Reflections on Teaching App Inventor for Non-Beginner Programmers: Issues, Challenges and Opportunities

Andrey Soares
asoares@siu.edu

Information Systems Technologies
Southern Illinois University
Carbondale, IL 62901, USA

Abstract

App Inventor has been used successfully to teach introduction to programming course for CS/IS/IT and Non-CS majors. Now, researchers are looking on how to include the tool in the curriculum of more advanced computing courses. This paper presents some Issues, Challenges and Opportunities observed while teaching courses on Mobile Application Development with App Inventor. In particular, this paper discusses the following topics that instructors should take into consideration when designing their courses with App Inventor: Pre-Requisite for the course, Visual vs. Textual Programming, Planning and Designing Apps, the use of Web Services, students new to Event-driven programming, the use of database and SQL, Lists, designing User Interfaces, discussing Data communications, and the Use versus the Creation of objects. The paper shows that App Inventor has great potential to be used for teaching more advanced computing concepts. For some of the topics, students may be required to have more than just basic programming skills.

Keywords: App Inventor, Mobile Applications, Non-Beginners, Programmers

1. INTRODUCTION

Since its release in 2010, App Inventor has been used as a teaching tool in many schools and universities. It seems that a common use of the tool is for teaching introduction to programming skills for (1) middle and high school students, (2) for beginners in Computer Science, Information Systems or Information Technology (CS/IS/IT) or other related technical majors, and (3) for non-CS/IS/IT students. In fact, App Inventor has been advertised as a tool that you can create your own Apps with no programming experience required (Tyler, 2011; Wolber, Abelson, Spertus, & Looney, 2011). The tool has also been used in a variety of educational events and formats from a few days of workshops and summer camps to semester-long courses. We can see an increasing number of publications reporting the success of using the tool for teaching as well as for recruitment and retention efforts.

In the Fall of 2011, I was exploring some websites in search of apps created with App Inventor. After checking several apps, I started wondering if students that are new to programming are creating such interesting apps, what could be created if they already have programming experience. Since then, I have created and offered face-to-face and online courses on App Development using App Inventor where the pre-requisite for the course is an introduction to programming course.

Many of the apps created by the students are available online through their personal web pages or the web pages of their course or instructor. Recently, the MIT Center for Mobile Learning has made available the App Inventor Community Gallery, an open-source repository of apps created with App Inventor (http://gallery.appinventor.mit.edu).

In this paper, I discuss issues, challenges and opportunities observed while teaching the course at a Midwest University. The discussions represent my experiences and observations of class activities and informal conversations with students.

2. BACKGROUND

App Inventor for Non-Beginners

App Inventor is a visual programming language developed by Google in 2010 and currently hosted and maintained by the MIT Center for Mobile Learning. It has been successfully used to teach introductory computer science concepts (CS0) and introduction to programming (CS1) skills for students in CS and Non-CS majors. In fact, not only CS but also the fields of Information Technology (IT) and Information Systems (IS) are using similar approaches. It is possible to see the terms CS0, IS0 and IT0 used interchangeably (Uludag, Karakus, & Turner, 2011) as well as the terms CS1, IS1 and IT1 (Lim, Hosack, & Vogt, 2010).

Professor David Wolber, from the Computer Science Department at the University of San Francisco, has created a set of course materials that can be used to teach introductory CS concepts for Non-CS majors (CS0 course) and can be adapted to teach CS majors (CS1 course). Dr. Wolber's Course-in-a-Box materials (www.appinventor.org/course-in-a-box) includes modules on Introduction to Event-Driven Apps, Games, Text/Location and other Mobile Technology, Data, Shared Data, Apps that Access Web Data, and Software Engineering and Procedural Abstraction. Similar materials are needed to support the teaching of more advanced computing and programming concepts for non-beginners.

Gestwicki & Ahmad (2011) suggest that App Inventor and their Studio-Based Learning approach can be used not only to "introduce non-CS majors to concepts of Computer Science-not just programming, but also ideas that tend not to be covered in conventional CS1 courses such as human-computer interaction, incremental and iterative design processes, collaboration, evaluation, and quality assurance" (p. 55).

