CAREER: Harnessing Implementation Flexibility to Enable Runtime Adaptation

NSF Proposal
20 Jul 2016

PI: Geoffrey Challen (University at Buffalo)

For today's apps, achieving good performance on billions of smartphones around the world requires adaptation. Apps must adapt to a wide variety of device capabilities; a constantly changing network environment; limited energy resources; fluctuating availability and accuracy of sensor information; different patterns of interaction; and a diverse community of users. These environmental differences can cause a successful approach on one device or user to fail for another device or user. For example, different implementations of the same algorithm can produce different energy-performance tradeoffs due to variations in hardware capabilities or user inputs, or different caching strategies can outperform each other due to variations in network performance or user mobility. And as increasing variation makes adaptation more important, it also makes it more difficult. Today, manually designing effective adaptation policies that anticipate all potential sources of variation challenges even talented developers. And no approaches exist to simplify the process of creating adaptive and effective apps.

This proposal investigates a novel approach to building adaptive mobile systems by exposing and harnessing implementation flexibility. Implementation flexibility refers to cases when multiple correct ways to accomplish something exist: such as multiple algorithm implementations, different intervals for repeating an operation, or different times at which a task can be performed. Currently, developers must both develop these adaptation mechanisms and formulate adaptation policies. In contrast, exposing implementation flexibility allows developers to write adaptable apps without providing adaptation policies. Instead, pre- and post-deployment testing combined with machine learning and continued developer oversight develop automated adaptation policies that harness implementation flexibility to adapt to changing conditions. All developers have to do is expose flexibility. The proposed system finishes the job of creating and maintaining an adaptive and effective app.

Intellectual Merit: The project investigates a transformative new way of building adaptive apps and systems. Realizing this approach requires addressing multiple challenges. First, developers must be able to separate the inflexible portions of their code required for correctness from the flexible portions that can be adapted. Next, representative automated testing is required to rapidly and efficiently investigate how to harness uncertainty developers have exposed. Once testing is completed, flexibility must be harnessed by machine learning algorithms to build effective static and dynamic adaptation policies. The proposal also investigates how to mine different implementation approaches from single developers, small groups, and crowds, providing flexibility that can be harnessed for adaptation. A novel modern take on self-modifying code uses code annotations and modifications to express automated adaptation policies to developers. Finally, the project explores the implications of anticipated discoveries about adaptation on the design of next-generation language features that promote flexibility to a first-class construct.

Broader Impacts: The integrated educational plan also harnesses implementation flexibility but uses it to develop courses that adapt to differences between students. Beginning with the creation of a collaborative concept library of modules for teaching the basics of the internet, a new system will allow educators to express implementation flexibility in aspects of course implementation such as the ordering of concepts and the ordering of explanations for a single concept. The effectiveness of the adaptation will be assessed using student outcomes. By identifying the role of implementation flexibility in two very different settings, the project will establish a foundation for future efforts that harness flexibility to build other adaptive systems, while having an immediate impact on two important societal challenges: building mobile apps and teaching students online.

Adaptation has long been considered a key to effective mobile apps and systems [14, 34, 42, 43, 48, 55, 60, 61]. As smartphones emerge as the dominant mobile computing platform, adaptation is more important and challenging than ever. Today's mobile apps continue to face traditional adaptation challenges: rapidly changing conditions caused by mobility, and resource constraints inherent to battery-powered devices. But for an app to perform well on billions of smartphones, it must also adapt to the significant differences between devices, users, and environments. Realizing the next generation of transformative mobile apps requires novel techniques to make adaptation easier and more effective. **I propose to explore a promising new approach to enabling adaptation that exposes and harnesses implementation flexibility.**

Consider the adaptation challenge faced by a developer writing an Android app. Android is today's most widely deployed smartphone platform powering over 1 billion devices worldwide [1]. Multiple versions of Android support tens of thousands of distinct devices. Each has varying numbers of cores, amounts of memory and storage, and mixtures of sensors and other features [2]. 100-fold variations in worldwide mobile network speed and latency have been measured [39]. While 10% of smartphone users spend 40% of their time disconnected, another 10% spend 98% of their time with fast connections [79]. Smartphone battery capacities vary and they are used where power is cheap and reliable and where it is expensive and unreliable. Users have highly variable mobility patterns [79] and use different mixtures of millions of available apps [3].

Today developers have two options for confronting this diversity. The easiest is to implement a "one-size-fits-all" solution and hope that it works in all environments. Given the amount of diversity cataloged above, this is unlikely. Mobile app split testing [6, 11, 13] improves but still converges on "one-size-fits-all" solutions. A second approach is to attempt adaptation. Unfortunately this is time consuming and difficult. Multiple entire research projects have explored adaptation problems such as energy-efficient but accurate location tracking [50, 63, 76, 80, 82], choosing between available networks [28, 41, 70, 74], and determining when to perform cloud offloading [26, 35, 37, 46]. While libraries may handle some common adaptation problems, there are still app-specific adaptation opportunities that libraries miss. Developers are unlikely have the time, skill, and resources required to create effective adaptation policies.

```
if (plugged == false && batteryLevel < 10) {
  updateRateSec = 120;
} else {
  updateRateSec = 60;
}
```

(a)

```
int updateRateSec = maybe.choose(60, 120);
```

(b)

Example 1: **Two Adaptation Approaches**

Note that adaptation currently requires developers to do two things: establish adaptation *mechanisms* and formulate adaptation *policies*. Mechanisms are ways that the app can change. Policies determine how use mechanisms to respond to changes in the environment. Example 1a shows an example of intermingled adaptation mechanism (timer rate) and policy (decrease the rate when the battery falls below 10%). Our key observation is that creating adaptation policies is much harder than exposing adaptation mechanisms. In Example 1a, the developer knows and has expressed the mechanism: the app can safely update at multiple rates. But what policy determines the rate to use? If the app is on a device with a small battery connected to a slow network, then it should always use the slower rate. But if the user charges the battery frequently and connects to energy-efficient networks, the faster rate may always be appropriate. Development is a good time to expose flexibility but a bad time to try to formulate adaptation policies.

Our novel approach begins by encouraging developers to expose *implementation flexibility*. Implementation flexibility refers to cases when there are multiple correct ways to accomplish something. Examples include equivalent algorithms, varying timer intervals, or different times at which a task can be performed. Exposing implementation flexibility allows developers to separate their app into two parts. The inflexible portion is required for correctness and might correspond to a provided suite of unit and integration tests. The flexible portion is harnessed as a mechanism to enable adaptation. Example 1b shows how one approach to exposing implementation flexibility. A new `maybe` library allows the developer to expose the same adaptation mechanism (the flexible timer rate) without providing an adaptation policy. The valid alternatives are expressed. Flexibility is exposed and can be harnessed. But no decision-making logic needs to be provided. Removing the development-time requirement of providing adaptation policies makes it more likely that developers will expose flexibility and avoid "one-size-fits-all" decisions.

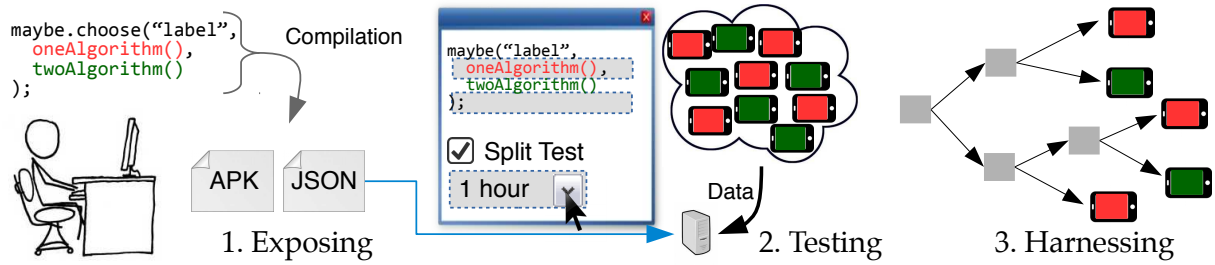

Figure 1: **`maybe` System Workflow.** The process of exposing, testing, and harnessing implementation flexibility is shown. In this example, (1) a `maybe` language keyword is used to expose implementation flexibility, (2) post-deployment split testing is used to gather data and (3) decision tree learning is used to automatically generate an adaptation policy. The proposed project also investigates other approaches at each stage.

More importantly, separating flexibility (that serves as an adaptation mechanism) from policy enables new approaches to policy formulation. We propose to explore approaches that use testing results and machine learning algorithms to automate adaptation policy design. Pre- and post-deployment testing is used to gather data about the effect of exposed implementation flexibility on app performance. Both common metrics such as speed and energy consumption and app-specific metrics provided by developers are measured. Combined with environmental information, testing data allows machine learning algorithms to automatically design adaptation algorithms. Developers may also use the dataset to test and evaluate their own custom policies, but with the benefit of data to evaluate policy effectiveness.

Separating adaptation policy from mechanism allows policy formulation to be moved to a better point in the development cycle. Before deployment it is hard to get policies right. After deployment and testing, with data to generate and evaluate policies, it is much easier. The approach also frees adaptation policy formulation from developer biases and errors. Testing may reveal that exposed implementation flexibility is not useful, or does not have the effect the developer intended or anticipated. These problems will not stop the system from formulating effective adaptation policies as long as useful implementation flexibility exists.

Both the proposed research and integrated education plan investigate how to enable adaptation by harnessing exposing implementation flexibility. Section 1 presents the intellectual merit components, Section 2 presents the broader impacts, and Section 3 presents our qualifications and results from prior support.

