Human-Oriented Software Engineering

Research Statement

Scott D. Fleming

Software developers continue to face considerable challenges designing, implementing, and maintaining software systems despite steady advances in software engineering (SE). The software engineering literature contains numerous tool designs that aim to help developers perform complex information-intensive tasks. Regardless of the particular problems these tools aim to solve, they often address fundamental questions like *what information would help the developer most at a given moment? How to present that information to the developer? How to intervene without interfering?* However, the tool creators generally lack a theoretical and principled basis for answering these questions, and instead, tend to make design decisions in an ad hoc manner, relying heavily on their intuitions.

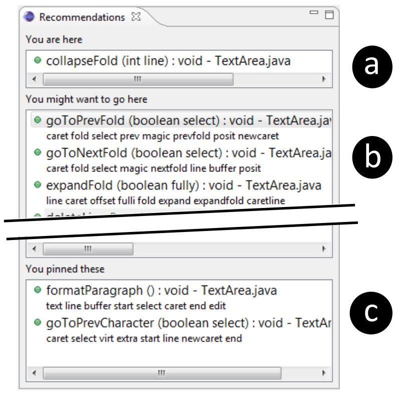
My research takes a human-oriented approach both to help software developers directly through improved tools as well as to help the creators of software engineering tools by providing foundational design guidance and methods. In particular, my work emphasizes the application of theories, principles, and methodologies from the areas of human-computer interaction and psychology to the problems of software engineering. Using this approach, my research has advanced understanding of information-seeking and sense-making behaviors of software developers, and provided principled and empirically grounded implications for the design of software engineering tools. In this research statement, I highlight my human-oriented research to date and outline my research agenda moving forward.

1 Information Foraging Theory for Software Engineering

One of my central research threads explores leveraging a human-behavioral theory from HCI and psychology, *Information Foraging Theory* (IFT), to advance knowledge on how developers go about seeking information and to provide tool creators with theoretically grounded design guidance and methods [11, 15, 16, 17, 18, 19, 20]. To date, software engineering researchers have made little use of theories of human behavior in designing SE tools. Such theories potentially could explain why SE tools succeed (or not), could predict whether an approach will succeed, could guide the design of tools, and could generate ideas for new tools. Information Foraging Theory has a strong record of accomplishing these things in the domain of user-Web interaction. My work seeks to harness IFT to bring such benefits to SE researchers.

By empirically viewing developers through an IFT lens, my research has produced new insights into developer foraging. For instance, my colleagues and my studies have demonstrated that IFT constructs characterize professional developer behavior more completely than popular older theories of program comprehension based on hypothesis-verifying behavior [15] Moreover, our studies have revealed that professional developers forage reactively, with rapidly evolving information goals [16], and we have mapped out the types of information that developers seek on tasks and the strategies they use to get that information [20]. Such findings help to flesh out a more complete picture of developer foraging, and also point to refinements for IFT itself to enhance the theory's applicability to software developers.

We have also harnessed IFT's predictive power to explore predictive models of developer foraging and to apply those models to the design of tools, such as recommender systems. Using one such IFT-based model, PFIS, we designed a recommender system, implemented as an extension to the Eclipse Integrated Development Environment (Figure 1). The premise of the recommender is that by observing the foraging decisions a developer is making, a system can infer the sort of information that the developer is seeking and can recommend places containing such information. In an initial evaluation involving professional developers, the developers successfully used our recommender system to navigate to new places in the code and also as shortcuts to navigate to known places [16].


Recommendations

You are here

collapseFold (int line) : void - TextArea.java

You might want to go here

goToPrevFold (boolean select) : void - TextArea.ja
caret fold select prev magic prevfold posit newcaret

goToNextFold (boolean select) : void - TextArea.ja
caret fold select magic nextfold line buffer posit

expandFold (boolean fully) : void - TextArea.java
line caret offset fulli fold expand expandfold caretline

You pinned these

formatParagraph () : void - TextArea.java
text line buffer start select caret end edit

goToPrevCharacter (boolean select) : void - TextAr
caret select virt extra start line newcaret end

Figure 1: IFT-based recommender system, displaying (a) the current method, (b) the recommended methods, and (c) methods "pinned" by the user.

