INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS.....

A SURVEY ON VARIOUS PARAMETERS OF COMPONENT SEARCH AND RETRIEVAL

Palak Wadhwa[1], ManishaGahlot[2]

[1,2]Department of Computer Science & Engineering
South Point Institute of Technology & Management
DCRUST, Murthal India
[1]palakwadhwa02@gmail.com

Abstract: *Component-based Software Engineering (CBSE) is concerned with the development of systems from software components, the development of components, and system maintenance and improvement by means of component replacement or customization. Building systems from components and building components for different systems requires established methodologies and processes not only in relation to development/maintenance phases, but also to the entire component and system lifecycle including organizational, marketing, legal, and other aspects. In addition to objectives such as component specification, composition, and component technology development that are specific to CBSE, there are a number of software engineering disciplines and processes that require methodologies be specialized for application in component-based development. Many of these methodologies are not yet established in practice, some have not yet been developed. Experiences from other areas, such as system engineering can be successfully applied on component based development, as there are many similarities in the basic concepts (for example, relations between systems and components). Also, with its focus on components and their specifications, CBSE can give better understanding of building systems in general, and in particular of computer-based systems whose significant part is software. The progress of software and system development in the near future will depend very much on the successful establishment of CBSE; this is recognized by both industry and academia. In this paper, we have surveyed various techniques of component based development and proposed a new technique based on UML diagrams.*

1. INTRODUCTION

A Component retrieval method [1, 2] can be described from three aspects: component representation, component query (user's requirements) specification, and component retrieval process. In this free-text-based retrieval method, components are represented as free-text-based documents; while a component query is described using keywords. The retrieval process is to look up the keywords in all component description documents. The components with most matched keywords will be selected. Vector space and indexing technology are used to facilitate documents organizing and matching. This method has low scores on both precision and recall. Researchers and practitioners have proposed to use general thesaurus to extend keywords, by including their synonyms and antonyms, to get more relevant component. In addition, general domain knowledge is also used to extend initial keywords to get more semantically relevant components [3]. However, both of these two improvements increase retrieval recall at the cost of retrieval precision.

2. SOFTWARE COMPONENT SEARCH AND RETRIEVAL

2.1 High recall and precision–This is the basic requirement, which must be considered in any search mechanism. High precision means that most retrieved elements are relevant. High recall means that few relevant elements are left behind, without being retrieved. Several techniques may be used to achieve this. To achieve a high recall ratio, most prefer to use a "relaxed" search, where not only exact matches are retrieved, but also related elements [4,5].

2.2 Security - Historically, security has been a minor issue in component search and retrieval. Since in most approaches reuse takes place in-house only, little effort is needed in order to avoid unauthorized access. However, in a globally distributed component market, where people from the entire world have access to these systems, security must be considered a major issue, since there is a wider range of possibilities for an unauthorized person to break security. Certification and authentication techniques, already used in distributed systems, are good candidates to solve this problem.

2.3 Query formulation –There is a natural information loss when the (re) user is formulating a query. As pointed out by, there is also the conceptual gap between the problem and the solution, since usually components are described in terms of functionality (how), and queries are formulated in terms of the problem (what). A search mechanism must therefore provide means to help the (re)user to formulate the queries, which will consequently reduce this conceptual gap [6].

2.4 Component description – During a search, the (re) user specifies a query, which is then "matched" against a series of component descriptions. The search mechanism is then responsible for deciding which components are relevant to the (re) user. There are two approaches of doing this: The first one is to use formalism. By formally describing a component's functionality and requiring the

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS…..

(re) user to formulate queries in the same formal language, it is possible to automatically check whether or not the component satisfies the (re) user's requirements. The other approach is to "roughly" describe components (using facets, or aspects, for example), and retrieve a set of components that "probably" satisfies the requirements. Then these components are presented to the (re) user, who inspects them and decides which one(s) to retrieve [7].

2.5 Repository familiarity - Reuse occurs more frequently with well-known components. Therefore, a search mechanism should help the user to explore and get familiar with the components, so that in future searches, it is easier to locate them. In a component market, this is even more important, since it allows (re) users to be aware of different components, and different options for a single type of component, facilitating his choices and stimulating the competition among vendors.

There are several ways of achieving this goal. Increasing recall through "relaxed" search is one of them. Henninger's idea of reformulating queries and evolving the repository also help. But the technique that seems more appropriate to a component market is the hypertext-based browsing. Its successful story in the Internet indicates its potential in allowing users to rapidly browse through different pages (components), in different web sites (repositories), acquiring knowledge while browsing.

