Volume 1 | No. 2 | July – December 2017

Software Atom: An Approach towards Software Components Structuring to Improve Reusability

Muhammad Hussain Mughal[1], Zubair Ahmed Shaikh[2]

Abstract:

Diversity of application domain compelled to design sustainable classification scheme for significantly amassing software repository. The atomic reusable software components are articulated to improve the software component reusability in volatile industry. Numerous approaches of software classification have been proposed over past decades. Each approach has some limitations related to coupling and cohesion. In this paper, we proposed a novel approach by constituting the software based on radical functionalities to improve software reusability. We analyze the element's semantics in Periodic Table used in chemistry to design our classification approach, and present this approach using tree-based classification to curtail software repository search space complexity and further refined based on semantic search techniques. We developed a Global unique Identifier (GUID) for indexing the functions and related components. We have exploited the correlation between chemistry element and software elements to simulate one to one mapping between them. Our approach is inspired from sustainability chemical periodic table. We have proposed software periodic table (SPT) representing atomic software components extracted from real application software. Based on SPT classified repository tree parsing & extraction to enable the user to program their software by customizing the ingredients of software requirements. The classified repository of software ingredients assists user to exploit their requirements to software engineer and enables requirement engineer to develop a rapid large-scale prototype with great essence. Furthermore, we would predict the usability of the categorized repository based on feedback of users. The continuous evolution of that proposed repository will be fine-tuned based on utilization and SPT would be gradually optimized by ant colony optimization techniques. Succinctly would provoke automating the software development process.

Keywords: *Classification, Development, Prototyping, Extraction, Parsing, Re-usability, Software, Software Periodic Table (SPT), Softwares Repository.*

1. Introduction

Software industry is growing swiftly. Computing devices are interacting with human through software program. From personal assistant to business management and ubiquitous computation services, software gives solution for everything by automation that improves the efficiency and accuracy. In this emerging technologically evolving world significantly transformed the software development and diversity of software products, software organizations need to meet the market and their client requirements within short duration and with optimal quality. With increase in magnitude and complexity of the project the maintainability becomes difficult[1]. Software reuse does not only saves time and cost, but also give us reliable software product by integrating tested and reliable software components. We do not need to develop software from scratch, we extract the software components, which meet the

[1] Center for Research in Ubiquitous Computing, Department of Computer Science, Sukkur IBA University, Sukkur, Pakistan.
[2] Center for Research in Ubiquitous Computing, Department of Computer Science, Muhammad Ali Jinnah University, Karachi, Pakistan.
*Corresponding author: muhammad.hussain@iba-suk.edu.pk

requirements, and the software is ready for use in short compelled by industry pressure.

Software product line [2, 3] a family based approach software contributed a lot for reusability. Software industry acceleration is not chaseable. There is huge collection of software component developed already. To reuse software artifacts, we need a classification scheme to classify our software repository from that enable developer easily search and retrieve software artifacts based on project requirements. We proposed classification scheme based on domains, attributes and, features and functions of software system. Classification of software [4] allows us to organize collections of software artifacts into a efficient searchable structure. In last decade, various techniques applied for software artifacts classification.

Our motivation behind this work is the "elements periodic table" in chemistry, where all the atoms exist in this world are classified. Everything that we see around us is either compound or mixture of these atoms. Similarly, through identification of the atomic functionalities of existing software, and their transition to other software by mutation of their internal functional composition we can develop a relation between software components. We will develop general classification scheme for software, and this approach will revolutionize the software reuse by identifying the pattern of basic functionalities in a software. We can pick these basic functionalities from our repository and create new software from existing software. Moreover, it will assist developer for RAD, prototyping, and even user-developed software.

The organization of remaining paper is as below: In section II, explains the related work and some classification schemes. Section III defines problem statement. We will explain our proposed approach in section IV. Section V, implementation, and VI reflect the integration of the semantic model approach along with function coding scheme. Section VII is conclusion of our work and its limitations.

