IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Revue d'Intelligence Artificielle
Vol. 36, No. 1, February, 2022, pp. 79-86
Journal homepage: http://iieta.org/journals/ria

Novel Optimized Reusable Component Repository Using Neural Networks

Krishna Chythanya Nagaraju[1*], Cheruku Ramesh Kumar Reddy[2]

[1] CSE Gokarau Rangaraju Institute of Engineering and Technology-Hyderabad 5000090, Telangana, India
[2] CSE Mahatma Gandhi Institute of Technology, RangaReddy 5000075, Telengana, India

Corresponding Author Email: kcn_be@rediffmail.com

https://doi.org/10.18280/ria.360109

Received: 26 November 2021
Accepted: 15 December 2021

Keywords:
repository, neural networks, reusable component, identification, challenges

ABSTRACT

For the success of the time bound development of software to meet with high demand of market, it is very much necessary for organizations to maintain a proper reusable component repository, either open source or proprietary. This was also endorsed by the survey authors conducted and analyzed data collected as shown in this work. This paper aims to bring out the opinions of actual users like developers in using the Repository. A survey was conducted by the authors of this paper. Basically, survey aims at identifying the experiences of developers using reusable components and trying to identify what developers are expecting from the reusable component repositories and the practices of usage of repositories in the companies. In this paper a Novel Repository building mechanism using Neural Networks is proposed. The normalized features are considered as input to Neural Network model which specifies whether required component exists or not in the repository. The experiment results were found to be giving 94% accuracy in identifying an existing component at the time of retrieval.

1. INTRODUCTION

A reusable component is any independent artifact including code, design, test cases etc. of a software development process, which is designed, developed and tested for its effective use and which can be stored and retrieved from a Reusable component repository. The generic properties of component include abstraction, independency, and modularity**.** For the success of the time bound development of software to meet with high demand of market, it is very much necessary for organizations to maintain a proper reusable component repository, either open source or proprietary. There are a few problems associated with the development environment of software using repositories. This paper aims to bring out the opinions of actual users like developers in using the Repository and also based on findings, a novel repository building using neural networks is tried. The paper aims to strengthen the motive of research of authors on reusable component retrieval from large component repository.

Reusability of anything can be considered as a boon for human kind. People belonging to diversified fields like biologists, chemists use, and reuse the existing standard instruments to record experimental results and also it is unreasonable to expect an electrical engineer to design and develop the transistor from first principles every time one is required, whereas when it is for software engineers, they are looked at as guilty for the same concept of reusing existing software. The reasons for which vary such as a lack of development standards; the not invented here syndrome, poor programming language support for the mechanical act of reuse and poor support in identifying, cataloging and retrieving reuse conditions

In the wake of the above facts, a survey was conducted online by the authors of this paper. Basically, the survey aims at identifying the experiences of developers using reusable components and trying to identify what developers are expecting from the reusable component repositories and the practices of usage of repositories in the companies. The main objective behind this gap identification is, to develop a robust optimized repository which can be helpful in easy retrieval of required reusable code component.

The purpose of this paper is to provide a new way of reuse repository building and applying neural network concept at the time of repository building and retrieval. A neural network model to handle required reusable component identification based on normalized features is implemented and an accuracy of 94% could be achieved. It can be considered as extension to [1] but differs on the way of repository built and applying neural network. The paper is organized as follows: Section 1 deals with the introductory part of topic, Section 2 deals with related previous work in the field and in Section 3 the background work for survey is discussed, questions and respondents' views are discussed in detail in Section 4, whereas Section 5 gives idea and implementation results and Section 6 deals with future scope and conclusion.

2. PREVIOUS WORK

One of the major problems that the software reusers face is the distinct names associated to the same or almost the same operation performing code. Considering the operation of push and insert of stack and queue or insert at begin of linked list, such a problem is referred as Vocabulary problem. Hence completely keyword-based retrieval is not a good solution. One method of implementation could be having a synonym database and going on pattern matching based on synonyms of keyword also.

The Basic Faceted Classification was first proposed by Prieto-Diaz and Freeman who identified six facets namely: function; object; medium; system type; functional area; setting [2, 3]. The software reuser locates software components by specifying facet values that are descriptive of the software desired.

The idea was further extended by Eichman and Atkins by incorporating a lattice as the principal structuring mechanism. The difference in the work done by the both groups was in the way queries can be posed. The Fichman and Atkins methods permit to use as many less facets as the reuser wishes to specify thus avoiding the use of wild cards as happens in the Freeman method i.e. Basic faceted Classification.

