Augmenting Undergraduate Computer Science Education With Programmable Smartwatches

Andrey Esakia, Shuo Niu, D. Scott McCrickard
Department of Computer Science
Virginia Tech
Blacksburg VA 24060-0904
{esakia,shuoniu,mccricks}@vt.edu

ABSTRACT
Smartwatches are emerging as wrist-based computers capable of complex calculation and communication, and the computer science curriculum should reflect the challenges and opportunities that they provide in the education domain. This paper puts forth an experience report focused on efforts to incorporate smartwatches in an upper-level undergraduate mobile application development class during two academic terms. Lectures, in-class activities, homeworks, and projects were tailored toward providing rich design and implementation experiences for the students that engaged them in developing for the smartwatch and a paired mobile device. Our experiences highlighted how incorporating smartwatches into a mobile app development class adds a valuable dimension in terms of design and implementation challenges and allowed students to exercise some of the fundamental computer science topics.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]

General Terms
Design, Human Factors

Keywords
Smartwatch; Pebble; Mobile; Android; Multi-platform

1. INTRODUCTION
A *smartwatch* is a wrist-based computing device with primary functionality as a watch but with added functionality made possible by the inclusion of a capable processor, graphical display, sensors, and wireless communication capabilities. Following many years of speculation and prototypes, major technology companies like Google, Samsung, LG, and Motorola are engaged in smartwatch development and support, and startups like Pebble are attracting record funding. Many of the smartwatch platforms provide open distribution channels for smartwatch applications (apps), providing opportunities to get apps in users' hands quickly and easily. As such, it is important to consider how smartwatches can fit into the computer science (CS) curriculum, toward identifying the most appropriate lessons to be learned from smartwatches and providing opportunities for students to design apps with broad availability.

Key in designing for smartwatches is consideration for their unique form factor. To ensure broad appeal and utility, smartwatches are intended to be attractive and compact, to be resilient to various environmental conditions, and to have a long battery life—supporting constant wear and close integration with daily tasks. Smartwatch manufacturers seek to balance features like screen size and sensor integration with appearance and durability, and while there is not yet consensus on essential features, most smartphones include a screen no larger than 2.5 inches on the diagonal, some sort of accelerometer or motion detector, and a means to communicate wirelessly like Bluetooth or Wi-Fi. Other features in select smartwatches include a camera, thermometer, GPS, touch screen, speaker, heart rate monitor, and external storage. The common features provide a baseline for designing educational activities, while the growing and changing feature set offers opportunities for innovation.

Increased miniaturization of computing and sensor technologies means that smartwatches can act as standalone devices, with significant computing power and capabilities (e.g., Samsung's Tizen smartwatches). But the greatest emerging opportunities seem to reside in coordinated use of smartwatches with other mobile and wearable technologies, seeking to leverage the strengths of each. This model is core in smartwatches such as Pebble, NikeFuel, and Samsung Galaxy Gear. As such, when teaching students to design for smartwatches, it is important to emphasize the multi-device coordination aspects that are essential to maximize utility.

Ironically, the popularization of the new technology brings to the forefront issues from the past that are often covered minimally in many CS education classes—most notably memory management, processing speed, power issues, and small screen design. The need to create a technology that is small and power-efficient leads to devices that lag in performance and capability compared to larger devices (i.e., desktops, laptops, and even tablets and mobile phones). With increased market presence of smartwatches and other very small technologies emerging, it is important that students understand how to design and program for hardware limitations.

In summary, this paper explores four fundamental CS topics that can be exercised through developing for smartwatches in an upper level mobile app development class:

- Considering a unique form factor
- Designing and programming for sensors
- Addressing multi-device coordination
- Handling hardware limitations

The novelty and popularity of smartwatches translates to enthusiasm among CS students, which offers a good building point. This paper puts forth an experience report for integration

SIGCSE '15, March 4–7, 2015, Kansas City, MO, USA.

http://dx.doi.org/10.1145/2676723.2677285

of smartwatches into the computer science curriculum. Specifically, we focused on a junior-senior level mobile design class that students take after completing CS2, and we chose the Pebble smartwatch as our design platform.

