Abstractionless Programming in App Inventor

Audrey Seo
Computer Science Department
Wellesley College
Wellesley, Massachusetts, U.S.
aseo@wellesley.edu

Abstract

App Inventor is a web-based blocks programming environment that allows people of all ages and experiences to build mobile apps, introducing new programmers to fundamental programming concepts such as variables, conditionals, iteration, and procedures. Past studies show that programmers do not use procedures, App Inventor's key means of abstraction for avoiding duplicate code, very often, which suggests that App inventor programmers, instead of using procedures, merely duplicate their code. This research focused on detecting duplicated event handlers in two datasets of App Inventor code. I found that about 13% of App Inventor event handlers are duplicates. Surprisingly, over 49% of this code cannot be extracted into a procedure without using the rarely used blocks for manipulating generic components, which could be further preventing App Inventor programmers from using procedures as abstractions. Therefore, more work needs to be done to make programmers more aware of generics and how to use them.

Keywords App Inventor, procedure extraction, computational thinking, blocks programming, abstraction

1 Introduction

App Inventor is a web-based blocks programming language that allows users with little to no coding experience to quickly create and deploy mobile applications. In the in-browser App Inventor IDE, programmers drag and drop user interface and functional components onto a simulated phone screen, and then combine programming blocks to specify the behavior of the app. As of July 2018, App Inventor has over 7 million users who have created 30+ million apps.

Two data sets of App Inventor projects were obtained for this research. The first data set, referred to as the 10K data set, consists of all of the projects created by 10,003 randomly selected App Inventor programmers. In total, it contains 30,983 projects. The second data set, dubbed the 46K data set, was chosen based on studying so-called "prolific" App Inventor programmers: those with 20 or more programs each. The 46K dataset contains 1,546,056 projects that were created by 46,320 programmers[1].

An App Inventor project, once downloaded, actually consists of several parts: JSON and XML files for each of the screens involved in the app. In order to streamline analysis, these two parts were combined and parsed back into JSON. However, several of the projects in each of the datasets failed to parse correctly, so the actual number of projects studied in this research was

- 30,851 of the 10K projects, and
- 1,545,284 of the 46K projects.

Last summer, it was discovered that procedures, a crucial abstraction mechanism in App Inventor, only 15% of random projects and and 18% of the 46K projects used procedures. [2].

2 Detecting Opportunities for Procedure Extraction

My main goal was to determine if App Inventor programmers are missing opportunities for procedure extraction, i.e. capturing nearly identical code patterns with procedures, as in Fig. 1. I wrote a program that looked for nearly duplicate event handler bodies, each of which could be replaced by a call to a single new procedure with the appropriate arguments within a single screen of an app, that had at least five blocks to be consistent with other work [1]. I found that duplication is prevalent in both data sets, where 18.7% of the 10K programmers and 87.6% of the 46K programmers had at least one missed opportunity for procedure extraction. Some other results are shown in Fig. 2 and 3.

An unexpected result was that 55% of the 10K and 49% of the 46K duplicated code could not be extracted into a procedure without using so-called generic components and blocks. These are needed in situations where a procedure must abstract over a particular component (such as a label or button) rather than a simple value like a number or string. See Fig. 1. As Table 1 indicates, generic blocks are very rare, possibly because programmers do not know about them or find them hard to use. Furthermore, very few resources that demonstrate how generics are used exist, and in the App Inventor IDE, generic component and generic method, getter, and setter blocks are kept in entirely different menus, which does

BLOCKS+, November 04, 2018, Boston, MA, USA
2018. ACM ISBN …$15.00
https://doi.org/

[1] 129 of these programmers overlap with the 10K programmers.

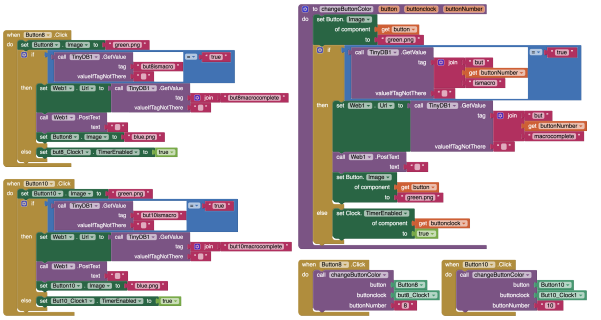

Figure 1. Left: two button click handlers with code that was identified as duplicates. Right: the resulting procedure that was abstracted, in addition to the two calls to the procedure that replace the duplicated code on the left.