Associative retrieval based on neural networks uses relaxations retrieving components based on partial/approximate / best matters. This is sometimes referred to as data fault tolerance. The neuro computing paradigm is characterized by asynchronous, massively parallel, simple computations. Since neural networks are massively parallel, retrieval from large repositories is easily possible using fast associative search techniques.

In general, when we refer the work previously done in this area, we observe neural network applied using SOM i.e. unsupervised learning algorithms [2, 4-7], whereas in case of [8-10] the authors made use of Back Propagation network variants, which emphasized on Reliability Assessment This subsequently gives us enough scope to say that we can apply supervised learning procedures for Reusable software component retrieval from repository [11-13] also. Gibb (2000) highlight two advantages of software reuse.

1. Those components that have been tested provide higher guarantees of robust and reliability in any future implementation.

2. Component reuse should lead to faster development of projects [14-16]. Even with the greater advantages, software reuse has not taken an integrate place in enterprise software engineering policy. There are still many misconceptions related with reuse such as:

i) *Software reuse is a technical problem:* Many developers feel that software reuse [17-19] is only technical issue but don't realize that it is a design practice which makes developed software more reusable. All organizations should inculcate design based on reusability as a culture.

ii) *Artificial Intelligence will solve the reuse problem:* This is another misconception that artificial intelligence can solve the reuse problem. Unless there are existing reusable components, even an artificial agent cannot be trained to handle reusable components.

iii) *Revisions code results in huge increase in productivity:* Not all the code components are exactly matching with the project on hand. Many times adaptation of component to suit with current requirement may be needed. Sometimes time spent in adaptation or finding reusable code component itself may be a penalty on productivity.

iv) *There is a myth that building a repository will motivate people for reusing components:* By mere presence of a repository will not motivate people to use it or reuse code components. The organization should bring in a reuse policy making it mandatory for at least 25% of code to be developed should be a reused one. Also, every member of the organization should be aware of available repository and the mechanism to access components should be familiar for all.

3. BACKGROUND WORK FOR ONLINE SURVEY

This section discusses the strategy applied, questions considered and the rationality behind the question and analyzes the responses given by the respondents of online survey.

3.1 Target people

The survey was basically targeted at developers in corporate industry who might have used repositories like Github, CVS, NVC, or any proprietary reusable component repository in their respective company. Even though we send the survey link to all and sundry, we had a special request for developers who used component repositories to participate actively. In this survey we considered both types of repositories whether Open source or ones' specifically developed and maintained by company for reusable purposes.

3.2 Reaching target people

When it comes for a survey to be conducted, it can be done in many ways as, making people of single organization responding, or making people belonging to multiple organizations to take part or more simply we can conduct an on-line survey to involve respondents from across the globe. The survey authors have chosen is an online survey. The survey was actually created using online free resources and a tiny URL was created for the survey. The tiny URL was shared to targeted group of people who were actually belonging to circle of close friends, relative and known old colleagues of the authors, who are working in software companies with varying experience. The URL was shared to targeted people using Emails, WhatsApp groups and personal messages. The advantage of this survey as compared to one performed previous ones is that it is not confined to one people working in one company or in similar developing environment where the survey may not hold correct. On the promise of anonymity, many of the people who participated in the survey gave their valuable inputs. The authors would like to place on record their earnest thanks to all those respondents who took out their valuable time and spent to answers the questions of survey. Even in survey, as people have a psychological effect when they give their complete details while answering survey questions, the authors avoided questions related to company where the survey respondents are working. As questions of survey also looks in the angle of psychological impact of using reusable repository by the developer or some user. Most respondents to our questionnaire considered their practices to be in accordance with current emerging trends.

3.3 Demographical analysis of respondents

The size and distribution of these respondents varies largely, giving a more in-depth idea on how people across the globe are looking at reusable component repository usages and probably the changes required in this area. Though the number of respondents is far less as compared to number in Ref. [8], we consider survey is qualitative as we were able to reach correct respondents across many countries. There were nearly 40 respondents who participated in this online survey belonging to working places in various countries like USA, U.K., Australia and also from different states of INDIA. As they work in different development environment, the authors hoped for a in depth insight in the inputs shared.

3.4 Categories of research questions

The research questions (RQ) were judiciously prepared to get the maximum required information in minimum time that can be expected from the respondent. There were 20 questions designed and they can be categorized in to following categories , which were identified based on the literature survey done by the authors and thought would be helpful in exploring the experiences of developers or users of reusable component practices in the aspects of, how generally sharing of different artifacts happen in their respective organization, practices of reuse, psychological impact on developer, impact on quality of developed product, security issues and human factors involved, support by the management of organization in implementing reuse practices, barrier for adoption, factors influencing success of products implemented using reusability, Copy right and legal issues, and basically how do they find reusable components. The only questions based on the profile of the respondent were, one to know about his years of experience in development and the other to record the number of projects respondent had worked using reusable repositories, making him more suitable candidate for valid input being shared. Authors have analyzed the most suitable questions and are addressed in this article.

