Teaching Mobile Apps for Windows Devices Using TouchDevelop

Simon
University of Newcastle
simon@newcastle.edu.au

David Cornforth
University of Newcastle
david.cornforth@newcastle.edu.au

Abstract

Even in a computer science degree, some students find it very hard to learn programming. In a less programming-oriented degree such as information technology the problem is amplified, and it is a real struggle to engage some students with the programming courses. A wealth of literature describes various approaches to teaching programming in the hope of addressing the perennial learning problems and encouraging the students to engage with the material. One approach to engagement is to teach the most current material, and one form of material that is highly current is programming apps for mobile devices. In this work, we report on one approach using a new programming language that was specifically designed for mobile app development. The approach was a success, with students becoming engaged. However, there are issues which we hope to address in the future.

Keywords: native mobile apps, programming education, TouchDevelop

1 Introduction

In the Bachelor of Information Technology degree at the University of Newcastle, Australia, the focus on learning programming skills is somewhat less than would be expected in a computer science degree. While all students must complete the first programming course, subsequent programming courses either form parts of particular majors or are entirely elective. In this context, it was decided that the second programming course would teach the development of apps for mobile devices, in the hope of attracting students who might otherwise take no programming courses beyond the first.

Jackson et al (2013) discuss reasons for teaching mobile app development, including relevance to business, the ease of programming games, the appeal to students who work all day with mobile devices, and a response to the challenge of retention. Another compelling reason is that the teaching of programming should reflect the current trend in computing platforms away from the desktop and towards mobile devices. As Tillman et al (2011) note, more touchscreen devices were sold in 2011 than laptop and desktop computers combined. It is important that graduates be made aware of this, and of the challenges it poses for the IT professional.

Mobile apps can be developed as either web apps, which are accessed by way of a web browser on the mobile device, or native apps, which are compiled expressly for the device in question. Web apps have the advantage that they can be accessed on any web-enabled platform, while native apps must at best be recompiled for each target platform, and in many development environments must be largely rewritten to suit a particular platform. On the other hand, web apps may be generic apps that have limited access to the hardware of the phone, whereas native apps tend to offer far greater access to the hardware and software specifics of the platform for which they are written. Whichever of these approaches is chosen, there is also the issue of testing the app on a mobile device and deploying it to the device. For this course we wanted to give students an awareness of both the web apps and the native apps approaches, including interaction with cloud services, as well as broad exposure to the device-specific features such as accelerometer, compass, contact lists, etc.

2 Android, iOS, or other?

When contemplating a native mobile apps programming course, the choice of platform is crucial for several reasons. Goadrich and Rogers (2011) discuss the range of development platforms and languages for native mobile apps, and conclude that there are two obvious options: Apple's iOS to develop apps for various Apple devices (such as the iPhone), and Android to develop apps for Android devices. After conducting a detailed comparison of the two, they conclude that while iOS probably has more teaching and learning overheads than Android, either is suitable. However Skelton et al (2013) note the need for an iOS or Android app to go through a review process before being made available to other users. This would represent an unwelcome delay for many students.

It is probably reasonable to expect that many of the students will already have an iPhone and the rest will already have an Android phone. Each group will presumably therefore feel disenfranchised if the course teaches programming for the device that they don't have.

However, the teaching and learning overhead implicit in the decision is far more important than the consideration of which students have which device. Notwithstanding the various IDEs available with either platform, programming an Android device tends to entail programming in Java, and programming an iOS device tends to entail programming in Objective C (Skelton et al 2013). If the course objectives include teaching the students a new programming language, the choice can be made in that light. But if the principal objective is to

expose students to the concepts of mobile apps programming, there could be virtue in choosing the language that has least additional overhead for the students.

Various authors (Roy 2012, Dabney, Dean, & Rogers 2013, Honig 2013) observe that Android is the obvious choice if the goal is to reduce the additional cost of learning a new language. However, they do this in the assumption (sometimes not explicit) that the students will already have learnt to program in Java; and while this assumption holds good for many computer science degrees, it is not universally valid.