1 — Intellectual Merit: Android App Adaptation Using the `maybe` System

The proposed research is organized into five thrusts each occurring during one year. Earlier thrusts address more fundamental challenges and work on them may continue into future years if needed. Most thrusts begins with a vignette and all end with an evaluation plan. Because `maybe` has already received a year of support from Google [22], we begin by describing what will happen next year as Thrust 0.

1.0 — Thrust 0: Exposing Flexibility, Static Adaptation, Synchronous Testing (2016–2017)

> *Alice has previously used the SixPack testing framework to AB test components of her Android app. In one case, she tested two different ways of ordering items on the home page, but the results were inconclusive. But after small changes to her app `maybe` is able to build an automatic adaptation policy for the home page layout based on screen size.*

One year of support from a Google Research Award [22] will allow us to begin work on `maybe` in 2016–2017. Efforts during Y0 will provide a solid foundation for the additional activities funded by this proposal.

1.0.1 — `maybe` overview. We begin by allowing Android app developers to expose implementation flexibility. Example 2 shows examples of how implementation flexibility can be exposed when setting variables (a) and choosing code paths (b). For each flexible *statement*, `maybe` allows developers to specify multiple *alternatives* and a *label* used to identify statements when associating logging or scoring information. Flexible variables are only adapted when assigned or reassigned, ensuring that their values do not change unexpectedly. Flexible code paths are reevaluated when they are reached.

Harnessing exposed implementation flexibility to enable adaptation requires two additional pieces of information. First, `maybe` needs to know any conditions that might effect the decision. These serve as inputs to policy formulation and to policies themselves. We automatically record many possible common conditions, such as device model, Android version, battery level, and connectivity changes. This information is recorded periodically and when `maybe` encounters and chooses alternatives. Cases where adaptation inputs may be app- or statement-specific are handled through the logging interface. Example 2b demonstrates how the length of the algorithm's argument is provided as a potential adaptation input. Note that `maybe` runs in the same context as the app, preventing privilege escalation. Privacy-sensitive inputs such as location can only be logged by apps that already have the appropriate permissions.

```
retryInterval = maybe.range("interval", 1, 16);
policy = maybe.choose("policy", "qual", "perf");
```

(a) **Flexible Variable Values**

```
// Statement-specific logging and scoring
maybe.log("alg", input.length());
ret = maybe.choose("alg", one(input), two(input));
maybe.score("alg", ret);
```

(b) **Flexible Code Paths**

Example 2: **Exposing Flexibility**

Second, `maybe` needs to understand the app's adaptation goals. Again, `maybe` automatically measures common goals such as low latency or energy consumption, periodically and when decisions are made. Cases where goals are app- or statement-specific are handled through the scoring interface. Example 2b shows how the algorithm's return value is provided as an adaptation goal. Unlike time and energy, scores that the `maybe` system already understands, the developer will have to use the `maybe` web interface to explain how to interpret the `"ret"` score. Note that `maybe` statement labels allow the developer to provide logging and scoring information wherever appropriate—not just near the statement itself. Logging and scoring information is uploaded periodically to a backend server when the device is charging and the battery is full.

1.0.2 — Y0 activities. The first goal for Y0 is to extend SixPack [13], an existing AB testing framework, to support static adaptation through synchronous post-deployment testing. We chose SixPack because it is open source and supports multiple languages. SixPack already includes many features required by `maybe`, including logging, random alternative testing, and a web interface. SixPack also allows us to leverage an existing user base. Android projects using SixPack will be able to try `maybe` with only minor code changes.

Several required changes to SixPack will be completed in Y0. We will add support for collecting device attributes using existing SixPack logging support. We will also need to broaden the idea of conversion typical to AB testing frameworks to incorporate `maybe` scoring. Changes to the SixPack backend will be required to utilize collected device attributes for automatic alternative selection. And we will extend the web interface to provide developers with control and visibility into automatic adaptation. All alternative testing will be performed online on normal app code paths. We explore asynchronous and pre-deployment testing as Thrust 1 in Y1 (§1.1). All adaptation will be based on slowly changing or unchanging features such as the device attributes or large geographic region. We explore dynamic adaptation as Thrust 2 in Y2 (§1.2).

The second goal for Y0 is to understand use of AB testing in the wild. We have obtained cooperation from GitHub to perform a source-based study of projects using existing AB testing frameworks, including SixPack, Optimizely [11], and Apptimize [6]. We are interested how often AB testing is performed, how long tests are run, and what app features are tested. Results from this study will aid our outreach to the developer community. It will also help us target active GitHub projects to approach as potential `maybe` users.

1.0.3 — Outputs and evaluation plan. The primary output of Y0 will be a fork of the SixPack AB testing framework supporting static adaptation based on post-deployment testing. We will deploy a SixPack backend with `maybe` support for developers to use. We will also begin relationships with several Android open-source projects using SixPack AB testing that are interested in automatic adaptation. We will evaluate our efforts through library test cases, feedback from developers, and using test apps.

1.1 — Thrust 1: Automated Alternative Testing (2017–2018)

Alice enjoys that `maybe` automates the process of adapting to users of her app. Previously, however, the process was quite slow and had the potential to annoy her users. However, with the ability to perform automated pre-deployment testing Alice is able to expose much more flexibility in her app. She can test statements faster without worrying about irritating users. Eventually this capability helps her build a much more adaptive and effective app.

User-facing testing used in the Y0 version of `maybe` suffers from several limitations. Because it can proceed only as fast as users encounter alternatives, it is slow. Slow testing causes long delays before enough data is available for adaptation policies to be formulated. As a result, developers are reluctant to expose all the flexibility in their app because more flexibility means slower time to adaptation. Slow testing makes testing corner cases even slower. `maybe` must wait for a user to encounter each possible environment to test all alternatives. If certain situations occur rarely, such as transitions between data networks, this slows down testing further. Finally, because user-facing testing may expose users to poor alternatives, it is disruptive.

Asynchronous offline testing of alternatives would make `maybe` much more powerful. Testing and adaptation policy generation could be performed before changes are deployed, tightening the development cycle and reducing user irritation. Testing could proceed much more quickly, allowing more alternatives to be tested and encouraging developers to expose more flexibilty. Automated testing could also easily test permutations of alternatives from multiple statements. The ability to manipulate the environment during offline testing simplifies testing across many environments, ensuring robust adaptation policies.

1.1.1 — Representative automated testing. The research challenge in Y1 is enabling representative automated app testing. Representative app testing differs from monkey testing, which does not aim to reproduce actual user behavior. Instead, monkey testing attempts to perform an exhaustive search of all app code paths to attempt to uncover undesirable behavior such as crashes. The simplest form of monkey testing provides random inputs to attempt to eventually exercise all app functionality. More sophisticated approaches such as Caiipa [49] and VanarSena [66] improve testing speed but do not address realism.

Because the goal of `maybe` is to use testing to build adaptation policies for users that are not monkeys, adapting to monkey behavior is inappropriate. Simply recording and replaying user inputs is also insufficient. Because adaptation decisions and environment differences will cause timing changes, blindly replayed traces will quickly lose synchronization with the current app state. For example, if the trace contains a click at a certain location, but the targeted interface element is not yet visible due to timing differences between the trace and testing environment, the click will miss and not produce the desired action. At that point the test is out of synchronization and will not reproduce the intended behavior.

1.1.2 — Automated app state inference. We propose to explore a novel approach to recording and replaying testing traces that uses automated inference of app states. We consider each app to consist of a set of user-visible states. Transitions between states result from one or several actions. User visibility is both important and a useful constraint because we are attempting to collect and reproduce user interaction. For example, a mail app might consist of a state showing the inbox or a particular folder, a state showing a specific message, a state showing a composition window, and multiple menu states allowing users to adjust app settings. App states and the overall state diagram may be parameterized by content, and so different users may have different numbers of certain app states (folders or messages). One user interaction with the mail app consists of repeatedly opening the folder view and then possible one or more messages.

The simplest way to identify such states is for the app developer to annotate their source code. To provide ground truth for automated approaches, in Y1 we will augment the `maybe` Android library to record developer-provided state annotations. However, relying on app developers to correctly provide state annotations for the millions of existing Android apps is unrealistic. Instead, our goal is to automate state classification using information that can be observed for unmodified apps. One potential source of useful state information is the Android view tree, which contains the app's user interface elements. Changes to the content and structure of the view tree may coincide with app state transitions. Other observable sources of potentially useful information include interactions between the app and the Android platform, libraries such as `bionic`, and with the graphics card via OpenGL commands.

By instrumenting apps that contain developer-provided annotations, we will attempt to build tools that can reproduce the annotations using only system observables. Challenges that we expect include differentiating between parts of the view tree that represent state (i.e, the user is viewing an email) from the parts that contain content (i.e., the contents of the email the user is viewing). We also must ensure that data collection is low overhead with the goal of performing it continuously and unobtrusively. Handling inputs to search dialogs and other text boxes in a way that preserves user privacy without compromising reproducibility is another concern. One approach we will investigate is to blind search inputs by replacing them with search terms drawn from publicly available distributions appropriate for the tested app.

Accurate state identification allows us to augment input-based record and replay traces with state information. Because the testing harness knows what state it is transitioning to and can determine when it has arrived, it can deliver inputs at the appropriate time. It can also determine when replay has diverged from the recorded series of states because an error has occurred or the testing app's behavior has changed.

