App Inventor and Real-World Motivation

David Wolber
Department of Computer Science
University of San Francisco
2130 Fulton Street
San Francisco, CA 94117
wolber@usfca.edu

ABSTRACT
App Inventor is a visual “blocks” language for creating mobile apps. As part of a Google pilot program, App Inventor was taught to university students in a core curriculum course at the University of San Francisco. This paper introduces App Inventor and the course, focusing on how the language facilitated interactions with the world outside of the classroom.

Categories and Subject Descriptors

General Terms: Design, Languages

Keywords
Visual Programming, Blocks Programming, Motivation.

1. INTRODUCTION
The number of computer science majors increased by 14% over the last two years, but enrollments are still nearly 50% below the count of early 2000 [6]. With CS-related fields expected to grow rapidly [4], there is a need for more computer science courses that engage students both in K-12 and the university level [16].

In the Summer of 2009 Google launched a pilot program with ten universities. They gave each university 20 Android phones and accounts to a yet-to-be-released tool, App Inventor for Android. The goal was to explore the use of this new visual language for building mobile apps. Here’s an except from the pilot announcement:

“open mobile platforms like Android can bring some of that same change to introductory Computer Science, to make it more about people and their interactions with others and with the world around them. It’s a vision where young people—and everyone—can engage the world of mobile services and applications as creators, not just consumers. Through this work, we hope to do the following:

* Make mobile application development accessible to anyone.
* Enhance introductory learning experiences in computing through the vehicle of Android’s open platform.
* Encourage a community of faculty and students to share material and ideas for teaching and exploring.” [1]

I had been teaching a general education course, “Computing, Robots, and the Web” for a few years, using the Lego Mindstorms system [9] and Media computing [8] for the programming component. Programming robots and multi-media motivated student to learn some programming and problem solving, but I felt that most had trouble seeing a direct application to their lives.

Before attending the Google pilot, I was considering Alice [5] and Scratch[11,13], both of which motivate programming through the storytelling metaphor and letting students create multi-media animations in a visual environment. But App Inventor and mobile apps intrigued me because students could create personalized software for their most precious device, the phone, and literally augment their own reality. App Inventor provides high-level components for process incoming SMS texts, interfacing with the GPS location sensor of the phone, scanning barcodes, and communicating with web APIs. The mobile world was exploding and here was this visual language the let beginning students be creative in that world.

I also knew that App Inventor used a similar blocks language to that of Scratch, which had proven successful not only with kids but as an initial module in a university course [11]. Though App Inventor was only partially complete at the initial summer workshop, Hal Abelson, the co-director of the project, stood and said, “Trust us, it’s Google.” I took Hal’s advice and re-designed my course. I figured my students wouldn’t be too displeased if the “Robot” in my course title was an Android phone!

My leap of faith was rewarded: teaching App Inventor has been my most satisfying teaching experience in seventeen years. The apps created by my students have been incredible, especially given that they had no prior programming experience and most took the class to fulfill the dreaded math core requirement.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE’11, March 9–12, 2011, Dallas, Texas, USA.
Copyright 2011 ACM 978-1-4503-0500-6/11/03...$10.00.
The stories exemplify an important aspect of App Inventor: in a manner of weeks, beginners can build apps that are not only fun, but have real-world utility. It allows creative people to transform their ideas into working, interactive apps that can be taken up by large companies, used by non-profit organizations, and turned into startups. In my experience, such real-world impact for student projects is rare, even for upper classmen.

The students were surprised and empowered by what they created. Here are a few excerpts from an end-of-course survey:

“This course helped me see how powerful modern technology really is. Before this class, I never in a million years thought that creating apps for phones was so accessible and simple to create/test.”

“I can actually participate in conversations about Computer Science when before I used to check out and not listen because I figured it was just something I didn't understand.”

“I thought we were going to be in a dark room working with these creepy monitors and everything would be code text but it is not. Its hands on, fun and a good challenge.”

App Inventor is now publicly available in Google Labs and has matured significantly from the initial pilot phase. Having observed the excitement of my students, I was not surprised at the large user community that quickly materialized [3] or the fact that the New York Times [10] and others saw its potential.

I have now based my course on App Inventor for three semesters and have developed course materials [18], tutorials for the App Inventor site [17] and a video breakdown series [19]. In this paper, I’ll introduce the environment and how I use it in my class, then share the three examples of student projects with a real-world impact.

2. “Hello World” in Three Languages
A student’s initial experience with programming should be both comprehensible and motivating. With Java, the prototypical first example is Hello World:

```java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("HelloWorld!");
    }
}
```

Because of its object-oriented nature, even this simplest Java example has a lot of packaging. Just about every term used in the sample is indecipherable to a beginning student. Beginners are not developmentally ready to learn any of it, and the teacher must wave his hand and move on.

