Improved Natural Language Searching by Included Structural Information

December 9, 2008

Abstract

As applications become more complex with millions of lines of code, maintaining this code becomes an increasing burden. One fundamental set of tools involved in helping code maintenance finds segments of code given a higher-level concept. Although a number of techniques exist for this task, they have a number of issues. This paper discusses these limitations and proposes a number of techniques and studies which help define the problem, create bench marks and significantly improve the results of state of the art methods.

1 Introduction

A large portion of development time is spent maintaining code. Maintaining code generally involves fixing bugs, adding new features and refactoring. Fixing bugs often involves reading bug reports which are written using abstract concepts of what is wrong with the code. These abstract concepts then need to be used to find the incorrect code. This is a difficult task and any tools that help are a great benefit to developers. Adding new features typically involves two things, finding where the code needs to be added for the new feature and finding related code which can be used. Both of these searches require the programmer to map concepts to code. In some cases it is faster for a programmer to rewrite code than to find it. This of course is a huge waste of time. Refactoring generally includes finding duplicates and regrouping code by concepts. Code can be refractored into what are called aspects. These are sections of code related to particular concepts. All of these tasks can be greatly improved by tools which help programmers find concepts in code. This of course is the concept assignment problem, the problem this proposal aims to create and evaluate tools to assist programmers in these tasks. The proposed methods and studies attempt to fulfill a number of goals. They attempt to set forth a stricter definition of the problem, including a better sense of the problem's upper bound. They also hope to achieve better results for solving this method than the state of the art approaches. Finally they hope to improve the output of find concept tools to make the end result more useful.

2 Background

The concept assignment problem deals with mapping sections of an applications source code

to a particular concept or feature. Concepts represent functionality which exists in the application. For instance, a library book tracking application may have functionality which allows books in the system to be marked as checked out. In this case "is a book checked out" is considered a concept which maps to sections of code which checks to see if a book is marked as checked out. Due to object oriented and procedural practices, code sections which map to a concept may be located throughout an application's source code. This makes the problem of locating concerns difficult. There are several major approaches to concern location – lexical searches, information retrieval techniques, static program analysis, dynamic program analysis, and combined approaches.

3 State of the Art

3.1 Lexical Searches and Information Retrieval Techniques

Simple regular expression-based searches are often helpful in locating concerns. These searches typically have high recall but low precision due to over-generalization. It can be challenging to write regular expressions which represent a concern. Also these methods have difficulty handling synonyms and different word forms. Grep[8] is a popular lexical searching tool.

Latent semantic indexing (LSI) has been successful in many information retrieval tasks. LSI works by reducing the dimensions of the search space. This is very useful since lower dimensions result in quicker searches and automatically includes synonymies in searches. LSI was first applied to this problem by Andrian Marcus et al and was shown in some cases to outperform Grep [7]. Google, a popular search engine, now includes LSI[2]. Google also provides a search tool, Google Desktop, which also implements LSI[1]. Dapeng Liu and Shaochun Xu explore challenges of using LSI on this problem and compare Google desktop search to grep[6].

3.2 Static Program Structure

Another approach uses the static structure of a program to find concerns. Call graphs, dependence graphs and other static analysis methods can group code that is scattered across an application. The intuition is that these natural code groupings belong to single concerns. For instance, a function which calls another function is likely related to the same concept and an object which inherits from another object is likely related to that object. Robillard and Murphy have done much work in providing the proof of concept for this intuition[11, 10, 9]. Using historical information, such as when and where code was added can help locate concerns[3].

3.3 Dynamic Program Structure

Some information about a program's structure can only be gathered by executing it. Execution traces and other dynamic analysis methods can provide insight into where a feature is located in an application's source code [4]. For this to be effective, the feature being looked for has to be executed. This causes some problems since not all features can be manually enacted. Also the feature may have bugs which affect its execution, since concern location is often used to find bugs.

