© 2018 IJSRST | Volume 4 | Issue 7 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

Retrieval of Best Fit Software Component Using Genetic Algorithm

*[1]Ramu Vankudoth, [2]Dr. P. Shireesha
*[1]Department of Computer Science, Kakatiya University, Warangal, Telangana, India
[2]Department of Computer Science and Engineering, Kakatiya University, Warangal, Telangana, India

ABSTRACT

Software reuse is the use of current software components' engineering expertise or concepts for the development of a new system. Many working products can be reused, including source code, designs, requirements, architecture and documentation. Source code is the most commonly used product for software reuse. Software reuse provides the basis for dramatic quality and reliability enhancements and a lengthy-term decline in software development and maintenance costs. Additional advantages of reuse include increased interoperability and support for rapid prototyping. To enhance the practicality of the reuse of software, you need to know where you are and understand the reusable component. Software reuse repositories with effective representation of software components must be developed. These repositories must allow the developer to very easily locate and retrieve the components as needed. Much of knowledge recovery research and development is designed to improve the efficacy and quality of recovery.

Keywords : Component Classification, Genetic Algorithm, Reusable Components, Reuse

I. INTRODUCTION

Software reuse is the use of engineering knowledge or artifacts from existing software components to build a new system. There are many work products that can be re used, for example source code, designs, specifications, architectures and documentation. The most common reuse product is source code. Software reuse is an important area of software engineering research that promises significant improvements in software productivity and quality. Successful reuse requires having a wide variety of high quality components, proper classification and retrieval mechanisms. Effective software reuse requires that the users of the system have access to appropriate components [9]. The user must access these components accurately and quickly, and if necessary, be able to modify them. Component is a well-defined unit of software that has a published interface and can be used in conjunction with components to form larger unit .Reuse deals with the ability to combine independent software components to form a larger unit of software. To incorporate reusable components into a software system, programmers must be able to find and understand them. Classifying software allows re users to organize collections of components into structures so that they can be searched easily. Most retrieval methods require some kind of classification of the components. The classification system will become outdated with time and new technology. Thus the classification system must be updated from time to time affecting some or all of the components due to

the change and hence it needs a reclassification. This project mainly focuses on implementation of a software tool with a new integrated classification scheme, to classify and build a comprehensive reuse repository. Software reuse is recognized as an effective way of increasing the quality and productivity of software systems. Software reuse greatly reduces the effort, development time and costs. Software reuse is a technique of software engineering, which uses existing software components to develop to develop new system. Component Based Software Engineering (CBSE) is an established area of software engineering. In Section 2 Related Work and Section 3 and 4 proposed work, Result Discussion.

II. LITERATURE SURVEY

Existing Classification Techniques: Previously, four different classification techniques were used to construct a repository reuse [1]

1. Classification of Free Text: The Free Text Recovery quest uses the text in the documents. Usually the recovery method is based on a keyword search [8]. All the document indexes are checked to try to find a suitable keyword entry. The biggest downside of this approach is the uncertainty of the keywords used. Another downside is that I am looking many components that are meaningless. The "grep" utility used by the UNIX manual system is a common example of free text retrieval. This method of classification produces high overheads when the content is indexed and when the time for a query is taken. Each of the documents relating to the item [3] has an index of the related text (usually file headers) and must then be searched from beginning to end when the question is made.

2. Enumerated Classification: The classification enumerated uses a number of mutually exclusive classes which are all in a single dimension hierarchy [6]. The Dewey Decimal framework for classifying books in a library is a prime example of this. Every topic area, e.g. Biology, chemistry etc. have their own code of classification. As a sub code, this is a specialized subject in the main topic. These codes can be sub-coded by the author again. The scheme of classification enables a user to find more than one item within the same section / subsection, supposedly if there is more than one item [4]. For example, there may be more than one book, each written by a different writer, about a specific subject. This form is a one-dimensional classification scheme that will not allow for more than one flexible classification of components. The classification mentioned alone does not therefore provide a good classification scheme for reusable software components.