A final thrust of this research thread has been to package IFT concepts in form useful to software engineering tool designers. A common strategy for aiding the creation of software is to provide *design patterns*, which are general, reusable solutions to a common design problems. Thus, we aim to contribute a catalog of patterns about how to support developers navigations during software development. Our approach is to map the theoretical constructs and propositions of IFT directly to design patterns for software development tools. To this end, we performed a systematic survey of the literature on software maintenance tools, and proposed an initial set of thirteen information foraging design patterns [11]. We then used this set to seed a community portal for which we have recruited software engineering tool researchers to collaboratively propose, review, and refine new patterns. In addition to being useful in their own right, these information foraging patterns also serve to educate the SE tool research community about IFT and its utility as a foundation for tool design.

2 Navigation Affordances in Code Editors

Following from my work on information foraging, another of my research threads aims to advance the design of navigation affordances in code editors to significantly enhance developer productivity by enabling them to navigate more efficiently [12, 13, 14]. Navigation of information spaces is a highly relevant problem in software engineering as well as in HCI more generally. For example, studies have found that developers spend over one third of their time on the mechanics of navigation alone. My own IFT work has found that developers spent around half of their time foraging, which is by its nature a navigation-heavy activity [20].

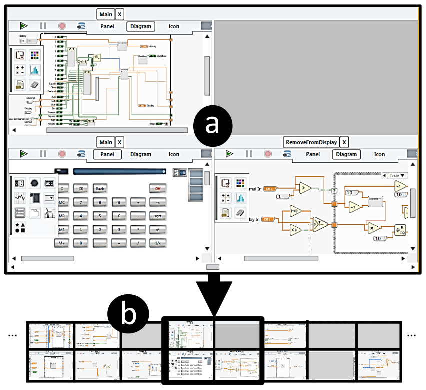


Figure 2: Patchworks code editor, with (a) sliding view into (b) grid strip.

To address this problem, I conceived of a new editor design, *Patchworks* (Figure 2), which my research group has been iteratively refining and evaluating. The design centers around a grid-strip interface idiom. Two key design decisions were to promote juxtaposing code fragments (i.e., placing them side by side on screen) with the fixed grid and to limit the space of navigation options to two (sliding left or right) with the grid strip.

Our empirical evaluations of Patchworks have produced promising results. We have implemented and evaluated Patchworks prototypes for both Java and the visual programming language LabVIEW. Our evaluations have found, for example, that programmers using Patchworks were able to navigate significantly faster than with Eclipse [12], that they made significantly fewer navigation errors [12], and that they juxtaposed code significantly more, and as a result, around three times as many of their navigations were to code fragments already on screen [14, 13].

3 Diversity among and within Programming Populations

One of the limitations of IFT is that it does not explicitly account for differences between individual developers, be they from the same population or from different populations; thus, several threads of my research aim to advance knowledge of diversity among and within programming populations. To address the issue of diversity *among* populations, I have studied two distinct populations of developers beyond traditional application developers: developers of concurrent software [8, 9, 10], and end-user mashup programmers [3, 4, 5, 6, 7]. To address diversity *within* populations, I have investigated whether and how theoretical gender differences from the sociology and education literature occur in the context of development environments [1, 2].

3.1 Developers of Concurrent Software

The dominant trend in computer architecture is toward parallelism; however, concurrency substantially increases complexity of software, presenting developers of such software with considerably different challenges than those faced by the traditional application developers. For instance, developers of concurrent software have difficulty localizing certain types of concurrency-related bugs because they cannot reliably reproduce the associated program failures, a side effect of the non-determinism inherent to concurrency. In an extreme case, the developers of Photoshop took 10 years to discover the cause of one subtle, persistent concurrency-related bug.

