Crowdsourcing Suggestions to Programming Problems for Dynamic Web Development Languages

Dhawal Mujumdar
School of Information
University of California, Berkeley
102 South Hall
Berkeley, CA 94720
dhawal@ischool.berkeley.edu

Manuel Kallenbach
Computer Science Department
RWTH Aachen
Lehrstuhl Informatik 10
52056 Aachen, Germany
manuel.kallenbach
@rwth-aachen.de

Brandon Liu
Computer Science Division
University of California, Berkeley
387 Soda Hall
Berkeley, CA 94720
bran_liu@berkeley.edu

Björn Hartmann
Computer Science Division
University of California, Berkeley
629 Soda Hall
Berkeley, CA 94720
bjoern@eecs.berkeley.edu

Copyright is held by the author/owner(s).
CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
ACM 978-1-4503-0268-5/11/05.

Abstract
Developers increasingly consult online examples and message boards to find solutions to common programming tasks. On the web, finding solutions to debugging problems is harder than searching for working code. Prior research introduced a social recommender system, HelpMeOut, that crowdsources debugging suggestions by presenting fixes to errors that peers have applied in the past. However, HelpMeOut only worked for statically typed, compiled programming languages like Java. We investigate how suggestions can be provided for dynamic, interpreted web development languages. Our primary insight is to instrument test-driven development to collect examples of bug fixes. We present Crowd::Debug, a tool for Ruby programmers that realizes these benefits.

Keywords
Debugging, Recommender Systems

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces – Training, Help, and Documentation.

General Terms
Design, Human Factors

المنارة للاستشارات

www.manaraa.com

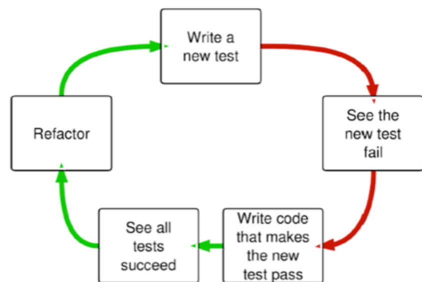

Figure 1: In test-driven development, tests are written before implementation code. Monitoring changes in test status enables Crowd::Debug to identify potential bug fixes.

Introduction

Understanding and successfully correcting program bugs is challenging. This is especially true for dynamically typed, interpreted languages that are popular for web development. Such languages often lack good analysis and debugging tools. Many programmers turn to web resources such as the Q&A site Stack Overflow for advice [7]. However, online sites and web search are often better at providing code examples for new functionality than for finding solutions to project-specific bugs. Reasons include the difficulty of searching for structurally similar code using engines optimized for plain text queries; and low incentives for developers to read others' broken code.

One way of sidestepping such problems is to collect examples of bug fixes without requiring programmers to explicitly publish such information, by observing the actions of programmers in their development environments. In particular, HelpMeOut [4] introduced the concept of crowdsourced bug fix suggestions: IDE instrumentation tracks compiler output and saves code changes that take a project from a broken to a fixed state into a shared database. To date, the HelpMeOut approach only applies to statically typed languages. Many web developers do not use such languages, but instead rely on dynamically typed, interpreted languages like Ruby and Python. In these languages, little to nothing is known about what type variables will have before execution. Thus, most program errors only occur at run-time, circumventing HelpMeOut's main strategy for detecting problems.

Our research investigates whether crowdsourced bug fix suggestions can be generated for dynamically typed languages, where they could be especially beneficial.

We are developing Crowd::Debug, a development tool that collects and displays bug fix suggestions for the Ruby programming language. *The primary insight behind Crowd::Debug is to leverage test-driven development (TDD) as the source of bug fix examples.* In TDD, tests are written before functions are implemented and implementation code is only written or corrected when there are failing tests (see Figure 1). Failing tests thus indicate both missing functionality and incorrect program behavior.

We implement Crowd::Debug for Ruby (and specifically, Ruby on Rails) because of its popularity for web development. Among Ruby programmers, test-driven development is widely practiced: In a survey[1] of 4000 Ruby programmers, more than 85% use methods of test-driven development. Adhering to TDD offers reasonable expectations of code correctness and robustness, in the absence of good static analysis tools.

