http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (11): 2138-2145, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.2138.2145

Ontology-based Active Repository System

Yang Tao
Key Lab of Information Network Security, Ministry of Public Security,
339 Bi Sheng Road, Zhangjiang Hi-tech Park, Shanghai, 201204, China

Abstract: Component-Based Software Development (CBSD) is becoming increasingly more important in software engineering research and software development; however, it encounters many problems regarding its application. Here, based on the active repository system CodeBroker, the authors introduce a new approach to push components to end users according to their personalized information. The integration of the component repository, retrieval methods, queries, and the developer coding process reduces the CBSD cost on training, as well as renders CBSD applicable in research. Framework of this approach containing the following modules: building domain ontology, repository access agent, code analysis, personality catch and ontology-based component retrieval and push. Our experimental evaluation of the SourceForge projects on database field shows that suitable components can be automatically pushed to user through this approach. Using ontology reasoning and personality filtering, the proposed approach can improve the quality of the search results.

Key words: Active repository, ontology, CBSD, component

INTRODUCTION

Component-Based Software Development (CBSD) is considered the most effective way to improve the software productivity and enhance the capacity of software reuse. However, it does not enjoy much attention in terms of practical applications for several reasons. First, an integrated and recognized repository, a component search engine and a user-friendly interface are unavailable, making it difficult for users to find suitable components for specific applications. Second, its components require different environments in order to function; therefore, setting up in actual operation is difficult. Third, end users encounter multiple problems, likely stemming from the lack of quality control by developers. Fourth, troubleshooting any technical problem that arises upon installation is difficult. Fifth, estimating the time, cost and manpower required to shift from the normal development method to CBSD can be a challenge (Klein and Bernstein, 2004).

Numerous studies have been conducted to resolve the issues concerning CBSD execution, including middleware-based software development (Sharma and Gupta, 2010), aspect-oriented software development (Filman *et al*., 2004), service-oriented software development (Erl, 2004) and active repository (Ye *et al*., 2000), in hopes of streamlining the CBSD process and eliminate existing software issues.

Ontology refers to the important concepts and relationships in a particular domain, providing a vocabulary for that domain and a corresponding definition of terms (Motik *et al*., 2009). Ontology is now central to many applications, including scientific knowledge portals (Maynard *et al*., 2008a), information management and integration systems, electronic commerce and other semantic web services (Fensel, 2011).

Based on the active repository method, CodeBroker (Ye *et al*., 2000), a new approach aiming to customize CBSD as required by its end users is introduced in the current study. The approach essentially combines CBSD components with ontology, such that any CBSD component specifically needed for the user application is automatically detected and fetched by the system. This ultimately results in easier CBSD usage and application, as well as reduced time and cost for training.

Ontology-based component retrieval: An important issue in CBSD deals with ensuring that the component fits the user requirement. This issue is often bound with the component query method that contains keyword, interface and type, IOPE process-based and other matches (Klein and Bernstein, 2004). These methods require the same terminology to the component description and the user query. For example, if a component is designed for graphics support and describes its function using the keyword “graphic, ” the user can only find this component by typing in this specific word; these traditional search methods will not include “drawing, ” “image” and similar words.

Ontology-based component retrieval is the latest research field on CBSD. Because this method can recognize the semantic information of component description and user request by ontology support, it provides the knowledge match and reasoning functions to improve the component search quality (Pande *et al*., 2010).

The ontology-based component retrieval method first builds a common ontology library as knowledge base. Based on this library (Paquette and Masmoudi, 2011), it sets up the relationship between different concepts. It operates by matching the user query and the component description, providing results that are sorted by relevance.

Repository and sourceforge: A software repository is a storage location from which software packages may be retrieved and installed on a computer. Such repositories typically provide a software management system, as well as tools to search, install and manipulate software packages taken from the repository. The repository has a key role in the repeated use of software: it provides the software component description scheme, component classification, component storage and component retrieval functions.

There are many kinds of repository construction methods, such as REuse Based on Object Oriented Techniques (REBOOT) and JavaBeans Component Library JBCL (Hassan, 2008). However, few practical applications use the repository and no large-scale component providers and users have been identified.