2. Related Work

Over the years, many researchers have proposed different classification schemes for software reuse. Following are some crucial approaches, which we have covered in our literature review.

Rajender Nath and Harish Kumar[5] used the keyword base search for software reuse. Their approach has three parts. For storage, the software component is stored in the form of component files, and an index is maintained which has the keywords related to the component. The authors in [6] proposed an approach for efficient software component retrieval for reuse. They suggested software component retrieval system based on ontology, metadata, and faceted classification for storage and retrieval of components. [7] Proposed an approach to use automatic classification of software identifiers by using text based classification algorithms such as N-gram to split the identifiers and extract domain knowledge concepts. They have reviewed many text based classification algorithms and proposed their own algorithm called sword. The algorithm creates a set of candidate splitting terms and matches these terms with the list abbreviation found and analyzed in source code, and a list of words from dictionary. The terms, which are fully matched are retained from the list and are ranked using the score function from samurai algorithm [6]. Software reuse library is organized using faceted classification[8] scheme. Search and retrieval of different software artifacts and library functions is very effective in this system. It is very difficult to organize a reusable software artifact that is why they have used the faceted classification scheme. This scheme gives higher level of accuracy and flexibility in classification. The limiting factor of technology used is its manual classification nature. They have also put some limitations on their reuse infrastructure. This paper has important role of domain analysis. Integration of reusable artifacts and their adaptation in the system is very difficult unless a high-level system is not proposed.

In [9], Lo, D. et al describes the technique depending on software reusable components are creation, management and extraction. Identifications for software function for reuse based on specification were used in[10]. Prieto-Diaz and Freeman [11] have proposed a software reuse library system, which they called Reuse Description Formalism (RDF), improves organization of software components. They have proposed two concepts forming the core of RDF's model: instance and classes. Instances include description of objects, which are reusable. Lo, Cheng [12], the step towards reliability of software using pattern mining techniques. They introduce classifier to generalize the failures and to identify the other unknown failures of it. Zina Houhamdi [13] defined the benefits of the software reuse, as it is a promising strategy for improvements in software quality, productivity and maintainability as it provides for cost effective, reliable, and accelerated. Software factories were developed extracting the pattern keeping in view critical axes of innovation from abstraction to specification[14].

3. Problem Statement

Software reuse enhances productivity and reliability of software product. It saves time and cost as there is no prior testing required for reusable software artifact. Our hypothesis is "To design a sustainable semantic software classification scheme that can incorporate the existing softwares designed without intent of reusability and support new software with semantic arrangement and efficient retrieval. From inspiration of "Element Periodic Table" in chemistry, we proposed semantic classification and retrieval.

3.1. Relation with Existing Approaches

In implementation of software classification, we have followed different research papers. Moreover, we found schemes relevant to our approaches given:

3.2. Faceted Classification:

Faceted classification scheme described by Gajala and Phanindra [4]presents solution to the problem many researchers face during classification. In faceted classification, classes and modules are assembled together and assigned predefined keywords from facets lists. It provides higher accuracy and flexibility in classification. Faceted classification scheme improves search and retrieval of reusable software artifacts and improves selection process of reusable artifacts.

In our approach, the periodic table and tree parsing are used for efficient organization of software components. Well-structured component improve accuracy of search and retrieval of artifacts and make flexible selection process of reusable artifacts.

3.3. Enumerative Scheme:

In this approach[4],all classes are predefined. It is mostly used in school libraries to arrange the books of different departments, like biology chemistry, computer etc. Librarian selects the books which best fit its location which illustration can be Dewey[13] Decimal system used to classify books.

In our project, we designed GUID for efficient search and storage. However, this scheme is one-dimensional with collision bucket support, means if we get more than massive similar reusable artifacts in with minor variation in one place, we save that item with collision number that would represent similar item with minor variation. Otherwise, it would not be scalable classification scheme. While in tree based, we evade this problem by determining the depth. Furthermore, we allocate the same position with structured metamorphosis of software component to improved scalability.

