ISSN 2250-3153

Searching Technique in Retrieving Software Reusable Components from a Repository

Kudikala Mahesh Babu, Kudikala Sravana Kumari, Pulluri Srinivas Rao

Abstract- Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. These engineering principles are frequently getting changed in order to serve various software development organizations. And all the teams in an organization will follow only one process which is running in their respective company. So the components developed for a software product may be useful for them if they develop similar product in future. Even though the components of a developed product related to one firm, else's teams may require those components. So the components that are identified as re-usable are stored in a repository so that other teams can use them to serve in to get quality product. But to get the re-usable components from a repository we need to search so that we can get our needed component. In this paper, we are introducing a simple searching technique that may effectively retrieve required component from a repository. Here we are trying a web-based search to retrieve the desired component. For this, we are also using test case driven technique in order to meet the required component (means fits our purpose) to be appear in the resulted search based upon its behavior etc.

Index Terms- software reuse, component search, development cost and effort, test cases, web-based search

I. INTRODUCTION

As we know software development is nothing but a process to be followed by the development teams in order to develop the required product that can have good quality. Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines [1]. These engineering principles are nothing but the road map that is to be followed by the development teams of an organization. There are different methodologies like waterfall, spiral model etc and also a specific methodology developed for a specific organization. Every software development company always tries to minimize the effort to develop a product and tries to make the product a high quality one. The components to be developed for a particular product can be costly in terms of person-month, LOC (Lines of Codes) etc. To develop these components indirectly result into more effort, high cost and sometimes less reliable. The software development teams develop the required product. There are various development processes available for them. Sometimes, organizations follow their own defined process for the development of a product. To develop a product or even a small component, it may require great effort, more time and high cost based on metrics like Source Lines of Code (SLOC) [2]. So the people of an organization always look up for the component which can be reusable. The software reusable component is nothing but a component developed in a product and used in the development of other new product. The software reuse is meant to reduce cost, effort to develop the new product and also increase the quality of the newly developing product [3].

A. Software Reuse

Software Reuse is the process of using existing work products instead of building them from scratch [4]. Reuse is the use of work products (such as code, design and test) which are the products or by products of the software development process, without modification in the development of other software. It includes multiple reuse programs in different division within the same company so that it has been largely positive. It maintains the result into products can be used multiple times. And also, the reusability increases the productivity because it does not require much works for consumers (developers) [5].

Reuse impacts on software development in two distinct ways:

1. Development with reuse
2. Development for reuse

The first aims to develop products by reusing existing components. The second aims to address the issue of how to create components that are potentially reusable [6]. The existing reuse approaches include composition (Library-based reusable components) and generative [7]. The reusable components must be preserved at some place so that whenever we need them, we can just search for them and use them. Tracz [8] has pointed out that the industries have established component reuse in the form of a reusable library consisting of procedures, functions and objects.

B. Component Reuse Techniques

Khayati and Girauddin summarize the main techniques that are currently used to retrieve components from software repositories [9].

1. Natural Language Processing (NLP)
2. Signature Matching
3. Behavioral Retrieval

The NLP for component retrieval usually suffers from low recall (percentage of relevant results retrieved) for searches that are too specific or low precision (percentage of correctly retrieved results) for searches that are too general. Signature Matching tries to match the types of an operation's formal parameters but is fairly ineffective on its own because of its low precision. Behavioral retrieval uses randomly chosen "samples" to execute all operations in software library with a signature that matches the required one. Since the developer has to calculate the expected result by hand, the approach clearly has some similarity to what we call black-box testing today.

C. The Web-based Search Prototype

The procedure is having six steps:

a) Define syntactic signature of desired component

b) Define semantics of desired component in terms of test cases
c) Search for candidate components with Google using a search term derived from (a).
d) Find the source units which have the exact signature defined in (a)
e) Filter out components which are not valid (i.e., compilable)
f) Establish which components are semantically acceptable by applying the tests defined in (b).

1. Component Description

The first step in the component reuse process is to describe the services that the desired component is required to deliver. Most development processes involve the creation of a system design which contains a description of the system's components in a UML class diagram such as the one depicted in the figure 1. Since our approach uses regular software development artifacts that are created in most projects we believe that it will be easy to integrate with almost all mainstream development processes.