Besides being hard to understand, the Java initial example is hardly motivating: if students can get it to run, “Hello World” is displayed in a terminal screen.

Next, consider the Whirling Butterfly introductory tutorial from the Scratch site:

![Whirling Butterfly](image)

Figure 1. An introductory tutorial in Scratch

With Scratch, you drag blocks from the left panel to the middle scripts panel, and immediately see your program run in the right panel. The blocks are understandable and the environment lets you explore their meaning interactively. The result is an engaging tool proven successful with kids and as an initial component of a university course [11].

App Inventor can be thought of as Scratch for mobile phones—the two languages even use the same underlying code library for their blocks [14].

Because App Inventor blocks encapsulate complex technologies like SMS texting and GPS location sensors, even an initial sample can be quite motivating. Consider this one, which I used this past semester on the first day of class:

![An Introductory App Inventor app](image)

Figure 2. An Introductory App Inventor app

This sample auto-responds to a text received by the phone, sending back the text, “Sorry, I’m driving, I’ll get back to you later.” It’s a simplified version of the “No Text While Driving” app which will be discussed later. The blocks are understandable, as with the Scratch example. However, the app’s effect is not enclosed within a virtual world, but has meaning in the real one. Because of this, students immediately ask questions. “Could you make it so the response sent back could be customized? Can you make it so the texts are spoken aloud? Can you write an app that let’s people vote for something by texting, like on American Idol?” As a teacher, I know that such inquiries translate into students wanting to learn and driving the learning process.

3. App Inventor and its use in the classroom
App Inventor has two main windows: a component designer for building the user interface and a blocks editor for defining an app’s behavior. You can also test your apps live directly on your phone or an emulator.

The Component Designer, shown in Figure 3, is a fairly typical WYSIWYG tool for designing user interfaces: you drag objects into a view, modify component properties like background color and width, and in general specify how the app will look. There are UI components for building form-based apps,
canvas and image sprite components for building graphical, animated apps, and video and sound components for multi-media.

In the Component Designer, you also specify the non-visual components your app needs for accessing the technology available on the Android device. These include components for sending and responding to SMS texts, for interacting with the GPS and orientation sensors, for text-to-speech and speech recognition, for scanning barcodes, for saving data persistently, and for talking to the web.

3.1 Programming Behaviors

Once the components of the app have been specified in the Component Designer, the app’s behavior is defined in the Blocks Editor (see Figure 4). The Blocks Editor has two palettes. The first, My Blocks, contains functional blocks for interacting with the components that have been added to the particular app. The second palette contains built-in blocks for standard programming control and functionality including foreach, if-else, list and text manipulation, and logical operators.

The blocks language provides a low-threshold entry to programming. The elimination of most typing dramatically reduces the frustrations beginners have with syntax, the blocks provide visual cues to ease the development task, and only some blocks lock into place, reducing the possibility of errors.

The language also works because you define an app’s behavior directly as a set of event-handlers: when event X occurs, do Y (think of the complexities of programming the response to even a button click event in Java). For the app in Figure 4, the programmer has specified three event handlers: Texting.MessageReceived for processing an incoming text, LocationSensor.LocationChanged for tracking the driver’s current location, and SubmitResponseButton.click for recording a custom response entered by the user. Within weeks, students can learn enough event-response programming to create simple location-aware apps, SMS processing apps, and games on the order of Mole Mash or Tic Tac Toe.

Once students become comfortable with event handling and function call blocks, I introduce more complexity and abstraction through a quiz app assignment. Quizzes are fun because students can choose a topic of interest and easily include video and sound with the provided high-level components. Motivation is important because building a quiz requires a significant amount of abstract thinking, logic and traditional programming skills, e.g., using an index variable to walk through a list. When you add in variations such as multiple-choice answers and user-generated quizzes, the task is even more difficult. Fortunately, the students have been “lured in” by this point and are thus sufficiently motivated to make the conceptual leap necessary.

Later in the semester we explore persistent data, accessing APIs, and multi-user apps using the high-level components provided. The TinyDB and TinyWebDB allow for persistent storage using simple key-value pairs. TinyDB is used to store data directly on the device, while TinyWebDB is used for shared databases and also for accessing web information sources.

The TinyWebDB bridge to the web is important because as students feel more empowered creative, they often want to build apps that access web information. I don’t want the response to their creativity to be, “no, that’s not possible”. Furthermore, I’ve found that class discussions concerning APIs, mashups, interoperability and cloud computing are much better when the students have had some direct experience.

To allow for web-enabled projects, my teaching assistants and I have created a number of App-Inventor-compliant “wrapper” APIs for student projects, including one that gets event information from the Eventful API and another that accesses San Francisco’s MUNI bus schedule. Creating the APIs requires traditional programming knowledge: we wrote them using
Python and App Engine (see appinventorapi.com for more information). Once created, the students can create client apps in App Inventor using a simple `getValue(tag)` call to the TinyWebDB component.

