Cabana: A Cross-platform Mobile Development System

Paul E. Dickson
Hampshire College
School of Cognitive Science
893 West St.
Amherst, MA 01002, USA
pdickson@hampshire.edu

ABSTRACT

Mobile application development is a hot topic in computer science education, and debate rages over which platform to develop on and what software to use for development. Cabana is a web-based application designed to enable development on multiple mobile platforms and to make application development easier. It uses an approach to application programming based on a wiring diagram that is supplemented with the ability to program directly using JavaScript. It is an ideal choice for application development in introductory computer science courses and for upper-level courses where the focus is on application design and not application programming. This paper introduces Cabana and describes its use in two different computer science courses.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science EducationComputer Science Education; D.1.7 [**Programming Techniques**]: Visual Programming

General Terms

Design, Languages

Keywords

Cabana, iPhone, Android, AppInventor, Xcode, Mobile Devices, apps, Smartphone

1. INTRODUCTION

Mobile device application development is a hot topic and to judge by the number of papers published about mobile development [1, 2, 3, 4, 5, 6, 7, 9, 11], a growing part of computer science education. Students are excited about and are asking for courses in it and faculty are scrambling to build those courses. Departments see mobile application development as a way to attract new students to computer science while providing a fertile environment for teaching fundamental computer science topics like memory management, networking, human-computer interaction, etc. The question is usually not whether a course should be offered in a computer science department but whether mobile application development should be integrated into existing courses or be a stand-alone course.

Many questions remain to be answered, one of the biggest being which platform to develop on. Apple has a free university program but this requires the school to own Apple hardware and sign a licensing agreements. Android is free to develop on but does not have the same application market as Apple [8]. Blackberry and other platforms exist but they have the same problem as iOS and Android: once you decide to develop for one platform you are locked into it unless you want to learn a second environment. Another choice would be to develop web applications using HTML5, which would run on all platforms, but in that case students would be taking a web design course and not a mobile applications course. In this paper we discuss Cabana, a piece of software for developing mobile applications that currently creates mobile web applications that can run on any smartphone browser and that will soon be able to create native applications for both iPhone and Android.

Cabana was developed by Department of Behavior and Logic, Inc. (DOBL) and is currently in beta testing. It is available at https://www.cabanaapp.com/. We used it in classes at Hampshire College for an academic year as it progressed from alpha to beta testing. Cabana is designed to make app development easier and uses an approach based on a wiring diagram for linking different code modules together. Cabana comes with a set of code modules that give basic programming functionality (if-then statements, for loops, counters, etc.) but also allows the user to create modules where the functionality is defined using JavaScript.

Our experience with Cabana has been that it is far easier to use in a classroom environment than is Apple's Xcode and that it makes mobile application development accessible to even a CS0 audience. We have found that it makes it possible to build a course that focuses on mobile application development and not on learning to program in a specific SDK. Cabana is a web-based application and therefore is cross-platform and it includes a simulator. It requires no hardware to be acquired since the web applications can be tested on any mobile device with a browser, something many students and faculty already own.

In this paper we describe how Cabana works and provide examples of the development environment. We also discuss how Cabana functioned in the classroom, both its strengths and weaknesses. We make the case that Cabana should be considered as an alternative method for developing mobile applications and not that it is the only solution.

2. RELATED WORK

A lot of ground work has been done with regard to development of courses that include mobile application development. This work has not led to any type of a consensus though because of factors of environment and the goals of the courses. A snapshot of the approaches to mobile application development and the strengths and weaknesses of each gives a good background as to why it is so hard to determine the best platform and environment to use.

Rogers [9] and Ivanov [4] did a great job of describing what is required to teach an iPhone development course. They describe the process of becoming a part of the Apple University program and what is required in terms of hardware and software. Ivanov in particular describes some of the issues of working with a developing platform and describes the problems created when Apple updated Xcode and iOS part way through the semester, changing Xcode functionality and portions of the SDK requiring revisions of the course plan. Some of the difficulties in learning the iOS SDK and Xcode environment including the need to learn Objective-C and Cocoa Touch are also discussed.