Karakus, Uludag, Guler, Turner, & Ugur (2012) also argue that App Inventor can be used in CS2 courses for computing majors. In particular, they contend that in a CS2 course "the emphasis is shifted more to the inner details of programming constructs, such as control structures, iteration, functions, recursion, algorithms, decision making, some basic data structures, etc." (p. 5). In addition, they consider that Robotics, Software engineering, Information Systems, and Networking, Database and Web Development courses could incorporate App Inventor into their curriculum. Arachchilage, Love, & Scott (2012), for instance, have demonstrated the use of App Inventor to create a mobile game to teach users about conceptual knowledge of avoiding phishing attacks, which is a form of online identity theft.

The MIT Center for Mobile Learning at the MIT Media Lab hosts the Annual App Inventor Summit, an event designed for educators and experienced users of App Inventor. In the 2012 App Inventor Summit, a working group discussed the role of App Inventor in CS/IS Education and its use in more advanced courses.

This paper is a contribution to the discussion of using App Inventor beyond the CS1/IS1/IT1 courses where students have taken additional programming or other related technical courses such as object-oriented programming, web development, database design, and software engineering.

The Course

The course was designed to allow the students to explore the features of Android phones by using App Inventor components, rather than being another elective programming course. So, the pre-requisite for registering to the course is to have taken some introduction to programming course where students would have been exposed to basic programming concepts, including logic, conditions, loop, variables, procedures, input and output. Nonetheless, senior students would more likely have already taken other upper division courses such as Web programming, Object-Oriented Programming, and Software Engineering. The visual approach of App Inventor would help students not to focus on programming and the syntax of coding.

Instead, they would concentrate more on the logic and events of the application.

The course uses the book *App Inventor: Create Your Own Android Apps* (Wolber et al., 2011) as reference. The book is available online in PDF format at www.appinventor.org/projects. The course starts by covering topics from the book, and then new topics are added or removed as needed to augment the students learning experience. Because of the required prerequisite for the course, the topics on fundamentals of programming are not covered in details. See some resources for instructors in the Appendix.

3. ISSUES, CHALLENGES AND OPPORTUNITIES

Teaching App Inventor for students with previous programming experience presents some challenges that instructors should take into consideration when designing their courses. Several assignments completed in class have provided great insights on using App Inventor to teach not only programming but also other computing concepts. Following is a list (in no particular order) of Issues, Challenges and Opportunities observed while teaching App Development with App Inventor for non-beginner programmers.

Pre-Requisite

The pre-requisite for the course is some Introduction to Programming course. The rationale is that the time used for teaching logic and the fundamentals of programming could be used to explore more features of the phone and the App Inventor tool. With this approach, students would just adapt their programming skills to the new environment; and the instructor would just show how things are done within the new environment.

While all the students have taken a programming course prior to the course with App Inventor, the level of programming skills may vary from student to student. On one hand, students may still be new to programming as they have taken the introduction to programming course in the semester prior to the App development course. In some cases, the last (or the only) programming course was taken between one and two years ago. On the other hand, students may be more experienced programmers as they have taken more courses in the area of application development and programming such as Object-Oriented programming, Client-Side and Server-Side Web Development, and Software Engineering.

The range of programming skills (or lack thereof) has posed as a challenge for the instructor to design and implement course assignments, especially in terms of difficulty level and time for completion. For instance, an assignment that uses lists should be fairly easy for students that have experience working with arrays, but it could be considered difficult for students that are seen the concept of lists for the first time, which would be the case when the concept of arrays is not covered in introduction to programming courses.

Visual vs. Textual Programming

When students are used to write textual source code, the change to a visual programming environment may be sometimes challenging, especially when they cannot see the source code. With App Inventor, a developer creates an application by putting blocks together like a puzzle.

Figure 1 and Figure 2 show examples of a code that handles the event of a button being clicked by the user on the screen. The visual code sample (Figure 1) was created with App Inventor and the textual code sample (Figure 2) was created with Eclipse and Android SDK.

