	Design	Indicates different groups of design documents, including data-flow diagrams, ER diagrams, use-cases diagrams, and sequence diagrams.
	Reports	Indicates different groups of reports, including progress reports, and maintenance reports.
	+more…	+description…

A complement contribution of this research work is introducing a new classification method based on the SDLC. Users look for reusable components that are typically associated with the current phase of the SDLC of their project. For example, in the planning phase two (or more) categories of reusable components; document and charts. Except for the coding phase, all other phases have similar types of reusable components, including 'documents,' 'charts,' and 'forms.' The source code will be handled differently and thus; it is the focus of this paper. Table 2 shows the complete set of possible categorizations of reusable components in each phase of the SDLC, except for the coding phase. Reusable code components will be handled through faceted-classification, clustering, and indexing approach.

Table 2. Illustration of possible SDLC-based classification of reusable components

Phase	Software artifact	Examples of related software artifacts
Planning	Documents	– Project-scope description
		– Alternative solutions and their feasibility
		– Resource plan description
		– Risk analysis description
		– Communication plan description
		– Feasibility report
	Charts	– Gantt-chart project plan
		– Work break-down structure chart
		– Organization chart, etc.
Analysis	Documents	– Problem description
		– Sources/sinks descriptions
		– Processing description
		– List of requirements
		– Final specification requirements
	Charts	– Existing system's functional modeling diagrams (eg. DFDs, SSD, use-case, activity, and state-machine)
		– Data modeling diagrams (eg. ERD, class, and cause-effect)
	Forms	– Questionnaires
		– JAD forms
		– User-cards/story-cards
Design	Documents	– Solution descriptions
		– Input and output descriptions
		– Structure English descriptions
		– Quality attributes descriptions
	Charts	– New system's functional modeling diagrams (eg. DFDs, SSD, use-case, activity, and state-machine)
		– Data modeling diagrams (eg. ERD, class)
		– GUI dialogue diagram
	Forms	– Input forms
		– Output forms
		– Error-messages
Coding	…………	Reusable code components will be handled through:
		– Multi-classification Approach
		– Clustering and indexing
Testing	Documents	– Testing strategy description. (eg. unit, integration, system, and validation tests)
	Test cases	– Numeric test cases
		– Alphabetical
		– Alpha-numeric
		– Dates

2.4. Step 4: indexing and clustering of reusable components

To facilitate an efficient search for the appropriate reusable component, the proposed approach uses three indexes to browse for components. The first set of components is grouped into clusters. Each cluster is a group of similar type components that are coherent internally. Clusters are created on the basis of selected attributes, namely functionality and type of components. For example, user interface objects are grouped in one cluster, sorting-functions in another cluster, controllers in a third cluster, and so on. Figure 2 illustrates a possible different grouping of several clusters within a selected application domain. Based on the idea of identifying different clusters, the first index (cluster index) is created to search clusters for a type of required component. Figure 3 illustrates the first index, which is the cluster index.

