Measuring the Programmatic Sophistication of App Inventor Projects Grouped by Functionality

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 web service that enables users with little to no previous programming experience to create mobile applications using a visual blocks language. We analyze a sample of 5,228 random projects from the corpus of 8.3 million and group projects by functionality. We then use the number of unique blocks in projects and measure the programmatic sophistication of apps to better understand the learnability and realized capability of using App Inventor to implement specific functionalities. We introduce the notion of a learnability score and our results indicate that introductory tutorials heavily influence the learnability of particular functionalities with App Inventor. Our findings suggest that the sequential nature of App Inventors learning resources results in users realizing only a portion of App Inventors capabilities and propose improvements to these resources.**

I. Introduction

MIT App Inventor is an environment that leverages a blocks-based visual programming language to enable people to create mobile apps for Android devices [1]. An App Inventor project consists of a set of components and a set of program blocks that enable the functionality of these components. Components include items visible on the phone screen (e.g. buttons, text boxes, images, drawing canvas) as well as non-visible items (e.g. camera, database, speech recognizer, GPS location sensor). The app is programmed using Blockly, a visual blocks-based programming framework [2]. Figure 1 shows the program blocks for an app to discourage texting while driving. When a text is received, a default message is sent back in response and the received text is read aloud.

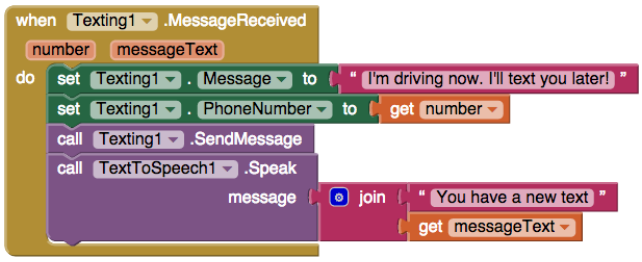

Fig. 1. Blocks for an App Inventor 2 project that automatically respond to texts received with a predefined message and reads the received text aloud.

There have been two main versions of App Inventor. App Inventor Classic (also known as App Inventor 1) was released in 2009 and ran its blocks editor in a separate Java application. In late 2013, App Inventor 2 (AI2) was released; the Javascript blocks editor now runs in a web browser. As of February 2015, 90% of the 100,000 active weekly App Inventor users use App Inventor 2. For this reason, this research focuses on App Inventor 2 data [3].

App Inventor is taught to a broad audience, ranging from grade school to college. 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 [4], to apps that organize community service cleanups [5]. The pattern we observe is that App Inventor enables ”situated computing” [6]. This quarter-century old concept suggests that the convergence of computing, connectivity, and content enables users to harness computing to bridge the gap between intentions and actions. App Inventor lets people leverage their mobile devices to solve everyday problems they encounter.

App Inventor also has copious resources for self-learners, typically in the form of self-contained tutorials. A survey of 129,130 self-selected App Inventor users found that 73% of respondents used App Inventor at home, suggesting a significant portion of App Inventor users learn to use the service on their own and not in a formal learning environment. The App Inventor resources page includes 26 tutorials ranging from beginner to advanced difficulty [7]. These tutorials typically involve creating an entire functioning app from start to finish. Each tutorial typically focuses on either introducing a new component or additional functionality for a previously introduced component.

To date, over 3.1 million users from 195 countries have created over 8.3 million apps with the MIT App Inventor service [1].

A. Objective

The goal of this paper is to evaluate the learnability and capability of App Inventor to create apps of differing functionality by analyzing the apps created with App Inventor. We define the learnability as the ease of learning to use the App Inventor service to create an app. When referring to capability, we refer to the extent of App Inventor potential that is realized by users to implement certain functionality.

A guiding principle to the creation of a programming environment is the idea of a low floor, high ceiling [8]. That is, the environment must be learnable enough such that beginners can easily create a functioning program (low floor), but also have extensible capabilities such that advanced users can also benefit (high ceiling). We are particularly interested

in comparing the learnability and capability of App Inventor for creating apps of differing functionality.