The rest of this paper is organized as follows: We first present a review of related work and a scenario that demonstrates the benefit of our tool. We then discuss the implementation of our prototype, and findings from initial user studies. We conclude with an outlook to future work.

Related Work

Prior research falls into three areas: automated debugging tools, help for locating relevant resources, and help for understanding the causes of bugs.

[1] http://survey.hamptoncatlin.com/survey/stats

المنارة للاستشارات
www.manaraa.com

Suggestions:

B

Suggestions 1

./app/controllers/topics_controller.rb

```
def dashboard
+  t = Topic.all
+  @titles = ""
+  t.each do |topic|
+    @titles << t.title
+  end
end
```

Suggestions 2

./app/controllers/topics_controller.rb

```
def dashboard
+  t = Topic.all
+  t.select{ |topic| topic.active? }
end
```

Figure - 2A: Crowd::Debug shows the exception, backtrace, and code suggestions for a failed test. **2B**: Close-up of code suggestions.

Automated debugging tools attempt to fix code with minimal user involvement. ReAssert, a plugin for the Eclipse Java IDE, helps developers fix failed unit tests [3]. ReAssert does not alter the source code in order to fix the bug; rather it tries to alter the test. BugFix [5] suggests possible solutions to programming errors from a knowledge base. Machine learning techniques are used to improve the suggestions for a given bug. Once a bug is fixed, developers are able to enter a new bug fix description into the knowledge base, but have to do so manually.

Code-specific search tools help developers **locate relevant resources**: Blueprint [2] integrates web search for code examples into the FlexBuilder IDE. Blueprint enhances query terms and adds links to copied source code indicating its origin on the web. DebugAdvisor [1] allows developers to search for information related to a bug with a fat query consisting of additional context of the bug. This query can include natural language text, core dumps, and debugger output. Results are retrieved from software repositories, logs, and bug databases.

Tools for novice or end-user programmers emphasize **understanding root causes** of program bugs. Ko's Whyline [6] allows programmers to ask "Why did" and "Why didn't" questions about their program's output. Backstop [8] aids novice programmers in fixing runtime errors of Java programming language by providing more user-friendly error messages when an uncaught exception occurs.

Crowd::Debug presents examples of previous bug fixes related to the corresponding failing test. It provides code fixes in form of suggestions that other programmers have applied to closely related errors in a test-driven development environment. In contrast to ReAssert, Crowd::Debug does not alter any of the tests in order to successfully pass them.

Scenario: Working with Crowd::Debug

Jane, a programmer interested in web application development, wants to write a 'dashboard' for her blogging application. She writes a basic controller test: "*GET dashboard should show a list of all the recent topics*". She runs her test suite and the Crowd::Debug interface is displayed, which shows that one test failed, with the exception "*NoMethodError: You have a nil object when you didn't expect it! The error occurred while evaluating nil.each*" Normally she would search the web for this error message, but Crowd::Debug suggests two fixes.

Both fixes show examples of code where *Topic.all* is called instead of *Topic.first* (see Figure 2), indicating a distinction between an element and a list of elements. This distinction would have been caught at compile time in Java, but in Ruby, such exceptions are not caught until the program is run. Jane implements the suggested change, and then runs the test suite again to see that all tests pass.

The Crowd::Debug Prototype

Our prototype is a web application that works on the client-server model. Crowd::Debug augments the existing test driven development process. Figure 3 describes the 3 main components:

المنارة للاستشارات
www.manaraa.com

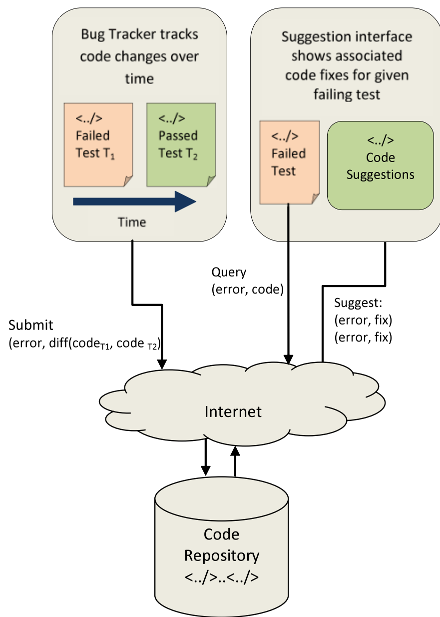

Figure 3: In the Crowd::Debug architecture, an augmented test framework on the client side is used to collect fixes, and to query for fixes. A central server stores all fixes for many users.

