QuizPower: a mobile app with app inventor and XAMPP service integration

David Meehan
University of New Hampshire, Manchester

Mihaela C. Sabin
University of New Hampshire, Durham, mihaela.sabin@unh.edu

QuizPower: A Mobile App with App Inventor and XAMPP Service Integration

David Meehan
University of New Hampshire
88 Commercial Street
Manchester, NH 03101
603 454 6552
david.meehan.unh@gmail.com

Mihaela Sabin
University of New Hampshire
88 Commercial Street
Manchester, NH 03101
603 641 4144
mihaela.sabin@unh.edu

ABSTRACT

This paper details the development of a mobile app for the Android operating system using MIT App Inventor language and development platform. The app, Quiz Power, provides students a way to study course material in an engaging and effective manner. At its current stage the app is intended strictly for use in a mobile app with App Inventor course, although it provides the facility to be adapted for other courses by simply changing the web data store. Development occurred during the spring semester of 2013. Students in the course played a vital role in providing feedback on course material, which would be the basis for the structure of the quiz as well as the questions. The significance of the project is the integration of the MIT App Inventor service with a web service implemented and managed by the department.

Categories and Subject Descriptors

D.1.7 [**Visual Programming**]; D.2.6 [**Programming Environments**]: Graphical environment; H.3.5 [**Online Information Services**]: Data sharing

General Terms

Design, Languages, Experimentation

Keywords

App Inventor, mobile computing, data store web service

1. MOTIVATION

1.1 Mobile Computing First and For Most

Computing Technology program at University of New Hampshire at Manchester has expanded its computing curricula with an introductory course that satisfies the University's Environment, Technology, and Society general education requirement, CIS 415 Mobile Computing First and For Most. The course teaches computational thinking in the context of solving problems of social and environmental relevance and emphasizes issues of inclusiveness of diverse cultural backgrounds, life experiences, and talents that all students bring to class. Students learn how to create mobile apps for Android operating system with App Inventor, a visual blocks-based programming language and development platform. The App Inventor project, led by Hal Abelson from MIT and Mark Friedman from Google was released publicly in 2010. App Inventor codebase, run-time environment, app projects' cloud storage, and community resources are now supported and hosted by the MIT Center for Mobile Learning [1].

COMP 415 was first offered in Spring 2013. The class had fifteen students: ten students in the B.S. Computer Information Systems (CIS) program and five students majoring in History, Communication Arts, Political Science, Biology, and Engineering Technology. Although enrollment sample size is small, gender and race demographic data breakdown of 13% female students (both CIS majors) and 20% underrepresented minorities in computing showed a higher participation from underrepresented groups than gender and race representation percentages in the CIS program (10% females and 5% underrepresented minorities).

The course requirements consisted of eight homework assignments (16% of the final grade), two exams (48%), one 6-week creative project developed by teams of two students (10%), one 6-week creative project developed by teams of five students (20%), and project presentation at the University's Undergraduate Research Conference (6%).

1.2 Guided Practice and Self-Assessment

App Inventor with Android smartphones, coupled with a studio-based teaching and learning model [2], suggests a positive effect on students' motivation, creativity, achievement, and attitudes towards computer science [3]. Class pedagogy of studio-based learning was further inspired by the course textbook [4] and its scaffolding approach to learning from telling the app's user story to creating an original app:

tell → build →customize →conceptualize → create

In this model, when students progress through these five layers, the scope of guidance they receive decreases, while expectation for higher-order cognitive skills increases.

Students in CIS 415 reported more study time out of class than in other classes and were fully engaged in peer learning activities: pair programming lab exercises; structured class forum participation; collaborative project activities during 3-hour weekly class meetings; and further collaboration outside class to share and review artifacts with Google Drive and Google Sites services.

SIGITE/RIIT'13, October 10 - 12 2013, Orlando, FL, USA Copyright 2013 ACM 978-1-4503-2239-3/13/10…$15.00.
http://dx.doi.org/10.1145/2512276.2512300

Students became proficient with *building* apps by following tutorial instructions. The use of peer learning helped students meet the *customization* requirements of apps built from tutorials. What remained a challenge was *conceptualizing* programming language constructs and design techniques. These competencies were critical for scaffolding competencies to *create* apps from description of the app's features and user stories.