3.4. Attribute Value:

Gajala and Phanindra [4] uses set of attributes to classify an artifact. For example, different books in library have different attributes, like title, author, publisher, ISBN number, date, and a classification code in Dewey decimal system.

In our proposed work, we used set of attributes like version number, domain, classes, projects, and functionalities for transitional threshold to classify software position in our proposed software periodic table.

3.5. Free Text

Classification: Free text approach states that search and retrieval is made using text in the documents of artifacts. It is the keyword-based search. However, there are disadvantages of this approach. One is its ambiguous nature, and second it may search many irrelevant objects.

We have used keyword based search approach in our implementation work to search the particular software artifact from the repository by generating a code of that particular keyword that accelerates the search efficiency.

4. Our Approach

We explored the dense literature related to our study; we came up with a novel approach of classifying software for reuse influenced by chemical element periodic table. It arranges all the known elements in an informative array. Elements are arranged left to right and top to bottom in order of increasing atomic number.

4.1. Mapping chemical elements to Software elements

In this section, we are developing relationship between software and software elements. We mapped software function (set of commands to compute) to chemical atom. Molecule to program for example, molecules of CO_2 and CO both contains same type of atoms, but due to difference in the number of atoms, they exhibit different behavior. It is in case of software where mutation single function would change the behavior of program interface. This simulate the variation same genre of software. A combination of basic functionalities of software with computability heuristic is valid user program compared to unstructured or invalid structure. We can embed these valid tested function and/or programs to develop software component. We can identify the valid programs from existing repository as well as upcoming software collections.

In Periodic table, the different rows of elements are called periods. The period number of an element signifies the highest energy level an electron in that element occupies and grouped based on semantic commonalities. We have developed mapping relation in chemical and periodic table characteristics as shown in table 1.

Software based on the functionality are categorized in groups such system software, application software, etc. People visualize elements from organization pattern. By examining an element's position on the periodic table, one can infer the electron configuration. Elements that lie in the same column on the periodic table (called a "group") have identical configurations and consequently behave in a similar fashion chemically. For instance, all the group 18 elements are inert gases. The periodic table contains an enormous amount of important information. People familiar with how the table are structured can quickly determine a significant amount of information about an element[15]. From the software table where software lie in groups based on the functionality exhibited by software that recursively from functions. .Therefore, in our software repository, we are classifying the software in groups based functionality and represented by ontology supporting high visibility. We designed coding scheme for efficient retrieval.

Table-I: Mapping of Chemical Elements and Software Attributes

CHEMICAL ATTRIBUTES	SOFTWARE ATTRIBUTES
Atomic Number: The atomic number indicates the number of protons within the core of an atom. The atomic number is an important concept of chemistry and quantum mechanics. An element and its place within the periodic table are derived from this concept [15].	We the version number of software has format Major, Minor, Build. It means release date or year, also it means some minor updating in software and major means some major changes in software, like altering the design of software.
Atomic Mass: the name indicated the mass of an atom, expressed in atomic mas units (amu). Most of the mass of an atom is concentrated in the protons and neutrons in the nucleus [15].	We can relate it to the granularity of the software. The usability of software according to functionalities and features it. It can vary every time the software updated.
Density: The density of an element indicated the number or units of mass of the element that are present in a certain volume of the medium.	In software terms, it is related to complexity of software a measure of resources expended by a system, interacting with a piece of software perform a given task.

5. Software Periodic Table (SPT) Approach

We have merged two different aspects for software classification into groups and assigning codes for their position.