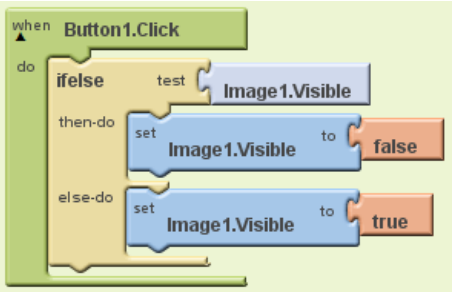

Figure 1: Visual Code for handling the event of a button clicked

```
Button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(Image1.isShown()) {
            Image1.setVisibility(ImageView.INVISIBLE);
        }
        else {
            Image1.setVisibility(ImageView.VISIBLE);
        }
    }
});
```

Figure 2: Textual Code for handling the event of a button clicked

In Figure 1, when the object *Button1* is clicked, the system generates an event called *Click* that checks the property *Visible* of the object *Image1* (i.e., Image.Visible). If the value is true, the system changes the content of the property *Visible* to false, which results on hiding the image. If the value is false, the system changes the property *Visible* to true, which will make the image to show.

Students usually comment that they like the blocks because they don't get stuck reviewing code for missing semi-colon, braces or for misspelled code. In fact, the goal for bringing App Inventor to class was to reduce some of these distractions with coding and to allow students to concentrate more on the functionalities of the application and what can be done with the phone.

Nevertheless, students with more programming experience would state that they knew what they want to do and they probably could write the textual code to get it done but somehow they struggled to put the blocks together. Finding the appropriate blocks to use has also been an issue for students with little programming experience. Every time the course is taught, maybe out of curiosity or frustration, at least one student would demonstrate interest in learning how to create the applications in Java. This would be a great opportunity to introduce the *App Inventor Java Bridge*, a library that allows integrating App Inventor components into apps created in Java and Android SDK (https://code.google.com/p/apptomarket).

Planning and Designing Apps

Programming and application development courses are a great opportunities to introduce and teach software engineering principles to students. After all, mobile apps are software. Planning and designing are often explored in most programming courses. The more programming courses students take, the more they understand the need to carefully plan and design an application before writing any code. New or less experienced programmers have a tendency to skip the planning and designing steps and they would go straight into the implementation.

Although App Inventor has a Designer screen that developers can use to build and view the app screens, it still poses some challenges for students to achieve the desired layout for their apps. More often than not students start the planning and designing of an app by building the application screens directly with App Inventor. When the application involves more elaborated screens, it usually ends with the developer switching to a paper and pencil design approach, or with the developer being stuck building complex screens and leaving the planning of the app behind.

Visual tools such as the Balsamiq Mockups (http://www.balsamiq.com) can help students to quickly design mockup screens for their apps. Designing the screens will force the students to think not only about the components but also about the underlying events, functions and blocks that need to be used to achieve the desired results. It will also help students to decide how to use screen arrangements to create the layouts they want. For example, Figure 3 shows a mockup screen for adding a new item for a Grocery List app (Figure 3, left) and the complete screen implemented with App Inventor (Figure 3, right).

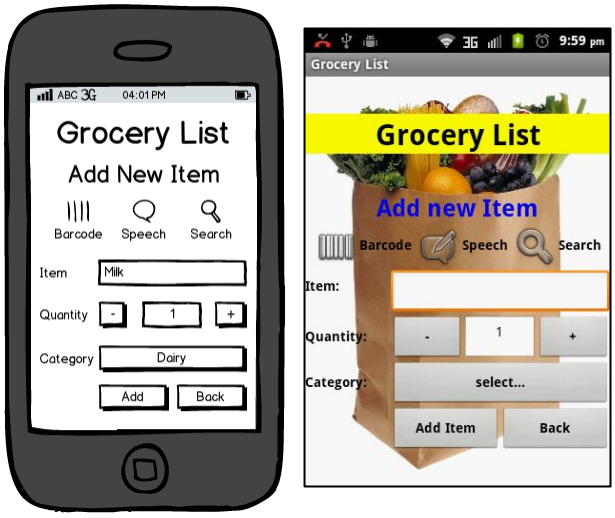

Figure 3: Mockup screens for App design

By generating the mockup screens, students can play with the screens before committing to the implementation of an app. For the Grocery List App, students should identify the need to use buttons to start the *BarcodeScanner*, *SpeechRecoginzer*, and *ActivityStarter* components. To provide a list of existing grocery categories, the developer will need the component *ListPicker* to be populated with all categories. The *ListPicker* component works like a drop-down list where the user can select an item from. Developers should also understand that after the user clicks the button "Add", the