2.6 Interoperability – In a scenario involving distributed repositories, it is inevitable to think about interoperability. A search mechanism that operates in such a scenario should be based on standard technologies, in order to facilitate its future expansion and integration with other systems. Due to the natural heterogeneity of this world-wide distributed scenario, misinterpretation is a constant risk. The simpler way to avoid this is to use standard ways of packaging information [2-6].

2.7 Performance – Performance is usually measured in terms of the response time. In centralized systems, the involved variables are the hardware processing power and the search algorithm complexity. In distinguished scenario, other variables must be considered, such as network traffic, geographical distance and, of course, the great number of available components [8-9].

Figure 1.a and figure 1.b shows the basic steps for designing a software component reuse system.

3. RELATED WORK

A Component retrieval method [1-2] can be described from three aspects: component representation, component query (user's requirements) specification, and component retrieval process. In this free-text-based retrieval method, components are represented as free-text-based documents, while a component query is described using keywords. The retrieval process is to look up the keywords in all component description documents. The components with most matched keywords will be selected. Vector space and indexing technology are used to facilitate documents organizing and matching. This method has low scores on both precision and recall. Researchers and practitioners have proposed to use general thesaurus to extend keywords, by including their synonyms and antonyms, to get more relevant component. In addition, general domain knowledge is also used to extend initial keywords to get more semantically relevant components. However, both of these two improvements increase retrieval recall at the cost of retrieval precision. High recall and precision is the basic requirements are relevant. High recall means that few relevant elements are left behind, without being retrieved.

Several techniques may be used to achieve this. To achieve a high recall ratio, most prefer to use a "relaxed" search, where not only exact matches are retrieved, but also related elements. Component-based development has been used successfully [3] in applications in any engineering and business domains such as desktop environments, graphing packages and mathematical applications and it is increasingly present in new domains, such as such as in real-time, safety-critical, mission critical, or, more generally, dependable systems. The communities from these domains show increasing interest in addressing their research and practical problems by applying component-based approach. We also see a trend combination of different approaches, such as component-based software engineering and aspect-oriented programming, component-based development and service-oriented development [3].

The Object Constraint Language (OCL) was introduced [4] as part of the Unified Modeling Language (UML). Its main purpose is to make UML models more precise and unambiguous by providing a constraint language describing constraints that the UML diagrams alone do not convey, including class invariants, operation contracts, and state chart guard conditions. There is an ongoing debate regarding the usefulness of using OCL in UML-based development, questioning whether the additional effort and formality is worth the benefit. It is argued that natural language may be sufficient, and using OCL may not bring any tangible benefits. This debate is in fact similar to the discussion about the effectiveness of formal methods in software engineering, but in a much more specific context.

This paper presents the results of two controlled experiments that investigate the impact of using OCL on three software engineering activities using UML analysis models: detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes.

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY
WINGS TO YOUR THOUGHTS.....