3. Attribute value classification: The classification scheme for attributes utilizes a number of attributes to classify one component For example: a book contains numerous attributes such as the author, the publisher, a specific ISBN number and the Dewey Decimal system classification code. These are just an example of the attributes. The attributes may be the number of pages, the height, the type of printed face, the date of publication etc., depending on who wishes to provide knowledge about a book. Obviously, the attributes of a book can be: The book can be graded multidimensional in different locations using different attributes. Bulky, All possible attribute variations could reach several tens, which at classification time could not be learned.

4. Faceted Classification: Faceted classification systems are the most important in the reuse of software. Like the attribute classification method, different facets classify components, but typically there are far less facets than possible attributes. Ruben Prieto-Diaz [2 ,8] has suggested a six-facetted system.

* The functional facets are: Function, Objects and Medium.

• The environmental facets are: System type, Functional area and Setting. In Faceted classification components, a number of terms or facets are defined. Facets are identical to the system of the attribute value. With facets, however, the choice of values is minimal.

This removes the issue of uncertainty in evaluating a word or attribute's best value [7]. The permissible values of "Operating Machine" could include DOS, Windows, MVS and OS/2. Search efficiency can be very good with this classification. Frakes and Poles have performed an inquiry into the most advantageous of those classification approaches within the software reuse group quite frequently.

III. PROPOSED APPROACH

Retrieval of knowledge from the archive of components is a repetitive process. The repository size is normally very high. The repository contains a large number and specification of components. Repository is the relation between reuse developments in which the components are manufactured and reused in which components are reused. To reuse the components of the repository effectively, the selection of the correct recovery technique is important. Different linear search-based retrieval techniques for information retrieval are available. We use the attribute classification scheme for classification and retrieval of software components in the proposed system. Each component is stored with the specified attributes in a repository.

We consider those attributes for the classification and retrieval of software components from reuse repositories in the proposed approach. Optimizing the solutions found is one of the main problems in the work being considered. Different soft computing techniques for optimising are available. One of the soft computing techniques that can be used to solve this problem is genetic algorithms. Genetic algorithms are algorithms for search and optimization based on natural genetic mechanics and natural selection. Genetic algorithms are somewhat different from most conventional methods of optimizing [5]. The genetic algorithms achieve the best solution by randomly interchanging information between increasingly fit samples and adding an independent random change probability. Genetic algorithms benefit from the old experience of the parent population to produce new, better performing solutions. Genetic algorithms are iterative methods that each step generates new populations. A new population is produced by means of performance assessment, selection procedures, recombination and survival from an established population. These processes proceed until the population is located and the optimal solution or some other stop condition is achieved. The proposed framework is a reuse repository classification and retrieval of software components.

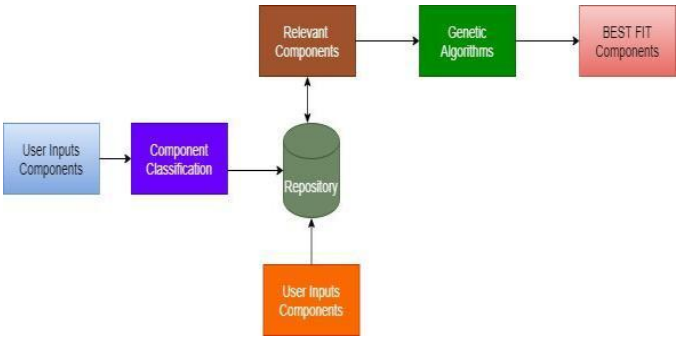

Fig: 2.1 Diagram of the proposed approach

1. Algorithm
2. Gencomp1 Begin
3. Create an initial N population for evaluation.
4. Defines an effective fitness function for individuals.
5. Carrying out genetic operations to generate offspring (crossover and mutation).
6. Analyze the fitness of each individual.
7. Select N superior individuals to form the next generation according to their fitness values.
8. If the end criterion is not fulfilled, go to step 3, stop the algorithm End otherwise.

