Measuring the Programmatic Complexity of Projects of Varying Functionality in App Inventor

Benjamin Xie, Isra Shabir, and Hal Abelson
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
Email: {bxie, ishabir, hal}@mit.edu

***Abstract*—MIT App Inventor is a visual blocks language that enables users with little to no previous programming experience to create applications for Android devices. We analyze a random sampling of projects to better understand what apps the App Inventor environment lends itself towards creating. In particular, we group projects of similar functionality and measure the programmatic complexity of these projects.**

I. Introduction

MIT App Inventor is an environment that leverages a blocks-based visual programming language to create mobile apps for Android devices. A project consists of a set of components and a set of blocks which give the components functionality. Components include visual items (e.g. buttons, text boxes, images, drawing canvas) as well as non-visible items (e.g. camera, accelerometer, speech recognizer, GPS location sensor). The app's functionality is programmed in Blockly, a visual blocks-based programming language. Figure 1 shows blocks for an app to prevent texting while driving. When a text is received, a predefined message is sent back to the person who sent the original text.

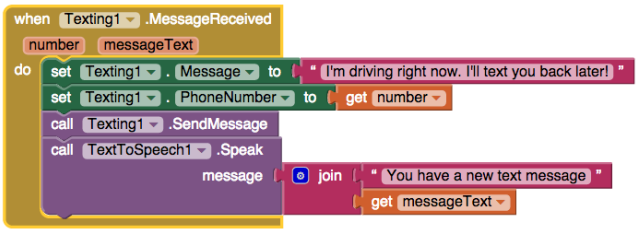

Fig. 1. Blocks for an app that automatically responds to texts received with a predefined message.

There have been two major versions of App Inventor. App Inventor Classic (also known as App Inventor 1) was released in 2009 and ran the blocks editor from a separate Java application. In late 2013, App Inventor 2 was released; the blocks editor now runs in a web browser in as a Javascript program. Although both versions of App Inventor are currently still in use, as of February 2015, 90% of App Inventor users use App Inventor 2. For this reason, the research here focuses on App Inventor 2 data.

To date, over 3.1 million users from 195 countries have created over 8.2 million projects. There are over 100,000 weekly users. We seek to better understand what users create with App Inventor and determine what kinds of apps the App Inventor environment lends itself towards creating. To do this, we analyze the types of apps created and the programmatic complexity of the projects from a random sample of projects. We group apps of similar functionality based on the components used in the app. To measure programmatic complexity, we assign each block a score where a higher score suggests more advanced programming. We then look at the unique blocks in a project and average the complexity score of each block to determine the complexity of the app.

A. Defining Programmatic Complexity

TODO

B. Previous Work

Previous analysis of App Inventor Classic was done mostly at a user level. Notable findings:

- 30% of projects have no blocks and therefore are not functional.
- Nearly 50% of users do not have a single block or component in their projects
- 53% of users only have 1 project.

From this, we gather that a significant amount of App Inventor Classic projects were never completed. It was theorized that a major contributing factor is the usability of the service. Whereas App inventor 2 is a single-page web service, App Inventor Classic required the deployment of an external Java service to program the app. Furthermore, the block language was overhauled and improved between versions of App Inventor.

App Inventor is taught at both a grade school and college level, to a diverse audience. Reports on courses taught depict App Inventor being used to create very diverse apps. These apps range from apps that discourage texting while driving to apps that track school buses to apps that organize community service cleanups. The pattern we see is that App Inventor enables "situated computing." This quarter-century old concept suggests that the convergence of computing, connectivity, and content enables users to use computing to bridge the gap between intentions and actions. App Inventor enables users to leverage their mobile devices to solve the problems everyday problems they encounter.

II. Technical Approach

We extract features from a random sampling of App Inventor 2 projects. We group projects based on the functionality of their components and also assign a complexity score based on the number of unique blocks in the project and a weighting system that considers the complexity of each block. From there, we look at each group and the distribution of the projects by complexity.

A. Data Source

Our source data is 5,228 random App Inventor 2 projects. We randomly selected projects according to their project ID, a a sequentially increasing number (so a project with a lesser project ID corresponds to a project made earlier). The 5,228 projects selected come from 5,174 users, suggesting that randomization was accomplished. Given the size of the dataset and the randomness of the selection, we assume that the sample is representative of all App Inventor 2 projects.

Of the 5,228 projects sampled...

- 21% (1,107 projects) are "certainly nonfunctional." That is, these projects either have no components or less than two blocks (the minimum required to have any functionality).
- At least 18% (937 projects) are recreations of tutorials. App Inventor has copious online teaching resources. Most of which consist of a step-by-step guide to creating an app. We find projects that are recreations of fourteen popular tutorials by matching by project name.
- 16% (845 projects) define procedures, suggesting code reuse.
- 29% (1,514 projects) initialize a global variable.

