An Experiment in Developing Small Mobile Phone Applications Comparing On-Phone to Off-Phone Development

Tuan A. Nguyen, Sarker T.A. Rumee, Christoph Csallner
Computer Science and Engineering Department
University of Texas at Arlington
Arlington, TX 76019, USA
{*tanguyen,sarker.ahmedrumee*}*@mavs.uta.edu,csallner@uta.edu*

Nikolai Tillmann
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
nikolait@microsoft.com

***Abstract*—TouchDevelop represents a radically new mobile application development model, as TouchDevelop enables mobile application development on a mobile device. I.e., with TouchDevelop, the task of programming say a Windows Phone is shifted from the desktop computer to the mobile phone itself. We describe a first experiment on independent, non-expert subjects to compare programmer productivity using TouchDevelop vs. using a more traditional approach to mobile application development.**

I. INTRODUCTION

Programming applications for mobile phones used to be a niche task. However the recent wide adoption of mobile devices and especially smartphones, together with the app store mobile application distribution model, has elevated mobile application programming to a common activity. To support mobile application development, each producer of a major mobile application platform (iOS, Android, Windows Phone, etc.) provides a convenient software development kit (SDK). For example Google's Android SDK integrates nicely with the popular Java IDE Eclipse. This makes it easy for programmers to start programming mobile phone applications. I.e., programmers install an SDK on their desktop computer and develop a mobile application there as they would develop any other application. The SDK typically contains a powerful emulator or virtual device that allows programmers to simulate how their mobile applications will behave on an actual mobile device.

TouchDevelop represents a radically new mobile application development model [2], as TouchDevelop enables mobile application development on a mobile device. I.e., with TouchDevelop, the task of programming say a Windows Phone is shifted from the desktop computer to the mobile phone itself. While shifting the device on which to program, TouchDevelop promises to maintain much of the traditional, expressive programming style. I.e., TouchDevelop is still a traditional, expressive, Turing-complete, Pascal-like programming language.

The declared scope of TouchDevelop is small programming tasks that can be achieved in tens or hundreds of lines of code. Clearly, the small screen makes it a lot harder to manage medium or large-scale programs. This limited program size rules out large-scale professional development. So the declared circle of programmers centers on students and hobbyists [2].

While developing small programs in a traditional programming style on a phone is an interesting concept, it is not clear if it can be done in a way that allows average programmers to solve programming tasks efficiently. For example, the lack of a traditional keyboard may make coding tedious and slow. In prior work the TouchDevelop authors performed an informal self-experiment from which they concluded that the time taken to code in TouchDevelop is in the same order of magnitude as coding on a PC [2]. In this paper, we perform a first experiment on independent, non-expert subjects to compare programmer productivity using TouchDevelop vs. using a more traditional approach to mobile application development.

II. BACKGROUND: TOUCHDEVELOP

TouchDevelop includes the Turing-complete, Pascal-like TouchDevelop programming language, the TouchDevelop IDE, as well as a cloud service. The TouchDevelop IDE runs on Windows Phones and enables programmers to write and run TouchDevelop programs ("scripts" or "apps") on their phone. To be usable on a small touchscreen, the IDE employs a semi-structured code editor that treats all program tokens atomically and thereby minimizes the possibility of syntax or compile errors. I.e., the user either creates a new token or selects existing tokens from adaptive lists that reflect the current program context.

The TouchDevelop IDE is connected to the cloud, which allows developers to publish their TouchDevelop apps. All published apps are made available with source code. Programmers are encouraged to download and extend apps written by others. The TouchDevelop cloud logs if a new app by programmer A is such a derivative of another app by programmer B. The cloud also logs if an app is simply a new version of an earlier app by the same author.

The TouchDevelop language is kept simple by focusing on tasks that can be accomplished on a phone. This allows

TouchDevelop to implicitly assume many facts that have to be stated explicitly in other languages, which leads to simpler, more compact programs. For example, for printing some text a Java programmer has to specify a particular output stream. TouchDevelop only has one output stream, so a TouchDevelop app does not have to specify this detail and can be more compact.

III. Research Questions (RQ), Expectations (E), and Hypotheses (H)

To evaluate this new TouchDevelop programming paradigm, we ask (a) how programmers have used TouchDevelop so far and (b) how their productivity using TouchDevelop compares with their productivity using a traditional mobile phone development approach. I.e., we investigate the following research questions and corresponding hypotheses.

