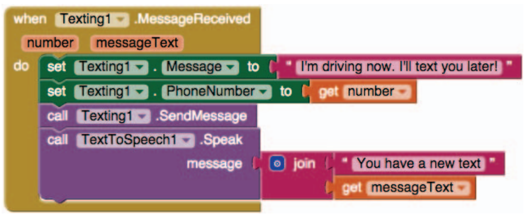
Skill Progression in MIT App Inventor

Benjamin Xie, Hal Abelson
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 01239
Email: {bxie, hal}@mit.edu

***Abstract*—This paper contributes to the growing body of research that attempts to measure online, informal learning. We analyze skill progression in MIT App Inventor, an informal online learning environment with over 5 million users and 15.9 million projects/apps created. Our objective is to understand how people learn computational thinking concepts while creating mobile applications with App Inventor. In particular, we are interested in the relationship between the progression of skill in using App Inventor functionality and in using computational thinking concepts as learners create more apps. We model skill progression along two dimensions: *breadth* and *depth* of capability. Given a sample of 10,571 random users who have each created at least 20 apps, we analyze the relationship between demonstrating domain-specific skills by using App Inventor functionality and generalizable skills by using computational thinking concepts. Our findings indicate that domain-specific and generalizable skills progress similarly; there is a common pattern of expanding breadth of capability by using new skills over the first 10 projects, then developing depth of capability by using previously introduced skills to build more sophisticated apps.**

I. Introduction

MIT App Inventor is an environment that leverages a blocks-based visual language to enable people to create mobile applications (”apps”) for Android devices [1]. An App Inventor project consists of a set of components and a set of program blocks that provide functionality to these components (Blockly, [2]). Components include items visible on the phone screen (e.g. buttons, text boxes) as well as non-visible items (e.g. camera, database, sensors). Figure 1 shows blocks used in an app that responds to text messages and reads them aloud.



```
when Texting1 .MessageReceived
  number  messageText
do  set Texting1 . Message to " I'm driving now. I'll text you later! "
    set Texting1 . PhoneNumber to get number
    call Texting1 .SendMessage
    call TextToSpeech1 .Speak
                  message  join  " You have a new text "
                                 get messageText
```

Fig. 1. Example of blocks programming language in MIT App Inventor. Upon receiving a text message, this program replies to the sender with a default message and reads the received message aloud.

App Inventor has been used in 195 countries and is taught in formal education environments and also self-taught [1]. It is taught to people ranging in age and experience from late elementary school students to professionals and end-user developers [3] [4]. Our analytics find that 50% of users program with App Inventor outside of formal educational settings. These self-taught learners primarily learn by following step-by-step app creation tutorials [5]. Our vision is to understand how people learn computational thinking using App Inventor so that these online, informal learning experiences can be integrated into STEM curricula.

This paper explores how people develop computational skills while creating apps with MIT App Inventor. In particular, we explore the relationship between developing skills that are specific to App Inventor (enabling app functionality) and developing skills that are generalizable to other programming domains (computational concepts). We measure skill progression across two dimensions: breadth and depth. Our data is a random sample of users who have created at least 20 projects. For each user, we measure breadth of capability by considering the number of new block types introduced at each projects and depth of capability by considering the total number of block types at each project. We separate blocks that relate computational concepts from other blocks and compare the progression of demonstrated skill.

We start by describing related work pertaining to skill progression and computational thinking frameworks (especially with Scratch), then detail our methodology. We show our results, comparing skill progression of using computational concepts with skill progression of using general App Inventor functionality. We then discuss our findings, noting a common pattern of developing breadth of capability before depth of capability. We note limitations and then list our contributions.

II. Related Work

Much of the recent research in measuring skill progression and computational thinking frameworks for open blocks programming environments has been done with Scratch, a blocks programming environment used to create media projects [6].

We adapt *computational (thinking) concepts* from the Scratch assessment framework from Brennan 2012 [7]. This computational thinking framework consists of three dimensions: computational concepts, computational practices, and computational perspectives. The development of computational concepts can be assessed by analyzing projects that users have created.

Scaffidi 2012 measured the skill progression of elementary programming skills in Scratch [8]. He found that the average breadth and depth of skill Scratch users demonstrated actually *decreased* over time. Explanations included early dropout of more skilled users, data inconsistencies, remixing, and community-wide decrease in complexity of projects. Matias 2016 replicated Scaffidi's studies and found opposite results (that breadth and depth of skill do not decrease over time), attributing Scaffidi's results to an unlucky data sample [9]. Both studies note a high dropout rate in the Scratch community.

