Expressing Uncertainty Using the maybe System

Geoffrey Challen (PI), Oliver Kennedy, Lukasz Ziarek (co-PIs)
University at Buffalo, Department of Computer Science and Engineering
338 Davis Hall, Buffalo, NY 14260-2500
{challen,okennedy,lziarek}@buffalo.edu :: http://blue.cse.buffalo.edu/projects/maybe/
Google Sponsor: Matt Welsh; Google Contacts: Jeremy Kahn, Sudip Roy.

Abstract: Android apps currently run on 18,796 distinct devices with different hardware capabilities. Those devices connect to multiple data networks with 100-fold variations in speed and latency. Android users are elderly adults and young children, have highly-variable mobility patterns and install different mixtures of the 1.5 million apps published on the Google Play Store. All of this diversity creates uncertainty for developers, who may be unsure how to adapt their app to achieve performance, correctness, and usability on any device, at all times, for anyone, anywhere. Our new system, `maybe`, addresses this challenge by allowing programmers to express *development-time uncertainty*. A new language construct and online infrastructure allow programmers to indicate where they are uncertain, what legitimate options are available, and what constitutes success or failure. Post-deployment testing and machine learning are then used to resolve the uncertainty by crafting effective data-driven adaptation strategies.

1 — Problem: Development-Time Uncertainty Caused by Mobile Environment Diversity

Today, online marketplaces allow developers to easily distribute apps to billions of mobile devices. However, this capability exposes developers to significant differences between the devices themselves (speed, storage capacity, available sensors), the networks they connect to (availability, performance, energy consumption) and their users (app usage, mobility patterns, charging habits). As a result, it may be impossible for developers to anticipate all of the different environments in which their app will be used. This produces *development-time uncertainty*: when a developer knows multiple ways to solve a problem but, due to uncertainty about the runtime environment, is not sure which one to use.

Today's mobile app programming languages force programmers to address development-time uncertainty in one of several unsatisfactory ways, two of which are shown in Figure 1. First, they can **guess at a single approach** and hope that it is appropriate for all environments. But if they were right to be uncertain then no single solution will work well everywhere on all devices for all users. Second, they can **attempt to adapt to the runtime environment** themselves, but this is both time consuming and hard to get right. Even if they incorporate existing approaches to common adaptation problems, app-specific adaptation opportunities will be missed. Finally, they can use **app A/B testing**, but this approach still focuses on identifying one winner—an increasingly inappropriate simplification given the diverse environments apps encounter.

```
// Hopefully this works for everyone.
int timerRateS = 60;

// Hopefully this is the right way to adapt
if (!plugged && batteryLevel < 10) {
  // Try to save energy
} else {
  // Don't try to save energy
}
```

Figure 1: Today developers must eliminate uncertainty.

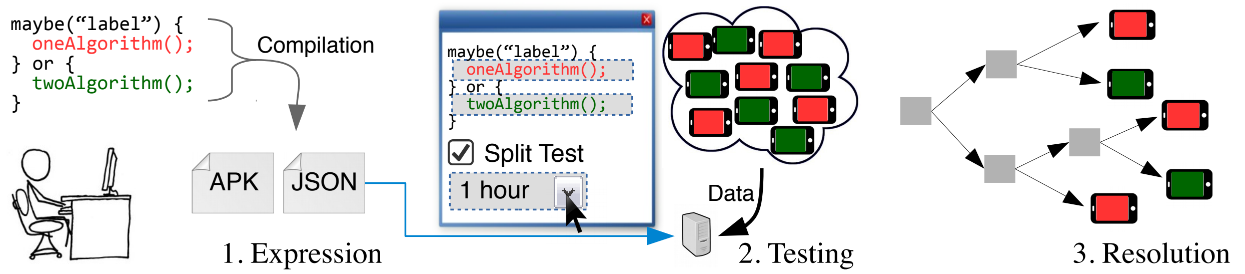
To address this challenge we take a novel approach motivated by the observation that **development-time uncertainty identifies opportunities for runtime adaptation.** We are building a system called `maybe` that allows developers to harness uncertainty to enable adaptation through pre- and post-deployment testing, data collection, and machine learning. `maybe` puts development-time uncertainty to work and frees developers from average-case guesses or brittle pre-deployment attempts at adaptation.