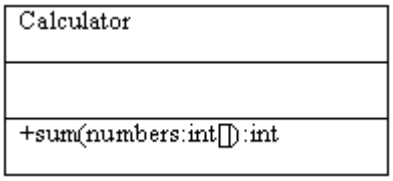

Figure 1: UML description of an example component

a) Syntactic Description:

A diagram such as that in figure 1 contains exactly the required syntactic description of the signatures of the operations supported by the component. In principle, any language has such syntactic description and would be acceptable, but in this paper we focus on Java since it is the widest spread programming language in Google search. Figure 1 shows a very good example of syntactic description of a very simple component to add together a sequence of integer numbers. The only operation required for this component is a method which receives an array of integers and returns an integer containing their sum. Most common UML Tools such as Rational Rose or Eclipse are able to generate Java code framework of the following kind from such a description.

```
 public class Calculator
 {
public int sum(int[] numbers)
   {
   }
}
```

b) Semantic Description:

The description of the semantics is much more challenging. In fact, creating a complete and consistent description of the semantics of an operation is often as difficult as coding the implementation itself. The most common way to write operation semantics is to use pre- and post-conditions. The pre-condition defines what must be true for the operation to succeed, and the post-condition defines what effects the operation must have if it is invoked with pre-condition satisfied. The approach which is most widely used today to define and check whether an operation meets its expectations is testing. A set of test cases only describe the semantics of an operation if it provides full coverage of the input and output space- something which is impossible in practice. However, in the absence of a mature technology for checking pre- and post-conditions, an incomplete set of test cases is a more useful description of semantics than nothing. Podgurski and Pierce showed that in general a maximum of twelve "samples" (i.e., test cases) is enough to discover operations purely with signature matching and random behavior sampling [10]. It seems that three tests are usually enough to discover an adequate component on the web. Table below illustrates some simple test cases that we used to check whether an implementation of the sum operation from previous section is satisfactory.

Table 1: Test cases for the sum operation

Input	Expected output
1 + 2 + 3	6
1 + 0 + -1	0
-1 + -2 + 1	-2

2. Candidate Discovery

In searching a component from a repository by using Google like search engine, there is an already specified technique by [11]. In this, augmenting the queries with some additional term can focus them on special topics and enhance their effectiveness. For example, when we are looking for Java components, the term "filetype:java" in conjunction with (in place of) "class" (as every Java file must contain the keyword class) improves the results significantly. So in this paper, we are enhancing this technique a little bit as we can add few more key terms such as "file-size: <int>", "use-count: <no. of repetitions>". This can improve the search as the user can only obtain his required component. The "file-size" term is a very crucial term as the reusable component must be as less as possible to faster the software or application in which it is used. There are also other advantages of having less-sized component in our application such as the application can be developed within no time, its size is also less, its fastness also increased and most importantly the cost and effort to develop the application drastically decreased.The syntax of the existing search technique is as follows:

Filetype: java –intitle: cus class "int sum int"

The syntax of our proposed search technique is as follows:

Filetype: java –intitle:cus class "int sum int" File-size: int use-count: "no. of rep" In our syntax, the attribute values of: File-size is the size of the reusable component, use-count is the number of times the component is searched and used. Its values usually like very frequent, frequent, rare and very rare, depending on a variable associated for it in the search program (such as, count). This variable value depends on the count variable in the repository.

After the search has made, the next step is to download the java classes returned by the search and process them further to detect those that:

i. Contain exactly the right method signature
ii. Are compilable Java sources

The (i) is achieved by taking the desired declaration as well as the downloaded source files in to consideration and comparing them with each other. Once a match is made, the body

of the retrieved method is automatically copied and pasted into a generated source file with the requested method header. The (ii) can simply be achieved by simply passing the generated source file through a Java compiler.

