# Blending an Android Development Course with Software Engineering Concepts

**ARTICLE** *in* EDUCATION AND INFORMATION TECHNOLOGIES · JULY 2015

DOI: 10.1007/s10639-015-9423-3



**4 AUTHORS**, INCLUDING:

Alexander Chatzigeorgiou

University of Macedonia

**145** PUBLICATIONS **1,079** CITATIONS

SEE PROFILE

George Violettas

Technical Trainers College, Riyadh, S.Arabia

**10** PUBLICATIONS **3** CITATIONS

SEE PROFILE

Stelios Xinogalos

University of Macedonia

**53** PUBLICATIONS **128** CITATIONS

SEE PROFILE

# Blending an Android development course with software engineering concepts

**Abstract** The tremendous popularity of mobile computing and Android in particular has attracted millions of developers who see opportunities for building their own start-ups. As a consequence Computer Science students express an increasing interest into the related technology of Java development for Android applications. Android projects are complex by nature and relatively large software products while their development calls for the application of established software engineering practices and tools. However, most software engineering courses focus on 'conventional' software development for desktop or web applications. In this paper we report on the design, implementation and assessment of a novel short course aiming at bridging the gap between software engineering and Android development. The goal is to demonstrate the need for applying software engineering principles on Android development as well as to emphasize that writing software for mobile devices should be regarded as an equally serious programming activity. The proposed course covers design principles, patterns, metrics, refactorings and collaborative software development. The course has been delivered to three groups of undergraduate and postgraduate students at two different institutes. The course has been evaluated: a) by performing a student satisfaction survey, b) through summative assessment of students' performance, c) by investigating whether the proposed course modified the students' career interests and d) by employing assessment by peers based on rubrics. The results indicate that such a short course is capable of increasing student's interest on Android development as well as their awareness of the importance of software engineering concepts on mobile application software development.

**Keywords** Software engineering, Android development, Teaching strategies, Classroom teaching

## 1. INTRODUCTION

Today Android is a very common development platform and its share is continuously rising (Whitney 2012). According to Gartner (2013) tablet shipments has grown by 53.4% in 2013 with shipments reaching 184 million units, with Android accounting for 38% of shipments. All device shipments excluding PCs for 2013 were 2.2 billion. From these figures it can be easily concluded that this is a vast market and that a very large percentage of mobile devices of any kind (smartphones, mobile phones, tablets etc.) are operated by Android OS. In the same study it is stated that "…the mobile phone market will continue to experience steady growth…" and "…Android will remain the leading device OS…". "Google Play", the Android's Software Application Market reached over 1 million apps on July 2013 with over 50 billion downloads (Victor 2013).

A direct consequence of the tremendous proliferation of mobile devices is an analogous exponential rise in the demand for skilled mobile application developers. Various studies emphasize the continuous shortage of skilled computer science graduates needed to fill vacancies in the mobile applications market. According to the European Vacancy and Recruitment Report for 2012 (European Commission 2012), mobile application development has been identified as one of the "top bottleneck occupations" (i.e. occupations with increased demand and limited supply) in the field of Information and Communication Technologies. According to another study of Mulligan and Card in 2014 (Mullingan and Card 2014), developers of applications targeting mobile and social platforms took in $23.7 billion in 2013 with a forecast of $85.3 billion for 2018.

To address this shortfall, tertiary education institutes around the world are adapting their curricula and accommodate courses on mobile application development, with Android being the OS of choice, mainly because development is based on the Java programming language, which constitutes a core course in most universities. Nevertheless, successful mobile application development requires a background not only in Android specifics pertaining to the Android libraries and the limitations imposed by mobile devices, but also in more general software engineering knowledge. Software projects aiming at the development of competitive mobile applications require the application of various principles and techniques in order to achieve high-quality, maintainable, reusable, testable and comprehensible software. An overview of software engineering research issues related to the emerging field of mobile application development can be found in the study by Wasserman (2010).

Computer Science and Information Technology curricula include software engineering as a key Knowledge Area (ACM 2013) and as a result most computer science related departments offer at least one "conventional" software engineering course. By conventional we mean that in most cases the introduced concepts are exemplified on either desktop applications standing individually on a personal computer, and recently on web applications employing server and/or client side development. To the best of our knowledge there are very few courses that introduce software engineering concepts to a mobile software development course. At the same time, software development for particular platforms is often perceived and taught as a purely technological skill with no direct connections to software engineering. The consequence is that even experienced mobile application programmers do not embrace or apply software engineering practices. Moreover, the lure of Android development can serve as a vehicle for conveying software engineering concepts to Computer Science students in an attractive way.