This paper first explores key elements of smartwatches that are important to cover in the upper level computer science curriculum. It then examines work related to mobile and wearable sensor-based platforms similar to smartwatches. The remainder of the paper describes two class sections for which smartwatches were an integral part of lectures, in-class activities, assignments, and projects—culminating in lessons learned and challenges for the future.

2. SMARTWATCHES AND EDUCATION

To appropriately integrate a novel device into a CS class curriculum it is important to analyze and understand the characteristics, features, and general nature of the device. We believe that a smartwatch is capable of adding additional dimensions to the educational experience that students get in an advanced (junior-senior) CS class—building on lessons learned in early courses such as C and Java programming, data structures, and algorithms. Additionally, smartwatches have the capacity to provide an exciting platform for key later topics.

This section explores smartwatch characteristics with regard to important aspects of CS education. Each discussion category highlights general features of smartwatches, both with regard to development as well as specific to education, with a focus on the junior-senior undergraduate level for which we have direct experience. We seek to frame the lessons in terms general to all smartwatches, though our experiences were with the Pebble.

Pebble seeks to create an inexpensive smartwatch with long battery life through judicious selection of features. Pebble hardware characteristics reflect a very power efficient embedded class device. Pebble smartwatches have a Cortex M3 CPU with a maximum frequency of 80mhz, 4MB of internal storage, 128kb of RAM (96kb allocated for the OS). In addition to conservative computational capabilities, it has a low resolution monochrome screen with 139 pixels per inch and total resolution of 144x168. For other interactions the Pebble has no touchscreen and no sound—though it can provide haptic feedback through a vibrating motor. Interaction is via four hardware buttons that are located on Pebble's edges. Pebble runs a proprietary operating system that must be programmed in C (limited features can be programmed in JavaScript). Pebble is intended to be paired with another device (e.g., a mobile phone) via Bluetooth, allowing it to serve as an extension to the device and to leverage its more extensive hardware capabilities.

Form factor. Smartwatches are designed to worn as watches—on the wrist at all times. Thus, the smartwatch utilization paradigm is different from the one that is with smartphones and other mobile devices. The dimensions of the smartwatch impose certain limitations on the types of interactions that are possible. For example, the watch-like appearance limits possible display and interactions due to fewer pixels, fewer buttons, and more limited finger input sets. Techniques like voice commands, gestures, and accelerometer inputs are promising, and deserve thorough exploration. These form factor features of smartwatches can present additional challenges to students designing smartwatch apps. Solving such usability problems will allow students to learn to better utilize the unique hardware resources. Lessons learned from such design processes will apply to other devices as well.

Sensors. Smartwatches contain various sensors and devices, that have included accelerometer, magnetometer, microphone, speaker, thermometer, barometer, GPS, touch screen, heart rate monitor, light sensor, and camera. While these sensors may not be new to students with smartphone experience, the limited form factor of the smartwatch often results in unique combinations of sensors, often of lower quality. In addition, having sensors on the wrist tends to have great appeal for students, creating new challenges for the design and implementation of apps.

Multi-device coordination. While it is possible to create standalone smartwatch applications, the most engaging applications for the majority of smartwatch platforms tend to leverage data from one or more companion devices (e.g., via an external internet connection, a larger memory store, or additional sensors). Since Pebble and a companion device run on completely different platforms the communication and its representation in code varies dramatically between the devices. Thus, implementing communication mechanisms on both platforms allows students to gain deeper knowledge about the concepts of device-to-device communication as they implement similar communication functionalities on different platforms.

Limited resources. Smartwatch form factor suggests that the quality of resources available will always lag behind larger devices (e.g., desktops, laptops, and even tablets and mobile phones). As such, programmers will need to address limitations in memory and processing speed when developing for this platform. For example, Pebble hardware runs on a very power efficient embedded class hardware with limited memory, and as such does not support memory allocation using traditional C commands like malloc and sprintf. Thus, limited resources of Pebble give students experience developing for low performance hardware—lessons that will serve them well for smaller devices.