B. Feature Extraction

We focus primarily on quantitative features to enable us to run clustering algorithms. In particular, we look at the number each kind of component in a project, as well as the number of each type of block. The type of a component or block is provided in the source code of the project.

Features Extracted from Projects:

- Project Name
- User Name (anonymized)
- Number of Screens in Project
- Number of Components by Type
- Number of Blocks by Type

While most block types correspond to exactly one block, there are exceptions. Of particular note are the component blocks. Every component (e.g. button, text box, canvas) that is added to the project has associated blocks to give the component functionality. Although these blocks vary by component, they all correspond to one of three types: Component events, methods, and setter/getters. Component events are outer or top level blocks that run the code within it when the event (e.g. when button clicked) occurs. Component methods are methods relating to the component that can be called (e.g. drawing a point on a canvas). Setter/getter methods are used to either change the component properties (e.g. set the paint color of a canvas) or retrieve the component properties (e.g. text within a text box). Figure 2 provides examples of each of three component block types.

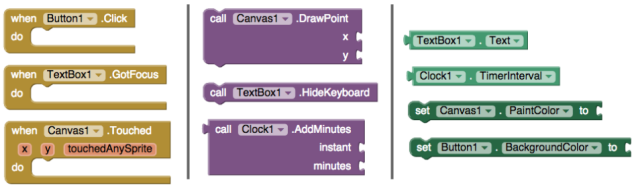

Fig. 2. Examples of component blocks of various types. Left Column: "Component Event" blocks. Center Column: "Component Method" blocks. Right Column: "Component Set/Get" blocks.

C. Grouping Projects

We group projects according to the components they contain. We group projects primarily on the grouping provided in the palette of the designer page in App Inventor 2 (Figure 3). The "User Interface" group contains basic components that almost all apps will use (buttons, text boxes, etc.). The "Layout" group, as it contains components that alter the aesthetics of the app but nothing further. Because layout components do not alter the functionality of projects, they are ignored. The "Media" group contains components that play or record text, sound, pictures, and video. The "Drawing and Animation" group contains components for drawing and basic animations. The "Sensors" group includes components that access the phone's sensors. They include the clock, GPS, and accelerometer. The "Social" group has components to access the phone's contacts, email, calling capability, texting, and Twitter. The "Storage" group is used to persist information. "Connectivity" includes bluetooth and web components to connect to other devices. "Lego Mindstorms" includes components to interface with the programmable Lego robotics kit.

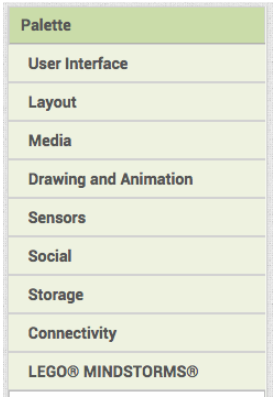

Fig. 3. The palette in App Inventor 2 groups components into categories. These categories are used to group projects by functionality

There are eight groups. Basic apps only User Interface components (and possible Layout components, but they are ignored). Media, Drawing, Sensor, Social, Storage, Connectivity, and Lego apps contain at least one component in the respective

group in the palette. This categorization allows for overlap, as project that contain components from multiple palette groups may be placed into multiple groups. In example, a project that uses both bluetooth (connectivity) and a twitter (social) component would be labelled both connectivity and social.

D. Measuring Programmatic Complexity

We look at the number of unique blocks in a project as the primary measure of the programmatic complexity. We choose the number of unique blocks rather than just the number of blocks in a project to account for projects that do not use reuse code when it would be appropriate to do so.

In example, consider the case where two functionally similar projects exist and Project A copies the same code in three locations whereas Project B defines a procedure and calls the procedure three times. We argue Project B is a more complex project as it leverages code reuse in the form of procedures. Project A has a greater number of blocks, but Project B has a slightly greater number of *unique* blocks with the blocks to initialize and call a procedure. Some blocks are more programmatically complex than others, so we assign the blocks a complexity score to weight them appropriately.

We weight blocks by assigning them a score where a higher score suggests the block is used in more complex settings. These scores were arrived at from a survey sent out to App Inventor experts. These experts were either part of the App Inventor development team or an App Inventor ”power user”.

So, the programmatic complexity of a project is defined as the sum of the weights of each unique block present in the project.

III. Results

Results go here.

IV. Discussion

Discussion goes here.

V. Conclusion

The conclusion goes here.

Acknowledgments

This research is funded by the MIT EECS - Google Research and Innovation Scholarship.

We thank Jeffery Schiller (MIT App Inventor) for help collecting the data and Ilaria Liccardi (MIT) and Franklyn Turbak (Wellesley College) for helping guide the analysis.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.