Unlike current approaches, `maybe` does not require developers to hide or address uncertainty. Instead, they use a new language keyword to *express* uncertainty by identifying multiple valid code paths or variable values. Figure 2 shows examples of the `maybe` statement expressing the uncertainty lurking in the code from Figure 1. `maybe` also does not require developers to implement adaptation decisions at development time. Instead, they only have to indicate what constitutes good or bad outcomes resulting from the uncertainty they have exposed. After post-deployment testing, developers can experiment with a library of existing adaptation policies, write their own, utilize machine learning algorithms to automatically construct policies, or exercise manual control—all based on post-deployment data from real devices.

```
// I know multiple rates that could work
int timerRateS = maybe 30, 60, 120, 600;

// I know multiple valid approaches
maybe {
  // Might save energy
} or {
  // Might not save energy
}
```

Figure 2: `maybe` allows it to be expressed.



```
maybe("label") {
  oneAlgorithm();
} or {
  twoAlgorithm();
}
```

Figure 3: maybe **Overview.** The process of expressing and resolving development-time uncertainty is shown. Here, post-deployment split testing is used to gather data and decision tree learning is used to develop an adaptation strategy.

2 — SOLUTION: The maybe System For Expressing and Resolving Uncertainty

A typical maybe workflow proceeds in the three stages illustrated in Figure 3.

Expression. First, the developer uses the maybe keyword, a new language construct expressing uncertainty. Figure 2 shows maybe applied to setting variable values and controlling code paths. In each case maybe associates a set of *alternatives* with a *label*. During compilation, maybe generates metadata about the app's maybe statements, which allows the developer to use maybe's web interface to control the system.

Testing. Next, the developer provides data, both to allow maybe to select the best alternative (scoring) and as inputs to the adaptation process (logging). The score and log functions provided by the maybe API allow developers to add scoring and logging data to labeled maybe statements anywhere in their code. Developers must use score to provide numeric or boolean scores evaluating the result of each maybe statement.

The maybe library automatically records many possible common adaptation inputs, such as device model and configuration, coarse-grained location, and battery level and connectivity changes, both periodically and every time a maybe decision is made. This frees developers from writing boilerplate code to record common information, while log still allows them to add additional app-specific adaptation inputs for a particular statement. Before collection, logged data is automatically annotated with a timestamp, a device identifier, and the alternative that was last selected before the logging call was made.

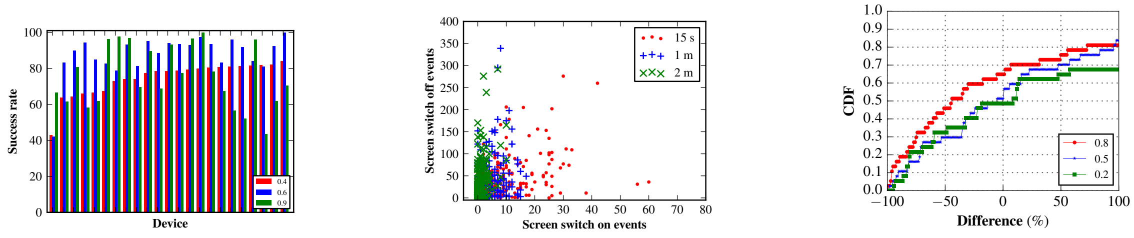
maybe currently supports post-deployment split testing to evaluate alternatives, although we are also exploring other techniques. Developers use a web interface to choose statements to test, set how often devices should switch alternatives, view testing statistics and retrieve testing data. During testing, maybe uploads logging and scoring data while the device is charging according to a configurable network usage policy.

Resolution. maybe's adaptation goal is to use logging information collected during testing to predict which alternative produces the best score in a particular environment. The first step is to assign a global utility or ranking to each score. maybe streamlines this process by defining defaults for commonly-used scores such as time and energy (smaller is better), and considers true and false values good and bad, respectively. If score is called multiple times during a single decision, contributions are averaged by default, but developers can override the defaults, explain unfamiliar scores, and specify different scoring aggregators.

Testing may reveal outcomes requiring adaptation strategies of varying complexity:

1. **No winner or single winner:** If no differences exist between alternatives, or one works best in all tested environments, maybe allows the developer to select a single option that is used by all devices.
2. **Static decisions:** If the best alternative depends on some slowly changing properties of the device or user, maybe will use that alternative on that device while also trying to learn how to predict the best alternative based on known attributes for new devices. Figure 3 shows an example of a decision tree generated by testing data which can be used to select alternatives for other devices.
3. **Dynamic decisions:** If the best alternative changes rapidly based on the device's environment, then adaptation needs to take place on the device rather than the server. We are also exploring ways to utilize machine learning to automate on-device adaptation. At this point developers are also free to implement, deploy, and test their own approaches, with data gathered by maybe allowing developers to evaluate strategies in ways impossible impossible before deployment.

2.1 — Prototype Implementation and Evaluation: We have implemented a maybe prototype supporting adaptation within the Android platform. It uses a rewriter which converts maybe statements to case statements and an Android service providing the maybe API. Currently only Java is supported, but it should be straightforward to extend to other languages. The maybe server uses Meteor to provide an interactive web interface and provides a REST API for retrieving maybe decisions and uploading testing data.