4. SURVEY QUESTIONS AND ANALYSIS OF COLLECTED DATA

Let's take look at the most prominent questions of the survey and the analysis of the data received. Each research question (RQ) is given and followed by the rationality to consider the question is also explained. The following Table 1 summarizes certain RQs(RQ1-RQ6) considered and their responses in % along with options given.

RQ1: Years of experience as a Developer and/or Design/Analyst/Manager?

The question was intended to check the familiarity of the respondents with the availability of repositories and their usage practices in the SDLC process. The more experience they gain they would be rationale enough to speak on the design issues and in the early days of coding many may be dependent on repositories for ease of use.

The survey has shown that, of the respondents nearly 23% of them were having around 15 yrs of experience and around 26% of respondents are having nearly 10 years of experience as in Figure 1 below, hence authors hope for the genuinely information provided by such middle level experienced developers or users of repositories (RQ1 of Table 1).

RQ2: Number of projects for which you used Reusable components at any level as Design, Code, Testing etc. in your career.

It was observed from responses as shown in RQ2 of Table 1 above that around 42% of the respondents were using reusable components in more than 5 projects and it should also be noted that there were only 5% of respondents who never used a repository in their development process. The less percentage of people who never used reusable components clearly indicates raging demand of reusable component repositories and the practices being made common in almost all companies.

RQ3: Does your company policy support using reusable components?

Surprisingly, 85% of the respondents have positively responded, mentioning YES the organization for which they are working does has it as a policy to make use of reusable components, which clearly throws light on the emerging trends on software development based on the shelf components to reduce time of development and improve efficiency of developed product with the help of already tested artifacts (RQ3 of Table 1).

RQ4: Did anyone ever insist you to use a reusable component?

The rationality behind using this question is twofold. One is to know whether it is voluntary from user side to reuse without any not here implemented psychological concept or it is because organization is insisting, they are using, which then does have some psychological negative impact on the user (RQ4 of Table 1).

Table 1. Certain prominent questions asked through survey conducted and recorded responses

RQ1: Years of experience as a Developer and/or Design/Analyst/Manager?						
Options	Fresher	<5	>=5&<10	>=10&<15	Above 15	
Response in %	5	38	26	23	8	
RQ2: Number of projects for which you used Reusable components at any level as Design, Code, Testing etc in your career.						
Options	Never Used	2	3	4	More than 5	
Response in %	5	24	11	18	42	
RQ3: Does your company policy support using reusable components						
Options	YES	NO	DON'T KNOW			
Response in %	85	3	12			
RQ4: Did anyone ever insist you to use a reusable component?						
Options	Never	Sometimes	Often	Advised	Yes, Mandatory	
Response in %	5	29	12	21	13	
RQ5: Do you think the code snippets that are generally developed by you are reusable?						
Options	No	A few	Specifically Designed for Reuse		Don't Know	
Response in %	0	49	38		13	
RQ6: The reason for failing to utilize reusable component, in your experience was						
Options	No attempt to reuse	Component Doesn't exist	Unavailability	Could not be found in time	Component Intangible	Not integrable
Response in %	26	4	15	15	7	9

It was observed, from the data collected that only 2% have said it as Never, whereas 61% mentioned it as sometimes or often and 24% mentioned that either it was advised or made mandatory to use reusable components. When we observe the answers in comparison with answers to RQ3 which queries on company policy to use reusable components, yet the insistent is only 61% as compared to 85% of policy support, which signifies some inhibiting factors which could be psychological or availability of component or maintenance of repository by organization etc. are not allowing it to be made compulsory to use reusable components.

RQ5: Do you think the code snippets that are generally developed by you are reusable?

It is observed that only 38% of the respondents have said that the code snippets that are generally developed by them are reusable. And another 49% of people mentioned that not all but a few components could be reusable (RQ5 of Table 1).

Now this becomes one of the major concerns for the success of Reusability. The authors strongly argue the organizations to look in to this point and implement practices which make people to develop maximum components with features of reusable components such that the Reusability concept is successful.

The list of commonly used repositories by different organizations can be shown as in the Table 2 below.

Table 2. List of commonly used repositories

SVN	CSS	XML CANON	TEST ASSETS
CODE REVIEW	IBM's iRAM	Mighty & Reuse Platform	IOC Container Framework

A Few of the respondents though not working in the same company has mentioned what their company used to have and a few have refrained from mentioning stating the company security reasons.