In the Bachelor of Information Technology degree at the University of Newcastle, Australia, the first programming course uses the Python variation of the Media Computation approach of Guzdial and Ericson (2013). Some of the students will additionally take an introductory software engineering course that uses Java, but this is by no means given, and indeed the software engineering course is not offered on all campuses of the university. Therefore it cannot be assumed that the students enrolling in the mobile apps course will have programmed in Java; and experience shows that most of them have not.

Both Objective C and Java are so different from introductory-level Python that a course using either will need to devote considerable time to teaching the language, leaving less time to teach the mobile apps concepts that are intended to be the focus of the course.

There are many options, some of which can be deployed on several platforms without the need to rewrite code. Some of these environments are conveniently summarised by Smith (2012). It was through this summary that we became aware of the new TouchDevelop language (Horspool & Tillman 2013), developed by Microsoft for Windows mobile devices. This language seemed significantly simpler than either Java or Objective C, suggesting that it could be taught alongside the mobile apps concepts rather than as a prelude to them. Moreover, it permitted a neutral approach to the students' own devices, as Windows phones are by no means common among the students at our university.

After some time spent exploring the TouchDevelop language, we therefore decided to proceed with the course using this language. Further reasons for this choice were:

- Requiring implicit rather than explicit knowledge of OO concepts
- Powerful built-in features
- Simple deployment to device
- Simple access to phone sensors
- Common development environment on web and phone
- Simple sharing of developed code
- Online community

Discussing best practices in mobile app development, Mahmoud (2011) stresses the importance of providing students with mobile devices, and mentions academic programs "that provide devices free of charge to academic institutions interested in integrating mobile devices and mobile application development into their curriculum."

Our course proved no exception in this regard: Microsoft contributed a number of phones that were close to being supplanted by newer models, and we ended up with enough for each student to borrow one for the duration of the course.

3 The TouchDevelop Language

TouchDevelop has been developed to create mobile apps that can then be deployed on a mobile device. However, unlike other mobile app development languages, a central feature of its design is the ability to develop code using the mobile device itself as a platform. A TouchDevelop program is written not by typing text but by tapping tiles, the usual means of interacting with software on a mobile device; and this can be done just as easily on a smart phone as on a desktop or laptop computer.

Notwithstanding the design intention, writing code on a mobile phone can be fiddly. Fortunately, it is also possible to develop code on a TouchDevelop web application, and the teachers and the students all agreed that this was by far the preferable approach.

TouchDevelop is cloud-based, and programs (scripts) and other data are stored on the cloud. Access is by way of a suitable account (Microsoft, Google, Facebook, or Yahoo), and synchronisation is automatic: a script developed on the web application is available more or less immediately on the phone by way of the same account. Developing a script on the web application and testing it on the phone is a reasonably seamless sequence.

Goadrich and Rogers (2011) illustrate Android and iDevice development with a simple program that displays random permutations of the letters of "Hello world".

Their iOS project consists of four files. They write of one of these files: "The syntax may appear daunting at first blush, but it takes just a few minutes in the classroom to explain to students familiar with OO-ideation" (Goadrich & Rogers 2011). They go on to emphasise that the complex-sounding process of creating the project is actually quite simple once the programmer is accustomed to it.

Their Android code, also in multiple files, includes at least 45 lines of xml and 30 lines of Java.

By way of contrast we illustrate TouchDevelop code with a script that does the same thing – though for good measure, the permutation is activated not just by a button press, as in the examples of Goadrich and Rogers, but also by shaking the phone. As shown in Figure 1, there are 17 lines of code (one line is wrapped to fit in the figure), in a single file. Even then, the code is more strung out than it needs to be: it would be easy to write a shorter version. This example alone should make it clear that TouchDevelop programming is simple in comparison with iOS and Android.