Recent work in trajectory-based measures of cumulative repertoire of programming concepts has been done for Scratch by Yang 2015 [10] and Dasgupta 2016 [11].

III. Methodology

A. Data Source: Projects from long-term users

Our data is a random sample of 10,571 users who have each created at least 20 projects. We extract the types of blocks used in each project, omitting blocks that have no functionality by ignoring blocks not connected to a header block. (In Figure 1, the header/outer block is Texting.MessageReceived). Disregarding blocks without a header block removes a significant source of noise in the data [12].

B. Computational concept blocks reflect generalizable programming structures

We adapt computational concepts from the framework for assessing computational thinking in Scratch for use with App Inventor (see II). We define six computational concepts for App Inventor: {procedure, variable, logic, loop, conditional, list}. Of the 1,333 different types of blocks found in projects we analyzed, 39 of them are computational concept blocks (**CC blocks**) that relate to computational concepts. Figure 2 shows examples of CC blocks from each of the six concepts.

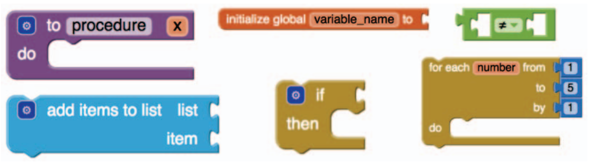


Fig. 2. Example of Computational Concept (CC) blocks. Clockwise from top left: Procedure, Variable, Logic, Loop, Conditional, List

C. Breadth: New block types in project

Breadth of capability reflects the broad understanding of knowledge and skill that users demonstrate. We model breadth of capability as the number of new, never before used block types in each of a user's projects.

We adapt the concept of a learning trajectory as originally defined for Scratch by Yang 2015 [10] to measure cumulative breadth of capability for a user across their first 20 projects. The notable difference with our approach is the omission of an Inverse Document Frequency (IDF) block weighting [13]. App Inventor has a larger feature set than Scratch, with a vocabulary size of 1,333 block types in this dataset, compared to 170 in Scratch [10]. Due to App Inventor's extensive feature set, IDF weighting would weight blocks pertaining to more obscure features heavily, rather than weight more advanced blocks heavily (as intended).

To model the breadth of capability:
(1) Isolate a specific set of block types, S. For our analysis, we choose the sets to be computational concept (CC) blocks and non-CC blocks. These sets are disjoint. (2) Create matrix P_{user}, which is the frequency of each type of block in each project. Each row is a project a user has created (in sequential order by creation time) and each column is the frequency of a certain block type. (3) Create the trajectory V_{depth} by summing the values in each row of P_{exist} (summing the total number of block types used in each project). (4) Create P_{sum}, the cumulative sum of P_{user}. (5) Create P_{binary}, which is an indicator matrix, by setting all nonzero values in P_{cum} to 1. (6) Create $V_{breadth}$ by summing the rows of P_{binary}. (summing the new block types used for the first time in a given project)

We then calculate T_{CC} (or T_{non-CC} depending on S) where each row is $V_{breadth}$ for a particular user. Each row of this matrix reflects the cumulative number of new block types introduced up to a given project for a user.

We also calculate the difference matrices $T_{diff,CC}$ (or $T_{diff,non-CC}$) by finding the first order difference between columns. These matrices measure the acquisition rate, or number of block types used for the first time at each project.

D. Depth: Total block types in project

Depth of capability refers to the mastery of certain features and functions. We model depth of capability as the total number of block types used in each of a user's projects.

To model depth of capability:
(1) Isolate a specific set of block types, S. For our analysis, we choose the sets to be computational concept (CC) blocks and non-CC blocks. These sets are disjoint. (2) Create matrix P_{exist}, which checks for the existence of each block type in each project (1 if in project, 0 otherwise). Each row is a project a user has created (ordered by creation time) and each column is the frequency of a block type. (3) Create the trajectory V_{depth} by summing the values in each row of P_{exist} (summing the total number of block types used in each project).

We then calculate D_{CC} (or D_{non-CC}) where each row is V_{depth} for a particular user. Each row reflects the number of block types used in a given project for a user.