information about the new grocery item should be stored into a Web database, which can be implemented with the component Tiny*WebDB*. The button "Back" would require the developer to hide the current "Add New Item" screen and to show the "Main Menu" screen. Finally, to obtain the layout presented in Figure 3, the developer must understand how the components *VerticalArrangement* and *HorizontalArrangement* can be used to organize the buttons and textboxes on the screen.

Web Services

While students get excited about creating applications to run on their phones, they have also demonstrated great interest in learning how to create apps that can be integrated with other phones and with data from a variety of online sources (ex: weather, sports, maps, etc.).

For some students, working with web services raises concerns with privacy, security, and availability regarding the data and the service. For instance, the component Tiny*WebDB* let us store data into a Web database that is accessible through a web service. App Inventor uses http://appinvtinywebdb.appspot.com as the default service. As a demo service, it stores only 250 entries into the database. Any entries beyond that will force the oldest entries to be deleted. In addition, the tags stored are not protected and can be easily accessed and overwritten. However, developers can create their own services and apply any protection they need, or they can use alternative services.

The website www.programmableweb.com lists thousands of Web APIs that could be incorporated into the apps. Several APIs are free and just require registration to obtain an API Key to access the service. Depending on the type of Web services, the results of the requests may be in different formats, such as XML (Extensible Markup Language) and JSON (JavaScript Object Notation), which will require the students to learn how to parse the messages. The component *Web* in App Inventor has a block called *JsonTextDecode* that transforms JSON text into lists, which makes it easier to manipulate data as App Inventor has several blocks to handle lists. For XML, however, developers need to create the code to parse the XML text. Figure 4 shows a sample of the *wind* information, in XML and JSON formats, from the Yahoo! Weather Forecast.

Another valuable resource is the Yahoo! Query Language (YQL), a SQL-like language that can be used to query data tables from a variety of web services (http://developer.yahoo.com/yql). Some YQL tables are free and can be accessed directly, while other tables require a Yahoo! login or an API Key to access the data. For example, the YQL statement `select * from weather.forecast where woeid = 2379574` would result in information about the weather (ex: temperature, wind speed, etc.) for the `woeid = 2379574` (Chicago, IL). The WOEID (Where On Earth ID) is a unique identifier provided by Yahoo! GeoPlanet.

XML

```
<yweather:wind
xmlns:yweather="http://xml.weather.yahoo.com
/ns/rss/1.0"
chill="66" direction="120" speed="5"/>
```

JSON

```
"wind": {
     "chill": "66",
     "direction": "120",
     "speed": "5"
}
```

Figure 4: Sample XML and JSON results for wind information

New to Event-driven programming

The events in App Inventor that can trigger activities on the phone fall into the following categories (Wolber et al., 2011, p. 223):

- User-initiated event (ex: User clicks a button)
- Initialization event (ex: App starts)
- Timer event (ex: 50 milliseconds passes)
- Animation event (ex: object collide with another object
- External Event (ex: phone receives a phone call)

Figure 5 shows an example of a series of events related to the process of taking a picture with the camera from a phone and showing the picture on the screen. When the object *Button1* is clicked the system generates the event *Click*. Then, the system starts the camera on the phone by calling the procedure *TakePicture*. When the camera application is open, the user can take a picture and click a button (e.g., Ok or Done) to confirm it. This will generate another event called *AfterPicture* that will handle the information about the picture. The procedure *AfterPicture* receives an argument called *image* that is the address of the picture within the

phone (e.g., SD Card). Finally, the picture address available through the parameter *image* is used to set the property *Picture* of the component *Image1*.