Our students at the University of Newcastle cannot be assumed to be familiar with OO concepts or with the languages used in the sample projects of Goadrich and Rogers. Furthermore, in their first programming course, every program that they encounter is written in a single file. For these reasons, among others, TouchDevelop would appear to be highly suitable as a next step for these students.

```
action main ()
   wall → add button("questionmark", "Permute")

private action permute ()
   var str := "Hello, world!"
   var newstr := ""
   for 0 ≤ i < str → count do
      var length := str → count
      var pos := math → random(length)
      newstr := newstr || str → at(pos)
      str := str → substring(0, pos) ||
           str → substring(pos + 1, length - pos - 1)
   newstr → post to wall

event shake ()
   ▷ permute

event tap wall Page Button (
   item : Page Button)
do
   ▷ permute
```

Figure 1: Complete TouchDevelop script for permutations of "Hello, World!"

3.1 Some features of TouchDevelop

This paper is not intended to be a guide to writing TouchDevelop code. However, reading the code in Figure 1 is likely to raise some questions, which this section will endeavour to answer.

Procedures/functions/methods are called 'actions'. The illustrated code includes two actions and two event-handlers.

Variables must be declared and initialised, and their types are deduced from the initial values.

Output is typically sent to the 'wall', which is the screen of the device on which the script is running. This builds on the familiarity of students with social media such as Facebook. The command in the *main* action displays a button on the wall, and the last command in the *permute* action displays the newly formed string on the wall.

The notation is different from the standard for C-based languages. The assignment operator from Pascal (:=) has been called back into use; members of objects are indicated with an arrow rather than a dot; inequalities are shown with the same symbols as in mathematics (eg ≤) rather than two-character compounds; and there are additional special symbols such as ▷ , which prefaces a call to an action, and ‖, which is the string concatenation operator. Special characters would be a problem with code entered from a standard keyboard, but of course are no problem at all when selected from a menu.

3.2 Writing TouchDevelop code

As indicated earlier, TouchDevelop code is written not by typing but by tapping tiles on the screen – or, when using a computer, by clicking them with the mouse. Even when a line of code is made up entirely of characters that can be found on the keyboard, an attempt to type that line will probably fail: the application does not always recognise, say, that the three letters 'v', 'a', and 'r' are intended to be the same as the 'var' token. Keyboard editing of an existing line of code is particularly troublesome.

Figure 2 illustrates an early stage in the development of a script. Five lines of code have been entered in the *permute* action, and the sixth is being entered. The three grey rows below the white code area contain tiles for anything that can legitimately be entered at the current stage of coding. The first six columns of this area remain invariant, providing continuous access to numerical and boolean constants; the remaining columns are context-dependent, changing for each stage of code entry. If there are more items than can fit into this space, there is a tile than can be tapped to move to a 'next' screen with further tiles.

Having recently tapped the tile for the variable *str*, the programmer has now tapped the *substring* tile. As a consequence of these two taps, the code `str→ substring(0, 0)` appears in the script; an explanation of the substring function appears at the foot of the code area; and the cursor is positioned immediately after the new code, ready for the programmer to move it back and, if required, replace either or both of the zeros with the desired arguments.

It is a consequence of this system of coding that syntax errors are extremely rare and semantic errors are rare. It is difficult to enter invalid code because the tiles on display are all valid in the current context. It is difficult to misunderstand what a command does because help is displayed automatically while the command is being completed. If the programmer does manage to enter invalid syntax, for example by leaving a statement before it is complete, a helpful error message is displayed below the statement and remains there until the error is fixed.

Writing code on the mobile device itself is similar, using the same principles, but squeezing the same options into a far more cluttered screen.

4 Positive Aspects

We found a number of positive aspects to the use of TouchDevelop as a development environment for the second programming course.

4.1 Using the phone to write code

It is potentially advantageous to be able to program on the phone itself, although the extra fiddliness of this approach means that the option of programming via the web application is generally preferred.