3. RELATED WORK

Smartwatches have been included in CS courses, but generally as an option for projects and not as a major focus point for engaging core CS concepts. However, the four CS topics defined in the previous section certainly have been addressed elsewhere within CS education, particularly in relation to mobile and wearable computing. This section provides a few examples for each topic with mention of their relevance.

Form factor has proven useful and appealing in courses that address issues of mobile and wearable computing throughout different stages of CS education. For example, the Android platform and App Inventor has been used to motivate K-12 students to get involved in computer science [4]. The authors provided students with projects and exercises that covered capabilities such as sensors and networking to teach Java programming and Android SDK development. The Sofia system provided a programming interface to simplify Android programming for early undergraduate CS classes [5]. The Android platform has been used in upper-level courses as well, such as an agile software engineering focus on socially relevant problems like disabilities [9] and in courses focused on operating systems [1] and databases [8] due to the unique dimensions provided or enhanced by the small form factor. The

smartwatch takes these challenges to another level, providing fewer pixels and inputs but an even more ubiquitous presence.

Sensors have grown in popularity as an educational topic with the rise of mobile and wearable computing. A working group on mobile computing courses noted that mobile devices provide opportunities for learning about sensors. Mobile and wearable sensors have been the key platform for teaching many sensor-driven application development undergraduate classes (e.g., [4,6,10]). From this prior work, it is clear the importance of sensor issues related to the reliability, lag, threading, and more, and as smartwatches become more sensor-rich, we expect they will be platforms to explore these issues.

Multi-device communication and networking issues are important in mobile phones. Burd and his colleagues note that programming for multiple platforms can put students in a better position to think critically about each platform [2]. Communication between platforms forces students to consider the growing concerns regarding Android OS security threats, toward understanding fundamental security concepts such as cryptography, network security, and security policy [7]. Since smartwatches (particularly the lower-priced ones like Pebble) rely on communication with another device for the bulk of their operations, communication and networking issues will be of even greater importance.

In addressing hardware limitations we can draw from the lessons from years past, when issues like memory and processor speed were important even for relatively small data sets. As noted by the Burd working group [2], issues like these provide the opportunity to introduce classic CS topics like responsiveness, memory allocation, algorithm complexity, and limited graphics and animation. We certainly found this to be true in teaching Pebble, which has extremely limited memory and a slow processor relative to student expectations.

Each of these topics are important on their own, but the issues become more interesting when considered together. Section 3 examines these topics in detail, balancing general facets of them with aspects specific to the Pebble smartphone and our target course. The subsequent sections provide details about smartphone integration into the course, focusing on these topics.

4. SMARTWATCH IN THE CLASSROOM

We taught two consecutive mobile app development courses aimed at junior/senior level CS major students—one in Spring 2014 and another in Summer 2014. Input about the first class was taken into account for the second iteration of the class. Course materials are at http://research.cs.vt.edu/ns/smartwatch

We were able to provide a Pebble smartwatch for over half the members of the first class, and for all of the members of the second class, thanks to a gift from Pebble. The smartwatches were handed out to students at least a week before the instruction about Pebbles to encourage students to use them and better understand potential utility of a smartwatch prior to designing and implementing any applications for it.

4.1 Spring course

The spring semester mobile app development course was our first effort to teach mobile software development on Android as a CS3 level course aimed at juniors and seniors of our university's computer science department. We divided the course into two-week modules, where every module covered fundamental aspects of mobile development with Android OS. One module focused primarily on smartwatches.

Our spring course had a relatively large enrollment of 63 students (56 males). Class met twice a week for 75 minutes per session over a 15-week semester. All students had or were given an Android mobile device for their use during the semester, and over half the students were loaned a Pebble. Pebble assignments were done in pairs, and students were given the option to include a smartwatch in experience reports and a group project.

All of the students had had multiple years of Java experience. Many of the students had CS2 mobile app development class taken that uses Sofia package as the teaching platform for Android development [5], and most others had Android experience gained through their own initiative or other opportunities. Almost all indicated familiarity with the C programing language to a level in which they felt comfortable with pointers, memory allocation, and other relevant features.

