Michelle Ichinco, Kyle Harms, Caitlin Kelleher (2015):
Utilizing Programmer Communities for End User Programmer Feedback.
In International Reports on Socio-Informatics (IRSI),
Proceedings of the CHI 2015 - Workshop on End User Development in the Internet of Things Era
(Vol. 12, Iss. 2, pp. 15-20)

Utilizing Programmer Communities for End User Programmer Feedback

Michelle Ichinco, Kyle Harms, Caitlin Kelleher
Washington University in St. Louis
{ichincom / harmsk / ckelleher}@seas.wustl.edu

Abstract. In this paper, we describe a system for large scale feedback and the potential role it could play in programmer communities. Imagine a system that crowdsources feedback from experienced programmers and automatically distributes it to less experienced end user programmers. We believe a system like this may be helpful for communities like maker spaces and open source software, which have diverse groups of contributors who collaborate and support each other. With an expected increase in end user programming communities for customization in the Internet of Things, such feedback systems would likely benefit end user programmers and the code they produce.

1 Introduction

The Internet of Things is likely to increase the already large population of end user programmers. The growing number of “smart” devices and possible features suggests that niche programmer communities could potentially form around specific interests and projects. We expect that these communities, like current open source and maker communities, will include programmers of varying skill levels, motivated by the desire to improve products that they use [8]. However, end user programmers often find various aspects of programming confusing, such as program behavior and how to create complex functionalities [7]. Current systems focus on either 1) helping end user programmers solve known problems, or 2) automatically locating issues, without providing help toward solving those problems. We propose a crowdsourced feedback system to provide both of these types of support for end user programmers.

Consider a novice end user programmer, Jamie, who is working on an application to control home access for visitors like nannies and maids. Jamie can ask questions of more experienced programmers through forums, but when Jamie implements a security feature based on information she found on a website, she may not realize that a more secure implementation exists. Experienced programmers may spend time answering questions, but they will not necessarily review every line of code. Now imagine a system that Jamie can have “crawl” her code to find opportunities for improvement, as suggested by more experienced developers in her community. These suggestions could provide example code designed to help Jamie learn and improve her applications. We have begun to explore a crowdsourced system for suggesting this type of code improvement to novice programmers.

2 Related work

Two existing types of systems help end user programmers improve their code: 1) systems that support programmers who can identify their issues, and 2) systems that can identify issues or opportunities for improvement in code.

Overcoming programming problems
Online forums and crowdsourced bug fixes help programmers to solve problems that they have found in their own code. StackOverflow and similar forums can help programmers to overcome problems as long as the end user programmers know which questions to ask [9]. Tools also exist to provide support for fixing bugs, such as HelpMeOut, which collects bug fixes from a population, has experts annotate the fixes, and then presents them to users who have the same issues [3]. Currently, this type of assistance can only help when the programmer already knows they have a problem or when an error alerts them to an issue.

Identifying programming problems
Static code analysis and code smells can help programmers to identify issues in code. A variety of static code analysis tools, such as FindBugs [4], check programs for common issues and even allow programmers to author their own checks. Code smell detection systems similarly automatically find potentially problematic code [2]. Yet, these types of systems do not provide information about how to fix the problems or how to improve programming skills.

3 A Crowdsourced Feedback System

With existing systems in mind, we wanted to design a system that would find issues that end users do not realize they have, as well as provide them with information about how to fix those problems. Since programmers often search the web for examples of code to solve their problems [1], we operationalize "feedback" as example code that can be used to solve a problem or improve code. We believe that making the system crowdsourced could reduce the amount of time each experienced programmer would need to spend to provide feedback at a large scale. The model for this system has three main components, as shown in Fig. 1:
1) create a code example and author a rule script for that example, 2) community review of feedback, and 3) feedback presentation to the end user programmer.

1) *Code Example Creation and Rule Script Authoring*
First, an experienced programmer creates a code example by improving an existing program and then annotates the example to provide assistance for using it. The annotated code example may focus on proper programming practices, such as a more efficient way to code a function. A code example could also show an example of a similar, but more advanced code snippet that might improve a basic program, such as improved security for a login feature. Additionally, end user programmers could likely create code examples as they begin to gain skills.