4.2 Web development environment

As a web application, the development environment requires no installation, just a reasonably current browser. The cloud storage for scripts means that one's scripts are available from any device at any time.

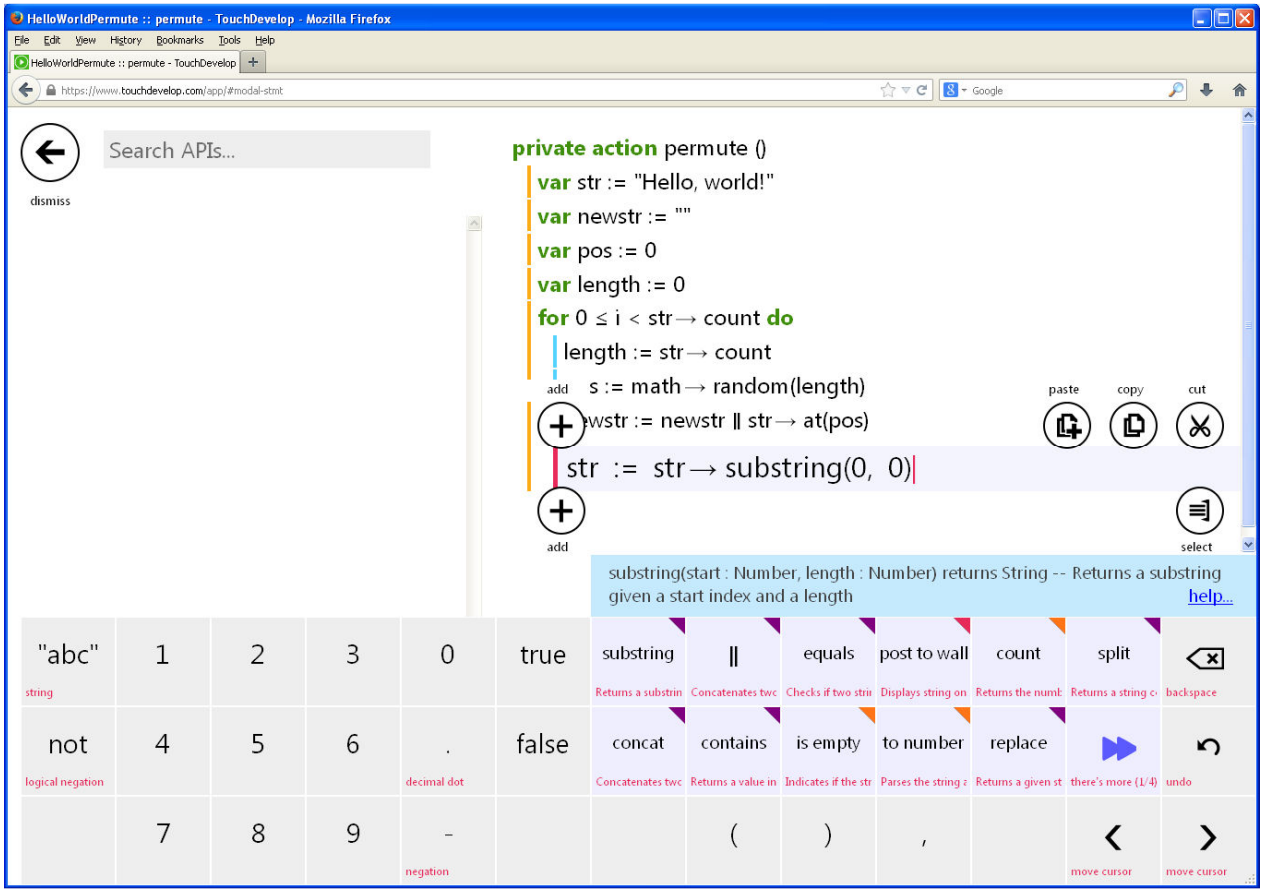

Figure 2: Entering code in the TouchDevelop web application

4.3 Language design

By and large, we found the programming language to be well designed. In particular, the language designers were prepared to throw away some (though not all) of the ill-conceived features of the C-based languages, such as the use of the equality symbol from mathematics to mean something very different from equality.