The following is list of the familiar repositories available in the market, but is not limited to this list. This list emphasizes on the point that many companies are now trying to develop their own repositories rather than using open source repositories which may or may not suite the project requirements aptly.

1) GITHUB 2) SVN 3) TFS-Team Foundation sever 4) Appache POI 5) BITBUCKET 6) Xml canon 7) Flex library 8) Azure library 9) Entities 10) underscore 11) minify 12) Jquery 13) iRAM 14) CODE REVIEW 15) Migration 16) Dynamic File generator 17) Accurev 18) Mapplets in informatica 19) NPM 20) Log4j 21) Microsoft Visual Component manager 22) Laravel components 23) Visual Studio Online (VSO) 24) Subversion 25) Mercurial 26) Perforce 27) AdaInformationClearinghouse 28) Ada Compiler Evaluation System (ACES) 29) Ada and Software Engineering [12] (ASE) CDROM 30) CodeBroker 31) CARDS (Comprehensive Approach to Reusable Defense Software) 32) PAL 33) STARS program 34) Betavine 35) Buddy 36) CloudForge 37) CodePlex 38) Assemblea.

A comparative statement among many repositories based on the experience of respondents is shown in Table 3 below-Y standing for yes. Yet it may not be actual statement of comparison as we have depicted only what respondents mentioned. Some believe this depends on the usage. It won't be good to say which is best. If it works for the context we use, it is good enough.

RQ6: How many times you felt, it is better to code from scratch than to use a on the shelf component, and why so?

Except in case like one respondent responded as "Few times because sometimes your requirements are unique and sometimes because the components are buggy and or poorly documented Depending on use-case aka requirement, there would be a need to look for a proven utility or to write from scratch, almost every one mentioned that it's better to use a reusable component as it saves both time and effort and as it is also tested previously.

RQ7: What do you think is the level of information regarding the availability of specific component existing in the repository being used by you or your friends?

From the set of respondents, around 66% of them mentioned that about 50% of the times the information about availability of required specific component in the repository was easy to know for them and in other cases they had to explore the repository to just check whether it exists or not. This is a prime factor for availability information of component searching and gives strength to our idea of building a fast-searching technique and repository itself being built in an optimized way such that the information is always available for user with in stipulated time of need of hour.

RQ8: Can you suggest or mention 3 qualities of best repository you dream of?

Rationality: This research question aims to know the requirements and expectations of users and what they expect more from the repository designs. The respondents have mentioned quite a few qualities, which are listed below.

➢ Easy Extraction
➢ regular updates
➢ Bug free
➢ documentation and examples
➢ code quality analysis info
➢ better handling multiple clones
➢ Atomic pushes Easy to reference
➢ quicker fixes
➢ Maintenance
➢ Scalability
➢ Continuous deployment
➢ Continuous integration

Table 3. A comparative statement among many repositories based on the experience of respondents

	Github	**SVN**	**Laravel Lumen**	**TFS**	**jquery**	**Accurev**
Friendly	Y	Y	Y	Y	Y	Y
East use	Y	Y	Y		Y	Y
Availability	Y	Y	Y		Y	Y
Plug &play	Y	Y	Y	Y	Y	Y
Help		Y			Y	Y
Illustration					Y	Y

As it is evident from the analyzed data Ease of extraction plays an important role and hence our research is focusing on ease of extracting the required reusable component.

RQ9: The reason for failing to utilize reusable component, in your experience was?

The respondents to the extent of 30% have mentioned the reason for failing to use a component was its unavailability or could not be found in time. This once again emphasis something needs to be done for better performance of repositories presently available in market. Another 33% of participants mentioned that as the component was not integrable they could not use the component. The result analysis can be seen in the pie chart of Figure 1 below. Where the Yellow and Light brown colors represent cases of responses belonging to category of unavailability and could not be found in time respectively. The blue color indicates participants representing who found component which is not integrable. Others include who have not used reusable components or component does not exist or component intangible.

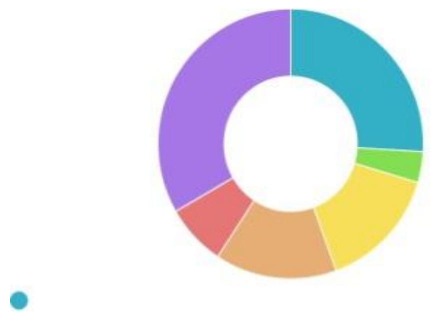


Figure 1. The reason for failing to utilize reusable component, in your experience was

