An adaptable model for teaching mobile app development

Andrey Esakia
Department of Computer Science
Virginia Tech
Blacksburg, VA
esakia@cs.vt.edu

D. Scott McCrickard
Department of Computer Science
Virginia Tech
Blacksburg, VA
mccricks@cs.vt.edu

***Abstract*--As mobile software development becomes more mainstream, universities recognize a need to integrate mobile platform programming into the curriculum. This integration requires an understanding of mobile software development that defines it not just as a collection of topics but that acknowledges cross-topic areas that serve as the basis for specific interests. This paper presents a platform-agnostic model for teaching mobile software development that identifies established foundational topics, offering an adaptable teaching model aimed at providing enhanced understating of the topics and their integration. The model identifies three core areas of importance to computer science education: asynchronous programming, model view controller, and platform underpinnings. This framework enables mobile-specific topics like location, notifications, sensors, and more that are positioned with respect to the core areas to assist with lecture and assignment planning. Testing this model showed that students exhibited better skills and knowledge of both the core concepts and the specific topics as observed in their integration of features in term projects. This suggests that, through emphasis on core areas and repeated exposure to them in the context of new topics, students exhibit higher quality of integration of various mobile topics and deeper understand of the core-cross topic areas for mobile software development. Next steps are adoption by the education community in project-based mobile development classes of various format, scope, level, and purpose.**

Keywords-Mobile devices; Education; Mobile Programming

I. INTRODUCTION

Smartphone market has witnessed a dramatic change in the past decade. In 2014 the global smartphone sales reached 1 billion units [1]. According to PewResearch [2] in 2014 64% of adults in the US own a smartphone, and over 70% of college students. Even moreso than traditional computers, smartphones are highly integrated in people's lives, viewed as an essential tech device. Smartphones feature multicore processors coupled with powerful GPUs and high resolution screens. Practically every smartphone includes at least one camera, communication technologies, multitouch screens, and myriad sensors.

Computer science students are aware of the popularity of smartphones and are interested in gaining skills and knowledge needed to develop smartphone apps. For example, at our institution, student demand for the mobile software development class that we teach more than doubled in less than 18 months. In addition, code distribution and promotion is easy for mobile apps, with possibilities for quick and inexpensive revenue stream pointing the way to emulate success stories of young mobile app developers. Even the more traditional companies and jobs have mobile app development positions; a search for mobile-related jobs reveals over thirty thousand vacancies.

Mobile software development has been part of undergraduate computer science courses for many years, appearing prominently in education literature over the last five. Indeed, there are far more topics that can be covered in a single course, necessitating informed selection of the most appropriate topics that best matches the course or curriculum under consideration. This paper puts forth an adaptable teaching model that situates common topics with respect to core learning areas. The model is applicable to various classes: semester long or short classes, classes dedicated to mobile programming or classes that include it in a small way, classes for beginners or advanced students. The model focuses on teaching various topics but also gives a foundation for understanding how features can be integrated into apps.

This paper next provides a brief review of related efforts at teaching mobile computing. Then, a model is outlined and described. The paper explains how the model applies to a junior/senior-level mobile software design course and compares results from the class to previous iterations of the same class, prior to the model. The paper concludes with an analysis of demonstrated knowledge based on the comparison with two previous classes.

II. RELATED WORK

Classes that teach programming on mobile devices are becoming an integral part of curriculums of computer science departments. Computer science education research community recognizes the significance of smartphones and studies various aspects of mobile programming and development in education.

Numerous papers have reviewed topical approaches to teaching mobile topics, including sensors [3], operating systems [4], security [5], and smartwatches [6]. For example, Chen et al. [7] recognized inherent difficulties associated with learning sensor programming and proposed a teaching model that simplifies the learning process by replacing the typical trial-and-error approach with a divide-and-conquer one that avoids sensor indeterminacy issues. Other papers provide experience reports detailing issues that arose in teaching mobile computing. For example, Gordon [8] identified essential teaching concepts for mobile development topics such as user interface design, device communications, data handling, and event driven programing. Sung et al. [9] presented their experience teaching two variations of mobile app development class: implementation oriented and design oriented. Burd et. al [10] address challenges in mobile computing education by surveying instructors of hundreds of mobile computing classes, identifying how common topics are situated in a class. Derek [11] examined how mobile computing can be used to teach Java and increase employability.

