INTELLIGENT RETRIEVAL
Open Journal

Review

Extracting and Organizing Software Elements via Intelligent Classification

Andreas S. Andreou, Dimitrios G. Vogiatzis, George A. Papadopoulos

Department of Computer Science University of Cyprus, 75 Kallipoleos Str, P.O.B. 537 CY-1678 Nicosia, Cyprus

**Corresponding author*
Andreas S. Andreou
Department of Computer Science University of Cyprus, 75 Kallipoleos Str, P.O.B. 537 CY-1678 Nicosia, Cyprus

Article information
Received: October 12th, 2023; **Revised:** January 4th, 2024; **Accepted:** January 17th, 2024; **Published:** February 5th, 2024

Cite this article
Andreou AS, Vogiatzis DG, Papadopoulos GA. Extracting and organizing software elements via intelligent classification. 2024; 2(1).
doi: https://doi.org/10.70705/ppp.ir.2024.v02.i01.pp11-14

ABSTRACT

An innovative approach to intelligent software component categorization and retrieval according to user-defined needs is presented in this study. By partitioning the pool of accessible components kept in a database into specific subsets (clusters), the classification method makes use of a specialized genetic algorithm to develop a limited set of classifiers. As a result, each classifier takes on the role of cluster leader. The user specifies the required qualities (component profile) while tracing a component, and these are compared to the available classifiers' features. The user is shown the "winning" set of components from each cluster in decreasing matching fitness, where the closest classifier matching the necessary criteria above a user-defined threshold is the outcome. We have successfully tested our methods on a simulated dataset of components, and the results are promising. Finally, we showcase the web app that was made to back up the intelligent categorization approach that was suggested.

Keywords
Intelligent software; Database; Retrieval.

INTRODUCTION

Instead of starting from scratch, software projects may be more reliably assembled using pre-existing parts that vendors provide. These parts can be "glued" together using their public interfaces [10].

So, consumers of software components need to look for components on the market, compare their features, and buy the ones that work best with their project's functional needs and limitations. Nevertheless, the present market methods for component finding and retrieval fall short in relation to: a comprehensive range of attributes that enable an effective mapping of user needs into component characteristics, a rapid and robust retrieval mechanism, and the ability to combine diverse component qualities and features for component location.

We present a novel approach to component classification with the goal of providing quick retrieval of the right components based on user preferences. In our proposal, the user may choose their preferred qualities from a preset list, and those traits will then be used to classify components. An intelligent algorithm takes into account the user's choices and returns software components whose attributes match the preferences up to a degree that the user specifies.

This is the remaining structure of the paper: A brief literature review is provided in Section 2. The suggested approach is detailed in Section 3. After introducing the supporting software tool in Section 3, a series of experiments is presented in Section 4. Section 6 concludes the study and provides recommendations for further study.

2. Relevant Work

Organizing a collection of components for quick retrieval is a key difficulty for software component vendors. Recently, component-based software engineering has made use of computational intelligence (CI) technologies including fuzzy sets [2,9], clustering algorithms [8], neural networks [10], and evolutionary computing [6].

Software components are often represented using two different approaches. The first is structured according to a hierarchical system that regulates the vocabulary. A lot of people think this approach is rigid and doesn't classify things well [4]. The second method presupposes a representation that takes values from a set of attributes. We have also included this technique into our present work because of its greater expansibility [4].

Classifying software reusable components using a faceted method was one of the first attempts [3]. Within the framework of the CdCE project, an additional component selection mechanism has been created [7]. The CLARiFi project has now implemented a dynamic categorization scheme [1].

3. An Intelligent Method for Classification and Retrieval of Software Components

Using a web-based software application, our technique intelligently classifies components and quickly retrieves the one you requested. Classification and retrieval operations are built on top of a specialized evolutionary algorithm that analyses a set of specified component properties. What follows is an explanation of our method's fundamental ideas: Data compression: To facilitate the process of genetic algorithm discovery of component classifiers, an encoding method is used to convert the original form of the software component characteristics—be it linguistic phrases, numerical kinds, etc.—into a format that the algorithm can understand. The length of the binary strings (series of 0/1) that we utilized to encode the component characteristics is determined by the sum of the bits required to represent all conceivable values of each individual feature. Discovering Classifiers: In its search for a better classifier, the genetic algorithm compares and contrasts many software components, hoping to find ones that have similar traits. Since the classification method seeks to identify big groupings of components with shared values based on component attributes, the classifying sets may share certain elements. We have a limited number of classifiers (twenty) compared to the typical high number of components. So, instead of looking at the actual components, the search for a component will be done quickly by comparing the user's preferences to the classifiers. Because the choice of a threshold parameter—that is, the number of precisely matched characteristics—specifies the similarity of a component with a classifier—it is also conceivable that certain components are not categorized at all.