We analyze a random sample of projects and group them based on the types of components used in the app. We then look at both the number of unique blocks in projects. We use this grouping and this metric to evaluate how well suited the App Inventor environment is to creating apps with various functionalities.

In this paper, we explain our technical approach of extracting information from raw project data, filtering and grouping projects, and comparing the *sophistication* of the projects we grouped given our metric (number of unique blocks). We then discuss our findings in the context of the App Inventor service and its teaching resources.

B. Previous Work

Prior to this work, analysis of App Inventor Classic data has been done by [**?**]. Some of the 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.
- 51% of procedures are only called 0 or 1 time.

This data indicates that a large number of App Inventor Classic projects were never completed. It was suggested 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. The high proportion of projects without blocks motivated the usability changes of the blocks in App Inventor 2.

A field study of end-user programming on mobile devices was conducted for TouchDevelop, a programming environment that enables users to develop mobile applications directly from mobile devices [9]. The study's objectives included measuring users' progress in developing TouchDevelop scripts. Researchers found that 71.3% of users learned a few features about the environment initially and then stopped learning new features. To encourage more continuous learning, researchers suggested providing an adaptive tutoring system that recommends tutorials similar to the kind of script a user is developing and avoids tutorials that cover features users already know. As discussed later in the paper, our findings suggest that App Inventor may also have a similar situation where users tend to only learn a subset of features available to them.

II. Technical Approach

We extracted features from a random sampling of App Inventor 2 projects. We grouped projects based on their functionality by considering the components they contain. We then measured the *total number of unique blocks* (NOUB) in the projects to determine the sophistication they exhibit. Finally, we examined the distribution of the NOUB in each group to answer our research question.

A. Data Source

Our source data is 5,228 App Inventor 2 projects selected at random from the total corpus of 8.3 million projects. We randomly selected projects according to their unique project IDs. 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 App Inventor 2 projects as a whole. We used Pandas, a Python data analysis library, for our data processing [10].

Of the 5,228 projects sampled...

- At least 16.4% (859 projects) are recreations of App Inventor tutorials. These recreations of the step-by-step tutorials were identified by matching project names. We considered only the 26 tutorials from the MIT App Inventor website, although many other tutorials made by other groups and individuals exist [11] [12]. Projects that are recreations of these tutorials are filtered out of our dataset.
- 21% (1,107 projects) are *Certainly Static*; that is, they are guaranteed to be apps that have no behavior and never change state. If a project has no components, then there is nothing the user can interact with or for the app to do, so the project must be static. For an app to be interactive and have behavior, in addition to at least one component, it must also have at least two blocks: One to handle an event and one to respond to that event. Figure 2 shows an example of a simple action from two blocks. No functionality can occur with fewer than two blocks. We say an app is Certainly Static if it either has no components or has fewer than two blocks.

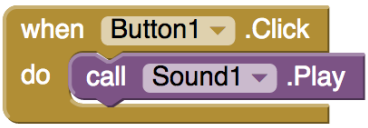

Fig. 2. The simplest app behavior requires at least two blocks: An event handler and a resulting action. Here, a sound is played when a button is pressed.

We chose to filter out the Certainly Static projects as well as projects that are recreations of tutorials, so our analysis was run over the remaining 3,289 projects. While we can guarantee that the removed projects are static, we cannot guarantee the remaining projects have behavior, as their blocks may not be connected in a manner that allows for any behavior. For the purpose of analysis, we assume the remaining 3,289 projects have behavior and are not recreations of tutorials.

B. Feature Extraction

We focused primarily on quantitative features for our analysis, particularly the number of each type of component in a project, and the number of each type of block. This information exists in the source code of the projects.

Features Extracted from Projects:

- Project Name

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

C. Grouping Projects

We use the components within a project to group projects by functionality. The palette in App Inventor organizes components by functionality, or behavior, and places each group in its own ”drawer” (Figure 3). Because the palette neatly organizes components into categories , we use it to define our groups. If an app has components from multiple palette drawers, it may be categorized in multiple groups, as explained later in this section.