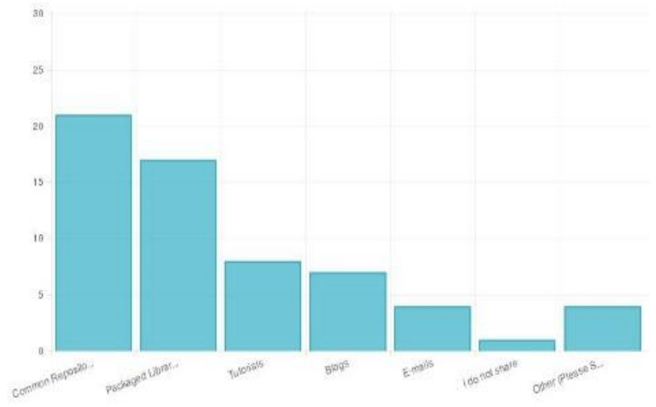


Figure 2. Sharing of artifacts

RQ10: What is the success rate of qualitative projects that made use of reusable components as per your experiences?

It is observed that 45% of people have mentioned that 80% of times their projects in which reusable components are used are successful and delivered the required objective with high quality.

RQ11: How do you share the artifacts of reusable components (Multiple answers can be selected)?

The query was focusing to identify whether repository or any other form of artifact sharing is happening in the companies.

Of the total respondents, 70% inclination was given for common reusable repository as shown in the figure below. The Figure 2 emphasizes the demand of repository practice in the software industry.

5. PROPOSED IDEA AND IMPLEMENTATION

However, to make software reuse operational the software developers need to be provided with large libraries containing reusable software component. The user of such a library must be assisted in locating components that are functionally close to the required component [7].

In the fast-growing technological world with high number of components being developed by a single person, it is no surprise that he himself finds it hard to locate a single piece of code. [9] The problem can be divided into three parts: classifying the original component, describing the component wanted, finding close enough matches between the two. The established solutions for the above mentioned are based on enumerated classification, facetted classification, free text indexing [9]. The Figure 3 below represents the classifications.

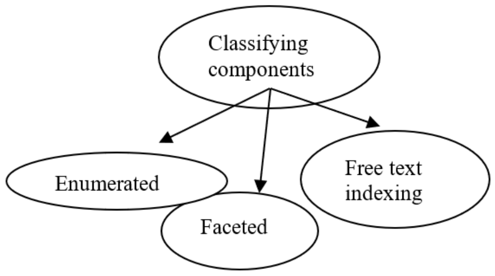


Figure 3. Established solutions of component classification

Neural network has emerged as a promising technology in application that requires generalization, abstraction, adaptation and learning. A neural network approach can serve as an economical and automatic tool, to generate reusability ranking of software components [6]. Inputs to neural network system are to be provided in form of structured attributes of software component as metric values and output is the reusability value category [6].

It may be revolutionary to mention that, we can altogether avoid maintaining any data structure for storing different software components but only for the identifying the location and corresponding unique id of a component we can make use of AVL tree structure.

The idea is to represent the each component that is being stored in form of some 17 features viz. Operating System, Programming Language, Name, Return type, No. of parameters, types of parameters, Recursion , Model of development, already modified, etc. and out of all thee seventeen features only 13 features are considered to be given for Input to the Neural Networks as features like name of component and author name does not exactly contribute in identifying the code component at the time of retrieval.

The input is generally at the time of inserting a valid component in to the repository, or while searching for a required component in the repository already built. The user is supposed to give or select appropriate values of the list of features which suites the required component. The application shall permit user to search for a component directly asking for faceted values, which are supposed to be entered by the user.

By considering each feature to have certain set of possible values say for Operating Systems, it can take for example one of 4 values like Windows or Linux or Unix or IoS, we consider 2 bits to represent this feature and so on. The values are normalized to set of 0,1 s of total 32 bits which would be acting as input vector for Neural Network. The sample data set is shown in Figure 6 below.

The idea is to train the Neural Network to remember the giving normalized input features of a particular component and make it identify when asked for again. In this process we need not store the components in a specific data structure internally but need to maintain only the location wherever they are stored in the system/server so that same can be used to open when the required component is identified by the Neural Networks as existing.

A drop-down list consisting of multiples possible values for each feature is provided in a GUI to the user. The input is read and is normalized. Say for example the possible values of one feature by name "Operating System" are Unix, Linux, Windows, Mac. Here we have 4 possible values and the equivalent normalized values can be 00,01,10,11 respectively. That means when a user selects "Windows" under feature of Operating System", the normalized input would be "10". Likewise normalized inputs for all features are considered and the final string of bits whose length is based on the possible values each feature can take is given as an input to already trained Neural Network model. The trained neural networks model tries to recollect from the training set whether the required reusable component exists in the repository or not, accordingly it gives location of component if it exists else an error message is displayed. The flow chart of proposed idea is shown in Figure 4 below.