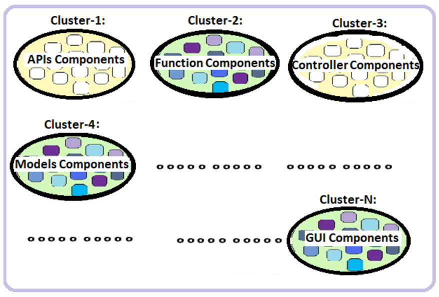

Figure 2. Grouping of clusters

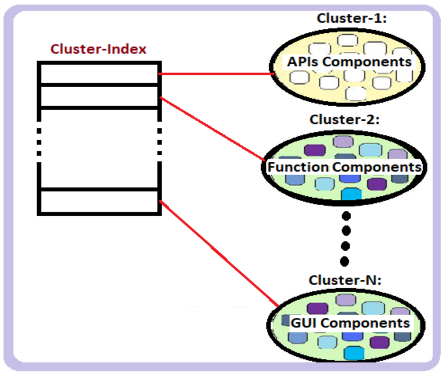

Figure 3. Indexing for clusters

The use of the cluster approach initially narrows the search space. For example, if the user is working on a graphical user-interface system, then cluster-N in Figure 3 will be the target to search for related components. Such components could be a ‘Form,’ ‘Button,’ ‘Label,’ or other related component. Now, further faceted classification method is applied to classify categories of similar components within a single cluster. Thus, a cluster might contain one or more categories of related reusable components, depending on the range of identified components. Figure 4 illustrates a possible categorization of cluster-2: function. Based on the idea of identifying different categories in a single cluster, the second index (category index) is created to search each category within a given cluster. Figure 5 illustrates the second index applied on cluster-2: function.

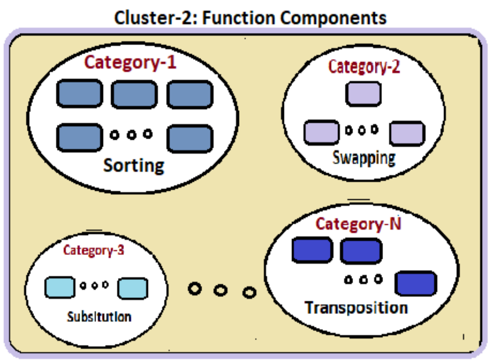

Figure 4. Possible categorization of single cluster

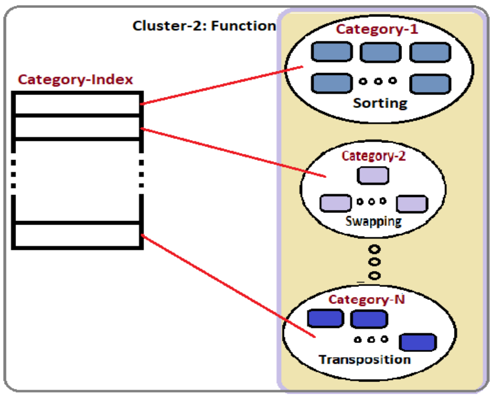

Figure 5. Indexing for categories

Faceted classification method is applied to classify each component within a cluster-category. Thus, a cluster-category might contain one or more reusable components, depending on the number of existing ones. To explain this last level of indexing, Figure 6 illustrates an example of using facet classification, assuming one sort of component among several others within category-1: sorting within cluster-2: function.

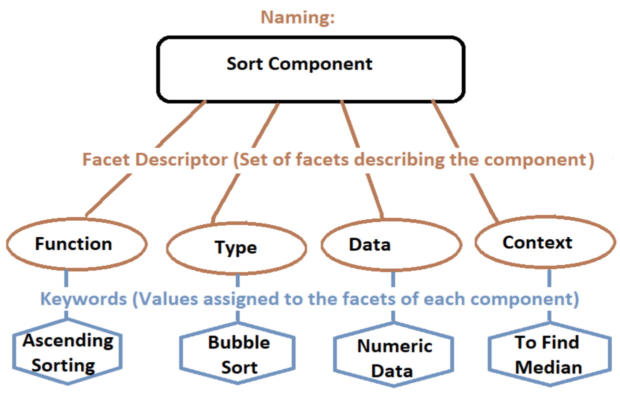
Naming:
Sort Component
Facet Descriptor (Set of facets describing the component)
Function
Type
Data
Context
Keywords (Values assigned to the facets of each component)
Ascending Sorting
Bubble Sort
Numeric Data
To Find Median

Figure 6. Illustration of using faceted classification on a sort component

Like the sort-component indicated in Figure 6, another can be added as a reusable sort-component with the following descriptor:

Sort-Component: #2
Function: Descending Sorting
Type: Quick Sort
Data: Alphabets
Context: Sorting Customers YTD purchases

Consequently, if multiple sort algorithms should be acquired in a repository, this requires several reusable components, including: Bubble-Sort, Quick-Sort, Selection-Sort, Insertion-Sort, Merge-Sort, Counting-Sort, Radix-Sort, Bucket-Sort, Heap-Sort, and Shell-Sort.