Some papers provide toolkits or modules that capture key teaching lessons in a sharable and reusable manner. Mahmoud et al. presented [12] an academic kit aimed at helping educators integrate Java ME and Blackberry programming into their curriculum, including slides, tutorials, quizzes, labs and assignments. Yuan et al. [13] created a collection of modules that focus on eight core areas related to mobile programming and mobile security. This prior work provides a strong foundation to consider models that identify and integrate the core themes of mobile computing. Only recently have the dominant mobile platforms started to mature to the point that educators can consider the ways that the many themes can integrate with core educational goals. This offers a great opportunity for the development of a teaching model that focuses on identifying the common core areas that span through the myriad of mobile related themes and provides an understand on how these themes can be defined in relation to the common core areas.

III. MOTIVATION

Although mobile software development is an exciting new area in computer science education, the multi-component nature of apps and reliance on platform's software development kit (SDK) features as well as the need to follow certain architectural patterns when building apps [14] makes it challenging for students with no mobile software development experience to focus on the mobile specific topics listed in the related work section. Therefore, teaching mobile topics will inevitably include foundational (in the context of a given platform) knowledge that acts as prerequisite for the topics of interest. Without the foundational knowledge students will likely exhibit gaps of knowledge needed for building multi-component apps. For example, if a teacher's goal is to teach about sensors on smartphones, most likely, the teacher would also have explain how to output sensor readings onto the user interface, how to start an applications and how to build a simple user interface among many other things, depending on the depth of teaching.

As mobile software development is becoming increasingly popular (and necessary!) in computer science education, there is a need to have a versatile and flexible framework serves as a guide for computer science educators to teach topics related to mobile software development. Curriculum integration can encompass a spectrum of possibilities, from short, beginner-focused module on specific topics related to mobile development to a semester-long class with broad topic coverage for experienced upperclassmen students. Throughout the semesters of our experience teaching Android mobile software development to junior/senior level CS major undergraduate students, we sought to identify the unique aspects of mobile software development and synthesize them into the categories of core areas that span across various themes studied in such classes.In so doing, we seek develop a teaching model that can serve as a guide for teachers adopting mobile app education for the first time, and as a refinement tool for those teachers that are trying to evolve their mobile app development class.

Building on related work (e.g., [8, 10, 14]) and our own teaching experience, we seek to identify a platform agnostic set of challenges that students with little to no experience in mobile software development encounter. These challenges stem from the following:

*Events and event handlers***.** Mobile devices with touchscreens are inherently user centered and the graphical user interfaces are event driven. Students with no experience developing interactive apps encounter difficulties with events and event handlers.

Interfaces with callbacks. Energy consumption is a concern for mobile; thus applications are "paused", "stopped" and "resumed" throughout their lifecycle, with associated callbacks that developers must understand. Similar to app lifecycle, mobile devices' sensors, location services, component-to-component communication mechanisms, networking and multi-threading require callback methods to communicate within the application—often a new experience for students.

*SDK features***.** SDK-specific features are unique to the platform of choice and therefore contribute to the learning curve, such as tools for building user interfaces, supporting inter-app communication, and sharing resources, calling for student mastery.

*Non-sequential programming***.** Smartphone operating systems do not generally allow developers to perform long running operations (e.g., networking, thread communication, sensors) on the main thread to retain responsive user interface. Thus, smartphone applications rely heavily on asynchronous programing. Even the simplified threading supported by many smartphone platforms requires some knowledge of asynchronous programming.

*Using software architecture patterns***.** Smartphone developer guides tend to package recommendations in components, or architecture patterns, to support easy reuse. Additionally, app lifecycles require that some of portions persist and some are recreated (e.g., during screen rotation, the display changes but background operations persist) requiring a modular architecture in which layers of the application are decoupled. This architectural division is unfamiliar to many students, but is captured in terminology such as model-view controller (MVC) [14].

IV. THE MODEL

The goal of our model is to identify categories that capture cross-topic challenges and present an interactive framework within which the process of teaching can be defined with relation to the challenges. The previous section identified platform-agnostic challenges characteristic for mobile development. The challenges are not associated with a concrete mobile software development topic—they occur within all topics to varying degrees.

The challenges of event driven programming, and familiarity with features and tools offered by the SDK, are a part of the inevitable learning process for anyone trying to build a native app for a given platform. We identify a third and final core area that encapsulates the challenges the items needed for obtaining the foundational layer of *platform underpinnings*.

One area of challenge is related to asynchronous programming, a sub-discipline of computer science for which mobile apps require a level of understanding. Mobile development frameworks offer tools that encapsulate challenges inherent to multithreaded programming, making it feasible to teach it as part of mobile development class. We view *asynchronous programming* as a core area in our model due to its importance to many topics.