To help students achieve conceptual learning outcomes, homework assignments and creative project iterations were enhanced with worksheets that required students to practice test questions. Retrieval, as a learning activity, has been found to be a powerful way to enhance conceptual learning in science [5]. To make retrieval practice integral part of the class learning activities, four worksheets with more than 80 questions were assigned over the course of the semester. A CIS senior and student tech consultant in the department (this paper's first author) and the course instructor (paper's second author) developed and refined the worksheets' questions and answer keys.

We had plenty of opportunities to learn about students' struggle with concept comprehension and inferences from in-class activities, tutoring and help sessions, and class forum discussions. Our quest for composing relevant questions and improving the clarity and conciseness of the answers gave us the idea of QuizPower, an app to guide students through a self-directed assessment and practice with test questions.

2. OBJECTIVE

QuizPower research and development project was intended to:

- Guide students in their practice with test questions that assess comprehension and require inferences;
- Be a case study of a creative project with educational relevance for students.

The prototype we developed is an interactive tool with which students:

(1) Get access to questions organized by topics
(2) Select topics to study and quiz themselves
(3) Go through topic questions in a random order
(4) Enter their answer
(5) Compare their answer with the tool's answer
(6) Self-assess quality of answer to remove question from further study if answered adequately
(7) Receive feedback on progress based on how many questions were answered correctly from those attempted.

Non-functional requirements of the project were that:

(1) QuizPower be developed primarily in App Inventor
(2) Questions be stored in a web data store.

3. PROJECT APPROACH AND METHODS

3.1 App Inventor Platform

App Inventor is a free and open source server software that runs in the MIT App Inventor's cloud. Developers need a Google account to access the App Inventor's development service[1]. To design an app's user interface and decide on the structure of its architectural components, developers need to use a browser, in which they run the Component Designer tool. To implement the app's behavior and logic of the component methods, developers use the Blocks Editor, a Java web application that is downloaded on demand from the Component Designer on the developer's client computer.

The App Inventor language has a rich library of component classes[2]. Component classes represent basic user interface facilities (text box, button, canvas, arrangement, etc.), sprites (touch-sensitive objects), and a wealth of built-in, smartphone-specific functional features (texting, phone calling, contact search, sensors, camera, sound player, video player, etc.). Apps developed with App Inventor can be compiled and packaged into an application package file (APK) to be distributed and installed on Android mobile devices. The packaging command, available in the Component Designer, generates the APK file and downloads it to the client computer, a USB-connected phone, or a phone connected over Wi-Fi. The downloading over Wi-Fi option requires the MIT AI Companion app, which establishes a connection between the Android mobile device and client computer over the MIT RendezVous Server.

3.2 Software Development Process

We developed the project in spring 2013 over a ten-week time period using an agile software development approach with weekly sprints. At the beginning of each sprint we chose the sprint backlog items. We held two or three meetings every week to track our progress and update the backlog of work items.

Before we sketched user stories and user interface mockups, we sampled student questions and answers from a variety of sources: class forum, in-class lab activities, student assignment self-evaluations, and help sessions. Our goal was to create practice worksheets that were informed by student common misconceptions. We classified over 80 questions into sixteen topical categories: animation, architecture, components, conditionals, control flow, data types, debugging, events and event handlers, expressions, lists, loops, procedures, sensors, software development process, variables, databases and web services. These questions became the basis of the students self-directed assessment and practice we envisioned for the QuizPower app.

As explained in the following sections, architectural and implementation decisions have changed during development iterations in order to address performance and maintainability issues.

3.3 Quiz Questions Data Store

3.3.1 App Inventor's Google App Engine Database Service

Early on in the development process we determined that the use of a data store was necessary to reduce the coupling between the quiz questions and the QuizPower engine. The data store also allowed us to experiment with different branches of the code base independently of acquiring, refining, and encoding quiz questions.

App Inventor library has a database component class called TinyWebDB. This class facilitates communication between the app and a specialized web-based database of (tag, value) pairs, hosted through the Google App Engine. TinyWebDB component has StoreValue( ) and GetValue( ) methods, by which tagged values are placed into and retrieved from the database. When a GetValue( ) server request succeeds, a GotValue event is

[1] http://beta.appinventor.mit.edu

[2] http://beta.appinventor.mit.edu/learn/reference/

generated with (tag, value) information that is made available to the app's logic (Figure 1).