The advantage of faceted classification is its flexibility. The linked keywords to the facets can be added, changed, or removed easily. On the other hand, the user who intends to make good use of the reusable components must acquire certain knowledge, which may be considered a limitation. However, the researcher believes that the benefits override such limited drawbacks since developers should acquire knowledge of functionality for different components. To conclude the methodology, the last step is to create the third index on components. Figure 7 illustrates the use of the third index (component-index).

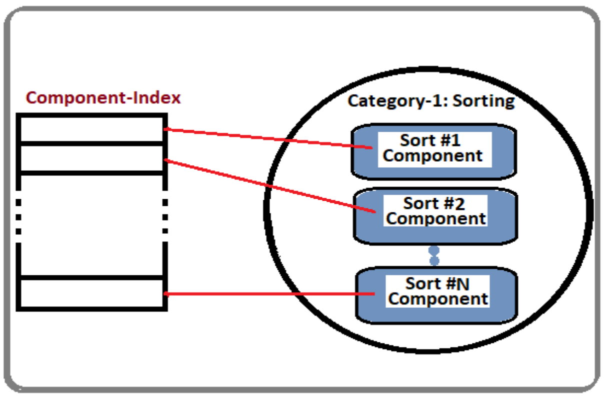

Figure 7. Indexing for components

3. DISCUSSION OF THE OUTCOME

To facilitate the process of finding appropriate reusable components for a given set of requirements, a new cloud-computing-based approach is proposed as described in section 2. The search process for a suitable component is guided by the current phase of the SDLC. For example, when the developer is working on activities during the analysis phase, the search process for a required item is straightforward. It is simply performed through free text classification with textual nature and key word, such as document. This will be also true for most of the phases except for the coding phase.

To facilitate the process of finding appropriate code component during the implementation phase the proposed multiple classification and clustering methods approach is used. It illustrates a hierarchy of reusable components grouped together according to application domains, and then clustered based on selected attributes, including functionality and type. Subsequently, within each cluster, coherent reusable components are grouped together as a separate category. When a developer wishes to find a reusable component, the first action is to pick an appropriate application domain, after which a key-search value, such as component name, is used for the search process. As a result, a list of related clusters will be displayed. The 'Cluster Index' will help the developer to find the appropriate cluster. Next, the 'Category Index' will be used to narrow the search by avoiding unrelated categories within a given cluster. Finally, the 'Component Index' will bring out the most related component. While examining components for suitable match to a given requirement, the developer makes a choice of adoption or re-starting a new search.

Furthermore, the following algorithm summarizes the search process as described in the proposed approach:

```
Do
        -Browse the library for the appropriate application domain.
        - Select the appropriate domain.
        - Enter (name) the current phase of the SDLC.
        - Begin if
                Planning, Analysis, Design or Testing
                Search for documents, charts or forms.
        Else (# current phase is coding)
            o   Use the first index to search for appropriate cluster.
            o   Use the second index to search for the most-related category.
            o   Use the third index to search for the most-related component.
        End if
        - Begin if found
                Adopt component
                Found = TRUE
        Else
                Message "Do another search"
                Found = FALSE
        End if
Until Found
```

Irrespective of the current phase of the SDLC, developers can perform any possible search process for a reusable component very efficiently. The short-cut of such a search will bring out an appropriate reusable component, and thus, over perform any other existing methods as described in section 2 (step 2).

4. CONCLUSION AND FUTURE WORK

There are several existing methods that can be used to classify reusable components, each of which has some limitations. It can be concluded that in this research work, a proposed approach demonstrates an efficient search process for reusable components based on multiple methods of classification guided by the current phase of the SDLC. The accordingly classified reusable components are residing on the cloud allowing high availability service and maximizing the benefits worldwide. However, there are some limitations to the proposed approach concerning the monitoring of such a cloud setup to gather statistics of the actual usage of the components. It will be a good idea to record developers' choices of reusable components. The record can contain date/time, module name, application domain, developer name. Consequently, recording components' adoption will be used to rank the components; so that a more reusable component has a higher rank. The rank value assigned to a component can be used to enforce priority of displaying the component in the next relevant search. Such a technique could further enhance the search process and facilitates finding the right component for the matching requirement. Thus, additional research is needed to incorporate usage features including ranking.