Another source of challenge lies in a need to follow architectural patters when putting together apps with different components. A mobile development class covers this to some extent (e.g., connecting swiping and menu patterns) but it grows in complexity. We view this source of challenges as one of the core areas in our model, referred to as *Model View Controller (MVC)*.

A. Core area descriptions

1) Platform underpinnings.

With this category we capture an array of tools, techniques and platform specific know-how that is needed to be prepared and to be able to implement various topics of interest. We envision this category as a collection of items that one would have to re-learn in switching between different developmental platforms. Items in this category include:

a) Overview of features offered by the SDK. This category focuses on the overview of what is given to the developer from the SDK and accompanying ways in which to install and deliver application to the end user. This includes subcategories such as installing and setting up the development environment, creating "hello world" app, familiarization with the tools for building application and user interface layout, preparing and publishing application on the store.

b) Basics of developing interactive graphical user interfaces. Understanding events and callbacks. Examples include: Building interfaces with UI elements included in the SDK (buttons, editable text, text, scrollable list, gallery), creating interface layouts, creating dynamic user interface containers, making interfaces interactive with callback functions that handle user induced UI events and basic ways of persisting GUI related data during interruptions.

c) Lifecycle methods for different components. Overview and practice on callback methods for a given components lifecycle methods (i.e. which method is triggered when the component is killed, created, paused)

d) Tools for inter-application communication and content sharing. Mobile platforms offer its internal tools for app developers to use. For example when an app launches another map (ex. user clicks on address and then navigation app launches) certain platform specific communication takes place, similarly when an app retrieves shared content such as images, it uses inter-component communication mechanisms.

e) OS-level events. This includes mechanisms for obtaining information about when device boots, low battery warnings, when it connects to Wi-Fi.

2) Asynchronous programing:

To achieve quality user experience, long running and computationally intensive tasks must be offloaded from the thread that is responsible for the user interface. This means that any application that has non-trivial functionality (apps with networking features, complex graphics, file I/O intensive) and has the ambition to offer competitive levels of user experience, it must employ some sort of asynchronous programming. So with this category we encapsulate a list of subtopics that explores platform's capabilities of executing and controlling background threads, long running background services and tools for controlling them:

a) Information exchange between threads. This includes overview of concepts behind threads (i.e. answering questions about what threads do and why use

them), ways to perform quick background tasks (e.g., network requests, database queries, file I/O operations, complex graphical computations), running parallel background threads (e.g., downloading large files).

b) Executing and controlling long running background operations. Examples include playing audio content, step-counting, network requests, and location updates.

3) Model View Controller: Non-trivial mobile apps have multiple components to them (e.g., networking, background services, sensors, multimedia, graphical user interface elements, storage) that need to work reliably, cohesively and efficiently. Given that mobile operating systems halt applications depending on circumstances (e.g., interruptions from user input or external events like phone call) it becomes difficult to organize and manage components to retain consistent and predictable app behavior—thus requiring organizational guidelines.

A well-known user interface architectural pattern—MVC—recommends separating *View* from *Controller* and *Model* [14]. By separating user interface from underlying components, handling the application lifecycle becomes more reliable from an end user perspective and clearer from developer's perspective. This enables consideration of questions regarding topics like user feedback of large file downloads, slow GPS feedback during wayfinding, and data visualization based on complex computations.

B. Ways to apply the model

Figure 1 represents the relationship between the three core areas as a "ring" arcing around a set of topics of interest. Our model is platform agnostic and adaptable to many teaching settings. The model supports explicit awareness of relative core mobile software development categories that act as a prerequisite for fuller learning experience for a topic of choice. To reinforce learning in one area, a focus on multiple topics in the area would provide learning opportunities for the area (e.g., networking and sensors leverage asynchronous programming knowledge in different ways), thus encouraging increased students skills.

It should be noted that for trivial applications only one area might be exercised; e.g., for a simple calculator app only the platform underpinnings would be needed. Therefore, for classes that seek a simplified platform experience (e.g., a mobile module in a parallel programming class), the topic bubbles could be shifted toward the area of relevance. Put differently, the depth at which class is taught determines not only the size but also the position of the topic bubble. The following sections demonstrate how this model can be used to design, assess, and compare classroom experiences.