Currently, SourceForge (Koch, 2009) is the most widely used open-source projects aggregation site to provide project release, project search and project source information, as well as download packages and repositories. Most CBSD researchers use SourceForge as a large-scale repository to support their research. In this study, SourceForge is used as a common repository for further research.

Active repository: Active repository is a methodology for software reuse. It includes the repository, source code editing tools and provides the active component query function for users.

CodeBroker (Ye *et al*., 2000) uses three agents to support the active repository's requirements: listener, fetcher and presenter. The listener integrates with the code editor and monitors the user input text containing the annotation, class name and function name. It generates the component query parameters which it then posts to the fetcher. The fetcher searches the repository using these parameters to find the suitable component. The presenter then dynamically displays the results in the code editor. Programmers can automatically access the components they need by merely typing several characters. However, because the query parameters CodeBroker uses are wholly dependent on text input, the results of the query are not very precise. CodeBroker currently supports only java; hence, the majority of its users are java programmers and developers.

iSPARQL (Kiefer *et al*., 2007) is another approach that supports the active repository. It provides a software repository data exchange format, named EvoOnt, to share, integrate and analyze data of various origins. Using the support of semantic web query language SPARQL, it provides the ability to mine software systems represented in the OWL data format. iSPARQL, however, does not support real-time code analysis and online component query.

ONTOLOGY-BASED ACTIVE REPOSITORY MECHANISM

The proposed ontology-based active repository system is built on the following mechanisms: (1) A domain ontology for component description and user query understanding: (2) A web repository agent that retrieves the component from the web repository and represents the component using ontology: (3) a code analysis method that retrieves user potential requests from the source file: (4) A personality catch module that encodes user personality, query history and execution environment and (5) An ontology-based component retrieval and push method that searches for the suitable component using code analysis and personality factor in the web repository and which then presents the result to the user's code editor.

Build domain ontology: A universally applicable description of the framework and standard terminology set are required in all stages of a query; hence, the proposed domain ontology was built to support these functions.

The domain ontology contains three parts. The first part, shown in Fig. 1, is called meta-ontology for CBSD domain knowledge. It contains the knowledge scheme of CBSD domain that represents the function in the "action-object" style (Cai *et al*., 2008). Action means the function's main work while object means the target of the function. The second part, shown in Fig. 2 is the