The question of whether to work on the Android platform instead of Apple's iOS has also been covered [2, 3]. Whereas Goadrich et al. [3] simply describe the benefits of one platform versus another, Fenwick et al. [2] encourage movement away from iOS because of how locked a platform it is. Both make the argument that a platform choice should be influenced by what hardware is present on campus. Good points are made about cost of development on given platforms, programming language restrictions, software requirements, etc. Fenwick et al. make the additional comparison to mobile web applications and describe the attractiveness of their being platform independent while mentioning that this does make it harder, in some cases impossible, to access on device hardware like GPS, sound, camera, etc. This is not to say that iOS and Android are the only platforms, as Mahmoud et al. [7] have done great work building a toolkit to make teaching mobile application development on the Blackberry easier and have shown how to integrate Blackberry development into a large number of courses.

Substantial work has also gone into making mobile application development easier. Wolber [11] describes Google's App Inventor and how it can be used to teach mobile application development and focuses on how easy and accessible App Inventor makes programming applications for Android. Great detail is given to the block programming method used that does not require/allow the user to write code. Using App Inventor for introductory courses has become sufficiently mainstream to have been discussed at SIGCSE in 2010 [10].

Others have focused primarily on how application development can be used to motivate students in a standard computer science curriculum [5, 6]. Kurkovsky [5] talks about using mobile game development to motivate computer science topics. He emphasizes how the mobile devices get students excited and give them a platform that helps them to connect concepts like computer networking, database management, etc. to the real world. Loveland [6] talks about integrating Android development into a human-computer interaction course. She talks about doing web development using the Google Web Toolkit to develop applications and then running those applications on an Android simulator.

3. BACKGROUND

Our initial decision to do application development in the classroom was in Fall 2009 when we decided to use iOS as the platform for a video game development course [1]. We were led to this decision to use the iPhone as a platform for a variety of reasons, the most important being that it would excite our students and limit the scope of games they could create and because we are primarily an Apple-based campus. Our approach to preparing the course was similar to that of Ivanov and Rogers [4, 9].

While students liked the course and felt like they learned a lot, they and the faculty found that the iOS platform got in the way of teaching video game design. With Apple updating Xcode 3 times during the semester, the class became experts in installing the latest 5GB download. Also, references and tutorials were often out of date within a month of being written, making it a constant scramble to try to keep course material up to date. Concepts as simple as flipping between different screens could take upwards of an hour to describe and demonstrate in class because of the amount of back end coding and SDK knowledge required. This meant that the course ended up being a 50% iOS and 50% game design course. That is not to say that students do not gain a lot from exposure to a real-world environment where platforms can change rapidly but instead to point out that the this was not the education in video game design we were hoping to give to our students in this course.

With this in mind, we started looking for a new development platform for our video game design course and for the mobile application course that we wanted to develop that would focus on design and human-computer interaction and not on programming. We did not seriously consider moving to Android because our students are primarily Apple based. In the three courses we have now taught that used mobile devices, we have averaged 1-2 students per course who have had an Android compared with the 85% who own iPhones or iTouches. We did not want to shift to a web-based application development platform that would be as much for web development as mobile development because we felt we would lose student interest if we did not have a tight mobile focus.

This led to our collaboration with DOBL, a company cofounded by one of our alumni. DOBL was developing an application that was intended to make it easier to build mobile applications and that would enable users to develop for multiple mobile platforms. They did not intend to have their applications compile cross-platform but instead to be module based so when you were developing for iOS you would include the iOS touch module whereas when developing for Android you would include the Android touch module. This seemed to be the best option for us since we hoped it would give us the ease of programming found with App Inventor while still enabling us to interact with the Apple devices our students primarily have.

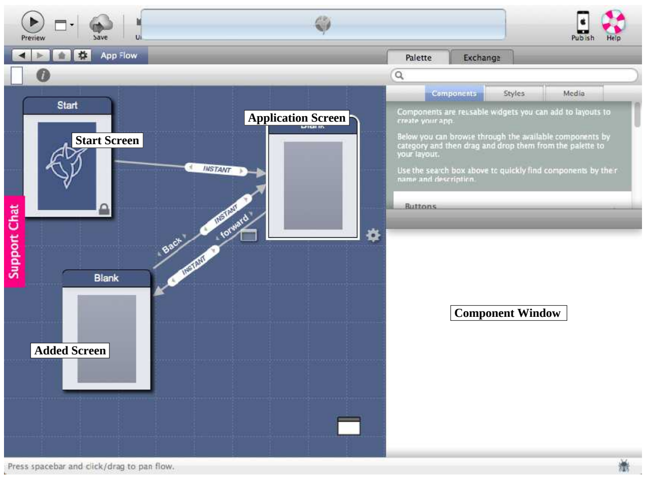
4. CABANA

By the time we started working with Cabana it had become a browser-based application. We alpha tested it in our first class that used it and early beta tested it in our second. For much of this time, our students were the only users working with Cabana. The specific details of our classroom experiences using Cabana are described after a description of how Cabana works.