4.1.1 Lectures

Two class periods were dedicated to teaching introductory aspects of smartwatches and two class periods specifically focused on teaching Pebble development. We sought to balance a historical view of the evolution of wearables and smartwatches as well as overview of the core features, while focusing also on Pebble specific development. We discussed at a high level the hardware aspects of Pebble; we mentioned hardware limitations such as low memory with slow bandwidth and the resulting restrictions from developer's perspective in C environment. Although the majority of the class was familiar with C, we decided to go over the more challenging parts of C such as pointers, structs and function pointers.

The Pebble API overview and instruction covered core concepts of form factor, multi-device connectivity, sensors, and hardware limitations. We spent significant time on multi-device connectivity, covering the Android side of the development on how to create a companion app on Android and how to exchange information with Pebble. We concluded by informing students about the ways in which the interactive development platforms can be set up for Pebble and how to use them.

One class was a lab-style session in which students practiced app and companion app development for Pebble. Students brought their Pebbles to class and worked individually on the lab, sharing Pebbles as needed to complete our tutorial. The tutorial consisted of series of steps necessary implement a shopping list app that allowed creating a shopping list on the phone and passing the list to the Pebble. Students were given a skeleton code that, when completed, resulted in a shopping list created on a mobile device that could be checked off with the Pebble. With this lab students were exposed to two key smartwatch topics: advantages gained from different form factors and multi-device coordination.

4.1.2 Assignments

Students were required to do the homework for Pebble that we designed for the class. As all students had completed a highly-structured lab, the homework was very open-ended. We asked students to utilize two core smartwatch functionalities: device-to-device communication (as in the lab) and integration of sensors (accelerometer, magnetometer, or light sensor).

4.2 Summer course

We taught the same mobile app development course as a summer course. The course met 5 days per week for 75 minutes

per class session over 7.5 weeks. The class underwent changes based on difference in format and on feedback received from the previous course. The background and demographics of this class were similar to the spring offering, though this class had a enrollment of only 14 students (12 males). Due to the smaller size of the class we were able to hand out Pebble smartwatches to every student and make all assignments individual.

4.2.1 Lectures

Most lectures were the same as in the spring, but we added a session in response to feedback received from the spring class (see results section) to provide students with more practical examples and more hands-on experience. Thus, prior to the lab session, we conducted a hands-on lecture for which students were asked to bring their laptops installed with the Pebble development environments. During the lecture we walked students through a development of an example app and asked them to follow along and implement the example app on their laptops. The example app was a synchronized counter that allowed incrementing and decrementing a counter on the Pebble as well as on the companion device (see Figure 1).

It should be noted that the process of displaying a number on the Pebble's screen was not trivial, as memory limitations of the smartwatch had to be addressed (string conversion without sprintf). This example app served as the basis for the homework assignment and it helped students with three core smartwatch lessons: leveraging various form factors, overcoming hardware limitations, and multi-device coordination.

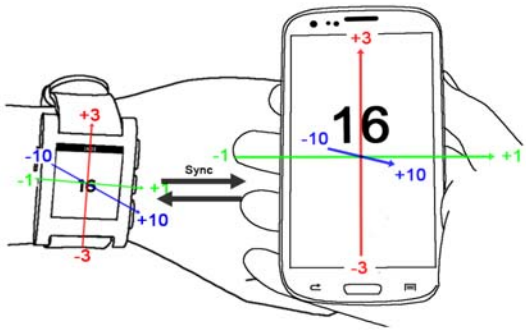

Figure 1. Pebble activity/assignment in which students leverage accelerometers to increment, decrement, and synchronize counters on both devices.

4.2.2 Assignments

For the summer class we decided to establish more control on the learning experience from the homework assignment. As one step, we assigned very specific homework that asked students to build on the counter app from the class activity and enhance it. The app from the activity needed one major improvement: it had no means of verifying successful and failed information deliveries between the Pebble and the companion device. With the app it was very common to see communication interruptions between the Pebble and the companion device, resulting in asynchronous numbers displayed on both devices. We asked the students to fix this problem by utilizing failed and successful message delivery handlers provided by Pebble SDK, thus synchronizing the counter on both Pebble and the companion device (and, of course, broadening the students' understanding of multi-device coordination). We also wanted students to practice accelerometer sensors programing, so we asked them to implement a simple linear gesture detection that would increment and decrement the counter by a number determined by the direction and the axis of the linear gesture. Students were supposed to implement this on both the Pebble and the companion device. This assignment helped students practice cross-platform communication and synchronization as well as accelerometer gesture detection and noise handling—both core concerns for this experience.

