More Natural End-User Software Engineering

Brad A. Myers, Andrew Ko, Sun Young Park, Jeffrey Stylos, Thomas D. LaToza, Jack Beaton
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
412-268-5150

bam@cs.cmu.edu http://www.cs.cmu.edu/~NatProg

ABSTRACT

The "Natural Programming" project at Carnegie Mellon University has been working for more than 10 years to make programming more "natural", or closer to the way people think. We have addressed the needs of all kinds of programmers: novices, professionals and end-user programmers. Many studies were performed which provided new insights and led to new models of programmers. From these insights and models, we created new programming languages and environments. Evaluations of the resulting systems have shown that they are effective and successful. This paper provides an overview of the entire 10-year Natural Programming project, but focuses on our new results since WEUSE-III in Dagstuhl.

Categories and Subject Descriptors

H5.2. Information Interfaces and Presentation – User Interfaces; D.2.6. Programming Environments. D.2.5 [Testing and Debugging]: Debugging aids, tracing. D.2.6 [Programming Environments]: Integrated environments.

General Terms

Design, Human Factors, Languages.

Keywords

Designer, Interactive Behaviors, Survey, Authoring, End-User Software Engineering, Natural Programming, Programming by Demonstration.

1. INTRODUCTION

The Natural Programming Project [33] has been applying human-computer interaction (HCI) techniques to develop and evaluate models and tools to help novice, professional, and "end-user" [32] programmers.

We started by studying how people think about programming concepts and algorithms [36, 38]. Participants in these studies did not know how to program, but they were familiar with a variety of computer applications. The goal was for the language to work in the way that people who do not have programming experience (novice programmers) expected. We then used these results to design a new programming language and environment [35, 37]. A summative study showed that it did help novice programmers create programs more easily [37].

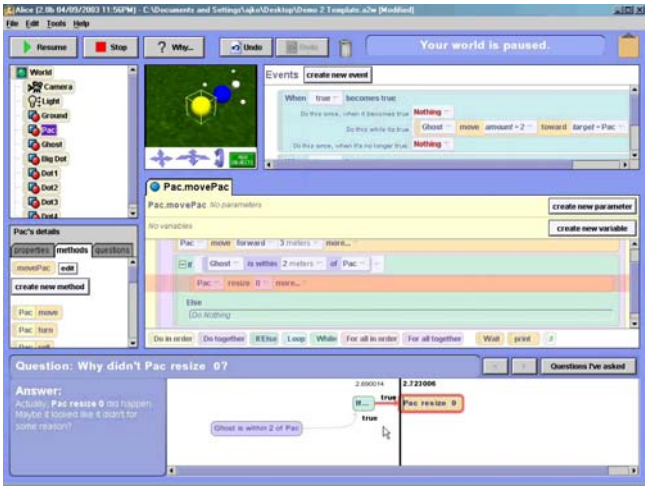

Figure 1: The original Whyline for Alice. The "Why" menu is at the top, and the time-line visualization of the answer is at the bottom. [19]

Next, we performed several studies of programming environment use. One study focused on novice programmers learning to use Visual Basic.NET [21]. Another looked at the influence of the programming environment on the types of errors that users of Alice [39] inserted into their code [18]. In another study, we investigated experienced programmers using Eclipse on several software maintenance tasks [22]. Our observations from these studies led to a number of design ideas for more helpful tools. For example, we designed the Whyline, which allows programmers to ask "Why" and "Why not" questions about their program's behavior, in order to help them better understand the causes behind their program's execution (see Figure 1) [19]. This led to the design of a similar tool, Crystal, which helped end users ask why questions about the behaviors of word processors, including the application's more complicated features, such as the "styles" mechanism and auto-correction [31]. We also created Mica, which helps programmers solve the vocabulary problem [13] by using Google to find example code and documentation from the words that the programmer can think of [44]. Our Jasper tool [9] allows programmers to select fragments of relevant code, and also

WEUSE IV'08, May 12, 2008, Leipzig, Germany.

choose other types of data from web pages and write notes about a task. All of this information is collected and presented in a single place, and saved in a single document, so that it can be returned to later and shared with coworkers who might also work on the task.

2. NEW WORK

In this workshop position paper, we focus on our new work, even though some of it is more related to professional programmers than EUP. Our earlier EUP systems are well-documented in other papers; those who are interested can see the citations above or our WEUSE-II overview [30]. The four topics on which we are currently working are how designers think about authoring behaviors, transitioning the Whyline to work for Java, studying how to make APIs more usable, and improving how people understand existing code.