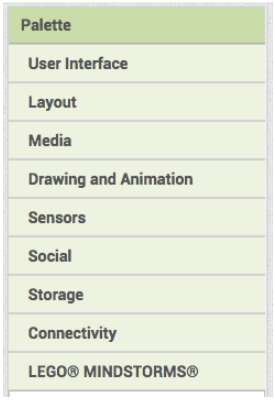

Fig. 3. The palette groups components into categories. We use these categories to group projects by functionality.

We follow the palette drawers to define our groups, with two notable changes: Disregarding the entire ”Layout” component drawer and the sound and clock components.

Layout components were removed because they do not add additional functionality and are therefore irrelevant for our groupings. These components only enable users to change the arrangement of an app’s visual components. Since we group projects by their functionality and layout components add no additional behavior to apps, they are disregarded.

The sound and clock components were removed to improve the differentiation between functionality groups. The sound component plays a sound whenever the user specifies. Examples include playing a ”meow” when an image of a cat is pressed and playing a famous speech in a historical quiz app. The clock component enables apps to keep track of time. Uses for this vary from keeping time in a stopwatch app to periodically moving a sprite in a game app. Because the sound and clock components have such broad uses, they do not help differentiate apps’ behaviors between groups and are also excluded in the consideration of functionality groups.

We categorize the 3,289 apps into eight groups. *Basic* apps only contain User Interface components. Apps in the *Media, Drawing, Sensor, Social, Storage, Connectivity*, and *Lego* groups contain at least one component from that respective drawer in the palette. This categorization allows for overlap, as projects that contain components from multiple palette drawers are placed in multiple groups. For example, a project that uses both Bluetooth (connectivity) and Twitter (social) components would be labeled both Connectivity and Social. The exception is the Lego group, which we deem to be a disjoint group because of the specificity of the components. Lego components are solely for integration with Lego Mindstorms [13]; if a project contains a Lego component, it is only grouped as Lego, regardless of other components it may contain.

Reiterating, Basic and Lego groups are disjoint from other groups and each other. Other groups may overlap. Table 1 provides a description of each group, the condition for a project to be in that group, and example apps and components from each group.

TABLE I. FUNCTIONALITY GROUPINGS

Group Name	Description of App Functionality {Example App}	Condition	Example Components
Basic	Only basic user interface components {Splits bill amongst certain number of people}	Only User Interface Components	Button, Image, Label, Notifier, Textbox
Media	Playing/recording audio or video {Click on picture of politician to hear their famous speech}	At least 1 media component (excluding ”sound”)	Camera, TextToSpeech, VideoPlayer, MusicPlayer
Drawing	Use screen as canvas for drawing {Draw on picture of cat}	At least 1 drawing component	Canvas, Ball, ImageSprite
Sensor	Response to phones sensors {Shake phone to roll a die}	At least 1 sensor component (excluding ”clock’)	Accelerometer Sensor, Location Sensor, NearField (NFC)
Social	Communication via phone or web {Click on a persons picture to call or text them}	At least 1 social component	Texting, Twitter, PhoneCall
Storage	Saving information {Add items to grocery list and save list}	At least 1 storage component	TinyDB, Fusiontables Control, File
Connectivity	Networking with other apps and phones {Get latest stock quotes from web}	At least 1 connectivity component	ActivityStarter, BluetoothClient, Web
Lego	Control Lego Mindstorm kits {Remote control for Lego Mindstorm NXT robot}	At least 1 lego components	NxtDrive, NxtLightSensor

We use the components to group projects and the blocks to measure the sophistication of them.

D. Measuring Programmatic Sophistication

We define the *sophistication* of an App Inventor project as a measurement of the skill involved to create an app as evidenced by the blocks used. More sophisticated projects tend to reflect good coding practices, such as code reuse through procedures. A more sophisticated app tends to either use more components or use blocks corresponding to these components more effectively.

Code reuse is a particular focus in our measure of sophistication. For 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 that procedure three times. We argue Project B is more sophisticated as it leverages code reuse in the form of procedures. Project A has a greater number of blocks, but Project

B has a greater number of *unique* blocks (NOUB) with the block to define a procedure and the block to call a procedure included. A project that appropriately uses a procedure rather than copying blocks shows evidence of greater computational thinking and therefore greater sophistication [14], even if the resulting apps have identical functionality.

