

# JOSEPH LAWRANCE

## RESEARCH STATEMENT

My work and research interests are at the intersection of HCI, software engineering, and psychology. My focus has been on advancing our understanding of how programmers make decisions during software maintenance. I have an empirical approach to research: I have conducted controlled experiments and field studies to test my hypotheses.

I have successfully pursued collaborations with industry and academic research partners, as well as government funding agencies. I have worked with colleagues at Microsoft, Oregon State University, IBM Research, and Harvard University to investigate both professional and end-user programmers. I have been awarded grants from the NSF and Air Force, and recently authored an NSF grant proposal to continue my dissertation research program.

In collaboration with Microsoft, I conducted empirical research to examine how code coverage visualizations influence the testing behavior of professional programmers [13]. A controlled, between-subjects experiment of 30 programmers revealed that such visualizations led programmers to write enough tests to achieve coverage, but no more than necessary. Overall, programmers overestimated the percentage of the failures that they uncovered with testing, but programmers exposed to code coverage visualizations were even more overconfident.

At Oregon State University, I developed a co-reasoning system that combined the strengths of multiple spreadsheet testing and fault localization systems in collaboration with Dr. Martin Erwig [7], studied the tinkering behavior of end-user programmers in collaboration with researchers at Drexel University and the University of Cambridge [2], and developed a Japanese-English cross-language information retrieval system in collaboration with Dr. Jonathan Herlocker and his students.

In collaboration with IBM Research, I investigated how information foraging theory can explain and

predict programmers' navigation through source code during debugging. More details appear in the Dissertation Work section.

Currently, I am at MIT collaborating with a group of experimental economists at Harvard University's Program for Evolutionary Dynamics to develop a new web-based platform, called cWeed, for running large-scale experiments developed by end-user programmers. The experimental economists are interested in running experiments of economic games, such as Prisoner's Dilemma, Ultimatum, and various kinds of auctions, using the Web to gain more insights into the decision-making behavior of thousands of people. We are working to allow the economists, as an important specialized category of end-user programmers, to create, test, and debug these kinds of games easily and publish them for anonymous web users to play.

In preliminary experiments, we were able to reproduce the same scale of experiment that the experimental economists already perform in the lab, but at significant lower cost in time and money. Whereas our collaborators used to spend days recruiting 30 subjects to come to the lab at a reserved time, at a cost of hundreds of dollars, cWeed is able to do the same job in 4 minutes, for only \$20.

### **Dissertation Work**

During debugging, source code navigation is a central task. Recent research has shown that programmers spend up to 35% of their time navigating source code [5, 6]. Little is known, however, about how programmers navigate source code and the extent to which information relates to their navigation.

Research into programmer navigation during debugging has yielded descriptive (but not predictive) theories of programmer behavior (e.g., [6]) and tools to assist programmer navigation (which lack a theoretical basis) [4, 5].

My research aims to address these shortcomings by adapting information foraging theory to model programmer navigation behavior during debugging. Adapting this existing theory to software debugging is more attractive than building new theories, because the theory has already proven itself to be effective at modeling information-seeking behavior on the web.

Information foraging theory explains how people seek and gather information [14]. It is based on optimal foraging theory, a theory of how predators and prey behave in the wild. Just as predators desire prey for the least possible effort, information seekers want relevant information without much effort. The predator/prey model can predict which web pages people select: cues associated with links (e.g., text surrounding a hyperlink) hint at the location of relevant information. Information scent represents how people decide which cues are worth following to answer their information need. Models of information foraging use spreading activation techniques to represent information scent [1].

To determine whether information foraging theory applies to software maintenance, I recruited 12 professional programmers from IBM, each having at least two years of Java experience. Participants were encouraged to think aloud as they worked for three hours on a bug report and a feature request in an open source news reader written in Java.

In adapting information foraging theory to debugging, I mapped the concept of “prey” to the bug report, and “information patches” to the code. Taking the view that classes in a program that have word-level similarities to the bug report provide scent for the location of bugs, I computed information scent as the cosine similarity of classes against the bug report, as is common in information retrieval systems.

The information scent computation predicted which classes programmers visited most often, both individually and as a group [8]. A separate analysis of data collected from Sourceforge.net revealed that information scent also predicted where software maintainers made fixes to close issues [10].

In a follow-up analysis [9], I built a predictive model, named Programmer Flow by Information Scent (PFIS), that builds upon the WUFIS algorithm (Web User Flow by Information Scent) [3], an empirically validated algorithm that has predicted web browsing behavior. The algorithm models the probability that programmers will follow a particular “link” from one class or method in the source code to another.

