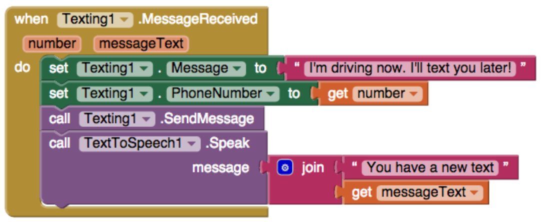
Breadth then Depth? Analyzing Trajectories of Informal Learning of Computational Concepts

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

***Abstract*—We analyze informal learning in MIT App Inventor, an open, online learning environment with over 4.4 million users and 13.5 million projects/apps created. Our objective is to understand the relationship between learning the breadth of App Inventor functionality and learning computational thinking concepts. We take steps towards this objective by modeling the learning trajectory of a user as the cumulative number of distinct blocks used at each of their projects. Given users who have created at least 20 projects, we compare the trajectory of learning computational concepts with the trajectory of learning App Inventor functionality. Our findings indicate an eventual plateau in the number of distinct concepts incorporated into new projects. We propose three hypotheses for the behaviors of learners during this time: focusing on improving understanding of previously learned computational concepts, applying previous knowledge to new scenarios, and iterating on past projects.**

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, images, text boxes) as well as non-visible items (e.g. camera, database, sensors). Figure 1 shows blocks used in an app that automatically 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 reached a broad, international audience from 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 and surveys 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 open, informal learning experiences can be integrated into STEM curricula.

This paper explores how increasing the breadth of App Inventor knowledge relates to learning general computational thinking concepts. Given users with at least 20 projects, we measure their learning trajectory by determining the cumulative number of blocks used for the first time in each project. We compare the trajectory of blocks relating to computational concepts with the trajectory of other App Inventor blocks.

We start by describing related work pertaining to learning trajectories and computational thinking frameworks, then detail our methodology. We show our results, comparing learning trajectories and learning rates of using computational concepts with using general App Inventor functionality. We then discuss our findings, noting a decreasing rate of using new blocks as users develop more projects and develop three hypotheses to explain the learning behavior as the breadth of knowledge plateaus. We note limitations and future work and then list our contributions.

A. Key Terms

We will refer to the following terms throughout the paper:

Learning Trajectory: We model the informal learning of a user as a trajectory of the cumulative vocabulary of blocks used across their first 20 projects. A value of n at a user's i-th project indicates that in the first i projects, the user has used a total of n types of blocks.

Learning Rate: The rate of change of the learning trajectory. A learning rate of n at project i says that n types of blocks were used for the first time at project i.

Computational Concept (CC): Concepts that are useful in programming with App Inventor and transferable to other programming languages. We refer to six concepts in our analysis: procedure, variable, logic, loop, conditional, list. These concepts are adapted from Brennan 2012 [6] (see II).

II. Related Work

Much of the recent research in learning trajectories and computational thinking frameworks for open blocks programming environments has been done with Scratch, a visual, blocks-based programming environment used to create media projects [7].

We reference *computational (thinking) concepts* from the Scratch assessment framework from Brennan 2012 [6]. This computational thinking framework consists of three dimensions: computational concepts, computational practices, and computational perspectives. Analyzing projects that users have created was shown to be effective at assessing computational concepts.

Work by Yang 2015 modeled learning trajectories and identified learning patterns at a microscopic (user) and macroscopic (cluster) level [8]. We will model learning trajectories in a similar fashion, looking at the number of blocks used for the first time in a given project. Dasgupta 2015 used learning trajectories to empirically verify that Scratch programmers could increase their programming skills and learn computational thinking concepts through remixing other users' code [9]. Previous empirical research by Scaffidi 2012 found that a decreasing level of demonstrated skill was observed as users created more projects in Scratch [10].

III. Methodology

A. Data Source

Given a sample of 10,000 random App Inventor users, we select all users who have created at least 20 projects (141 users). We then extract the types of blocks used in each project, removing blocks not connected to a header block because a block not attached to a header block has no functionality (In Figure 1, the header block is the outermost block: Texting.MessageReceived). Disregarding blocks without a header block removes a significant source of noise from the data, as identified by previous quantitative analysis with App Inventor [11].