Token	Replacement string	Example
VarNameToken	INST_VAR / CONST / VAR	@foo → INST_VAR Foo → CONST foo → VAR
MethNameToken	METHOD	def hello → def METHOD
StringToken	STRING	"hello" → STRING
SymbolToken	SYMBOL	:hello → SYMBOL
NumberToken	NUMBER	42 → NUMBER

Figure 4: Examples of substitutions performed by the lexical analyzer to generalize code.

A **bug tracker** collects information related to source code in both error and error-free state.

The **code repository** stores source code associated with errors and their test cases in a database. The code repository returns code suggestions based on an exception message, and the associated file that caused the exception.

The **suggestion interface** shows code fixes associated with the failing tests that are returned by the code repository. The suggestion interface also shows a backtrace of the program, exception messages and the location of the tests in the user's file system.

Implementation

The suggestion interface is implemented by extending RSpec[2], a popular Ruby testing framework. RSpec displays results for each test on the console when the test suite is run. Crowd::Debug instead shows this information in a web browser and adds suggestions. Multiple fixes are displayed vertically (see Figure 2).

Crowd::Debug's bug tracker sends the result of each test to the server. If the test fails, the bug tracker records information of the test exception. The bug is assigned an identifier (the name of the failing test), and the exception message, stack trace, and the state of all source files are recorded. File state is recorded using the Git version control system[3]. Stack trace information is helpful, since the exception may have been thrown deep inside a library or the Rails framework. When the same test passes later, the server is notified and the state of the source code is recorded to create a pair of broken and working code fragments.

In order to facilitate matching of code between different projects, project-specific tokens, such as class names and string literals, are replaced with general placeholders such as CONST or STRING (see Figure 4). This generalization is performed for all code fragments during matching. Normalized Levenshtein distance between strings is used to compare both the exception messages and the stack traces with existing records in the database. We determined that weighing the exception message's distance twice as much as the source code gave good results. The five most similar source matches are then returned to the user.

Prototype Evaluation

To evaluate the utility of our tool and determine the relevance of the system's suggestions, we conducted an initial user study with 8 users.

We asked our participants to write programs for two applications: an interactive quiz and a blog backend. Those implementing a blogging application were given an abstract task description such as: "*A get request to the index action should render the index view*". Participants had to first write a test for the intended functionality, and then implement it. This particular task was chosen since in practice, different programmers would naturally write different implementations. We also asked our participants not to use code generation and to strictly follow test-driven development practices.

[2] http://rspec.info

[3] http://git-scm.com

المنارة للاستشارات
www.manaraa.com

The authors first performed the task themselves to seed the database with fixes. Our user studies resulted in 8 person-hours of data.

Results

We used screen capture to record all sessions. We then counted the number of useful suggestions that were presented by Crowd::Debug. We considered a suggestion to be useful if its application led to the success of the failed test.

Our user studies generated a total of 161 fixes. Participants queried Crowd::Debug 211 times, and Crowd::Debug suggested useful fixes in 120 cases (57%). 38 times (18%) suggestions were not useful, and for 30 bugs (14%), no fixes were suggested. In 23 cases (11%) the tests contained errors that prevented execution to reach to the point where the Crowd::Debug interface was shown.

Three participants in our user study were working for the same employer. We were particularly interested in their performance as a group, given that they shared similar coding practices and styles. For these participants, suggestions given by Crowd::Debug were useful in 63% of the cases versus 53% when compared to other participants. This suggests that common development practices can increase the utility of Crowd::Debug.

Discussion and Future Work

Benefits

Like systems for compiled languages, Crowd::Debug can successfully suggest fixes to syntactical errors. These can be debugged with knowledge of a language's small set of syntactical rules.

