What Would Other Programmers Do? Suggesting Solutions to Error Messages

Björn Hartmann
Stanford University HCI Group
Computer Science Department
Stanford, CA 94305
bjoern@cs.stanford.edu

ABSTRACT
Interpreting compiler error messages is challenging for novice programmers. Presenting examples how other programmers have corrected similar errors may help novices understand and correct such errors. This paper introduces *HelpMeOut*, a social recommender system that aids debugging by suggesting solutions that peers have applied in the past. HelpMeOut comprises IDE instrumentation to collect examples of code changes that fix compiler errors; a central database that stores fix reports from many users; and a suggestion interface that, given an error, queries the database and presents relevant fixes to the programmer.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: debugging, recommender systems

INTRODUCTION
The web has fundamentally changed how programmers write and debug code. For example, more code gets written by opportunistically modifying found examples [1]. By and large, our authoring tools have not caught up to this change — most still make the assumption that writing and debugging code happens individually, in isolation. As a result, most social exchanges around programming tasks happen through online forums and blogs. We believe that there is significant latent value in integrating social aspects of development and debugging directly into authoring tools.

As a step into the direction of social authoring tools, this paper introduces *HelpMeOut*, a social recommender system that aids novices with the debugging of compiler error messages by suggesting successful solutions to the errors that other programmers have found. It is known that novice programmers have difficulty interpreting compiler errors [6]. We hypothesize that presenting relevant solution examples makes it easier for novices to interpret and correct error messages, as example modification has a lower expertise barrier for end-users than creation from scratch [7].

UIST'09, October 4–7, 2008, Victoria, BC, Canada.
ACM xxxxxxxxxx/xx/xxxx.

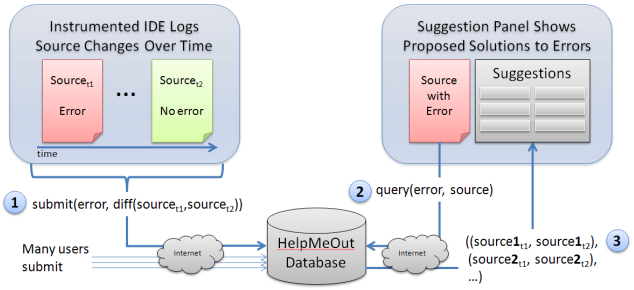

Figure 1. HelpMeOut suggests corrections to compile time errors based on IDE instrumentation.

The HelpMeOut system realizes example-centric debugging by augmenting an existing development environment (IDE). HelpMeOut comprises three components (Figure 1): ❶ An *observation component* that tracks code evolution over time, collecting modifications that take source code from an error state to an error-free state (a "fix"). ❷ An *online database* which stores fixes and which can be queried for most relevant examples, given a compile time error and code context. ❸ A *suggestion interface* inside an IDE that, given a compiler error, queries the online database and then presents a list of possible fixes to the user.

SCENARIO
The following short scenario demonstrates how HelpMeOut can aid users: Jim works on a visual animation that displays a graphical sprite following his mouse. In his code, he incorrectly initializes a variable array:

```
float[] x = new float[];
```

When trying to compile his code he receives the error message "*Variable must provide either dimension expressions or an array initializer*." Not sure what either of the two options are, he consults the HelpMeOut suggestion panel (Figure 3). Looking over the suggestions, he sees that he can either add a size to the right-hand side of his variable intialization, or provide explicit values. He copies the second suggested fix, and pastes the line into his code.

ARCHITECTURE AND IMPLEMENTATION
We are currently implementing the HelpMeOut architecture by augmenting the Processing multimedia programming environment. Processing, based on Java, was chosen because of its growing popularity as an introductory teaching tool and a hobbyist programming environment.

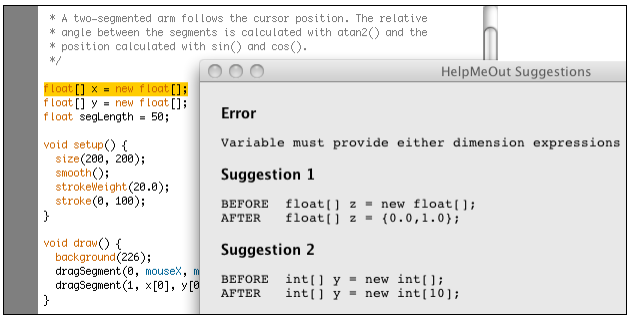

Figure 2. The HelpMeOut suggestion panel shows possible corrections for a reported compiler error.

