Zachary P. Fry – Research Statement

My research focuses on studying the software maintenance process, especially as it relates to humans, in an effort to reduce the overall cost of maintaining large systems. I feel that this problem is of paramount importance as maintenance accounts for the great majority of the cost associated with the software lifecycle. It has remained largely a human-centric, manual process despite increasing levels of automation in related fields. For instance, while techniques have been developed to manage bugs and system understandability, the problem continues to grow as systems become larger and more complex. Thus, I believe more can be done to ease this burden in practice. As such, I have studied the software maintenance process from beginning to end, focusing on finding and fixing bugs while also ensuring continued system understandability and maintainability. My work emphasizes automation wherever possible as well as strong, concrete, human-centric evaluations to better understand both the problems and their solutions.

The work I have done to date has two main thrusts: first, I attempt to alleviate known software maintenance bottlenecks by providing automated solutions to difficult, often manual problems. Secondly, I study human-centric concepts related to understandability and continued system quality. For example, finding and fixing bugs is one of the primary concerns associated with software maintenance. As systems grow larger and more complex, the number of bugs often grows proportionally and may become harder to manage. Recognizing the magnitude and difficulty of these problems, I focus on automating parts of the bug localization and bug fixing processes to reduce the human burden associated with these maintenance tasks. Examples of my work in this area include **clustering similar defect reports, automatically localizing manually reported bugs, and improvements to an automatic program repair technique**. I believe a full examination of the software maintenance process must also explore secondary maintenance concerns including how humans find and fix bugs as well as what aids in making code more understandable. To this end, I have completed **user studies examining both fault localization and code understandability**.

Following these philosophies, my two high-level research goals are as follows:

- develop automated techniques to reduce maintenance costs
- empirically study human aspects of the software maintenance process

This parallel approach to **developing solutions to known, important maintenance problems while studying the impact such solutions might have on human-centric concerns** characterizes the nature of my personal research focus.

Reducing the Burden of Maintenance Tasks

Software maintenance tasks are being increasingly automated and streamlined, but these tasks remain largely manual. I believe further progress can be made to ease the human burden associated with maintaining software. To this end, **I have adopted a beginning-to-end approach to improving the maintenance process — focusing on bug reporting, localization, and fixing as well as continued system understandability**. As an example, I found that static analysis tools are effective at finding many potential defects in real software, but they often inundate users with thousands of defect reports all at once. My personal experience with one such commercial tool revealed that this is a concern shared by the tools' developers and the users alike. To reduce the human effort of triaging and fixing these potential defects I developed an automatic clustering technique so that similar defect reports may be handled in parallel. To cluster highly-related defect reports, I leveraged both the natural language inherent in the suspected-buggy code as well as the structural patterns in the reported call sequences. At a level of 90% cluster accuracy my technique is capable of parallelizing over 20% of the effort associated with managing the nearly 9,000 defects I studied. By surveying real developers, I determined that my tool's accuracy was deemed acceptable and also found that developers deemed the associated level of time savings to be helpful, in practice.

Locating manually-reported bugs is also a difficult process that represents a significant human effort. I developed a technique that exploits often-overlooked information to localize manually-reported defects. By treating both the bug reports and the code as structured documents, I was able to automatically identify meaningful language-based similarities (e.g., between the title of the bug and the method calls) to localize real bugs in software. Using a confidence model to return answers only for bugs for which the matching was empirically strong, my technique was able to pinpoint the bug 70% of the time, for over 500 real bugs.

