Research Statement

Chris Bogart (bogart@eecs.oregonstate.edu)

My research lies at the intersection of Human-Computer Interaction and Software Engineering. I believe that we can empower more people to program and to customize the technology around them if we work to make programming and debugging a fluid, reactive interaction pitched at user-appropriate levels of abstraction. Programming is really just the process of turning abstract and ill-formed human intentions into action plans, well-formed and concrete enough to allow technological agents to save us more time than their errors might cost us. Humans have done this kind of work naturally without computers, as we have been codifying and revising laws, training animals and children, enforcing social norms and household rules, etc. I want to study these natural and ubiquitous human-human interaction patterns with the goal that someday programming a computer can feel more like explaining a task to a human assistant.

Thesis Research

For my thesis research I show how a methodology based on observation and modeling of human interaction can lead to better tools for one interactive aspect of programming: the evolving set of "evaluation abstractions" programmers use to understand program behaviors. My target audience consists of scientists writing scientific models, specifically psychologists writing cognitive models in specialized languages such as ACT-R. I studied their abstractions both when they were using software tools to examine their models, and when they talked and reasoned about their models with other scientists. Three aspects of the thesis I will discuss below are: (1) a theory, EAST, describing the affordances needed for a tool to support evaluation abstractions, (2) an extention I devised to the Natural Programming methodology, for studying evaluation abstractions, and (3) a prototype tool (EAST-Env) I built informed by the new theory and methodology.

Theory of evaluation abstraction support (EAST): This theory attempts to describe the factors influencing a modeler's decision to spend some of their cognitive capacity, already burdened by the demanding task of debugging, on the use of evaluation abstration features. The theory suggests that modelers will use such features if, among other things:

- They get immediately useful feedback as they incrementally build the abstraction the feature relies on (i.e. progressive evaluation [G92]), and can immediately see how their changes will affect the tool's response;
- They can reuse and compose the abstractions they build;
- The feature's GUI affordances are cued both in terms of the operation they will perform, and the results they will return.

For example, the theory would suggest that Eclipse designers could increase usage of the tool's conditional breakpoint feature (which currently is a special-purpose dialog box the user must fill out) by:

- Immediately rewarding the user with some useful tidbit of information (probably using some fast static analysis) about the circumstance(s) when that breakpoint would be hit, especially as compared to their most recent change to the breakpoint;
- Capturing the criterion in a way independent from its use as a breakpoint, so it can be easily repurposed as part of an assertion, a unit test, lines in a log file, or other program analyses and visualizations;
- Visually tying the criterion to the variables it references in source code, rather than showing it only as an expression in a separate dialog box.

Using the dialog-based interface as it exists today requires a programmer to deliberately set aside their current question about program behavior, and devote explicit thought to the mechanics of the problem-solving tool. However if a designer ties the feature's configuration as closely as possible to the elements of the problem the programmer is thinking about, the theory claims this will decrease the task-switching cost associated with employing the feature.

This focus on the cognitive impact of the structure and labeling of tool affordances was related to a broader research effort with my advisor, Margaret Burnett, and her research group, aimed at understanding in general how programmers navigate through code and other software engineering artifacts when debugging. Beyond the cognitive modeling domain, we have been working on a specialization of Pirolli's theory of Information Foraging [PC99] to the domain of Java programmers navigating through large code bases. Information foraging theory describes human searches for information using models originally developed to describe animals foraging for prey. Our group evaluated IFT as a way of characterizing and predicting programmer navigations [LBB13]; we compared the usefulness of several different factors for predicting navigation (such as structural or textual ties between methods) [PFS11] and we later extended the theory to account for the evolution of programmers' goals over time [PFS12].

Extending the Natural Programming Methodology [BBD12]: The methodology I used for my thesis integrated language design with empirical human subjects research. Natural Programming [PM06], as originally described, involves observing users first, then designing tools in response to those observations, iterating on the design with users, then evaluating the resulting tool. In my thesis I began with an empirical study [BBD10], which enumerated and described the evaluation abstractions modelers were using, and the roles their abstractions served. Rather than simply going straight to tool designs, however, I first designed a domain-specific language to describe the space of abstractions that modelers talked about, and used that language as a codeset for analyzing the transcripts of a user study. This let me empirically refine and validate the langauge as a representation of modelers' evaluation abstractions, and it also later served as a specification for a prototype tool supporting modelers' exploration of their models' behaviors. It also enabled forms of validation earlier in the design cycle than is called for by the original methodology.

Prototype tool: (EAST-Env) To further explore and test both the language describing evaluation abstractions, and the theory of evaluation abstraction support, I built an experimental tool for cognitive modelers, designed based on these theories, as a plugin to

the Eclipse IDE, supporting five of the cognitive modeling frameworks that my study participants were using for their own work.