First, PFIS traverses the Java abstract syntax tree to build a graph that models the topology of links from usages to definitions (e.g., from a method invocation to a method definition), and computes the similarity of the text at the head of each link in relation to the bug report. Then, PFIS simulates programmers randomly following links and “activating” nodes in the topology, with a preference for following links that have a higher scent with respect to the bug report. Finally, after several iterations, the “activation” of each node in the topology converges, and this “activation” acts as a predictor of where programmers will allocate their attention in the source code.

In terms of predicting which classes individual programmers visited most frequently, the PFIS model predicted navigation behavior better than: information scent from the bug report [9], information scent from programmer think-aloud verbalizations [11], and other individual programmers’ navigation [9]. Furthermore, the predictive power of PFIS was not significantly different from the aggregate number of visits to classes as a predictor of individual visits, indicating that the PFIS model could stand in for a team of programmers necessary to inform recommender systems like Team Tracks [5].

From what I learned in evaluating PFIS, I developed a new model called PFIS 2 [12]. PFIS 2 is similar to PFIS; however, PFIS 2 accounts for changes to source code over time, and furthermore, PFIS 2 does not require an explicit description of the prey to make predictions. After conducting a seven-month field study of professional programmers at IBM, I found that half of the time that programmers navigated between methods, the next method they visited (out of over 700 possible methods) was within the top three predictions produced by PFIS 2. Over a quarter of the time,

the next method was the top prediction PFIS 2 made.

### Future work

I aim to provide a rigorous predictive model and theory of debugging and other software maintenance activities and develop systems based on my dissertation work. From this, I hope to make obsolete existing practices of building software maintenance tools ad-hoc, instead enabling a foundation for such tools based on principles demonstrated to work.

The encouraging results so far suggest rich opportunities for further investigation, both in the short term and beyond. That PFIS or PFIS 2 were able to make accurate predictions is somewhat incredible, considering that neither account for: programmers' existing knowledge or runtime information (such as the value of variables) necessary to debugging.

In the near-term future, I believe I can improve our understanding of how programmers debug and also improve predictive power by computing semantic similarity as a measure of information scent (to account for programmers' knowledge), and by examining how runtime information influences programmer behavior (and adapting the models as appropriate).

I also look forward to opportunities to continue my other work, and I also look forward to applying theories and principles from psychology, machine learning, information theory, and economics to understand and address the needs of programmers, software engineers, and other populations of professionals and end users.

### References

- [1] Anderson, J. R., Bothell, D., Byrne, M., Douglass, D., Lebiere, C., Qin, Y. An integrated theory of mind, *Psychological Review*, 111, 4, (2004), 1036–1060.
- [2] Beckwith, L., Kissinger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A. and Cook, C. Tinkering and Gender in End-User Programmers' Debugging, *Proc. CHI 2008*, ACM Press (2006), 231-240.
- [3] Chi, E., Pirolli, P., Chen, K. and Pitkow, J,

Using information scent to model user information needs and actions on the web. *Proc. CHI 2001*, ACM Press (2001).

- [4] Cubranic, D., Murphy, G., Singer, J., and Booth, K., Hipikat: A project memory for software development, *IEEE Trans. Soft. Eng.* 31, 6 (2005), 446-465.

- [5] DeLine, R., Czerwinski, M. and Robertson, G., Easing program comprehension by sharing navigation data, *Proc. VLHCC, IEEE* (2005), 241-248.

- [6] Ko, A, Myers, B., Coblenz, M., Aung, H., An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks, *IEEE Trans. SE* 32, 12 (2006), 971-987.

- [7] Lawrance, J., Abraham, R., Burnett, M. and Erwig, M. Sharing Reasoning to Improve Fault Localization in Spreadsheets, *Proc. VLHCC, IEEE* (2006), 35-42.

- [8] Lawrance, J., Bellamy, R. and Burnett, M. Scents in programs: Does information foraging theory apply to program debugging? *Proc. VLHCC, IEEE* (2007) 15-22.

- [9] Lawrance, J., Bellamy, R., Burnett, M. and Rector, K., Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. *Proc. CHI 2008*, ACM Press (2008), 1323-1332.

- [10] Lawrance, J., Bellamy, R., Burnett, M. and Rector, K. Can information foraging pick the fix? A field study, *Proc. VLHCC, IEEE* (2008).

- [11] Lawrance, J., Bellamy, R., Burnett, M., Rector, K. Information foraging theory of program navigation: The roles of hypotheses and scent. Submitted to CHI 2009.

- [12] Lawrance, J., Burnett, M., Bellamy, R., Bogart, C. and Swart, C. Reactive Information Foraging for Evolving Goals, *Proc. CHI 2010* (To appear).

- [13] Lawrance, J., Clarke, S., Burnett, M. and Rothermel, G. How well do professional developers test with code coverage visualizations? An empirical study, *Proc. VLHCC, IEEE* (2005), 53-60.

- [14] Pirolli, P. and Card, S. Information foraging, *Psychology Review* 106, 4, (1999), 643-675.