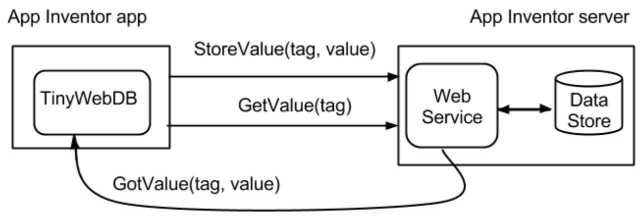


Figure 1: App Inventor communication with Google App Engine web service via TinyWebDB methods and event handler[3].

Data store hosting enabled by the Google App Engine has the advantage of eliminating any administrative overhead, including administration tasks for scaling the service instance. The database administration tasks left to us were to manage entries in the data store (get, store, view, and delete operations), control write access to the data store for testing purposes only, and monitor usage.

By default, TinyWebDB component communicates with a web service[4] that manages a data store shared by all possible App Inventor users. The service limits, however, the number of database entries to 1,000 per user. To create a custom web service for our QuizPower app, we used the Google App Engine Launcher client application to:

- Adapt a custom web service template in the textbook [4] such that it uniquely identifies the QuizPower data web store among Google App Engine web services
- Set QuizPower web service properties (title, URL, and other web service required properties).

The drawbacks with this solution were query performance and ease of data manipulation. We did not want to limit the number of questions per topic that a student would be interested in practicing with. What we observed, however, was that the time to retrieve the questions for a single topic (on average 20 questions/topic) was more than one second (on average 3 seconds) when we run experiments that were using the college's Wi-Fi network. This was not problematic until the interface was refined to include the possibility of a student requesting questions from more than one topic.

We also expected to be able to change quiz question data in the web data store easily and efficiently. Google App Engine data web service has very simple data manipulation operations. There is no facility to store multiple entries from a data file or selectively update or delete multiple entries from the data store based on filtering conditions. Given these restrictions, our focus shifted to a relational database solution that integrates MySQL with App Inventor.

3.3.2 MySQL Service Integration

Google App Engine supports a SQL database instance via Google Cloud SQL. This is a fully managed web service that offers a scalable MySQL database running on Google infrastructure. Although this service met our performance needs, it is not a free and open source software solution, and we did not have the expertise to delve in it right away. The question we were interested in was whether we can integrate our own instance of a MySQL database with App Inventor.

To establish communication between the QuizPower app and MySQL database instance, we used the App Inventor Web component class. This class has methods that make HTTP GET and POST requests to a given URL to get or post data in text stream format. There are two methods of the Web component class that we used for integration with a relational database:

1. get( ) request - initiates communication with the web app specified in the URL property of the Web component instance. SaveProperty value of the WebService instance determines whether a GotText or GotFile event is triggered. In our implementation, this value is set to false to trigger GotText event. It performs an HTTP GET request.
2. gotText( ) event handler - takes four parameters: URL, response code, response type, and the actual content of the response. Response content is in text stream format and received upon successful completion of the get( ) request.

To bridge the App Inventor communication with a relational database, we configured a run-time environment that had MySQL, PHP, and Apache web server support bundled together. We used a server machine managed by our department and installed a XAMPP[5] stack instance. Next, we created a relational database using MySQL to store quiz questions organized by topics. We then developed a PHP application to retrieve and produce a text stream with quiz questions from the database – all of them or filtered by topics. In the QuizPower app, we created WebService, an instance of the App Inventor's Web component class, and initialized it with the URL of the PHP app.

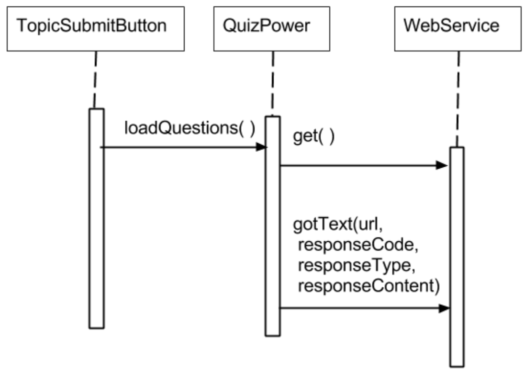


Figure 2 QuizPower interaction with WebService.

In Figure 2, we show the interaction between the user's selection of topics and the WebService operations that perform data store retrieval of topic questions.

[3] Figure 1 is adapted from Chapter 10 Persistent Data [4].