Description of Genetics: The component's genetic description includes the vector encoding component and weight vectors. Seven attributes are assigned to an item in the repository. In the application of genetic algorithms, we only consider three main component attributes and assign weights based on the significance

of the attribute values. The attribute weights vary from 0 to 1. On the initial population, genetic operators are applied. The initial population is randomly generated. This involves a genetic pool that represents a number of possible solutions. There are three genes or attributes for each chromosome in the population. Each gene has its own weights. After the population is initialised, the objective chromosome role defines how the chromosome is suited for each issue and which chromosomes can survive in the next generation. Genetic algorithms have the objective purpose to be optimised using the genetic method. It is very important to choose an acceptable objective function.

Fitness value is evaluated for each individual in the population as

$$\sum_{i=1}^{10} Ga_i V_i$$

Wj indicates weight vector that matches the vector attribute, where j indicates the total number of attributes of each chromosome, i.e., 3. The value of the objective function must be maximised to achieve the best component. Generally, with each generation, the objective role values increase. Genetic Algorithm Operations: The crossover operation gives offspring to two preferred individuals in the population by sharing some bits between them. The offspring therefore retains certain features of each parent. The mutation operation produces offspring by a random alteration of one or more bits. Offspring may also have different characteristics than their parents. Mutation eliminates local searches of the search area and raises the probability that optimal solution will be identified.

The selection operation selects survival descendants in compliance with pre-defined guidelines. This holds the population size stable and brings a high possibility of good offspring into the next generation. When using genetic algorithms to solve a problem, the first step is to identify a picture that describes the problem states. The most popular way of doing this is with the bit string. An initial population is then established and the next generation is generated by three genetic operations (crossover, mutation and selection). During the entire evolutionary process conventional genetic algorithms use a single crossover operator and a single mutation operator. This procedure is repeated until the criterion of termination is satisfied.

IV. RESULT AND DISCUSSION

Example for dry run of the algorithm: Assuming the input given by user is “Vb.Net” as “Programming Language”. By attribute classification technique the most relevant components are as follows:

1. Linear Search
2. Binary Search
3. Armstrong
4. Fibonacci
5. Palindrome
6. Weighted rank
7. Bubble sort
8. Linear sort
9. Factorial
10. Merge sort
11. Quick sort

These 11 components are encoded and respective weights are assigned as follows:

Components	Component Encoding	Attribute Weight Vector		
		W1	W2	W3
C0	0000	0.1	0.8	0.2

C1	0001	0.2	0.8	0.3
C2	0010	0.4	0.8	0.1
C3	0011	0.1	0.8	0.5
C4	0100	0.8	0.8	0.8
C5	0101	0.8	0.8	0.6
C6	0110	0.6	0.8	0.7
C7	0111	0.5	0.8	0.2
C8	1000	0.7	0.8	0.1
C9	1001	0.5	0.8	0.8
C10	1010	0.2	0.8	0.4

According to the Genetic Algorithm random components C0, C2, C6, C8, C9, C10 are selected for first iteration and steps of the Genetic Algorithm are applied as follows:

Table 2: Iteration 1

Components	Components Encoding	Fitness Function Value(f)	PSelect	Actual count from Roulette Wheel	Mate random selection	Mating pool after reproduction	Crossover Size (randomly selected)	New Population
C0	0000	1.1	0.694	0	C6	0110	2	0101 (C5)
C2	0010	1.3	0.821	0	C9	1001	2	1010 (C10)
C6	0110	2.1	1.326	2	C6	0110	2	0100 (C4)
C8	1000	1.6	1.010	1	C8	1000	2	1010 (C10)
C9	1001	2.0	1.263	2	C9	1001	2	1010 (C10)
C10	1010	1.4	0.884	1	C10	1010	2	1001 (C9)
Total Fitness (∑f) =9.5						Average Fitness (∑f / 6) = 9.5 / 6 =1.583		

Table 3: Iteration 6

Components	Components Encoding	Fitness Function Value(f)	PSelect	Actual count from Roulette Wheel	Mate random selection	Mating pool after reproduction	Crossover Size (randomly selected)	New Population
C4	0100	2.4	1	1	C4	0100	2	0100 (C4)
C4	0100	2.4	1	1	C4	0100	2	0100 (C4)
C4	0100	2.4	1	1	C4	0100	2	0100 (C4)
C4	0100	2.4	1	1	C4	0100	2	0100 (C4)
C4	0100	2.4	1	1	C4	0100	2	0100 (C4)
C4	0100	2.4	1	1	C4	0100	2	0100 (C4)
Total Fitness (∑f) = 14.4						Average Fitness (∑f / 6) = 14.4 / 6 =2.4		

As the average fitness value after successive iterations is converged within a difference of 0.01, so the genetic algorithm process will be stopped. The best fit component obtained is C4 with fitness value of 2.4. Decoding C4 we get "Palindrome" as the "Best Fit Component."