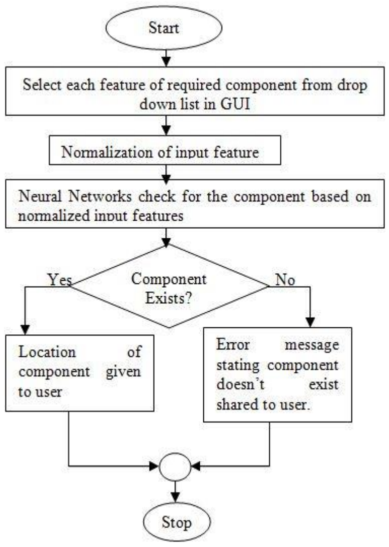

Start
Select each feature of required component from drop down list in GUI
Normalization of input feature
Neural Networks check for the component based on normalized input features
Yes
Component Exists?
No
Location of component given to user
Error message stating component doesn't exist shared to user.
Stop

Figure 4. Flowchart of idea implemented

The algorithm of component retrieval use by the authors can be as shown below:

Component Retrieval Algorithm. (f1,f2,f3…f13)

Input: Selected 13 features of a component being searched for.

Output: 1. Location of component. if found in repository.
2. Error message mentioning component not found.

Begin

Step 1: Read the features from user interface. Say f1..f13.

Step 2: Normalize the inputs: convert each feature in to equivalent bit string based on number of possibilities the value of feature can take to form a 32 bits input.

Step 3: Input the normalized data to already trained Neural Networks Model.

Step 4: Model outputs either location of component if it exists in repository or a message of "component not exists" to the user.

End.

6. IMPLEMENTATION RESULTS

To implement our idea, we made use of Python along with NumPy, pandas, keras, matplotlib libraries and created a Neural Network of 4 layers of which first layer is Input layer having 32 input variables and second and Third layers are the First hidden Layer and Second Hidden Layer with 6 nodes each and then the ultimate layer is an output layer with One output node that has to indicate whether the input features of a component match with already known component or not.

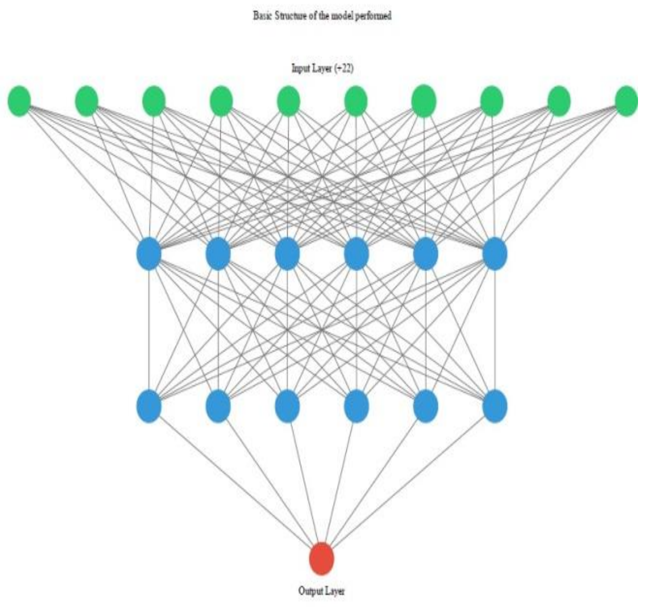


Figure 5. Schematic diagram of neural net

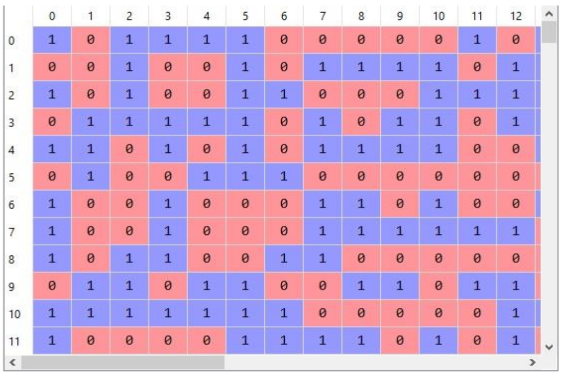
First, we created an input layer with 32 neurons and first hidden layer with 6 neurons. The synaptic weights of first hidden layer are initialized from the standard normal distribution. Then we add a second hidden layer of 6 neurons and even its initial synaptic weights are initialized from standard normal distribution. And finally, we add the output neuron. The activation function used for the neurons in the first 2 hidden layers is rectified linear units (ReLU) and the activation function used for output layer is Sigmoid.

We compiled our model using the optimizer stochastic gradient descent and the metrics to optimize the weights is based on accuracy.