5.1. Categorization

Categorization is essential aspect software periodic table. In which, we will classify the software into groups (based on the grouping semantic of periodic table). We utilize the concept of periodic table in order to categorize software's according to their level of complexities. As in periodic table, its elements are categorized with their respective to chemical properties, atomic number, and electron configuration. Therefore, in SPT the organization of the software's according to their category of respective functionality and type features they provide. We assign the ID to each category based on type of software. We establish grouping scheme for top-level hierarchy for reducing search space complexity. Complexity of search space directly referred to the repository size of a particular dataset. It means for the software enumeration problem, we should have structure which traverse the repository easily and at every level downward reduce complexity. We should have the knowledge of iteration for a particular search referring the dataset; more precisely enumerator can point directly our required search based on GUID coding scheme. Our proposed structure can be indexed at any level and based on segments characteristic. This will increase the search efficiency. Large set problem is broken down in subset of category in order to reduce search space. The detailed layout of the distribution of the bits and calculations shown in the Figure 1. We consider the realization of software search problem as:

- All Software as a universal set S,
- Software Types (Application Software and System Software,…) ST are subset of S
- Software Category (For each in ST) are Set SCA

- SCS which are again subset of each software type
- S ={ Set of All Softwares}
- ST={All Application Software, System Software}
- SCA ={All software Categories of Application Software }
- SCS ={All software Categories of System Software }
- O ={ All software of other category }
- SCA= { ERP Solution, Grid solution, realtime application, ...n}. Similarly, SCA={Operating Systems, disk utilities, device drivers.. n
- SCS ={All software Categories of System Software }
- O ={ All software of other category }

Since SCA, SCS, and O etc, are disjoints set. Therefore, search space is divided at each level up to more than 50% finite number the categories are explained as follows.

- SCA= { ERP Solution, Grid solution, real-time application ...n}. Similarly, SCA={Operating Systems, disk utilities, device drivers.. n}

Now if we consider set of operating system

- OS ={ windows , Linux, mac, ..} and disk utilities
- DU = {dr. disk, disk cleaner...} then both OS and DU are disjoint set results reduction in search space.

5.1.1. Software Node: This is top node. Initially controled search pointer will be here as shown in Fig. 2. Down to the hierarchy the complexity of the search space will be reduced gradually.

5.1.2. Software Type Node:

This will be the second node in the software enumerator. There major type nodes would be assigned ID of 32 bit for each type. This segment of 32 bits (From left to right bit 1 to bit 32) will contain the information for System Software as per given below detail SCA={ERP Solution etc } bits (From bit 9 to bit 16) will be length of function, module, package, class name as following classification.

- Software Project
- Package/release
- Software Module
- Software Sub-Module
- Software Class/Structure

5.2. GUID coding

Since the periodic table the elements are being classified according to their atomic number in the increasing order, we classify the software with their GUID in SPT on some particular location.

GUID for SPT is assigned on semantic of software functionality bases groups in form of classes as shown in figure 2. The evolved software component with additional feature and functionalities change its version number to a higher one. For example, if we have a software name A with have GUID, evolved version will be stored by almost the same ID but flipping a bit on collision bucket. We design an enumerator in java programming language that reads the name of the software system, software project, package, module, sub-module, class-name, and function then generate the code for that in and if there are more than one function with the same name then C bit is set 1 for second 10 for third and so on for counting collisions. The anomalies in this scheme are that the sum of ASCII of different function may result in the same code, but there would be very few collision comparative reducing to search complexity less than O(logN).

This coding scheme is performed for each of the following node. The binary format for each part of 32 bits separated by: and further divided in 2 parts that's separated by "." The first part is further divided by - and part left side of "-"is number of collision C and right side of the"-"and before "." is the length L of that software system, software project, package, module, sub-module, class-Struct-name, function, and second part after "." will be the number of the function. Software system, software project, package, module, submodule, class-Struct-name, function. FN is number code of that Function assign that is generated base on summation of ASCII character of the Function name. For example, SUM =83+85+77= 245. Function/subroutine the enumerator converts the each category node (software project name, software package name, software module name, software sub-module name, software class-struct and function name) into equivalent binary code. Once the conversion is being completed then enumerator looks the repetition of each attribute in a file where the binary code value for each function detail is placed. If it finds the repetition then deals the particular attribute as collision and makes increment in the value by 1. Fig.1 represents the detail of collision of attribute and it also shows that how the binary value is represent in scheme for each of above five categorization nodes.