The table 4.1 describes initial population of genetic algorithm process. Initial population is created by choosing components randomly from all retrieved relevant components. In this experiment C0, C1, C4, C6, C9, C12 components are chosen for initial population and are shown in column one.

Components	Components Encoding	Fitness Function Value(f)	PSelect	Actual count from Roulette Wheel	Mate random selection	Mating pool after reproduction	Crossover Size (randomly selected)	New Population
C0	0000	1.1	0.58	0	C4	0100	2	0100 (C4)
C1	0001	1.3	0.62	0	C6	0110	2	0100 (C4)
C4	0100	2.4	1.27	2	C4	0100	2	0110(C6)
C6	0110	2.1	1.11	2	C6	0110	2	0101 (C5)
C9	1001	2.1	1.11	1	C9	1001	2	1010 (C10)
C12	1100	2.3	1.22	1	C12	1100	2	1100 (C12)
Total Fitness (∑f) = 11.3						Average Fitness (∑f / 6) = 11.3 / 6 =1.88		

Table 4(a) Iteration 1 of GA Process

New population generated in the first iteration is taken as input to process second iteration. Here the new population components are C4, C5, C6, C10 and C9. From initial population set C0, C1, C9 components are eliminated and two new components are added (C5 and C10) to the set. Genetic operators are applied and calculated average fitness value is 1.88 shown in table 4(a).

Components	Components Encoding	Fitness Function Value(f)	PSelect	Actual count from Roulette Wheel	Mate random selection	Mating pool after reproduction	Crossover Size (randomly selected)	New Population
C4	0100	2.4	1.12	2	C4	0100	2	0100 (C4)
C4	0100	2.4	1.12	2	C4	0100	2	0101 (C5)
C5	0101	2.2	1.03	1	C5	0101	2	0100 (C4)
C6	0110	2.1	0.98	0	C4	0100	2	0100 (C4)
C10	1010	1.4	0.65	0	C4	0100	2	0100 (C4)
C12	1100	2.3	1.07	1	C12	1100	2	1100(C12)
Total Fitness (∑f) = 12.8						Average Fitness (∑f / 6) = 12.8 / 6 =2.13		

Table 4(b) Iteration 2 of GA Process

New population is generated using genetic algorithm in the second iteration as shown in table 4(b). The new population components are (C4, C5 and C12). This time C6 and C10 are eliminated and C4, C8 are added. The average fitness value in this iteration is 2.13.

Components	Components Encoding	Fitness Function Value(f)	PSelect	Actual count from Roulette Wheel	Mate random selection	Mating pool after reproduction	Crossover Size (randomly selected)	New Population
C4	0100	2.4	1.02	2	C4	0100	2	0100(C4)
C4	0100	2.4	1.02	1	C4	0100	2	0100(C4)
C4	0100	2.4	1.02	1	C4	0100	2	0100(C4)
C4	0100	2.4	1.02	1	C4	0100	2	0100(C4)
C5	0101	2.2	0.93	1	C4	0100	2	0100(C4)
C12	1100	2.3	1.97	0	C4	0100	2	0100(C4)
Total Fitness (∑f) = 14.1						Average Fitness (∑f / 6) = 14.1/ 6 =2.35		

Table 4(c) Iteration 3 of GA Process

Again new population is generated using genetic algorithm in the third iteration as shown in table 4(c). This time C5 and C12 are eliminated. The Genetic Algorithm now converges at C4 component. The average fitness value in this iteration is 2.35.