An experienced programmer then authors a rule script that codifies the conditions under which this code example might help other users improve their program. These rules enable experienced programmers to define which programs qualify for specific feedback, such that the system could then automatically distribute feedback.

2) *Community Review of Feedback*
Other experienced programmers would then review feedback to ensure quality and generalizability. This process would involve a number of experienced programmers contributing a very small amount of time to check over existing feedback.

3) *Code Example Presentation*
The system would then use the rule scripts (created in step 1) to determine which programs qualify for certain code examples. A code example would then be shown to the end user programmer as a suggested way for them to improve their code. A user could, hypothetically, receive these suggested code examples automatically, or select to see them on request.

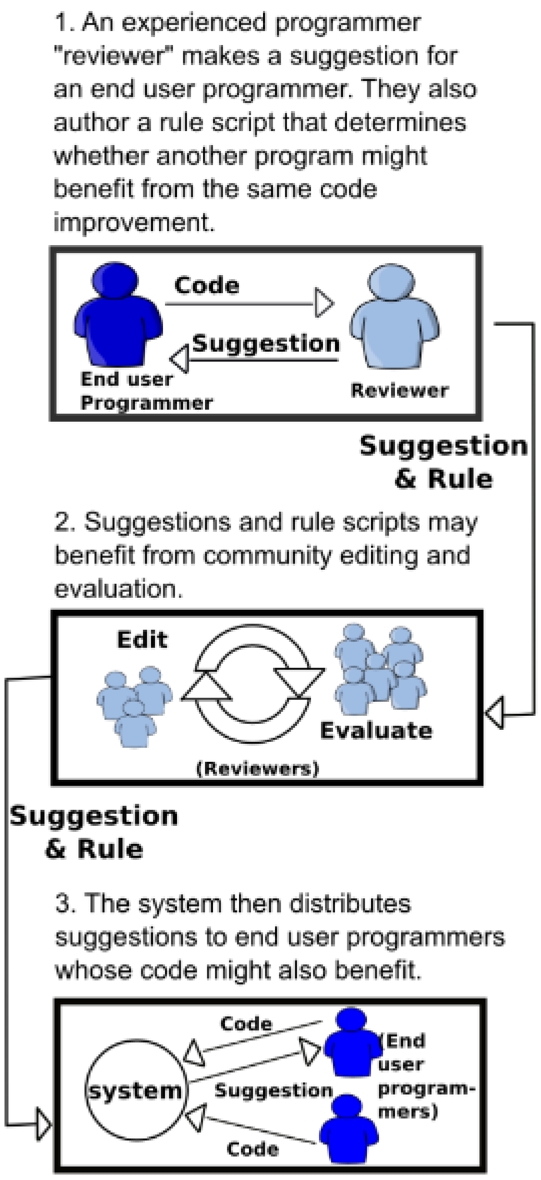
1. An experienced programmer "reviewer" makes a suggestion for an end user programmer. They also author a rule script that determines whether another program might benefit from the same code improvement.
Code
Suggestion
End user Programmer
Reviewer
Suggestion & Rule
2. Suggestions and rule scripts may benefit from community editing and evaluation.
Edit
Evaluate
(Reviewers)
Suggestion & Rule
3. The system then distributes suggestions to end user programmers whose code might also benefit.
Code
system
Suggestion
Code
(End user program-mers)

Fig. 1: Hypothesized workflow for a crowdsourced feedback system for end user programmers