In this paper we introduce a short course aiming at illustrating selected software engineering principles, concepts and techniques in the context of Android application development. The goal is to demonstrate that Java programming for Android applications -which is usually perceived as an isolated activity- can benefit largely from applying software engineering best practices. The course is structured around a base Android application that is gradually enhanced by discussing limitations or opportunities that can be addressed by appropriate techniques or guidelines such as design principles, design patterns, refactorings, software metrics, testing, version control systems etc.

The proposed course has been delivered to three different student groups (two groups of undergraduate and one group of postgraduate students) at two different higher education institutes, in two countries. Students had a mixed prior knowledge on the Java programming language, Android development and software engineering allowing an assessment of the weaknesses and strengths of the proposed course. The course is supported by a customized Content Management System (described in Section 4) offering all versions of the examined application which can be developed either in a sequential manner or start from the appropriate base point, based on previous experience.

An attempt to evaluate the proposed course employing multiple sources of data has been performed. In particular, the course has been evaluated in four ways, of which the first three correspond to the first three levels in evaluating training programs (Kirkpatrick and Kirkpatrick 2006). In order to investigate students' satisfaction and their perception of the value of the course contents a survey has been delivered capturing prior knowledge levels and opinion on different aspects of the course. Similarly to other approaches that introduced a novel course we employed also summative assessment at the end of the course to investigate how well the learning goals have been attained. As a third way of assessing the course we have studied whether the course has modified students' interest in careers related to Android development and software engineering. Finally, we have sought the opinion of experts in pedagogy regarding the design and implementation of the course employing a standardized rubric.

The rest of the paper is outlined as follows: An overview of related work on Android development and software engineering education is provided in Section 2. Section 3 presents an outline of the proposed course; the detailed structure along with the introduced software engineering concepts at each step. The accompanying web page that supports the proposed course is presented in Section 4. Section 5 focuses on the evaluation of the course by presenting the applied methodology as well as the findings for each type of assessment. A discussion on the main challenges in the implementation of the proposed course, the adherence to curriculum guidelines, as well as the interpretation of the evaluation results is outlined in Section 6. Finally, we conclude in Section 7.

## 2. RELATED WORK

As already mentioned, despite the tremendous popularity and adoption of the Android OS, teaching of application development in Android is considered to be still very limited in most tertiary education institutes around the globe: "…*Given the newness of the platform* (android), *no textbook currently exists to teach Java or any other upper-division courses using Android examples. Further, most Java language textbooks are intended for CS1/CS2 courses*.." (Riley 2012).

The integration of mobile devices into the computer science curriculum has been discussed in a study by Mahmoud (2008) where the need to consider the important factor of restricted resources (small screen,

limited memory etc.) is stressed. Moreover, the author highlights the difference of compiling an application for a "normal" PC and for a mobile device. It is claimed that introducing mobile devices and the basics of mobile application development as early as possible in introductory programming courses is a necessity. In the context of the described course the students were asked to develop two versions of a mortgage calculator application: one for a desktop and one for a mobile platform so they could distinguish the differences (and similarities) between the two approaches.

A different approach is adopted in the work of Akopian et al. (2013) where the authors rely on already developed templates (android programs) to teach in a short course the basic principles of Android development. The students were asked to alter specific aspects in the program by modifying the existing code. The questions and activities have been very carefully structured and explained, in order to guide the student to the correct position of variable or code snippet that needs to be changed. Through structured colored examples the activities for the students are slowly escalating to the most difficult ones. According to one pre-course and one post-course survey, student's satisfaction has been achieved along with a strong belief that basic skills have been obtained through the course.

Although the structure of that course has similarities to the proposed Android short course, it should be mentioned that it has been designed for electrical engineering departments, with significant time constraints on programming courses. As a result its content is rather simplistic, and more importantly it is not addressing any software engineering issues in the context of Android development.

Evidence from evaluations performed by Heckman et al. (2011) implies that teaching lower level programming courses with more advanced and current technologies such as mobile devices can be beneficial. In particular, the authors taught Java and software engineering courses at the University of Virginia and North Carolina State University utilizing the Android OS platform. Although details about the course are not provided, the goals are similar to the proposed course in this paper, i.e. to teach both Java (such as abstraction) and software engineering concepts (such as design, testing and patterns) using Android.

Petkovic et al. in 2006 (Petkovic et al. 2006) claimed that practical teaching of software engineering should also focus on teamwork, communication skills and organizational issues to reflect the globalization and open-source aspects of real software development projects. The authors recommend that students should be divided into small groups of four to six persons in order to simulate real life software development and undertake the task of constructing a web application through five milestones. The course also emphasizes the need of collaboration without face-to-face meetings and to this end the use of version control systems is essential.