My research has also focused on two areas of software maintenance normally associated with human intuition and creativity, which can be difficult to automate in practice. Fixing bugs is an important and historically human-intensive part of the maintenance process. Bugs are being reported so rapidly for many modern systems that companies have resorted to paying unknown developers to fix them through "bug bounty" programs, just to keep up. To deal with this deluge of bug reports, researchers have proposed automated tools to help suggest potential patches. For example, GenProg, an automatic program repair tool, made progress in this area by using a directed search to find a set of small program changes that patch a given bug. When examining both the nature of the bugs and specific parts of the GenProg algorithm, I noticed that many of the resulting patches were semantically redundant but textually different. Using this insight, I developed a new version of the technique that first removes redundant code changes from the search space and then exhaustively tests the remaining candidate patches. This new algorithm reduces the computing effort (and thus the monetary cost) of suggesting candidate patches by over 70% . Additionally, in the interest of continued system understandability, I am studying the impact of "stale" or out-of-date comments in the context of code changes in real systems. I have shown that humans struggle to recognize comments with incorrect or missing information and I am developing a tool that automatically identifies such comments and suggests fixes. I feel this work demonstrates my dual commitment to both the functional (e.g., bug fixing) and more abstract, non-functional (e.g., documentation quality) aspects of software maintenance.

The techniques I developed improve aspects of the software maintenance process from beginning to end and show concrete cost benefits. I believe software tools will be more readily adopted if their barriers to entry are low. Thus, I have paid particular attention to developing tools that do not require additional human input, where possible. As a result, all of the projects I have described require only existing software artifacts as input, adhering to the strictest definition of "automated" techniques by design. I also believe that a tool has not been fully evaluated until it is tested on real-world off-the-shelf software, ideally by humans. All of the tools described were evaluated on off-the-shelf programs with millions of lines of code and, where applicable, thousands of defects. I have further validated my research using actual humans and real maintenance tasks, where applicable. For instance, in the case of automatically clustered bug reports, I presented developers with the clusters produced by my technique and they agreed that the bugs described were, in fact, similar over 99% of the time.

Grounded Human Evaluations of Maintenance Concerns

I believe that a better understanding of the maintenance process fundamentally helps to motivate important problems and shape their solutions. To measure inherently abstract, human-centric properties, I have performed **extensive human studies to explore the maintenance process and systematically evaluate some of my work in context**. The results were somewhat surprising and forced me to reexamine my assumptions about how developers carry out maintenance tasks.

I believe the process of developing software maintenance tools can greatly benefit from answers to the difficult question: "What actually makes code harder to understand and debug?" If we know which bugs are harder to fix and why, we might tailor maintenance techniques toward helping developers in those cases. I have performed two human studies to answer these questions and provide further insight into how developers interact with code. The first of these two studies measured which types of bugs (e.g., a missing conditional clause or calling the wrong method) and code features (e.g., "provides a heap abstraction" or "uses multiple for loops") make it difficult for humans to locate bugs. I found that certain abstractions make bugs harder to find — for instance, humans are over three times more accurate at finding defects in code that provides an array abstraction than in code that provides a tree abstraction.

The second human study evaluated different types of patches in terms of human understanding. Based on my previous work with automated program repair, I decided to compare the future maintainability of automatically-generated patches with those written by humans. I found that automatically-generated patches are as understandable as human-written patches with the aid of minimal machine-generated documentation. I believe this type of human-centric, non-functional information can shape future advancement in software maintenance, for instance, by directing fault localization or automatically adding documentation to more obscure or complex code.

The results from these human studies have only increased my desire to further study human-centric maintenance concepts. One theme that characterizes the results of both studies is that humans often fail to understand even their own behavior with respect to maintenance tasks. Specifically, I found that participants were overwhelmingly bad at identifying features that correlated with their own personal accuracy when answering comprehension questions. Additionally, I found that neither measured nor human-perceived difficulty correlates well with traditional program understanding metrics (e.g., code readability or pedagogical "difficulty"). I formulated a model of code understandability from these results that is more than two times as correlated with humans' actual ability to find bugs than existing code quality and understanding metrics. These somewhat surprising results led me to reevaluate my own assumptions and have taken my research in unexpected but fruitful directions. I am excited by this sort of exploratory research that helps us understand exactly what impedes the maintenance process and how we might fix the some of the related problems.

Conclusion

In conclusions, my research to date has focused on concretely improving maintenance tasks while studying the more abstract, human-centric tasks associated with the software maintenance process. I have developed techniques to help with clustering similar defect reports, automatically locating manually reported bugs, and facilitating automatic program repair. Additionally, I have conducted comprehensive human studies examining how humans both find bugs and understand code. I am excited to continue working to ease the software development and maintenance burdens.