We measure programmatic sophistication of App Inventor projects by looking at the NOUB that exist in the project. We choose the NOUB instead of the total number of blocks so the measure of sophistication is not affected by redundant code.

III. Results

We show the division of the projects into groups then show the distribution of the number of unique blocks (NOUB) in projects of each group.

A. Grouping

After grouping projects by functionality, we find that 78.1% of projects can be categorized into exactly one group. The remaining projects were categorized across multiple groups (see 2C for explanation). The 3,289 projects were categorized into 4,282 groups; on average, a project fit into 1.3 groups. Figure 4 shows the division of projects into groups.

Based on the distribution of projects into groups, we observe a relationship between this distribution and App Inventor tutorials. Due to the simplicity of functionality that defines the group, the largest group is the Basic group. Over half of the App Inventor beginner tutorials involve the creation of a drawing app [7] and we see that the Drawing group is the second largest group. These observations suggest that the large number of drawing apps users create are projects that are very similar in functionality to tutorials. The Lego group is the smallest, containing only 0.7% (27 projects) of the data. One likely explanation is the additional hardware requirement (Lego Mindstorm kits) to use an app grouped as Lego. Another is that there are no official tutorials for Lego projects, so users do not have a way to learn how to use the Lego components. We hypothesize that the number of projects in each functionality group correlates with the number of functionally similar tutorials available. We further address this in our Discussion section.

B. Number of Unique Blocks

We plot the distribution of the NOUB in each group and compare these subsets of projects to each other and to the entire set of projects. Figure 5 and Table 3 show the NOUB for projects within each group, as well as the distribution for all projects ("All" in Figure 5).

Each subset and the entire set of projects exhibits a positive skew, suggesting that each group contains a few outlier projects that have a significantly greater NOUB and are likely well-developed apps.

The Storage group has the greatest median NOUB, the widest distribution, and contains the project with the most unique blocks, suggesting that apps that utilize storage functionality tend to be the most advanced and sophisticated. This could be because storage components often require structures

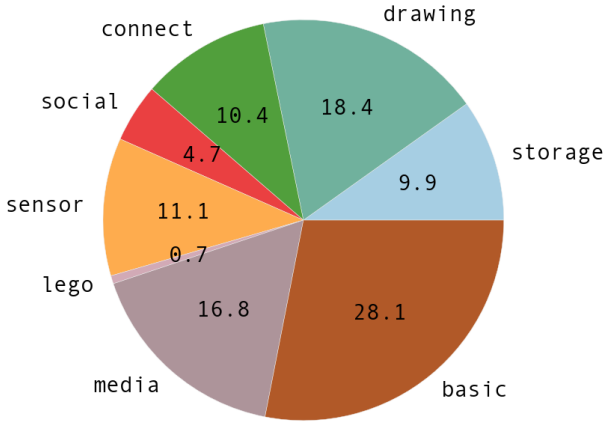

Fig. 4. Size of Functionality Groups. 78.1% of projects are categorized into exactly one group, with the others categorized across multiple groups.

such as lists and loops to leverage its more advanced functionality. An example would be using a loop to iterate over the keys and values in a database (TinyDB) component and saving values into a list.

The wide lower quartile and narrow overall distribution of the Lego group suggests its capabilities are limited. The Lego group has a wide lower quartile (lower whisker in Figure 5) relative to its narrow distribution, suggesting that even a simple project involving Lego components requires more unique blocks to create. The narrow distribution and low median for the Lego group suggests that the capability to create Lego apps is limited. The need for more unique blocks to create even a simple app with Lego components and the limited functionality of these apps suggests that developing these apps is not as intuitive and therefore more difficult to create.

Because 21.9% of projects fit into multiple groups, one project can be represented in multiple plots. This is most evident in the outliers. The greatest outlier is a password keeper app with 56 unique blocks in it; it is categorized as a Storage, Connectivity, and Media app because it has components of each of those types.