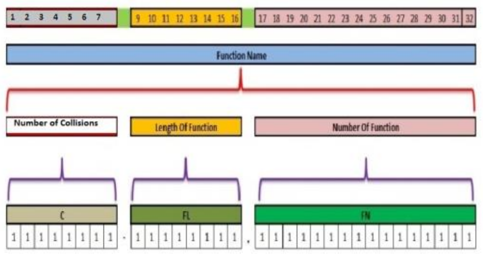


Figure 1: Layout of generating GUID for function name

The all bits one's represents absence associations in upper & lower hierarchy. Where the all bits zero represent absence of the particular nodes or skip codes. All other codes represent some node. The path root node to desired category is established and required source code file for the link is displayed to user. In the case of skip code or multiple version of same function or class multiple files are shared to user. Tree based representation. A tree is useful for exploring a large search space and reduces search complexity. Now in relation with our approach nodes in the tree are considered to be the domains, sub-domains, software, features, modules, classes etc. Branches of nodes are disjoints and leaves here would be the features, classes, modules, and functions of the software. We have used tree to show readers the flow of forthcoming software component and their auto-placement in the tree in their esteemed domain. The path from root node to leaf represents the semantics of the feature usability. Decision trees are very much helpful structures in building and interpreting because they are straightforward. The tree representation is efficient visualization and retrieval.

5.3. Software Quality

Quality of software is essential attribute in the Software development processes. The quality of the software has directed relation with customer satisfaction and organization's reputation. Developer appraisals, scheduling, and deadlines of release are affected by magnitudes of bugs in software products. The reusable software component enhance the productivity [16] and quality by via tested software component's reusability[17]. Software quality classification techniques described in [18]. The classification based modeling technique have proven to be better in achieving software quality facilitated the developer most relevant and minimum customize OTS component. The software quality evaluation techniques include CART (classification and regression tool)[19], S-PLUS[20], C4.5 Algorithm[21, 22], Treedisc algorithm[23], Sprint-sliq algorithm, logistic regression, case-based reasoning.

6. Implementation

Now according to our approach we have classified some of the real time software categories. We have extracted some basic functional components from GITHUB[24, 25] the one of most popular open source projects repository, collected from local industry and generated for experiments. The software component were stored in cloud based repository and structure is defined in ontology and data stored using xml format. The concept was implemented and evaluated by designing a system as shown in figure 3.

6.1. Functionalities based Classification technique

In this technique, we have classified the software functionalities present in our software repository into different concerned domains and sub-domains. It behaves like a software tree in which at the first level the core domain of that functionality/feature is present and then at the next level its sub-domains are discovered and further sublevel are identified untill there are no more labels exists. Then on selecting a particular sub-domain, a user will then select their required functionalities from a list of those present in software repository. These nodes of functionalities behave like leaf as in n-ary tree with n number of disjoint categorization.

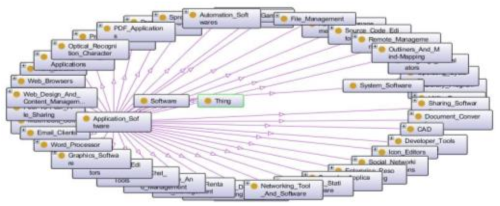


Figure 2: Classification of Software Components in Successive Hierarchy

6.2. Keyword-based technique

In this technique, we have considered a number of different keyword-based search techniques in order to select required functionalities. We applied keyword based search on each node for exploring the number similar function in the same domain. Further, it supports user to exclude the restriction of domain and exploring the repository on keyword based on parent node hierarchy. For optimization of search time the name of child node are assigned a unique key from their names ASCII sum. The collision are handled and assigned codes and position adjacent to colliding nodes.

