A MOCK- UP TOOL FOR SOFTWARE COMPONENT REUSE REPOSITORY

P.Niranjan
Asst. Professor and Head
npolala@yahoo.co.in

Dr. C.V.Guru Rao
principal
guru_cv_rao@hotmail.com

Department of Computer Science and Engineering
Kakatiya Institute of Technology and Science
Warangal, Andhra Pradesh, India – 506 015

Abstract

Software Reuse effectiveness can be improved by reducing cost and investment. Software reuse costs can be reduced when reusable components are easy to locate, adapt and integrate into new efficient applications. Reuse is the key paradigm for increasing software quality in the software development. This paper focuses on the implementation of software tool with a new integrated classification scheme to make classification build of software components and effective software reuse repositories to facilitate retrieval of software components depending upon user requirements.

Keywords: Software Re-use, Component, Classification Techniques, Re-use Library.

1. Introduction

Software reuse is the use of engineering knowledge or artifacts from existing software components to build a new system [11]. There are many work products that can be reused, for example source code, designs, specifications, architectures and documentation. The most common reuse product is source code. Four different classification techniques had been previously employed to construct reuse repository, namely, Free Text, Enumerated, Attribute Value, and Faceted classifications.

The biggest problem of software reusability in many organizations is the inability to locate and retrieve existing software components. To overcome this impediment, a necessary step is the ability to organize and catalog collections of software components, to quickly search a collection to identify candidates for potential reuse [2, 16] which would also become an aid to the software developer. Software reuse is an important area of software engineering research that promises significant improvements in software productivity and quality [4]. 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. 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 [3].

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 paper mainly focuses on implementation of a software tool with a new integrated classification scheme, to classify and build a comprehensive reuse repository.

This paper is organized into four sections. Section 2 illustrates existing classification methods. The proposed system is described in Section 3. Section 4 deals with experimentation on the proposed system and algorithms in detail. The results of the system when implemented are analyzed with examples in section 5.the section concludes the research work and projects future trends followed by bibliography.

2. Survey

2.1 Free text classification

Free text retrieval performs searches using the text contained within documents. The retrieval system is typically based upon a keyword search [16]. All of the document indexes are searched to try to find an appropriate entry for the required keyword. The major drawback with this method is the ambiguous nature of the keywords used. Another disadvantage is that a search my result in many irrelevant components. A typical example of free text retrieval is the 'grep' utility used by the UNIX manual system. This type of classification generates large overheads in the time taken to index the material, and the time taken to make a query. All the relevant text (usually file headers) in each of the documents relating to the components are index, which must then be searched from beginning to end when a query is made.

2.2 Enumerated classification

Enumerated classification uses a set of mutually exclusive classes, which are all within a hierarchy of a single dimension [6]. A prime illustration of this is the Dewey Decimal system used to classify books in a library. Each subject area, e.g. Biology, Chemistry etc, has its own classifying code. As a sub code of this is a specialist subject area within the main subject. These codes can again be sub coded by author. This classification method has advantages and disadvantages pivoted around the concepts of a unique classification for each item. The classification scheme will allow a user to find more than one item that is classified within the same section / subsection assuming that if more than one exists. For example, there may be more than one book concerning a given subject, each written by a different author.

This type of classification schemes is one dimensional, and will not allow flexible classification of components into more than one place. As such, enumerated classification by itself does not provide a good classification scheme for reusable software components.

2. 3 Attribute value

The attribute value classification scheme uses a set of attributes to classify a component [6]. For example, a book has many attributes such as the author, the publisher, a unique ISBN number and classification code in the Dewey Decimal system. These are

only example of the possible attributes. Depending upon who wants information about a book, the attributes could be concerned with the number of pages, the size of the paper used, the type of print face, the publishing date, etc. Clearly, the attributes relating to a book can be:

- Multidimensional. The book can be classified in different places using different attributes.
- Bulky. All possible variations of attributes could run into many tens, which may not be known at the time of classification.

2.4 Faceted

Faceted classification schemes are attracting the most attention within the software reuse community. Like the attribute classification method, various facets classify components however there are usually a lot fewer facets than there are potential attributes. Ruben Prieto-Diaz [2, 8, 12, 17] has proposed a faceted scheme that uses six facets.