5. EXPERIENCE REFLECTIONS

Although our spring and summer classes varied dramatically in terms of enrollment and duration, our results indicate that incorporating Pebble development component into a junior-senior level mobile app development class, did encourage students to explore CS topics in a novel way.

5.1 Spring course experiences

Due to high enrollment in our spring course, students worked on the Pebble homework in pairs. We were able to allocate a full course module (two weeks) for the Pebble homework. Thus, students were given adequate time not only to innovate in terms of app design but also to implement quality apps.

A total of 33 submissions were made for the Pebble homework. Upon analyzing the functionality of the applications submitted we observed a significant diversity in terms of both, design and implementation. The following six applications caught our attention as the most prominent ones that addressed the core lessons we sought to communicate.

(1) Camera trigger: A Pebble app that triggers taking a picture from a companion device equipped with a camera.

(2) Speedometer: A Pebble app that receives information about the changing location of a companion device, measured by GPS, and calculates and displays the speed on its screen. The app warns the user if certain speed is exceeded.

(3) Navigator: A Pebble app that periodically receives navigational directions from a companion device and displays them on its screen.

(4) Pacekeeper: This Pebble app allows for a recording of a tapping pattern on the companion device, uploading the pattern to the Pebble and then recreating it via Pebble's vibration feature.

(5) MLB score updater: This Pebble app receives latest Major League Baseball scores from the web, through a companion device, and displays the scores on its screen.

(6) Realtime Twitter: For this Pebble app, user enters a search term into a companion device and then receives realtime updates, for that term, onto the Pebble's screen.

These example apps serve to demonstrate of how an integration of smartwatch device development into a mobile app development class can add extra dimensions to design and implementation aspects as well as revisit some of the core computer science problems in novel setting. Camera trigger app, for instance, was built as an addition to one of the earlier Android image capture homework apps, adding support for taking pictures remotely from the companion device directly or via a timer. Several of the apps required students to overcome transfer limitations of Pebble to support the intended functionality, in which Pebble does not receive information in chunks bigger than 128 bytes—students divided up the

information into 128 byte chunks on the companion device, then sent one chunk at a time and merge upon receipt of all chunks.

We asked all of the students to describe the Pebble homework experience and to identify some of the challenges identified. Upon analyzing the responses we discovered three most prominent issues related to the Pebble development. One on the issues being the lack of proper documentation and consistence across different versions of SDK e.g.,

"*One of the big limitations was the documentation. It seems that there are several built-in functions that do not work from version to version.*"

And

"*In comparison to most other devices and languages, there was hardly any documentation on developing for the Pebble. Even simple things, like reading in data from the accelerometer, were difficult to find. The methods themselves were there, but examples were rare and in some cases, nonexistent.*"

Another obstacle was the lack of support of standard C libraries due to the limited memory resources of the Pebble. Many students were unpleasantly surprised to discover that Pebble does not support conversion from numbers to strings using **sprintf()** function, e.g.,

"*Pebble's current API does not currently support some standard C programming functions. We encountered this problem when attempting to convert a "long" type variable to a "char *" type using C's standard library function "snprtinf". As a workaround, we were able to find an open source adaptation a Pebble user developed to overcome this limitation, but it was surprising to realize Pebble did not support such calls*"

Additionally, a lot of students complained about the lack of a dedicated Pebble SDK for Windows. Currently the only supported operating systems are Linux and Mac OS. Pebble does offer a cloud based IDE, but it lacks debugging support.