The main motivator for their work was that the combination of traditional classroom teaching of processes, teamwork and communication with a major final project, intensive instructor interaction, and realistic simulation of real software lifecycle is critical. The benefits of cooperative learning where students work together in small groups enhancing each other's learning have been recognized early in the computer science education community (Tenenberg 1995). The same belief is shared by the authors of the current work and has been taken into account during the design of the proposed course. In our case students have been also divided into small groups as described in (Petkovic et al. 2006), with the exception that they were left alone to form the groups. This approach seemed more appropriate to the Arabic or the Greek cultures, where students tend to lean towards the existing bonds among them.

The challenges of designing a smartphone software development course based on Android have been discussed by Hu et al. (2010). The paper introduces a full semester course with an emphasis on the underlying platform: the first two parts of the course consist in an introduction to smartphones and their OS. The next three parts are presenting the environment for Android development, while only the seventh part is devoted to software development. The authors are acknowledging three design principles with the second stating that the teaching content should place an emphasis on practice rather than theory and the third mentioning that course contents should meet the needs of the industry.

Some of the major challenges that the authors have mentioned and have also been faced during the teaching of the proposed course were the time of the course needed to be spent discussing the nuances of the OS and the special Android-centric visualization, the ever-changing landscape of mobile OSs and the unavoidable continuous change of Android plugins, libraries, versions etc.

The short course that is proposed in the paper in hands is in accordance with the findings of the Joint Task Force on Computing Curricula of the IEEE Computer Society and the Association for Computing

Machinery (ACM) (LeBlanc and Sobel 2004), where a set of guidelines for effective education activities have been established. These guidelines emphasize the need to focus on professional issues necessary to begin practice as a software engineer, the need for students to complete tasks that involve both individual work as well as tasks that require collaboration among peers.

## 3. THE COURSE

### 3.1 Learning Outcomes and Course Structure

The proposed course (which can be delivered either in an advanced undergraduate or a postgraduate program) aims at introducing software engineering concepts and techniques in the context of Android development. In case students have a background in both areas (software engineering and Java/Android development) the objective is to highlight that mobile application development can benefit from the adoption of best practices as learned in a software engineering course. In case students lack background in any of the involved areas, the course can also serve as an introductory seminar for the corresponding topic, given that it includes theoretical aspects as well as lab activities.

The short course is in agreement with the 2013 curriculum guidelines for undergraduate programs in Computer Science by ACM (2013) where Platform-Based Development (PBD) has been elevated to a Knowledge Area (KA) in the Computer Science Body of Knowledge. The guidelines suggest that electives should be offered (in the area of Programming Interactive Systems) covering the needs for specific programming paradigms and mobile development and these are designated as "high demand" courses. Within the ACM report it is noted that Platform-based development as a general skill of developing with respect to an Application Programming Interface (API) or a constrained environment is directly related to other Knowledge Areas such as Software Development Fundamentals (SDF). As a result, the relationship between mobile development and software engineering is profound.

According to the constructive alignment approach (Biggs and Tang 2011) it is important to start reasoning about the outcomes that a planned course aims at achieving and then align the teaching and assessment strategies to those explicitly stated outcomes. Learning outcomes are statements of what a learner is expected to know, understand and/or be able to demonstrate after completion of a lecture, course or entire program. Compared to teaching aims or objectives, outcomes offer the advantage of being more precise, clearer and easier to compose (Kennedy et al. 2006).

We formulate the learning outcomes of the proposed short course as follows, placing them in the appropriate knowledge and cognitive process dimension of the revised Bloom's taxonomy (Krathwohl 2002) in Table 1:

(1) Explain the benefits from applying software engineering practices to Android development.
(2) Apply Design Principles and Design Patterns in Android applications.
(3) Identify and Resolve design issues in Android applications.

**Table 1** Course Learning Outcomes in terms of the Revised Bloom's Taxonomy

| **Knowledge Dimension** | **Cognitive Process Dimension** | | | | | |
|---|---|---|---|---|---|---|
| | **1. Remember** | **2. Understand** | **3. Apply** | **4. Analyze** | **5. Evaluate** | **6. Create** |
| **A. Factual Knowledge** | | (1) | | | | |
| **B. Conceptual Knowledge** | | (1) | (2) | (3) | | |
| **C. Procedural Knowledge** | | | (2) | (3) | | |
| **D. Metacognitive Knowledge** | | | | | | |

Informed choices regarding the curriculum have to clearly define the desired Information & Communication Technology (ICT) skills, besides the underlying knowledge area(s) and learning outcomes. A tool that can be used for this purpose is the Skills Framework for the Information Age (SFIA), which provides 6 categories and 19 sub-categories including 96 ICT specific skills in total (SFIA Foundation, 2015). These skills are defined quite generally and independently of the underlying

knowledge areas, so that they are transferable to different ICT domains/disciplines. In addition to the general definition, SFIA provides a definition of each skill across the following 7 levels of responsibility (that apply in each case): (1) follow; (2) assist; (3) apply; (4) enable; (5) ensure, advise; (6) initiate, influence; (7) set strategy, inspire, mobilise. Utilizing a tabular approach (Herbert et al. 2013) the main SFIA skills that the proposed course aims at are presented in Table 2.