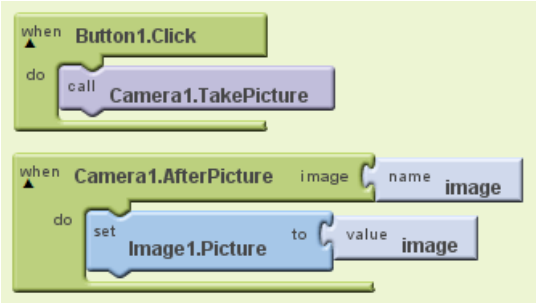

Figure 5: Taking and Showing a picture

Although events can be explored in different programming courses, they are more noticeable when students are learning to program Graphical Unit Interface (GUI) and they have to deal with a variety of graphical components and event listeners. In Java, for example, an *ActionListener* is triggered when a user performs certain actions such as clicking on a button or choosing a menu option. As GUI programming may not be covered in introduction to programming courses, students that are not familiar with these concepts may experience some difficulty identifying all the necessary events for their apps. It might be possible to see students omitting events or trying to handle an event inside another event. For instance, some students might try to set the property *Picture* of the object *Image1* within the event *Button1.Click* (Figure 6), instead of using the event *AfterPicture* as shown in Figure 5. To change the object *Image1* with the picture taken from the camera, the system will need the parameter *image* created by the event *AfterPicture*.

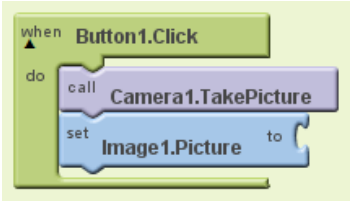

Figure 6: Incorrect event handling

Database and SQL

App Inventor has two main components that can be used to store data. The *TinyDB* stores data to the device's long-term memory, and the *TinyWebDB* stores data to a Web database that is available through a Web service provider. Although these components are relatively easy for students to understand and work with data persistency, they are limited databases. Nonetheless, *TinyWebDB* could be used to access a data source API written in PHP or other languages. More experienced programmers could implement their own services to respond to *TinyWebDB* requests.

An alternative component to be used for data persistence is the *FusiontablesControl*, which is a non-visible component that communicates with Google Fusion Tables (experimental). The component requires an API key to send SQL queries to the server and to receive the query results. The query results are in CSV or JSON formats and can be transformed into lists with the appropriate blocks in App Inventor.

The Fusiontables SQL queries can be used to handle data from tables with INSERT, UPDATE, DELETE and SELECT commands. The use of SQL queries and rapid user interface design can provide a great opportunity for using App Inventor in Database courses. However, students may be required to have prior experience at least with fundamentals of database design and SQL to implement Fusiontables into their apps. In particular, students should learn about the implicit ROWID column, which is the identifier for the row of a table. The ROWID is required to perform INSERT, UPDATE and DELETE statements and can be obtained through a SELECT statement.

Lists and List of Lists

The concepts of Lists, and Lists of Lists are similar to what other computer languages call arrays and multi-dimensional arrays, respectively. These are typically not easy concepts to grasp for a student that is seen it for the first time. Even students that are already familiar with the concepts of arrays may need a period of adjustments to translate and adapt their prior knowledge with arrays into the new environment. The level of programming and the experience with other languages, such as PHP, may be a contributing factor to help with this transition. For instance, students may be familiar with languages that allow using different data types within a list, creating an array without specifying the size prior to using it, or omitting data types for the variables created.

Figure 7 shows a sample of a static List of Lists that is created and populated by the developer. In this example, the list called Employees has two items. Each item is a sub-list with information about an employee (e.g., name and age).

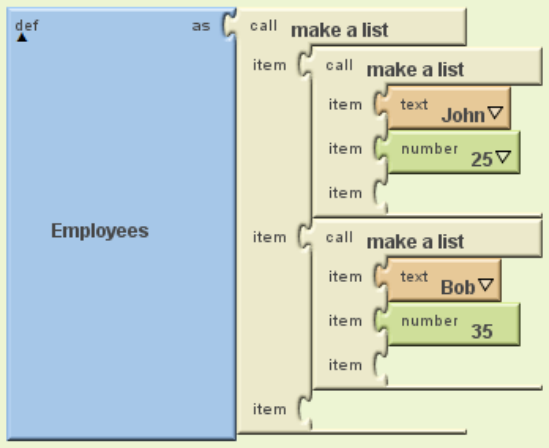

Figure 7: A sample List of Lists

Figure 8 shows the blocks needed to select the second item (35) of the second employee (Bob) from the list of Employees.