The overall goal of state-based record and replay is to test `maybe` alternatives before deployment. State traces collected from many users of the app will be used as-is or combined to produce a meta-user that reproduces many common user behaviors. The output will be used to drive a simulator or a dedicated testing device. During repeated traces different `maybe` alternatives and statements can be tested. The testing device's environment can also be manipulated to investigate the effect of changing conditions. An important challenge will be rectifying the initial conditions of the testing device with collected traces. For example, if the mail app under test does not have any messages in its inbox it cannot open one. If it does not have enough messages or folders it cannot switch between. These problems should be detected during testing due to differences between the testing state and the recorded state. Addressing this issue may be as simple as a one-time process of building initialization scripts for tested apps. But it may require more research if initial conditions vary in certain ways between different app users.

1.1.3 — Other applications of state inference. Although designed for pre-deployment testing, this approach can also be used to test alternatives on each user's own device. For example, when plugged in and fully-charged, instead of sleeping the device could use traces collected earlier the same day to perform automated testing. In this use case it is important to ensure that testing does not lead to any externally-visible actions—for example, sending an email. We also anticipate that automated state tracking will benefit projects other than `maybe` adaptation. Previous attempts at parallel cloud-device execution have struggled to maintain state between devices Our novel state tracking approach could provide a solution. Investigating transitions within each apps state diagram could also help detect and diagnose poor user interface design. For example, if common user actions require transitions between large numbers of states, then the app may have an overly-complicated menu design. Finally, automated state tracking will aid studies attempting to understand user qualify of experience (QoE). When there are long delays in transitions between states, finding out why may help hide user-perceived waiting and improve overall app QoE.

1.1.4 — Outputs and evaluation plan. Outputs from Y1 will include extensions to the `maybe` library to do manual state annotations, instrumentation required in the Android SDK to support automated state annotations, tools to collect and process state-augmented record and replay traces, and a testing harness that can rerun state-augment traces and drive either a simulator or a dedicated device. We will evaluate the accuracy of state labeling by comparing automatically-generated labels with manual labels for simple apps and apps used by developers using the `maybe` library. We will also close the loop by comparing automated `maybe` testing results to the results obtained from Y0 that are based on post-deployment testing.

1.2 — Thrust 2: Dynamic Adaptation Policies (2018–2019)

Automated test-driven adaptation has allowed Alice to build a more adaptive and effective app. However, it's adaptation policies can only handle static attributes. Essentially, `maybe` allows Alice to build customized version of her app as needed for different devices, users, and environments. But it does not allow her apps to adapt to changing conditions such as network quality. Support for dynamic adaptation allows Alice's app to handle even more adaptation scenarios and improves its existing adaptation policies. Eventually this capability helps her build a much more adaptive and effective app.

In Y0 `maybe` only built static adaptation policies that characterized unchanging or slowly-changing device features. By their very nature static adaptation decisions do not change once deployed to the device, and adaptation decisions can be made entirely offline. But static adaptation decisions cannot respond to many of the changing conditions that mobile smartphones experience. In Y2 we address this limitation and focus on building dynamic adaptation policies that can handle changing conditions.

Several challenges emerge at this stage. Machine learning approaches must be explored that can build decision algorithms, not just make decisions. Those decisions must be distributed to devices and reevaluated when appropriate. Dynamic adaptation is more likely to produce tradeoffs between desirable outcomes that require additional guidance to resolve. Finally, a related challenge explored in Y2 is determining when existing adaptation policies must be revisited.

1.2.1 — Possible adaptation outcomes. To begin, we review the possible `maybe` adaptation outcomes. Testing may reveal that there are no meaningful differences between the alternatives or that one always wins. In these cases `maybe` selects a single alternative for all devices. `maybe` inherits this standard AB testing functionality from SixPack.

Static adaptation policies that depends only on unchanging or slowly changing features are supported after Y0. In this case `maybe` runs a classifier on the backend and distributes static decisions to devices.

If the adaptation policy produced by testing depends on rapidly changing inputs, such as the system load, network utilization, or battery level, then it may need to be rerun each time the flexible statement is executed. In these dynamic cases running adaptation policies on the server introduces too much overhead and delay. Instead, `maybe` must serialize the adaptation policy and transmit it to the device for execution.

Finally, it may prove impossible to automatically create an effective adaptation policy for a particular statement. Several options exist to handle this case. First, developers can manually select an alternative, with guidance provided by the testing results. Second, developers may able to provide additional logging inputs allowing an adaptation policy to be successfully generated after additional testing. A third option is to perform per-device alternative testing over all device configurations using the automated testing framework developed in Y1. Results can either be used to choose the single alternative for that configuration or autogenerate a per-device dynamic adaptation policy. A fourth option is for developers to create a custom policy offline using the testing dataset to evaluate their its effectiveness. Finally, the statement can be converted into app setting. Because settings are confusing for users, we consider this a last resort.

1.2.2 — Creating dynamic adaptation policies. We propose to automate the process of creating dynamic adaptation policies using machine learning. When generating dynamic policies for multiple devices, machine learning has the potential to make the formerly complicated process of developing an adaptation policy completely painless. And if no suitable adaptation policy exists spanning multiple devices, creating device-specific adaptation policies for each device is also feasible because the process is automated. We propose to investigate standard supervised learning approaches to automated adaptation policy creation, including decision tree learning and support vector machines.

If an effective adaptation policy cannot be automatically generated, developers still have access to the testing dataset and can use it to develop their own adaptation policy offline. In some cases they may want to start from scratch, and in others cases the auto-generated policy may serve as a starting point that is improved by developer oversight. We also plan to compare the two approaches, both in terms of the time required and the structure and effectiveness of the resulting adaptation policy. Studying the structure of autogenerated adaptation policies may also yield new discoveries about adaptation in mobile systems, and uncover surprising differences between devices, users, or versions of Android.

1.2.3 — Handling tradeoffs. `maybe` allows developers to assign multiple scores to a single statement. In this case, `maybe` considers an alternative better than another if and only if it improves all scores. Other cases represent tradeoffs and are flagged for the developer to examine. They may choose to drop or combine scores to produce a single result and thereby eliminate the ambiguity.

In other cases trading off multiple conflicting scores may require a second round of decision making. As an example, when should an app sacrifice performance or quality to save energy? When trading off between two common desirable outcomes, we propose to investigate a multi-tier approach allowing `maybe` to delegate choices producing such tradeoffs. In the example above, the statement's adaptation policy may delegate the tradeoff to a global device policy. It may decide to only sacrifice performance to save energy when the energy savings are considerable, or when the battery is below a certain threshold. This is highly complementary to our ongoing work on Jouler, a new framework enabling flexible and effective smartphone energy management [53, 54]. Jouler energy managers implement per-device energy management policies that use signals to tell apps that they should consume less or more energy. These could be used to make energy-performance tradeoffs exposed by `maybe`.

1.2.4 — Continuous and triggered retesting. The `maybe` life cycle does not end after adaptation policies have been distributed. The set of users for each app is constantly changing, as are devices, users, and environments. Ensuring that adaptation policies remain effective requires continuous testing of existing policies and periodic policy regeneration. `maybe` performs continuous testing by introducing a small amount of randomness into alternative selection after deployment.

A more important question is how to trigger a more thorough round of retesting. The continued effectiveness of an adaptation policy is threatened by the installation of the app on new devices, or by new categories of users, or in environments that introduce adaptation inputs not seen during testing. For example, the network performance threshold used in an adaptation policy decision tree may have been set without data from devices connected to either a much slower or much faster network that the tested devices. Once the app begins to be installed on such devices, the existing adaptation policy may no longer be appropriate.

We propose to investigate the use of both global and policy-specific similarity metrics to compare the set of tested devices with the set of devices currently using the app for the purposes of triggering retesting. Global metrics compare the sets using all common and app-provided adaptation inputs, while policy-specific metrics use only inputs that are used as part of the adaptation policy. Global metrics may result in too many false positives because they are considering aspects of dissimilarity that are irrelevant to the adaptation policy. On the other hand, policy-specific metrics may result in too many false negatives because they are ignoring new sources of dissimilarity that need to be considered by a new policy to perform effectively. We will also explore combining these results with continuous testing results, or using similarity scores to weight continuous testing results to make retesting sensitive to poor performance on lightly tested devices.

1.2.5 — Outputs and evaluation plan. The primary output of Y2 will be support for dynamic adaptation in the `maybe` library. We will continue to leverage the automated testing framework developed in Y1 to test alternatives in dynamic settings. Integration with the Jouler energy management framework to manage energy-related tradeoff is also planned. Dynamic adaptation will be evaluated in ways similar to those used to examine static adaptation in Y0. But we will also examine difference in performance between human-guided adaptation policies and machine-generated ones to motivate our approach. We will also build and evaluate metrics to trigger alternative retesting within the `maybe` backend. These will be evaluated by determining if they can detect cases where adaptation policies clearly need to be reformulated. False positives are less important than false negatives in this case due to the low overhead of automated testing.

1.3 — Thrust 3: Developing Implementation Flexibility (2019–2020)

Alice appreciates that `maybe` has helped her harnessing the implementation flexibility she has exposed in her app. But in certain places she is not sure about how to expose the flexibility that `maybe` needs. Fortunately, a new adaptation-driven optimization feature allows `maybe` to build testing environments helping her to add needed flexibility. After she completes optimizing her app for several of these environments, `maybe` has enough implementation flexibility to improve her app's adaptation policies.