Coding and Testing Conventions: Crowd::Debug is especially effective for errors that arise from being unfamiliar with the conventions of a framework or library, such as assumptions about the structure and naming of methods. In addition, if development conventions are consistent enough between programmers (*e.g.*, if there are strong style guides within an organization), Crowd::Debug may act as an auto-complete tool for code that returns useful suggestions for frequently used implementation patterns.

Leverages Expert Users: In HelpMeOut, expert programmers can only contribute meaningful fixes if they intentionally try to compile broken code. In Crowd::Debug, we take advantage of the fact that experts run failing tests, so they also contribute fixes.

Limitations

Detection of Errors: If an error does not raise an exception, then no information is provided. Logical errors, such as the ordering of results, cannot be detected at all.

Code Privacy: Crowd::Debug does not employ any mechanisms to regulate visibility and sharing of code. Organizations may not want to expose their proprietary code to outsiders. Even within organizations, not everyone has an access to all code assets. A possible remedy is to add authorization levels to parts of code that should not be shared.

Duplication of Fixes: Crowd::Debug fails to detect duplicate fixes. While presenting code suggestions, Crowd::Debug does not check whether code fixes are duplicated or not. This results in many structurally

المنارة للاستشارات
www.manaraa.com

identical fixes being stored. As a result, users may be presented with many nearly identical results in code suggestions. Future versions could suppress display of duplicates and instead display a variety of fixes.

Result Relevance: Our system uses string edit distance to compare signals and return relevant fixes. In addition, the ranking of suggestions is fully automated in the current prototype. These measures may not be fine-grained enough for a large deployment with hundreds of contributing programmers. Future work should investigate other code comparison techniques and incorporate direct user ratings as a feedback signal.

Scaling: It is an open question how the Crowd::Debug approach scales as codebase size increases. We hypothesize that the scope of each test is more important than total code size: if unit tests are narrowly scoped, Crowd::Debug should remain effective even as application complexity grows.

Summary and Conclusions

This paper presented our ongoing work on Crowd::Debug, a programming tool for suggesting solutions to program errors. Generating suggestions requires collecting examples, which is difficult in dynamic, interpreted languages. *The primary insight behind Crowd::Debug is to leverage test-driven development as the source of bug fix examples.* We described the general architecture for such a system, our current prototype implementation, an initial evaluation and a discussion of the potential benefits and limitations of our tool.

References

1. Ashok, B., Joy, J., Liang, H., Rajamani, S.K., Srinivasa, G. and Vangala, V. DebugAdvisor: a recommender system for debugging. In *Proceedings of the ACM SIGSOFT symposium on the foundations of software engineering*, (ESEC/FSE'09), 373-382.
2. Brandt, J., Dontcheva, M., Weskamp, M. and Klemmer, S.R. Example-centric programming: integrating web search into the development environment. In *Proceedings of CHI 2010*, ACM (2010), 513-522.
3. Daniel, B., Jagannath, V., Dig, D., and Marinov, D. ReAssert: Suggesting Repairs for Broken Unit Tests. In *Proceedings of ASE 2009,* 433-444.
4. Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. R. What Would Other Programmers Do? Suggesting Solutions to Error Messages. In *Proceedings of CHI 2010*, ACM (2010), 1019-1028.
5. Jeffrey, D., Feng, M., Gupta, N., and Gupta, R. BugFix: A learning-based tool to assist developers in Fixing bugs. *In Proceedings of ICPC '09,* 70-79.
6. Ko, A.J. and Myers, B.A. Designing the WhyLine: a debugging interface for asking questions about program behavior. *In Proceedings CHI 2004,* ACM (2004), 151-158.
7. Mamykina, L., Manoim, B., Mittal, M, Hripcsak, G., and Hartmann, B. Design Lessons from the Fastest Q&A Site in the West. In *Proceedings of CHI 2011,* ACM (2011).
8. Murphy, C., Kim, E., Kaiser, G. and Cannon, A. Backstop: a tool for debugging runtime errors, *SIGCSE Bulletin 40, 1 (March 2008)*, 173-177.

المنارة للاستشارات

www.manaraa.com