To automatically collect examples of compiler errors and fixes, HelpMeOut monitors return codes from the Processing compiler. If compilation fails with an error, the error message and a snapshot of the source are saved. If subsequently, a compilation succeeds again, a diff report comparing the initial error state and the error-free state is generated (Figure 3). The initial error message and the diff report are sent to the remote HelpMeOut database.

The database of example fixes has to be reachable from many individual users' machines, store submitted reports, and return related fixes. In our prototype, we implement a web service based on Apache and Python CGI scripts to respond to remote procedure calls. Whenever a compilation fails with an error, HelpMeOut generates a query to this database to retrieve related fixes, based on the error message as well as the code referenced by the error.

What is a good related fix? First, the error messages of query and database entry have to match. From that set of candidate fixes, which are most relevant? We hypothesize that a fix is relevant if the source code of the broken state in the fix contains a line that is as close as possible to the source line referenced by the error in the query. HelpMeOut passes source code through a lexical analyzer to replace identifiers, literals, and comments, as these are highly variable across difference programs. HelpMeOut then computes the edit distances between tokenized query error and candidate fixes. Low edit distances are considered good matches.

To increase the likelihood of finding a useful suggestion, each query returns an n-best list of results. This list is then visualized in a separate pane inside the user's IDE. The visualization juxtaposes before (with error) and after (without error) states of the code. We are working on techniques to merge the found example into the user's code.

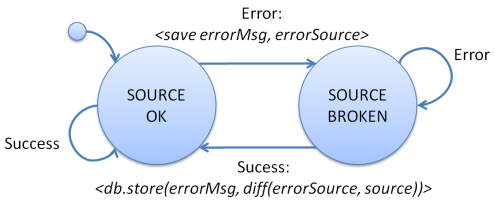

Figure 3. Final state machine for calculating and storing error fix reports to the HelpMeOut database.

RELATED WORK

Debugging by novices has been well-studied in the Computer Science Education community. For a recent survey, see [6]. Ko's WhyLine [4] is notable for its framing of debugging as a cognitive activity of asking and answering a set of "why" and "why not" questions. *Bug detection* is an active research area in software engineering. BugMem [3] finds and correct bugs based on data mining of team source repositories. Liblit et al. [5] instrument application binaries to collect statistical data of runtime behavior on a central server. Prior research in *application instrumentation* has investigated how to extract usage and usability data from user interface events. For a survey, see [2]. Terry et al. recently instrumented an open source image manipulation program to collect usage information [8].

EVALUATION PLAN AND FUTURE WORK

The main questions our evaluation seeks to answer are:

1. What is the scope of errors that can be detected and corrected by HelpMeOut?
2. How do peer-generated suggestions compare to looking up compiler errors using web search and to structured editors that prevent some errors from occurring at all?
3. Can presenting examples of compiler errors aid programmer understanding of error messages?
4. Can we quantify how large the example corpus needs to be to provide sufficient coverage or the error space?

We believe that the general approach of automatically collecting usage data, aggregating data over many users, and then suggesting actions based on that data has wider applicability beyond the realm of compiler errors. We are currently working on suggesting corrections for runtime errors and conventions of commonly used code patterns.

REFERENCES

1. Brandt, J., et al. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. *CHI 2009*, 1589-1598.
2. Hilbert, D.M. and Redmiles, D.F. Extracting usability information from user interface events. *ACM Comput. Surv. 32*, 4 (2000), 384-421.
3. Kim, S., Pan, K., and E. E. James Whitehead, J. Memories of bug fixes. *SIGSOFT FSE 2006*, 35-45.
4. Ko, A.J. and Myers, B.A. Debugging reinvented: asking and answering why and why not questions about program behavior. *ICSE 2008*, 301-310.
5. Liblit, B., et al. Scalable statistical bug isolation. *PLDI 2005*, 15-26.
6. McCauley, et al. Debugging: A Review of the Literature from an Educational Perspective. *Computer Science Education 18*, 2 (2008), 67-92.
7. Nardi, B.A. *A small matter of programming*. MIT Press, 1993.
8. Terry, M., Kay, M., Vugt, B.V., Slack, B., and Park, T. Ingimp: introducing instrumentation to an end-user open source application. *CHI 2008*, 607-616.