

Raymond Buse – Research Statement

“The most radical possible solution for constructing software is not to construct it at all.” -- Fred Brooks

Background

Powerful development environments, data-rich project management systems, and ubiquitous software frameworks have fundamentally altered the way software is constructed and maintained. Today, professional developers spend less than 10% of their time actually writing new code and instead primarily try to **understand existing software** [1]. Increasingly, programmers work by searching for examples or other documentation and then assembling pre-constructed components [2]. Yet, code comprehension is poorly understood and documentation is often incomplete, incorrect, or unavailable [3]. Moreover, few tools exist to support structured code search making the process of finding useful examples ad hoc, slow and error prone. It is estimated that software defects, often related to code understandability, cost the US economy many billions of dollars each year [4].

*My overarching goal is to **help humans better understand software** at many levels and in doing so improve development productivity and software quality.*

Approach

My research lies at the intersection of Programming Languages and Software Engineering. My work is motivated by practical problems encountered by programmers, including myself.

Algorithms and tools are often the product of my research. I use *Programming Language techniques* (e.g., symbolic execution, test case generation, specification mining, bug finding, etc.) to (1) better understand the relationship between developers and code (e.g., [5]) and (2) synthesize documentation and other artifacts to help developers better understand software (e.g., [6]). I design algorithms that *scale* to large systems often by leveraging *machine learning* and *statistical methods*.

Empirical findings ground my work. I leverage large repositories of code, defect reports, and testing artifacts both for experimental purposes [7] and to assist developers [8]. I am often motivated by my experience interacting with developers and project managers during my time at Microsoft Research [9]. At a high level, I want to help users move from only answering questions of information like “What happened?” to also answering questions of insight like “How did it happen and why?”

Creative experimental design is a distinguishing characteristic of my research. Evaluating tools and conducting research pertaining to program understanding is a unique challenge; it increasingly hinges on studies involving human-created artifacts [8] or actual humans [10]. I have significant experience in this domain as the principle researcher on four IRB-sanctioned and published human studies. Additionally, I was the lead author on a paper which empirically characterized the benefits and barriers of human studies in over 2,000 software engineering research papers published in the last ten years [11].

Collaboration is crucial and I am equally comfortable working on my own or with faculty, graduate students, and undergraduates. I’ve established multiple working relationships with external collaborators in academia [11] and in industry [9]. As the leader of a research team I would endeavor to grant students the flexibility to pursue their own ideas whenever possible; in my experience, helping to evolve a student’s own ideas often yields the best results.

Research Experience

I've published six conference papers, one journal article, and one position paper. I also have authored two additional conference papers and a journal article that are currently under review. I have received an ACM distinguished paper award and in 2009 I was awarded the UVa Computer Science **Graduate Research Award**. In this section I'll elaborate on a few of the projects that have led to these publications.

A **Readability Metric** is an automated tool for objectively measuring the understandability of text. I am the lead author of a 2008 paper which introduced the first such metric for program source code [7]. Our model for readability, which is internally based on logistic regression, was *learned* from data gathered from 120 study participants. The importance of measuring code readability is highlighted by the observation that reading code is now the most time consuming part of the most expensive activity in the software development process [12].

Readability metrics for natural language have been in use for over 50 years. The Flesch-Kincaid metric, for example, has not only been integrated into popular text editors including Microsoft Word, but has also become a United States governmental standard. The Department of Defense requires many documents and forms to meet have a Flesch readability grade of 10 or less (DOD MIL-M-38784B). I believe that readability metric can serve a similar role in the context of software. An implementation of the metric has been made available and work has been cited at least thirty-five times since 2008 and directly extended at least once [13].

Documentation Synthesis is the process of generating explanatory text from software artifacts. A 2005 NASA survey found that the most significant barrier to code reuse is that software is too difficult to understand or is poorly documented [14]. As software systems grow ever larger and more complex, I believe that automated documentation tools will become indispensable.