REFERENCES

[1] V. R. Basili and H. D. Rombach, “Support for comprehensive reuse,” *Foundations of Empirical Software Engineering*, pp. 179–205, 2005, doi: 10.1007/3-540-27662-9_14.

[2] R. Guha, “Toward the intelligent web systems,” in *2009 1st International Conference on Computational Intelligence, Communication Systems and Networks, CICSYN 2009*, Jul. 2009, pp. 459–463, doi: 10.1109/CICSYN.2009.25.

[3] X. Chen, M. Usman, and D. Badampudi, “Understanding and evaluating software reuse costs and benefits from industrial cases—A systematic literature review,” *Information and Software Technology*, vol. 171, p. 107451, Jul. 2024, doi: 10.1016/j.infsof.2024.107451.

[4] E. S. de Almeida and G. M. Kapitsaki, “Introduction to the special issue on ‘Software Reuse,’” *Journal of Systems and Software*, vol. 137, p. 216, Mar. 2018, doi: 10.1016/j.jss.2017.11.049.

[5] D. Bombonatti, M. Goulão, and A. Moreira, “Synergies and tradeoffs in software reuse – a systematic mapping study,” *Software - Practice and Experience*, vol. 47, no. 7, pp. 943–957, 2017, doi: 10.1002/spe.2416.

[6] J. L. Barros-Justo, F. Pinciroli, S. Matalonga, and N. Martínez-Araujo, “What software reuse benefits have been transferred to the industry? A systematic mapping study,” *Information and Software Technology*, vol. 103, pp. 1–21, Nov. 2018, doi: 10.1016/j.infsof.2018.06.003.

[7] J. L. Barros-Justo, N. Martnez-Araujo, and A. González-Garca, “Software reuse and continuous software development: a systematic mapping study,” *IEEE Latin America Transactions*, vol. 16, no. 5, pp. 1539–1546, May 2018, doi: 10.1109/TLA.2018.8408452.

[8] J. Guber and I. Reinhartz-Berger, “Privacy-compliant software reuse in early development phases: a systematic literature review,” *Information and Software Technology*, vol. 167, p. 107351, Mar. 2024, doi: 10.1016/j.infsof.2023.107351.

[9] J. Long, “Software reuse antipatterns-revisited,” *Software quality professional*, vol. 19, no. 4, p. 4, 2017.

[10] Z. Shen and M. Spruit, “A systematic review of open source clinical software on GitHub for improving software reuse in smart healthcare,” *Applied Sciences (Switzerland)*, vol. 9, no. 1, p. 150, Jan. 2019, doi: 10.3390/app9010150.

[11] N. Ali, H. Daneth, and J. E. Hong, “A hybrid DevOps process supporting software reuse: a pilot project,” *Journal of Software: Evolution and Process*, vol. 32, no. 7, Jul. 2020, doi: 10.1002/smr.2248.

[12] A. A. Magableh, H. B. Ata, A. A. Saifan, and A. Rawashdeh, “Towards improving aspect-oriented software reusability estimation,” *Scientific Reports*, vol. 14, no. 1, p. 13214, Jun. 2024, doi: 10.1038/s41598-024-62995-z.

[13] A. Rawashdeh, “Assessment of system development models with regard to software reuse practice,” *The 1st International Conference on Digital Communications and Computer Applications*, pp. 1164 –1177, 2007.

[14] R. Guha and D. Al-Dabass, “Impact of Web 2.0 and cloud computing platform on software engineering,” in *Proceedings - 2010 International Symposium on Electronic System Design, ISED 2010*, Dec. 2010, pp. 213–218, doi: 10.1109/ISED.2010.48.