IV. Results

A. Frequency of computational concept blocks

Figure 3 shows a histogram of the number of projects each CC-block appears in. The five most common CC blocks get a global variable value, declare a global variable, provide an if-else *statement*, set a global variable, and provide a boolean. The least used CC blocks pertain to more advanced list operations, local variables, while loops, and if-else *expressions* (expressions return a value; statements execute code).

We also find that more procedures are defined than called, suggesting some procedures are defined but never called. This may be because procedure definition blocks (top left in Figure 2) are header blocks, so they are counted even if no blocks are placed within the procedure (see III-A for more on header blocks). We also note that procedures without return values are defined and called over four times as often as procedures with return values. In the context of App Inventor, this suggests that procedures are often used to provide similar functionality to multiple components (e.g. 3 buttons providing color options for a painting app) but not often used to perform repeated calculations and return values.

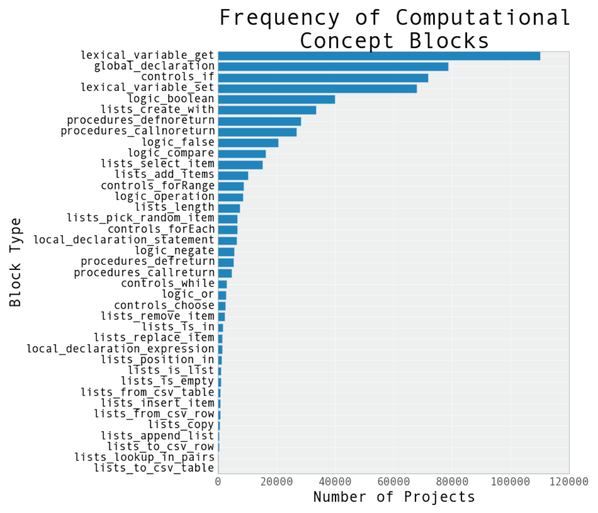


Fig. 3. Histogram of Computational Concept (CC) block types in projects

B. *Breadth of capability begins to plateau*

We measure the number of new blocks introduced at each of a user's projects to show how the progression of the breadth of capability to use App Inventor functionality relates to the breadth of capability to use computational concepts.

Figure 4 shows the cumulative sum of block types introduced at each project, averaged over all users. CC blocks and non-CC blocks are shown as separate trajectories. Given that there are a total of 39 CC blocks and 1,294 non-CC blocks, the divergence of CC and non-CC trajectories is expected. We see a decreasing rate of new block acquisition (fewer new blocks being introduced) as users create more projects.

It is also of note that even by the 20th project, the average number of CC and non-CC blocks used is nowhere near the total number of blocks (25% of CC blocks, 5% of non-CC blocks). This suggests that users are not exploring all of App Inventor's functionality and therefore are *not* being upper-bounded by the limits of the App Inventor environment (discussed further in [14]).

We normalize the trajectories of block acquisition in Figure 5. In this figure, a diagonal normalized learning rate would suggest users introduce new blocks at a constant rate across all projects. The progression of developing skills to use App Inventor functionality and the progression of developing skills to use computational concepts follow each other closely. This suggests that as users demonstrate domain-specific App Inventor skills, they also demonstrate generalizable skills using computational concepts. We note that the normalized CC trajectory lags slightly below the normalized non-CC trajectory

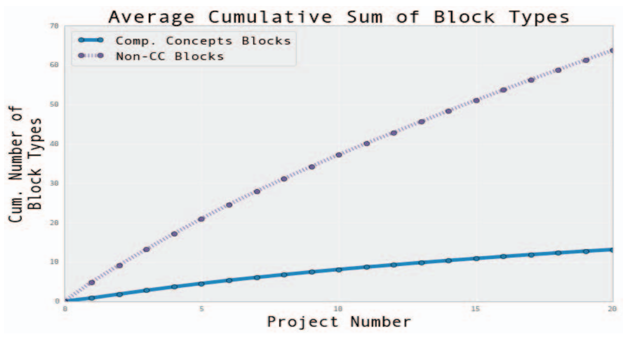


Fig. 4. Cumulative number of new blocks introduced at a given project, averaged across all users.

for the first 9 projects, perhaps reflecting a period where users familiarize themselves with the App Inventor environment.