We believe that developing adaptation policies is more difficult than exposing implementation flexibilty. But exposing implementation flexibility may also be challenging, particularly for single developers or small teams. In Y3 we address this challenge in several ways. First, we propose to explore adaptation-driven optimization. This new approach allows `maybe` to use testing and post-deployment data to help direct developers where implementation flexibility is needed. Second, we will investigate encapsulating `maybe` adaptation into libraries covering common smartphone-related adaptation scenarios. The libraries themselves will continue to adapt without any direct developer involvement. Finally, we believe that it may be possible to mine the crowd for implementation flexibility through coding competitions and other forms of collaborative development.

1.3.1 — Adaptation-driven optimization. Performance optimization is a common development task. Beginning with a set of benchmarks and Amdahl's Law, developers iteratively identify and address performance bottlenecks. Normally the goal is to improve performance in a wide variety of environments. Unfortunately, given the diversity of environments smartphone apps may run in, optimization may result in solutions that are tailored to features of the testing environment and not generally applicable.

Fortunately, `maybe` allow us to turn what would normally be an optimization pitfall into an adaptation enabler. The key observation is that while normal code must work well everywhere, `maybe` allows developers to optimize the flexible parts of their code for different environments. Then `maybe` can use the additional flexibility to enable adaptation in the usual way through testing and automated policy generation. Given the testing framework developed in Y1, it is not even necessary for the developer to provide benchmarks. They can be derived from common subsets of user behavior.

We propose to explore a adaptation-driven optimization process that works as follows. First, `maybe` uses the testing results to identify a part of the app with a large impact on performance. Second, `maybe` builds multiple testing environments for the developer that represent the extremes of the environments under which the candidate code runs: the most and least memory, the fastest and slowest network, etc.

The next step is for the developer to iteratively optimize the selected part of the code in each environment. This may involve applying algorithms that make space or time tradeoffs, reducing quality levels, trading bandwidth for energy, deferring work, or other latency-hiding techniques. Each optimization provides another flexible alternative. The result in a new `maybe` statement providing a new and useful source of implementation flexibility. Instead of having to introduce flexibility by thinking about all of the different environments in which their app could run, developers can just do something that they are used to: optimize a single test.

1.3.2 — Adaptive libraries. Another way of encouraging developers to add flexibility to their apps is by using libraries that themselves incorporate `maybe`-driven adaptation. As a trivial example, a library may provide a drop-in replacement sort algorithm that uses `maybe` adaptation internally. This source of flexibility only requires that developers use new libraries, not expose flexibility or even use `maybe` directly. As a way of promoting adoption and handling common adaptation tasks, we will augment the `maybe` library with adaptive versions of commonly-used tasks. Examples include downloading files, manipulating data structures, and searching content. Our choices will also be guided by experience with the `maybe` system during Y1 and Y2 and feedback from developers using `maybe`.

1.3.3 — Crowdsourcing flexibility. Crowds are another source of flexibility that we will explore. Programming competitions such as Bloomberg's CodeCon [7] crowdsource many solutions to common development tasks. A large amount of implementation flexibility is likely to be present in these code archives, which could provide an ideal way to build the adaptive libraries described above. We will also experiment with soliciting solutions using systems such as Amazon Mechanical Turk [9].

We will request a library of answers from an existing code competition and, if that fails, hold our own to build up a solution library. Next, we will feed all of the solutions into `maybe` and test across a wide variety of environments and inputs. At that point the `maybe` adaptation policy formulation process can choose which solutions to include and which to reject. Given that crowdsourcing provides much more implementation flexibility than any developer or team of developers could provide alone, this will also serve as an interesting investigation into the benefits and limits of large amounts of flexibility.

1.3.4 — Outputs and evaluation plan. Y3 will produce several outputs. First, we will augment `maybe` to support the adaptation-driven optimization process as a beta feature. We will invite as small group of existing `maybe` developers to test this capability and continue development based on their feedback. Assuming this approach is successful, we will begin automatically hunting for apps that could benefit from additional implementation flexibility. Their developers will be notified and encouraged to use adaptation-driven optimization to provide additional flexibility. This new feature will be evaluated based on how well it identifies missing flexibility and how easily developers can correct the problem. Before and after testing will determine whether it helps produce more effective adaptive apps.

During Y3 we will also begin development of the `maybe` adaptation library. The library will not only be useful to developers, but also provide a way for us to continue to test and experiment with the system produced in Y2. We will evaluate the library based on usage and effectiveness. Finally, we will experiment with crowdsourcing-based flexibility, both to determine the marginal utility of larger and larger numbers of alternatives and as stress test for the entire `maybe` system.

1.4 — Thrust 4: Self-Modifying Adaptive Code (2020–2021)

> *Although Alice enjoys the fact that `maybe` automatically adapts her app, in certain cases she is not sure it is making good decisions. Overall she wishes that she had more insight into the adaptation policies that were being used. Fortunately, newer versions of `maybe` include Git and Android Studio integration allowing her to understand, visualize, and collaborate with autogenerated policies.*

`maybe` introduces a novel form of human-machine collaboration in developing computer code. Overall our goal is to leverage what each contributor does best. Although exciting efforts are underway to allow computers to automatically generate source code meeting a design specification, these projects are still in their infancy [69].

Such systems cannot yet replace a human developer, and will probably still not be able to in 2022. As a result, human developers continue to play a major role in `maybe`, defining both the inflexible and flexible parts of their app. However, when working with large amounts of data and creating policies to make constrained decisions, computers improve on human capabilities. As a result, `maybe` uses computer algorithms to design adaptation policies to make the choices exposed by the developer.

Until Y4 this process is open loop. Human developers expose flexibility. Computer algorithms harness it. But in Y4 we begin to explore how humans and computers can collaborate in a continuous adaptation development process. This requires addressing several new research challenges centering around the process of development using `maybe`. First, `maybe` must be able to communicate to developers about why and how it is using the flexibility they have exposed. Second, developers need ways to participate in the process, both by adding or removing flexibility as needed, and by tweaking adaptation policies. We believe that the solutions to these challenges lie in communication through the source code. And so research efforts in Y4 focus on integration with two existing and widely-used development tools: Git [8] and Android Studio [5].

1.4.1 — Adaptive self-modifying code. Unlike a typical open source library where the implementation might be hidden behind an interface but available for inspection, `maybe`'s operation can be quite opaque. While there is some computer code that runs each time a `maybe` decision is made, it is not visible to the developer. In the case of static adaptation it is run on the backend server. In the case of dynamic adaptation a decision function appropriate to each device is sent on demand and updated as needed. However, in both cases the call could be replaced with some code that is visible to the developer and runs on the device.

To make `maybe`'s adaptation policies more transparent, we propose to explore ways for `maybe` to act as a project collaborator. Once developers begin using `maybe` they are essentially collaborating with machine learning algorithms. When `maybe` updates a decision that corresponds to a change to the source code, or to a commit in a typical version controlled source code model. In Y4 we will let developers integrate `maybe` with their Git repositories so it can represent and update adaptation policies.

We will explore multiple ways to do this to ensure that this new `maybe` Git bot does not interfere with human-led development. One option is to only have `maybe` maintain comments that contain its decision policy, or write them to special files. We also must ensure that the representation of adaptation policies produced by `maybe` is intelligible and modifiable by human developers. This will allow them to tweak `maybe`'s policies if needed, or to halt continued adaptation development for a particular statement by replacing the call to the `maybe` library with the policy formulated at that point in time. Once both `maybe` and the human developer begin editing adaptation policies things become more complicated. But we will explore ways to support this workflow.

1.4.2 — Android Studio integration. Development using `maybe` involves reasoning about data. Data drives the process of alternative selection, and after Y3 can help guide developers toward places where flexibility should be added. Integrated development environments can be ideal places for exposing developers to information collected and used by `maybe`. And so in Y4 we also propose to develop and distribute an Android Studio plugin to aid in the process of `maybe` development for Android.

Android Studio is the official integrated development environment (IDE) for Android. It has a plugin architecture that will make it possible to integrate with `maybe`. Because the `maybe` workflow is new, it will require experimentation to determine what information is helpful for developers. The ability to view and visualize autogenerated adaptation policies will serve as a starting point and leverage ways of representing policies developed to support the Git bot described above. Android Studio also provides an ideal interface for viewing the data that drives `maybe` decisions, both derived from pre-deployment testing approaches developed in Y1 and from post-deployment testing present from Y0.

1.4.3 — Outputs and evaluation plan. Y4 will produce software to run the `maybe` Git bot, represent `maybe` decisions in a straightforward way, and a plugin integrating `maybe` visualizations into Android Studio. We will evaluate these components through continued interaction with the `maybe` developer community. Our primary concern with the Git bot is that it provide useful information and not interfere with the normal development cycle. We will use the Git bot ourselves and query developers about its effectiveness, particularly at helping them understand the adaptation policies that `maybe` has designed. The main goal of the `maybe` Android Studio plugin is to improve usability. And so we will evaluate it partly based on how widely it is used by existing developers and based on their qualitative feedback.

1.5 — Thrust 5: New Environments and Language Features (2021–2022)

Adaptation has long been recognized as important to developing mobile systems and apps. It is also increasingly important in other areas. Y5 is devoted to both wrapping up tasks from prior years and leveraging project discoveries to expand the scope to other types of systems.

1.5.1 — Support for other environments. Single pieces of source code must now run effectively on devices ranging from powerful supercomputers to energy-harvesting Internet of Things nodes. We expect that `maybe` can also aid in the process of adapting other kinds of code. In Y5 we will enable this by adding support for other languages. This will also allow us to broaden our developer base past simply Android. Fortunately, the SixPack library that we have chosen as a starting point already includes libraries for multiple languages. So we should be able to leverage a common backend to support `maybe` in multiple languages.