### 3.2 Testing, Deploying, and Sharing

In the summer of 2010, App Inventor added live testing to the environment: you can now plug in a phone during development and interact with the app you are building in real-time without compiling or downloading after modifications.

Live testing helps immensely with user interface refinement: you can modify component properties and immediately see how they’ll look on the phone. It also helps with testing behaviors. For example, after plugging in some or all of the blocks in the Texting.MessageReceived event of Figure 4, a student can ask the person next to them to send them a text, and immediately know if the app is responding as it should.

Live testing is also integrated with an Android emulator, so you can develop and test apps even if you do not have a device available. This is especially important for education and settings in which not every student has access to a phone.

Once an app is tested, you can deploy it by packaging it in an Android app (.apk file) or generating a barcode for it. The barcode appears on screen and you can then scan it on to your phone.

Finally, you can generate source code for an app that you want to share. Because the source “blocks” are actually decipherable, App Inventor apps tend to be open-source in practice and not just theory. This sharing will be even further enhanced when App Inventor launches its planned Scratch-like community site.

### 4. Impact Outside the Classroom

The USF Computer Science Department has a Senior/MS Project course in which the students develop software for a client from industry or academia. At the end of the semester, the students present their work at our CS Night. The course has been successful in giving students real-world experience and helping them get internships and jobs.

Whereas the project course is a capstone for our most advanced students, the App Inventor course is on the other end—beginners with no programming experience from every major in the university besides computer science. Despite this, the App Inventor course has provided many of the same benefits as the advanced project class. In fact the students in the course now present alongside the seniors and masters students at our CS Night. This section tells the stories of three App Inventor projects that had a particular “real-world” impact outside of the classroom.

#### 4.1 No Text While Driving and State Farm

Daniel Finnegan is a creative writing major who I’m sure spends more time thinking about the human condition than cell phone apps. But he had read about the dangers of texting while driving: incredibly, something like 28% of accidents involve a phone [12]. He came up with an idea for an app to help and built the precursor to the SMS auto-response app shown in Figure 4.

Daniel’s app struck a chord with people. The App Inventor team described it on their About page and it was cited in a New York Times article [10]. I had begun writing tutorials for the App Inventor site and created an annotated and refined version of the app. In August 2010, we were excited to see that State Farm Insurance launched “On the Move”, an Android app that was quite similar to “No Text While Driving”: [Figure 4. App Inventor Blocks for No Text While Driving.]
“It is our hope that this widget will prevent crashes and save lives,” said Laurette Stiles, Strategic Resources vice president at State Farm. “This new service will help drivers manage the temptation to read or respond to text messages when they are behind the wheel. We wanted to make this widget available free-of-charge as just one of the ways we’re working to keep our roadways safe for drivers.” [15]

The State Farm app may not have been directly inspired by “No Text While Driving”, but it’s quite possible it was. In pondering this question, I realized the incredible nature of it: an app created in an introductory course, by an English major who had never programmed a computer, possibly helping inspire a mass-produced piece of software! And one that saves lives!

![State Farm’s “On the Move” app](image)

Figure 5. State Farm’s “On the Move” app

### 4.2 BroadcastHub in Helsinki

Ken Banks gave a talk in my fall App Inventor course on his FrontlineSMS tool [7]. Ken, one of the leading authorities on mobile technology for the public good, had observed that many people in developing countries have cell phones but no Internet access. Sending texts is expensive, but receiving them is not. So he built FrontlineSMS, a PC-based tool that serves as a broadcast hub for SMS communication. With it, people send an SMS text to the hub to register, then receive broadcasted messages from the group. Amongst other things, FrontlineSMS has been used to monitor election fraud in Afghanistan and help with health care in Malawi.

Carly Kralji, a student in my class, was especially enthused by Ken’s work. Her idea was this: because some situations, such as election monitoring, are so volatile, why not make the hub itself run on a mobile Android device (instead of software for a desktop or laptop, as with FrontlineSMS).

Ken Banks and his lead engineer, Alex Anderson, became interested in Carly’s project and generously gave their time to help Carly along. One thing Carly learned was that, ideally, the hub would make use of the contact groups on the phone. App Inventor does not yet allow full access to the contact groups, so Carly had to make due with a separate broadcast list in her version. The discussions with Ken and Alex, and the analysis between the ideal and the possible, gave Carly some great experience with real project specification and implementation.

Carly completed her Android version of the broadcast hub and posted it on her course portfolio site as well as the course’s “USF Android Market” page. Later that summer, Ken Banks contacted me to say that his group was creating a full-fledged Android version of FrontlineSMS. He very politely pointed out that Carly had named her app Android FrontlineSMS and he thought it would be less confusing if Carly changed the title.