**Table 2** Targeted SFIA Skills of the Course

| **Category ~ Subcategory** | **Skill** | **Level** | **Code*** |
|---|---|---|---|
| Solution development and implementation ~ Systems development | Programming/software development (*The design, creation, testing and documenting of new and amended Android programs*) | 2 – assist<br>3 – apply<br>4 – enable | PROG |
| Solution development and implementation ~ Systems development | Systems design (*The specification and design of information systems to meet defined business needs. The identification of concepts and their translation into implementable design*) | 2 – assist<br>3 – apply<br>4 – enable | DESN |
| Strategy & architecture ~ Technical strategy and planning | Software development process improvement (*The provision of advice, assistance and leadership in improving the quality of software development, by focusing on measurement*) | 5 – ensure, advise<br>6 – initiate, influence | SPIM |
| Strategy & architecture ~ Technical strategy and planning | Methods & tools (*Ensuring that appropriate methods and tools for the development, testing and maintenance of systems are adopted*) | 4 – enable<br>5 – ensure, advise | METL |

*Refers to the unique skill code in the SFIA Framework

The course is structured around a base Android application (which can be initially downloaded from the accompanying Web page) that is gradually enhanced over 13 consecutive steps. At each step a different software engineering concept, practice or technique is introduced which is then exemplified through additions to the functionality of the application or restructuring of its design.

In accordance to Keller's Motivation Theory (Keller 1983) the course is designed in order to promote learning by increasing students satisfaction and attention. A feeling of satisfaction is achieved by providing students with opportunities to apply on a small, but real Android application the newly learned knowledge and the interest of the students is engaged by interleaving different software engineering practices and techniques on the same example. Since the course is structured around a central lab activity which is gradually enhanced, it adheres to the learning theory of "Learning by Doing" which is one of the most well-known theories (Dewey, 1938). Instructors can decide on the importance/time that should be placed on the theoretical presentation of each software engineering concept or technique and the corresponding hands-on assignments, depending on the subjects' background. The particular contents of each step are discussed in the next section.

### 3.2 Introduced Software Engineering Concepts and Students' Activities

As already mentioned, the proposed course is structured as a series of incremental additions to an initial Android application with the goal of introducing a different software engineering principle, concept or technique each time. The outline of the course is shown in Table 3, depicting the corresponding students' activity in each step, the functionality of the resulting Android Java application as well as the introduced concept. A more detailed presentation of the introduced software engineering concepts as well as the source code for the project corresponding to each functional version can be found in the accompanying Web page1. A brief overview of the activity that the students should perform at each step is provided next.

**Table 3** Course Structure

| **Students' activity** | **Functionality of the Resulting Software** | **Introduced software engineering Concept/Technique** |
|---|---|---|

[1] http://androidse.wikispaces.com

| | | | |
|---|---|---|---|
| 1 | Downloading of initial version | Listing of future appointments in ListView | Model-View-Controller Pattern (Ghezzi et al., 2002) |
| 2 | Introduction of Android Activity | Addition of Input Screen | Externalization of Resources (Mednieks et al. 2012) |
| 3 | Development of Task hierarchy | Submission of tasks to ListView | Use of Inheritance for modeling similar entities. Liskov Substitution Principle. (Liskov and Wing 1994) |
| 4 | Intent Handling | Display of task description and date & selection of appropriate color for each task type | Use of Polymorphism. Open-Closed Principle (Martin 2003) |
| 5 | Preservation of state between invocations of an activity | Display of multiple tasks in the ListView | Singleton Design Pattern (Gamma et al. 1995). |
| 6 | Use of TreeSet structure for ordering | Ordering of Tasks based on their date | Dependency Inversion Principle (use of TreeSet class – implementation of the Comparable interface) |
| 7 | Installation of metrics plugin. Examination of metric values. | No change. Assessment of quality | Software Metrics and their use for software quality evaluation. Software Ageing. |
| 8 | Identification of code smells | No change. Assessment of quality | Code smells (Large Method) |
| 9 | Unit test preparation | No change. | Unit Testing. Regression Testing. |
| 10 | Refactoring application. Execution of unit tests | No change. Design Improvement. | Refactorings as a means to improve software. (Extract Method Refactoring) |
| 11 | Addition of code. Identification of smell. Refactoring application | Setting task description text to Title Case | Smells + Refactorings (Extract Method / Move Method) |
| 12 | Committing to a Git repository | No change. | Distributed Version Control |
| 13 | Uploading to a common remote repository. Downloading by different students, edits and merging. Viewing history | Any modification is acceptable | Collaborative Software Development |