[15] T. Diamantopoulos and A. L. Symeonidis, *Mining software engineering data for software reuse*. Cham: Springer International Publishing, 2020.

[16] G. Botterweck and C. Werner, *Mastering scale and complexity in software reuse*, vol. 10221 LNCS. Cham: Springer International Publishing, 2017.

[17] R. Capilla, B. Gallina, and C. Cetina Englada, “The new era of software reuse,” *Journal of Software: Evolution and Process*, vol. 31, no. 8, Aug. 2019, doi: 10.1002/smr.2221.

[18] A. Rawashdeh, B. Matalkah, and A. Hammouri, “A hybrid AHP-VIKOR methodology to evaluate for adoption COTS database components based on usability,” *International Journal of Computer Applications in Technology*, vol. 56, no. 4, pp. 264–274, 2017, doi: 10.1504/IJCAT.2017.089090.

[19] M. Z. Khan and M. N. A. Khan, “Enhancing software reusability through value based software repository,” *International Journal of Software Engineering and its Applications*, vol. 8, no. 11, pp. 75–88, 2014, doi: 10.14257/ijseia.2014.8.11.07.

[20] N. Malik, “Proposed classification approach for software component reuse,” *International Journal of Electronics and Computer Science Engineering (IJECSE)*, vol. 1, pp. 1993–1999, 2012, [Online]. Available: www.ijecse.org.

[21] K. S. Jasmine, “A new process model for reuse based software development approach,” *Proceedings of the World Congress on Engineering*, vol. 1, p. 576, 2008.

[22] I. Reinhartz-Berger, “Challenges in software model reuse: cross application domain vs. cross modeling paradigm,” *Empirical Software Engineering*, vol. 29, no. 1, 2024, doi: 10.1007/s10664-023-10386-9.

[23] G. Melo, T. Oliveira, P. Alencar, and D. Cowan, “Knowledge reuse in software projects: retrieving software development Q&A posts based on project task similarity,” *PLoS ONE*, vol. 15, no. 12 December, 2020, doi: 10.1371/journal.pone.0243852.

[24] R. Adnan and M. Bassem, “A new software quality model for evaluating COTS components,” *Journal of Computer Science*, vol. 2, no. 4, pp. 373–381, 2006.

[25] G. Giordano, G. Festa, G. Catolino, F. Palomba, F. Ferrucci, and C. Gravino, “On the adoption and effects of source code reuse on defect proneness and maintenance effort,” *Empirical Software Engineering*, vol. 29, no. 1, p. 20, Jan. 2024, doi: 10.1007/s10664-023-10408-6.

[26] N. Mäkitalo, A. Taivalsaari, A. Kiviluoto, T. Mikkonen, and R. Capilla, “On opportunistic software reuse,” *Computing*, vol. 102, no. 11, pp. 2385–2408, 2020, doi: 10.1007/s00607-020-00833-6.

[27] I. Schaefer and I. Stamelos, “Introduction to the special issue on ‘international conference on software reuse 2015,” *Journal of Systems and Software*, vol. 131, pp. 323–324, Sep. 2017, doi: 10.1016/j.jss.2017.07.007.

[28] M. Svahnberg and T. Gorschek, “A model for assessing and re-assessing the value of software reuse,” *Journal of Software: Evolution and Process*, vol. 29, no. 4, Apr. 2017, doi: 10.1002/smr.1806.

[29] J. L. Barros-Justo, F. B. V. Benitti, and S. Matalonga, “Trends in software reuse research: a tertiary study,” *Computer Standards and Interfaces*, vol. 66, p. 103352, Oct. 2019, doi: 10.1016/j.csi.2019.04.011.

[30] E. Colvin and D. H. Kraft, “Fuzzy retrieval for software reuse,” *Journal of the Association for Information Science and Technology*, vol. 67, no. 10, pp. 2454–2463, Oct. 2016, doi: 10.1002/asi.23584.