3. Candidate Evaluation:

a) Semantic Matching

After getting syntactically matching candidates or components, they need to be checked for semantic compatibility. As explained before, this is done by establishing whether or not the candidates satisfy the test cases defined in previous section. Here we use JUnit style test cases to our generated code, and if all of the test cases are passed, we assume that a correct version of the desired operation has been found.

b) Evaluation

The information retrieval normally evaluates search results by calculating recall and precision [12]. Here it is not possible since we easily reach a recall of 100% simply by looking at each of Google's search results. So the time required to retrieve a working method in our prototype implementation is usually under one minute. For example, the technique similar to this is used in [10] depicts an example. Consider a search for standard sorting components that sorts an array of integers from the smallest to the largest. Table 2 provides an overview of some method signatures whose implementations were easily found with that prototype:

Table 2: Example Prototypes of Different Program Components

Method	Signature
Addition	Int sum(int [])
Faculty	Long fac(int)
Body Mass Index	Float bmi(float, float)
Generic Sorting I	Int[] sort(String [])
Generic Sorting II	String[] sort(String [])
Quicksort	Int[] quicksort(int [])
Web-download	String download(String)
File-Reading	String loadTextFile(String)

This table demonstrates that "stateless" operations like mathematical functions or sorting algorithms can be discovered with relative ease when using straight-forward method names, parameter types and order.

II. CONCLUSION AND FUTURE WORK

In this paper, we are trying to load reusable components in a repository. We are established techniques to search a component from that repository effectively and speedily. But we are not concentrated on how to store reusable components in the repository. This can be done based on number of times a component is searched so far, so that we can have schemas in the repository for each and every component based on their frequent visit value. For example, we can have a schema for each of the previously defined values. And whenever the counter value is changed on a component, it will shift from one schema to other. Thus, we can reduce the search time since there is no need to search the entire repository, rather we can search only on the schema specified by the key term "use-count" [13]. And also we can make use of sorting algorithms to store reusable components in a repository. We are also not concentrated on the reusable objects of a program. This is the discovery of "objects" which possesses multiple, interdependent operations and attributes that describe internal states of a product. This is a challenging problem than the specified one.

ACKNOWLEDGEMENT

We extremely thank our Principal and the management for their continuous support in Research and Development. We are also very grateful to our faculty members for their valuable suggestions and their ever ending support. Especially, we thank our college Principal and management for their financial support for receiving the sponsorship.

REFERENCES

[1] "Software Engineering: A Practitioner's Approach" by Roger S. Pressman, McGraw-Hill International Edition.

[2] Albrecht, A.J., and J.E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation", IEEE Trans. Software Engineering, November 1983.

[3] Wayne C. Lim, "Managing Software Reuse", Prentice-Hall, 1995.

[4] "Software Reuse and its impacts on Productivity, Quality and Time-to-maket", Eunjung Lee, Department of Computer Science, University of Houston.

[5] W.C.Lim, "Effect of Reuse on Quality, Productivity and Economics", IEEE Software, Sept.1994.

[6] Sommervelle. I and Ramachandran. M 1991: Reuse Asssessment, First International Workshop on Software Reuse, Dortmund, Germany.

[7] Biggerstaff, T.J. and Perlis, A.J. (Eds) 1989: Software Reusability, Vol 1 & 2, ACM Press, Addison-Wesley.

[8] CAMP 1987: Common Ada Missile Packages, Final report, Airforce Armament Lab, Florida.

[9] Khayati,O., Girauddin, J: "Component Retrieval Systems", Reuse in Object-Oriented Information Systems Design, OOIS workshop, 2002.

[10] Oliver Hummel & Colin Atkinson: "Extreme Harvestin: Test Driven Discovery and Reuse of Software Components", IEEE,2004.

[11] Baumann. S., Hummel. O.: "Using cultural Metadata for Artist Recommendations", Proceedings of the International Conference on Web Delivering of Music (Wedelmusic), Leeds, 2003.

[12] Baeza-Yates R., Ribeiro-Neto B., Modern Information Retrieval, Addison Wesley, 2002.

[13] Aho, Alfred V. and Jeffrey D. Ullman [1983], "Data Structures and Algorithms", Addison-Wesley, Reading, Massachusetts.

AUTHORS

First Author – Kudikala Mahesh Babu, M.Tech, Kakatiya Institute of Technology and Science, India, Email: - kudikalamahesh.mtech@gmail.com

Second Author – Kudikala Sravana Kumari, M.Tech, Jayamukhi Institute of Technological Sciences, India, Email: - kudilakalasravana@gmail.com

Third Author – Pulluri Srinivas Rao,M.Tech(Ph.D) Jayamukhi Institute of Technological Sciences, India, Email: - srimarao@yahoo.co.in