Components	Components Encoding	Fitness Function Value(f)	PSelect	Actual count from Roulette Wheel	Mate random selection	Mating pool after reproduction	Crossover Size (randomly selected)	New Population
C4	0100	2.4	1	1	C4	0100	2	0100(C4)
C4	0100	2.4	1	1	C4	0100	2	0100(C4)
C4	0100	2.4	1	1	C4	0100	2	0100(C4)
C4	0100	2.4	1	1	C4	0100	2	0100(C4)
C4	0100	2.4	1	1	C4	0100	2	0100(C4)
C4	0100	2.4	1	1	C4	0100	2	0100(C4)
Total Fitness (∑f) = 14.4						Average Fitness (∑f / 6) = 14.4 / 6 =2.4		

Table 4(d) Iteration 4 of GA Process

The population set with only one component C4 (highest fitness value) is taken as input to process iteration 4.In this iteration average fitness value generated is 2.4 as shown in table 4(d)

When the Genetic algorithm process is completed, the output of GA process is the most relevant component. That is the highly fitted component is generated from all relevant software reusable components. In the

above sample experimentation the most relevant component is C4 with the highest fitness value 2.4.
As the average fitness value after successive iterations is converged within a difference of 0.01, therefore the genetic algorithm process will be stopped.

Graphs:

The results obtained in the experiment conducted in the previous section are graphically presented and analysed in this section. The following graph shows the variations in component fitness of all GA process iterations. The fitness variations of the components are reduced from iteration 1 to iteration 6. Fitness values for components range from 1.1 to 2.4. Only one component fitness value 2.4 is available in the last iteration. This is seen below figure. Figure1 shows fitness function value changes for all the iterations. It is evident from the figure that the fitness value increases with iteration.

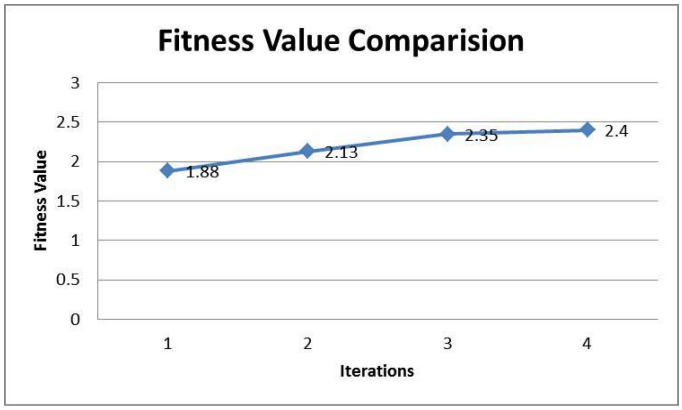

Figure 1 Fitness Value vs Iterations

The Figures 4.2 indicates the weight variations in the first iteration of Genetic algorithm process. In this iteration of Genetic process the output weights (as per fitness fuction) are varied from 1.1 to 2.4. All components are having different fitness values. In the last iteration weight variation is reduced to zero.

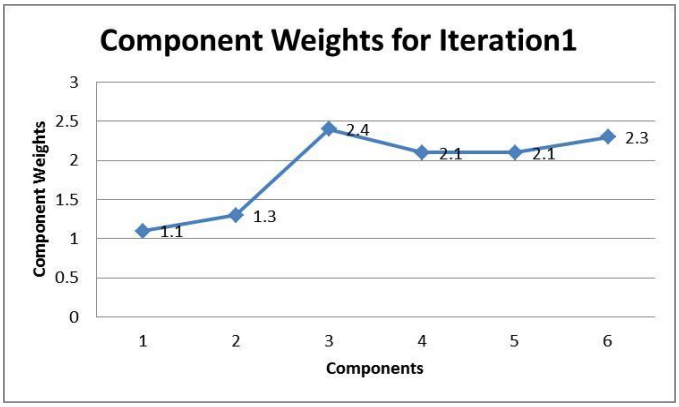

Figure 2 Fitness Value vs Iteration1

The Figures 3 indicates the weight variations in the second iteration of Genetic algorithm process. In this iteration of Genetic process the output are varied from 1.4 to 2.4. In this iteration the components with lowest fitness values (1.1,1.3) are eliminated and new component (2.2) is added to new population.