V. APPLYING THE MODEL

We applied this model to a junior/senior mobile development class, with post-CS2 students knowledgeable in Java as well as the basics of data structures and algorithms. The class seeks to complement their existing CS skills with mobile software development experiences. The model shaped all class aspects: homework aassignments, in-class activities, labs, and lectures. We defined a set of sub-areas within each area of the model and applied them for each topic taught throughout the six weeks of the summer session as listed in Table 1. We hypothesized that the application of this model in the class would manifest into higher quality software produced by students.

A. Class format and student body

The model was applied to a six-week summer session. elective class that used Android OS as the platform of choice. The class had an enrollment of 32 students (5 juniors and 27 seniors, 22 Computer Science majors and 10 Computer Engineers).

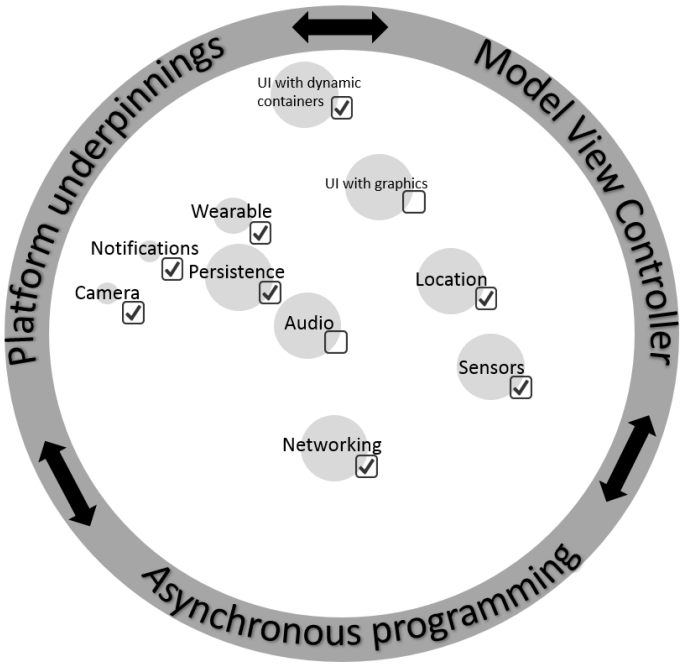

Figure 1. Adaptable teaching model, encapsulating key topics for mobile development. The model positions 3 core areas of study on the perimeter, with mobile development topics in the center. Proximity of topics to each area represents relative time spent within each area. Topic size reflects total time spent on the topic. The check mark represents coverage in our Summer 2015 class (when not all topics were covered).

The class met five times a week for 75 minutes each session. Once or twice a week were dedicated to in-class coding activities. All lectures concluded with example projects highlighting and complementing concepts from class. The class had 5 homework assignments and a semester project for pairs (6 students chose to work individually).

B. Theme, topics covered and assignments

We covered a broad range of topics throughout the term, answering questions related to the topics in terms of the orbit's three components.

1) Week 1 - Platform Foundation:

a) Lectures.

Introduction to platform development tools, overview of Android platform foundation (Activities, Services, Content Providers, Broadcast Receivers, Intents), Android user interface architecture, basic user interface design and implementation, application lifecycle methods and implications.

b) Assignments and in-class activity:

In-class activity and the first assignment explored basics of building user interfaces and incorporating user interactions through buttons and editable text fields. The first homework assignment asked students to build a mock client application for a statewide health intervention project with a login screen and main applications screen.

2) Week 2 - Platform Foundation & Model View Controller:

a) Lectures.

Continued on the "orbit of core mobile concepts" by covering Android's inter-component communication (Intents), followed by more interfaces, including versatile dynamic user interface elements (Fragments) and dynamic lists (ListView and Adapter, allowing building dynamic user interface wrapper for array of data). We covered concepts of MVC, and discussed best practices of using MVC in the context of our project theme, and reviewed tools offered by Android for establishing MVC.

b) Assignments and in-class activity:

Explored building dynamic and flexible user interface with Fragments. Students were asked to refine the application from the first assignment by building a skeleton interface for the client app with five Fragments capable of communicating data with the encapsulating Activity.

3) Week 3 - Platform Foundation & Model View Controller & Asynchronous programming:

a) Lectures.

Introduced asynchronous programming and its relation to MVC. We covered two main asynchronous approaches in Android (AsyncTask, Thread) and ways to communicate between the main application thread with background thread (Handler). We also covered Android's tools and mechanisms for establishing long running background services, ways of communicating with them in the context of the MVC pattern, and mechanisms for system wide communication (BrodcastReceiver).