Of particular interest is building adaptive web pages. Similar to mobile apps, a single web page may be served to powerful desktops and energy-constrained mobile devices. The JavaScript code that many web pages include confronts a similar set of challenges as mobile apps, including differences in computational power, network performance, and the need to conserve energy. While responsive design principles have been widely applied to adapt the content of web pages to the device's display, adapting the programmatic parts of the web page remains an open challenge. During Y5 we will explore adding `maybe` support to JavaScript and using it to build adaptive web pages.

This will also allow us to experiment with novel forms of cross-device adaptation. JavaScript is one of the most active and widely-deployed languages and can be run both in the browser and on backend servers using `node.js` [10]. By combining an Android Java app with a JavaScript backend, we can determine how to jointly adapt systems across device boundaries. Use of external interfaces is very common in modern apps, and those interfaces may have a role to play in enabling effective adaptation. In other cases, the results of adaptation decisions that are made on either the backend or the mobile smartphone are only visible on the other end of the interface. For example, an app's decision to offload search functionality to a backend server may cause its load to increase and number of clients it can support. As smartphones and cloud servers continue to be integrated into a single seamless programming environment, adaptation policies must be able to cross the smartphone-cloud divide. During Y5 we will explore ways to bridge this gap.

1.5.2 — Language support for adaptation. For ease of deployment `maybe` is built as an Android library. However, as adaptation becomes increasingly important to building robust computer systems, we want to determine whether flexibility deserves first-class language constructs. For example, the flexibility exposed by the `maybe` library could also be exposed using a `maybe` statement. Structured similarly to current `if-else` conditions familiar to all programmers, the `maybe` statement would express flexibility at the language level. Temporal flexibility in scheduling particular task might be expressed through a `sometimes` keyword.

During Y5 we explore integrating support for the `maybe` statement into the Polyglot Java compiler front end [62]. Incorporating flexibility as a language feature and exposing it to the compiler has several benefits. The compiler may be able to more efficiently add built-in logging and scoring to `maybe` statements. Compiler support combined with static analysis can also help developers debug problems with `maybe` statements including unintentional dependence between statements. Similarly, if the compiler can determine that two statements are entirely independent, this will simplify the process of testing since permutations of both statements do not need to be separately tested.

1.5.3 — Outputs and evaluation plan. Y5 will result in additional support for `maybe` in at least one language other than Java, with JavaScript being the preferred starting point. We will reach out to `maybe` developers to solicit use cases where adaptation must cross device boundaries and add any necessary support to `maybe` to enable this use case. Finally, we will augment the Polyglot compiler to allow `maybe` statements to be used in lieu of library calls. We will study the usability of this feature compared with the library approach. Time permitting, we will explore other ways to leverage the additional visibility into flexibility at the compiler level using Polyglot.

Overall we anticipate that the discoveries made throughout the proposed project will allow us to argue for or against flexibility as first-class language construct. It will be exciting to see if future language designs change to encourage the flexibility that is so important to enabling adaptation. And how computers and humans continue to find ways to collaborate in adaptive software development.

1.6 — Related Work

While we believe that `maybe` is the first system using exposed implementation flexibility to enable runtime adaptation, there have been related efforts to adapt mobile systems and address uncertainty and context adaptation at the language level.

1.6.1 — Mobile systems adaptation. Attempts to enable more adaptive mobile systems date back to efforts like Rover [43] and Odyssey [61]. In many ways Android's design reflects principles espoused by these early efforts, particularly the goal of enabling *application-aware adaptation* by providing information about environment changes and resource availability. Unfortunately, while necessary, awareness is not sufficient to enable adaptation. Just because developers know what happened does not mean they know what to do.

A taxonomy of approaches to enabling adaptation on early mobile systems [14] reflects the focus of early efforts on incorporating adaptation into libraries that could be used by multiple apps, and this is reflected by recent efforts that address adaptation through single solutions to common decisions made by mobile apps. Refactoring code may allow certain problems to be addressed by experts, who are more likely to be able to design effective adaptation policies. But even expert developers currently lack and could utilize a way to expose flexibility, and `maybe` could be used to build better data-driven adaptation libraries. Libraries also fail to address the app-specific adaptation challenges `maybe` can help resolve.

1.6.2 — Split testing. There is a great deal of recent interest in split or A/B testing as a hypothesis testing tool for improving software after deployment, and companies such as Apptimize [4] are bringing split testing to mobile apps. However, existing approaches consider the process to terminate by reaching a global best decision. `maybe` can be seen as a natural extension of this technique to the diverse world of mobile devices, where one approach rarely fits all. Split-testing libraries provide syntax that is tailored to hypothesis testing which will eventually converge to a single value, not code that may perpetually contain flexibility.

1.6.3 — Language support for uncertainty. New systems such as EnFrame [78] reflect growing interest in managing uncertainty at the language level. EnFrame focuses on enabling programming with uncertain data, rather than the runtime adaptation enabled by the flexibility exposed by `maybe`. Aspect oriented programming (AOP) [45] aims to increase modularity through the separation of cross-cutting concerns. The programmer expresses cross-cutting concerns in stand alone modules, or aspects, which specify a computation to be performed as well as points in the program at which that computation should be performed. Fundamentally, the goals of AOP and the `maybe` system differ, with AOP focusing on modularity and `maybe` focused on enabling adaptation by exposing implementation flexibility. Although we believe that Aspects could be used as a mechanism to realize `maybe` statements, it is unclear whether the overheads of the require dynamic weaving [64] offset the software engineering benefits that aspects provide for expressing `maybe` statements that are already self-contained and localized.

Context oriented programming (COP) [24,68], much like AOP, aims to simplify the expression of computation in a more modular fashion, but focuses specifically on providing the ability of adapting behavior dynamically to the current execution context. We view `maybe` statements as a potential candidate for expressing context oriented adaptability. However, the condition that determines which computation to perform need not be determined by the context in which the `maybe` statement is executing. Indeed, `maybe`statements can leverage information gathered over time, can learn from the execution of similar computations, or can take a more holistic and system wide view for decision making. Moreover, context does not need to be specified explicitly when using `maybe` statements, allowing for adaptability to unknown or unpredicted situations. Fundamentally, while context oriented programming attempts to provide more information to the current computation, it still prevents programmers from exposing flexibility.

Previous work has explored adapting Java to improve performance for high-performance computing applications by applying loop and array transformations [51], but this was done without using exposed flexibility. Adaptive Java [56] realized a different cleavage between observing and changing program behavior, but we believe that this separation is less useful than the one achieved by exposing implementation flexibility. `maybe` shares similarities with language-based approaches to adapting energy consumption for sensor network systems such as Eon [75] and Levels [47]. However, these approaches still require programmers to eliminate flexibility by associating code certain energy states, rather than allowing the `maybe` system to determine which energy states are appropriate. Other approaches like Pixie [52] require rewriting existing code using new programming models.

2 — Broader Impacts

The `maybe` system will benefit society by enabling transformative smartphone apps. My proposal also includes an educational plan closely integrated with the research agenda. Section 2.1 describes how to transform my new modular online course on the internet into an adaptive course that uses implementation flexibility to improve educational outcomes. By utilizing a new concept library, contributions from multiple instructors will provide the flexibility needed to adapt to students. This section also describes a new graduate seminar and my other curriculum development efforts (§2.2), and my outreach activities (§2.3).

2.1 — Online Course Adaptation

Starting in Fall 2016 I will teach CSE 199, a new freshman course on the internet. The internet is the most significant thing computer scientists have ever built. It's development is rich in history, design principles, exciting engineering, societal relevance, and deep human implications. I cannot imagine a more exciting topic or a better way to introduce students to computer science.

But my interest in developing CSE 199 is not only a result of the material. The course is part of UB's new Freshman Seminar program, which is intended to allow freshman to interact with faculty in small groups of up to 25 students. However, the "seminar" that I am designing will be taught by multiple faculty to groups of 100 students. Swamped by growing interest in computer science, my department does not have enough faculty to offer 25-student seminars to incoming students. Instead, our plan is to use a combination of online learning and flipped classroom activities to provide personalized learning. (Note that the flipped classroom activities are not the focus of my integrated educational plan and so not described here.) Our challenge is to use technology to scale the small-group seminar experience to large groups of students.

To address this challenge, I propose to enable adaptive courses using the same approach I propose to enable adaptive smartphone apps: exposing implementation flexibility. Educators will expose implementation flexibility and student assessments will test the alternatives. The results will be used to create courses that improve learning by adapting to students. The first realization of this approach will be `internet-class.org`, an online learning framework focused on teaching internet concepts for CSE 199. To illustrate our vision, we begin by describing how a student will interact with `internet-class.org`.

2.1.1 — Student interaction with adaptive courses. When Alice logs on to `internet-class.org` to learn about the internet she is presented with a series of concepts. For each concept there are one or more explanations and assessments. Explanations may be videos, animations, slides, or text. Most concepts have multiple explanations provided by different instructors drawn from the `internet-class.org` concept library described below. For example, three videos have been uploaded by three different instructors explaining packet routing. Another instructor has contributed a diagram. Assessments may be a multiple choice quiz or a written or video explanation and be graded synchronously or asynchronously. Synchronous assessments are used to ensure that Alice masters each concept before continuing. The results of asynchronous assessments may send her back to review concepts once the assessments are graded by the course staff.