[4] http://appinvtinywebdb.appspot.com

[5] Free and open source cross-platform distribution of Apache, HTTP server, MySQL database, and interpreters for scripts in PHP and Perl (www.apachefriends.org/en/xampp.html)

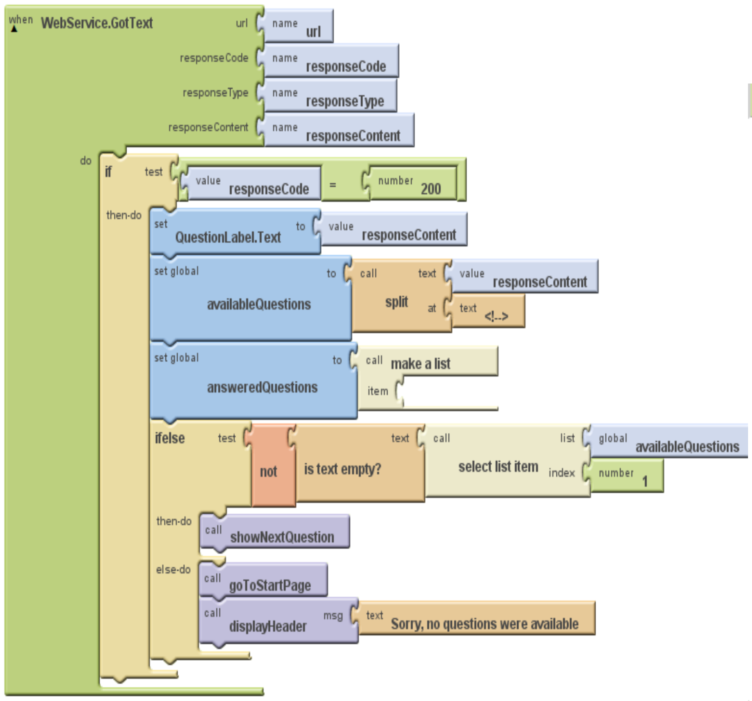

when WebService.GotText
url name url
responseCode name responseCode
responseType name responseType
responseContent name responseContent
do if test value responseCode = number 200
then-do set QuestionLabel.Text to value responseContent
set global availableQuestions to call split text value responseContent at text <!-->
set global answeredQuestions to call make a list item
ifelse test not is text empty? text call select list item list global availableQuestions index number 1
then-do call showNextQuestion
else-do call goToStartPage
call displayHeader msg text Sorry, no questions were available

Figure 3. QuizPower Blocks Editor view of the gotText event handler.

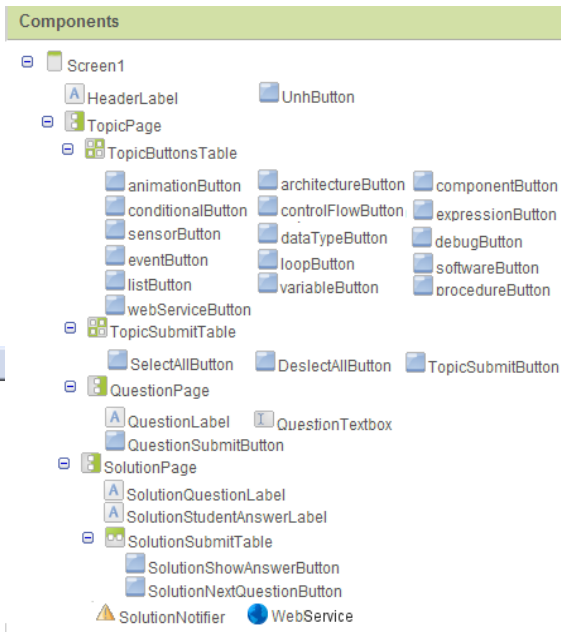


Figure 4. QuizPower's Component Designer view.

After a student selects any number of question topics, the QuizPower app calls WebService.get( ) to make a request to the URL of the PHP application. The underlying HTTP GET request carries topic information in the URL query string. The PHP application extracts topic information from the $_GET associative array and communicates with the MySQL database to retrieve a list of corresponding questions and answers. The list is then "echoed" into a dynamic HTML page that is sent back as a text stream captured by the responseContent parameter of the WebService.gotText( ) event handler. Figure 3 illustrates the App Inventor implementation of the GotText( ) event handler.

4. RESULTS

The result at the end of the spring 2013 semester was a mobile application developed with App Inventor for the Android platform. The app maintains a list of questions related to topics the student selects when starting the app. The decisions we made during development have improved the app's performance and usability. The app can be easily scaled to work with any number of quiz questions and topics by simply adding new questions and topics to the database and corresponding selection buttons to the app's interface.