		Topics
Platform Underpinnings	Platform overview & "Hello world" app	<All topics>
	IDE tools for building UI	Persistence, Wearable, Camera, UI with dynamic containers, UI with graphics
	Building simple interactive UI with callbacks	Sensors, Location, Persistence, Networking, Wearable, Audio, UI with dynamic containers, UI with graphics, Camera
	Application lifecycle methods overview	<All topics>
	Application and cross-appplication components communication	Sensors, Location, Persistence, Networking, Wearable, Audio, Notifications, Camera
	Building and managing dynamic user interface containers	UI with dynamic containers, UI with graphics
	Lifecycle of dynamic user interface containers	<All topics>
	Arrays of data as interactive list UI	Persistence, Networking
	Controlling background service from the main application	Sensors, Location, Networking, Audio
	Sending and receiving system wide events	Sensors, Location, Persistence, Networking, Audio
	Tools for persisting small data across lifecycle events	Persistence, UI with dynamic containers, UI with graphics
	Creating pop up dialogs	UI with dynamic containers
	Registering apps to handle OS features	Persistence
	Reading/modifying calendar entries, phonebook entries and other system resources.	Persistence
	Building user interface container for settings	UI with dynamic containers
	Creating notifications	Notifications
Model View Controller	Model View Controller overview	Sensors, Location, Persistence, Networking, Wearable, Audio, UI with dynamic containers, UI with graphics,
	MVC in the context of the platform	Sensors, Location, Persistence, Networking, Wearable, Audio, UI with dynamic containers, UI with graphics,
	MVC and dynamic user interface containers	UI with dynamic containers, UI with graphics
	Overview for MVC strategies in the context of app lifecycle	Sensors, Location, Persistence, Networking, Audio, UI with dynamic containers, UI with graphics,
	How to retain C and M across lifecycle events	Persistence, Networking, Audio, UI with dynamic containers, UI with graphics
	Establishing callbacks across M, V and C	Networking, UI with dynamic containers, UI with graphics,
	Background threads and MVC	Networking, Audio, UI with dynamic containers, UI with graphics
	Strategies for keeping the View responsive	Sensors, Location, Persistence, Networking, Audio, UI with graphics
	Long running services and MVC	Sensors, Location, Networking, Audio
	Communication of threads across M,V and C	Sensors, Location, Networking, Audio, UI with graphics
	Handling continuous communication between M,C and C during app lifecycle events	Sensors, Location, Networking, Audio, UI with graphics
Asynchronous Programming	Threads overview	Sensors, Location, Persistence, Networking, Wearable, Audio, UI with graphics
	Implement background operations for quick tasks	Sensors, Location, Persistence, Networking, Wearable, UI with graphics
	Running background tasks in parallel	Sensors, Location, Persistence, Networking, UI with graphics
	Implementing long running background service	Sensors, Location, Networking, Audio, UI with graphics
	Communication with main application thread	Sensors, Location, Persistence, Networking, Audio, UI with graphics
	Application lifecycle and threads	Sensors, Location, Networking, Audio, UI with graphics
	Build a long running background service with periodic updates	Sensors, Location, Networking, Audio
	Making background service start on reboot	Sensors, Location, Networking

Table 1. Application of the model in a mobile class. It lists the sub-areas within each of the 3 areas of the model and (repeated) topics covered in the subareas.