Since we used stochastic gradient descent, took batches of 10 samples each and trained the whole data over 100 epochs. The schematic diagram of Neural Network is as shown in Figure 5 above.

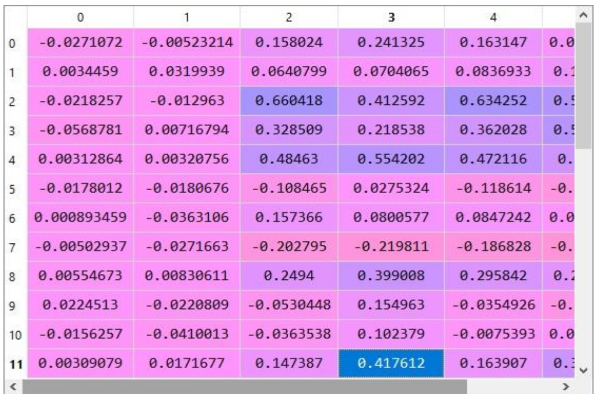
For our experimentation purpose we considered 170 samples of 32 bits each. The Figure 6 shows the feature matrix sample of 170 components.

Finally, a weight matrix of optimized weights is as in Figure 7 is obtained after training.



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	0	1	1	1	1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0	1	1	1	1	0	1
2	1	0	1	0	0	1	1	0	0	0	1	1	1
3	0	1	1	1	1	1	0	1	0	1	1	0	1
4	1	1	0	1	0	1	0	1	1	1	1	0	0
5	0	1	0	0	1	1	1	0	0	0	0	0	0
6	1	0	0	1	0	0	0	1	1	0	1	0	0
7	1	0	0	1	0	0	0	1	1	1	1	1	1
8	1	0	1	1	0	0	1	1	0	0	0	0	0
9	0	1	1	0	1	1	0	0	1	1	0	1	1
10	1	1	1	1	1	1	1	0	0	0	0	0	1
11	1	0	0	0	0	1	1	1	1	0	1	0	1

Figure 6. Feature matrix of 170 samples of 32 bits each representing one reusable component (Owing to space constraint, only a part of matrix shown)



	0	1	2	3	4	
0	-0.0271072	-0.00523214	0.158024	0.241325	0.163147	0.0
1	0.0034459	0.0319939	0.0640799	0.0704065	0.0836933	0.:
2	-0.0218257	-0.012963	0.660418	0.412592	0.634252	0.!
3	-0.0568781	0.00716794	0.328509	0.218538	0.362028	0.!
4	0.00312864	0.00320756	0.48463	0.554202	0.472116	0.
5	-0.0178012	-0.0180676	-0.108465	0.0275324	-0.118614	-0.
6	0.000893459	-0.0363106	0.157366	0.0800577	0.0847242	0.0
7	-0.00502937	-0.0271663	-0.202795	-0.219811	-0.186828	-0.
8	0.00554673	0.00830611	0.2494	0.399008	0.295842	0.:
9	0.0224513	-0.0220809	-0.0530448	0.154963	-0.0354926	-0.
10	-0.0156257	-0.0410013	-0.0363538	0.102379	-0.0075393	0.0
11	0.00309079	0.0171677	0.147387	0.417612	0.163907	0.:

Figure 7. Weight matrix

The accuracy of the system was found to be over 94%. The accuracy graph can be seen as in Figure 8 below.

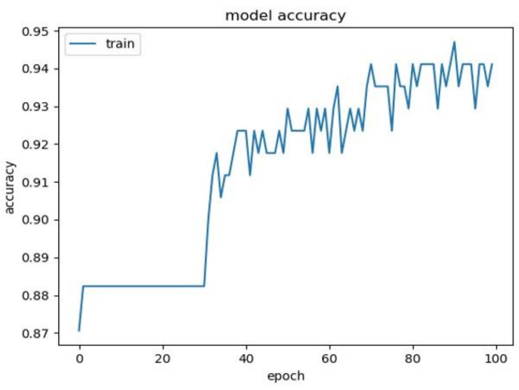


Figure 8. Accuracy graph

So basically, first the 32 neurons in the input layer are fully connected to the hidden layer over random synaptic weights and the neurons in the hidden layer receive information and further it is propagated to second layer and so on till output layer. The weights are then back propagated to the first hidden layer and the initial weights are adjusted accordingly using back propagation algorithm This adjustment of weights took over 100 epochs to converge. The optimizer used to converge the weights is stochastic gradient descent The accuracy increased over time with increase in epochs and converged to over 94% when trying to identify an existing component in the component repository.