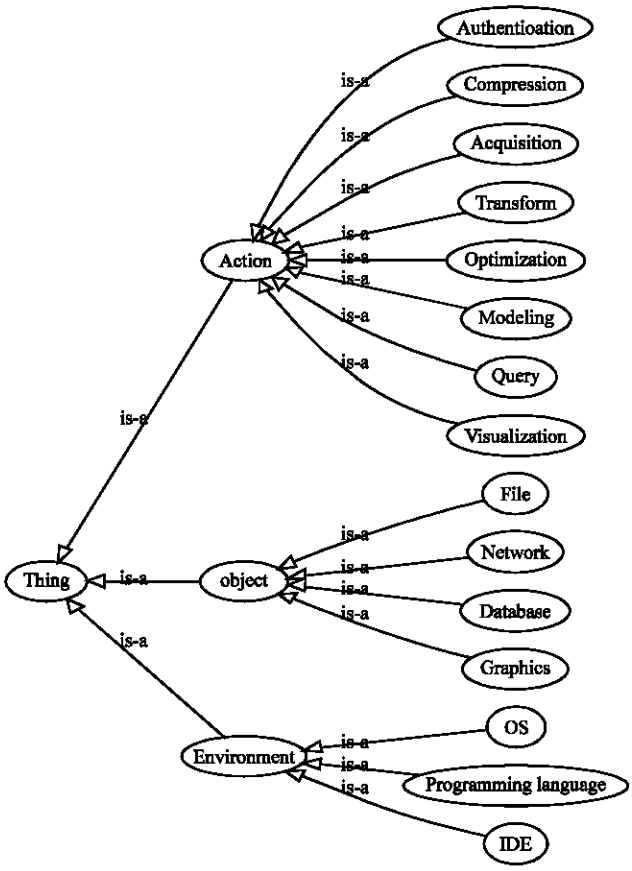


Fig. 1: CBSD Domain Knowledge

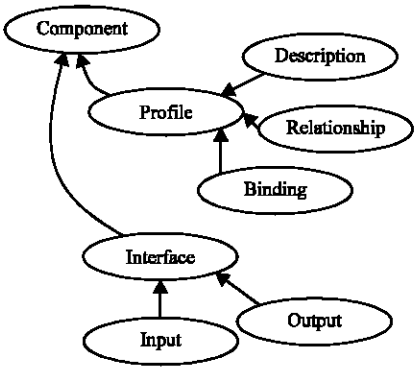


Fig. 2: Component meta-ontology

meta-ontology for component description. The profile is the main content of the component which contains three parts: description which refers to the functions of the component associated with the CBSD domain; relationship which means the dependence to other components and binding which denotes the binding runtime. Interface is the detailed description of the component's functions. Each function's interface has an input and an output. Using this meta-ontology, the user can represent a component and a component query.

The third part is the meta-ontology for personalized user information description (Fig. 3). The personality meta-ontology uses four properties to represent user behavior:

- Environment roperty refers to the codes auto-generated from the user's programming environment
- Application property refers to information of the program being used which can also be auto-generated from the source code
- Query history is the history of system which means that every query and result is recorded for future use
- User habit is the pattern of behavior which is derived from the user's operation and coding preferences. For example, a user who prefers JDBC uses the connection and statement class, whereas a user who prefers the hibernation system uses the data table entity class. By analyzing the code, the user habit can be inferred using the java database check

Web repository agent: Most repositories are typically web-based, unlike mainstream programs. To integrate the programming environment with the repository, a network-based code access agent or a repository access agent must be provided. Generally, CVS and SVN can support the network function and provide an appropriate solution for collaboration programming; however, they are not designed for repository access. A more consistent tool is necessary for CBSD to support repository access, as well as to help in active component search.

SourceForge was chosen as the web repository and a tool to access its component was designed. The tool has two major functions. First, a spider is provided for background retrieval from SourceForge, retrieving information such as keywords, URLs and function descriptions. These are then saved in a local database using the pre-defined component scheme. The component's description is also generated using the "action-object" style defined in the CBSD domain knowledge by an Natural Language Processing (NLP) to ontology processor (Maynard *et al*., 2008b). Second, a component access method is then provided for users to access the suitable component list and automatically download the selected component from SourceForge or view the information through a web browser.

Code analysis: The most important piece of information in technical research is usually a code. For example, the user writes a function, such as the following:

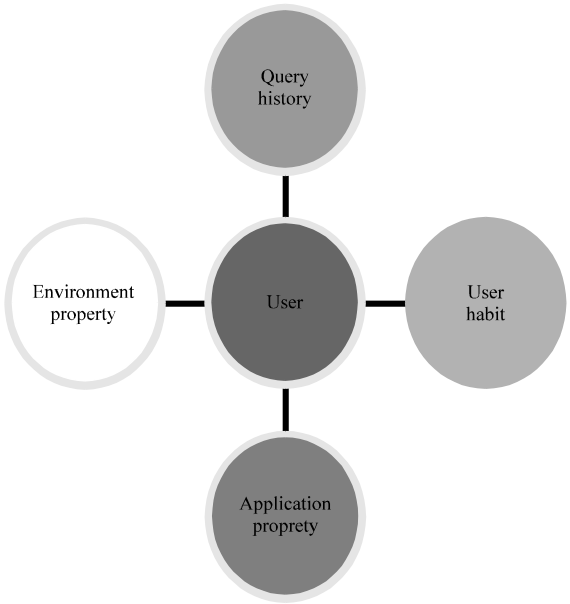


Fig. 3: Personality meta-ontology

boolean userLogin (String strName, String strPassword)

This means that a string of codes and parameters are used, such that the user can log into a system via usernames and passwords. The active repository system would perform a component query in the field of user login or user authentication and present a list of usernames similar to what the user has typed in.