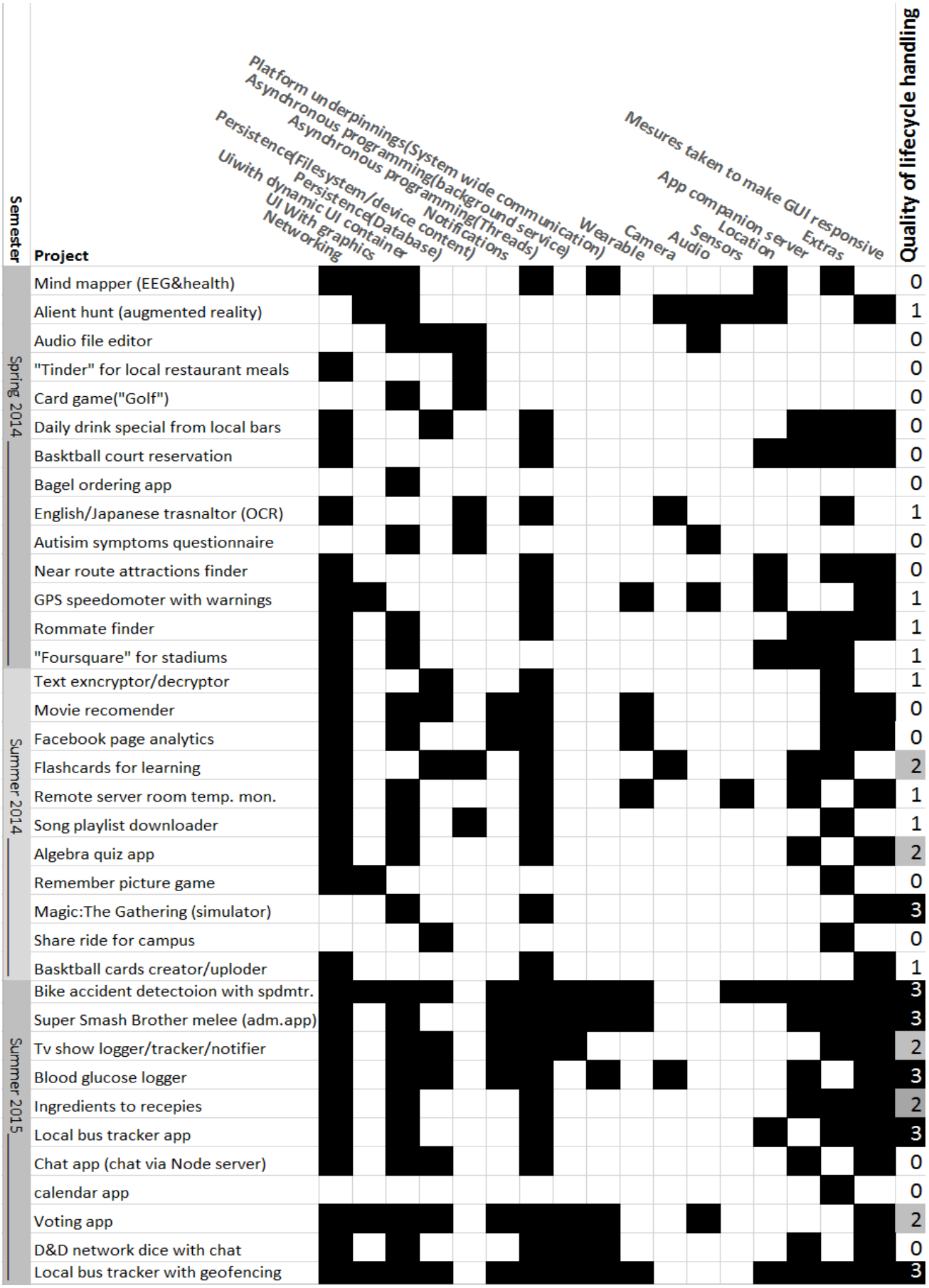

Table 2. Term projects and the degree of integration in terms of lifecycle handling (rated on a scale from 0 to 3). Each black box shows a component integrated into the project.

b) Assignments and in-class activity:

Allowed students to explore ways that long-running operations can be executed in the background. The homework assignment asked students to take the skeleton interface from the previous assignment and incorporate mock session-based log in functionality using background threads to imitate network calls and MVC design for the app behavior for situations when login is successful, unsuccessful, and expired.

4) Week 4 - Platform Foundation & Model View Controller & Asynchronous programming & Networking & Persistence & Camera:

a) Lectures.

Explored networking and data persistence in the context of our "orbit of core mobile concepts", including Android techniques to establish HTTP connections (HttpUrlConnection) and Bluetooth. These topics reflected the importance of following MVC with networking operations and the role that background threads play in establishing this pattern. For data persistence we covered Android's content providers, mechanisms for persisting small textual data throughout application lifecycle (Bundle, SharedPreferences), Android's SQLite database, and file storage.

b) Assignments and in-class activity:

Provided students with example NodeJS server set up to handle GET/POST requests and a simplistic Mongo database for simple CRUD operations. Students practiced full stack development with client mobile devices, establishing two-way server communication. To exercise this, students practiced taking pictures, inserting image data into JSON and then uploading it to server. The homework assignment for this week asked students to continue with

the app from the previous assignment, replacing mock network calls with actual ones and establishing two-way communication with the actual server that is part of statewide health intervention project.

5) Week 5: Platform Foundation & Model View Controller & Asynchronous programming & Smartwatches and sensors:

a) Lectures.