To address the problems of developing concurrent software, my dissertation work applied a human-oriented approach to investigate the barriers that such developers face, their strategies for coping, and tools for supporting them. My studies showed that successful developers tended to employ certain debugging strategies that unsuccessful developers did not [9, 10]. For instance, one such strategy linked to success was *failure-trace modeling* in which a developer modeled scenarios of thread interaction to determine if a suspected error state was reachable [10]. However, the developers generally did such modeling informally, running through scenarios in their heads, and they often had difficulty correctly reasoning about the potential behavior of concurrent software. To address this problem, I developed an extension to UML sequence diagrams with features for representing multithreaded interactions [8]. In an evaluation study, developers who created concurrency-extended sequence diagrams made fewer errors in reasoning about a buggy multithreaded program. These results illustrate the benefits of applying a human-oriented approach to special populations of developers.

3.2 End-User Mashup Programmers

In contrast to traditional developers, *end-user programmers* often lack formal training in computer science, and are not necessarily interested in acquiring such training. This population of programmers are on the rise: one study estimated that there would be 13–55 million end-user programmers in American workplaces in 2012, compared to only 3 million professional programmers. However, despite their brisk growth, the software engineering literature has only begun to understand and address the special needs of end-user programmers.

To help fill this gap, my colleagues and I performed an in-depth investigations of end-user programmers who do “mashup” programming in the CoScripter programming environment. A central goal of our work was to provide effective support for helping end-user programmers problem-solve their own way around barriers they encounter. This led to the creation of the *Idea Garden*, a concept designed to help end-user programmers generate new ideas and problem-solve when they run into barriers [3, 4]. It uses an integrated, just-in-time combination of scaffolding for problem-solving strategies, and for programming patterns and concepts. Our empirical results showed that the Idea Garden helped end-user programmers overcome several key barriers (e.g., not knowing how to compose components) [4, 5, 6].

3.3 Gender Differences and Programming Environments

Gender is an easily recognizable source of diversity within programming populations. It is well known that females are underrepresented in the computing field, and I am concerned that their problem-solving needs may not be well met by programming tools and environments. Further fueling this concern, the literature describes theoretical gender differences that play out in technology; however, the possibility of gender issues in the use of programming tools and environments has received almost no attention. Understanding such gender issues is critical to a foundation for SE tools that aim to help all programmers, regardless of gender.

To help address this concern, I applied a human-oriented approach to understanding gender differences in programming environments [1, 2]. In particular, my colleagues and I conducted an empirical investigation of nearly 3,000 people from a variety of programming populations, including professional and hobbyist programmers [1]. Our results showed significant gender differences in programming-tool usage across all the populations. For instance, males and females favored different kinds of features within programming environments, and males exhibited greater willingness than females to tinker with features in programming environments. Thus, SE tools that emphasize the types of features favored by males or that require tinkering will tend statistically to benefit males more than females. These findings illustrate the importance of accounting for gender in the design of SE tool, and they serve to inform the design of tools that help both genders.

4 Research Agenda

My research vision is to create a foundation for the design of SE tools that support software developers in their information-intensive tasks. Moving forward, my research agenda will continue to leverage a human-oriented approach based on theories, principles, and methodologies from HCI and psychology for understanding developers and enhancing design.

One thrust of my work will be to make information foraging theory accessible to software engineering researchers and industrial tool builders. My recent IFT work has yielded a catalog of design patterns linking theory to tool designs. However, I still need to discover how to help tool builders put these new patterns into practice when they need to create a new tool. What I will seek, specifically, is a *design method* that will enable a tool builder, who has requirements about assisting some sort of foraging, to design an effective tool. Methods are common kinds of contributions in software engineering research, and have been proposed, for example, for designing real-time systems, web sites, and software agents. My goal therefore will be to develop a design method that will enable tool builders to practically and effectively harness IFT for their design tasks. Achieving this goal will require establishing and validating step-by-step guidance for tool builders, as well as supporting materials that aid tool builders in applying IFT, validated through field studies and other empirical methods.

Another thread of my future research will explore *recommender systems for software engineering* (RSSEs) to aid developers in navigation and information foraging. For example, I envision an RSSE that monitors a developer's context, including user history, software repository data, web-forum data, etc., and dynamically provides navigation affordances that enhance developers' productivity and reduce their effort. My previous work on an IFT-based recommender was a promising first step [16]; however, research opportunities remain both for deciding *what* information to recommend and *how* to deliver that information in a form that the developer finds usable. Regarding what information to deliver, I will investigate combinations of IFT-based models and other models, such as degree-of-interest and degree-of-knowledge models, from the literature. Regarding how to deliver it, I will investigate extending my Patchworks design [12] to seamlessly integrate the recommendations into the existing interaction design.