2.1 Studying Designers

Designers are skilled at sketching and prototyping the look of interfaces, but to explore various behaviors (what the interface does) typically requires writing scripting code using Javascript, Flash or other programming tools. There have been many previous studies of the processes, techniques and tools that are used by designers, but none has focused on how the interactive behavior of the interface is created and communicated. We conducted field studies of 13 designers, and a web-based survey, which received 231 responses, to investigate the particular issues for the design of interactive behaviors. We particularly focused on people who are not programmers, but rather who are trained and work on Interaction Design, Graphics Design, Information Architecture, Experience Design, Visual Design, User Interface Design, or equivalent.

Many of our findings confirmed what others have reported in previous surveys, for example that designers prefer to start by sketching (about 97% in our survey), and most designers (88% in our survey) also use storyboards.

However, we did find some interesting new results that have not been previously reported:

- By a large margin, the participants in our survey agreed that prototyping the behaviors was more difficult than the design of the appearance (86% said prototyping the behaviors was more difficult).
- Sketches and storyboards cannot adequately convey the behaviors by themselves, so designers must augment them with annotations such as arrows and many textual descriptions of the desired behaviors (see Figure 2).
- The purpose of implementing the interactive behaviors, and for the annotations on the pictures, is often primarily to serve as documentation and specifications for others. Almost all designers worked in teams, and communicating with others is a key part of their jobs. Communicating the design of behaviors to developers was reported to be difficult by 40% of the designers in our study.
- The behaviors that the designers wanted were quite complex and diverse, beyond what could plausibly be provided by a system that provided only a few built-in behaviors or a selection of predefined widgets, and therefore seemingly requires full programming capabilities.

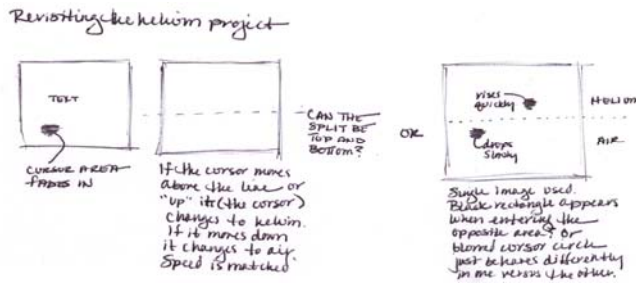

Figure 2: Sketches by a contextual inquiry participant showing two different options being investigated for an interaction, with lines and textual annotations to explain what is intended.

- As reported for other kinds of design [6], in our survey, the designers agreed that the design of interactive behaviors emerge through the process of exploration. In other words, the designers do not have a final conception of the behavior before they start. However, whereas iterating on the look of the interface can be easily done by sketching, designers felt it difficult to iterate on the behavior. Today's authoring tools make it difficult or impossible to have two implementations of behaviors side-by-side to compare them, and even keeping around and reverting to old versions of code is difficult.

We are currently designing a new study to evaluate how designers express the low-level components of behaviors. We have constructed a Flash program that shows various primitive animations, such as an object disappearing, moving, changing colors, responding to the mouse, etc. We will ask designers to describe what happens in their own words. In previous studies [36, 38], this kind of exploration has revealed interesting data about how people naturally express and think about these behaviors.

2.2 Whyline for Java

Our research showed that virtually all debugging sessions start with asking "Why" and "Why Not" questions – why something did something, and in about 60% of the cases, why something did *not* do something [18]. Our initial prototype tool, called the "Whyline" (see Figure 1) [19] allowed programmers to pop up menus to ask "Why" and "Why Not" questions in the Alice programming environment for kids [39]. In lab studies, the Whyline decreased the time to find and fix bugs by a factor of eight, and increased programmer productivity by 40%.

We have since developed a version of the Whyline in Java [20] and found similar success. In a study comparing rank novice programmers to experienced programmers, novices were able to complete a debugging task significantly faster, describing the work "like a treasure hunt" and saying "It was fun! I didn't know debugging was like this." In a second study comparing experts using the Java Whyline to experts using a conventional debugger, Whyline users completed tasks twice as fast and with significantly higher rates of success, saying, "My god, this is so cool" and "When can I get this for C?"