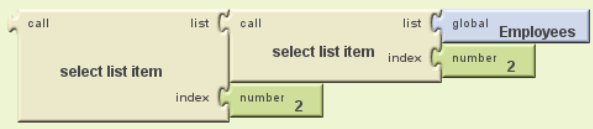

Figure 8: Selecting Bob's age

The visual approach of App Inventor makes it easier for students to understand and create static lists as they can visually see the structure of the list. However, the use of dynamic lists is still intimidating. As lists are used quite often in course assignments, learning or reviewing these concepts during the course would help students to better implement lists to their apps.

User Interface

As applications usually involve some sort of interaction with the user, students are forced to think about the user's experience with the app being created. This is a great opportunity for students to bring their own (good and bad) experiences using mobile applications to design the layout and behavior of their future apps. For example, it is very common to see students discussing about defining the size and color of components to improve readability, notifying the user about whether an operation was successfully completed, or validating users' input to not allow phony data into the system. In addition, students take into consideration usability during the planning and designing of their apps.

Many apps will require multiple screens to organize the application and to help the user to navigate through its different functionalities. Students can create multiple screens by using the button "Add Screen" in the App Inventor environment or by creating screen arrangements to act as screens. The arrangements can be hidden or displayed to create the illusion of working with multiple screens.

While the first option would be preferred, it still has restrictions, regarding the definition of variables and procedures, that seem to influence the students' decision to adopt it. As each screen has its own components and blocks editor, the components, variables and procedures created for one screen are not available to other screens. For example, if a procedure responsible to perform some calculations in Screen1 is needed inside Screen2, the developer will need to re-create the procedure inside the Screen2 as the first procedure cannot be accessed from another screen. In addition, if the blocks of a screen need to change a component in another screen, the developer will need to pass parameters between the screens or use *TinyDB* to store the data to be used by the other screens. Copying blocks between screens is not yet supported by the current version of App Inventor. On a positive note, the use of multiple screens (not arrangements) will force the students to carefully plan their apps and how the screens will communicate with each other.

Data communication

A class project that used Bluetooth to create a simple game of tic-tac-toe, helped with a discussion on the use of special messages, protocols to communicate with paired phones, and the roles of client and server. The discussion also helped students to use Bluetooth communication with codified messages to create other games (e.g., Checkers).

Figure 9 shows an example of exchanging messages with the Bluetooth Chat app. These messages can be the start point of a discussion on the content and format of the messages as well as how to exchange more elaborated messages.

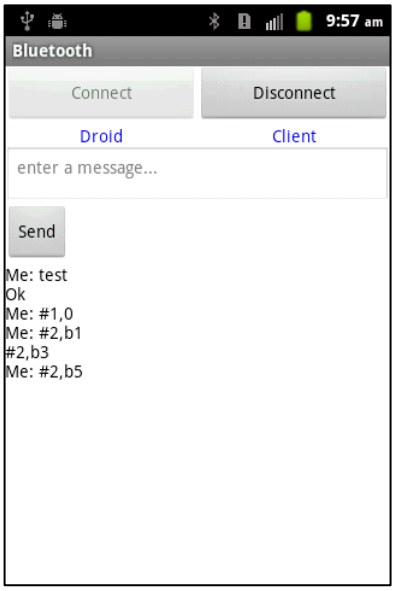

Figure 9: Bluetooth Chat, From text to codified messages

In a Bluetooth Tic-Tac-Toe game, for example, a message `#1,O` could be used to inform that a player is inviting another player to play. The capital `O` means that the user chooses to play with the symbol `O` instead of `X`. When the symbols representing the players are selected, both systems set the variables *Me* and *You* with the respective symbols. These variables can be used to display the symbols above the game board. The role of the Bluetooth connection (i.e., client or server) can be used to define the symbols. The game has a setup screen where the user can select his or her preferred symbol. However, the player with the role of a Server will have priority on the symbol selection.