[31] M. A. F. Osman, M. N. Masrek, and K. A. Wahid, “Software reuse practices among malaysian freelance developers: a conceptual framework,” in *International Academic Symposium of Social Science 2022*, Sep. 2022, p. 30, doi: 10.3390/proceedings2022082030.

[32] A. Belfadel, E. Amdouni, J. Laval, C. B. Cherifi, and N. Moalla, “Towards software reuse through an enterprise architecture-based software capability profile,” *Enterprise Information Systems*, vol. 16, no. 1, pp. 29–70, Jan. 2022, doi: 10.1080/17517575.2020.1843076.

[33] T. Mikkonen and A. Taivalsaari, "Software reuse in the era of opportunistic design," *IEEE Software*, vol. 36, no. 3, pp. 105–111, May 2019, doi: 10.1109/MS.2018.2884883.
[34] W. B. Frakes and K. Kang, "Software reuse research: status and future," *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 529–536, Jul. 2005, doi: 10.1109/TSE.2005.85.
[35] R. Capilla, B. Gallina, C. Cetina, and J. Favaro, "Opportunities for software reuse in an uncertain world: From past to emerging trends," *Journal of Software: Evolution and Process*, vol. 31, no. 8, Aug. 2019, doi: 10.1002/smr.2217.
[36] X. Chen, D. Badampudi, and M. Usman, "Reuse in contemporary software engineering practices – an exploratory case study in a medium-sized company," *E-Informatica Software Engineering Journal*, vol. 16, no. 1, p. 220110, 2022, doi: 10.37190/e-Inf220110.
[37] X. F. She and B. B. Bian, "A software reuse technology based on common factor method and its application in fast programming," *Journal of Physics: Conference Series*, vol. 2158, no. 1, p. 012001, Jan. 2022, doi: 10.1088/1742-6596/2158/1/012001.
[38] H. M. Koo and I. Y. Ko, "Construction and utilization of problem-solving knowledge in open source software environments," *Journal of Systems and Software*, vol. 131, pp. 402–418, Sep. 2017, doi: 10.1016/j.jss.2016.06.062.
[39] A. Hudaib, A. Huneiti, and I. Othman, "Software reusability classification and predication using self-organizing map (SOM)," *Communications and Network*, vol. 08, no. 03, pp. 179–192, 2016, doi: 10.4236/cn.2016.83018.
[40] Z. Ma, Z. Yuan, and L. Yan, "Two-level clustering of UML class diagrams based on semantics and structure," *Information and Software Technology*, vol. 130, p. 106456, Feb. 2021, doi: 10.1016/j.infsof.2020.106456.
[41] D. Manjhi and A. Chaturvedi, "Software component reusability classification in functional paradigm," in *Proceedings of 2019 3rd IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2019*, Feb. 2019, pp. 1–7, doi: 10.1109/ICECCT.2019.8869123.
[42] P. T. Nguyen, J. Di Rocco, L. Iovino, D. Di Ruscio, and A. Pierantonio, "Evaluation of a machine learning classifier for metamodels," *Software and Systems Modeling*, vol. 20, no. 6, pp. 1797–1821, Dec. 2021, doi: 10.1007/s10270-021-00913-x.
[43] K. Namitha and G. S. Kumar, "Learning in the presence of concept recurrence in data stream clustering," *Journal of Big Data*, vol. 7, no. 1, p. 75, Dec. 2020, doi: 10.1186/s40537-020-00354-1.
[44] Q. Liu, Q. Hu, S. Liu, A. Hutson, and M. Morgan, "ReUseData: an R/Bioconductor tool for reusable and reproducible genomic data management," *BMC Bioinformatics*, vol. 25, no. 1, p. 8, Jan. 2024, doi: 10.1186/s12859-023-05626-0.
[45] S. N. Brohi and M. A. Bamiah, "Challenges and benefits for adopting the paradigm of cloud computing," *International Journal of Advanced Engineering Sciences and Technologies*, vol. 8, no. 2, pp. 286–290, 2011.
[46] R. Millham, "Software asset re-use: migration of data-intensive legacy system to the cloud computing paradigm," in *Software Reuse in the Emerging Cloud Computing Era*, IGI Global, 2012, pp. 1–27.
[47] S. Sinha, V. Bhatnagar, P. Agrawal, and V. Bali, *Integration of Cloud Computing with Emerging Technologies Issues, Challenges, and Practices*. Boca Raton: CRC Press, 2023.
[48] C. V. G. Rao and P. Niranjan, "An integrated classification scheme for efficient retrieval of components," *Journal of Computer Science*, vol. 4, no. 10, pp. 821–825, Oct. 2008, doi: 10.3844/jcssp.2008.821.825.
[49] H. Coullon, L. Henrio, F. Loulergue, and S. Robillard, "Component-based distributed software reconfiguration:a verification-oriented survey," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–37, Jan. 2023, doi: 10.1145/3595376.
[50] Z. Houhamdi, "A formel language for software reuse," *Computer Science Department*, 2003.