- RQ1: How large are TouchDevelop applications?
  - E1: Given the limited size of a phone screen, we expect that most TouchDevelop apps are small.
  - H1: Programmers use TouchDevelop to write small applications, which have few low LOC.
- RQ2: For a given task, how do TouchDevelop solutions differ from solutions obtained with a traditional mobile phone development approach?
  - E2: By assuming and hiding many facts that have to be stated explicitly in other languages, we expect that a TouchDevelop app is typically smaller than a corresponding app in a traditional approach.
  - H2: For the same task, a TouchDevelop solution has fewer LOC than a corresponding solution using Android SDK.
- RQ3: For a given set of tasks, how does programmer productivity using TouchDevelop compare with using a traditional mobile phone application approach?
  - E3: The simplicity of the language may make TouchDevelop programmers more productive. But the difficulty of programming on a tiny screen without a keyboard or mouse likely hampers productivity. So we expect TouchDevelop programmers to be less productive overall.
  - H3: Given the same amount of time and the same set of small programming tasks, TouchDevelop programmers finish fewer tasks than programmers using Android SDK.

IV. Design for Answering Research Questions

This section describes the experiments we designed to test the hypotheses of Section III.

A. Testing H1, Counting LOC

To test H1, we downloaded from the TouchDevelop cloud all apps that have ever been submitted. We removed all prior versions of an app published by the app's original author under the same app name. By doing that we only count the current version of each app. However we do not remove such apps if the prior and current app are by different authors or have different names.

H1 and H2 require us to count LOC. To make counting reproducible and comparable across techniques, we first normalized all TouchDevelop and Android apps and then counted their logical source statements (LSS) [1]. We did not count the content of configuration files such as the xml files each Android app uses to define layout, styling, etc.

B. Lab Experiment to Test H2, H3

To test H2 and H3, with IAB approval we recruited 27 graduate students of the CSE 5324 software engineering class taught by one of us. CSE 5324 has a team project. Each team builds its own Android app. We designed a lab experiment that took place during one class period (80 minutes), towards the end of the course. At this point, most subjects had some experience developing for Android.

We stressed that experiment participation will not influence grades. To simulate individual development, we allowed subjects to consult web sources but forbid other communication during the experiment, except with the instructor and teaching assistants (the first two authors).

We defined a set of 11 simple programming tasks. For the experiment, Microsoft Research loaned us 10 mobile phones, which allowed us to have subjects develop TouchDevelop apps on a phone. We randomly assigned 10 subjects to one phone each and the control-group of the remaining 17 subjects to a lab Windows PC each that had installed an Android IDE (Eclipse) and SDK. The Android IDE contains an interactive wizard that generates a working hello-world Android app. The experiment used TouchDevelop v2.4.0.0 beta and Android v1.6.

The first 10 minutes were spent reading and signing the informed consent form and the Microsoft phone loan agreement. 60 minutes were available to subjects to configure their phone or PC (each taking about 5 minutes) and working on our programming tasks. Then we asked subjects to email us their solutions (PC) or return the phones. 10 minutes were then used for a questionnaire. 2 subjects did not email us solutions, leaving us with 15 Android subjects.

Some 10 weeks before the experiment we held an in-class exercise with the same subjects in the same lab with the same 10 phones, using different tasks and a different subject assignment, to gain experience for our IAB submission. Twice, before this trial and before the experiment, we emailed subjects a link to the TouchDevelop website and to a short introductory video[1]. Otherwise we did not train subjects in TouchDevelop.

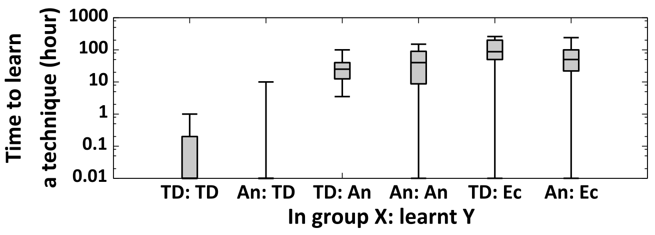

Figure 1. Before the experiment, subjects had spent more time learning Android than learning TouchDevelop (Answers to question "How many of the following have you done before this exercise? Hours spent learning Y (watch video, read website, api, etc.)"). I.e., all TouchDevelop subjects reported having learned TouchDevelop for one hour or less (TD: TD). Ec is Eclipse, a popular Java and Android IDE.