As it interacts with Alice `internet-class.org` is using implementation flexibility to adapt to her and help her learn. Multiple explanations are one source of flexibility. They allow the site to steer Alice toward explanations that work well for her, reserving others for reinforcement or review as needed. Concept ordering is a second source of flexibility. This allows the site to provide Alice with choices about what to learn next based on concept prerequisites and maps provided by instructors. Periodically `internet-class.org` also has Alice help measure available alternatives. For example, to gather data it might show her a random explanation first rather than the one it thinks will be most effective.

2.1.2 — Building an collaborative concept library. Lecture-based courses provide few opportunities to expose flexibility. Lectures bundle multiple concepts together and make it impossible to reorder concepts, insert inline assessment, or repeat explanations when needed. As a result, realizing this vision for adaptive online courses requires starting with a modular course with exposed flexibility.

We are designing `internet-class.org` to support adaptation. One way it achieves this is by building on top of a public concept library. I am creating and seeding the library with focused concepts, short explanations, and targeted assessments, all appropriate for CSE 199 and similar courses. By the end of 2016 `internet-class.org` will contain at least one video explanation authored by me for all required concepts. Each year students will also record additional explanations each year as one CSE 199 assignment, and I anticipate that some will be effective enough to add to the permanent concept library.

Modularity also makes it possible for multiple instructors teaching the same course to create and share compatible and mutually reinforcing concept explanations. As described above, overlapping concepts contributed by multiple instructors serve as a source of flexibility that the adaptive course can harness to enable adaptation. We expect to discover that different instructors have different styles that may appeal to different students, as well as different strengths allowing them to provide more effective explanations for certain concepts. Over time, collaboration will generate a large and varied concept library including optional advanced material for students that want to learn more about certain concepts.

The library is designed to encourage collaboration. It will be freely available and collaboratively maintained on GitHub and YouTube. Concepts are as simple and as atomic as possible, meaning that explanations and assessments are short and focused. This makes it easy for instructors to incrementally improve existing explanations and design new courses based on existing concepts. We are beginning by recruiting other UB faculty as contributors since this also helps introduce students to the department. As part of the proposed project we will also reach out to experts in industry and at other academic institutions. `internet-class.org` is being designed to make it simple to invite experts and for them to contribute short video explanations.

2.1.3 — Implementation flexibility in adaptive courses. Beginning with a modular concept library makes it easy to expose implementation flexibility. Below we describe each source of implementation flexibility in more detail and how we propose to explore it using `internet-class.org`.

1. **Ordering and presentation of explanations for a single concept.** For non-optional concepts the common case for adaptation will be that multiple explanations are available. During Y1 (2017–2018) we will augment the existing 2016 concept library with at least two additional video explanations for all concepts required by CSE 199. I will be the primary author but will also engage others in the process. Starting in Y2 new explanations will be added as needed to broaden the concept library by providing optional material or address weaknesses with existing explanations.

   Harnessing this flexibility for adaptation requires measuring the effectiveness of each explanation and then determining which explanations should be shown (or hidden) and in what order to each student. During Y2 all explanations will be available presented in a random order. In Y3, based on Y2 assessment results, we will remove poorly performing explanations and reorder the others based on effectiveness across all students. In Y4 and Y5 we will experiment with adaptive ordering using features such as explanation effectiveness, explanation instructor, student attributes, and performance of similar students.

2. **Ordering of multiple concepts organized into a single course.** `internet-class.org` will allow instructors to annotate concepts with prerequisites or contribute concept maps providing a total concept ordering. Ordering flexibility then arises in two ways: through different concept maps from multiple instructors, and when concept prerequisites constraints fail to establish a single path.

   Harnessing this flexibility for adaptation requires measuring the effectiveness of each ordering and then determining how to direct each student. In Y1 and Y2 we will recruit other instructors to provide alternate orderings for CSE 199 concepts. During Y3 we will require that students choose and strictly follow one of these orderings. In Y4, we will allow students to choose their own ordering based on concept prerequisites, but make suggestions based on the results from Y3. And in Y5 we will experiment with adaptive ordering by matching the route chosen by each student to an established concept map contributed by an instructor.

To evaluate the approach all student interaction with `internet-class.org` will be recorded, including what explanations are viewed, for how long, and in what order; the path and timing of student progression through the concepts; and performance on all assessments. Data will be made available to each instructor and shared between instructors at different institutions subject to Institutional Review Board (IRB) approval.

2.1.4 — Rollout, evaluation, transparency, and student interaction. In Y2 when the concept library is robust enough to support adaptation it will be made available to CSE 199 students as an optional feature. Assuming a successful Y2 experiment, adaptive features will become mandatory in Y3. Data from Y3–Y5 will be used to comparatively evaluate the effectiveness of different adaptation approaches. We will work closely with UB's IRB to ensure human subject compliance for all experiments with the adaptive course.

We also feel that it is important to create a system that students feel is helping them—not just experimenting on them—and make sure that students are always in control of their learning experience. To do this, the adaptive course will provide transparent adaptation. Students will be provided information explaining why

explanations are ordered in a particular way, why some are hidden, or why the system is making a particular suggestion. Students will be allowed to override the system by revealing and viewing any hidden concept explanation and proceeding to any concept that they have met the prerequisites for, regardless of the adaptive course's ordering suggestions. Finally, we will solicit student feedback on all explanations and assessments through an optional rating system. While these ratings may not be as reliable a measure of the success or failure adaptive course components as the assessments, including them helps engage students.

2.1.5 — Related work. Adaptive and personalized education has long been a goal of online courses [16,27], and today companies like RealizeIt [67] provide platforms for creating adaptive courses. My contribution to this goal is to explore the role of implementation flexibility, and particularly the idea of using implementation flexibility as a way of building adaptive courses that harness contributions from multiple instructors.

While the emergence of MOOCs is in the process of changing university education, the dominant form of instruction on existing MOOC platforms such as edX [32], Coursera [25], and Udacity [77], is non-adaptive and non-collaborative. While efforts are underway at MOOC platforms like edX to enable comparative evaluation through techniques such as A/B testing, the goal is still to accept or reject different equivalent components rather than using them as flexibility to enable adaptation.

The idea of breaking content into small modules is not new, and has been used heavily in mathematics education both at the college and K–12 level [33]. One source of implementation flexibility borrows from the idea of concept maps [81], which have long been used by instructors when laying out material for courses. However, we are not aware of any efforts to use multiple concept maps to enable adaptation.

2.2 — Curriculum Development Activities

During the proposed project my graduate research seminar will track the currently active research thrust. The course will provide students with an opportunity to guide the development of the `maybe` system, learn about mobile systems adaptation, modify the Android platform, use machine learning techniques to automatically generate adaptation policies, and help support developers using `maybe`.

I will have a chance to apply insights resulting from the adaptive `internet-class.org` to other courses and to the curriculum as a whole. Since Fall 2016 I have been leading a holistic overhaul of our entire CS curriculum. Simultaneously, I have been heavily involved in the design of a new introductory programming sequence. Both of these curriculum development activities will benefit from my experience with `internet-class.org` and integrate well with the project's research activities. If the adaptive `internet-class.org` features are effective, I will also apply them to my undergraduate course on operating systems—`ops-class.org`—and to our new introduction to programming.

2.3 — Outreach

Over the past few years, 12 women have contributed to my research group: Rizwana Begum, Drexel Ph.D student; Aishani Bhalla, Gela Malek Pour, and Lakshmi Ethiraj, UB undergraduates; Vinu Charanya, Rajeshwari Adapalam, Anuja Raval, Bhaavya Kapoor, Denise Blady, and Ramya Rao, UB Masters students; Sonali Batra and Anudipa Maiti, UB Ph.D. students. I have also worked with undergraduates from underrepresented groups as part of UB's Louis Stokes Alliance for Minority Participation (LSAMP) program.

I am involved in several efforts to improve gender diversity within CS. UB's undergraduate CS programs are only around 10% female, which lags behind a national average which is itself far too low. As a creative way of encouraging diversity, I organized the creation of a Diversity in Computer Science Mural. Students submitted images celebrating diversity, and the winner was an iconic 1960s photo showing Grace Hopper leading a diverse team working on the COBOL project. I led the fundraising effort, and support from colleagues provided a prize for the winning submitter and paid for the mural's installation.

I also initiated the establishment of a local chapter of the Scientista Foundation, a national organization dedicated to improving female STEM representation. Our chapter was launched in 2014 and since then has organized many events to raise awareness about female participation in CS, including meet ups, panels, and monthly brunches. With the help of a donation from Bloomberg, the group expanded its activities next year—including sending a delegation to the 2015 Grace Hopper Celebration of Women in Computing (GHC). Although this issue will take more time and energy to address, there are some promising signs of change at UB. For example, for the first time last year the UB Student Chapter of the ACM, which I advise as a faculty mentor, elected both a female president and vice-president.

2.3.1 — Undergraduate research and engagement. I am committed to introducing undergraduates to research and have had the privilege of working with 18 undergraduates on a variety of projects, several of which have produced publications. Nick DiRienzo built a system called PocketMocker allowing smartphone users to conceal their true activities by injecting fake data [29,30]. Frank Rossi assisted in the development of the PocketLocker system which creates personal storage clouds from multiple personal devices [59].