- The functional facets are: Function, Objects and Medium.
- The environmental facets are: System type, Functional area, Setting.

Each of the facets has to have values assigned at the time the component is classified. The individual components can then be uniquely identified by a tuple, for example.
< add, arrays, buffer, database manager, billing, book store >

Clearly, it can be sent that each facet is ordered within the system. The facets furthest to the left of the tuple have the highest significance, whilst those to the right have a lower significance to the intended component. When a query is made for a suitable component, the query will consist of a tuple similar to the classification one, although certain fields may be omitted if desired.

The most appropriate component can be selected from those returned since the more of the facets from the left that match the original query, the better the match will be.

Frakes and Pole conducted an investigation as to the most favorable of the above classification methods [9]. The investigation found no statistical evidence of any differences between the four different classification schemes, however, the following about each classification method was noted:

- Enumerated classification
  Fastest method, difficult to expand
- Faceted classification
  Easily expandable, most flexible
- Free text classification
  Ambiguous, indexing costs
- Attribute value classification
  Slowest method, no ordering,

3. Proposed System

Existing software components for reuse can be directly classified in the classification scheme into one among the above specified classifications presented in the

previous section and stored in the reuse repository. Sometimes they need to be adapted according to the requirements. As classification scheme relies on one of the techniques discussed in the previous section which shall inherently affect the classification efficiency. New designs of software components for reuse are also subject to classified to classification scheme before storing them in the reuse repository. User will retrieve his desired component with required attributes from reuse repositories. The architecture of proposed system is shown in the figure1.

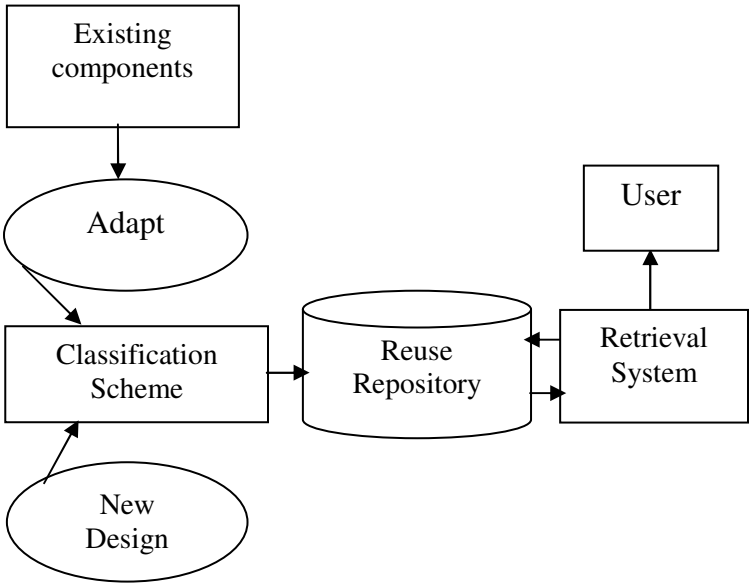

Fig. 1. Proposed System Architecture

An integrated classification scheme, which employs a combination of one or more classification techniques, is proposed and likely to enhance the classification efficiency. The proposal is described in the following sub section. This had given rise to development of a software tool to classify a software component and build reuse repository.

3.1 Integrated Classification Scheme

Integrated classification scheme which combines the attribute value and faceted classification schemes to classify components with the following attributes.

- Operating system
- Language, Function
- Inputs
- Outputs
- Domain
- Version

The attributes when used in query can narrow down the search space to be used while retrieval.

The proposed software tool will provide an user friendly interface for browsing, retrieving and inserting components. Two algorithms are proposed for searching and inserting components as part of this software tool.