The key to my approach is adapting programming language techniques (e.g., symbolic execution) to create output that is directly comparable to existing human-written artifacts. This has two key advantages: (1) it simplifies evaluation by permitting objective comparison to existing documentation, and (2) it enables tools to be used immediately – no significant change to the development process is required. To date, I've developed documentation synthesis algorithms for exceptions [6], code changes [8], and APIs (under submission – ICSE '12).

Other research projects are ongoing. I am always on the lookout for collaboration opportunities. As an example, I've worked with architecture researchers to develop an algorithm suitable for conducting alias analysis – a key type of program analysis – in a SIMD context (i.e., suitable for Graphics Processors - GPUs). By combining PL insights about constraint ordering and satisfaction with hardware insights about GPU memory hierarchies we created a prototype that runs 2.5 times faster than the state of the art.

Future Work

I believe that research in **software comprehension and automatic documentation is generally wide open**. In this section I'll describe several promising lines of research that I intend to pursue.

Readability models, as they exist today, are relatively nascent. Like metrics for natural language, existing models for code are static and based on a small number of semantically shallow features (e.g., the length of identifier names). However, the structured nature of code coupled with powerful existing analysis tools holds out the promise of much more precise readability models in this domain.

I propose to move beyond shallow features and build readability metrics capable of **leveraging semantically rich information** such as types, dataflow, and aliasing. Furthermore, because readability may be highly variable among individuals and organizations I propose to specialize with **online learning**. Ideally, a model should continuously

adapt over time. Finally, at conferences I am often asked what steps can be taken to increase the readability of existing code. Unfortunately, current models are purely descriptive; they do not support prescriptive advice (e.g., how can I make this function more readable?). Therefore, I propose to develop a normative model for readability capable of providing **actionable feedback**. Several strategies are possible: from mining repositories for changes that impact readability to lightweight human studies.

Example synthesis takes automatic documentation a step further. Where previous algorithms can explain specific static artifacts (e.g., *What* will happen if F is called?), I plan to generate new useful code (e.g., *How* should I use F?). I propose to automatically **synthesize representative examples** for arbitrary data types. One candidate algorithm mines a code base for usages, clusters these uses on factors like statement ordering and type information, and then distills syntactically valid and semantically representative examples.

Furthermore, I propose to support **more complicated queries** (e.g., how can I use class X with class Y?). Existing programming communities (e.g., StackOverflow.com) may constitute a novel test bed. Such forums can be used to characterize the type of questions programmers commonly ask as well as validate the output of an automated tool. Another possibility is not to require an explicit user-level query at all and instead **automatically form hypotheses** about programmer intent and suggest code on the fly. Such a tool could use pattern matching (perhaps based on tree differencing algorithms on ASTs) to access the compatibility of candidate patterns.

Software Development Analytics is an original concept recently described by myself and Tom Zimmermann of Microsoft Research [9]. We observe that while many powerful tools and metrics exist today, it is not unusual for major software projects to fail or be delayed [15]. We conjecture there is a substantial disconnect between the information needed by project managers to make good decisions and that which is delivered by existing tools.

For example, a tool might report *What* the readability of an artifact is (e.g., readability of the GUI module is low), but this simple measurement is difficult to act on. In contrast, an analytics tool might report *Why* that's important and *How* it happened (e.g., readability of the GUI module has decreased significantly in the past two weeks and 90% of the movement can be explained by the commits of a certain developer). **The idea of analytics is to transform information into insights by layering different kinds of analyses** that allow for summarizing, filtering, modeling, and experimenting.

To benefit from analytics we must first **characterize the information needs of managers**; in comparison to developer needs, management needs are rarely investigated by researchers. A preliminary study based on a survey of over 100 Microsoft project managers is currently under submission. We believe this study lays the groundwork for the next step: **designing analytics tools**. An early prototype, for example, combines anomaly detection on repositories and topic analysis [16] on commit messages to warn project managers about unusual events when they happen. A final approach is **training and education**. We believe that software analysts could be trained as part of a software engineering curriculum modified to emphasize quantitative skills.