The best way of showing the Cabana environment is to build a simple application in Cabana. We will do this in two parts; first we will build an application that flips between screens by pressing buttons; next we will add a JavaScript node that implements the Pythagorean theorem. The first application was chosen to demonstrate Cabana since the simple flipping between screens took an hour to teach during class using Xcode and iOS but only about 5 minutes in Cabana. The applications are kept to the minimum with no effort put into styling or formatting in order to keep them simple.

4.1 Screen-flipping Application

When starting an application in Cabana, you are asked to choose between a multi-screen application and a single-screen application and whether you want it to be landscape or portrait. We chose a multi-screen portrait application and the software opened a view similar to that seen in Figure 1. The left side of the screen gives an overview of the application and the right side contains information and components that can be added to the interface on the left. All Cabana applications begin with a start screen and then shift to the application screen. We added a second screen and drew lines between the screens that control movement between them and labeled Forward and Back. The names represent actions that can be called from the code so that when the action forward is called in the application screen, the application will shift to the added screen.



Components are reusable widgets you can add to layouts to create your app.

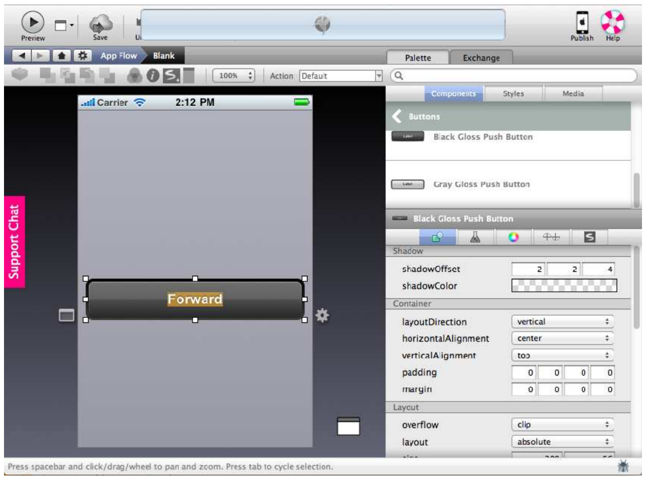
Below you can browse through the available components by category and then drag and drop them from the palette to your layout.

Use the search box above to quickly find components by their name and description.

Figure 1: Overall application flow.

The user then interacts with a specific screen by double clicking on it to get an enlargement (Figure 2). A list of components on the right gives the user options of different objects that can be added to the screen like the button we have added. All components are dragged from the component list to the desired location on the screen. Below the list of components appears a list of options that can be set for a given component that has been added to the screen.

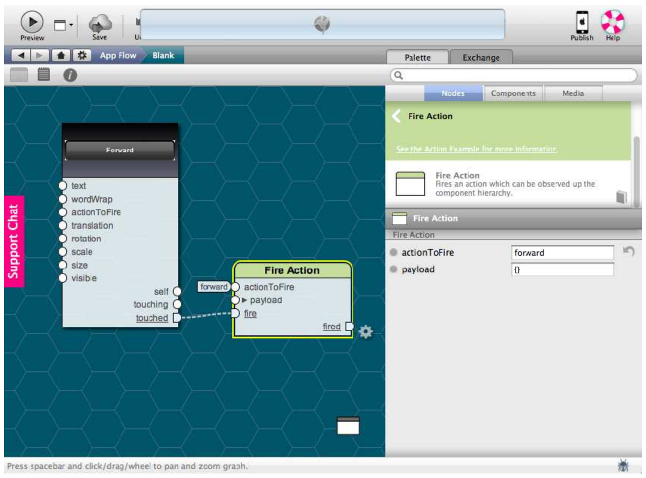
Since we want to use the button to jump from the application screen to the added screen, we need to link it to the