Although the Whyline for Java is designed explicitly for experienced Java developers, many of the issues of scale and generality that we have addressed in the design will influence the design of a

Whyline tool for simpler languages meant for domain experts. For example, we have found that illustrating program execution by annotating code is perhaps even more effective than listing events in a timeline. This calls for rich, interactive code editors in EUP tools and not just plain text. We have also developed algorithms and data structures that allow recording, asking, and answering to work at interactive speeds. To support Whyline questions, these techniques need to be integrated into the runtime environments for EUP languages. We have also generalized the notion of "program output" and found that questions about output in a particular medium requires a careful design customized to the form of output—intuitively asking questions about rendered graphics is quite different than asking about text printed to a console. Since many end-user programming languages produce specific kinds of domain output, it will be important to customize the question-asking experience to these domains and users' expert knowledge.

2.3 API Usability

Most of programming today involves using complex software libraries, toolkits, software development kits (SDKs), frameworks and other application programming interfaces (APIs). This is equally true of novice, professional and end-user programmers. In fact, most EUP code serves to control and glue together high-level complex operations. However, to use an API, the programmer must first understand how the various functions work, and in particular, how they work together. Our research has shown that this is a significant barrier [21]. Making it even more difficult is that most APIs have not been designed with usability as a design goal. Designers of APIs have many competing goals and many design decisions to make [43]. We have begun a series of studies to investigate how to make APIs more usable, and to develop models and new tools to help with API use.

In the first study [42], we compared objects with "required parameters" that must be supplied when an object is created, versus having "default constructors" with no parameters that just create an empty object. We found the surprising result that, contrary to the intuition of experienced API designers, all of the kinds of programmers we studied were more effective at using objects that did *not* have required parameters. Some API designers had asserted that having required parameters on constructors would make it clearer to users that instances are not valid until those parameters were provided. Our second study [12] showed that APIs that used "factory" classes instead of constructors to create objects (a pattern recommended by API designers [4, 10] and software engineers [14]) came with severe usability disadvantages not previously documented. Our current study is looking at how programmers investigate documentation to find classes to implement a desired function, and how object hierarchies and namespaces can be designed to increase usability. Along with recommendations for the design of new APIs, we will also create new tools and documentation techniques to increase the usability of existing APIs.

In a collaborative project with SAP (the third largest software company in the world), we examined the usability of their "BRFPlus" business rules framework API, and used our observations to create a prototype for a new version of the API. The business rules framework is used by programmers, but it is also used by business experts who are not programmers. The business rules framework comes with GUI tools with which business experts create rules that are end-user programs, specifying case-logic and procedural behavior. As business rules systems become more popular, business experts are becoming an increasingly important class of EUP. There remains much work to be done in improving the usability of the business rule tools that these business experts use.

Inspired by these results SAP has asked us to investigate the usability of their new APIs for Enterprise Service Oriented Architecture (E-SOA). These APIs may be used by professional programmers, but the eventual goal includes allowing business process experts to create their own E-SOA applications through EUP.

2.4 Understanding Code

A significant portion of all programmers' time in the real world is spent trying to understand how code works. Surprisingly little is known about what programmers do during this time, or what tools would help. A new series of studies by our group [17, 24, 26] and elsewhere [29, 40, 41] are showing that these investigations start with questions that programmers want to answer (e.g., "under what conditions does this code need to be called?") and result in a set of facts that the programmer must keep track of (e.g., "this code must be called each time the cursor moves"). Developers seek, learn, critique, and explain these facts to generate proposals of what to change and reject proposals which violate perceived constraints.

One interesting new result is that a number of questions about what a program is doing are related to *update paths* [25]. An "update path" is a sequence of method calls and field assignments which link a trigger application state change to an effect application state change. Developers navigate long sequences of control and data flow relationships to discover these rules. Today's tools make this difficult. For example, our research shows that when programmers are trying to determine what code caused a particular program behavior, they often search the code for a keyword they think might be related, but that such guesses led to relevant code in only 12% of searches [22]. Existing development environments provide static views (e.g., Eclipse's call hierarchy tree of callers or callees) and dynamic views (e.g., the call stack of the current execution). Our study revealed numerous ways in which using the static call hierarchy to answer update path questions resulted in wasted time, inferior changes, and bugs. These problems occurred because the tools do not filter by trigger, show infeasible update paths, do not show how update paths interact, and do not show class structure or fields. Dynamic views, such as the Whyline or the Eclipse debugger, show only a single execution path rather than all feasible paths, and are not a reliable way to discover *all* relevant effects. In typical GUI code, it is common for there to be large switch statements, such as branching based on the event type, so our static analyses that can propagate values (such as which kind of event), can provide a display that is significantly pruned from the full call graph, and therefore much more useful and understandable. From these insights, we are creating new tools to address these problems.