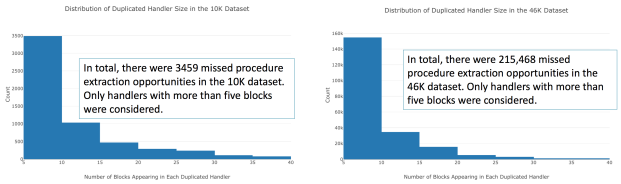

Figure 2. The distributions of the numbers of blocks in each case where a procedure could be extracted.

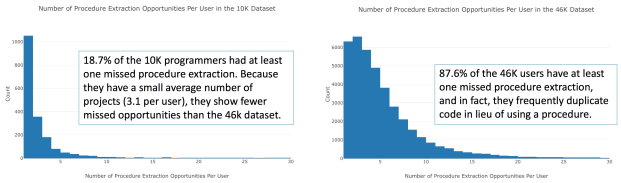

Figure 3. The distributions of the number of opportunities to create procedures per user in both datasets.

not suggest to a beginner programmer that they are meant to be used together and exacerbating this disconnection.

Block Type	Frequency	
	10K	46K
Generic methods, getters, and setters	0.014%	0.21%
Nongeneric methods, getters, and setters	40.3%	44.6%

Table 1. Block frequencies for the generic and nongeneric component method, getter, and setter blocks for all projects in each dataset.

3 Related Work

This work and [2] are in the area of blocks programming research on procedure extraction. Missed extraction opportunities are an example of a *code smell*, i.e., a pattern of bad programming style, in App Inventor. Other work has studied this and other smells in other blocks languages such as Scratch, a blocks programming language targeted specifically at children. In [3] it was found that Scratch projects were less likely to be "remixed" if they had the code smells Long Script, Duplicate Code, or Unused Variable. [1] found that Scratch programmers tend to have duplicated code, just as we found in this study, in addition to also examining the code smells dead code, large script, and large sprite.

4 Presentation Proposal and Next Steps

At my poster presentation, I would like to present evidence for the aforementioned problems in addition to proposing a possible solution involving procedure extraction and changes to the App Inventor interface regarding generic blocks. This would be guided by the following questions.

1. Is App Inventor code easier to copy and paste than a text-based language like Python or Java's code, simply because it is a blocks programming language?
2. How can the App Inventor language or environment be changed to encourage programmers to use more procedures? Examples of improvements include highlighting procedure extraction opportunities to programmers, or even automatically performing the extractions
3. How aware are App Inventor programmers of the other abstraction mechanisms in App Inventor: variables, lists, loops, generics, and files? How effectively does App Inventor allow them to use these abstractions?

Acknowledgments

This research was funded by the 2018 Wellesley College Science Center Summer Research program through the IBM Faculty Research Fund for Science and Math. I would also like to thank my advisor, Lyn Turbak, for mentoring me throughout this past summer on this project, and the rest of the App Inventor and Blockly development teams for their valuable guidance and insight.

References

[1] Efthimia Aivaloglou and Felienne Hermans. 2016. How Kids Code and How We Know: An Exploratory Study on the Scratch Repository. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, New York, NY, USA, 53–61. https://doi.org/10.1145/2960310.2960325

[2] I. Li, F. Turbak, and E. Mustafaraj. 2017. Calls of the wild: Exploring procedural abstraction in app inventor. In *2017 IEEE Blocks and Beyond Workshop (B B)*. 79–86. https://doi.org/10.1109/BLOCKS.2017.8120417

[3] P. Techapalokul and E. Tilevich. 2017. Understanding recurring quality problems and their impact on code sharing in block-based software. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 43–51. https://doi.org/10.1109/VLHCC.2017.8103449