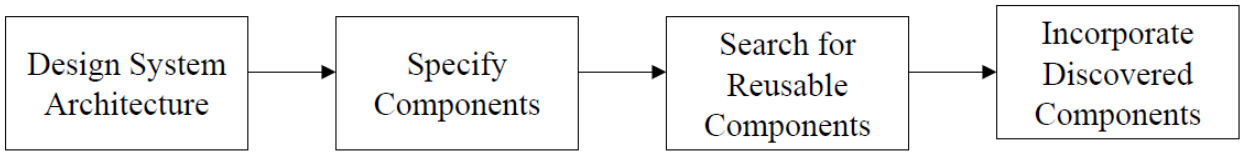

Figure 1(a): Steps for designing a software component reuse system

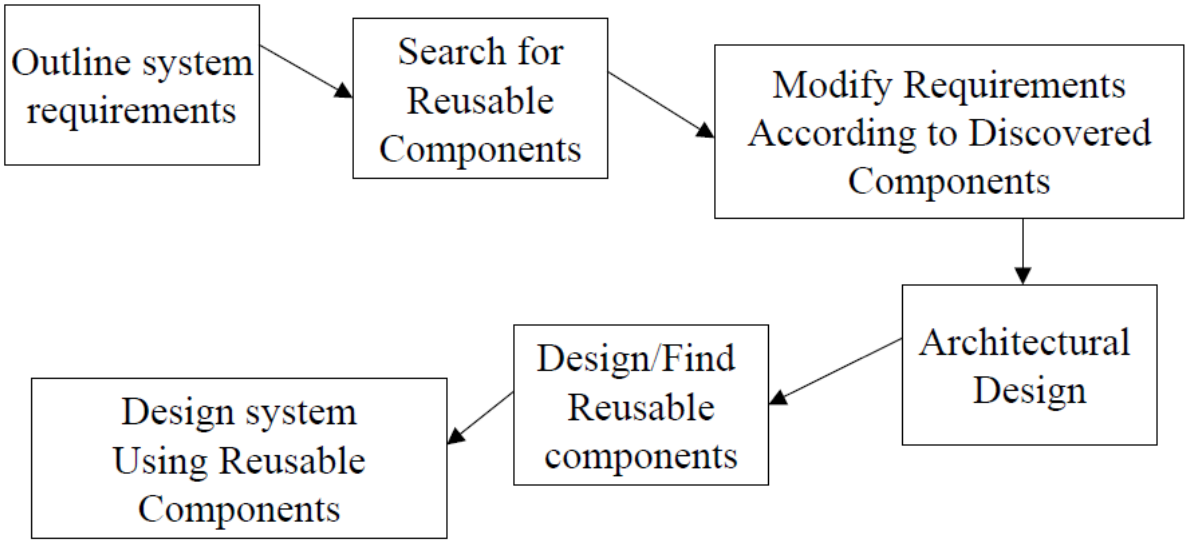

Figure 1(b): Steps for designing a software component reuse system

The results show that, once past an initial learning curve, significant benefits can be obtained by using OCL in combination with UML analysis diagrams to form a precise UML analysis model. But, this result is however conditioned on providing substantial, thorough training to the experiment participants [4].

Historically, security has been a minor issue in component search and retrieval. Since in most approaches reuse takes place in-house only, little effort is needed in order to avoid unauthorized access. However, in a globally distributed component market, where people from the entire world have access to these systems, security must be considered a major issue, since there is a wider range of possibilities for an unauthorized person to break security. Certification and authentication techniques, already used in distributed systems, are good candidates to solve this problem [5]. There is a natural information loss when the (re) user is formulating a query.

A search mechanism must therefore provide means to help the (re) user to formulate the queries, which will consequently reduce this conceptual gap [6].

During a search, the (re) user specifies a query, which is then "matched" against a series of component descriptions. The search mechanism is then responsible for deciding which components are relevant to the (re) user.

There are two approaches of doing this: The first one is to use formalism. By formally describing a component's functionality and requiring the (re) user to formulate queries in the same formal language, it is possible to automatically check whether or not the component satisfies the (re) user's requirements. The other approach is to "roughly" describe components (using facets, or aspects, for example), and retrieve a set of components that "probably" satisfies the requirements. Then these components are presented to the (re) user, who inspects them and decides which one(s) to retrieve [7].

Reuse occurs more frequently with well-known components. Therefore, a search mechanism should help the user to explore and get familiar with the components, so that in future searches, it is easier to locate them. In a component market, this is even more important, since it allows (re) users to be aware of different components, and different options for a single type of component, facilitating his choices and stimulating the competition among vendors. There are several ways of achieving this goal. Increasing recall through "relaxed" search is one of them. Henninger's idea of reformulating queries and evolving the repository also help. But the technique that seems more appropriate to a component market is the hypertext-based browsing. Its successful story in the

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS…..

Internet indicates its potential in allowing users to rapidly browse through different pages (components), in different web sites (repositories), acquiring knowledge while browsing [8]. In a scenario involving distributed repositories, it is inevitable to think about interoperability. A search mechanism that operates in such a scenario should be based on standard technologies, in order to facilitate its future expansion and integration with other systems. Due to the natural heterogeneity of this world-wide distributed scenario, misinterpretation is a constant risk. The simpler way to avoid this is to use standard ways of packaging information [8]. Performance is usually measured in terms of the response time. In centralized systems, the involved variables are the hardware processing power and the search algorithm complexity. In distinguished scenario, other variables must be considered, such as network traffic, geographical distance and, of course, the great number of available components [9].

4. GAPS IN EXISTING TECHNIQUES

Through the analysis of the four current major approaches of software classification and retrieval , we think that the behavior based approaches and the denotation semantics approaches are not suitable for classifying and retrieving object-based components , they are more appropriate for functional-based component retrieval; the formal specification- based methods are difficult to implement; the advantage techniques of the descriptive classification methods are suitable for object-based component retrieval. The major drawbacks of the traditional descriptive classification schemes for software component retrieval are:

i. There is Ambiguity problem in keyword based search; substantial disagreement over the choice of keywords can occur when different words mean different things to different people.
ii. Effort to narrow down the search results is quite high.
iii. Precision and Recall is not high.
iv. They are based on a controlled vocabulary that must be constructed manually for each application domain.
v. Both classification and retrieval require important human effort because users must select appropriate terms for each facet in the classification scheme from usually list of terms in the controlled vocabulary.