3. RELATED WORK

Of course, for all of these projects, there is significant work by others that is related, but there is only room here for a brief overview. Please see the related work sections of our other papers for more.

In the area of studying designers, it long been well known that designers prefer sketching for early phases of design [6, 34, 46]. Other surveys have shown that designers make extensive use of informal tools [34] and storyboards [11]. Many research tools have been created to help designers with sketching and authoring behaviors, for example, SILK [23], DENIM [34], DEMAIS [2] and Designer's Outpost [16].

The Whyline for Java builds on more than half a century of research on debuggers [27], including recent work such as Abraham and Erwig's goal-directed debugging [1], which allows a developer to choose a wrong value in a spreadsheet and specify the correct value. The analyses that the Whyline uses are based on static and dynamic program slicing [3].

The API usability work was directly inspired by usability studies of specific APIs done at Microsoft [7, 8] and elsewhere [5, 28]. Our approach instead focuses more on patterns used by many APIs, so the results will be more directly generalizable.

Finally, there is an enormous literature on reverse engineering and code understanding (e.g. [22, 40, 45]). All show differences between novices and experts, and some have documented questions that programmers investigate [41, 45]. New tools have been based on these results, such as Mylar [15] which presents relevant code and methods to help with navigation and selection.

4. CONCLUSIONS

The Natural Programming project has followed a human-centered approach to software engineering. This approach of designing from data about what people do and what is natural for people, has resulted in new knowledge, models and tools that are relevant for novice, professional, and end-user programmers.

Although we have described multiple projects led by different people, they are all related. A future system should combine the dynamic analysis of the Whyline with the static analysis of update paths to produce a more comprehensive debugging and understanding environment, since these tasks are often linked. New tools for designers will require APIs that are quite usable, along with a good debugging system such as those in the Whyline. The models we have developed in all projects about how programmers work is relevant to future design of all tools.

We look forward to discussing our results with other members of the WEUSE-IV workshop.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under NSF grant IIS-0329090 and the EUSES consortium under NSF grant ITR CCR-0324770. Additional support has come from IBM, SAP and Adobe.

REFERENCES

[1] Abraham, R. and Erwig, M. "Goal-Directed Debugging of Spreadsheets," in *VL/HCC 2005.* September 20-24, 2005. Dallas, TX: pp. 37-44.

[2] Bailey, B.P., Konstan, J.A., and Carlis, J.V. "Supporting Multimedia Designers: Towards more Effective Design Tools," in *8th International Conference on Multimedia Modeling.* 2001. pp. 267-286.

[3] Baowen, X., Ju, Q., Xiaofang, Z., Zhongqiang, W., and Lin, C., "A brief survey of program slicing." *SIGSOFT Software Engineering Notes*, 2005. **30**(2): pp. 1-36.

[4] Bloch, J., *Effective Java Programming Language Guide.* 2001, Boston, MA: Addison-Wesley.

[5] Bore, C. and Bore, S., "Profiling software API usability for consumer electronics." *Consumer Electronics*, 2005.

[6] Buxton, B., *Sketching User Experiences: Getting the Design Right and the Right Design.* 2007, San Francisco, CA: Morgan Kaufmann.

[7] Clarke, S., "Measuring API Usability." *Dr. Dobbs Journal*, May, 2004. pp. S6-S9.

[8] Clarke, S. "Describing and Measuring API Usability with the Cognitive Dimensions," in *Cognitive Dimensions of Notations 10th Anniversary Workshop.* 2005. www.cl.cam.ac.uk/~afb21/CognitiveDimensions/workshop2005/Clarke_position_paper.pdf.

[9] Coblenz, M.J., Ko, A.J., and Myers, B.A. "JASPER: An Eclipse Plug-In to Facilitate Software Maintenance Tasks," in *Eclipse Technology eXchange (ETX) Workshop at OOPSLA 2006.* October 22-23, 2006. Portland, Oregon: pp. 65-69.