Developing for smartwatches, integrating them with Android device apps. We chose two smartwatch platforms: Android Wear and Pebble. Students practiced basics of GUI, interactions, communication with smartphones and sensors. We also taught sensors on Android devices, including common sensors, location services, and ways in which to integrate background step counter and geo-fencing capabilities.

b) Assignments and in-class activity:

Practiced basics of interactive interfaces on Android Wear and Pebble, and communication between smartwatch and smartphone. Students also practiced reading sensor values (including GPS) on Android. The homework assignment asked students to either continue refining the app from the previous homework or to build a thematically related application on Pebble and Android Wear (i.e., supporting a calorie counter for a smartwatch that communicated with the phone, or incorporating a background service to check for server updates). 23 students opted to do the smartwatch version of the homework.

6) Week 6:

The last week focused on homework assignments and term projects.

VI. RESULTS

To assess model effectiveness in evaluation and comparison, we used the model to compare student's term projects from the past three semesters. It should be noted that in Spring 2014 and Summer 2015 students implemented their projects in groups, whereas in Summer 2014 students worked individually.

We assessed projects based on the quality of implementation and quality of app and degree to which it conformed to students' term project plans, we did not grade on the sole number of different components. Nonetheless, during each semester we committed progressively more efforts to encourage students to incorporate more topics into their projects. As a result during Spring 2014 the average number of features (as seen presented in Table 2) was 4.78, 4.91 for Summer 2014 and 8.93 for Summer 2015. In many cases (50% in Spring 2014, over 63% in Summer 2014 and over 66% in Summer 2015) students incorporated features that we had not covered in class, thus going beyond the scope of what was learned in class. Some example features included Google Maps integration, integration with various APIs (such as Facebook, Twitter, Fitbit).

Our evaluation focused on assessing quality of integration of various components (see Table 2). Specifically, we rated quality in handling application lifecycle events across the component sets. Since the application lifecycle could disrupt the entire application if its components are not appropriately organized (e.g., separating View from Controller) and if students do not apply their knowledge regarding meaningful integration of components into a multicomponent setting. Our ratings of quality of integration are on 0 to 3 scale and representative of the degree to which the project complies with official guidelines for Android:

0: no explicit steps taken to address application lifecycle events

1: application recreates data but loses references to GUI related background operations during lifecycle events.

2: app has a persistent component for maintaining background operation references. (e.g., Retained Fragments in Android OS not affected by lifecycle events and are

officially recommended [15] to be used for handling lifecycle events)

3: exceeds 2 by also utilizing persistent component for retaining large data sets (e.g., for large arrays) across lifecycle events.

The number of components that the projects contained indicate the scope of the project and the scale of efforts needed to handle lifecycle events. Note that the results show that with every semester students continued to improve quality of integration of various components and the number of components as we moved in the direction of applying the model. During "Summer 2015", student projects yielded the highest number of 3's and 2's (Table 2) suggesting the usage of model being correlating with improving student's improved understanding of the material (assessed based on the quality of handling the lifecycle events) .

VII. DISCUSSION

Although we were aware of the importance of the quality of integration of app components in the first semester of the analysis, the projects from Spring 2014 had poor component integration (as seen in Table 2). We attribute this to a focus on teaching large numbers of topics without emphasizing integration. As the teaching material was modularized and the modules were decoupled, activities and homework assignments could be tailored be more topic- and module-specific. In early semesters, the cross-topic core areas were taught in the first two weeks and never explicitly revisited again. Students did perform well within those module assignments, but their term projects often failed to connect the materials. The Summer 2014 class reused the class material and assignments from the Spring 2014 class, however due the nature of the class (14 students in during Summer 2014 with daily lectures vs 63 in Spring 2014 with lectures two times per week) we were able to spend more time with students individually and therefore spend more time helping them with their term projects. As a result, students demonstrated better component integration in their projects, but with room for improvement. These observations from the Spring and Summer 2014 classes encouraged us to revisit our teaching strategy.

Prior to teaching the Summer 2015 class we had prepared our model. We used it refine class lectures, assignments and in-class activities. Although the topics that we included for the class were the same as during previous two classes, the topics were not isolated: since the core cross-topic areas required more time to learn, we revisited them throughout the semester. The assignments not only introduced new topics but also kept a central theme and required feature integration. As a result, students practiced not only the topics but also the core cross-concept areas—exercising integration of features into one app. We attribute the sharply increased number of features and improved integration in projects to the homework assignments that helped them understand cross-topic integration.

Further analysis of term project apps beyond just the quality integration of various features will paint a richer picture on how the application of our model is capable of altering students' performance. Nonetheless, the results that we obtained do show that the model has a potential to affect the quality of software that students produce, since many of them demonstrated understanding of nuances needed for reliable integration of various components into multicomponent apps.