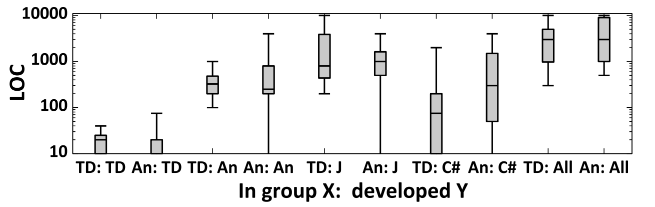

Figure 2. Before the experiment, subjects had written more Android than TouchDevelop LOC (Answers to question "How many of the following have you done before this exercise? Lines of Y code written"). I.e., all TouchDevelop subjects reported having written 40 or fewer TouchDevelop LOC (TD: TD). J is non-Android Java. All is code written in any language.

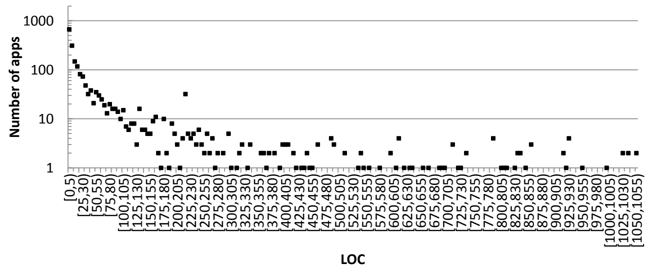

Figure 3. Size in LOC of the 2,081 TouchDevelop apps published in the TouchDevelop cloud as of 17 February 2012. Not shown are the two largest apps of 1,742 and 1,675 LOC. Each dot represents all apps in one bin of 5 LOC. E.g., the left-most dot represents the 674 apps from 0 to 4 LOC.

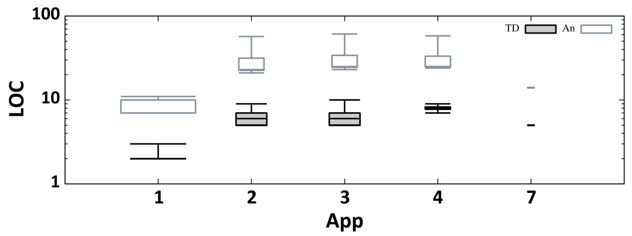

Figure 4. The size of the developed (correct) apps differs between Android and TouchDevelop. I.e., for each task, each correct Android app is larger than each of the correct TouchDevelop apps. The width of each box-and-whisker is proportional to the number of correct solutions we received. Not shown are tasks for which we did not receive any correct app.

C. Subjects' Prior Programming Experience for H2, H3

Figures 1 and 2 are box-and-whisker plots of our post-experiment questions that asked subjects to estimate their prior programming experience. Subjects were not given instructions on how to estimate.[2] In summary, Figures 1 and 2 show that our subjects had one or two orders of magnitude more experience with Android than with TouchDevelop, when measured as estimated time spent learning or as estimated LOC written for that approach.

V. Results

A. Cloud: TouchDevelop Apps Are Small (H1)

Figure 3 shows the 2,081 TouchDevelop apps that have been published in the TouchDevelop cloud, grouped by their respective size in LOC. 47% (987) are 9 LOC or less and 60% are 19 LOC or less. This confirms that, at least so far, the majority of TouchDevelop apps are (very) small.

B. Experiment: TouchDevelop-LOC $<$ Android-LOC (H2)

Figure 4 is a box-and-whisker plot of the size in LOC of all correct solutions submitted by the subjects. For each task, the average correct Android solution was about four times larger in LOC than the average correct TouchDevelop solution. This confirms our expectation that TouchDevelop apps are smaller than corresponding Android apps.

```
package edu.uta.cse.program1;
import android.app.Activity;
// .. two more import statements
public class P1Activity extends Activity
{
  @Override
  public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    TextView tv = new TextView(this);
    tv.setText("CSE 5324");
    setContentView(tv);
  }
}
```

Listing 1. Example task 1 solution in Android ("Hello World").

Listings 1 and 2 illustrate this difference in LOC. Both listings are example correct solutions of task 1 ("Any 'Hello World' program that prints 'CSE 5324' on the screen"). To work, the Android solution in Listing 1 must have a new class that (a) extends the Activity class and (b) overrides the onCreate method, which (c) must[3] call the overridden method. The onCreate method further should (d) create a text view, (e) set "CSE 5324" on the text view, and (f) add the

[1]http://channel9.msdn.com/Blogs/Peli/TouchDevelop-Getting-Started

[2]If a subject reported a range, we used the average of the lower and upper bounds, for a number with a plus or less-than ("100+", "<10") we ignored the operator, for an empty answer we used zero. A unit larger than hour we first converted to weeks (year = 52, semester = 12, month = 4) and then to hours (5), hours/week we assumed were given for one semester.