7. CONCLUSIONS

In this paper authors have summarized the observations made using online survey done. The gaps identified based on data collected appropriately justified the need of an optimized reusable code component repository to be built. The work differs from other studies on the basis of diversified locations from where survey inputs were taken ranging from different states to continents. Also, survey was majorly responded by richly experienced developers who worked with reusable component repositories. In this work a novel mechanism of implementing reusable component repository containing code components of different platforms is experimented using neural network model. A 13-feature set normalized input for each of 170 reusable components is used as training data set. The whole mechanism saves lot of space in storing detailed information of a component as happens in other works. An accuracy of 94% was observed in this work. The authors could collect responses from only a small number of developers which can be looked at as one limitation of this work. The scaling up of data set is required to check for robustness of system. It is planned to extend this work by observing the operation under different training algorithms for neural networks with varied parameters and to experiment with different architectures to improvise performance measure to achieve accuracy of above 98%.

REFERENCES

[1] Niranjan, P., Rao, D.C.G. (2010). A mock-up tool for software component reuse repository. International Journal of Software Engineering & Applications, 1(2): 1-12. https://doi.org/10.5121/ijsea.2010.1201

[2] Kurfess, F.,Wang, Z., Jololia, L. (2000). Content-based component retrieval. European Conference on Artificial Intelligence ECAI 2000, Berlin, Germany.

[3] Merkl, D. (1995). Content-based software classification by self-organization. In Proceedings of ICNN'95-International Conference on Neural Networks, 2: 1086-1091. https://doi.org/10.1109/ICNN.1995.487573

[4] Tangsripairoj, S., Samadzadeh, M.H. (2005). Organizing and visualizing software repositories using the growing hierarchical self-organizing map. In Proceedings of the 2005 ACM Symposium on Applied Computing, pp. 1539-1545. https://doi.org/10.1145/1066677.1067023

[5] Frakes, W.B., Fox, C.J. (1995). Sixteen questions about software reuse. Communications of the ACM, 38(6): 75-87. https://doi.org/10.1145/203241.203260

[6] Sandhu, P.S., Kaur, H., Singh, A. (2009). Modeling of reusability of object oriented software system. World Academy of Science, Engineering and Technology, 56(32): 162-165. https://doi.org/10.5281/zenodo.1084404

[7] Frakes, W.B., Kang, K. (2005). Software reuse research: Status and future. IEEE Transactions on Software Engineering, 31(7): 529-536. https://doi.org/10.1109/TSE.2005.85

[8] Wang, J.A. (2000). Towards component-based software

engineering. In Proceedings of the Seventh Annual CCSC Midwestern Conference on Small Colleges, pp. 177-189.

[9] Aslund, S. (2003). Software Architecture: Classifying and locating reusable software components. Overview of Techniques.

[10] Gill, N.S. (2003). Reusability issues in component-based development. ACM SIGSOFT Software Engineering Notes, 28(4): 4. https://doi.org/10.1145/882240.882255

[11] Alkazemi, B.Y., Nour, M.K., Sahraoui, A.E.K. (2014). Guidelines for designing reusable software components. International Journal of Computer and Information Technology, 3(6): 1356-1361.

[12] Kalia, A., Sood, S. (2014). Characterization of reusable software components for better reuse. Int. Journal of Research in Engineering and Technology, 3(5): 584-588, https://doi.org/10.15623/ijret.2014.0305107

[13] Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y. (2016). Androzoo: Collecting millions of android apps for the research community. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 468-471.

[14] Bhatia, R.K. (2007). Collaborative Tagging for Software Reuse (Doctoral dissertation). https://www.researchgate.net/publication/266342398_Collaborative_Tagging_for_Software_Reuse.

[15] Constantopoulos, P., Doerr, M., Vassiliou, Y. (1993). Repositories for software reuse: the software information base. In Information System Development Process, pp. 285-307. https://doi.org/10.1016/B978-0-444-81594-1.50022-2

[16] Agresti, W.W. (2011). Software reuse: Developers' experiences and perceptions. Journal of Software Engineering and Applications, 4(1): 48-58. https://doi.org/10.4236/jsea.2011.41006

[17] De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Baresi, L., Becker, B. (2013). Software engineering for self-adaptive systems - A second research road map. In Dagstuhl Seminar. https://doi.org/10.1007/978-3-642-02161-9_1

[18] Prieto-Diaz, R. (1991). Implementing faceted classification for software reuse. Communications of the ACM, 34(5): 88-97. https://doi.org/10.1109/ICSE.1990.63636

[19] Ortu, M., Murgia, A., Destefanis, G., Tourani, P., Tonelli, R., Marchesi, M., Adams, B. (2016). The emotional side of software developers in JIRA. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 480-483. https://doi.org/10.1145/2901739.2903505