*Step 1: MVC Pattern.* A partial illustration of the Model-View-Controller architecture (or pattern) can be achieved with the use of ListView objects in Android, which show items in a vertical scrolling list. The students' activity in this step consists in downloading the initial version of the examined Android project, importing it in the employed Integrated Development Environment and identifying the classes and layout XML files that play the corresponding roles of the MVC pattern[2]. The functionality of the initial version includes simply the listing of a set of (future) tasks that have to be done (To-do list).

*Step 2: Externalization of Resources.* Android encourages the application of a valuable practice, namely the externalization of non-code resources like images and string constants or even entire layouts. The students' activity in this step consists in the creation of a simple Activity and placing emphasis on the externalization of string values that specify the text to be displayed. After changing the introduced activity to a launcher activity, when the application starts, it displays a simplified input form for adding new tasks (without any functionality in the submit button so far).

*Step 3: Use of Inheritance for modeling similar entities*. The next activity consists in the addition of functionality to the "submit" button, so that whenever a new task is added it will be displayed on the TasksView activity. Once the "submit" button is pressed, the information concerning the added task (date, description and urgency in this simplified example) should be transferred to the ListView, along with an intent that will cause the appearance of another activity's view (Mednieks et al. 2012). All pieces of information should be embedded into an appropriate Task object, whose class (considering an appropriate hierarchy) will designate the actual type of the task. To make the distinction more clear, we assume that for urgent tasks the corresponding background in the ListView will be red (blue for normal tasks), and that the description of each task will have as prefix the urgency of the task ("Urgent", "Normal"). The functionality of the application after this step enables the user to add a new task and view the corresponding description. Only a single task can be viewed at this point.

[2] We suggest using a Java IDE like Eclipse, Netbeans or IntelliJ IDEA due to the integration of graphical layouts along with XML formats and because of the provided support for refactorings

*Step 4: Polymorphism and Open-Closed Principle.* The real benefit from the use of polymorphism becomes evident when the handling of the received intent is performed in the TasksView class to display the newly added task. The students' goal is to color the background of each task based on the type of the received task. Once the TasksView retrieves the Task object from the intent, within the adapter method responsible for filling in the corresponding line items of the ListView, the color for the background of each task should be obtained by a polymorphic method call, where dynamic binding will be employed to invoke the appropriate method of the corresponding Task subclass.

*Step 5: Singleton Design Pattern*. The application so far suffers from a rather serious drawback: only one task is displayed regardless of how many are introduced. The reason is that whenever the TasksView activity is invoked it leads to the construction of a new adapter, associated with a new data model, and previous tasks are lost. In other words there is no preservation of state between activity invocations. At this point it would be very useful to remind the notion of design patterns as a means of resolving commonly recurring problems (Gamma et al. 1995). In this case, the Singleton pattern offers an elegant solution to the problem of data preservation. The goal is to ensure that only one instance of a class is created and that a global access point is provided to that object. Turning the Model class which holds the list of tasks to be displayed to a Singleton, ensures that at any time the same, unique data model is used. In terms of functionality the application allows the display of numerous tasks.

*Step 6: Dependency Inversion Principle.* So far, the application displays tasks in the order in which they have been added. Obviously, there is a need for ordering tasks according to their date. This can be easily accomplished employing a data structure in which stored elements are automatically ordered. An appropriate Java data structure is the TreeSet where elements are retrieved based on their natural ordering. For any domain specific class the developer should specify the way in which items should be ordered by implementing the Comparable interface and overriding the compareTo() method (or alternatively by providing an appropriate Comparator). The resulting functionality at this point is a listing of tasks according to their date, i.e. closer tasks in time are displayed first. While the use of a TreeSet is a rather widely-known technique to Java developers, it would be important to emphasize that the designers of Java essentially adopt the "Dependency Inversion Principle" (DIP) (Martin 2003) in this context.

*Step 7: Metrics for Software Quality Assessment.* At this point the application has a reasonably large number of lines of code and methods. To assess the quality of the underlying design, students should install a metrics collection and analysis tool, preferably one that is integrated into the employed IDE in order to assist the next steps related to preventive maintenance. Students can obtain a first impression and reason about the properties of the design by assessing metrics related to size, complexity, coupling and cohesion.