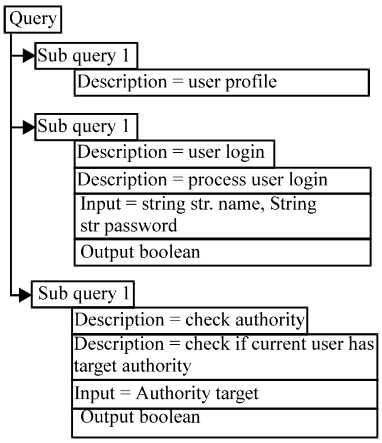
There are three types of analyses to process code editing: name, memo and detail code. Name analysis processes the file name, class name and function name and obtain semantic content from these names. Memo analysis then uses a NLP tools to obtain the main words and mapping these with ontology to obtain its semantic content. Detail code analysis processes the code to identify the object creation and call statements, for example, new () is an object creation code and user. Login () is a call code and then generate the relationship between objects to help the system recognize the application property and user habit. Table 1 shows a code analysis sample.

Personality catch: Normally, each user has a unique programming environment and coding habit and this information will evidently affect the repository search result. For example, a function name in a Java project with

Table 1: Code analysis sample

```
UserProfile.java
class UserProfile{
private String m_User;
public boolean userLogin (String strName, String strPassword){
//process user login
…
}
public boolean checkAuthority (Authority target){
//check if current user has target authority
…
}
}
```

Analysis result



Linux environment uses a component different from a C# project with Windows environment. Another example, a programmer familiar with JDBC and a programmer familiar with hibernate will each write different codes using different components when faced with a database application.

Meta-ontology for personalized user information description accumulates the user data and search patterns whenever a user accesses the active repository. The data in the active repository filters and sorts the results in a manner that is reflective of the previous search patterns.

Ontology-based component retrieval and push method: Component retrieval and push is the main process of the active repository. Component description has been recorded into the database by ontology annotation and by domain ontology; user's query parameters are also presented in ontology format to support the query execution.

The component retrieval process is split into two phases. In Phase 1, the ontology-based query engine uses code analysis to obtain data from the query, then stores the data in the local database. In Phase 2, the personality factor ontology is combined with the component description ontology of the results list which will further refine the results.

Upon completion of component retrieval, the repository access agent will take the result set to edit interface of the user. The user can access the suitable component list by automatically downloading the selected component from SourceForge or view the detail information on SourceForge in web browser. System will record the user's operation log to fill the personality information.

EXPERIMENT

Experiment background: The research was patterned after an agriculture project, named massive agriculture knowledge and resources management system. The main aim of this project was to integrate massive data from many agriculture knowledge systems and resource databases into one portal to make it more user-friendly.

Many of programming applications can be used accomplish this project; however, authors used CBSD because many components can be used to reduce development time and cost. For example, a project issue is to connect to multiple databases and integrate the data into a common interface. If this function is developed from scratch instead of integrating pre-existing components, more work will be required to make this function executable.

The authors provided an ontology-base active repository prototype system called OntoAR, based on the Eclipse platform and its plug-in module. The authors then evaluated the system function on automatic component push in the database domain.

System framework: OntoAR contains several modules, including code analysis, personalized information collector and component matchmaker. The overall framework of OntoAR is shown in Fig. 4.

In OntoAR system, Eclipse and SourceForge serve as the IDE and repository. Local Component Database is a local cache generated by the spider called Component Broker of repository access agent. The main data in local database are composed of the component index, its semantic parameters and the URL on SourceForge. The Eclipse Plug-in module contains two main functions: (1)

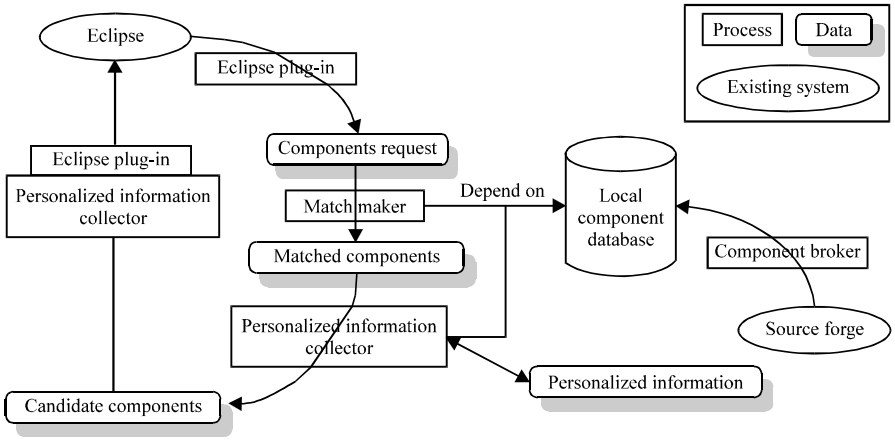


Fig. 4: System framework

code analysis which processes the project file and code file in Eclipse and generates the potential Component Request and the push candidate component, used in an Eclipse internal window, whether either downloads the component necessary for a project, or opens a web browser to view the component page on SourceForge.