4.1 Components

The foundation for all apps developed in App Inventor is a rich library of component classes. Component instantiations from these classes are objects that perform the app's work. When creating components, the developer initializes the component property values.

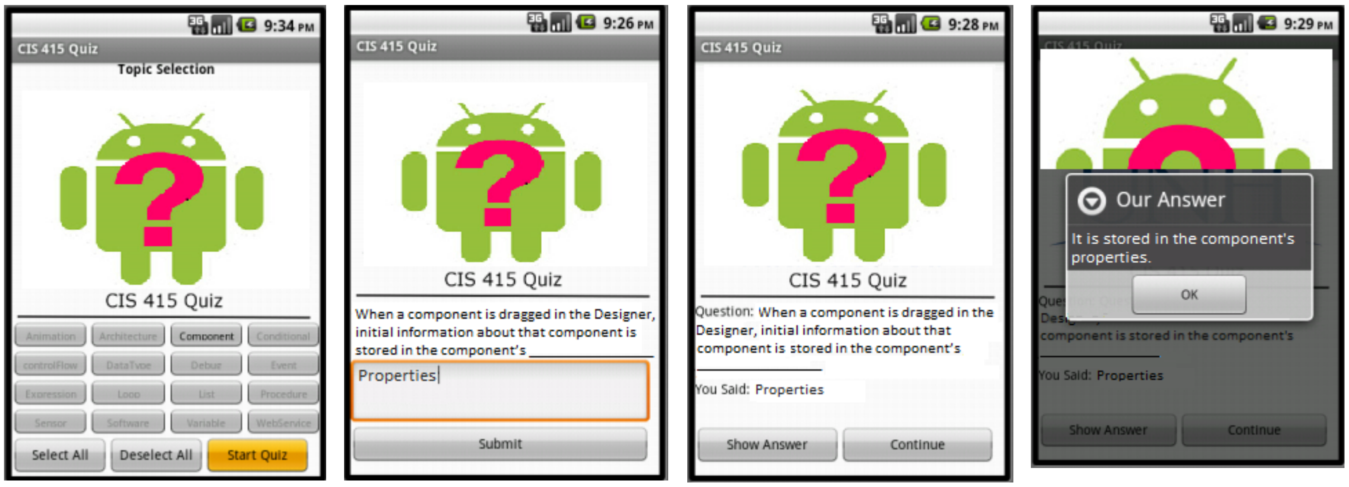
Figure 4 has the list of all components used by the Quiz Power app. The object hierarchy shows aggregation associations among QuizPower components, as displayed in the Component Designer tool. Except for the SolutionNotifier and WebService components, all the other components model the app's interface, with buttons, labels, and horizontal and table arrangements.

4.2 User Interface

User interface arrangements are hidden or displayed depending on the user activity. In total, there are three main views: topic selection view modeled by the TopicPage, question view modeled by the QustionPage, and the solution view corresponding to the SolutionPage (Figure 4).

When students first start the app, they are presented with the topic selection view (Figure 5). Upon making their topic selection, they see the question view, in which they answer questions picked at random from the selected topics. Upon submitting an answer to a question, the solution view presents the correct answer. Students self-assess their answer and continue practicing with the remaining questions.

Our initial design had multiple-choice questions. In that scenario, the student answer was checked against a correct choice. Although this version was simple to implement, figuring out meaningful choices to pick from was the difficult part. Literature on peer instruction with concept test questions points to the challenge of writing effective multiple choice questions [6]. We decided to start with short answer questions and gain more insight about student misconceptions before moving to multiple-choice questions.



When a component is dragged in the Designer, initial information about that component is stored in the component's ________

Question: When a component is dragged in the Designer, initial information about that component is stored in the component's

You Said: Properties

It is stored in the component's properties.

Figure 5. QuizPower user experience sample.

The final result was a simple solution that displays the student answer along with the correct answer. In this case students could use prompt feedback to judge the accuracy of their answers. In Figure 6 we show the activity diagram of the app's user interface.