3.4 Combined Approach

One of the biggest disadvantages of using program structure is that there is no clear sense of where to start. Program structure is useful in

grouping code into potential concerns but which group represents the sought concern? Generally the user must seed the tools that use program structure. Natural language based-searching can be a great way to find a good starting seed and can provide information which is lacking in program structure. Information provided by program structure is going to be different than the information provided by natural language. Thus combining both natural language and program structure should show an increase in performance. The work of Pollock, Shanker, Shepherd, Hill and Fry demonstrate this increase and provide tools for using both kinds of information to locate concerns[5, 12].

4 Challenges and Goals

The concept assignment problem is difficult because code semantics is difficult to extract. There is no definite, objective meaning to code. The same code can be mapped to different concepts. For instance, a function that sorts a list might be organizing music in playlists or ranking search results. Thus the notion of a concept is tied to an application. Approaches which use only source code will have a difficult time creating this mapping since the concept assignment algorithm is blind to the purpose of the application. More specifically precision and recall should be increasable with new techniques. Current techniques generally map a concept to a procedure but more then one concept can exist in a function.

4.1 Lexical Searches

Pure lexical searches also have trouble with this mapping since the search queries are very close to the abstract idea of the concept. This is to say that search queries know nothing about the code they are searching. Identifiers provide some information however the same identifiers likely reappear throughout the code, regardless of what the code is doing. Information retrieval mitigates this a little by weighing identifiers which are very common in the code less.

4.2 Combined Approach

Combining both structural and identifier information seems like the right idea however it is not so straightforward. Structural information suffers from lack of knowledge about the goals and intents of the application. The code segments have meaning in terms of their relation to other code segments but they have no semantic bindings to the world. Identifier and other natural language information tends to be a bit too abstract to describe what the code is doing. Some attempts have been used to combine both types of information but they generally are sequential instead of parallel. Lexical search results can be filtered with structural information or vice versa, but this is using the information independently rather than dependently.

5 Proposed Research

Given the aforementioned limitations there are a number of things which could greatly improve the state of the concept assignment problem. The following ideas range from studies which attempt to better define the problem to creating new forms of output which try to increase the utility of existing approaches. All of the proposed ideas attempt to fix limitations in state of the art methods and generally improve upon existing tools.

1. Perform a study to compare the state of the art techniques to humans performing the concept assignment. On small manageable projects, automatic concept assignment is likely upwardly bounded by human ability. A study which measured humans' ability to map concepts to code may give the field a better sense of how they are doing in solving this problem.

2. Use identifier information to try to come up with more worldly semantic bindings. For instance, if the code is sorting a list, identifiers may contain information about what the list is holding and why it is being sorted. A model of the code could be built. This model or parts of it could be bound to concepts based on the identifiers and comments in the code. This model concept mapping could then be searched.

3. Lexical searches could interpret queries in a more source code-friendly way. For instance, if add is in the query: We might want to search for *.push(*). This could be based on the application being searched. With some notion of what adding an element to a data structure requires, methods which perform this task in the application could be found. Then these method names could be included in the lexical search. Just adding language-specific functions and keywords related to the query to the query might greatly improve performance.

4. Another way to combine structural information and lexical search is to search through an expanded version of the code. A lot of structural information comes from the call and inheritance structure. If a method call is replaced by its definition, and classes contain copies of all of their inherited methods, this should help improve IR techniques which use term frequency. Obviously, recursion would have to be handled. For recursion, only the top call should be replaced. Natural language information which occurs in recursion and other loops, could be weighted or considered plural. In addition to call and inheritance information, type information carries a lot of information. Every variable V could be replaced by V:TypeOf(V). The general idea is to organize structural information into the occurrence natural language information so lexical searching can be improved.

5. Treat the concept assignment problem like the document classification problem. Source code across applications is not that different. Given a large set of mapped code and concepts. Perhaps statistical methods could learn the relationship.

6. Create a commenting system that allows programmers to bind their code to a concept. Some sort of ontology could be used to make sense of this concept.