Expanding upon my work on end-user mashup programmers, I will investigate improving *refactoring* affordances in end-user programming environments. Refactoring generally involves the transformation of existing code to improve design and to enhance non-functional properties, such as maintainability, reusability, readability, and performance. However, refactoring in end-user programming languages, such as the visual dataflow language LabVIEW, has received almost no research attention. To help fill this gap, I will build a detailed empirical characterization of refactoring in visual dataflow languages. This work will answer, for example, *what types of refactorings are common in visual dataflow languages? To what extent do these overlap with refactorings common in text-based languages? What specific barriers do end-user programmers face when performing refactorings?* Based on these answers, I will design and evaluate new empirically grounded refactoring affordance for end-user visual-dataflow programmers. As first steps toward these goals, I have secured funding from National Instruments (the makers of LabVIEW), and my preliminary investigations are underway.

Finally, the success of Information Foraging Theory suggests that other human-behavioral theories might hold promise for understanding and supporting software developers' other activities, such as collaborative problem solving and design. I will seek out theories that can describe developers activities in ways that explain why SE tools succeed, that generalize over multiple software engineering research areas, and that uncover new patterns or principles, leading to productive research opportunities. Such theories will enable software engineering researchers to build SE tools on solid theoretical foundations that have been empirically demonstrated effective.

References

[1] Margaret Burnett, **Scott D. Fleming**, Shamsi Iqbal, Gina Venolia, Vidya Rajaram, Umer Farooq, Valentina Grigoreanu, and Mary Czerwinski. 2010. Gender Differences and Programming Environments: Across Programming Populations. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10)*. 28:1–28:10. DOI:http://dx.doi.org/10.1145/1852786.1852824

[2] Margaret M. Burnett, Laura Beckwith, Susan Wiedenbeck, **Scott D. Fleming**, Jill Cao, Thomas H. Park, Valentina Grigoreanu, and Kyle Rector. 2011. Gender Pluralism in Problem-solving Software. *Interacting with Computers* 23, 5 (Sept. 2011), 450–460. DOI:http://dx.doi.org/10.1016/j.intcom.2011.06.004

[3] Jill Cao, **Scott D. Fleming**, and Margaret Burnett. 2011. An exploration of design opportunities for "gardening" end-user programmers' ideas. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '11)*. 35–42. DOI:http://dx.doi.org/10.1109/VLHCC.2011.6070375 ***Best Paper Award***.

[4] Jill Cao, **Scott D. Fleming**, Margaret Burnett, and Christopher Scaffidi. 2015. Idea Garden: Situated Support for Problem Solving by End-User Programmers. *Interacting with Computers* 27, 6 (2015), 640–660. DOI:http://dx.doi.org/10.1093/iwc/iwu022

[5] Jill Cao, Irwin Kwan, Faezeh Bahmani, Margaret Burnett, **Scott D. Fleming**, Josh Jordahl, Amber Horvath, and Sherry Yang. 2013. End-user programmers in trouble: Can the Idea Garden help them to help themselves?. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '13)*. 151–158. DOI:http://dx.doi.org/10.1109/VLHCC.2013.6645260

[6] Jill Cao, Irwin Kwan, Rachel White, **Scott D. Fleming**, Margaret Burnett, and Christopher Scaffidi. 2012. From Barriers to Learning in the Idea Garden: An Empirical Study. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '12)*. 59–66. DOI:http://dx.doi.org/10.1109/VLHCC.2012.6344483

[7] Jill Cao, Kyle Rector, Thomas H. Park, **Scott D. Fleming**, Margaret Burnett, and Susan Wiedenbeck. 2010. A Debugging Perspective on End-User Mashup Programming. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '10)*. 149–156. DOI:http://dx.doi.org/10.1109/VLHCC.2010.29

[8] **Scott D. Fleming**, Eileen Kraemer, R. E. K. Stirewalt, and Laura K. Dillon. 2010. Debugging Concurrent Software: A Study Using Multithreaded Sequence Diagrams. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '10)*. 33–40. DOI:http://dx.doi.org/10.1109/VLHCC.2010.14

