RESEARCH STATEMENT

EMILY HILL

Today's software is large and complex, with systems consisting of millions of lines of code. Developers who are new to a software project face significant challenges in locating code related to their maintenance tasks of fixing bugs or adding new features. Developers can simply be handed a bug and told to fix it---even when they have no idea where to begin. In fact, research has shown that a developer typically spends more time locating and understanding code during maintenance than modifying it.

My primary research interests are in **software engineering**; specifically, my work focuses on reducing software maintenance costs through intuitive software engineering and program comprehension tools. To build more effective software engineering tools, I developed a model of word usage in software. Currently implemented for Java, the model provides software engineering tool designers access to both structural and linguistic information about the source code, where previously only structural information was available. I have applied this model to software search and program exploration tools, and evaluation has shown it to outperform competing state of the art approaches.

In addition to my thesis research in software maintenance, *I have significant research experience investigating a variety of other topics in software engineering*, including: testing web applications, protecting the privacy of run-time debugging feedback, automatically mining potential program refactorings, as well as designing and analyzing software engineering experiments. Through my varying research experiences, I have collaborated with and mentored a diverse range of students.

DISSERTATION RESEARCH

During software development and maintenance, human programmers read and modify the code that they and others produce, creating code artifacts that are readable as well as runnable. While the algorithm is conveyed by the programming language syntax and semantics, higher-level algorithmic steps and domain concepts are expressed through identifier names and comments. This textual information from comments and identifiers has been used to build software search tools for maintenance, but to date, their accuracy is not yet high enough for widespread adoption by practitioners.

There are two main issues which lead to inaccurate search results in software: (1) the developer chooses poor query words, or (2) the search mechanism does not take the context of the query words into account. My dissertation research addresses both of these issues with an inter-disciplinary approach that combines aspects of software engineering, program analysis, natural language processing, computational linguistics, information retrieval, text mining, and machine learning.

Selecting good query words. The first step in producing accurate search results is to find the "right" query words. Good query words not only describe the code for which the developer is searching, but also match the words used in the software's implementation. In my research, I address this problem in two ways. First, I developed a novel query reformulation tool that enables the developer to (1) quickly discriminate between relevant and irrelevant results, and (2) spark further query refinements [ICSE 09]. Second, I developed text mining techniques to automatically expand abbreviations in code [MSR 08] and collaboratively developed more accurate identifier splitting techniques [MSR 09]. Abbreviation expansion and identifier splitting improve search accuracy by ensuring the query words match the words extracted from the source code (e.g., ensuring the query word "string" will match the abbreviation "str" in the identifier "strlength").

Utilizing query words in context. The next step in producing accurate search results is to match the query words in context. For example, consider searching for the query "add item" in item processing software. The words "add" and "item" are so general and appear so frequently that this query will return many irrelevant results using existing approaches. In my research, I address this problem by more highly ranking search

results where the query words appear within the same linguistic or structural context. I take linguistic context into account by searching for the query words within specific word relationships, for example, by searching for the verb "add" and its object, "item". To realize this linguistic context, I developed a word usage model for software that extracts the noun phrases and verbs, along with their arguments, which describe the entities in the program. From these highly relevant results, I take structural context into account by automatically exploring structural edges and adding any relevant entities to the search results [ASE 07].

A word usage model for software. In the course of solving the practical problem of program search for maintenance, I developed a theoretical model of word usage in software, currently implemented for Java. The software word usage model (SWUM) is designed to be an interface between software engineering tool designers and researchers working on improved linguistic analyses for software. To date, I have developed and implemented a set of rules to automatically extract SWUM for Java. The rules were developed by analyzing naming conventions in over 9,000 open source Java programs using exploratory data analysis techniques. In addition, I have applied the model to build more effective software engineering tools for program search and exploration. Beyond program search and exploration, SWUM has a wide range of applications, including: program comprehension tools (automatic generation of UML diagrams or single-sentence comments for method declarations and invocations), software quality (testing with abstract types, detecting poorly named methods), and further development of customized linguistic analyses for software.