B. Computational Concept Blocks

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 650 different types of blocks found in projects we analyzed, 38 of them are related to computational concepts. Figure 2 shows examples of *computational concept (CC) blocks* from each of the six concepts.

C. Learning Trajectories

We model informal learning as a trajectory of cumulative block use. With a temporal variable as the project sequence for the first 20 projects created, we measure the number of distinct block types used for the first time at a given project.

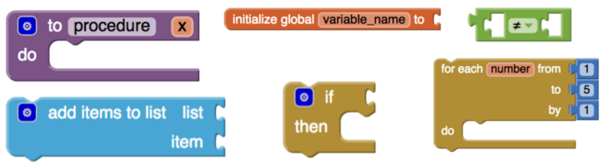


Fig. 2. Example of Computational Concept (CC) blocks. Clockwise from top left: Procedure, Variable, Logic, Loop, Conditional, List

Our approach to acquiring this learning trajectory is similar to Yang 2015, [8], with a notable difference being the omission of an Inverse Document Frequency (IDF) block weighting [12]. App Inventor has a larger feature set than Scratch, with a vocabulary size of 650 block types in this dataset, compared to 170 in Scratch [8]. 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).

We extract trajectories by doing the following for each user:

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 S for 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 P_{cum}, the cumulative sum of P_{user}.
4) Create P_{binary}, which is an indicator matrix, by setting all nonzero values in P_{cum} to 1.
5) Create V_{user} by summing the rows of P_{binary}.

We then calculate trajectory matrices T_{all} (or T_{cc}, T_{non-cc} depending on S) where each row is V_{user} for a given user. We also calculate difference matrices T_{diff} (or $T_{diff_{cc}}$, $T_{diff_{non-cc}}$ depending on S) by finding the first order difference of values between columns. These difference matrices measure learning rate.

IV. Preliminary 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 (`lexical_variable_get`), provide an if statement conditional (`controls_if`), set a global variable (`lexical_variable_set`), provide a boolean (`logic_boolean`), and create a list (`lists_create_with`). The least used CC blocks (bottom half/19) pertain to advanced list operations, local variables, and loops (while, for each item in list).

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.

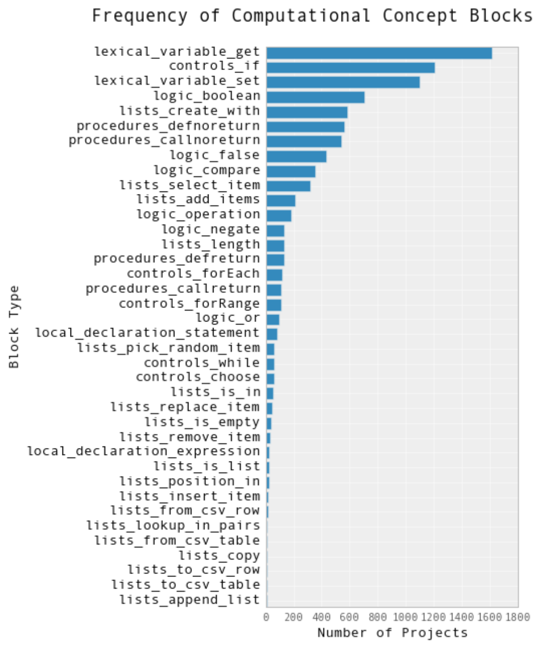


Fig. 3. Histogram of Computational Concept (CC) block types

B. *Comparing Computational Concept and Non-CC Trajectories, Learning Rates*

We show how increasing the breadth of App Inventor knowledge relates to learning computational concepts by comparing *learning trajectories* and *learning rates* for CC blocks with non-CC blocks (see I-A for definitions). The average learning trajectory for all 141 users is shown in Figure 4. Given that there are a total of 38 CC blocks and 612 non-CC blocks, the divergence of CC and non-CC trajectories is expected. We see that the learning rate decreases as users create more projects, showing that users introduce new types of blocks at a decreasing rate as they create more projects. This trend appears to be more extreme for CC blocks, as the CC trajectory plateaus more so than the non-CC trajectory.