My group currently includes 9 undergraduates working on projects including quantifying user quality of experience (Brijesh Rakholia), improving smartphone energy management (Kyle Schoener), investigating the interaction between Android and "modder" communities (Edwin Santos), benchmarking Android databases (Grant Wrazen and Lakshmi Ethiraj), using crowdsourcing to optimize Wifi networks (Grant Wrazen and Vighnesh Iyer), and `internet-class.org` content and infrastructure (Greg Bunyea, Aishani Bhalla, and Wesley Csendom). I integrate undergraduates into my group by having them work alongside us in the lab, giving them tasks appropriate to their training, and providing graduate students mentors. I will continue to involve undergraduates in both the research and integrated education components of the proposed project.

2.3.2 — Industrial and developer outreach. Critical to the success of the proposed project will be continued outreach to the Android developer and industrial communities. I am already engaged in discussions with Optimizely [11], a company providing a commercial Android AB testing framework. They are aware of and interested in our work on SixPack. We will also engage in outreach to the Android developer community, both by making `maybe` source code available and through attendance at Android developer conferences.

3 — Qualifications and Results from Prior Support

I lead the blue Systems Research Group, direct the PhoneLab smartphone platform testbed, and maintain an existing online course on operating systems (`ops-class.org`) and a new online course on the internet (`internet-class.org`). PhoneLab consists of 120 UB affiliates that run custom Android smartphone platform images including instrumentation and novel features in exchange for discounted service [12, 58]. In addition to the proposed project and projects described below, my group also uses PhoneLab to explore new approaches to smartphone privacy [30,31], design new distributed file systems [59], invent novel crowdsourcing approaches [57, 71–73], study smartphone databases [44], and improve smartphone sustainability [20].

3.1 — Results from Prior NSF Support

I am the PI of four current NSF awards [17, 19, 21, 23] supporting three projects:

1. **CSR: Small: Jouler: A Cross-Device Application Energy Management Framework for Smartphones** (CSR-1423215, $499,185, 9/1/2014–8/31/2017 [17]). *Intellectual Merit:* Jouler explores personalized energy management on smartphones. We have deployed and evaluated Jouler components on PhoneLab. Jouler also supported initial work on harnessing implementation flexibility that led to this proposal. *Broader Impact:* a new graduate seminar was held to explore harnessing implementation flexibilty. Once the Jouler framework is complete an open energy management will be held using PhoneLab devices. *Evidence of Research Products:* the project has produced 3 publications [18, 53, 54].
2. **CSR: Medium: Collaborative Research: Architecture and System Support for Power-Agile Computing** (CSR-1409367, $561,766, 9/1/2014–8/31/2017 [21]). *Intellectual Merit:* this project focuses on improving *power agility*, the ability to reallocate energy consumption across components to match demand and improve efficiency. We are again using PhoneLab to collect data and evaluate new approaches in pursuit of improving power agility. *Broader Impact:* the project has contributed features back to the `gem5` hardware simulator. *Evidence of Research Products:* the project has produced 2 publications [15,65].
3. **PhoneLab: A Programmable Participatory Smartphone Testbed** (CI-ADDO-NEW-1205656, $1,322,510, 6/1/2012–12/31/2016 [19]; CRI-SUSTAIN, $75,000, 9/1/2016–8/31/2017 [23]). *Intellectual Merit:* PhoneLab opened its platform sources to external researchers in February 2015 and has hosted multiple complete and ongoing experiments. *Broader Impact:* PhoneLab software and datasets are available to researchers subject to human subjects review. *Evidence of Research Products:* PhoneLab has produced 13 internal [18, 20, 30, 31, 44, 53, 54, 57–59, 71–73] and 3 external [36, 38, 40] publications.

Results from these projects have motivated and are synergistic with the proposed project. It was through developing new mechanisms to improve power agility and manage smartphone energy consumption that we developed an awareness for the importance of policies. PhoneLab provides a way for us to deploy both changes to Android and novel smartphone apps to evaluate the proposed research thrusts.

REFERENCES

[1] http://www.engadget.com/2014/06/25/google-io-2014-by-the-numbers/.

[2] http://opensignal.com/reports/2014/android-fragmentation/.

[3] http://www.appbrain.com/stats/free-and-paid-android-applications.

[4] www.apptimize.com.

[5] Android studio. https://developer.android.com/studio/index.html.

[6] Apptimize. https://apptimize.com/.

[7] Codecon. http://codecon.bloomberg.com/.

[8] Git. https://git-scm.com/.

[9] Mechanical turk. https://www.mturk.com/mturk/welcome.

[10] node.js. https://nodejs.org/en/.

[11] Optimizely. https://www.optimizely.com/.

[12] The PhoneLab smartphone platform testbed. http://www.phone-lab.org.

[13] Sixpack. https://github.com/seatgeek/sixpack.

[14] B Badrinath, Armando Fox, Leonard Kleinrock, Gerald Popek, Peter Reiher, and Mahadev Satyanarayanan. A conceptual framework for network and client adaptation. *Mobile Networks and Applications*, 5(4):221–231, 2000.

[15] Rizwana Begum, Guru Prasad Srinivasa, David Werner, Jerry Ajay, Geoffrey Challen, and Mark Hempstead. Energy-performance trade-offs on energy-constrained devices with multi-component dvfs. In *Proc. of the 2015 IEEE International Symposium on Workload Characterization (IISWC'15)*, October 2015. https://blue.cse.buffalo.edu/papers/iiswc2015-agility

[16] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. In *Adaptive hypertext and hypermedia*, pages 1–43. Springer, 1998.

[17] Geoffrey Challen. CSR: Small: Jouler: A cross-device application energy management framework for smartphones. NSF Proposal CSR-1423215, University at Buffalo, 2014. https://blue.cse.buffalo.edu/proposals/2014-csr-jouler/

[18] Geoffrey Challen, Jerry Ajay, Nick DiRienz, Oliver Kennedy, Anudipa Maiti, Anandatirtha Nandugudi, Sriram Shantharam, Jinghao Shi, Guru Prasad Srinivasa, and Lukasz Ziarek. `maybe` we should enable more uncertain mobile app programming. In *Proc. of the 16th Workshop on Hot Topics in Mobile Computing Systems and Applications (HotMobile'15)*, February 2015. https://blue.cse.buffalo.edu/papers/hotmobile2015-maybe/

[19] Geoffrey Challen, Murat Demirbas, Steven Y. Ko, Tevfik Kosar, and Chunming Qiao. CI-ADDO-NEW: PhoneLab: A programmable participatory smartphone testbed. NSF Proposal CI-ADDO-NEW-1205656, University at Buffalo, 2012. https://blue.cse.buffalo.edu/proposals/2011-cri-phonelab/

[20] Geoffrey Challen, Scott Haseley, Anudipa Maiti, Anandatirtha Nandugudi, Guru Prasad, Mukta Puri, and Junfei Wang. The mote is dead. long live the discarded smartphone! In *Proc. of the 15th Workshop on Mobile Systems and Applications (HotMobile'14)*, February 2014. https://blue.cse.buffalo.edu/papers/hotmobile2014-sustainability

[21] Geoffrey Challen and Mark Hempstead. CSR: Medium: Collaborative research: Architecture and system support for power-agile computing. NSF Proposal CSR-1409367, University at Buffalo, 2014. https://blue.cse.buffalo.edu/proposals/2014-csr-poweragility/

[22] Geoffrey Challen, Oliver Kennedy, and Lukasz Ziarek. Expressing uncertainty using the maybe system. Google research award proposal, University at Buffalo, 2015.
https://blue.cse.buffalo.edu/proposals/2015-gra-maybe/

[23] Geoffrey Challen, Z. Morley Mao, and Chunming Qiao. CRI-SUSTAIN: Collaborative research: Sustaining successful smartphone testbeds to enable diverse mobile experiments. NSF Proposal CRI-1929894, University at Buffalo, 2014.
https://blue.cse.buffalo.edu/proposals/2016-cri-phonelab/

[24] 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.
http://doi.acm.org/10.1145/1146841.1146842

[25] Coursera. Coursera.
https://www.coursera.org/

[26] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[27] Declan Dagger, Vincent Wade, and Owen Conlan. Personalisation for all: Making adaptive course composition easy. *Journal of Educational Technology & Society*, 8(3):9–25, 2005.

[28] Shuo Deng, Anirudh Sivaraman, and Hari Balakrishnan. All your network are belong to us: A transport framework for mobile network selection. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, HotMobile '14, pages 19:1–19:6, New York, NY, USA, 2014. ACM.
http://doi.acm.org/10.1145/2565585.2565588

[29] Nick DiRienzo, Gino Buzzelli, and Geoffrey Challen. Smartphone users want to be mocked (poster). In *Proc. of the 15th Workshop on Mobile Systems and Applications (HotMobile'14)*, February 2014.
https://blue.cse.buffalo.edu/papers/hotmobile2014-pocketmocker/

[30] Nick DiRienzo and Geoffrey Challen. Controlling smartphone user privacy via objective-driven context mocking. In *Proc. of the 6th International Conference on Mobile Computing, Applications and Services (MobiCASE'14)*, November 2014.
https://blue.cse.buffalo.edu/papers/mobicase2014-pocketmocker

[31] Nick DiRienzo and Geoffrey Challen. Should smartphone users mock apps? In *Proc. of the 6th ACM HotPlanet Workshop (HotPlanet'14)*, October 2014.
https://blue.cse.buffalo.edu/papers/hotplanet2014-pocketmocker

[32] edX. edX.
https://www.edx.org/

[33] Engage NY. New York State Mathematics Curriculum Modules for Grades P–12.
http://www.engageny.org/mathematics

[34] Jason Flinn and Mahadev Satyanarayanan. *Energy-aware adaptation for mobile applications*, volume 33. ACM, 1999.