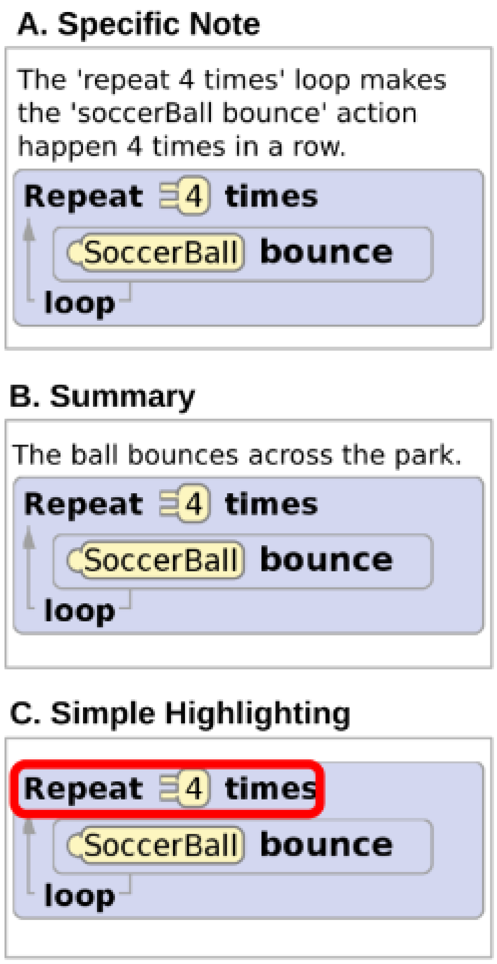

Fig. 2: We found that these three of example annotations did not significantly affect novice programmer’s ability to complete a task using an example.

4 Studies

We ran three studies on crowdsourcing feedback: a study of the feasibility of experienced programmers creating examples and authoring rules, a study on the workflow of a tool for this type of system, and a study looking at the presentation of examples.

Our exploratory study of example creation and rule authoring showed that experienced programmers often created useful code example suggestions and authored rules in pseudocode [6]. Their psuedocode rule scripts also demonstrated how experienced programmers conceptualized authoring rule scripts, providing insight into how to design a tool to support rule-authoring.

Communities of programmers will likely include programmers of varying skill levels, all of whom we believe can be valuable contributors to a crowdsourced feedback system. We developed a tool prototype, which involves four sub-tasks: I) explore an end user'sprogram, II) evaluate existing feedback for that program, III) create feedback for that program, and IV) author a rule script for that feedback [5]. Preliminary results showed that a broad population of programmers (97% of participants) can evaluate and create feedback, while 71% can effectively author new rules. One concern, however, is that participants spent about 5.5, 11.5, 9 and 14 minutes on each of the four sub-tasks, respectively. While these times may be reasonable with respect to answering questions thoroughly on a forum, ideally, each sub-task would be completed in a shorter amount of time.

We may be able to reduce feedback creation time by changing how experienced programmers annotate example code. We ran a preliminary study to investigate how different annotation styles affect novice programmers' abilities to complete tasks. Results showed that simple highlighting, as shown in Fig. 2-C, can focus a programmer's attention on the critical aspect of a code example just as effectively as two types of textual descriptions (Fig. 2-A and Fig. 2-B). This supports the use of crowdsourcing for providing feedback, since simple annotations likely require less time and revision than textual annotations.

5 Conclusion and Future Directions

As communities of workers form with the Internet of Things, we believe it will be important to harness the knowledge of experienced programmers to improve the programming skills of the end user programmer population. End user programmers may not always know which questions to ask or how to improve their code, so we believe that leveraging the crowd of experienced programmers may improve the code and skills of end user programmers.

6 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1054587.

7 References

[1] Brandt, J., Guo, P.J., Lewenstein, J., Dontcheva, M., and Klemmer, S.R. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. Proc. of the SIGCHI Conf. on Human Factors in Computing Systems, ACM (2009), 1589–1598.

[2] Fowler, M. and Beck, K. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999.

[3] Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S.R. What would other programmers do: suggesting solutions to error messages. Proc. 28th int. conf. on Human factors in computing systems, (2010), 1019–1028.

[4] Hovemeyer, D. and Pugh, W. Finding bugs is easy. SIGPLAN Not. 39, 12 (2004), 92–106.

[5] Ichinco, M., Dosouto, Y., and Kelleher, C. A tool for authoring programs that automatically distribute feedback to novice programmers. Visual Lang. and Human-Centric Computing (VL/HCC), 2014 IEEE Symp. on, IEEE (2014), 207–208.

[6] Ichinco, M., Zemach, A., and Kelleher, C. Towards generalizing expert programmers' suggestions for novice programmers. Visual Lang. and Human-Centric Computing (VL/HCC), 2013 IEEE Symp. on, IEEE (2013), 143–150.

[7] Ko, A.J., Myers, B.A., and Aung, H.H. Six learning barriers in end-user programming systems. Visual Lang. and Human Centric Computing, 2004 IEEE Symp. on, IEEE (2004), 199–206.

[8] Shah, S.K. Motivation, governance, and the viability of hybrid forms in open source software development. Management Science 52, 7 (2006), 1000–1014.

[9] Stack Overflow. http://stackoverflow.com/.