[10] Cwalina, K. and Abrams, B., *Framework Design Guidelines.* 2005, Upper-Saddle River, NJ: Addison-Wesley.

[11] Davis, R.C. and Landay, J.A. "Informal Animation Sketching: Requirements and Design," in *AAAI 2004 Fall Symposium on Making Pen-Based Interaction Intelligent and Natural.* October 21-24, 2004. pp. 42-48.

[12] Ellis, B., Stylos, J., and Myers, B. "The Factory Pattern in API Design: A Usability Evaluation," in *International Conference on Software Engineering (ICSE'2007).* May 20-26, 2007. Minneapolis, MN: pp. 302-312.

[13] Furnas, G.W., Landauer, T.K., Gomez, L.M., and Dumais, S.T., "The vocabulary problem in human-system communication." *Commun. ACM*, 1987. **30**(11): pp. 964-971.

[14] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns.* 1995, Reading, MA: Addison-Wesley.

[15] Kersten, M. and Murphy, G.C. "Mylar: a degree-of-interest model for IDEs," in *AOSD '05: 4th international Conference on Aspect-Oriented Software Development.* March 14 - 18, 2005. Chicago, Illinois: pp. 159-168.

[16] Klemmer, S.R., Thomsen, M., Phelps-Goodman, E., Lee, R., and Landay, J.A. "Where do web sites come from? capturing and interacting with design history," in *Proceedings of CHI'2002: the SIGCHI conference on Human factors in computing systems.* 2002. Minneapolis, Minnesota: pp. 1-8.

[17] Ko, A.J. and DeLine, R. "A Field Study of Information Needs in Collocated Software Development Teams," in *International Conference on Software Engineering (ICSE'2007).* May 20-26, 2007. Minneapolis, MN:

[18] Ko, A.J. and Myers, B.A. "Development and Evaluation of a Model of Programming Errors," in *IEEE Symposium on End-User and Domain-Specific Programming (EUP'03), part of the IEEE Symposia on Human-Centric Computing Languages and Environments.* October 28-31, 2003. Auckland, New Zealand: pp. 7-14.

[19] Ko, A.J. and Myers, B.A. "Designing the Whyline, A Debugging Interface for Asking Why and Why Not questions about Runtime Failures," in *Proceedings CHI'2004: Human Factors in Computing Systems.* April 24-29, 2004. Vienna, Austria: pp. 151-158.

[20] Ko, A.J. and Myers, B.A. "Debugging, Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior," in *ICSE'2008: 30th International Conference on Software Engineering.* 10 - 18 May, 2008. Leipzig, Germany: pp. To appear.

[21] Ko, A.J., Myers, B.A., and Aung, H.H. "Six Learning Barriers in End-User Programming Systems," in *IEEE Symposium on Visual Languages and Human-Centric Computing.* September 26-29, 2004. Rome, Italy: pp. 199-206.

[22] Ko, A.J., Myers, B.A., Coblenz, M., and Aung, H.H., "An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks." *IEEE Transactions on Software Engineering*, Dec, 2006. **33**(12): pp. 971-987.

[23] Landay, J. and Myers, B., "Sketching Interfaces: Toward More Human Interface Design." *IEEE Computer*, March, 2001. **34**(3): pp. 56-64.

[24] LaToza, T.D., Garlan, D., Herbsleb, J.D., and Myers, B.A. "Program comprehension as fact finding," in *ESEC/FSE 2007: ACM SIGSOFT Symposium on the Foundations of Software Engineering.* September 3-7, 2007. Dubrovnik, Croatia: pp. 361-370.

[25] LaToza, T.D. and Myers, B.A., "How Developers Reason about Update Paths." *Submitted for publication*, 2008.

[26] LaToza, T.D., Venolia, G., and DeLine, R. "Maintaining Mental Models: A Study of Developer Work Habits," in *Proceedings of the International Conference on Software Engineering (ICSE'2006).* 2006. Shanghai, China: pp. 492 - 501.

[27] Lieberman, H., "The Debugging Scandal and What to Do About It." *CACM*, April, 1997. **40**(4): pp. 26-78. Special Issue.

[28] McLellan, S.G. and Roesler, A.W., "Building More Usable APIs." *IEEE Software*, 1998. **15**(3): pp. 78-86.

[29] Murphy, G.C., Kersten, M., and Findlater, L., "How are Java software developers using the Eclipse IDE?" *IEEE Software*, Jul/Aug, 2006. pp. 76-83.