(a) (b) (c)

Figure 4: **Exposing Uncertainty in the Android Platform.** Figures are explained in more detail below.

We conducted a preliminary evaluation of `maybe` on the PHONELAB smartphone platform testbed. Our changes were distributed as an over-the-air (OTA) update to 196 PHONELAB participants and we ran a four-day experiment investigating uncertainty in the Android platform. By analyzing the Android source code, we identified several places where constant values could possibly be adapted on a per-user basis:

1. Figure 4a shows per-user success rates for the pattern-based screen unlock. The results demonstrate that different levels of sensitivity work best for different users. This value can be adapted to avoid frustrating users with different finger sizes or motor skills.
2. Figure 4b shows per-user counts of manual screen locks and unlocks for different timeout settings. Minimizing the number of manual locks and unlocks produces different timeouts for different users.
3. Figure 4c shows how the link avoidance threshold used by Android affects the amount of mobile data usage. Given the prevalence of fast 4G mobile data networks, this value can be adapted on to allow the user to avoid as many bad Wifi links as possible while not exceeding their mobile data limit.

3 — PLAN: Implement Automated `maybe` Adaptation for Android Apps

Our work on `maybe` is just beginning and currently unsupported by external funding agencies. Support from Google will be used to support two tasks. First, we will release `maybe` for Android apps including our Java rewriter and an updated web interface and app-level service. We will deploy our backend to Google Compute Engine to provide consistently performant, scalable, highly secure and reliable service. Releasing `maybe` will allow us to gather data about how developers utilize this new language capability.

	Description	Price ($)
RA	Salary	22,000
	Tuition	15,156
Travel		6000
	Total:	$43,156

Table 1: **Budget.**

Second, we will explore how to apply supervised and unsupervised machine learning techniques to automatically design static and dynamic adaptation strategies for `maybe` statements. We will continue to use PHONELAB to gather data on uncertainty within Android itself while using our public app-level `maybe` service to gather data on app adaptation. We will also use `maybe` to expose uncertainty in some of the apps that are included in the AOSP or that have sources available online (such as Chromium) and deploy the modified versions on PHONELAB. Our goal will be to understand what machine learning techniques work best and how accurately we can select correct alternatives for untested devices, while also determining how much developer guidance improves the outcome.

3.1 — Data Policy: Results will be presented at appropriate conferences and data sets posted online.

3.2 — Budget: Our budget includes one year of PhD support and a small amount of travel funding.

4 — Related Work

Systems such as EnFrame [3] reflect growing interest in managing uncertainty at the language level, although EnFrame focuses on uncertain data rather than runtime adaptation. Aspect oriented programming (AOP) [2] aims to increase modularity through the separation of cross-cutting concerns, rather than enabling adaptation by expressing uncertainty. Like AOP, Context oriented programming (COP) [1], aims to simplify the expression of computation in a more modular fashion, but focuses on adapting behavior dynamically to the current execution context and still requires developer certainty.

REFERENCES

[1] Pascal Costanza and Robert Hirschfeld. Language constructs for context-oriented programming: An overview of contextl. In *Proceedings of the 2005 Symposium on Dynamic Languages*, DLS '05, pages 1–10, New York, NY, USA, 2005. ACM.

[2] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP*. SpringerVerlag, 1997.

[3] Sebastiaan J van Schaik, Dan Olteanu, and Robert Fink. Enframe: A platform for processing probabilistic data. *arXiv preprint arXiv:1309.0373*, 2013.

Results from Prior Google Support

Geoffrey Challen was a co-investigator on a project entitled "PhoneLab: A Participatory Smartphone Cloud Testbed" which received $60,994 from the Google University Award program in 2011.

We used this funding to purchase 40 Nexus S 4G smartphone from Sprint and equip them with voice and data service. In addition, two students were supported during the 2011–2012 academic year. Equipment and student support were used to design a small prototype of the large open-access smartphone PhoneLab testbed that the University at Buffalo will open in Fall 2013. In addition to the core infrastructure work, a series of research projects were initiated last fall as part of a graduate-level systems research class that focused on smartphone research and development, and these early projects provided valuable insights into the features that the testbed would need to support to accelerate smartphone research.

With support from Google, Sprint, and many of our research colleagues at other institutions, the PhoneLab project recently received a 3 year 1.3 million dollar Computing Research Infrastructure (CRI) grant from the National Science Foundation. This funding allowed us to run PhoneLab in beta mode with 191 phones in 2012, open it to the public with 288 phones in 2013, and grow even larger in 2014. A stated goal when we applied and received money previously from Google to support PhoneLab was to eventually scale it to a size that the University Awards program would have been unable to support, and we are pleased that we were able to use the initial seed funding from Google to do so.