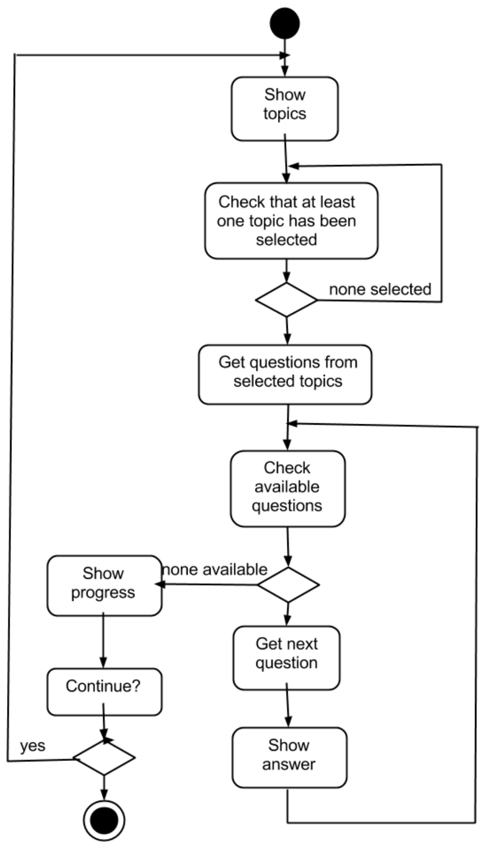

Show topics
Check that at least one topic has been selected
none selected
Get questions from selected topics
Check available questions
none available
Show progress
Continue?
yes
Get next question
Show answer

Figure 6. Activity diagram of the QuizPower user interface.

4.3 MySQL Integration

The most important finding of the QuizPower development project was the integration of MIT's App Inventor development services with a web service hosted by the department infrastructure. The MySQL database implementation proved to be successful in reducing query times and simplifying database management operations. The end result was a high performance question and answer retrieval app, which could scale well regardless of the number of topics selected. Using a MySQL database instead of the conventional TinyWebDB service simplified our App Inventor codebase and eliminated tedious data manipulation and encoding schemes imposed by TinyWebDB. MySQL integration with App Inventor via a PHP application opens many opportunities for using App Inventor platform to develop more sophisticated apps with much larger and complex data stores.

5. CONCLUSION

The goal of our project was to develop QuizPower, an app for Android mobile devices using the App Inventor language and MIT App Inventor development and packaging service. Our motivation for creating QuizPower is supported by evidence that retrieval practice is a powerful way to promote meaningful learning of complex concepts commonly found in science education. With QuizPower, students in a general education course for all majors could practice test questions to better learn concepts in a mobile computing course. The project's current product is a working prototype written in App Inventor, PHP, and MySQL. This implementation solution integrates App Inventor development and packaging service with a XAMP-based execution environment.

Further work with this project has multiple directions. First, we plan to deploy the app in a live, production environment for use in the CIS 415 course in our department, as well as similar courses in other STEM disciplines in our college. The goal of making QuizPower publicly available is to improve and expand the collection of test questions. Second, we are interested in converting the app's codebase from App Inventor to Java to gain more control over the app's performance and functionality. We would also like to provide students with a preliminary evaluation of their answers based on the presence of desired keywords. A keyword could be a simple word or grouping of words with similar meaning. Finding keywords indicative of the goodness of an answer would require learning from the app's usage and application of machine learning techniques.

Finally, we plan to develop a MakeQuizPower app for course instructors to manage the topics and questions in MySQL database, thus eliminating the need to do any SQL programming.

6. ACKNOWLEDGMENTS
We would like to acknowledge the Spring 2013 CIS 415 students, who worked with us in the development of this app.

7. REFERENCES
[1] MIT Center for Mobile Learning. 2013. MIT App Inventor. Retrieved from http://appinventor.mit.edu.

[2] Brown, J.S. 2006. New Learning Environment for the 21st Century: Exploring the Edge. *Change* (September/October).

[3] Ahmad, K. and Gestwicki, P. 2013. Studio-based learning and App Inventor for Android in an introductory CS course for non-majors. In *Proceeding of the 44th ACM technical symposium on Computer science education* (SIGCSE '13). ACM, New York, NY, USA, 287-292.

[4] Wolber, D., Abelson, H., Spertus, E., and Looney, L. 2001. App Inventor: Create Your Own Android Apps. O'Reilly Media.

[5] Karpicke, J. D., & Blunt, J. R. 2011. Retrieval practice produces more learning than elaborative studying with concept mapping. *Science*, 331, 772-775.

[6] Grissom, S., Simon, B., Beck, L., and Chizhik, A. 2013. Alternatives to lecture: revealing the power of peer instruction and cooperative learning. In *Proceeding of the 44th ACM technical symposium on Computer science education* (SIGCSE '13). ACM, New York, NY, USA, 283-284.