Shadow
shadowOffset 2 2 4
shadowColor
Container
layoutDirection vertical
horizontalAlignment center
verticalAlignment top
padding 0 0 0 0
margin 0 0 0 0
Layout
overflow clip
layout absolute

Figure 2: Screen layout with Cabana tools.

code that controls the application. We do this by clicking on the rectangular symbol at the lower left corner of the item (the button in Figure 2) we wish to punch through to our wiring diagram (Figure 3). All Cabana logic and control is handled by wiring diagrams. Again, the components (nodes) are available on the right side of the screen and the wiring is shown on the left. The large box is the wiring for the button and the smaller box is a node that we have added that fires an action, in this case calling forward to send us to the added screen. This is accomplished by connecting the touched signal on the button box to the fire signal on the action box. We specify that the action to run is forward by connecting forward to the action to fire input on the action box. This is all that is required to create an application that shifts from one screen to another with a button push. The application can be previewed or published with a push of a button. Demonstrating this process takes about 5 minutes versus the approximately 60 minutes needed for Xcode. Some of the reasons this takes so long in Xcode is that new files need to be created for the new screen, code needs to be written that relies on a solid knowledge of the iOS SDK, and the code needs to be linked into the interface builder.



Fire Action
Fires an action which can be observed up the component hierarchy.

Fire Action
actionToFire forward
payload {}

Figure 3: Wiring diagram with Cabana tools.

4.2 Pythagorean Theorem Application

The example of building an application to run the Pythagorean theorem is given to show how JavaScript nodes function. The basic layout of the application was created by dragging and dropping the desired fields onto the screen (Figure 4). Each component is punched through to the wiring diagram (Figure 5). The text written into each text field and the signal from the buttons touched are linked to the sampleNode, which is a custom JavaScript node, and its result is connected to the display text area. The inputs and outputs of the JavaScript node are specified in the JavaScript node (Figure 6).

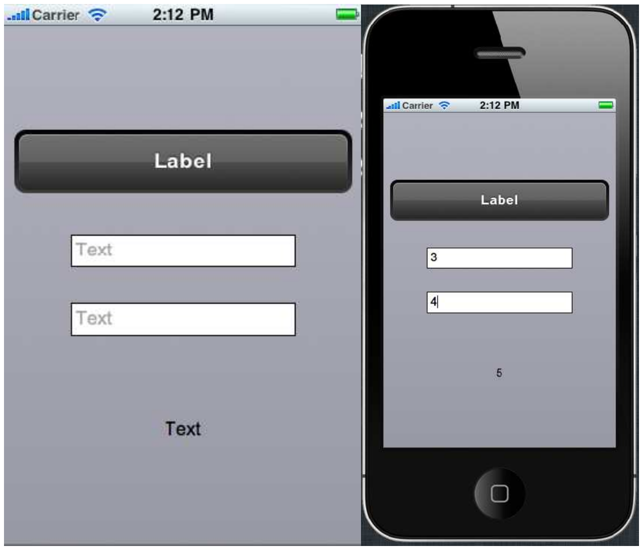


Figure 4: Enlarged screen layout and application preview.

By double clicking the JavaScript node, users enter an editing node that enables them to label the node and write JavaScript to specify how that node will work. Figure 6 shows an enlargement of this editing window with the top being where the control is specified while the bottom specifies the input and output types. Inputs and outputs are defined as either signals or data where the data is specified by data type. The function of the node is defined using standard JavaScript; sample output of this application can be found in Figure 4.

5. COURSES

The decision to use Cabana in our video game development course was made in Spring 2010 when DOBL assured us that it would be ready by the start of Fall 2010. We had a student intern with DOBL over the summer of 2010 to ensure that we would have an on-site expert for the system. As a result of our experiences using Cabana in the video game design course, we next decided to use it for our inaugural mobile application course.