A message `#2,b1` could be used to inform that the player has clicked on one of the buttons of the game board. Other codified messages can be added to improve the user's experience with the app. For example, when a match is over, the system could ask if the player wants to play again and send a message with the user's response to the other player (e.g., `#3,Y`). If the user decides to disconnect, the system could automatically send a message `#4,end` when the button *Disconnect* is clicked, which would inform the other player they are no longer playing the game.

With that in mind, students can create a Tic-Tac-Toe board by adding new components to the existing Bluetooth Chat layout, and using the messages received to interact with the game and change the board accordingly (Figure 10).

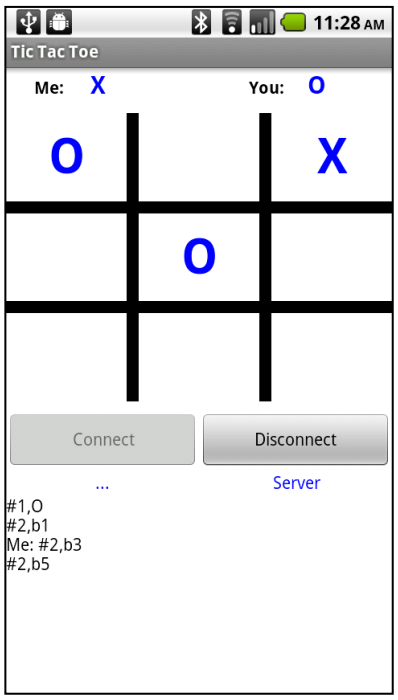

Figure 10: Tic-Tac-Toe with Bluetooth

The game board is composed of 9 buttons (*b1* to *b9*) that are organized with a table screen arrangement to provide the expected layout. The first row has buttons *b1* (top left corner), followed by the button *b2* (top middle) and button *b3* (top right corner). The middle row has buttons *b4*, *b5* and *b6* (from left to right). The bottom row has buttons *b7*, *b8* and *b9* (from left to right). The layout also has several labels that are used to design the vertical and horizontal lines of the board. To get the effect of a thick line, the student can remove the text of a label and set the background color to black.

In this example, receiving the message `#2,b1` would force the system to set the text of button *b1* (top-left button) with the letter O, which is the symbol of the other player. Similarly, if the player clicks the button *b3*, the system will set the text of button *b3* with `X` (i.e., the player's symbol), and will send a message `#2,b3` to the other player so that his or her phone can update the screen with the new game move.

Using versus Creating objects

Students that are used to object-oriented programming may find an issue related to using versus creating objects. For example, it is

possible to create a new instance of a button by dragging it to the screen. After that, all the properties and pre-defined procedures for a button component are available (see Figure 11). However, it is not yet possible for developers to create their own classes of objects with attributes and procedures, such as, a class of enemies for a game.

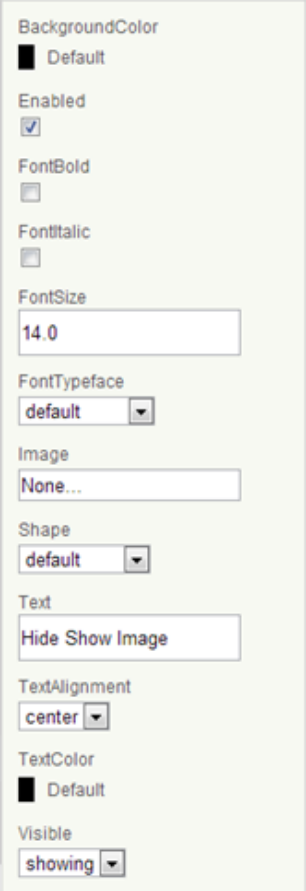

Figure 11: Properties of a component Button

While writing code using only general procedures doesn't seem a concern to complete the assignments, keeping the students' mind from thinking in terms of object-oriented programming may be challenging, especially when they want to organize, reuse and protect their code.