4.4 Ease of code sharing

On creation, every app is given a unique code of four or more letters. To publish the app entails just a few simple steps: there is no code review or similar barrier to publication. Once it is published, any user can search for and install the app with a given code. When our students submit apps, they tell us the corresponding codes. We can then use those codes to install and assess the apps, and can pass the codes to other students so that they can examine the apps and provide constructive comments on them.

4.5 Online developer community

TouchDevelop is more than just a programming language: it is an online developer community. Students are exposed to the collaborative nature of the programming task, and are encouraged to examine other scripts and learn from them. We played to this feature by requiring all students to comment on ten draft scripts produced by other students, and to provide considered responses to the comments on their own scripts. Many students at first considered this an unwelcome imposition; but by the time the assignment was complete most of them appear to have risen to the challenges, both providing positive feedback on other students' scripts and responding well to the feedback on their own.

4.6 Environment for further development

For programmers who feel the limitations of the language, it is possible to export a TouchDevelop script written for Windows 8 in a format that can be loaded as a project into Visual Studio. In the future we plan to explore the option of continuing development of a script in that environment, either through this means or by using TouchDevelop as a vehicle for rapid prototyping.

4.7 Novice-friendliness

The TouchDevelop environment is designed for and well suited to novices. The error reporting is excellent, and the immediate yet unobtrusive help makes it difficult for students to generate erroneous code, and extremely easy for them to diagnose and correct errors when they do arise.

This feature makes TouchDevelop ideal for the purpose for which we are using it: teaching the students to write apps for a mobile device, while at the same time teaching them a new language almost by stealth.

4.8 Support

As the students and teaching staff confronted a number of issues with the TouchDevelop language and environment, we were fortunate to have contact with people at Microsoft who responded very quickly to our requests for help.

5 Issues

Offsetting the positive aspects of the language and the development environment, we certainly had some concerns that affected both the academic staff and the students. However, the language and the documentation are in constant change. Having taught our course in semester 1 of 2013, we expect that some of the problems we encountered will have been remedied by the time we next teach it in semester 1 of 2014.

5.1 Code input

When using the web application from a normal computer, the inability to type most code on the keyboard can be frustrating. This is not such a problem when entering symbols that are not found on the keyboard, but is disconcerting, to say the least, when entering what appears to be straight keyboard text. For example, the expected way of entering a string is to select the "abc" token on the far left of the set of touchable tokens. This opens a new line below the line that is currently being written. The string is entered in that line, without quotation marks, and entry is terminated by touching or clicking back in the line being written. The string then appears in the correct place in the command, complete with quotation marks.

A keyboard-savvy programmer entering the same line of code might well type a quotation mark. This opens the same 'string-entry' line below the line being edited, but the programmer is possibly confused that the quotation mark is not shown. She proceeds to type the string, ending with a closing quotation mark; but anything typed in the string entry line is taken to be part of the string, so when eventually the programmer clicks back in the statement, the typed quotation mark has become part of the string, and gives rise to a syntax error – or, more recently, to a string that includes a spurious quotation mark at the end.

It doesn't take long for most people to learn to click or touch even when typing looks possible; but die-hard keyboard users continue to be frustrated by problems such as this.

5.2 Language in development

The language is still in development, and is being changed quite frequently. For example, during the semester of our course, the language appears to have dispensed with a particular type of layout box and the possibility of creating a tile to launch the app. We were able to find an undocumented workaround for some deprecated code, but this is presumably a temporary fix.