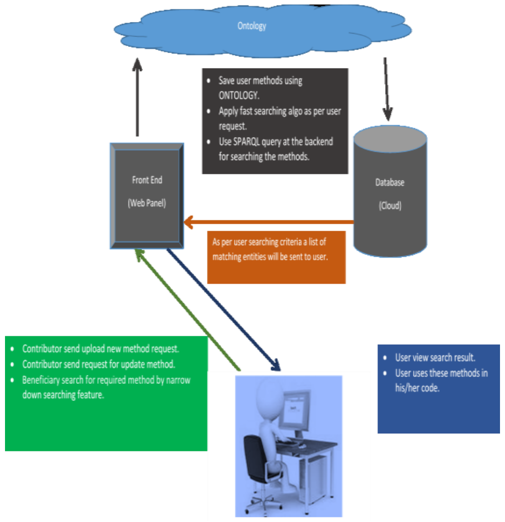

Ontology
Save user methods using ONTOLOGY.
Apply fast searching algo as per user request.
Use SPARQL query at the backend for searching the methods.
Front End (Web Panel)
Database (Cloud)
As per user searching criteria a list of matching entities will be sent to user.
Contributor send upload new method request.
Contributor send request for update method.
Beneficiary search for required method by narrow down searching feature.
User view search result.
User uses these methods in his/her code.

Figure 3: Experimental System Description

6.3. Hybrid technique:

This technique is the combination of the classification-based and keyword-based technique. In this approach, we have divided the softwares in different domains and sub-domains in the same way we have done in the classification-based technique. The element of keyword-based technique comes into play when user selects their required domain and sub-domain and then at that point they specify their required functionalities for that software domain. As all of the above techniques require us to maintain a repository for different functionalities of softwares from different domains, we have also considered in our system using an online software repository such as github[24] for accessing different software functionalities to evaluate the owr coding scheme. In this study, user simply specifies the programming language (such as java, C#) from which to obtain required software functionalities and then names those functionalities, and our system will fetch those software codes from the website based search narrow down by user. The distribution codes in our repository in respect of programming languages are shown in Table 2.

6.4. Semantic modeling approach

Semantic representation and manipulation of huge repository of the software component play essential role in retrieving right information with support of contextual flow to that particular components. We grouped the elements based on semantic resemblance characteristics as shown in figure 4.We developed a software repository to evaluate semantic storage and retrieval of different context software function, but with the same name such as withdraw () is different functionality in banking and university management system. The keyword based search will retrieve all functions with this keyword but by using semantic approach the right function will be reflected to software developed depending on the domain knowledge s/he working on the module, package, project context. The available functionality is provided to use and not existing functionality will be appended to that repository for later use, same or other software organizations.

Table- II: Distribution Programming Language Codes in Our Project

Rank	Languages	%Projects	Rank	Languages	%Projects
1	C++	30%	6	ASP.NET	5%
2	C	20%	7	CSS	5%
3	C#	10%	8	JavaScript	5%
4	PhP	5%	9	Java	10%
5	HTML	5%	10	Html 5	5%

6.5. Unique identification and representation.

Even domain, sub-domain and module distribution leads towards the constant time complexity. To improve more searching efficiency and reducing computation complexity we store the coding in binary.

Furthermore, we will be plugin that will parse the tree further reduce the number collision. To resolve the issues of the same computation but different naming would be resolve using name aliasing feature of the ontology development. Keyword based scheme would be embedded to facilitate exploration functionality of the system.

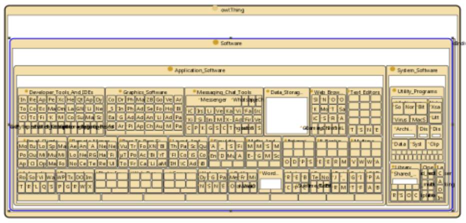
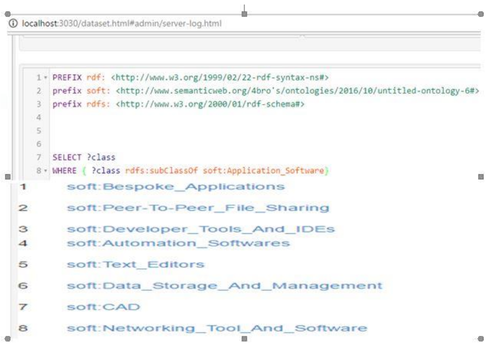