From an instructor's perspective the model reinforces focus on the cross-topic areas. When adding a new topic, the model suggests revisiting the "ring" to understand platform-specific, MVC, and asynchronous programming aspects of the topic. As hardware evolves, the model still encourages reflection on these core areas, helping the instructor situate the new topic in an integrative way.

VIII. CONCLUSIONS AND FUTURE WORK

This paper proposes an adaptable platform-agnostic model for teaching mobile app development. The model emerged from extensive literature review and multiple semesters of experience in teaching mobile software development at the junior/senior level. The goal of the model is to help educators teach mobile programming topics appropriate to the current and desired skill levels of their students by placing emphasis on the cross-topic areas needed to understand how to integrate the knowledge and skills on a topic in the context of a multi-component app.

We demonstrated how to adapt this model in the context of a six-week-long mobile software development junior/senior level class.

To assess the effectiveness of the model we evaluated student projects based on number of topics integrated and the quality of integration of the components, contrasting results with previous iterations of the class. Results show students taught from this model produced projects with more and better components integration resulting in higher quality software. As such, this paper offers multiple contributions. It shows cross-topic areas that are repeated in the context of different mobile specific topics. Additionally, it proposes a way to consolidate the cross-topic areas into three distinct categories and define various mobile specific topics in terms of the degree to which each of the three cross-topic categories is part of the teaching process of the given topic. Finally, it shows in detail how this model can be applied.

Considering the adaptive nature of the model, in the future we want to explore its effectiveness in a wide context of different situations, from one-week modules on select topics to advanced, twelve-week classes aimed at experienced mobile developers to high school outreach, as this model has not been tested and validated outside of our institution. We welcome other collaborations as well. At the moment of writing this paper, one other university expressed interest in adapting this model for their class.

IX. REFERENCES

[1] (2015). *Gartner Report*. Available: http://www.gartner.com/newsroom/id/2996817

[2] (2015). *Mobile Technology Fact Sheet*. Available: http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/

[3] M. H. Dabney, B. C. Dean, and T. Rogers, "No sensor left behind: enriching computing education with mobile devices," presented at the Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA, 2013.

[4] J. Andrus and J. Nieh, "Teaching operating systems using android," presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education, Raleigh, North Carolina, USA, 2012.

[5] M. Guo, P. Bhattacharya, M. Yang, K. Qian, and L. Yang, "Learning mobile security with android security labware," presented at the Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA, 2013.

[6] A. Esakia, S. Niu, and D. S. McCrickard, "Augmenting Undergraduate Computer Science Education With Programmable Smartwatches," presented at the Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, Missouri, USA, 2015.

[7] H. Chen and K. Damevski, "A teaching model for development of sensor-driven mobile applications," presented at the Proceedings of the 2014 conference on Innovation & technology in computer science education, Uppsala, Sweden, 2014.

[8] A. J. Gordon, "Concepts for mobile programming," presented at the Proceedings of the 18th ACM conference on Innovation and technology in computer science education, Canterbury, England, UK, 2013.

[9] K. Sung and A. Samuel, "Mobile application development classes for the mobile era," presented at the Proceedings of the 2014 conference on Innovation & technology in computer science education, Uppsala, Sweden, 2014.

[10] B. Burd, Jo, #227, o. P. Barros, C. Johnson, S. Kurkovsky*, et al.*, "Educating for mobile computing: addressing the new challenges," presented at the Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups, Haifa, Israel, 2012.

[11] D. Riley, "Using mobile phone programming to teach Java and advanced programming to computer scientists," presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education, Raleigh, North Carolina, USA, 2012.

[12] Q. H. Mahmoud, T. Ngo, R. Niazi, P. Popowicz, R. Sydoryshyn, M. Wilks*, et al.*, "An academic kit for integrating mobile devices into the CS curriculum," *SIGCSE Bull.,* vol. 41, pp. 40-44, 2009.

[13] X. Y. e. al., "Teaching mobile computing and mobile security," *ed:FIE,* 2016.

[14] V. P. Pauca and R. T. Guy, "Mobile apps for the greater good: a socially relevant approach to software engineering," presented at the Proceedings of the 43rd ACM technical symposium on Computer Science Education, Raleigh, North Carolina, USA, 2012.

[15] (2015). *Handling Runtime Changes*. Available: http://developer.android.com/guide/topics/resources/runtime-changes.html