*Step 8: Code Smells.* According to Fowler (1999) deeper problems often manifest themselves in the form of easily identifiable code smells, i.e. surface indications that something might be wrong at the code or design level. Hints for the presence of smells can be provided by excessive metric values or values that deviate from the mean. As an example it should be observed that method getView() in the TasksAdapter class is much longer than the rest of the methods, has a higher complexity and a larger number of parameters, a potential indication of a "Long Method". This is often the case when a method takes over more than one responsibilities. Although the logic of method getView() is quite simple, it contains two distinct responsibilities: a), the inflation of a new view object from the corresponding XML resource, and b) the population of this view based on the information retrieved from a task object.

*Step 9: Unit Testing.* At this step students are asked to generate test cases for the TasksAdapter class, prior to any attempt of improving the class structure. The testing framework of choice to support the preparation and execution of test cases is JUnit (2014) which is an open-source, widely acknowledged and easy-to-use unit testing environment. In this particular case, the goal is to test the behavior of the getView() method of the TasksAdapter class.

*Step 10: Refactoring Application (Extract Method).* Now that the source code is equipped with test cases, students should attempt to improve the design by applying appropriate refactorings. A refactoring is a way to improve the internal structure of software without changing its observable behavior (Fowler 1999). In most cases, refactorings are applied in order to resolve identified code smells. After any change,

test cases can serve as a check point against which functionality can be verified. For the "Long Method" smell the appropriate refactoring would be to split the existing method into two or more cohesive methods. According to Fowler's catalog, this corresponds to the "Extract Method" refactoring, whose goal is to "*turn a fragment of code into a method whose name explains the purpose of the method"*. For the getView() method, the functionality related to the retrieval of information from the Task object in order to populate the view's contents, should be extracted as a separate populateView() method, which will be invoked by the original method.

*Step 11: Refactoring Application (Extract Method + Move Method).* As software evolves, additional functionality is gradually added to address the needs of clients. We assume that the populateView() method of the TasksAdapter has been enhanced to display the description of the forthcoming tasks in title case. The method's code has once again become large and relatively complex and since the conversion of a string to title case is a rather distinct responsibility, it should be extracted as a separate method by applying the Extract Method refactoring as previously. However, in this case the extracted method convertToTitleCase(Task task) has nothing to do with the TasksAdapter class in which it resides. The method accesses only methods from another class and this is a clear symptom of the "Feature Envy" smell (Fowler, 1999). The appropriate refactoring is the "Move Method" which consists in moving the method to the target class (Tsantalis and Chatzigeorgiou 2009). In this case the convertToTitleCase() method should be moved to the Task class, where it also belongs conceptually.

*Step 12: Distributed Version Control Systems.* Software systems are multi-version projects that continuously undergo maintenance. Even the simplified example outlined here has evolved over a number of versions. Version Control Systems (VCS) refer to software that allows the management and monitoring of changes that occur in any artifact of a project during its initial development or maintenance (Spinellis 2003). Due to its widespread adoption, emphasis is placed on Distributed VCS such as Git (2014). At this step of the course, students are asked to install the EGit Eclipse plug-in, create a Git repository and commit the project to the repository[3]. Students are asked to enter a log message that will be attached to each individual commit. Once the commit is performed, students will be able to track the history of the project. Students can be introduced to the concept of the main development branch (master), the HEAD pointer referencing the branch that the user is currently on, the importance of good commit messages and the unique identifier assigned to each Git commit object.

*Step 13: Collaborative Software Development.* Contemporary software development teams consist of members which often are not located in the same geographical area and might even be working at different time zones. To enable the collaboration between multiple developers, VCSs require the use of a common, remote repository that can be used for sharing software artifacts. A popular hosting provider for Git projects is Github (2014) and can be used for the proposed course since public repositories are hosted for free. The instructor (or any student) can create a repository and add fellow students as collaborators to the project. Students can then start experimenting with collaborative software development based on the project that has been developed. For example, the instructor can push his own project to Github and allow students to clone the existing version. Students can simulate the resolution of conflicts and merging of changes by editing files and pushing to the remote repository.

*Further or Alternative Steps.* Obviously there is no need for an instructor to strictly follow the order, number and content of the steps that have been presented. Based on the students' prior knowledge and the particular goals of each curriculum, the course can be modified accordingly. As an example, the course could be extended to a full semester course by adding an introduction to the basics of Android development as well as additional steps illustrating the use of additional software engineering concepts or more cases of the already examined ones (i.e. more design patterns, metrics and refactoring types). Another axis along which the course can be extended concerns the functionality of the application itself, which can be turned into a much more complicated one, involving 2D graphics, storage in a database and communication with a server. However, it should be borne in mind that the essential goal of such a

[3] Details about EGit usage can be found at the corresponding EGit documentation http://www.eclipse.org/egit/documentation

course is not an introduction to Android programming per se, but rather the leveraging of well-established software engineering practices to the benefit of mobile application development.