One point to consider is the makeup of computer science courses at Hampshire College. We do not have set majors so there is no way to guarantee student background for our classes. Even though these courses are upper level, the only restriction placed upon students is that they must have had at least one programming course. This means that some

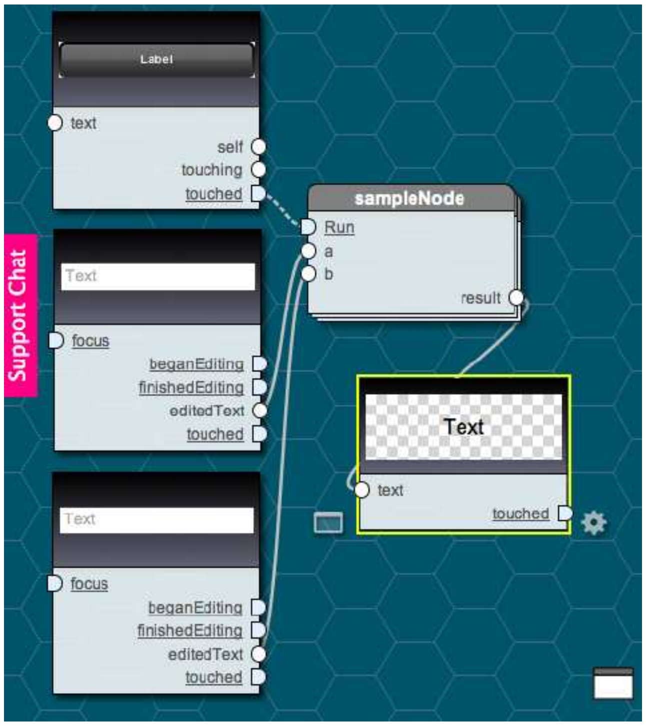


Figure 5: Enlarged wiring diagram with JavaScript node.

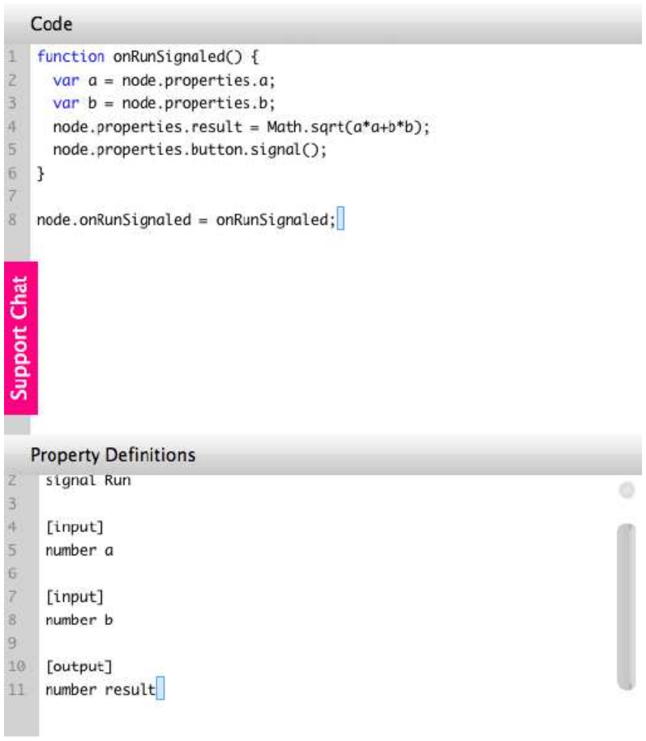
of the students in these courses have only a high school programming background while others have nearly complete computer science majors. This has a large effect on the work and expectations for our classes.

5.1 Video Game Development

We began the semester with an early alpha version of Cabana and with only our student who had interned at DOBL having a good handle on how Cabana worked. Cabana being in early alpha testing had a large number of bugs and large parts of it were rewritten during the semester. Whenever bugs were discovered, DOBL quickly worked to fix them for us.

Cabana was not a complete success for this game development course. Cabana is intended for making applications and not games and so support for high-level graphics, quick responses, character spawners, etc. did not exist. Also, large-scale revisions of Cabana often broke code that had been written by students weeks earlier. These difficulties were not that different from those experienced with iOS and Xcode, which are also under constant refinement. Students did seem to pick up Cabana far faster than they had iOS and Xcode the previous semester and this enabled us to focus the course more fully on video games.

Despite the setbacks, Cabana was not a complete failure. At the end of the video game design course that used Xcode, only 2 of 5 student groups within the class had games that could be played on the iPhone. At the end of the semester using Cabana, 4 out of 5 groups had working games. While many of the students complained about the constant updates and bugs within Cabana, all who had done equivalent work with iOS found that Cabana, although frustrating, was enormously easier to use than Xcode.



```
function onRunSignaled() {
  var a = node.properties.a;
  var b = node.properties.b;
  node.properties.result = Math.sqrt(a*a+b*b);
  node.properties.button.signal();
}

node.onRunSignaled = onRunSignaled;
```

```
 signal Run

 [input]
 number a

 [input]
 number b

 [output]
 number result
```

Figure 6: JavaScript node code.