[9] **Scott D. Fleming**, Eileen Kraemer, R. E. K. Stirewalt, Laura K. Dillon, and Shaohua Xie. 2008a. Refining Existing Theories of Program Comprehension During Maintenance for Concurrent Software. In *Proceedings of the IEEE International Conference on Program Comprehension (ICPC '08)*. 23–32. DOI:http://dx.doi.org/10.1109/ICPC.2008.40

[10] **Scott D. Fleming**, Eileen Kraemer, R. E. K. Stirewalt, Shaohua Xie, and Laura K. Dillon. 2008b. A Study of Student Strategies for the Corrective Maintenance of Concurrent Software. In *Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE '08 Education Track)*. 759–768. DOI:http://dx.doi.org/10.1145/1368088.1368195

[11] **Scott D. Fleming**, Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. 2013. An Information Foraging Theory Perspective on Tools for Debugging, Refactoring, and Reuse Tasks. *ACM Transactions on Software Engineering and Methodology* 22, 2 (March 2013), 14:1–14:41. DOI:http://dx.doi.org/10.1145/2430545.2430551

[12] Austin Z. Henley and **Scott D. Fleming**. 2014. The Patchworks Code Editor: Toward Faster Navigation with Less Code Arranging and Fewer Navigation Mistakes. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '14)*. 2511–2520. DOI:http://dx.doi.org/10.1145/2556288.2557073

[13] Austin Z. Henley, **Scott D. Fleming**, and Maria V. Luong. 2015. Toward Principles for the Design of Navigation Affordances in Code Editors: An Empirical Investigation. in submission. (2015).

[14] Austin Z. Henley, Alka Singh, **Scott D. Fleming**, and Maria V. Luong. 2014. Helping programmers navigate code faster with Patchworks: A simulation study. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '14)*. 77–80. DOI:http://dx.doi.org/10.1109/VLHCC.2014.6883026

[15] Joseph Lawrance, Christopher Bogart, Margaret Burnett, Rachel Bellamy, Kyle Rector, and **Scott D. Fleming**. 2013. How Programmers Debug, Revisited: An Information Foraging Theory Perspective. *IEEE Transactions on Software Engineering* 39, 2 (Feb. 2013), 197–215. DOI:http://dx.doi.org/10.1109/TSE.2010.111

[16] David Piorkowski, **Scott D. Fleming**, Christopher Scaffidi, Christopher Bogart, Margaret Burnett, Bonnie John, Rachel Bellamy, and Calvin Swart. 2012. Reactive Information Foraging: An Empirical Investigation of Theory-based Recommender Systems for Programmers. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '12)*. 1471–1480. DOI:http://dx.doi.org/10.1145/2207676.2208608

[17] David Piorkowski, **Scott D. Fleming**, Christopher Scaffidi, Margaret Burnett, Irwin Kwan, Austin Z. Henley, Jamie Macbeth, Charles Hill, and Amber Horvath. 2015. To Fix or to Learn? How Production Bias Affects Developers' Information Foraging during Debugging. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME '15)*. 11–20.

[18] David Piorkowski, **Scott D. Fleming**, Christopher Scaffidi, Liza John, Christopher Bogart, Bonnie E. John, Margaret Burnett, and Rachel Bellamy. 2011. Modeling programmer navigation: A head-to-head empirical evaluation of predictive models. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '11)*. 109–116. DOI:http://dx.doi.org/10.1109/VLHCC.2011.6070387

[19] David Piorkowski, Austin Z. Henley, Tahmid Nabi, **Scott D. Fleming**, Christopher Scaffidi, and Margaret Burnett. 2015. Foraging and Navigations, Fundamentally: A Developer Problem of Predicting Value/Cost. in submission. (2015).

[20] David J. Piorkowski, **Scott D. Fleming**, Irwin Kwan, Margaret M. Burnett, Christopher Scaffidi, Rachel K.E. Bellamy, and Joshua Jordahl. 2013. The Whats and Hows of Programmers' Foraging Diets. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '13)*. 3063–3072. DOI:http://dx.doi.org/10.1145/2470654.2466418