Research Statement

Paul “Will” McBurney

My research area is software engineering. The focus of my work is automatic source code documentation. Specifically, my research has studied what information should be conveyed in automatic source code summaries, how that information can be conveyed, and when automatic summarization can replace manual efforts. My research thus far has been used in automatic summarization tools, features discovery, and effort prioritization work.

Current Research Program

Programmers rely on high-quality software documentation. High-quality documentation can tell programmers the purpose of code, how the code relates to software behavior, and how that code can be used. However, programmers are notoriously poor at documenting source code. Because of this, researchers have developed a number of automatic software documentation techniques. Automatic software documentation techniques could be used to alleviate maintenance cost, ensure high source code readability, and document legacy code that is lacking.

Presently, however, there is no consensus on what high-quality documentation is. This lack of consensus creates difficulty in finding automatic documentation solutions. Researchers often disagree over what information in source code is important. Some automatic source code summarization techniques focus on selecting certain lines of code or identifiers to highlight important functionality. Other approaches leverage developer communications to determine the purpose of source code. If automatic software documentation is to continue to improve, researchers need understanding of what information is most important for a summary to convey.

My overall research goals are threefold. 1) Understand what makes source code documentation high-quality. 2) Apply this understanding to develop better automatic summarization techniques. 3) Create algorithms that apply both automatic and manual documentation techniques to improve source code comprehension. My research has contributed to achieving these goals, and opened up paths for future work in automatic software documentation.

Understanding High-Quality Documentation

To understand what makes high-quality source code documentation implies, I performed an empirical study into how authors and readers of source code summarize code. The findings of this study verify the existing assumption in software engineering research that that source code summaries should be similar to identifier names. Additionally, my research found that authors of source code often bring in information from outside of source code. One such example is contextual information, such as how different sections of source code interact. This research has been published in Empirical Software Engineering [1].
Developing Better Summarization Techniques

Using this understanding, I constructed an automatic source code summarization approach that summarizes methods context. Interestingly, I found that programmers preferred summaries which emphasized external method context, over a state-of-the-art automatic summarization approach [4], which selected and summarized internal lines of code. As yet, automatic summaries are not of the same quality of manually written summaries. However, further research will likely change this. This research received the Best Paper Award at International Conference on Program Comprehension 2014 [2], and a journal extension has been accepted to appear in IEEE Transactions on Software Engineering [3]. This journal extension found that manual summarization still outperforms my automatic approach with regards to accuracy and conciseness. However, surprisingly, programmers found that my automatic approach provided better understanding of how to use code and the impact of changing the code. Future research will explore other means of interpreting high-level concepts from source code to generate even better automatic summaries.

Prioritizing Documentation Effort

While automatic summarization of source code is outperformed by manually-written summaries, it still can serve an important purpose in software development. This is because automatic summaries are significantly faster and, by extension, cheaper than manually-written documentation. I have performed initial research into finding techniques for supplementing the documentation process with automatic summarization. This research is driven by the fact that, when understanding software, programmers focus on small sections of code, rather than trying to comprehend an entire system. Therefore, human documentation can be prioritized towards certain aspects of source code. Other sections of code less important to understanding the high-level concepts of software can be documented automatically. My initial research found that textual information in source code can be a strong indicator of documentation priority. Surprisingly, however, static source code attributes, such as source code size and complexity, are very poor indicators of documentation priority. This further highlights that automatic documentation techniques should continue to focus on interpreting high-level concepts from source code. This research is currently in submission. Future research will analyze textual information to find generally applicable guidelines for documentation prioritization.

Research Agenda

My first research proposal will explore using semantic information to improve automatic source code summarization. At present, nearly all source code summarization techniques rely on information retrieval (IR) of literal keywords from source code. However, these IR techniques rely almost exclusively on the frequencies of these keywords, and do not consider the meaning of the words. This creates a significant blind spot in automatically analyzing source code, as the concept assignment problem suggests that it is nearly certain different programmers within the same project will use different language to describe the same ideas. Semantic analysis can fill in this blind spot, as semantic analysis examines the meaning of words. In many cases, semantic analysis may find important relationships between sections of
source code that IR techniques would miss. Previously, I have used semantic analysis to examine the similarity between source code and summaries. This experience will prove useful in mining source code for semantic relationships.

Another improvement to automatic documentation could lie in dynamic source code analysis. Automatic documentation techniques have primarily used static source code analysis, largely ignoring dynamic source code analysis. Static source code analysis examines the source code directly, while dynamic analysis observes software behavior by executing the program. Most approaches rely on static source code analysis to collect information to document. Dynamic analysis could determine common patterns in source code execution. This could be valuable information to document, as it could better inform developers of software behavior than static source code analysis.

I have collaborated with researchers from multiple institutions, both academic and industrial. Currently, I am collaborating with DePaul University, the college of William and Mary, and the University of Kentucky on TraceLab, a reproducibility suite for software engineering research. TraceLab provides a framework for implementing repeatable software test studies. Previously, I have collaborated with an industry partner, ABB Corporate research, in studying documentation effort prioritization. Collaboration is especially important in software engineering research, as researched techniques must be put into practice to judge their efficacy. Given this, I plan to continue reaching out to industrial partners, grounding my research in real world application.

**Broader Impacts**

My research has several potentially broader impacts in improving economic competitiveness. Automatic documentation techniques could significantly decrease the barriers to entry to the marketplace for software developers. At present, documentation costs are prohibitively expensive. However, if software is under-documented, it can become difficult to maintain or use. This limits opportunities for start-up software developers, especially those in communities traditionally underrepresented. Decreasing documentation costs using automatic techniques could allow more opportunities in these communities. This would not just benefit start-up software firms. Thorough documentation will improve software readability of existing software projects, especially large projects, allowing new developers to be added to projects as they mature.

**References**