I used EAST-Env to look at how modelers sequenced their abstractions in the course of reasoning about a model, in a Wizard of Oz study that had modelers state their queries to me in English instead of having them use an interface. I used the content of questions that modelers asked me while solving a problem to refine and validate the language, to better represent the abstractions embodied in their questions. By analyzing the temporal *sequencing* of abstractions in their questions, I was able to make sure the language represented temporally adjacent English questions with similar expressions in EAST-Env's language. This let the language capture both the structure and sequence of modelers' abstractions. Finally I used the same tool to do a summative study of the evaluation abstraction support theory.

Future Research Plans

One promising avenue of follow-on research from my thesis would be to see how the output of EAST-Env's queries could be combined with backward slicing to produce new tools for understanding causation in program traces. Andrew Ko's Whyline [KM08] was a debugging tool that made backward slicing accessible to programmers by helping them easily trace the "cause" of an event. It did this by having users choose from a complex menu of contributing prior events at different levels of abstraction, and as output it yielded a single chain of actual events. I think it should be possible to do this for a class of similar event chains simultaneously, taking advantage of natural experiments that arise from similar sequences of program behavior to support probabilistic and/or conditional reasoning about causes of program misbehaviors. EAST-Env's database of "interesting" evaluation abstractions elicited from the user could also ease the programmer's burden in configuring the tool, since the abstraction database could serve as a model of user interest or user-relevance. The result could give programmers very high-level, project-specific visualizations of how program behaviors arise, with less time investment needed to configure, run, and interpret it.

Another interesting question remaining from my thesis relates to nested execution. EAST-Env was designed for cognitive modeling languages, many of which do not have anything equivalent to a "call stack". It would be interesting to see how programmers think about call stacks and nested structures, and extending the language I developed for cognitive modeling to support them would be a good starting place. I might start with formative work to understand how programmers go about constructing and debugging parsers, since this seems to be a challenge for some, and parser construction directly confronts interesting issues of how people think about nesting and recursion.

In the longer term, I would like to explore cross-pollination between cognitive modeling and communication theory (e.g. Rhetorical Structure Theory [TM98]), applied to reasoning about program behaviors, how programmers "forage" for information within programs and their behavior traces [PC99], and theories of how people reason with deduction, abduction, and induction [P78].

References

[BBD12] **Chris Bogart**, Margaret Burnett, Scott Douglass, Hannah Adams, Rachel White, "Designing a debugging interaction language for cognitive modelers: an initial case study in Natural Programming Plus", *Proceedings of SIGCHI Conference on Human Factors in Computing Systems* (CHI), 2012, pp. 2469-2478.

[BBD10] **Chris Bogart**, Margaret Burnett, Scott Douglass, David Piorkowski, Amber Shinsel, "Does my model work? Evaluation abstractions of cognitive modelers*". IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC), 2010, pp. 49-58.

[G92] T. R. G. Green. When Visual Programs are Harder to Read than Textual Programs. In G. C. van der Veer, M. J. Tauber, S. Bagnarola, and M. Antavolits, editors, Human-Computer Interaction: Tasks and Organisation, Proc. ECCE-6 (6th European Conference on Cognitive Ergonomics), number 1987. Rome, 1992.

[KM08] Andrew Ko and Brad Myers, "Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior," *International Conference on Software Engineering* (ICSE), 2008, pp. 301-310.

[LBB2013] Joseph Lawrance, **Chris Bogart**, Margaret Burnett, Rachel Bellamy, Kyle Rector, Scott D. Fleming, "How Programmers Debug, Revisited: An Information Foraging Theory Perspective," *IEEE Transactions on Software Engineering*, (accepted, to appear in 2013).

[P78] C. S. Peirce, "Illustrations of the Logic of Science: Deduction, Induction, and Hypothesis", Popular Science Monthly, vol 13, 1878, pp. 470-482.

[PC99] Peter Pirolli and Stuart K. Card, "Information foraging," *Psychological Review*, vol. 106, no. 4, 1999, pp. 643–675.

[PFS11] David Piorkowski, Scott Fleming, Chris Scaffidi, Liza John, **Chris Bogart**, Bonnie John, Margaret Burnett, and Rachel Bellamy, "Modeling Programmer Navigation: A head-to-head empirical evaluation of predictive models". *IEEE Symposium on Visual Languages and Human-Centric Computing* (VL/HCC) , 2011, pp. 109-116.

[PFS12] David Piorkowski, Scott Fleming, Chris Scaffidi, **Chris Bogart**, Margaret Burnett, Bonnie John, Rachel Bellamy, Calvin Swart, "Reactive Information Foraging: An empirical investigation of theory-based recommender systems for programmers", *Proceedings of SIGCHI Conference on Human Factors in Computing Systems* (CHI), 2012, pp. 1471-1480.

[PM06] John Pane and Brad A. Myers, "More Natural Programming Languages and Environments," in *End User Development, vol. 9 of the Human-Computer Interaction Series*, H. Liberman, F. Paterno, and V. Wulf, Eds. Dordrecht, The Netherlands: Springer, 2006, pp. 31–50.

[TM98] S. A. Thompson and W. C. Mann, "Rhetorical structure theory: Toward a functional theory of text organization," *Computers & Mathematics with Applications*, vol. 23, 1998, pp. 133–177.