Figure 4: Semantic Periodic Table

7. Evaluation and Results:

We evaluated our software ontology to validate the software structuring hierarchy and information retrievals. The SPARQL[26, 27] is used for querying and semantic consistency validation. We configured apache jena fuseki [28] server on our system to query using SPARQL. We explored the software structure using SPARQL queries and results are shown in Fig.5. We hosted the software ontology and extracted the results from ontology from anywhere using, URL and adding prefix for names spaces for ontology. We can query and node, searching sub-tree form any particular node. We can traverse top down from root node to function leaves and vice versa. First few results of query are shown in Fig.5.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix soft: <http://www.semanticweb.org/4bro's/ontologies/2016/10/untitled-ontology-6#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>



SELECT ?class
WHERE { ?class rdfs:subClassOf soft:Application_Software}
```

```
1    soft:Bespoke_Applications
2    soft:Peer-To-Peer_File_Sharing
3    soft:Developer_Tools_And_IDEs
4    soft:Automation_Softwares
5    soft:Text_Editors
6    soft:Data_Storage_And_Management
7    soft:CAD
8    soft:Networking_Tool_And_Software
```

Figure 5: SPARQL query language for ontology

8. Conclusion

In our research, we have focused on the methodologies that focus more on the reuse of existing software components rather than developing a system right from the scratch that utilize a reasonable amount of efforts and resources. We have proposed a novel approach to classify software based on their basic functionalities.

In our proposed scheme, the software tree is designed in a way that helps in organizing software components in a hierarchical way, which in turn facilitates an efficient reuse of software components. The software tree is designed in such a way that inputting new software to this tree will prompt the process of traversing that software through the hierarchy of domains and sub-domains and finally assigning it to an appropriate node. The aspect of software periodic table in our methodology is that the software in different domains are so divided within the table that on adding some additional functionality/feature within them will promote them to become a new different version of software within a domain or a sub-domain.

9. Future work

The limitations regarding the proposed methodologies involve the collection of large amount of software for the software reuse repository. They feature for searching through different software available online because of their availability in compressed folder/files.

Our research work was an initiative towards the classification of software like periodic table. We will extend this research to design a classification model where all the software can be classified with sustainable structure semantic of SPT. We will use feedback mechanism to stabilize the position of software atom in logically justified position in SPT. Different genetic algorithm can be applied for location optimization of software components [29, 30]. Ant colony optimization [31] would be used to assure the quality of software component. We design and develop programming IDE Add-In that crawls our repository, facilitate developed available feature for reusability of existing component and update repository for custom build function from users

ACKNOWLEDGEMENT

We acknowledge the support of Sukkur IBA University & Muhammad Ali Jinnah University, Karachi for this research. We pay thanks to the CRUC (Center for research in Ubiquitous computing) team that provided us the environment promoting such research activities. We are thankful for github and, industry for sharing the software samples.

REFERENCES:

[1] Hu, J., et al. (2015). Modeling the evolution of development topics using Dynamic Topic Models. in Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on. 2015. IEEE.

[2] Clements, P. and L. Northrop, (2002). Software product lines: practices and patterns.

[3] Linden, F.J., K. Schmid, and E. Rommes, (2007). Software product lines in action: the best industrial practice in product line engineering. Springer Science & Business Media.

[4] Gajala, G. and M. Phanindra. A Firm Retrieval of Software Reusable Component Based On Component Classification.