TABLE II. Summary Statistics for the Number of Unique Blocks by Group

	all	storage	drawing	connect.	social	sensor	lego	media	basic
med.	7	14	11	10	8	7	6.5	5	6
mean	9.17	15.40	11.29	12.34	10.19	9.80	7.86	8.14	9.17
std. dev.	7.11	9.61	6.83	8.79	8.73	7.43	4.86	6.94	5.80
max (w/o outliers)	26.5	38.0	31.0	33.5	29.0	29.0	17.5	23.0	20.5
# outliers	90	11	6	14	10	12	1	26	41

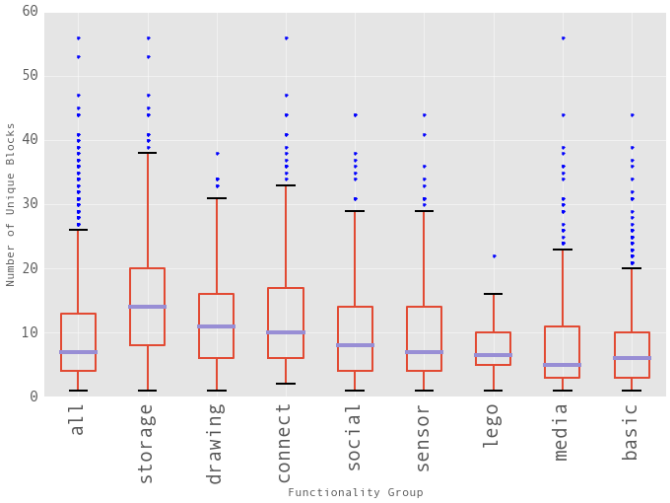

Fig. 5. Distribution of Number of Unique Blocks by Functionality Group

IV. Discussion

We critique our use of the number of unique blocks as a metric for measuring sophistication and analyze the intuitiveness of creating different types of apps with App Inventor. We then discuss the capability realized by end-users. Finally, we relate this discussion on learnability and capability to App Inventor tutorials.

A. Analysis of Metric

When measuring the sophistication of projects, our challenge is to ensure that project categories do not bias our metrics. That is, our measurement of app sophistication is not affected simply because apps include a specific component and hence fall under a certain group. We want to measure apps solely according to the sophistication exhibited by the blocks. We argue that our metric of the NOUB is not dependent on the functionality of the app and is therefore a generalizable metric of programmatic sophistication.

Because App Inventor has component-specific blocks, the NOUB in a project is not directly dependent on its components. App Inventor is event-driven, meaning the programming of App Inventor involves responding to an action, or event, from a component. Each component has its own unique blocks to handle events, get and set attributes of the component, and call component functions. Because of this, using one component instead of another does not inherently change the NOUB in a project. App Inventor blocks respond to events, get/set attributes, and trigger component actions. Figure 6 shows an example of custom component blocks handling an event, calling a function, and setting an attribute. Because of App Inventor's component-specific blocks, the NOUB in a project is a suitable metric to measure the sophistication of projects.

Because the NOUB does not systematically vary according to the components used in the projects, we find this metric suitable for our analysis.

B. Learnability

We claim a group to have high learnability if it does not require many different blocks to create a project. If a group has high learnability, we expect many projects to be categorized into that group. We define a *learnability score* of a group as the number of projects in the group divided by the the median number of unique blocks for that group. The results for the different groups are depicted in Figure 7.

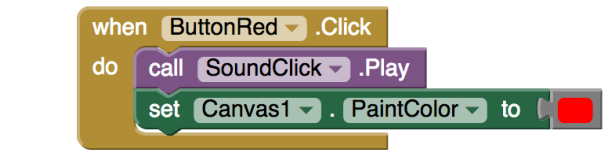

Fig. 6. Component-Specific Blocks. The button component has a block to handle being pressed, the sound component has a block to play a sound, and the canvas component has block to set the color.