Evaluation. Throughout the research process, I evaluated my research in terms of accuracy and effectiveness. For example, I evaluated the accuracy of my program exploration tool by predicting what structurally connected components were relevant to a given maintenance task [ASE 07]. Relevance of program elements for each maintenance task was pre-determined by three independent software developers. The experiment showed that my program exploration tool, which uses both linguistic and structural information, outperformed the competing state of the art approach, which uses only structural information. As I complete my thesis, I will revisit this evaluation using the improved linguistic information provided by SWUM.

I have also evaluated the accuracy of automatically expanding abbreviations [MSR 08] and splitting identifiers [MSR 09]. For both studies, the gold set was created by two independent programmers. In an effort to reduce the human subjects' work, they were given identifiers not contained in an English dictionary, since non-dictionary words are more likely to contain abbreviations and identifiers which need splitting. In both studies, our automated techniques outperformed the competing state of the art.

In addition, I have performed a preliminary study of the accuracy of SWUM's extraction rules. Four human subjects were asked to identify the verb and objects for 20 method declarations. When compared with this gold set, SWUM's automatic extraction rules achieve 84% accuracy. I am currently expanding this study to include additional programmers and method declarations.

Finally, I have also evaluated the impact of my research on software engineering tools. Thus far, I have evaluated my query reformulation tool in terms of effort and effectiveness. In the study, 22 software developers were asked to search for code related to 28 maintenance tasks, 14 per subject. Compared with the competing state of the art, my query reformulation technique required less effort from the subjects and was more effective in locating relevant methods. In completing my thesis, I will extend this study to quantitatively evaluate my improved search technique.

FUTURE RESEARCH PLANS

As I complete my thesis, I will continue refining my search algorithm and evaluate the impact of my software word usage model (SWUM) on program search and exploration. Beyond that I plan to pursue extensions to my thesis work as well as take my research in new directions.

Extensions

In the short term, I will continue refining the extraction algorithm for automatically constructing SWUM for Java, and extend it to other programming languages such as C and C++. I plan to evaluate the success of using SWUM for other search-based tasks, such as documentation to source code traceability.

In addition, I would like to explore using SWUM for novice program comprehension. It is possible to use SWUM to automatically generate an English phrase for an arbitrary program statement. I plan to investigate whether automatically generated phrases for complicated Java statements improve program comprehension time in novices.

New Directions

The idea of using natural language information to improve software engineering tools is relatively new, and has only been applied in a few situations. In the long term, I plan to use my insights from developing SWUM to improve software engineering tools beyond program search:

Automatic Extraction of UML Diagrams. UML class diagrams are used to comprehend legacy systems as well as to design and document newly created ones. Software engineering tools like IBM's Rational Software Architect can automatically extract UML class diagrams. However, existing automatic techniques simply extract all class entities, without differentiating between methods and fields implementing high-level concepts versus low-level implementation details, which may distract from comprehension. By developing advanced linguistic and structural analyses such as those used in SWUM, I can create automated techniques capable of extracting UML diagrams at different levels of granularity and thus enhance comprehension.

Random Test Generation. Building on my previous research into testing as well as my thesis work, my next challenge is to use the linguistic and structural information in SWUM to improve random test generation for object-oriented programs. Software testing helps improve software reliability by exposing bugs, but developing effective test cases is notoriously time consuming. Carlos Pacheco, et al. recently suggested a feedback-directed random test generation technique that uses structural co-occurrence to determine which previously created objects are effective inputs to the next method call in the test case. Based on observations from creating SWUM, I would like to extend the idea of structural co-occurrence to include linguistically related objects that will create more diverse test cases capable of exercising more of the software with fewer tests.

In pursuing these research directions I will continue facilitating the creation of intuitive software engineering tools. My goal is for software engineering tools to support developers using the very information human developers would use---linguistic information from source code and other software artifacts in addition to programming language syntax and semantics. I plan to continue my interdisciplinary research by collaborating with researchers both internal and external to the university, as well as by getting feedback from software engineers in industry.