Algorithm 1:
Component Insert(Component facet and attributes)
Purpose: This algorithm inserts a component into the reuse repository with integrated classification scheme attributes.
Input: Component facet and attributes
Output: Component insertion is success or failure.
Variables: rrp: reuse repository array, rp: repository pointer,
flag : boolean

```
begin
/* checking for same component is exist in repository */
 while (i <= rp)
  begin
     if((rrp[i].language <> lan ) and  rrp[i].function <> fun ) and (rrp[i].domain <> dom )
     and (rrp[i].os <> os ) and (rrp[i].ip <> ip ) and  (rrp[i].op <> op ) and (rrp[i].ver <>
     ver ))
           i++;
     else
          flag = true;
           break;
    endif
 endwhile
 if (flag)
       rrp[rp].language = lan;
       rrp[rp].function = fun;
       rrp[rp].os = os;
       rrp[rp].domain = dom;
       rrp[rp].ip = ip;
       rrp[rp].op = op;
       rrp[rp].ver = ver;
       return success;
     else
      Component is already exists;
   endif
 end.
```

The insert algorithm stores the newly designed or adapted existing component into the reuse repository. When component attributes are compared with existing repository component attributes and determines no similar components are found then component is inserted successfully otherwise component not inserted in repository and exits giving message that component already exists.

Algorithm 2:
Search_Component(Component facet and attributes)
Purpose: This algorithm searches for relevant components with given component facet and attributes from reuse repository.
Input : Component facet and Component attributes.
Output: list of relevant components
Variables: rrp : reuse repository array
rp: repository pointer
table: result array
i.j : internal variables
flag: boolean

```
begin
  if (component facet <> null )
  for ( i=1; i <= rp ; i++ )
    if ((rrp[i].language = lan ) and
       (rrp[i].function = fun ))
      table[j].language = rrp[j].language
      table[j].function = rrp[j].function
      table[j].os = rrp[j].os
      table[j].ip = rrp[j].ip
      table[j].op = rrp[j].op
        j++;
     else
      flag = 0;
     endif
 endfor
 endif
if (component facet <> null ) and (any of the other attributes <> null )
 for ( i =1;i <= rp ;i++ )
  if (( rrp[i].language = lan) and
          (rrp[i].function = fun))
   if ((rrp[i].os = os ) or (rrp[i].ip = ip) or (rrp[i].op=op) or rrp[i].domain=dom)
or(rrp[i].ver = ver))
      table[j].language = rrp[i].language;
      table[j].function = rrp[i].function;
      table[j].os = rrp[i].os;
      table[j].domain = rrp[i].domain;
      table[j].ip = rrp[i].ip;
      table[j].op = rrp[i].op;
      table[j].ver = rrp[i].ver;
    endif
   endif
  endfor
 endif
 if ( !flag )
```

no component is matched with given attributes
endif
end.

The search algorithm accepts component facet and attribute values from user and retrieves relevant components from reuse repository.

The proposed software tool is developed by implementing the following modules.

1. User Interface

The user must be able to insert and search the components in the reuse repository. A user friendly interface is designed to select relevant attributes in above stated.

2. Query Formation

The user when desirous of searching a component may enter some keywords. He may also select some list of attributes from the interface. The query formation module should accept all the keywords entered and form the query using those keywords.

3. Query Execution

When user sends a query to retrieve component by query execution on all the components which satisfy the criteria that is specified by user in advanced search of user interface.

4. Formatting Results and Presentation

The results obtained in the previous module are formatted so that the user can clearly understand the functionality of component before choosing one. All the results are taken and are displayed along with their details. Now the user can select his choice of component to download or save a component in the location specified by the user.

4 Experimentation

. This tool provides the options to store or retrieve components from repository. The following test cases describe integrated classification components tool when executed on the alogorithms explained in previous section.

Sample test cases:

Case 1. Inserting a component.

Component-id : 009
Operating system: Windows
Language , Function: Java , Sorting
Input : Data items
Output : Sorted data items
Domain : Educational
Version : 2.0
Result: Component is successfully inserted.

In above test case given component attributes are captured and compared with repository components. The search algorithm does not find a matching component in

reuse repository with given attributes, so current component inserted in to the repository successfully.

Case 2. Inserting a component

Component-id : 018
Operating system: Windows
Language , Function: Java , Sorting
Input : Data items
Output : Sorted data items
Domain : Educational
Version : 2.0

Result: This software Component is already exists in the reuse repository.

In above test case given component attributes are captured and compared with repository components. The search algorithm finds a matching component in the reuse repository with given attributes, so current component not inserted in to the repository and displays a message that component is already exists.