5.2 Mobile Application Development

Based on our experience with Cabana in the game development course, we decided to use it for our mobile application course. The fact that Cabana was moving into beta testing and that mobile applications are its intended market suggested that we would have greater success. The focus of the mobile application course was on the development process and human-computer interaction and not on the details of programming mobile applications, though students would be expected to do this. Roughly one-third of the students in the mobile application course had taken the video game design course the semester before and had experience with Cabana.

Cabana turned out to be a huge success for this course. Students found it easy to work and develop with Cabana. The wiring-based logic was a little complicated for students to pick up at first but soon became second nature. Our students were able to rapidly build application and test them. With the ease of building simple applications, our students were able to build Tamagotchis in under a week. This made it possible for the focus to be on high-level design and not debugging. Even our novice programmers were able to build mobile applications rapidly.

The entire process of using Cabana was not completely smooth. Bugs still existed that caused failures with correctly written code. Also, a couple of changes were made to Cabana even though it was in beta, which caused previously working applications to break part way through the semester.

This course was only made possible by Cabana. Without the ability to rapidly create a prototype that Cabana provides, the course would never have been able to focus on high-level concepts. Equally, without the ease of use, quite a few students would have been unable to complete their final projects: all but one student had a successful final project. The final projects show the ability to polish applications with Cabana (Figure 7).

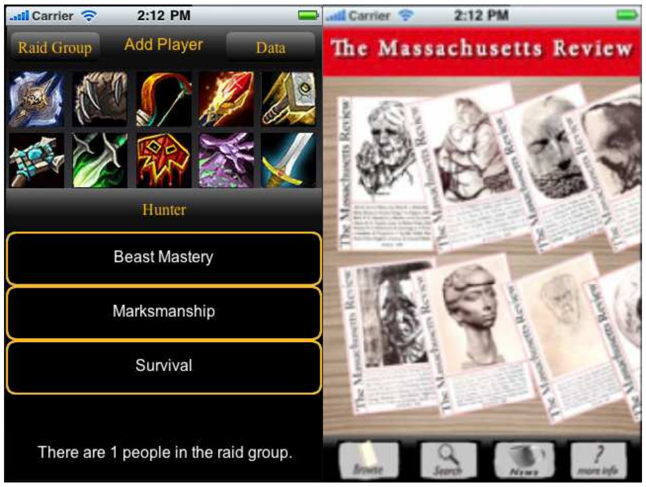


Figure 7: Final project mobile application samples.

6. DISCUSSION

Cabana is a program for developing mobile applications and the best way to see where Cabana fits into application development is to compare it with the major alternatives. As seen in Table 1, Cabana is the only development environment for mobile applications that is capable of creating web-based mobile applications and hence cross-platform applications. While other methods of creating mobile web applications exist (for example, developing with Google Web Toolkit), Cabana is the only one that we know of that is only oriented toward creating mobile applications. As of right now Cabana does not enable a user to create native applications but that will be changing soon. Like App Inventor, Cabana is easy to learn and to use and does not require the user to learn a large amount of the device SDK. The downside to this is that Cabana, like App Inventor, does not give the developer full control over the device.

	Web	Native	Device	Easy	Control
Cabana	Yes	Soon	Yes	Yes	Partial
iOS	No	Yes	Yes	No	Yes
Android	No	Yes	Yes	No	Yes
App Inventor	No	Yes	Yes	Yes	Partial
Other Native	No	Yes	Yes	No	Yes

Table 1: Comparison of mobile application developing environments. Web: create web-based applications, Native: create native applications, Device: have access to device hardware, Easy: easy to get started for a novice, Control: provides programmer with full control over device though SDK, Other Native: developing native applications for another platform.

All of this suggests that the focus of the project needs to be considered when deciding whether to use Cabana. If the goal is to look into data transfer between devices or processor management within the scope of a limited device, then choosing to develop for iOS, Android, or another platform using its native environment makes the most sense. Equally, if the goal is to expose students to a new platform and what it is like to develop in a real-world environment with its limitation, then native application development again makes the most sense.