The reference book provided on the TouchDevelop site is almost of necessity perpetually out of date. It includes deprecated code that cannot be entered, describes features that are no longer in the language, and fails to mention new features that are. It is very clearly a reference book rather than a textbook. Some of the examples in the book are not very well thought out, or not very well programmed, or both. On the other hand, it is (at the time of writing) free, which is great advantage for the students.

5.3 Inconsistencies between web and phone

While programming on the web application is generally preferable to programming on the phone's TouchDevelop app, there are significant differences between the two. Some of these are quite subtle. For example, the code in figure 1 includes the expression `str→ substring(pos + 1, length – pos – 1)`. If the –1 is omitted, the expression defines the substring from index *pos+1* to one beyond the end of the original string. The web application deals with this comfortably, returning the substring from index pos+1 to the end of the string. But when the same script is run on the phone, it correctly generates an index-out-of-bounds runtime error. Some of our students chose not to borrow a phone (one student explaining that he already has too many phones), and were therefore unaware that the code they were handing in might not run on the mobile device even though it runs on the web application.

A related problem is code that runs on the web application but simply does not exist on the phone app, either because it is a new feature that has not yet been implemented on the phone or because it is an old feature that has not yet been removed from the web application. Code in this category does not simply fail to run on the phones: it fails to appear on the phone's list of scripts available for download from the cloud. A programmer might write four scripts on the computer. Within minutes, three of them can be found on the phone; but the fourth simply never appears. We eventually discovered how to find out why this was happening, and how to check whether a script had such problems. But because it was well into the semester when we found these things, some students never caught up with the discovery, and ended up submitting code that would not port to the phones.

Any script developed by editing another script comes with a history, and can be traced right back to its first ancestor. Perhaps the worst aspect of the problem described above is that if a script has code that prevents it from porting to the phones, every descendant of that script inherits the condition – even if the offending code is removed. Thus a script with no phone-incompatible code will fail to port to the phone because one of its ancestor scripts had some phone-incompatible code. All we could do for students in this position was suggest that they start a completely new script and enter all of the code again.

5.4 Cloud availability

There were times when part of the TouchDevelop cloud itself appeared to go offline for several days at a time. At such times it was still possible to write code on the web application, but it was not possible to log in from a mobile device or to access other people's work. This happened only two or three times during the semester, but, perhaps inevitably, one of those times was a weekend when students' work was due to be submitted.

5.5 Graphics

One issue that gave us serious concern was the matter of graphics. Any graphics used in a TouchDevelop script must first be uploaded to the cloud; the script then downloads the graphics to the device each time it is opened. All graphics uploaded to the cloud in this manner

become accessible to all TouchDevelop developers. While each graphic can be accompanied by a comment explaining who created it and who uploaded it, this information is not mandatory, and the TouchDevelop cloud is therefore host to a growing number of images with varying degrees of attention paid to their intellectual property and to associated privacy issues. Suppose, for example, that a student wishes to write an app that cycles through a series of photographs of her family while playing an appropriate soundtrack: then those photographs must first be uploaded to the cloud and become public property. From the academics' point of view, this has the potential to teach students an important lesson about privacy and intellectual property. However, especially under the pressure of deadlines, it also has the potential to encourage laxness with regard to these same questions.

5.6 Language design

Finally, while we have great respect for the language designers, we do wonder about some of the design decisions. For example, while the language has collections, it appears not to have arrays, a staple of programming for many decades. And while it has a *for* loop, the index of that loop must be integer, can start only at zero, and must count upward in ones. Of course it is possible to write the expression that converts from this constrained index to the desired sequence. But we would suggest that if one wants to count backward from 100 to 34 in steps of three, it is clearer to have an index going from 100 to 34 in steps of –3 than to have an index going from 0 to 22 in steps of 1 and to refer constantly to 100 minus three times that index.

It must be noted that the course was taught using TouchDevelop for Windows 7 devices. By the time this paper appears, that product will have been discontinued in favour of TouchDevelop for Windows 8 devices. The reference book has already been updated, and we anticipate that some of the issues discussed here will have been addressed in the new version of the language and platform.