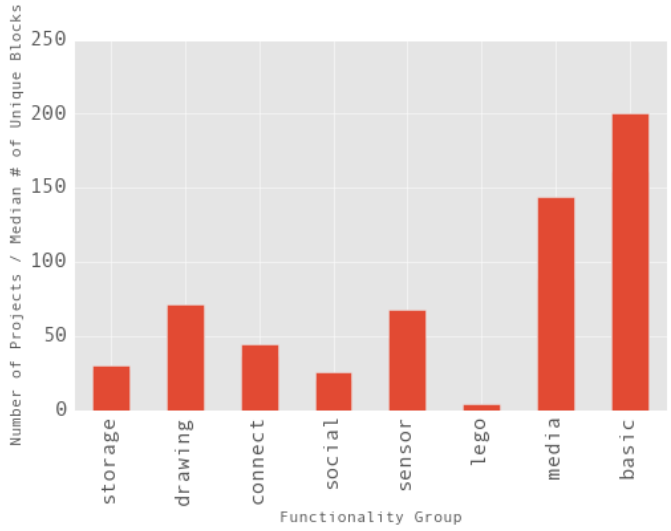

Fig. 7. Learnability Score (Ratio of the Number of Projects to Median Number of Unique Blocks) of Functionality Groups

Apps in the Basic group have the highest learnability score and are therefore the easiest to learn. This is not surprising because we narrowly define the Basic group to contain apps that only use user interface components. The Drawing and Media groups also have high learnability scores. This is likely because the tutorials heavily focus on creating Drawing apps and Media apps. We argue that the learnability score is influenced by the beginner tutorials and address this later in the Discussion section.

C. Capability

We focus our discussion on *realized* capability, or the maximum potential of App Inventor that users actually reach. We are not referring to the "true" capability of App Inventor, the capability that is technically possible but in practice almost never implemented by users.

To assess the realized capabilities for App inventor to create apps of certain functionalities, we are interested in the projects in each group that have the greatest sophistication (greatest NOUB). It is these projects that best reflect the capability of App Inventor to accomplish certain functionality. We choose to look at the the maximum NOUB in each group excluding

outlier projects. This is the end of the upper whiskers in Figure 5 and also shown in Table 2. We argue that these apps best represent the capability realized by App Inventor users.

We say that a greater NOUB correlates with the ability to use App Inventor to create apps that have more advanced functionality. Having the least capability are Lego apps, with the narrowest distribution and lowest maximum NOUB. On the highest end of the spectrum are Storage apps which leverage databases or tables to persist data.

We see that apps with the greatest capability tend to connect to other mobile features or apps or with the web. Storage and connectivity apps have the highest maximum NOUB so we say these functionalities have the greatest realized capability to be extensible. Storage apps connect to some form of data persistence (database, table, file). Connectivity apps connect with other apps in the phone, utilize Bluetooth, and connect with the web. While Social apps connect to other features in the phone (contact list, texting, etc.) or with web services such as Twitter, their realized capabilities are lower. This could be a result of a lack of learning resources relating to this particular functionality limiting the known capability of the group. In general, we see that App Inventors most advanced apps tend to connect to the web and other apps and services. This opportunity for extensibility while maintaining the scaffolding that is the App Inventor environment is critical to an environment that fosters computational thinking [15].

D. Relation to Learning Resources

The close correlation between learning scores and the sequence of tutorials suggests that users build apps based on knowledge from the tutorials that they complete. On the App Inventor website, tutorials are displayed as a list in sequential order, starting with *Beginner* tutorials and ending with *Advanced* tutorials. Table 4 shows the number of projects that were found to be tutorial recreations as well as the number of tutorials for each group.

Most users tend to start with the beginner tutorials, but the number of tutorials followed until users create their own original projects varies. The earlier a tutorial appears in the sequence on the website, the more users will use it as evidenced by the high proportion of tutorial recreations that are of beginner difficulty (Table 3). We see Drawing, Media and Sensor apps appear as beginner tutorials; these groups also account for most of the tutorial recreations and have the highest learnability scores. There are no Lego tutorials, and the very low learnability score reflects that. Although there are six tutorials involving Storage functionality, they are classified as Intermediate and Advanced, so there are fewer recreations of these tutorials. The Connectivity and Social groups also have a low learnability score and few projects recreate these tutorials. We see that lower learnability scores correlate to groups that have fewer tutorial recreations; we see that the farther along a tutorial exists in the sequence, the fewer times it will be recreated.

