Mark Ha '13

Adviser: Brian Kernighan

Final Thesis Outline

03/25/13

**"EasyAndroid" - Developers' Tool for Android Beginners**

Android Development Made Simple

Intro (3-5)

Motivation (1-2)

Related Works/Research (3-4) + pictures

Design (~10)

Implementation (~10)

Limitations (2-3)

Challenges (1-2)

Evaluation (2-3) + pictures

Market Viability (5-7)

Future Steps (2-4)

Conclusion (2-3)

Total: ~41-54 pages

Table of Contents

1. Introduction
   1. Summary of EasyAndroid – what it does, how, very *general* conclusions/results
   2. My approach to figuring out what to build in EasyAndroid – built my own “simple” Android apps, and figuring out 1) what people would want, and 2) what is required to build those kinds of apps?
   3. Explanation of some terminology (for the purposes of the paper)
      1. Android app
      2. apk
      3. API
      4. widgets (objects)
   4. What do I consider “Core functionality”
      1. Command line
      2. end-to-end process. From the coding to the apk file – from idea to Android app
   5. Original timeline – and changes to the outline
      1. “Plan of Attack” – successfully followed (!?)
         1. Core functionality first, then build on top of that
         2. Iterations with user testing to improve when done
2. Motivation
   1. Mobile is hot – Android even hotter than iPhone, in some ways
   2. “Entry” cost for Android development is high
      1. Download and install software
      2. Learn Java (for some)
      3. Learn Android “library”
   3. There’s no good way to just “try out” developing for Android.
      1. AppInventor probably the best option right now. But it’s a completely new “language,” that’s visually oriented, and does not carry over to real Android development – basically if you start using AppInventor, you have to continue to use AppInventor and lose your invested time if you choose to transition to traditional development.
      2. So, “trying out” Android means downloading and installing the SDK, and then going through the Android tutorials on the Android website.
   4. At the beginning of the year, I thought that nothing even close to the idea existed for Android development.
      1. A product that simplifies the Android app development process
      2. How to simplify the process? “DroidDev” simplified it by hosting things on a server (cloud), which meant no installations or downloads required. Drawback is that it required an Internet connection. Internet connection is becoming more reasonable to assume that people have, but it’d be nice to be able to develop on the subway, or during a thunderstorm.
      3. Then... simplify the coding process?
   5. Lots of upfront work just to do a “HelloWorld”
      1. Did I succeed in removing this? Still require downloads and installs so not really.
      2. Once installed however, HelloWorld is just a few clicks (and typing “HelloWorld”).
   6. Working with the Android library is a hassle
      1. What is an “Activity”? What is “Context”?
      2. XML or Java?
      3. File structure is delicate, especially for the “res” folder
3. Related Works/Research (a couple in detail)
   1. Scratch
      1. Visual Learning
      2. For kids
   2. AppInventor
      1. Hosted online
      2. Emphasis on “learning” visually
      3. made from Scratch (pun intended)
   3. “Droid Dev”
      1. Senior Thesis last year
      2. Took away the downloading and installing part
      3. Included a phone preview (essentially the Android emulator?)
   4. Twill
      1. Instead of browsing the web, interact with the command line
      2. Python interface
      3. My “minimal” version tries to turn coding into command line interactions
   5. IfThisThenThat
      1. Almost “ideal” model
      2. User does minimal work, does not know what goes on behind the scenes – black box
      3. 7 steps:
         1. Choose Trigger Channel (service)
         2. Choose Trigger
         3. Define Trigger properties
         4. Choose Action Channel
         5. Choose Action
         6. Define Action properties
         7. Create and activate (confirm)
      4. Create recipes of IF *trigger* THEN *action*.
         1. Triggers are categorized by service (ie. Facebook, Gmail) and then the options dealing with those services (ie. new message, new email).
         2. Actions are the same – categorized by service, then option. But the options are different, since in this case, you’re taking an *action*.
      5. Example: I made a “recipe”: IF (Gmail: new mail received), THEN (SMS: send a text to ###)
      6. Other course, requires login/registration of your other services
      7. My Design follows similar model (tries to). Example: User chooses Button -> Defines the Button -> Confirms its addition to the project
      8. Limitations: triggers and actions are what IFTTT gives you, nothing more, and nothing less. No customization outside of defining the trigger/action properties
   6. Andromo (see my Marketing Viability paper)
      1. VERY similar product: Allows user to add activities of certain types/templates
      2. Offers more common services – a YouTube page, links to Twitter/Facebook pages, etc.
      3. All online
      4. Decent success