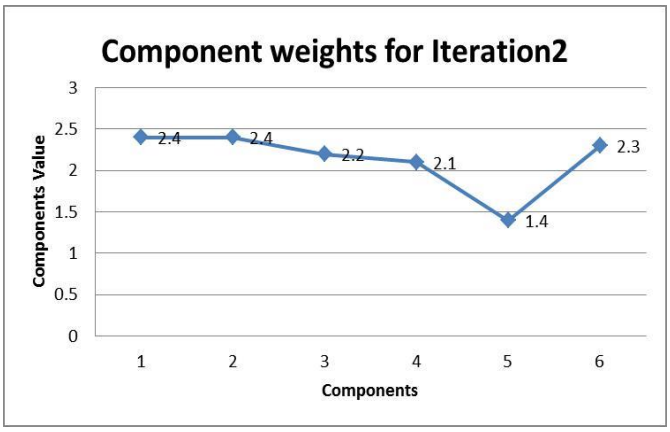

Figure 3 Fitness Value vs Iteration2

The Figures 4 indicates the weight variations in the third iteration of Genetic algorithm process. In this iteration of Genetic process the output weights are varied from 2.2 to 2.4. In this iteration the component with lowest fitness values (2.1 and 1.4) is eliminated and no new components are added to new population.

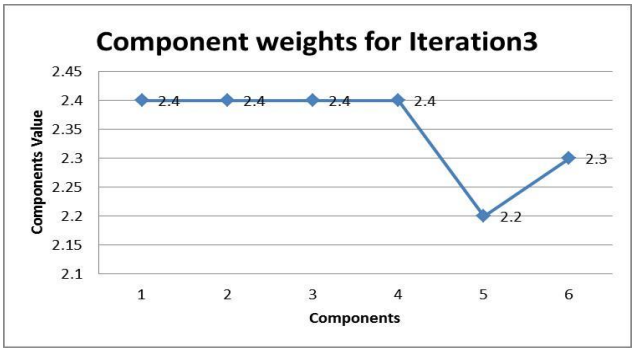

Figure 4 Fitness Value vs Iteration3

The Figures 5 indicates the weight variations in the fourth iteration of Genetic algorithm process. In this iteration of Genetic process the output weights are only with 2.4. In this iteration the component with lowest fitness values(2.2) is eliminated and no new components are added to new population.

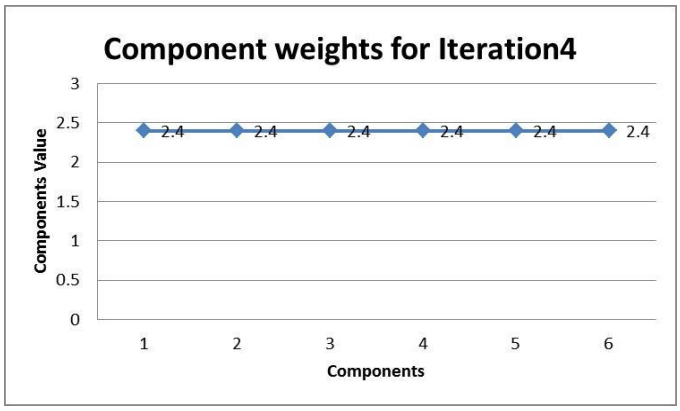

Figure5 Fitness Value vs Iteration4

As the average fitness value after successive iterations is converged within a difference of 0.01, therefore the genetic algorithm process will be stopped. The component C4 with highest fitness value 2.4 is survived in all populations of Genetic process. Therefore the most relevant software reusable component generated C4.

V. CONCLUSION OF PRESENT WORK

Efficient knowledge retrieval from a repository is time-consuming and difficult. The reuse of software is focused on the efficient retrieval of information. In the absence of an appropriate recovery mechanism, the software reuse is significantly reduced.

Today, quality software that focuses on cost reduction is very difficult to build. The reuse of software appears as one of the best solutions for the industries of software development. The process of reuse is the development of new software using existing assets. The key challenge in building any successful repository is the component representation in the repository. The user sets specified component attributes, and these attributes are used for component indexing and retrieval. The success of the reuse software depends on the classification methodology used to build a repository for software reuse. The software engineers and other users in the development of the new software are supported.