This correlation between the number of tutorial recreations and the learnability scores for a given functionality suggests that users build off the knowledge from tutorials when creating an app. The relationship between few tutorial recreations for groups and low learnability scores suggests that if users do not learn a concept directly from a tutorial, they tend to have trouble generalizing knowledge from other tutorials. Therefore, users tend not to create apps that are functionally different from completed tutorials. This is troublesome as [9] noted that 71.3% of users of the similar TouchDevelop programming environment tended to learn only a few features initially and not seek to learn more later. We see that users do not complete enough tutorials to gain a thorough understanding to create apps of differing functionality with App Inventor, so there exists a need to make tutorials more available and the knowledge from them more generalizable.

TABLE III. NUMBER OF APPS THAT ARE TUTORIAL RECREATIONS {NUMBER OF TUTORIALS} BY FUNCTIONALITY GROUP AND DIFFICULTY LEVEL

		Tutorial Difficulty			
		Beginner	Intermediate	Advanced	Total
Groups	All	683 {9}	95 {10 }	54 {7}	832 {26}
	Storage	0 {0}	5 {1}	48 {5}	53 {6}
	Connectivity	0 {0}	9 {1}	26 {2}	35 {3}
	Drawing	447 {4}	70 {5}	5 {2}	522 {11}
	Social	0 {0}	6 {1}	0 {0}	6 {1}
	Sensor	142 {2}	0 {0}	35 {4}	177 {6}
	Lego	0 {0}	0 {0}	0 {0}	0 {0}
	Media	198 {2}	4 {3}	0 {0}	202 {5}
	Basic	81 {1}	11 {1}	0 {0}	92 {2}

To better prepare users to create apps with more diverse functionality, we propose changes to the App Inventor learning resources to ensure tutorials are more accessible and contain more transferrable knowledge. We consider the following changes to App Inventor tutorials:

- Avoid Pre-defined Paths for Tutorials: As of now, App Inventor tutorials exist as a list where tutorials often do not build off of knowledge from previous ones. So, we suggest that tutorials be offered in such a way that users do not feel obligated to follow an unnecessary predefined ”path” when recreating tutorials. Instead, users should be more inclined to select tutorials relevant to their interests.
- Organizing Learning Resources: Because learning resources tend to be separate from each other on the MIT App Inventor website, users may not know where to go when they encounter a problem, or they may not even know given resources exist. In example, concept cards, which explain specific concepts, can serve as quick references for users [16]. They are placed with the teaching resources on the App Inventor website, entirely separate from tutorials. Ensuring that users have a centralized and organized point to access resources would better support users.
- Adaptive Tutorials: Suggested in a similar context by [9], an adaptive tutoring system that recommends resources relevant to what the user is creating and avoids resources of concepts already learned or implemented.[17]

V. CONCLUSION

In this paper, we evaluate the learnability (low floor) and capability (high ceiling) of App Inventor, a web-based

environment that enables users to create mobile apps. From our sample of 5,228 apps, we filtered out Certainly Static apps and apps that are recreations of our tutorials and grouped apps by similar functionality. We then measured the number of unique blocks for projects in each group and compared the groups. Our findings include (1) there exists a strong correlation between the learnability and the number of tutorial recreations for a given functionality; (2) users tend to follow tutorials sequentially and therefore often do not complete more than beginner tutorials; (3) users tend to develop apps that are functionally similar to completed tutorials. Based on these findings, we suggest that the realized capabilities of App Inventor are limited by the provided learning resources. We have provided a list of recommendations for improving these learning resources for end-users.