      5. Same limitations as IFTTT
   7. AppsGeyser (see my Marketing Viability paper)
      1. Takes a website and turns it into an Android app
      2. All online
      3. Very successful
      4. User has 0 say in the process – you give it a URL, it gives you the apk file.
   8. Appcelerator Titanium
      1. Titanium takes JavaScript code, analyzes, preprocesses, and pre-compiles it into symbols that the native mobile compiler understands.
      2. Essentially converting JS code to mobile source code
      3. Developers need to learn and interact with Titanium API
      4. Speed of development, vs. “efficiency of app” –
      5. Allows JS developers to create a mobile app
4. Functionality
   1. BRIEF description of Easy Android
   2. API
   3. Flow of the tool (Make a diagram/flowchart)
      1. User has an app idea
      2. User inputs that idea via GUI
      3. Input from GUI is normalized to regular commands
      4. Commands are put into a parser, which converts the commands into a project state
      5. Project state is translated into meaningful Android/Java code
      6. X step: BUILD – Code is compiled and built into apk file via Android tools.
      7. X+1 step: apk installed via ant build script to attached USB device
5. Design Decisions
   1. Scope of my thesis – what were the options, and determining factors for each of these decisions?
      1. “Mission statement” / goal
         1. simply the Android development process
      2. Target audience
      3. How many / which features?
      4. Importance of GUI to users
         1. If I were targeting 70-year-old thesis advisers, I probably could have stopped at the command line interface
         2. GUI is a must for the non-CS majors / non-techies
      5. Related to command line/GUI -> documentation problem
         1. If you have a GUI, can just show the options... but still need explanation / description?
         2. Either way, need documentation: “Help” docs
      6. Importance of capturing the Android development process from start to finish, everything completely within EasyAndroid. That is, users can’t be expected to know how to open up the command line and run an ant script to build the project / generate the apk file.
      7. Importance of SDKs, libraries, etc.
      8. Limited resources (me): prioritization of features is important as a result. Even I don’t do everything that I was planning, at least I have something that works, and does what I said I would do, even if in limited capacity.
      9. Designing for normal user vs. malicious user
         1. someone trying to use the tool vs. someone trying to break the tool – very different
      10. Realized that objects need to be global variables (mainly for custom functions)
      11. define “generic” functions, ie. ADDITION – let user define params
   2. General approach
      1. Hard-code everything vs. abstract/design models for everything
      2. Given time limitations, what is the appropriate amount?
      3. Ideally: abstract everything, have maximum flexibility in the design structure, think 100 steps ahead and be prepared
      4. Obviously, that is impossible, so make do: think as far ahead as possible.
   3. Lots of example/sample apps
      1. Introduction: “Design by example”
         1. Code example app -> generalize/improve tool to be able to make that app
      2. Hello World
      3. TipCalc
      4. Accelerometer
      5. Camera
      6. Contacts List
      7. GPS
   4. LOTS of data structures
      1. Make sure all of them are consistent, and create a “general format” for them
      2. Avoid “hard coding” things as much as possible
   5. NO XML approach
      1. Explain Android’s typical utilization of XML layout files, vs. Java code approach
      2. Easier for me, because don’t have to deal with creating multiple files, deep within project directory. Plus, good for teaching Java.
      3. Use of “root” layout, and linear vs. relative vs. other base layout styles.
   6. Architecture of the project?
      1. Flowchart
      2. Object-oriented approach: commands from command-line modify the project object (ActivityCode), also need to modify AndroidManifest (though the user doesn’t see this part), can modify Imports, Custom Functions, Permissions
      3. Each new widget goes into a list. That list gets translated into Java code.
6. Implementation – describe each available command, how it was implemented, design decisions/challenges associated with that particular command
   1. Implementation Decisions
      1. Everything was done in Java
         1. I questioned early on whether other languages would be easier, but soon realized that doing things with a little extra effort in Java, my best language, would probably be better than learning a new language AND learning how to do the same things in that new language, even if the new language was specialized.
         2. Additionally, my project expands across several different types of modules (Data structures, Parsing, GUI), which makes it unlikely that there would be a language that makes all of those easy-to-implement
      2. Run as a java program (executable)
   2. “Global” versus “Local” Commands