## 4. ACCOMPANYING WEB PAGE

In order to support the teaching process and the communication among students and instructor, a wiki page has been developed for the proposed course. "Wikispaces" (www.wikispaces.com) has been selected as the development environment. Wikispaces provides teachers and educators with a wiki suitable for classroom use as it provides rich functionality for content distribution, class management, discussion forum, customized security and project/homework collection. The web address for the proposed course is http://androidse.wikispaces.com and is open for any member of Wikispaces, requiring no further permissions.

In general, a wiki is a web application that gives to the users the ability to collaborate on-line through the usage of a number of functionalities such as collaborative content editing and participation in discussion forums (Parker and Chao 2007). Wikis can enhance the educational process with new opportunities for developing online interaction through a simple, flexible and open technology promoting sharing of knowledge, open discussions and exchange of ideas. Moreover, Wikis enable peer and self-assessment as they provide strong support for on-line classroom management (Schwartz et al. 2004).

Utilizing a wiki, students master various skills through their collaborative activities and peer learning. Furthermore, wiki-based activities are an effective way to develop teachers' abilities. The same applies to students, who are often eager to spend additional effort on the wiki tasks, as long as tutors are willing to provide guidance (Lai and Ng 2011).

The wiki page that has been developed for the proposed course can be viewed from the perspective of a content delivery system and that of an online examination platform.

### 4.1 Website description

Through the main page of the website the user can access four main options (Figure 1):

1. Course Objectives: Information about the main objectives and goals of the course.
2. Software Infrastructure: Description of required software and illustration of the steps for downloading, installing and setting up the development environment in order to conduct the practical assignments.
3. Course outline: This is the backbone of the webpage. It consists of 13 hyperlinks to webpages that describe each one of the consecutive steps of the course. For each step of the course a specific webpage (Figure 2) provides an overview of the objectives for each step, an explanation on how these objectives can be achieved, additional educational material such as PowerPoint presentations as well as a link to download the appropriate source code example that is needed for the corresponding step.
4. Assessment: Contains the assessment for the course. The user has two options:
    a. download the exam as a document file that can be printed and answered offline,
    b. connect to the on-line examination system described next

Fig. 1. Wikispaces course main page

In addition, the webpage provides a forum allowing online discussion where participants can hold conversations, post messages and questions regarding the course. For operational reasons posting messages to the forum requires authentication from the wikispaces account manager.

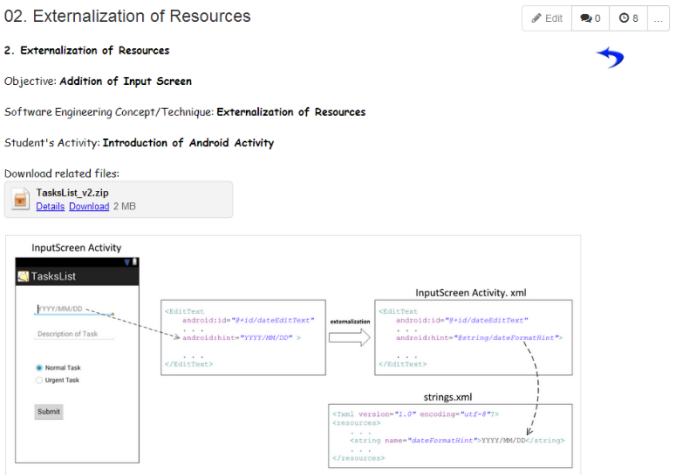

Fig. 2. "Externalization of Resources" step of the course with the download link for the corresponding code. Each step of the course has a dedicated webpage

### 4.2 On-line exam description

Assessment can be conducted through an online examination system operating over the web. An on-line examination offers several advantages such as 24/7 availability, duration tracking, automatic correction and immediate feedback for correct answers, from the perspective of the participant. From the side of the course organizers major advantages consist in the ability to compare subsequent results, obtain automated reports and retrieve and analyze statistical data.

In order to develop and conduct the online assessment in this course, the "*ThatQuiz*" online examination system has been used. "*ThatQuiz*" is a free testing service that can be used from the

instructors to build and conduct on-line assessments for their students (available through http://www.thatquiz.org). When the course participants decide to take the test, following the online assessment link in the wiki page will redirect them to the "*ThatQuiz*" Android online examination. The examination consists of multiple choice questions as shown in Figure 3. When the exam is completed, the examinee receives the result and a report on the wrong and correct answers. The exam is anonymous but the results are stored for further statistical analysis and evaluation.