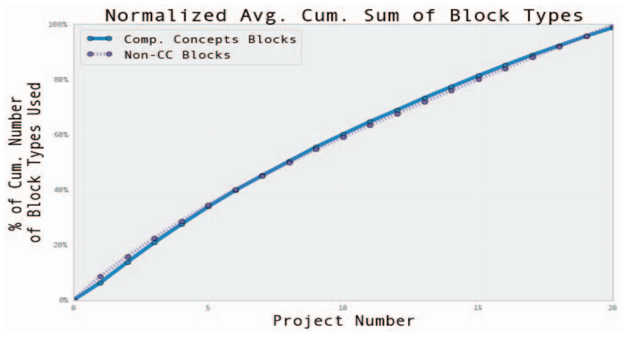


Fig. 5. Normalized average cumulative number of new blocks introduced at a given project.

The normalized rate of introducing new block types to projects (averaged across all users) is shown in Figure 6. As users create more projects, they introduce fewer blocks to their vocabulary, as evidenced by the decreasing rate of introduction. The rate of CC blocks introduction appears to decrease more substantially than non-CC blocks in the later projects (after 15). This could suggest that there may be a "saturation point" where a user's breadth of skill using CC blocks has encompassed roughly all of the computational concepts in App Inventor. We address this in V-B.

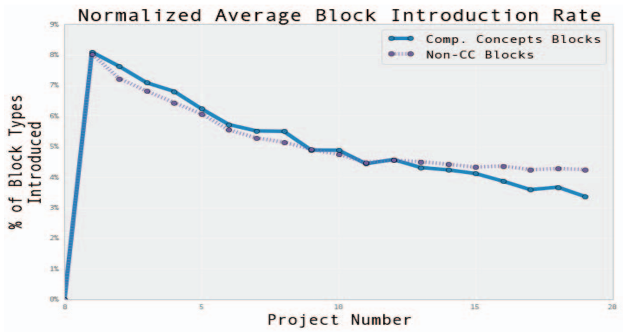


Fig. 6. Normalized average rate of introducing new block types to projects.

C. *Depth of capability increases*

We now analyze the depth of capability or mastery of features and functions. Our metric to measure depth of capability is the total number of block types used in a given project.

Figure 7 shows the total number of block types used in each project, averaged over all users. The first five projects likely reflect a period of familiarizing with App Inventor. After those projects, the increase in depth of capability is somewhat linear for both CC and non-CC blocks. In general, we see an increase of depth of capability as users create more projects, suggesting that as users create more projects, they tend to make more sophisticated apps that utilize more blocks to enable more robust functionality.

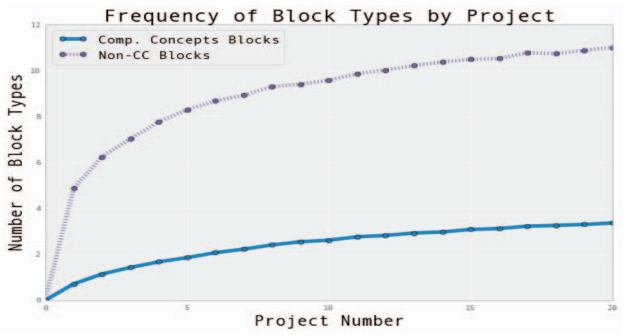


Fig. 7. Average number of block types used in each project (to measure depth of capability)

V. Discussion

We propose a common behavior of *breadth before depth* where users tend to familiarize themselves with a wide array of components and block types in earlier projects then develop mastery of previously used skills in later projects. While learners may only use a subset of CC blocks after 20 projects, they get exposure to almost all computational concepts. We do recognize that most users do not use App Inventor for long and must consider learning benefits for short-term use.

A. *Validating previous work on measuring skill progression*

In general, we see that both CC blocks and non-CC blocks follow a pattern of *breadth before depth*. As users create more projects, these projects tend to have fewer new block types introduced (decreasing breadth) but a greater total number of block types (increasing depth). It is significant that users tend to develop domain-specific skills relating specifically to App Inventor functionality (non-CC blocks) in a similar pattern as they develop generalizable skills using computational concepts (CC blocks). Previous work that used blocks to measure sophistication (in other environments) did not distinguish or isolate blocks that relate to generalizable skills [10], [15]. We believe our findings validate the use of blocks to measure sophistication even if the blocks are not directly related to generalizable skills.