4. CONCLUSION

This paper discussed issues, challenges and opportunities that instructors should take into consideration when designing their courses with App Inventor:

- Pre-Requisite for the course
- Visual vs. Textual Programming
- Planning and Designing Apps
- The use of Web Services
- Event-driven programming
- The use of database and SQL
- Lists
- Designing User Interfaces
- Discussing Data communications
- Using versus Creating objects

Even though App Inventor has been used and advertised as a tool for teaching basic programming skills, it has great potential to be used for teaching students that already have programming experience. Despite programming not being required, some App Inventor components require a great deal of computing skills such as SQL and database design.

App Inventor has the potential to be included in the curriculum of other courses where students could take a basic course on App Development early in their curriculum and then more advanced courses would use the tool to explore the concepts and topics to be covered in class. For instance, in a Software Engineering course, students could use the tool to help with requirements and interface design. For more advanced programming courses, students could use App Inventor Java Bridge to write code and integrate it with App Inventor, which could help to overcome the limitation of programming in an object-oriented style and working collaboratively to create applications.

In sum, App Inventor can be explored beyond introductory programming courses for novices. However, instructors should be aware of potential issues and challenges related to the required pre-requisite for the course and the students' prior experience with programming and other computing skills. In terms of opportunities, instructors could explore the use of App Inventor to cover a variety of computing concepts such as:

- Application development life cycle
- Web Services and Distributed computing
- Information Assurance and Security
- Software Engineering
- Data communication
- Database Design

5. ACKNOWLEDGMENTS

The author is grateful to State Farm® for supporting this project through the *2011 State*

Farm Technology Grant, which was partially used to acquire Android devices for class instructions.

6. REFERENCES

Arachchilage, N. A. G., Love, S., & Scott, M. (2012). Designing a Mobile Game to Teach Conceptual Knowledge of Avoiding "Phishing Attacks". *International Journal for e-Learning Security, 2*(2), 127-132.

Gestwicki, P., & Ahmad, K. (2011). App inventor for Android with studio-based learning. *J. Comput. Sci. Coll., 27*(1), 55-63.

Karakus, M., Uludag, S., Guler, E., Turner, S. W., & Ugur, A. (2012, 21-23 June 2012). *Teaching computing and programming fundamentals via App Inventor for Android.* Paper presented at the Information Technology Based Higher Education and Training (ITHET), 2012 International Conference on.

Lim, B. L., Hosack, B., & Vogt, P. (2010). *A web service-oriented approach to teaching CS/IS1*. Paper presented at the Proceedings of the 41st ACM technical symposium on Computer science education, Milwaukee, Wisconsin, USA.

Tyler, J. (2011). *App Inventor for Android: Build Your Own Apps - No Experience Required!* : Wiley Publishing.

Uludag, S., Karakus, M., & Turner, S. W. (2011). *Implementing IT0/CS0 with scratch, app inventor forandroid, and lego mindstorms*. Paper presented at the Proceedings of the 2011 conference on Information technology education, West Point, New York, USA.

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor: Create Your Own Android Apps*: O'Reilly Media.

Appendix: Resources for Instructors

1. App Inventor: http://appinventor.mit.edu
2. App Inventor 2 (Alpha): http://ai2.appinventor.mit.edu
3. App Inventor Community Gallery: http://gallery.appinventor.mit.edu
4. App Inventor: Create Your Own Android Apps (book): http://www.appinventor.org/projects
5. App Inventor Java Bridge: https://code.google.com/p/apptomarket
6. App Inventor TinyWebDB (default service): http://appinvtinywebdb.appspot.com
7. Balsamiq Mockups: http://balsamiq.com/products/mockups
8. Google App Engine: https://developers.google.com/appengine
9. Google Fusion Tables API: https://developers.google.com/fusiontables
10. Professor David Wolber's Course-in-a-Box: http://www.appinventor.org/course-in-a-box
11. ProgrammableWeb APIs: http://www.programmableweb.com
12. Yahoo! Query Language (YQL): http://developer.yahoo.com/yql