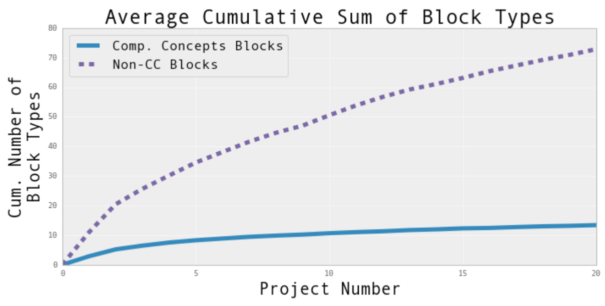


Fig. 4. Average learning trajectory of all users.

We normalize the trajectories in Figure 5. A diagonal normalized learning rate would suggest users introduce new blocks at a constant rate. For both CC and non-CC trajectories, we see that the introduction of new block types decreases as users create more projects. This decrease in learning rate is more pronounced for CC blocks, as evidenced by the higher curve in Figure 5.

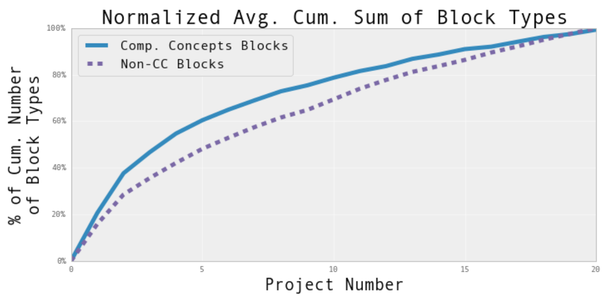


Fig. 5. Normalized average learning trajectory of all users.

The normalized learning rate is shown in Figure 6. As users create more projects, they introduce fewer blocks (CC and non-CC) to their vocabulary, as evidenced by the decreasing rate of learning. We also see that after 6-8 projects, normalized learning rate of CC stays below the normalized learning rate of non-CC.

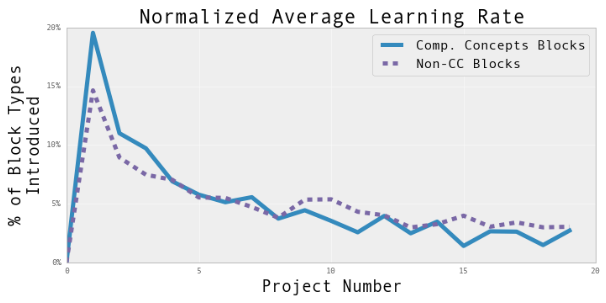


Fig. 6. Normalized average learning rate of all users.

The higher normalized learning trajectory for CC blocks in Figure 5 and lower normalized learning rate in Figure 6 suggest that users tend to learn computational concepts in their earlier projects but tend not to continue learning new concepts as they develop more projects. See V for further discussion.

V. Discussion

We focus our discussion on the general decrease in learning rate for both CC and non-CC and then specifically on the plateauing of the CC trajectory. We offer three hypotheses regarding learning patterns we observe from the data.

A. *General Decrease in Learning Rate*

The general decrease in learning rate for both CC and non-CC trajectories is consistent with previous findings in Scratch ([8], [10]), but we find that this decrease in learning rate is more pronounced in App Inventor. This is likely explained by the more diverse capabilities of App Inventor. This is best evidenced by the number of block types, with App Inventor having a vocabulary of 650 blocks (in our dataset) and Scratch having 170 blocks. Most of App Inventor's blocks are component-related, responding to events or enabling functionality. For example, a simple button component has 6 event blocks (e.g. when clicked), 14 setter functions (e.g. set visibility) and 13 blocks to retrieve properties (e.g. background color).

B. Plateauing Computational Concept Trajectory

The plateauing of the CC trajectory observed in the later $2/3$ of users' projects (Figures 4, 5) suggests that users develop their breadth of computational concept knowledge in earlier projects. We offer three hypotheses to explain the behavior of learners in later projects:

H1: Learners are developing their depth of knowledge. While learners may not be expanding the breadth of their blocks vocabulary by using new blocks, they may be using their knowledge of computational concepts learned in earlier projects in more sophisticated ways. Empirically, the number of CC blocks would increase in later projects while the CC trajectory (cumulative number of CC block *types* used) would plateau.