The existence of non-functional projects and recreations of tutorials in our dataset offer opportunities for future work. While we filtered out projects that were certainly static by ensuring all projects had the minimum number of components and blocks, there still exist projects that do not have any functionality. Disregarding blocks that are not connected to other blocks and components with no programmed functionality would better filter out the non-functional projects. And although we filter out projects that are recreations of tutorials by matching project names, a more robust method of identifying projects that are merely recreations would improve the dataset. We could focus analysis on specific advanced structures more closely tied with computational thinking such as iterators and procedures, as defined by [18]. Finally, we plan on analyzing user and temporal data, investigating how users and their apps develop over time.

Acknowledgments

We thank Jeffery Schiller (MIT App Inventor) for helping collect the data, Ilaria Liccardi (MIT) and Franklyn Turbak (Wellesley College) for helping guide the analysis, and Aubrey Colter (MIT App Inventor) for significantly helping proofread this paper.

This research is funded by the MIT EECS - Google Research and Innovation Scholarship as part of the 2014-15 MIT SuperUROP Program.

References

[1] "Mit app inventor — explore mit app inventor," 2015, last accessed 14-May-2015. [Online]. Available: http://appinventor.mit.edu/explore

[2] "Blockly," 2015, last accessed 10-April-2015. [Online]. Available: https://developers.google.com/blockly/

[3] F. Turbak, D. Wolber, and P. Medlock-Walton, "The design of naming features in app inventor 2," *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, last accessed 29-April-2015. [Online]. Available: http://dx.doi.org/10.1109/vlhcc.2014.6883034

[4] F. Turbak, "Mobile computational thinking in appinventor 2," 2014, slides of a talk given to CSTA-RI,April 10, 2014, Rhode Island College, RI. [Online]. Available: http://cs.wellesley.edu/~tinkerblocks/RIC14-talk.pdf

[5] D. Wolber, H. Abelson, and M. Friedman, "Democratizing computing with app inventor," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 4, pp. 53–58, 2015, last accessed 1-May-2015. [Online]. Available: http://dx.doi.org/10.1145/2721914.2721935

[6] A. V. Gershman, J. F. McCarthy, and A. E. Fano, "Situated computing: Bridging the gap between intention and action," in *Wearable Computers, 1999. Digest of Papers. The Third International Symposium on*. IEEE, 1999, pp. 3–9.

[7] "Tutorials for app inventor 2," 2015, last accessed 14-May-2015. [Online]. Available: http://appinventor.mit.edu/explore/ai2/tutorialsd41d.html

[8] S. Grover and R. Pea, "Computational thinking in k–12 a review of the state of the field," *Educational Researcher*, vol. 42, no. 1, pp. 38–43, 2013.

[9] S. Li, T. Xie, and N. Tillmann, "A comprehensive field study of end-user programming on mobile devices," in *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*. IEEE, 2013, pp. 43–50.

[10] "Python data analysis library," 2015, last accessed 5-May-2015. [Online]. Available: http://pandas.pydata.org/

[11] "Course in a box," this is a series of video tutorials. Online; last accessed 29-April-2015. [Online]. Available: http://www.appinventor.org/content/CourseInABox/Intro

[12] "App inventor tutorials," series of tutorials with pictographic explanations. Online; last accessed 04-May-2015. [Online]. Available: http://www.imagnity.com/tutorial-index

[13] L. Group. [Online]. Available: http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com

[14] M. Sherman, F. Martin, L. Baldwin, and J. DeFlippo, "App inventor project rubric - computational thinking through mobile computing." [Online]. Available: https://nsfmobilect.files.wordpress.com/2014/09/mobile-ct-rubric-for-app-inventor-2014-09-01.pdf

[15] A. Repenning, D. Webb, and A. Ioannidou, "Scalable game design and the development of a checklist for getting computational thinking into public schools," in *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 2010, pp. 265–269.

[16] M. A. Inventor, "App inventor concept cards," ast accessed 15-May-2015. [Online]. Available: http://appinventor.mit.edu/explore/resources/beginner-app-inventor-concept-cards.html

[17] "Scratch," 2015, last accessed 01-Aug-2015. [Online]. Available: http://scratch.mit.edu

[18] M. Sherman and F. Martin, "The assessment of mobile computational thinking," *Journal of Computing Sciences in Colleges*, vol. 30, no. 6, pp. 53–59, 2015.