The process by which a user seeks for a particular component is known as component retrieval. The first step in developing preferences is deciding what values the component qualities should have. Secondly, the matching threshold value is determined by her or him. It goes without saying that a lower threshold value will result in more components being returned. The next step is for the system to compare the user's request, which is now a bit string, to all of the classifiers. The classifier that returns the most closely matching components is called the "winning" classifier, and it is indicated by the classifier that is closest to the original input. You may find a high-level description of the retrieval procedure in

In Figure 1, the ellipses depict the sets of categorized components. Next to each set of components is the classifier that corresponds to that class. It has already been stated that the classes may overlap. In addition, we can see a collection of unlabeled components that might potentially arise from the threshold.

3.1 Components characteristics and encoding

In the current experimental setting each component is described with a set of 15 characteristics, which were identified by examining a component from different perspectives (functional and non-functional). Our selected characteristics are: General functionality, Specific functionality, Platform, Implementation language, Operating system independence, Synchronisation, Visibility of implementation, Price, Processor utilisation, Memory utilisation, Disk Utilisation, Binding, Data encryption, Data open format compatibility. Each of the aforementioned characteristics is encoded as a string of 0/1. The encoding of a component is essentially the series connection of its characteristics' bit strings. We calculated the required bits for all characteristics and concluded to a string length equal to 60 bits. Figure 2 shows an example of a component with specific characteristics and its relevant bit encoding at the top.

3.2 A Genetic Algorithm for identifying the classifiers

A dedicated GA [5] was developed to evolve candidate classifiers and select the optimal solution in terms of number of components in the corresponding classes, which works in discrete steps as follows:

1. Create a random population of 100 chromosomes - potential classifiers
2. For every generation of the genetic algorithm:

2.1 Apply crossover to every pair of classifiers, where each pair is randomly selected according to a crossover probability

2.2 Apply mutation to a randomly selected classifier according to a mutation probability

3. Perform component classification: for each of the 100 classifiers:

a. Compare each classifier's values of characteristics with those of each component. If a component is close enough (determined by a threshold) to a classifier then assign the component to the class represented by this classifier. Normally, a large number of components will be assigned to each chromosome-classifier

b. Select the top 20 classifiers (chromosomes) in terms of the number of assigned components. Then find the average number of assigned components of the 20 classifiers. This is the average fitness of the current generation

c. If the average fitness of the current generation is greater than that of the previous generation then create a new population by selecting chromosomes according to their fitness and repeat from step 3. Otherwise do not create a new population and repeat from step 2

The above algorithm is repeated until a termination condition is reached. In our case the algorithm terminates if no improvement in the average fitness of the population is observed for 100 generations. A very important parameter is the value of the threshold, which determines whether a component belongs to a certain classifier. For example, a value of 40% means that at least 40% of the values of the classifier characteristics are identical to those of a component. This threshold essentially determines the "success" level of a classifier to gather a rich number of components in his class.

4. Experiments and Results

We looked at component retrieval in the second phase of the studies, after focusing on pooled component categorization in the first. We generated 1000 components at random, with 60 bits apiece, for the categorization step. Each set of results is an average over a hundred separate runs. The fifteen features listed in Section 3.1 provide the basis for the component categorization. Our technique relies heavily on the threshold parameter, which is a measure of similarity.

between the features of the components and those of the classifier.

We used the percentages 30%, 40%, 50%, 60%, 70%, and 80% as our baseline values for the criterion. Classifiers were created with a 30% value, and each of them was able to classify almost all of the software components that were accessible. That means the resulting classifiers aren't able to tell the parts apart. Also, the 80% cutoff was not helpful, as all 20 classifiers only managed to correctly identify components with a degree of complexity between 1 and 3, which is obviously problematic. Similarly, when the threshold was set to 70%, the results were poor.