We quickly changed the name to “Broadcast Hub” and I made a note to do a better job monitoring projects. I also realized I had a great example for use in class discussions involving ethics and copyright issues.

As with the “No Text While Driving” example, I was again struck by the fact that Carly, an International Design major in a core-curriculum course, had created something with utility to the world, something that Ken Banks cared about. Because of this, she (and I) had gained some great real-world experience and dealt with issues rarely faced in the classroom.

The story got better. Some weeks later, I created a version of the broadcast hub as a tutorial available on the App Inventor site [17]. Soon thereafter, I was contacted by Myles Byrne of a nonprofit group based in Helsinki, Finland:

“My wife and I are respectively a cancer scientist and a bioinformatician who moved from San Francisco to Helsinki last year. We’re working extracurricullarily with a local arts organization to try to lift a central neighborhood to a more creative and inclusive plane, especially regarding the growing immigrant population. So there is at least some relatedness to the mission of FrontlineSMS.

So we’re organizing events from now till 2012, when Helsinki is World Design Capital. The first such event is Saturday, and we’re trying to get broadcastHub working...

I was planning to learn App Inventor at leisure. But hours of searching for commercial SMS providers doing what broadcastHub does (for Helsinki users) turned up nothing."

This first email went on to ask about some minor technical issues he was having with App Inventor. I received a second email a few days later:

Dave, our community raising event in Helsinki went off really well. About 1000 people came (see [http://www.punajuuri.org/](http://www.punajuuri.org/))

We needed SMS broadcast capability for up to a thousand people over a few days. We looked at FrontlineSMS, Clickatell, and others – nothing had the right fit. Then I got my invitation to App Inventor, looked at the broadcasterHub tutorial, and realized with a shock this was the perfect solution. In a few hours I was able to modify the broadcastHub tutorial app to fit exactly our needs:

Any SMS’s sent to my phone number with the app running received an autoreply: “Text ‘beatroot’ to this number to sign up for our message list.” Any numbers sending an SMS to me containing ‘beatroot’ or ‘Beatroot’ were added to the broadcast list. But because our event was in Helsinki, the broadcast messages needed to be in Finnish. So instead of writing them myself, I told the app to take any messages sent to me from Heta and Jon, the Finnish organizers of the event, and automatically re-send it to all the numbers on the broadcast list.

It really was incredibly fast and easy to modify the broadcastHub app to do only what we needed. The app worked perfectly and can be easily re-used, modified, and shared. Having the capabilities of an android phone plugged into a graphical programming environment is an amazing experience.
You’re not just learning logic, you’re learning it in the context of the social world connected to your phone.

Thanks David and Google for bringing the next level.

Carly’s project had not only given her great experience, it had been customized for a real event! It occurred to me that App Inventor was not only an experiment in motivating beginners, but an experiment in open source software. Plenty of software is theoretically open source, but in practice is only “open” if you have the key: expert programming knowledge and the time to rebuild a project from the source. Myles and his group were able to use the BroadcastHub sample because, unlike most programming code and systems, App Inventor source is decipherable, even to people who are not expert programmers.

4.3 SOS Emergency App

Chris Witte and Elena Miguelena were in my first App Inventor class of Fall 2010. The first version of App Inventor began working about a week before the semester began and the bugs were not yet ironed out.

Despite these drawbacks, Chris and Elena came up with a great idea for an app: you’re walking home at night, approaching that dark and street that always gives you the creeps. Before you turn onto it, you start the app and press a button. If you don’t press the button again within the next minute, an SOS text message is sent to a pre-determined list of numbers.

They built a prototype of the app in class, and Chris showed it to his mother. Late in the semester, his mother mentioned Chris’s app to a man sitting next to her at a basketball game. The man happened to be a former executive at Apple, and really liked the idea. He gave Chris’s mother his contact information and told her to have Chris call him. Soon Chris was involved with a startup developing the SOS app and applying for a patent on it.

One interesting side note to the story is that Chris is dyslexic. He’d tried programming before and was overwhelmed trying to learn the Java programming language, which is text-based. Chris had much greater success with App Inventor’s visual language and its color-coded blocks.

5. Summary

App Inventor combines a low-threshold entry to programming with the ability to create apps that have a real impact. The visual, event-based nature of the language reduces the syntax problems common to beginners and gets them past the first “bump” in learning programming. They can almost immediately build apps with real-world utility, which motivates them to tackle the second “bump”: learning how to solve hard logic problems and specify interactive behavior with a static (though visual) language.

6. ACKNOWLEDGMENTS

Thanks to Google for its grant support and the App Inventor team for their support of App Inventor at USF.

7. REFERENCES