Funding

As a graduate student, I had the opportunity to participate in developing proposals for research funding. In 2011, the National Science Foundation awarded a grant entitled *Synthesizing Human-Readable Documentation* (award number 1116289) in the amount of \$470,955 to my advisor, Wes Weimer. The proposal was based on a subset of my dissertation work and I contributed significantly to the proposed ideas and text. I believe this provides additional evidence, beyond publication success, of the potential merit and impact of my research ideas.

As a professor I plan to continue to seek funding for my work under **NSF:SHF** (Software and Hardware Foundations) as well as through an **NSF CAREER** award. I will also pursue funding through **non-NSF government** (e.g., DARPA) and

industry collaboration (e.g., Microsoft). Additionally, I intend to collaborate on larger funding efforts with members of my new department and others. Finally, I believe a significant portion of my work may lend itself well toward **interdisciplinary study** (e.g., cognitive science and psychology) and I would investigate those opportunities as well.

Conclusion

The majority of my research is based on the observation that the focus of software engineering is shifting from implementation to design and composition concerns. Increasingly, **software comprehension is a critical factor in modern development**. However, software remains difficult to understand. This has led to poor developer productivity, software defects, and delayed or abandoned projects. I plan to continue to investigate three parallel solution tracks: (1) models for software understandability, (2) algorithms for documentation synthesis, and (3) tools for software analytics.

I have a demonstrated ability to independently formulate and solve research problems and then publish the results at top-tier venues. When given the opportunity, I've also had success helping to obtain external funding. I look forward to pursuing the challenges I've outlined in this document as well as many others. I hope you grant me the chance to do so at your institution.

References

- [1] P. Hallam, "What do Programmers Really do Anyway?," in Microsoft Developer Network (MSDN) C# Compiler, 2006.
- [2] S. de Souza, N. Anquetil and K. de Oliveira, "A study of the documentation essential to software maintenance," in International Conference on Design of Communication, 2005.
- [3] D. Novick and K. Ward, "What users say they want in documentation," in International Conference on Design of Communication, 2006.
- [4] National Institute of Standards and Technology, "The economic impacts of inadequate infrastructure for software testing," Technical Report 02-3, Research Triangle Institute, 2002.
- [5] **R. Buse** and W. Weimer, "The road not taken: Estimating path execution frequency statically," in International Conference on Software Engineering, Vancouver, 2009.
- [6] **R. Buse** and W. Weimer, "Automatic Documentation Inference for Exceptions," in International Symposium on Software Testing and Analysis, Seattle, WA, 2008.
- [7] **R. Buse** and W. Weimer, "A Metric for Software Readability," in International Symposium on Software Testing and Analysis, Seattle, WA, 2008.
- [8] **R. Buse** and W. Weimer, "Automatically Documenting Program Changes," in International Conference on Automated Software Engineering (ASE), Antwerp, Belgium, 2010.
- [9] **R. Buse** and T. Zimmermann, "Analytics for Software Development," in FSE/SDP Workshop on the Future of Software Engineering Research, Santa FE, NM, 2010.
- [10] **R. Buse** and W. Weimer, "Learning a Metric for Code Readability," IEEE Transactions on Software Engineering, vol. 36, no. 4, p. 546–558, 2010.
- [11] **R. Buse**, C. Sadowski and W. Weimer, "Benefits and Barriers of User Evaluation in Software Engineering Research," in Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA), Portland, OR, 2011.
- [12] R. Glass, Facts and Fallacies of Software Engineering, Addison-Wesley, 2003.
- [13] D. Posnett, A. Hindle and P. Devanbu, "A simpler model of software readability," in Mining Software Repositories, Honolulu, HI, 2011.
- [14] National Aeronautics and Space Administration, "Software Reuse Working Group," 2005. [Online]. Available: http://www.esdswg.com/softwarereuse/Resources/library/working_group_documents/survey2005.
- [15] T. Addison and S. Vallabh, "Controlling software project risks: an empirical study of methods used by experienced project managers," in SAICSIT, 2002.
- [16] D. Blei, A. Ng and M. Jordan, "Latent Dirichlet Allocation," Journal of Machine Learning Research, vol. 3, no. 4-5, pp. 993-1022, 2003.