BIOGRAPHIES OF AUTHORS

Dr. Adnan Rawashdeh holds a Ph.D. in CS/Software Engineering from Illinois Institute of Technology (IIT), Chicago, IL, USA (1996). He occupied a Sys Admin position with a municipal bonds trading firm, HSE, in Chicago, USA (1992-1996). He has been working as an IT faculty member with several domestic and regional universities, and occupied several administrative positions, including, Vice Dean, and Head of IT Department. He taught several courses, including: systems analysis and design, software engineering, scientific research methods, and data analysis and cybersecurity. Dr. Rawashdeh's research interests include: software reuse, software quality models, and cybersecurity. Currently, he is a faculty member at the IT Department, Yarmouk University, Irbid, Jordan. He can be contacted at email: adnan.r@yu.edu.jo.

Professor Mouhammd Alkasassbeh earned his degree from the School of Computing at Portsmouth University, UK, in 2008. He holds a full professorship in the Cybersecurity Department at Princess Sumaya University for Technology. His areas of research encompass network traffic analysis, network fault detection, network fault and anomaly classification, as well as the application of machine learning within the realm of computer networking and network security. He can be contacted at email: m.alkasassbeh@psut.edu.jo.

Dr. Radwan Dwairi holds a Ph.D. in Computer Science from the University of Bradford, UK (2010). Currently, he is an associate professor at the IT Department, Faculty of IT and CSs, Yarmouk University, Irbid, Jordan. He occubied admin positions, including Head of the IT Deptartment. Dr. Radwan's research interest include: electronic commerce, social commerce, technology adoption, data analytics, and systems analysis and design. He can be contacted at email: r.aldwairi@yu.edu.jo.

Prof. Hani Abu-Salem holds a Ph.D. in Computer Science from Illinois Institute of Technology (IIT). Currently, he is a full professor at the CS Dept, University of South Carolina, Aiken, USA. His research lies at the intersection of computer science and linguistics, specifically in Information Retrieval (IR) and Natural Language Processing (NLP) with a focus on multilingual data handling. He believes in open research and cooperation, utilizing interdisciplinary techniques to produce creative solutions. His aim is to foster cross-disciplinary collaboration and bridge the gap between academia, industry, and government. He can be contacted at: haniA@usca.edu.

Prof. Hashem Al-Mattarneh a Ph.D. in Civil Engineering from Universiti Kebangsaan Malaysia. Currently, he is a full professor at the Civil Engineering Department, Yarmouk University, Jordan. His research interest include: nondestructive testing and evaluation of civil engineering infrastructures, numerical modeling and optimization, repair and rehabilitation of concrete structures, development and evaluation of civil engineering materials, and composites using NN and AI. Dr. Al-Mattarneh is a member of several international organizations including, JEA, ACI, ASNT, IACT, IEEE, and ASCE. He was honered and awarded three gold medals from Switzerland, Spain, and Sweden for his scientific research. He can be contacted at email: hashem.mattarneh@yu.edu.jo.