Cabana eliminates a lot of the low-level work, such as provisioning devices, that is necessary for native development and makes it easy for beginners to begin developing mobile applications. Like App Inventor, Cabana provides an environment that novices can quickly grasp but unlike App Inventor it does not limit development to one platform. Cabana's cross-platform nature also makes it ideal for environments with more than a single mobile platform and may enable a school to avoid purchasing a class set of any particular mobile device.

Cabana is ideal for courses where the focus in on interface design because rapid prototyping enables students to quickly build, test, and revise designs. It provides an interface that easily enables users to build attractive interfaces but still gives the user the power to build complicated programs.

The combination of wiring diagram and JavaScript node creation for application functionality makes it accessible to all levels of programmer. For the novice, nodes exist that perform functions like addition and subtraction and others that provide logic like if statements. Experienced programmers who prefer to write code can simply write JavaScript nodes that include all of their logic and avoid having to try to use the nodes. This gives Cabana an advantage over App Inventor since you are not restricted to a novice-centered programming environment like App Inventor's block language and an advantage over native environment programming since programming background is not required.

Cabana is not perfect and is at the time of this publication still not completely established. Some functionality is still changing though this does not differ greatly from our experience developing under iOS. Also, although wiring nodes together to build programs provides an easy entry into logic and algorithm design for some students, others, including some students who can program easily, never fully grasp it.

One final point for consideration is cost. Developing for Android be it directly or using App Inventor is free. Apple has an educational program that allows schools to develop for iOS for free. DOBL has not yet finalized a pricing model for Cabana but plans to have both a free plan and an educational discount. The free plan will likely limit the number of projects that can be worked on and application publication. The educational discount is likely to be the free plan with fewer restrictions and also free.

7. CONCLUSIONS

A lot of choices have to be made when deciding what approach to take for developing mobile applications. Cabana provides a cross-platform environment for developing cross-platform mobile applications. It is accessible to novice students and powerful enough to enable experienced students to create impressive applications. It does not provide students with the level of control they would have developing in a native environment but does provide enough control for most applications. We found it to be extremely useful in the classroom and recommend its use both in introductory courses and courses focused on high-level design.

8. ACKNOWLEDGMENTS

Thanks to DOBL for providing Cabana for use in our classes and for providing consistent timely development of features as we needed them.

9. REFERENCES

[1] Paul E. Dickson. Experiences building a college video game design course. *J. Comput. Small Coll.*, 25(6):104–110, 2010.

[2] James B. Fenwick, Jr., Barry L. Kurtz, and Joel Hollingsworth. Teaching mobile computing and developing software to support computer science education. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 589–594, New York, NY, USA, 2011. ACM.

[3] Mark H. Goadrich and Michael P. Rogers. Smart smartphone development: ios versus android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 607–612, New York, NY, USA, 2011. ACM.

[4] Lubomir Ivanov. The i-phone/i-pad course: a small college perspective. *J. Comput. Sci. Coll.*, 26:142–148, June 2011.

[5] Stan Kurkovsky. Engaging students through mobile game development. *SIGCSE Bull.*, 41:44–48, March 2009.

[6] Susan Loveland. Human computer interaction that reaches beyond desktop applications. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 595–600, New York, NY, USA, 2011. ACM.

[7] Qusay H. Mahmoud, Thanh Ngo, Razieh Niazi, Pawel Popowicz, Robert Sydoryshyn, Matthew Wilks, and Dave Dietz. An academic kit for integrating mobile devices into the cs curriculum. *SIGCSE Bull.*, 41:40–44, July 2009.

[8] Elinor Mills. Report: Android app market growing faster than iphone apps. http://reviews.cnet.com/8301-13970_7-20032228-78.html, Oct 2011.

[9] Michael P. Rogers. Wrong number: avoiding the hidden perils in iphone development. *J. Comput. Small Coll.*, 25:300–305, May 2010.

[10] Ellen Spertus, Mark L. Chang, Paul Gestwicki, and David Wolber. Novel approaches to cs 0 with app inventor for android. In *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, pages 325–326, New York, NY, USA, 2010. ACM.

[11] David Wolber. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 601–606, New York, NY, USA, 2011. ACM.