The main purpose of the proposed work is to efficiently identify and retrieve software components from the repository. The classification scheme for attributes here is very versatile and simpler to use. In this work Genetic algorithms are presented very effectively to find optimal solutions that select the best fit component from all relevant components. The framework implemented takes into view the mental understanding of the user and classifies reusable components of software along with user knowledge or experience. In this suggested system, users can just define the classification scheme attributes of the repository using their keywords when using the framework. The system is self-learning as more people use the system to increase their vocabulary and ultimately their ability to return components needed to solve problems. In this work, the modern integrated classification system is carried out with extreme precision and precision for the most effective classification of the best reusable components of software.

VI. FUTURE SCOPE

Future work involves identifying multimedia software components and intelligent classification of components for very efficient component selection. To be successful, the scheme should be strengthened in order to meet people's interests. The most surprising result was that developers are more likely to look for personal assets than to check for them. More research is required to clarify the deciding factors. Work may also be done to determine the effect of tightening the reuse environment in the production environment of software developers.

Genetic algorithms may also be combined in order to refine results with other soft computing methods (Neural Networks, Ant Colony Optimization).

VII. REFERENCES

[1]. Ben Khalifa, H., Khayati, O., & Ben Ghezala, H. H. (2008). A behavioral and structural components retrieval technique for software reuse. Proceedings of the 2008 Advanced Software Engineering and Its Applications, ASEA 2008, 134–137. https://doi.org/10.1109/ASEA.2008.45

[2]. Burégio, V. A., Almeida, E. S., Lucrédio, D., & Meira, S. L. (2007). Specification, design and implementation of a reuse repository. Proceedings - International Computer Software and Applications Conference, 1(Compsac), 579–582. https://doi.org/10.1109/COMPSAC.2007.195

[3]. Ezran, M., Morisio, M., & Tully, C. (2002). Reuse Repository. 49–63. https://doi.org/10.1007/978-1-4471-0141-3_3

[4]. He, J., Li, X., & Liu, Z. (2005). Component-based software engineering* the need to linie methods and their theories. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3722 LNCS(May 2014), 70–95.

[5]. Jalender, B., Govardhan, A., & Premchand, P. (2014). A Novel approach for classifying software reusable components for upload and download. Proceedings of 2014 International Conference on Contemporary Computing and Informatics, IC3I 2014, 71–75. https://doi.org/10.1109/IC3I.2014.7019669

[6]. P, N., & C.V, G. R. (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

[7]. Padhy, N., Singh, R. P., & Satapathy, S. C. (2017). Software reusability metrics estimation: Algorithms, models and optimization techniques. Computers and Electrical Engineering, 69, 1–16. https://doi.org/10.1016/j.compeleceng.2017.11.022

[8]. Pole, T. P. (1994). An Empirical Study of Representation Methods for Reusable Software Components. IEEE Transactions on Software Engineering, 20(8), 617–630. https://doi.org/10.1109/32.310671

[9]. Poulin, J. S., & Yglesias, K. P. (1993). Experiences with a Faceted Classification Scheme in a Large Reusable Software Library (RSL).

[10]. Prieto-Díaz, R., & Freeman, P. (1987). Classifying Software for Reusability. IEEE Software, 4(1), 6–16. https://doi.org/10.1109/MS.1987.229789

[11]. Rao, C. V. G. (2011). A Multilevel Representation of Repository for Software Reuse. Journal of Computer Science, 9(9), 114–119.

[12]. Ruben Prieto – Diaz. (n.d.). Implementing faceted Classification for Software Reuse.

[13]. Zozas, I., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Avgeriou, P., & Stamelos, I. (n.d.). Reusability Index: A Measure for Assessing Software Assets Reusability.

Cite this Article

Ramu Vankudoth, Dr. P. Shireesha, "Retrieval of Best Fit Software Component Using Genetic Algorithm", International Journal of Scientific Research in Science and Technology (IJSRST), Online ISSN : 2395-602X, Print ISSN : 2395-6011, Volume 4 Issue 7, pp. 1378-1387, March-April 2018.

Journal URL : http://ijsrst.com/IJSRST2018021