Case 3. . Retrieving a software component from the reuse repository

Component-id : -
Operating system: -
Language , Function: Java , Sorting
Input : -
Output : -
Domain : -
Version : -

Result:

Comp-Id	version	
003	3.0	Download
018	2.0	Download
020	1.0	Download

In this test case language and function attributes are captured and compared with software components available in reuse repository. The algorithm found three relevant software components in the reuse repository. The results are displayed with full details of software components retrieved from reuse repository.

.Case 4. . Retrieving a software component from reuse repository.

Component-id : -
Operating system: Unix
Language , Function: Java , -

Input : -
Output : -
Domain : -
Version : -

Result: Full specifications of software component are not passed. Software component retrieval is failure

In above test case total facet attributes are not given only language attribute is given. The search algorithm displays a message that function facet is not mentioned.

5 Results

The search performance is evaluated with different test results and compared with existing schemes.
Search effectiveness refers to how well a given method supports finding relevant items in given database. This may be number of relevant items retrieved over the total number of items retrieved. The following box-plots in figure 2 shows search performance.

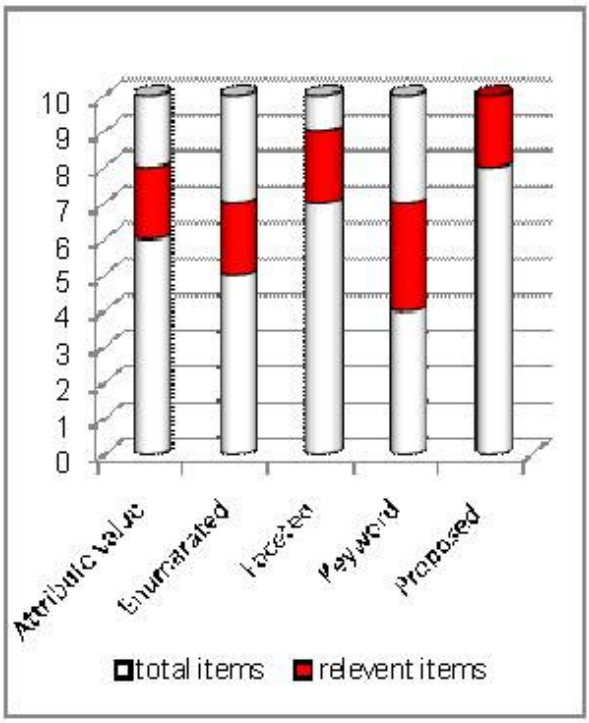

Figure 2. Finding Relevant Components

Existing classification schemes and integrated classification scheme on the horizontal axis for the number of Data items as mentioned on the vertical axis. Total data items

retrieved are shown with white color and colored area indicates the percentage of relevant items among all the retrieved data items.

Faceted classification scheme marked highest performance of search among all the existing classification schemes. Keyword classification scheme registered the lowest performance. Where as our proposed integrated classification scheme out performed to retrieve more relevant items in comparison to all those existing schemes.

Search time is the length of time spent by a user to search for a software component. The following box-plots in Fig.3 gives search time consumed by the existing classification schemes and Integrated classification scheme.

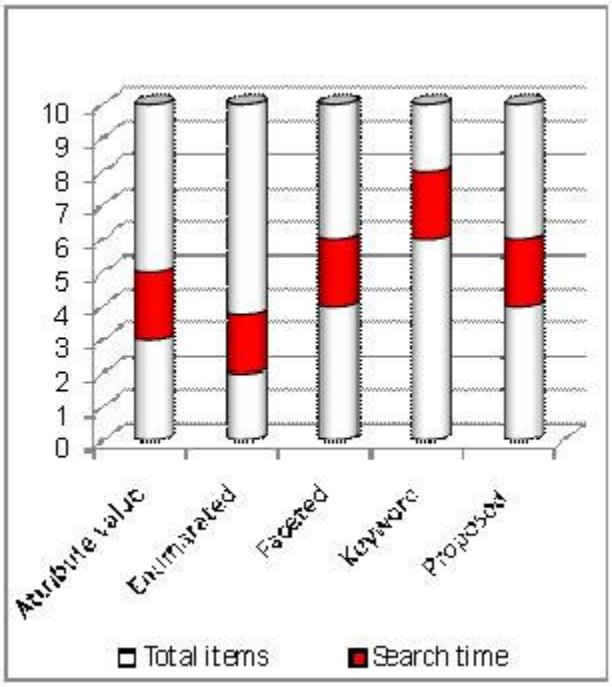

Figure 3. Search Time of Components

On the horizontal axis and the search time consumed on the vertical axis. Total data items retrieved are shown with white color and colored area indicates the search time to retrieve those data items.