6 What Did the Students Think?

The overall impression of this new course is that it was a success. It was clearly engaging, and most students appreciated the opportunity to program apps for a mobile device, although of course some would have preferred it to be for their chosen style of mobile device.

The main reason for choosing TouchDevelop over iOS or Android development environments was simplicity of the language, knowing that many of our students had programmed only in Python in the context of media computation. In this we clearly succeeded: all of the students who persisted with the course did well, and few of them had serious problems coming to grips with the language.

In a somewhat complicated transition arrangement, many of the students who took the first offering of this course had already completed two programming courses, typically in C#, and many of these students found TouchDevelop too simple. Pertinent comments from the standard end-of-course survey (possibly all from the same student) include:

- *The skills taught in the course in no way apply to the real world due to the fact that TouchDevelop is a hobbyist language for kids and is in no way used in the real world.*
- *The course was far too easy and that was due to TouchDevelop being used, go back to industry standards like C# or Azure, a course like this in no way helps me with skills I need in the workplace.*
- *teaches students to make small scale novelty apps for a platform that has no weight in the real world and the skills from TouchDevelop are not transferable as the skills are very basic in that they have been learnt in previous [first-year] courses.*

On the other hand, some students appreciated the simplicity of the language:

- *good content, good pace and understanding of student engagement. Didn't overload us or cram too much work into any one lecture.*
- *Actually, learning about APIs was pretty nifty piece of information.*
- *Touchdevelop can be annoying at times, but it is a really easy language to develop/ prototype in.*

Students definitely picked up on the fact that TouchDevelop is a work in progress:

- *Touchdevelop is still in beta with copious amounts of bugs, NOT suitable for a professional university course. Even if it were at production level it then might be suitable*
- *The platform for this course is under-developed, there are many problems which needed to be worked out, and would be inclined to postpone the use of the current platform until it is improved ... In future it could be a good platform to use, but as of yet, i would postpone the use of it.*

Notwithstanding the clear negative comments, it does appear that the overall impression was positive:

- *This subject is useful for student who interested in coding language. I realized that modern new TD language has similar concept with other coding language but it has wide as wide range of application so I interested in learning this course.*
- *This has actually been one of the courses I have enjoyed most at uni so far ... It doesn't feel like we are learning archaic concepts either, which makes it more motivating.*

In order to address the comments of students, we are considering changes to the course for next year. The perception that the course was too simple will be largely addressed by completion of the current transition phase. In future years, almost all students entering the course will have only the minimal programming background of a single semester using Python. For those students with more advanced programming skills we plan to allow a more advanced option where students may choose to complete their project using Visual Studio instead of TouchDevelop. However they will still be required to use TouchDevelop as a prototyping tool. This will allows us to offer a challenging path, using JavaScript and HTML, for advanced students, while still meeting the needs of students struggling to master the basics of programming.

The course material will of course be more highly developed for the second offering, so this should address some of the other issues that were raised by the students.

7 Discussion and Conclusions

Responding to the need to rework the second programming course in our information technology degree, we decided that a course in programming mobile apps would engage the students and persuade more of them to continue their programming education. However, we faced a substantial problem: having learnt only one semester of programming, using Python in a media computation context, the bulk of our students had no familiarity with any of the C-based languages.

The newly designed and implemented TouchDevelop language appeared to offer a solution to this problem. It is a simple language with robust error checking, and its use could free us from having to devote large amounts of time to teaching a programming language as opposed to the concepts and techniques of programming apps for mobile devices. Nguyen et al (2012) have empirically shown the efficacy of code produced, and the positive impact that this has on productivity. They showed that students using TouchDevelop completed more tasks than those using Android, and that the likelihood of a task being completed was greater for those using TouchDevelop than for those using Android.