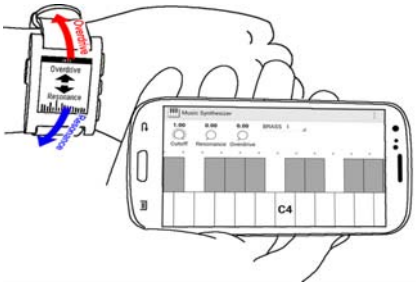

Figure 2. Synthesizer app that allows users to control overdrive and reverberation using Pebble's accelerometer.

5.2 Summer course experiences

In contrast to the spring course, the summer class had far fewer students (14). The duration of the session was half as long, which meant half the time per assignment—only one. With this in mind, and with consideration that all coding assignments were individual, we provided more design details and specific requirements for the assignments. Thus, the Pebble homework did not allow design creativity as much so as in the spring. However, we encouraged students to incorporate Pebble components into their semester project by offering them bonus points and the opportunity to skip the Pebble homework without penalty. As a result, six mobile app term projects incorporated Pebble, with three example term project apps that incorporate Pebble functionality listed here (see Figure 3 for screenshots).

(1) Facebook page monitor: An app that aggregates information from the Facebook page into a single screen and notifies its user of updates (see Figure 3). Update notifications (for example, "page like") are pushed both to the mobile device and Pebble within a user specified period, or the Pebble select button can be used to request an update.

(2) Synthesizer: An app that simulates a synthesizer keyboard with various sound effects on the mobile device. Pebble acts as an additional interaction device, by pivoting the Pebble against x, y or z axes the synthesizer changes its sound characteristics; e.g., reservation, echo (see Figure 2 & 3).

(3) Temperature monitor: An app that allows temperature and humidity monitoring of a remote room. Temperature and humidity sensors are placed in a room and the readings are being sent to a web server. The mobile app retrieves the data from the server and then allows Pebble to request the data from the phone. Pressing the select button on Pebble, downloads temperature and humidity information from the mobile device, and displays it on its screen (see Figure 3)

The Facebook page monitor app was originally conceived as a standalone mobile app, without any wearable components. However, after hearing about the Pebble, the student decided to incorporate its functionality into the term project that had already been designed and partially implemented. Incorporating the Pebble functionality prompted the student to reconsider her design for the app and add the notification functionality, thus naturally simulating a design evolution experience caused by external factors.

Similarly, the student behind the synthesizer app had not planned for Pebble inclusion originally. His original plan was to use mobile device's touchscreen the way rotate virtual knobs that changed synthesizer sound characteristics. After getting accustomed with Pebble and learning about the sensor capabilities through the lecture and homework, he decided to augment the sound altering controls for the accelerometer. The student discovered how to address accelerometer sensitivity issues by processing the accelerometer values prior sending them to the synthesizer app.

Since summer Pebble assignments were individual, all students undertook both design and implementation for Pebble. However, issues that the students experienced were similar to the previous class. Students complained about poor online documentation and lack of important programming supports. We received no complaints about the lack of Windows SDK as we provided students with the instructions that allowed them to install Ubuntu virtual machine that allowed them to use the full Pebble SDK with debugging tools.

Figure 3. Pebble screenshots from three of the summer apps, highlighting the look and feel of Pebble apps.

5.3 Discussion

By incorporating a novel and "cool" device into a mobile development class, we leveraged student enthusiasm for smartwatches, and wearables in general. Tasks of designing and implementing apps for a smartwatch, helped students think about design and implementation in terms of two devices and helped them think of interactions of users with two devices.

The limitations of our device of choice, Pebble smartwatch, played an important educational role. The limitations often forced the students to devise workarounds in the form of algorithms, in order to implement their ideas. The manifestation of limitations was natural and physical, in a form a smartwatch device. Thus, students solved limitation-induced problems in a natural way—with an actual device with limited capabilities.

From an instructor's perspective, the benefits that stem from the usage of a novel device like a smartwatch come at a price, especially when such device comes from small vendor. Just like our results indicate, the novelty of a device is usually synonymous with limited online documentation and code example availability. This could result in extra workload on teaching staff as the students inevitably will resort to the most available help—instructors and teaching assistants.