<Global>

* 1. Open File, Save File
     1. “Elegant Solution”: record project “state”
        1. Requires definition of “state” as a data structure / format
        2. Project has X classes, each of which are made of Y objects, each of which are made of Z properties.
        3. Doesn’t seem *too* complicated, but would require careful planning and design for the definition of a project state.
     2. “Dirty solution”: record user’s input (valid) and re-run it all upon load/startup
     3. Long-run, I would want to implement the elegant solution, but short term, not enough time -> dirty solution. The two are structurally different, but user sees no difference except during load time – a performance difference that doesn’t even become painful except for larger projects.
     4. Elegant solution scales MUCH better. Would be mandatory for a real product.
     5. Dirty solution better for a minimal project, but quickly out-scaled by Elegant solution
     6. Elegant solution requires a separate, special “state” parser to be built for load file as well, as opposed to simply throwing the saved input into the usual parser.
  2. Build
  3. Install
  4. Reset
  5. Help

<Local>

* 1. Path
  2. Project Name
  3. Main Class
  4. Package Name
  5. Add File
  6. Class Name
  7. “Custom function”
  8. Custom import
  9. Add (Object)
     1. TextView
     2. EditText
     3. Button
     4. Contacts List
     5. ...
  10. Generating my own Android Manifest
  11. Permissions
  12. GUI
      1. Design
      2. Implementation

1. Limitations
   1. Not trying to make a complex, full-feature app, ie. Facebook or Angry Birds.
      1. Does not scale too well with complexity of app due to limited functionality available (ie. can add a button, textfield, etc. – a finite number of things, not *everything* available to Android typically).
      2. If making a “real” mobile app is the goal, hire a real Android developer.
   2. Never really solved the problem of removing the download/install software step
      1. Still depend on Android/ant/Java software existing. Also things like environment/PATH variables could definitely cause problems
      2. The cost of not hosting things on the cloud
   3. Open/Save file – need to implement elegant solution
   4. No “compiling” or error-detection before the actual compilation.
   5. Incomplete (at the moment)
2. Challenges
   1. Working with various Java libraries (I/O, Swing) that I’m unfamiliar with
   2. design challenges, remaining flexible and general
   3. Figuring out what I want. **FOCUS**
   4. Target audience too big? Targeting both non-programmers AND programmers to some extent, even if focus was on non-programmers
      1. Does this even matter? The main feature is the Java code, and it is not a huge extra step
   5. Not the BEST coding practices shown by the Java code that is generated
      1. But it does the job
      2. If someone knows the “best coding practices” themselves, then they can learn Android and do it themselves, directly.
      3. Target audience is someone who has no idea that “best coding practices” even exist
3. Evaluation
   1. “Experiment” / User testing setup
      1. Have the user build some simple example, ie. HelloWorld or TipCalc, using EasyAndroid
      2. How many succeed + how long?
      3. Rate the experience
      4. What was good / what was bad?
      5. Would you use it again?
   2. Report data
   3. Analysis of data
   4. What’s most important?
      1. What goes into the second iteration of code?
4. Market Viability
   1. Analysis of competitors
   2. Analysis of the market
   3. Comparative Pros/Cons of EasyAndroid
   4. How would the incumbent products affect EasyAndroid’s product path/roadmap
   5. Verdict: succeed / fail?
5. Future Steps
   1. LOTS of potential
   2. GUI improvements...
      1. Drag and drop is key
   3. “Elegant solution” for open/save file – use idea of state
   4. Idea of a “meta app” – generate code in order to generate the code of the EasyAndroid application, which generates the code for the Android app.
   5. Data scraper/reader to convert Android docs into data format that my program can parse.
      1. Removes the need of custom definition of each object/widget that can go into the Android code.
      2. Make the computer do the work instead a human
   6. Can always add more features – shortcuts to interacting with YouTube, Twitter, Facebook, etc. (commonly used apps)
   7. Display of available objects
      1. as the user adds widgets to the project, add them to a list that can be selected from (as parameters for example)
   8. “Preview” screen (Android emulator style)
      1. Assume user has USB device
      2. (This can also go in the design decisions section?)
   9. Abstract to the max
      1. Allow for more complicated code to be abstracted out so that a simple user can use, ie. layout designers
      2. event or trigger based code is definitely doable, ie. scratch
      3. Definitely makes sense with mobile as most apps are just built around user interaction (the rest can be done start-up, or through environmental or time-based triggers)
      4. Lots more to say about this
6. Conclusion/Reflection (Things I could have done better)
   1. Started off great, slowed down due to other classes, etc.
   2. Ultimately, just need more time if I wanted to do everything that I had planned / imagined to do
   3. Narrow my focus more / maintain focus on what I had said I would.
      1. Trying to do too much with too little time
   4. More emphasis on research early on, related products and such – had assumed that the 3-4 that I knew about already would be enough, perhaps could have helped me shape my design differently
   5. Even spending as much time as I did on design, there were still things that I didn’t think of, which I had to go back and change because of them
      1. For example, specifying custom functions: originally created a new BufferedReader, but that didn’t work well with open/save file. Had to change so that I could parse it seamlessly
      2. Separation of “global” and “local” commands
         1. global commands don’t get saved in the project data – get saved as global settings / don’t get saved at all (ie. print, debug)
         2. only local commands need to be saved into the project data
      3. imports
      4. permissions
      5. custom functions
      6. Arithmetic
   6. Sincerely believe that if I were developing this full time, it could be a real, profitable product
   7. What did I learn? What were the takeaways?
      1. Not an easy task to “abstract” an entire platform
      2. Important to understand my limits – being ambitious/a visionary is good and important, but so is being realistic
   8. All in all, the product / technology is pretty cool.