7. Instead of returning a list of methods or a graph perhaps a code coloring would be useful. This way methods could be divided as needed. Programmers would see their concept highlighted so they could more quickly address the code they are searching for.

6 Evaluation Plan

1. The study described under 1 in the previous section would be evaluated using f-measure, the harmonic mean between recall and pre-

cision. It would try to answer these questions: How does the state of the art compare to human performance? And what is the upper bound on how well a concept assignment method can perform?

2. The idea described under 2 in the previous section should be evaluated using f-measure against other state of the art methods. It should be run on large open source projects. The evaluation set should be created by an unbiased party. This idea is trying to answer: Can a model of the code be used to create more worldly semantic bindings for code snippets? This idea would implement a model generator which would create a model of the code. To simplify this idea a small domain should be tried first.

3. The third idea should also be evaluated using f-measure as described in 2. Work with java so all the standard java functions and keywords could be known by a query interpreter. The query interpreter would map this knowledge to the user query. Query expansion would be done by adding functions which call known functions, semantics would be combined to generate meaning for the new function. This idea would try to answer the question: Does program language knowledge improve the f-measure?

4. The fourth idea should also be evaluated using f-measure as described in 2. This would implement a code expander. The code expander would expand java code as described under this idea. This idea would try to answer the question: Does expanding the code to represent how it runs improve searching across it?

5. The fifth idea should also be evaluated using f-measure as described in 2. This idea requires a large set of mapped code and concepts. This is not readily available and would have to be constructed. It may be possible to construct this using some sort of boot strapping. This idea would try to answer the question: Can the concept assignment problem be treated as a version of the document classification problem?

6. The sixth idea would require a detailed specification of the concept language to be written. The system could be incorporated into something akin to JavaDocs. To evaluate this idea a study should be done to see if this idea saves time. The time taken to write in comments in this concept language should be subtracted from the time gained in searching for concepts.

7. Instead of returning a list of methods or a graph perhaps a code coloring would be useful. This way methods could be divided as needed. Programmers would see their concept highlighted so they could more quickly address the code they are searching for.

7 Summary of Foreseen Contributions

These ideas should help improve the concept assignment problem results. This will allow programmers to have a far easier time managing code. At least one of these ideas should improve the f-measure. Creating an easier way for programmers to specify what a portion of code is doing will have many benefits. This includes fixing bugs, adding features and refactoring.

References

[1] Google desktop, http://www.desktop.google.com/.

[2] Google, http://www.google.com/.

[3] Silvia Breu, Thomas Zimmermann, and Christian Lindig. Mining eclipse for cross-cutting concerns. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 94–97, New York, NY, USA, 2006. ACM.

[4] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of feature component maps by means of concept analysis. In *CSMR '01: Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, page 176, Washington, DC, USA, 2001. IEEE Computer Society.

[5] Emily Hill, Lori Pollock, and K. Vijay-Shanker. Exploring the neighborhood with dora to expedite software maintenance. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 14–23, New York, NY, USA, 2007. ACM.

[6] Dapeng Liu and Shaochun Xu. Challenges of using lsi for concept location. In *ACM-SE 45: Proceedings of the 45th annual southeast regional conference*, pages 449–454, New York, NY, USA, 2007. ACM.

[7] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 214–223, Washington, DC, USA, 2004. IEEE Computer Society.

[8] G. Project. grep. online, September 2006.

[9] Martin P. Robillard. A representation for describing and analyzing concerns in source code. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 721–722, New York, NY, USA, 2002. ACM.

[10] Martin P. Robillard and Gail C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 406–416, New York, NY, USA, 2002. ACM.

[11] Martin P. Robillard and Gail C. Murphy. Representing concerns in source code. *ACM Trans. Softw. Eng. Methodol.*, 16(1):3, 2007.

[12] David Shepherd, Zachary P. Fry, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development*, pages 212–224, New York, NY, USA, 2007. ACM.