After the code analysis, component request is generated and sent to Matchmaker for component retrieval. Matchmaker is an ontology-based component retrieval engine that matches the component request and the repository data and then returns the Matched Component by order of relevance. Matched components are then refined by Personalized Information Collector. This module provides the function that collects the user environment, application properties, user habit and query history and then uses these personalities to filter and sort the result.

Experiment and evaluation: OntoAR was developed via Java 1.7, with MySQL as the local database. The local database server is an Intel Core i7 PC with 4 GB RAM, running Windows Server 2003, with the component broker installed. The client, an Intel Core i5 PC with 2 GB RAM running Windows7, was installed with Eclipse software and the OntoAR in Eclipse plug-in format.

Before using the system, domain ontology is set up based on the computer ontology created by the Owl language. A small ontology database is created containing 165 classes, 31 restrictions and 52 properties in computer field, 131 of which are specifically related with the database field. To simplify matters, all of the individuals are ignored or converted into a subclass.

The repository spider then grabs all of the database-related projects from SourceForge, saves them into the local database and annotates them with the ontology tag. Prior to the experiment, the total amount of grabbed projects was recorded at 13, 012.

After the domain ontology and the local component database were ready, an experiment was conducted to examine the prototype system's ability. An empty project was built and a java file was created using the file name and code listed in Section 3.3. The component matchmaker initially found 130 results and, after filtering using personalized information as environment, were pared down to 63, some of which contain suitable components such as "SQLRunner" and "IBAccess." In contrast, after a direct query of the "user profile" in the database category of SourceForge, it returned 587 results, with the first appropriate component for the project located on page 3. Finding an appropriate relevant component in such a large results list is therefore more cumbersome.

A Time-consuming Analysis (Fig. 5) was also performed based on the total component number. The results in red with square nodes indicate the time cost of the component retrieval step which retrieves the component information from SourceForge and saves it to local component database. Figure 5 shows that the average time costs of one component are similar and the total time cost shows linear growth according to the total component number. Another result with triangle nodes is the time cost of semantic matchmaking step which uses Eclipse plug-in to analyze the user's code and generates the component query, retrieves the candidate components

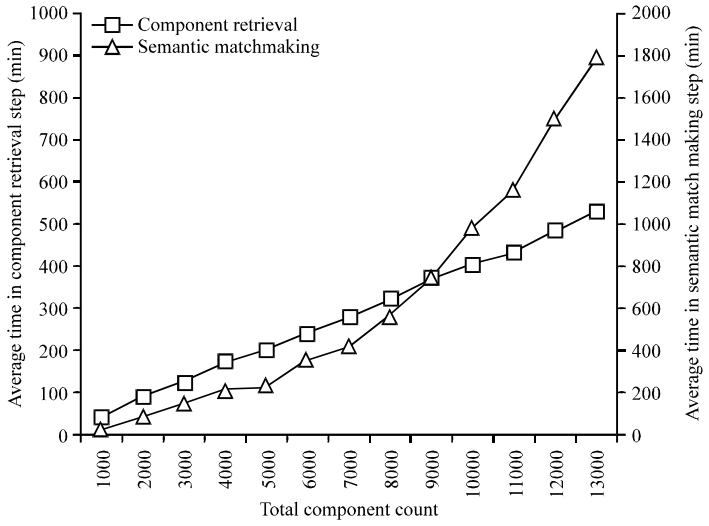


Fig. 5: Time-consuming analysis of 2 main steps in OntoAR

and shows the results. The main operation in this step is an ontology-based component retrieval. The ontology inference is most time-consuming process. The time cost of this step is a cubic growth of the total component number in the local database.