B. *Subset of CC blocks reflects exposure to all concepts*

Learners only use a subset of the total CC blocks available in App Inventor but still get exposure to almost all computational concepts. After 20 projects, we find that on average, learners have used 25% of the 39 total CC blocks. We believe that this fraction of CC blocks at least touches on almost all computational concepts in App Inventor. The 8 most commonly used CC blocks include blocks from every computational concept except loops/iterators (see Figure 3)). App Inventor's event-based environment does not typically lend itself to using iterators. Using only a fraction of CC blocks is not necessarily alarming because most CC blocks relate to more obscure or specific functionality such as local variables and list operations. We believe that App Inventor enables at least exposure to computational concepts.

C. *Limitations: Using $\neq$ learning; most users aren't long-term*

Using a block in a project does not necessarily suggest that a user *learned* the skill relating to that block. So, measuring the use of blocks in projects may be a convenient metric to analyze at scale, but alone is not suitable for indicating whether a concept was truly learned.

As consistent with Scratch ([8], [9]), user retention is a challenge in App Inventor. This analysis looked at users who created at least 20 projects, which reflect only the top 1.5% of users. So, these users are not representative of a typical user. While we develop long-term curriculums and courses for online informal environments like App Inventor, we should also consider the beneficial short-term benefits for typical users who don't use App Inventor for long.

VI. Conclusion

Our long-term vision is to understand how people learn computational thinking in the informal online App Inventor environment. We take steps towards this vision by quantitatively analyzing the progression of using computational concept skills and modeling how users become more sophisticated with using these skills which generalize to other programming domains. We find evidence that users tend to learn new blocks in their earlier (first 10) projects, then use previously learned blocks in more sophisticated ways in later projects.

Contributions of this paper: 1) Modeled skill progression quantitatively across two dimensions (breadth, depth); 2) Verified relationship between the progression of domain-specific skills (using App Inventor functionality) and generalizable skills (using computational concepts); 3) Identified pattern of developing breadth of capability before depth of capability.

Acknowledgments

We thank members of the MIT App Inventor team, especially Jeffrey Schiller for data collection and Aubrey Colter, Emily Giurleo, Natalie Lao, and Aaron Suarez for proofreading. We also thank Sayamindu Dasgupta of the Scratch team for his guidance and suggestions.

REFERENCES

[1] "MIT App Inventor," accessed March 14, 2016. [Online]. Available: http://appinventor.mit.edu/explore/

[2] "Blockly," accessed March 14, 2016. [Online]. Available: https://developers.google.com/blockly/

[3] "Computing, Mobile Apps and the Web," accessed March 14, 2016. [Online]. Available: https://sites.google.com/site/appinventorcourse/

[4] "Mobile Computing with App Inventor CS Principles," accessed March 14, 2016. [Online]. Available: https://www.edx.org/course/mobile-computing-app-inventor-cs-trinityx-t002x

[5] "Tutorials for App Inventor," accessed March 14, 2016. [Online]. Available: http://appinventor.mit.edu/explore/ai2/tutorials.html

[6] "Scratch," accessed March 14, 2016. [Online]. Available: https://scratch.mit.edu/

[7] K. Brennan and M. Resnick, "New frameworks for studying and assessing the development of computational thinking," in *2012 annual meeting of the American Educational Research Association*, 2012.

[8] C. Scaffidi and C. Chambers, "Skill progression demonstrated by users in the scratch animation environment," *International Journal of Human-Computer Interaction*, vol. 28, no. 6, pp. 383–398, 2012.

[9] J. N. Matias, S. Dasgupta, and B. M. Hill, "Skill progression in scratch revisited," in *CHI 2016*, 2016.

[10] S. Yang, C. Domeniconi, M. Revelle, M. Sweeney, B. U. Gelman, C. Beckley, and A. Johri, "Uncovering trajectories of informal learning in large online communities of creators," in *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, 2015.

[11] S. Dasgupta, W. Hale, A. Monroy-Hernandez, and B. M. Hill, "Remixing as a pathway to computational thinking," in *19th ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW 2016)*, 2016.

[12] B. Xie, I. Shabir, and H. Abelson, "Measuring the usability and capability of app inventor to create mobile applications," in *Proceedings of the 3rd International Workshop on Programming for Mobile and Touch*, ser. PROMOTO 2015. ACM, 2015.

[13] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," in *Document Retrieval Systems*. Taylor Graham Publishing, 1988, pp. 132–142.

[14] B. Xie, "Progression of computational thinking skills demonstrated by app inventor users," Master's thesis, Massachusetts Institute Of Technology, 2016.

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