The keyword classification is the slowest and the fastest method is enumerated classification. Even though the integrated classification scheme till be consume more time to search but data items fetched are more relevant making the search effective.

5 Conclusion and Future Work

An effective software tool with user friendly interface is designed and successfully implemented with integrated classification scheme which restricts search space and reduces search time increasing the efficiency of classification of software component.

Future work involved with this classification scheme will be to refine the scheme for Multi-Tired or Multimedia presentation of components.

References

[1] S.Henninger, "An Evolutionary Approach to Constructing Effective Software Reuse Repositories", ACM Transactions on Software Engineering Methodology, no 2, 1997, pp. 111-150

[2]Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", Communication of the ACM, Vol. 34, No.5, May 1991

[3] Gerald Kotonya, Ian Sommerville and Steve Hall, "Towards A Classification Model for Component Based Software Engineering Research", Proceeding of the of the 29th EUROMICRO Conference © 2003 IEEE

[4] William B. Frakes and Thomas. P.Pole, "An Empirical Study of Representation Methods for Reusable Software Components", IEEE Transactions on Software Engineering vol.20, no.8, Aug. 1994, pp.617-630.

[5] Lars Sivert Sorumgard Guttorm Sindre and Frode Stokke, "Experiences from Application of a Faceted Classification Scheme" © 1993 IEEE, pp 116-124.

[6] Jeffrey S. Poulin and Kathryn P.Yglesias "Experiences with a faceted Classification Scheme in a Large Reusable Software Library (RSL)", In The Seventh Annual International Computer Software and Applications Conference (COMPSAC'93), 1993, pp.90-99

[7] Vicente Ferreira de Lucena Jr., "Facet-Based Classification Scheme for Industrial Automation Software Components"

[8] Ruben Prieto-Diaz, "Implementing Faceted Classification for Software Reuse" © 1990 IEEE, pp.300-304

[9] Klement J. Fellner and Klaus Turowski, "Classification Framework for Business Components", Proceedings of the 33rd Hawaii International conference on system Sciences- 2000, 0-7695-0493-0/00 © 2000 IEEE

[10] Vitharana, Fatemeh, Jain, "Knowledge based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis", IEEE Transactions on Software Engineering, vol 29, no. 7, pp, 649-664.

[11] William B.Frakes and Kyo Knag, "Software Reuse Research: Status and Future", IEEE transactions on Software Engineering, VOL.31 NO.7, JULY 2005

[12] R.Prieto-Diaz and P.Freeman, "Classifying Software for Reuse", IEEE Software, 1987, Vol.4, No.1, pp.6-16.

[13] Rym Mili, Ali Mili, and Roland T.Mittermeir, "Storing and Retrieving Software Components a Refinement Based System", IEEE Transactions of Software Engineering, 1997, Vol.23, No.7, pp. 445-460

[14] Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Hamid Mcheick, "Another nail to the coffin of faceted controlled vocabulary component classification and retrieval", Proceedings of the 1997 symposium on software reusability (SSR'97), May 1997, Boston USA, pp.89-98.

[15] Hafedh Mili, Fatma Mili, and Ali Mili, "Reusing Software: Issues and Research Directions", IEEE Transactions on Software Engineering, Vol.21 No.6, June 1995.

[16] Gerald Jones and Ruben Prieto-Diaz, "Building and Managing Software Libraries", © 1998 IEEE, pp.228-236.

[17] Prieto-Diaz, Freeman, "Classifying Software for Reuse", IEEE Software, vol.4, mo.1, pp.6-16, 1997

[18] Nancy G. Leveson, Kathryn Anne Weis, "Making Embedded Software Reuse Practical and Safe "12 th ACM SIGSOFT, October, 2004.

[19] William B. Frakes and Kyo Kang, "Software Reuse Research Status and Future" IEEE transactions on Software Engineering, Vol. 31, No.7, July 2005.