[5] Rajender Nath, H.K.,(2009). Building Software Reuse Library with Efficient Keyword based Search, International Journal of Computing Science and Communication Technologies, VOL. 2, NO. 1.

[6] Suresh Chand Gupta1, P.A.K.,(2013). Reusable Software Component Retrieval System, International Journal of Application or Innovation in Engineering & Management (IJAIEM), Volume 2, Issue 1.

[7] P. Warintarawej, M.H., M. Lafourcade, A. Laurent , P. Pompidor,(2014). Software Understanding: Automatic Classification of Software Identifiers, Intelligent Data Analysis (IDA Journal) 18(6) (2014) in press.

[8] Prieto-Diaz, R.,(1991). Implementing Faceted Classification for Software Reuse, Software Production Consortium, Herndon, VA.: New York, USA. p. 88-97.

[9] Shireesha P., S.S.V.N.S.,(2010). Building Reusable Software Component For Optimization Check in ABAP Coding, International Journal of Software Engineering & Applications 1.3.

[10] Cimitile, A., A. De Lucia, and M. Munro. (1995). Identifying reusable functions using specification driven program slicing: a case study. in Software Maintenance, 1995. Proceedings., International Conference on. 1995. IEEE.

[11] Prieto-Diaz, R. and P. Freeman, (1987). Classifying software for reusability. IEEE software, 4(1): p. 6.

[12] Lo, D., et al. (2009). Classification of software behaviors for failure detection: a discriminative pattern mining approach. in Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. 2009. ACM.

[13] Zina Houhamdi, S.,(2001). Classifying Software for Reusability, Courrier du Savoir: Algeria.

[14] Greenfield, J. and K. Short. (2003). Software factories: assembling applications with patterns, models, frameworks and tools. in Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2003. ACM.

[15] Greenwood, N.N. and A. Earnshaw, (2012). Chemistry of the Elements. Elsevier.

[16] Case, A.F., (1985). Computer-aided software engineering (CASE): technology for improving software development productivity. ACM SIGMIS Database, 17(1): p. 35-43.

[17] Tahir, M., et al., (2016). Framework for Better Reusability in Component Based Software Engineering. the Journal of Applied Environmental and Biological Sciences (JAEBS), 6: p. 77-81.

[18] Khoshgoftaar, T.M. and N. Seliya, (2004). Comparative assessment of software quality classification techniques: An empirical case study. Empirical Software Engineering, 9(3): p. 229-257.

[19] Steinberg, D. and P. Colla, (2009). CART: classification and regression trees. The top ten algorithms in data mining, 9: p. 179.

[20] Khoshgoftaar, T.M., E.B. Allen, and J. Deng, (2002). Using regression trees to classify fault-prone software modules. Reliability, IEEE Transactions on, 51(4): p. 455-462.

[21] Khoshgoftaar, T.M. and N. Seliya, (2003). Software quality classification modeling using the SPRINT decision tree algorithm. International Journal on Artificial Intelligence Tools, 12(03): p. 207-225.

[22] Quinlan, J.R., (2014). C4. 5: programs for machine learning. Elsevier.

[23] Khoshgoftaar, T.M. and E.B. Allen, (2001). Controlling overfitting in classification-tree models of software quality. Empirical Software Engineering, 6(1): p. 59-79.

[24] Dabbish, L., et al. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. in Proceedings of the

ACM 2012 conference on Computer Supported Cooperative Work. 2012. ACM.

[25] Vasilescu, B., V. Filkov, and A. Serebrenik, (2015). Perceptions of diversity on GitHub: A user survey. CHASE. IEEE.

[26] Harris, S., A. Seaborne, and E. Prud'hommeaux, (2013). SPARQL 1.1 query language. W3C recommendation, 21(10).

[27] Prud, E. and A. Seaborne, (2006). SPARQL query language for RDF.

[28] Bansal, R. and S. Chawla, (2014). An Approach for Semantic Information Retrieval from Ontology in Computer Science Domain. International Journal of Engineering and Advanced Technology (IJEAT), 4(2).

[29] Bouktif, S., H. Sahraoui, and G. Antoniol. (2006). Simulated annealing for improving software quality prediction. in Proceedings of the 8th annual conference on Genetic and evolutionary computation. 2006. ACM.

[30] Washizaki, H. and Y. Fukazawa, (2005). A technique for automatic component extraction from object-oriented programs by refactoring. Science of Computer programming, 56(1): p. 99-116.

[31] Srivastava, P.R. and T.-h. Kim, (2009). Application of genetic algorithm in software testing. International Journal of software Engineering and its Applications, 3(4): p. 87-96.