[3]Documented as "Derived classes must call through to the super class's implementation of this method. If they do not, an exception will be thrown." http://developer.android.com/reference/android/app/Activity.html

text view to the main screen. In contrast, the corresponding TouchDevelop solution in Listing 2 consists of a single step, posting “CSE 5324” to the default output stream (the wall).

```
action main() {
  'CSE 5324'→post_to_wall;
}
```

Listing 2. Example task 1 solution in TouchDevelop.

C. Experiment: TouchDevelop-Productivity > Android-Productivity (H3)

Table I
TOUCHDEVELOP SUBJECTS ON AVERAGE SUBMITTED CODE FOR MORE TASKS AND COMPLETED MORE TASKS THAN ANDROID SUBJECTS.

	Subjects	Some code	Correct
TouchDevelop (TD)	10	3.7	2.4
Android (An)	15	3.2	1.7

Table I shows the number of tasks for which subjects submitted some code and correct code. TouchDevelop subjects on average finished more tasks (correctly) than Android subjects. Similarly, as shown by Figure 5, the likelihood that a task is finished was greater for TouchDevelop than for Android subjects. Given the low familiarity of our subjects with TouchDevelop and their comparatively strong background in Android, this finding surprised us.

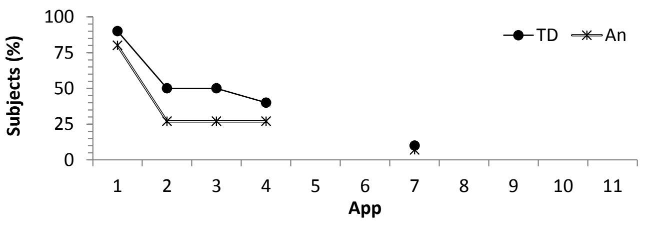

Figure 5. Using TouchDevelop made it more likely that a task will be finished correctly. For example, the percentage of subjects finishing task 3 was 50% for TouchDevelop but less than 30% for Android.

VI. Observations and Open Questions

One subject reported that the TouchDevelop IDE crashed (“When few lines of code were cut then TD crashed, had to retype”). From observing the subjects during the experiment we assume that this was not a problem for other subjects.

All Android subjects used the interactive Android IDE wizard to generate a simple hello-world app. This made it relatively easy to complete task 1. Other tasks however asked for interactive programs. Several subjects struggled to make the necessary modifications to the generated Android app.

We did not ask TouchDevelop subjects to publish apps during the experiment and nobody did. However, subjects also did not adapt any of the existing sample apps that are part of the TouchDevelop IDE. Maybe subjects did not judge it worthwhile to reuse an existing sample app because TouchDevelop apps are more compact, so there is less boilerplate code that can be reused?

More generally, we propose to investigate the following open questions, using appropriate user evaluation techniques.

- How do subjects reuse code between tasks?
- What code editing and other actions do subjects perform when working on their apps?
- How would our results change for tasks that require larger programs?
- How would our results change if subjects received more (or less) training in Android or TouchDevelop?

VII. Threats to Validity

Our subjects are not a random sample: Our findings may not generalize well to the intended TouchDevelop audience. I.e., we recruited subjects from a graduate course that clearly emphasized software engineering using Android and subjects self-selected this course. However, when selecting the course, subjects were not aware of this experiment.

Hands-off administration of tasks: We did not instruct subjects on how to work on the given tasks. Forcing subjects to work on certain tasks for a certain amount of time may produce results that are easier to compare. However, this level of control would prevent subjects from using a more natural programming style, in which subjects can switch between tasks and reuse solutions they found for a later task on an earlier task.

VIII. Conclusions

Our findings indicate that (a) programmers so far have written TouchDevelop apps that are small and (b) a TouchDevelop app is smaller than a corresponding traditional mobile phone app (i.e., using Android). Surprisingly, (c), for small tasks, even with very little to no training in TouchDevelop, a student programmer who has prior Java and Android programming experience is still more productive in writing TouchDevelop apps on a phone than writing Android apps in a traditional PC-based fashion. Our web site contains additional details: http://cseweb.uta.edu/~tuan/tdexp/

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1017305 and 1117369. We thank Microsoft Research Connections for providing Windows Phones for the duration of the semester.

References

[1] R. E. Park. Software Size Measurement: A Framework for Counting Source Statements. Technical report, Software Engineering Institute, Sept. 1992.

[2] N. Tillmann, M. Moskal, J. de Halleux, and M. Fähndrich. Touchdevelop: Programming cloud-connected mobile devices via touchscreen. In *Proc. 10th SIGPLAN ONWARD*, pages 49–60. ACM, Oct. 2011.