[30] Myers, A.J.K.B.A., Coblenz, M.J., and Stylos, J. "End-User Programming Productivity Tools," in *The Next Step: From End-User Programming to End-User Software Engineering (WEUSE II) at CHI'2006.* April 23, 2006. Montreal, Canada: http://www.cs.cmu.edu/~ajko/papers/Ko2005ProductivityTools.pdf.

[31] Myers, B., Weitzman, D.A., Ko, A.J., and Chau, D.H. "Answering Why and Why Not Questions in User Interfaces," in *Proceedings CHI'2006: Human Factors in Computing Systems.* April 22-27, 2006. Montreal, Canada: pp. 397-406.

[32] Myers, B.A., Ko, A.J., and Burnett, M.M. "Invited Research Overview: End-User Programming," in *Extended Abstracts, CHI'2006.* April 22-27, 2006. Montreal, Canada: pp. 75-80.

[33] Myers, B.A., Pane, J.F., and Ko, A., "Natural Programming Languages and Environments." *CACM*, Sept, 2004. **47**(9): pp. 47-52.

[34] Newman, M.W. and Landay, J.A. "Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice," in *Designing Interactive Systems, DIS 2000.* August, 2000. New York City: pp. 263-274.

[35] Pane, J., *A Programming System for Children that is Designed for Usability.* PhD Thesis, Computer Science Department, Carnegie Mellon University, 2002, Pittsburgh, PA. http://www.cs.cmu.edu/~pane/thesis/. Computer Science Technical Report CMU-CS-02-127.

[36] Pane, J.F. and Myers, B.A. "Tabular and Textual Methods for Selecting Objects from a Group," in *Proceedings of VL 2000: IEEE International Symposium on Visual Languages.* September 10-13, 2000. Seattle, WA: IEEE Computer Society. pp. 157-164.

[37] Pane, J.F. and Myers, B.A. "The Impact of Human-Centered Features on the Usability of a Programming System for Children," in *CHI.* Apr 1-6, 2002. Minneapolis, MN: pp. 684-685. http://www-2.cs.cmu.edu/~pane/research.html. Extended Abstracts for CHI'2002.

[38] Pane, J.F., Ratanamahatana, C.A., and Myers, B.A., "Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems." *International Journal of Human-Computer Studies*, February, 2001. **54**(2): pp. 237-264. http://www.cs.cmu.edu/~pane/IJHCS.html.

[39] Pausch, R., Burnette, T., Capehart, A.C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S., and White, J., "Alice: A Rapid Prototyping System for 3D Graphics." *IEEE Computer Graphics and Applications*, 1995. **15**(3): pp. 8-11. May.

[40] Robillard, M.P., Coelho, W., and Murphy, G.C., "How Effective Developers Investigate Source Code: An Exploratory Study." *IEEE Transactions on Software Engineering*, December, 2004. **30**(12): pp. 889-903.

[41] Sillito, J., Murphy, G.C., and De Volder, K. "Questions programmers ask during software evolution tasks," in *SIGSOFT'06/FSE-14: Proceedings of the 13th ACM SIGSOFT and 14th international symposium on Foundations of Software Engineering.* 2006. Portland, Oregon: pp. 23 - 34.

[42] Stylos, J. and Clarke, S. "Usability Implications of Requiring Parameters in Objects' Constructors," in *International Conference on Software Engineering (ICSE'2007).* May 20-26, 2007. Minneapolis, MN: pp. to appear. Submitted for Publication.

[43] Stylos, J. and Myers, B. "Mapping the Space of API Design Decisions," in *2007 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'07.* Sept 23-27, 2007. Coeur d'Alene, Idaho: pp. 50-57.

[44] Stylos, J. and Myers, B.A. "Mica: A Programming Web-Search Aid," in *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'06.* Sept 4-8, 2006. Brighton, UK: pp. 195-202.

[45] Vans, A.M., Mayrhauser, A.v., and Somlo, G., "Program Understanding Behavior during Corrective Maintenance of Large-Scale Software." *Int'l J. Human-Computer Studies*, July, 1999. **51**(1): pp. 31-70.

[46] Wong, Y.Y. "Rough and Ready Prototypes: Lessons from Graphic Design," in *Extended Abstracts, SIGCHI'92.* May, 1992. Monterey, CA: pp. 685.