Table 2 displays the findings for two different threshold values: 40% and 60%. Each classifier's average number of components categorized is shown in the "Average" column, while the number of components that have not been classified is shown in the "Not classified" column. There are no unclassified components and each classifier covers almost half of the components (47.5%), thus the scores for 50% are extremely effective. That is why it is sufficient to look for half of the components during the retrieval phase. Similarly, the 60% is good since there are many unclassified components despite the limited number of components in each class (58.3 on average). Almost every component is categorized by each classifier, making the 40% threshold the worst. On the other hand, this cutoff was also tested during the retrieval phase. In addition, we have tried with 5, 10, and 15 classifiers, but none of them worked well enough.

To test the retrieval phase we created 10 random user requests searching for software components. Then we set the threshold from 40% to 70% at increments of 10% as shown in Table 3. We can observe that the 40% threshold returned a richer number of components, but not all of them were relevant to the users' query as expected. The 50% and 60% values retrieved less but more relevant components. The 70% threshold returned results for some of the queries only. However, the few cases for which we obtained results were highly relevant to the users' requests.

The experiments were conducted with a tool that exists both as standalone system and as a web application. The main functions of the tool are summarized to searching for software components, inserting a new component in the database and some administration functions. The tool differentiates between simple users (reusers) who can only search for components and producers (component developers) who can insert new components. In Figure 3 we can see the characteristics that may be selected when searching for a component.

5. Conclusions and Future Work

We have developed and deployed a smart system for the categorization and retrieval of software components. A limited collection of classifiers that have been trained using a specific genetic method constitute the basis of classification. Each GA-evolved classifier makes an effort to categorize as many software components as feasible based on shared properties. The next step is to compare the developer's needs with the classifiers' requirements in order to get the applicable components. The components that belong to a classifier are returned if the criteria are sufficiently similar to those of the classifier. So, to save a lot of time and effort during retrieval, it is better to compare component specs with the classifiers' instead of all the components'. The evolutionary process of the classifiers also makes use of a threshold, which establishes the minimum percentage of similarity between a component and a classifier needed for component classification. Both the design phase (with GAs) and the retrieval phase (without them) of the classifier are significantly affected by the threshold value. The results showed that a similarity level between 40% and 60% was best. We plan to look at the best course of action when a new component is available in the future. You may either run the genetic algorithm again to rearrange the components or search for the classifier that is the closest match and add it to the appropriate class. It would be helpful to score the components supplied by the system as well. Keep in mind that the system delivers all the components that match to a classifier. A ranking strategy like this might take into account the component's proximity to the classifier or user-defined criteria that give certain attributes more weight than others (for example, memory utilisation is more significant than pricing). Our effort will also take a page out of the work given in [11] by including a semantic interpretation of the user's or developer's requests into our repository.

REFERENCES

4. IST-1999-11631, CLARiFi, http://clarify.eng.it

[2] In the Proceedings of the ACM Symposium on Applied Computing in Tennessee, US, from 1995 (pp. 542–547), E. Damiani and M. G. Fugini discussed "Automatic thesaurus construction supporting fuzzy retrieval of reusable components."

[3] "Implementing Faceted Classification for Software Reuse" by R. P. Diaz, published in 1991 in Communications of the ACM, volume 34, issue 5, pages 88–97.

This sentence is paraphrased from an article published in the IEEE Transactions on Software Engineering archive in 1994. It is titled "An Empirical Study Representation Methods for Reusable Software Components" and was written by W. B. Frakes and T. P. Pole.

5. "Genetic Algorithms" by D. E. Goldberg, published by Addison-Wesley in 1989.

[6] "Software Engineering with Computational Intelligence" by J. Lee, published by Springer in 2003

[7] "Intelligent Component Selection" by V. Maxville, J. Armarego, and C.P. Lam was presented at the 28th Annual International Computer Software and Applications Conference (COMPSAC) in 2004 and covers pages 244–249

[8] "Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification" (International Journal of Computer & Information Science, vol. 5, no. 1, March 2005), S. Nakkrasae, P. Sophatsathit, and W. R. Edwards.

The article "Fuzzy clustering in software reusability" was written by W. Pedrycz and J. Waletzky in 1997 and was published in the Software - Practice and Experience journal.

In their 2002 book "Component Software," C. Szyperski, D. Gruntz, and S. Murer discuss this topic.

The paper "Towards a semantic-based approach for software reusable component classification and retrieval" was presented at the 42nd annual Southeast regional conference in 2004 and can be found on pages 110-115.