H2: Learners are applying their knowledge to different scenarios. Given the broad functionality of App Inventor, learners may be applying the same computational concepts learned in earlier projects to create apps of different functionality in later projects. Empirically, the CC trajectory would plateau but the non-CC trajectory would continue to increase. Given that App Inventor is common among end-user developers who tend to focus on creating apps to solve problems and are not interested in learning computational thinking, this is likely to be true for some of users of App Inventor.

H3: Learners are iterating on previous projects. Learners may be copying and iterating on previously created projects. These iterations may consist of aesthetic changes or incremental improvements to functionality. Empirically, both the CC and non-CC trajectories would plateau in later projects and the frequency of blocks to increase in later projects.

Figure 7 shows the average number of blocks used in each project. The jump in CC blocks in the first 3 projects is likely due to users recreating tutorials to learn how to use App Inventor and may not be representative of their actual knowledge. Ignoring that, we find a slight increase in the number of CC blocks as users create more projects. This could support our first hypothesis that users are improving their depth of knowledge. The jump in non-CC block frequency with minimal increase in CC block frequency in projects 11-13 could support our second hypothesis where learners are adapting computational concept knowledge to create apps of varying functionality. Further analysis can provide more insight into characterizing learner behavior.

C. Limitations

Our analysis of learning trajectories models the *breadth* of learning well but does not model the depth of learning.

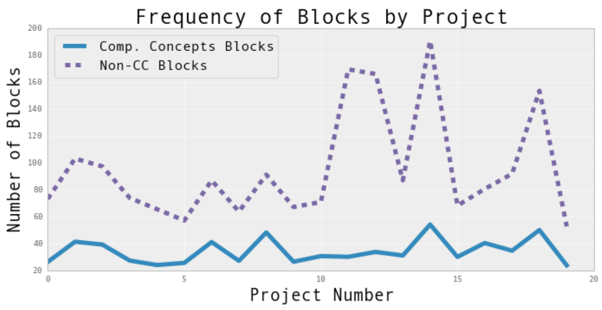


Fig. 7. Number of blocks in each project. Note that this is not cumulative and we are counting blocks (not types of blocks).

Focusing specifically on the reuse of computational concepts may also be of interest to measure depth of learning. For example, we would be interested in seeing if procedures are used in later projects after a learner uses it during a tutorial.

These tutorial recreations are also a source of bias in the data as blocks used to recreate a tutorial do not necessarily reflect the knowledge of a learner. Evidence of this may exist in the temporary increase in blocks frequency in the 2nd and 3rd projects in Figure 7.

D. Future Work

We plan on clustering users and qualitatively describing clusters based on representative users. We are interested in the number, order, and type of tutorials users recreate as this can shed light on their prior knowledge [8]. We are also interested in the order in which computational concepts are learned and the diversity of a user's projects.

VI. Conclusion

Our long-term vision is to model and understand how people learn computational thinking in the open, informal App Inventor environment. We take steps towards this vision by comparing learning computational thinking concepts with learning the breadth of the App Inventor functionality. We find evidence that, on average, the breadth of learning tends to slow in growth by a user's eighth project. We offer three hypotheses to explain learners' behaviors after this plateau.

Our contributions: 1) Modeled the learning of computational concepts as a trajectory; 2) Compared the learning trajectories and learning rates of computational concepts with breadth of App Inventor functionality; 3) Proposed three hypotheses regarding learning patterns after the rate of learning new computational concepts (breadth of learning) decreases.

Acknowledgments

The work done at the MIT Center for Mobile Learning is very collaborative and this paper is no different. We thank members of the MIT App Inventor team, especially Jeffrey Schiller for help collecting the data and Aubrey Colter, Emily Giurleo, Natalie Lao, and Aaron Suarez for help clarifying the writing. We also thank Sayamindu Dasgupta of the Lifelong Kindergarten group at the MIT Media Lab 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] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.