This paper proposes an approach combining main advantages of descriptive classification methods for component retrieval in order to improve retrieval effectiveness and provide a friendlier user interface through the use of queries in MDL format.

5. PROPOSED METHODOLOGY

Step 1: To model UML Diagrams

To model UML diagrams some software for modeling is required. In this work, we can use software's such as Visual Paradigm or Rational Rose, which is the product of IBM Corporation. In Rational Rose/ Visual Paradigm all the diagrams of the UML can be modeled and stored in a single file of MDL file format. Hence various sample cases are taken and are modeled in Rational Rose.

Step 2: Storage of UML Diagrams and Java Code in Database

The repository should be indexed and classified according to the projects. Here, our repository includes two assets, namely source codes and designs. Designs are the diagrams, which are modeled in Rational Rose and stored in the Repository. Source codes consist of programs in java. Designs and Source Codes about a project are stored in the Repository.

Step 3: Development of Search Engine

The search engine should search upon the queries built to match the relevant components. The search input consists of MDL file. The output can be designs or source codes, as required by the user. The resultant should not only be most relevant matched items, but also some relevant matched items for browsing. Component Retrieval Search Engine is developed in C#. Net or VB.Net and MS Access. The repository can be changed or integrated with any other standard databases available like SQL Server or Oracle. For the reporting purposes the data reports of VB.Net are used. The input provided to search engine is modeled in Rational Rose to produce MDL file.

Step 4: Input to the Search Engine

The input provided to search engine is modeled in Rational Rose to produce MDL file. The search engine should search upon the queries built to match the relevant components. The search input consists of MDL file. The output can be designs or source codes, as required by the user. The resultant should not only be most relevant matched items, but also some relevant matched items for browsing.

Step 5: Retrieving the Results

The search engine should search upon the queries built to match the relevant components. The search input consists of MDL file. The output can be designs or source codes, as required by the user. The resultant should not only be most relevant matched items, but also some relevant matched items for browsing.

6. CONCLUSION

Software reuse refers to using software modules that were developed on a previous software project as part of a new software development project. Software reuse is a worthwhile goal since it has shown to reduce software costs, and improve software quality as well as programmers productivity. Software components have played an important role in modern software and system development. The main contribution of software components is reuse which helps reduces development cost and time, and increase productivity. While libraries of

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS.....

Unified Modeling Language (UML) diagrams and source codes do exist, one of the challenges that still remain is to locate suitable designs and source codes, and adapt them to meet the specific requirements of the software designer. The Use case diagram MDL file format contains valuable information about the requirements specification of software. These include use cases and actors. If we search the repository on the basis of attributes of MDL file descriptions, the search result would be better and thus giving higher precision, as compared to keyword based search. In this paper, we propose a tool named as Component Retrieval Search Engine, which assists the software designers in the retrieval of the designs as well as source codes.

References

[1] Wenbin (William) Dai, Valeriy Vyatkin, "A Component-Based Design Pattern for Improving Reusability of Automation Programs", IEEE 2013.

[2] Ivica Crnkovic, "Component-based Software Engineering: Building Systems from Software Components", Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02) IEEE, 2002.

[3] Burton, B. A., Aragon ,R. W. , Bailey ,S .A., Koehler ,K .D., and Mayer ,L .A, "The Reusable software library", IEEE Software 4, 4, 25-33, 1987.

[4] Lionel C. Briand, Yvan Labiche, Massimiliano Di Penta and Han (Daphne) Yan-Bondoc, "An Experimental Investigation of Formality in UML-Based Development", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 10, OCTOBER 2005.

[5] Robert A Andrews, Behan Webster, "A Component Oriented Software Engineering Approach to a Deeply Embedded Firmware based Control Platform", CCECE/CCGEI, Saskatoon, IEEE May 2005.

[6] Patrıcia D. L. Machado, Jorge C. A. Figueiredo, Emerson F. A. Lima, Ana E. V. Barbosa, Helton S. Lima, "Component-Based Integration Testing from UML Interaction Diagrams", IEEE 2007.

[7] Marco Tulio Valente, Member, IEEE Computer Society, Virgilio Borges, Leonardo Passos, "A Semi-Automatic Approach for Extracting Software Product Lines", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 38, NO. 4, JULY/AUGUST 2012.

[8] Robert B. France, James M. Bieman, SaiPradeepMandalaparty, "Repository for Model Driven Development (ReMoDD)", IEEE 2012.

[9] Cristina Seceleanu and Ivica Crnkovic, "Component Models for Reasoning", IEEE 2013.