Athreya et al (2012) observe that TouchDevelop is an appropriate environment for producing small apps that have simple features, but that are reliable and easy to get working. This provides further evidence of the relevance of TouchDevelop for teaching programming. Athreya et al (2012) also comment on the wide variety of scripts that users typically produce when using TouchDevelop. This is of benefit when offering projects to students for assessment. We find ourselves in clear agreement with these observations.

While we do have issues with TouchDevelop, the overall impression of both staff and students is that this was a good approach to teaching mobile app programming. Furthermore, the existence of a clear developer community led us away from the punitive develop-code-in-isolation model to one that acknowledges the role played by peers when developing code.

It is clear to us from the first offering of the course that, as covered in the reference book, there is not enough material for a full-semester course. In the next offering we hope to extend the course to add interaction with cloud-based data and to cover further development of TouchDevelop scripts in Visual Studio.

In closing, we should address one question that might arise when reading the students' feedback on the course: if TouchDevelop is so simple, why not use it for the first programming course rather than holding it back for the second? We have given this question serious consideration, and decided that this is not a change we are prepared to make. The students' comments were made in the context of having completed one, two, or possibly more programming courses. We do not believe that complete novices would find the language so easy to pick up. Indeed, we believe that the tile-based nature of the programming and the link to mobile devices might distract novice students from the essence of programming, which is where we focus in the introductory course. And finally, we think it unlikely that we would ever have enough phones to be able to lend one to each of our first-year students. For these reasons, we confidently expect that this course in programming for mobile devices will remain our second programing course for at least the next few years.

8 Acknowledgements

The authors would like to thank the Microsoft TouchDevelop team for the provision of phones for use in the first offering of this course, and for the generally impressive level of support provided during the course.

9 References

Athreya, B., Bahmani, F., Diede, A. and Scaffidi, C. (2012): End-user programmers on the loose: A study of programming on the phone for the phone. *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*, 75-82.

Dabney, M.H., Dean, B.C., Rogers, T. (2013): No sensor left behind: enriching computing education with mobile devices. *Proc. ACM SIGCSE Technical Symposium, SIGCSE'13*, 627-632.

Goadrich, M.H. and Rogers, M.P. (2012): Smart Smartphone Development: iOS versus Android. *Proc. ACM SIGCSE Technical Symposium, SIGCSE'12,* 607-612.

Guzdial, M.J. and Ericson, B. (2013): *Introduction to Programming and Computing in Python: a Multimedia Approach*, 3rd edition, Pearson Education Inc.

Honig, W.L. (2013): Teaching and assessing programming fundamentals for non majors with visual programming. *Proc. ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE'13*, 40-45.

Horspool, N., and Tillmann, N. (2013): *TouchDevelop: Programming on the Go*, 3rd Edition, Apress, 2013, ISBN 1430261374,9781430261377.

Jackson, S., Kurkovsky, S., Mustafaraj, E. and Postner, L. (2013): Mobile Application Development in Computing Curricula. *Proc. ACM SIGCSE Technical Symposium, SIGCSE'13,* 107-108.

Mahmoud, Q.H. (2011): Best practices in teaching mobile application development. *Proc. ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE'11*, 333.

Nguyen, T.A., Rumee, S.T.A., Csallner, C. and Tillmann, N. (2012). An Experiment in Developing Small Mobile Phone Applications Comparing On-Phone to Off-Phone Development. *Proc. IEEE Workshop on User Evaluation for Software Engineering Researchers*, 9-12.

Roy, K. (2012): App Inventor for Android: report from a summer camp. *ACM SIGCSE Technical Symposium, SIGCSE'12*, 283-288.

Skelton, G.W., Jackson, J. and Dancer, F.C. (2013): Teaching Software Engineering Through The Use Of Mobile Application Development, *Journal of Computing Sciences in Colleges* **28**(5):39-44.

Smith, N. (2012): Teaching Mobile Development. http://teachingmobiledevelopment.blogspot.com.au/. Accessed October 2013.