[35] Huber Flores and Satish Srirama. Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. In *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*, pages 9–16. ACM, 2013.

[36] Zhaoyu Gao, Arun Venkataramani, James F. Kurose, and Simon Heimlicher. Towards a quantitative comparison of location-independent network architectures. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 259–270, New York, NY, USA, 2014. ACM.
http://doi.acm.org/10.1145/2619239.2626333

[37] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojicic. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, 3(3):66–73, 2004.

[38] Marian Harbach, Alexander De Luca, and Serge Egelman. The anatomy of smartphone unlocking: A field study of android lock screens. In *Proceedings of the 2016 ACM Conference on Human Factors in Computing Systems (CHI'2016)*, 2016.

[39] J Huang, F Qian, Q Xu, Z Qian, ZM Mao, and A Rayes. Uncovering cellular network characteristics: Performance, infrastructure. Technical report, and Policies. Technical Report MSU-CSE-00-2, 2013.

[40] Ryan Huang, Tianyin Xu, Xinxin Jin Jin, and Yuanyuan Zhou. Defdroid: Towards a more defensive mobile os against disruptive app behavior. In *Proceedings of the 14th International Conference on Mobile Systems, Applications, and Services*, MobiSys'16, 2016.

[41] Suk Yu Hui and Kai Hau Yeung. Challenges in the migration to 4g mobile systems. *Communications Magazine, IEEE*, 41(12):54–59, 2003.

[42] Jin Jing, Abdelsalam Sumi Helal, and Ahmed Elmagarmid. Client-server computing in mobile environments. *ACM computing surveys (CSUR)*, 31(2):117–157, 1999.

[43] Anthony D Joseph, Joshua Tauber, and M Frans Kaashoek. Mobile computing with the rover toolkit. *Computers, IEEE Transactions on*, 46(3):337–352, 1997.

[44] Oliver Kennedy, Jerry Ajay, Geoffrey Challen, and Lukasz Ziarek. Pocket data: The need for tpc-mobile. In *Proc. of the 7th Technology Conference on Performance Evaluation and Benchmarking (TPCTC'15)*, August 2015.
`https://blue.cse.buffalo.edu/papers/tpctc2015-pocketdata/`

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

[46] Dejan Kovachev, Tian Yu, and Ralf Klamma. Adaptive computation offloading from mobile devices into the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 784–791. IEEE, 2012.

[47] A Lachenmann, P J Marron, D Minder, and K Rothermer. Meeting lifetime goals with energy levels. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[48] Baochun Li and Klara Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *Selected Areas in Communications, IEEE Journal on*, 17(9):1632–1650, 1999.

[49] Chieh-Jan Mike Liang, Nicholas D Lane, Niels Brouwers, Li Zhang, Börje F Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, and Ranveer Chandra. Caiipa: Automated large-scale mobile app testing through contextual fuzzing. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 519–530. ACM, 2014.

[50] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 285–298. ACM, 2010.

[51] Shun Long and Michael O'Boyle. Adaptive java optimisation using instance-based learning. In *Proceedings of the 18th annual international conference on Supercomputing*, pages 237–246. ACM, 2004.

[52] Konrad Lorincz, Bor rong Chen, Jason Waterman, Geoffrey Werner-Allen, and Matt Welsh. Resource aware programming in the pixie os. In *Proc. of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[53] Anudipa Maiti and Geoffrey Challen. The missing numerator: Toward a value measure for smartphone apps. In *Proc. of the 15th Workshop on Hot Topics in Mobile Computing Systems and Applications (HotMobile'15)*, February 2015.
`https://blue.cse.buffalo.edu/papers/hotmobile2015-numerator`

[54] Anudipa Maiti, Yihong Chen, and Geoffrey Challen. Jouler: A policy framework enabling effective and flexible smartphone energy management. In *Proc. of the 7th International Conference on Mobile Computing, Applications and Services (MobiCASE'15)*, November 2015.
`https://blue.cse.buffalo.edu/papers/mobicase2015-jouler/`

[55] Philip K McKinley, Seyed Masoud Sadjadi, Eric P Kasten, and Betty HC Cheng. A taxonomy of compositional adaptation. *Rapport Technique numéroMSU-CSE-04-17*, 2004.

[56] PK McKinley, EP Kasten, SM Sadjadi, and Z Zhou. Realizing multi-dimensional software adaptation. In *Proceedings of the ACM Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)*, 2002.

[57] Anandatirtha Nandugudi, Taeyeon Ki, Carl Nuessle, and Geoffrey Challen. Pocketparker: Pocketsourcing parking lot availability. In *Proc. of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'14)*, September 2014.
`https://blue.cse.buffalo.edu/papers/ubicomp2014-pocketparker/`

[58] Anandatirtha Nandugudi, Anudipa Maiti, Taeyeon Ki, Fatih Bulut, Murat Demirbas, Tevfik Kosar, Chunming Qiao, Steven Y. Ko, and Geoffrey Challen. PhoneLab: A large programmable smartphone testbed (invited). In *Proc. of the 1st International Workshop on Sensing and Big Data Mining (SenseMine 2013)*, November 2013.
`https://blue.cse.buffalo.edu/papers/sensemine2013-phonelab/`

[59] Anandatirtha Nandugudi, Carl Nuessle, Geoffrey Challen, Emiliano Miluzzo, and Yih-Farn Chen. The pocketlocker personal cloud storage system. In *Proc. of the 6th International Conference on Mobile Computing, Applications and Services (MobiCASE'14)*, November 2014.
`https://blue.cse.buffalo.edu/papers/mobicase2014-pocketlocker/`

[60] Brian Noble. System support for mobile, adaptive applications. *Personal Communications, IEEE*, 7(1):44–49, 2000.

[61] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, SOSP '97, pages 276–287, New York, NY, USA, 1997. ACM.
`http://doi.acm.org/10.1145/268998.266708`

[62] Nathaniel Nystrom, Michael R Clarkson, and Andrew C Myers. Polyglot: An extensible compiler framework for java. In *Compiler Construction*, pages 138–152. Springer, 2003.
`http://www.cs.cornell.edu/projects/polyglot/`

[63] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 299–314. ACM, 2010.

[64] Andrei Popovici, Thomas Gross, and Gustavo Alonso. Dynamic weaving for aspect-oriented programming. In *Proceedings of the 1st International Conference on Aspect-oriented Software Development*, AOSD '02, pages 141–147, New York, NY, USA, 2002. ACM.
`http://doi.acm.org/10.1145/508386.508404`

[65] Guru Prasad, Scott Haseley, Rizwana Begum, Mark Hempstead, and Geoffrey Challen. New interfaces for achieving power agility on mobile devices (poster). In *Proc. of the 15th Workshop on Mobile Systems and Applications (HotMobile'14)*, February 2014.
`https://blue.cse.buffalo.edu/papers/hotmobile2014-agility/`

[66] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 190–203. ACM, 2014.

[67] RealizeIt.
`http://realizeitlearning.com/`

[68] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. Context-oriented programming: A software engineering perspective. *J. Syst. Softw.*, 85(8):1801–1817, August 2012.

[69] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[70] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 295–300. ACM, 2013.

[71] Jinghao Shi, Zhangyu Guan, Chunming Qiao, Tommaso Melodia, Dimitrios Koutsonikolas, and Geoffrey Challen. Crowdsourcing access network spectrum allocation using smartphones. In *Proc. of the 13th ACM Workshop on Hot Topics in Networks (HotNets'14)*, October 2014.
`https://blue.cse.buffalo.edu/papers/hotnets2014-pocketsniffer`

[72] Jinghao Shi, Liwen Gui, Dimitrios Koutsonikolas, Chunming Qiao, and Geoffrey Challen. A little sharing goes a long way: The case for reciprocal wifi sharing. In *Proc. of the 2nd Workshop on Hot Topics in Wireless (HotWireless'15)*, September 2015.
`https://blue.cse.buffalo.edu/papers/hotwireless2015-sharing`

[73] Jinghao Shi, Lei Meng, Aaron Striegel, Chunming Qiao, Dimitrios Koutsonikolas, and Geoffrey Challen. A walk on the client side: Monitoring enterprise wifi networks using smartphone channel scans. In *Proc. of the 2016 IEEE International Conference on Computer Communications (INFOCOM'16)*, May 2016.
`https://blue.cse.buffalo.edu/papers/infocom2015-scans/`

[74] Qingyang Song and Abbas Jamalipour. A network selection mechanism for next generation networks. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 2, pages 1418–1422. IEEE, 2005.

[75] J Sorber, A Kostadinov, M Brennan, M Garber, M Corner, and E D Berger. Eon: A Language and Runtime System for Perpetual Systems. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[76] Arvind Thiagarajan, Lenin Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod. Accurate, low-energy trajectory mapping for mobile devices. In *NSDI*, 2011.

[77] Udacity. Udacity.
`https://www.udacity.com/`

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

[79] Daniel T Wagner, Andrew Rice, and Alastair R Beresford. Device analyzer: Understanding smartphone usage. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 195–208. Springer, 2014.

[80] Yi Wang, Jialiu Lin, Murali Annavaram, Quinn A. Jacobson, Jason Hong, Bhaskar Krishnamachari, and Norman Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, MobiSys '09, pages 179–192, New York, NY, USA, 2009. ACM.
`http://doi.acm.org/10.1145/1555816.1555835`

[81] Wikipedia. Concept map.
`http://en.wikipedia.org/wiki/Concept_map`

[82] Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 315–330. ACM, 2010.