6. CONCLUSIONS AND FUTURE WORK

This paper seeks to highlight the potential of smartwatches in enhancing computer science education by incorporating a Pebble smartwatch development into our upper level undergraduate mobile application design class. We taught this class during two semesters and used the experience to evolve our teaching model. Through the smartwatch lectures and assignments the students not only gained knowledge about specific design and development techniques for the unique form factor and integrated sensors, but also tackled some of the fundamental computer science problems that arise due to resource limitations and multi-device communication issues.

At the start of this experience, we identified four key areas relevant to that we felt were most interesting and relevant to smartwatches. We revisit them here, capturing key lessons and challenges for each that seem important for future consideration in incorporating smartwatches into CS courses.

- Consider unique form factor. By repurposing the watch, the smartwatch is restricted to small size—but high accessibility in daily use, even more so than a mobile phone. It is vital to ask students to design for usability, but these lessons must be balanced with opportunities to explore new smartwatch uses.
- Design and program for sensors. The teaching of sensors has been explored and discussed, but the high visibility, small form factor, and decreasing cost of smartwatches positions them as a learning tool throughout the curriculum—from an early-course initial experience with sensor programming to one of many devices in an upper-level course focusing on advanced topics like distributed computing and networking.
- Address multi-device coordination. The vast majority of smartwatches require device pairing through Bluetooth, Wi-Fi, or another communication mechanism to generate broad and interesting apps. Designing for a multi-device situation will be unfamiliar to most students, requiring careful consideration by teachers to provide early experiences addressing it while allowing students to explore possibilities that maximize the capabilities of all coordinated devices.
- Handle hardware limitations. Even though hardware will continually improve, it seems likely that there will continue to be a performance gap between the smaller smartwatch and the larger desktop, laptop, and even tablet and mobile phone (just as [2] noted a gap between mobiles and desktops). This limitation can be viewed as an opportunity—providing the chance for students to broaden their skill set and revisit (or rediscover) design and programming techniques.

This paper puts forth initial areas for consideration in integrating smartwatches into the CS curriculum. Google's Android Wear and Apple's Apple Watch have recently joined the smartwatch market, suggesting that this emerging technology has promise. The computer science education community must be poised to help students design and implement for smartwatches armed with appropriate knowledge and insight. Future plans will leverage the unique hardware, rich interaction capabilities, and seamless developmental experience to exercise human computer interaction aspects associated with the form factor and the information affordance offered such smartwatches.

7. REFERENCES

[1] J. Andrus and J. Nieh, "Teaching operating systems using android. in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 2012, pp. 613-618.

[2] B. Burd, J. P. Barros, C. Johnson, S. Kurkovsky, A. Rosenbloom, and N. Tillman, "Educating for mobile computing: addressing the new challenges," in *Proceedings of the Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups*, 2012, pp. 51-63.

[3] H. Chen and K. Damevski, "A teaching model for development of sensor-driven mobile applications," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, 2014, pp. 147-152.

[4] M. H. Dabney, B. C. Dean, and T. Rogers, "No sensor left behind: enriching computing education with mobile devices," in *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 2013, pp. 627-632.

[5] S. H. Edwards and A. Allevato, "Sofia: the simple open framework for inventive android applications," in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, 2013, p. 321.

[6] A. Forster and M. Jazayeri, "Hands-on approach to teaching wireless sensor networks at the undergraduate level," in *Proceedings of the 15th ACM Conference on Innovation and Technology in Computer Science Education*, 2010, pp. 284-288.

[7] M. Guo, P. Bhattacharya, M. Yang, K. Qian, and L. Yang, "Learning mobile security with android security labware," in *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 2013, pp. 675-680.

[8] Q. H. Mahmoud, S. Zanin, and T. Ngo, "Integrating mobile storage into database systems courses," in *Proceedings of the 13th Annual Conference on Information Technology Education*, 2012, pp. 165-170.

[9] V. P. Pauca and R. T. Guy, "Mobile apps for the greater good: a socially relevant approach to software engineering," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 2012, pp. 535-540.

[10] S. Rollins, "Introducing networking and distributed systems concepts in an undergraduate-accessible wireless sensor networks course," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 2011, pp. 405-410.