EXISTING PROBLEM AND FURTHER RESEARCH

In this study, the authors have introduced an ontology-based active repository approach to produce relevant components to the end users according to their personalized information. However, this mechanism still has a few issues, the most critical of which is the ability to support different programming environments. This approach combines the repository, automatic component query generation and component retrieval into an integrated IDE. In order for this method to be feasible, platforms aside from Eclipse and repositories other than SourceForge should be developed. Another problem is the automatic component composition and code adjustment method which, if resolved, will enable extreme ease of usability.

In future research, the following topics will be focused on:

- Completion of the realization of the prototype system and practical projects to examine the usability and query accuracy
- Support of different kinds of repositories by a repository metascheme

CONCLUSION

Based on the active repository system CodeBroker, the authors introduce a new approach to push components to end users according to their personalized information. By combining five steps of domain ontology building, repository agent access, code analysis, personality catch and ontology-base component retrieval and push, this approach provides an easy method for analyzing the user requirement and retrieves the suitable component for the user. Additionally, using ontology reasoning and personality filtering, the proposed approach can improve the quality of the search results. This integration of the component repository, retrieval methods, semantic queries and the developer coding process reduces the CBSD cost on training and improves the CBSD user-friendliness, especially for research. Experimental evaluation of the SourceForge projects on database field shows that suitable components can be automatically pushed to user through this approach.

ACKNOWLEDGMENTS

This work is supported by the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant No. 2012BAH95F03.

REFERENCES

Cai, Y.F., X. Peng and L.Q. Qian, 2008. An interactive query generation method for semantics-based component retrieval. Acta Electron. Sin., 8: 28-28.

Erl, T., 2004. Service-Oriented Architecture. Prentice Hall, Englewood Cliffs, USA.

Fensel, D., 2011. Semantic Web Services. Springer-Verlag, London, UK., ISBN-13: 9783642191930, Pages: 357.

Filman, R.B., T. Elrad and S. Clarke, 2004. Aspect-Oriented Software Development. 1st Edn., Addison-Wesley Professional, USA.

Hassan, A.E., 2008. The road ahead for mining software repositories. Proceedings of the Frontiers of Software Maintenance, September 28- October 4, 2008, Beijing, China, pp: 48-57.

Kiefer, C., A. Bernstein and J. Tappolet, 2007. Mining software repositories with isparol and a software evolution ontology. Proceedings of the 4th International Workshop on Mining Software Repositories, May 19-20, 2007, IEEE Computer Society, Washington, DC, USA., pp: 01-10.

Klein, M. and A. Bernstein, 2004. Toward high-precision service retrieval. IEEE Internet Comput., 8: 30-36.

Koch, S., 2009. Exploring the effects of source forge. Net coordination and communication tools on the efficiency of open source projects using data envelopment analysis. Empirical Software Eng., 14: 397-417.

Maynard, D., Y. Li and W. Peters, 2008a. NLP techniques for term extraction and ontology population. Proceedings of the 2008 Conference on Ontology Learning and Population: Bridging the Gap Between Text and Knowledge, June 16, 2008, IOS Press, Amsterdam, The Netherlands, pp: 107-127.

Maynard, D., Y. Li and W. Peters, 2008b. Nlp techniques for term extraction and ontology population. In: Ontology Learning and Population: Bridging the Gap between Text and Knowledge, Buitelaar, P. (Ed.). IOS Press, Amsterdam, Netherlands, pp: 107-127.

Motik, B., P.F. Patel-Schneider and B. Parsia, 2009. OWL 2 web ontology language: Structural specification and functional-style syntax. W3C Recommendation 27 October 2009/ http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/.

Pande, J., R.K. Bisht, D. Pant and V.K. Pathak, 2010. On some quality issues of component selection in CBSD. J. Software Eng. Appli., 3: 556-560.

Paquette, G. and A. Masmoudi, 2011. Ontology-based software component aggregation. Comp. Eng.: Concepts, Methodol. Tools Appli., Vol. 223. 10.4018/978-1-61520-839-5.ch013

Sharma, V.K. and N.P. Gupta, 2010. Component-based software development. Int. J. Comput. Sci, Network Secur., 10: 132-134.

Ye, Y., G. Fischer and B. Reeves, 2000. Integrating active information delivery and reuse repository systems. ACM SIGSOFT Software Eng. Notes, 25: 60-68.