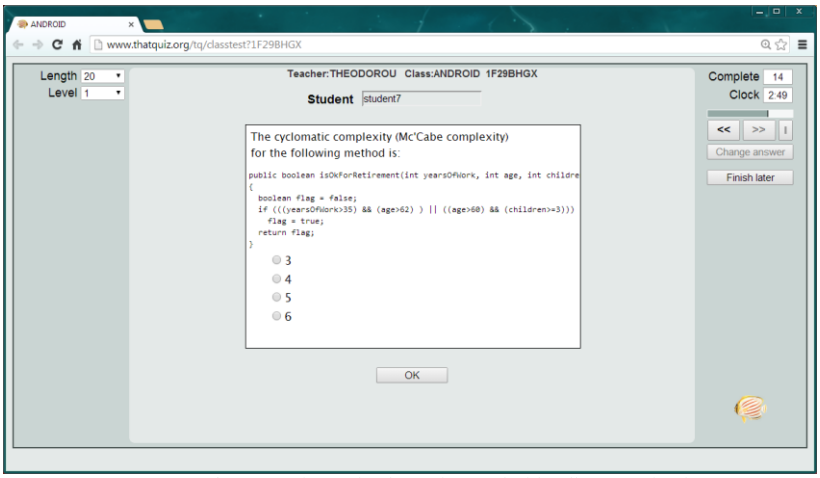


Fig. 3. One of the questions of "ThatQuiz" Android online examination.

## 5. COURSE EVALUATION

### 5.1 Methodology

The goal of the study presented in this paper was to investigate the attractiveness and effectiveness of a short course that aims at bridging the gap between software engineering and Android development. The educational goal of the proposed course was to train students in integrating software engineering principles in Android development. It should be noted that due to the nature of the course (i.e. for two of the groups the course had the form of a short seminar which was embedded in other courses) full evaluation of the course was not possible in all cases. Nevertheless, we have attempted to investigate the following questions, even in the presence of responses obtained from one group of students:

Question 1: *How do students evaluate the proposed course*?
Question 2: *What are students' achievements with respect to the intended learning outcomes*?
Question 3: *How does the course affect students' interest on Android development*?
Question 4: *How do instructors evaluate the course*?

The course was delivered to three different groups of undergraduate and graduate students during the winter semester of 2013-2014, at two institutes. At Institute 1 the short course was independently taught (as a three-day seminar, with 4 hours per day), while in the other two groups the course was part of a larger semester course. All courses took place in a computer lab, employing the Eclipse IDE and the Android Development Tools (ADT) plug-in. Details about each group location, size and prior knowledge are provided in Table 4.

**Table 4** Details of groups to which the course has been delivered

| Group ID | Degree | Department | University/College | Enrollment | Prior Java course | Prior Android course | Prior SE course |
|---|---|---|---|---|---|---|---|
| 1 | undergraduate | Information Technology | Institute 1 | 16 | yes | no | no |
| 2 | undergraduate | Computer Science | Institute 2 | 30 | yes | no | no |
| 3 | postgraduate | Computer Science | Institute 2 | 12 | yes | no | yes |

The evaluation of the course was based on the Kirkpatrick Model (Kirkpatrick and Kirkpatrick 2006). This four-level model fitted well both with the nature of the course and the questions under investigation:

***Level 1 – "Reaction"***: evaluation of students' reaction to the course (e.g. topics, material), which corresponds to Question 1. For this level of evaluation an online questionnaire was used for assessing whether students liked the training, as well as for retrieving information on the background of the students. The questionnaire form can be found at http://androidse.wikispaces.com/Evaluation and consists of 24 closed-ended questions and 3 open-ended questions. For twenty-two of the closed-ended questions, responses are required in a five-level Likert-type scale. In an attempt to obtain feedback on several aspects of the proposed course, the questionnaire contains questions regarding the achieved objectives, the students' background prior to the course start date, the novelty and value of the delivered content as well as information regarding the particular topics that have been taught. Moreover, the questionnaire prompts students for suggestions regarding topics that should be omitted, topics that should have been included, as well as any other suggestion for improvement.

***Level 2 – "Learning"***: evaluation of the amount of learning that has occurred, which corresponds to Question 2. For this level of evaluation summative assessment was utilized. Summative assessment is a well-known method for evaluating the extent by which the intended learning outcomes of a module or program of study have been achieved, in terms of students' performance (Biggs and Tang 2011). Moreover, summative evaluation can also reveal unexpected results that may have emerged from the use of the course's material (Persico 1996).

The summative assessment for the proposed short course was performed at the end of the course and was carried out in the form of Multiple-Choice Questions (MCQs).

***Level 3 – "Behavior"***: this type of assessment measures the changes that have occurred in learners' behavior and attitude due to the program of study, which corresponds to Question 3. In particular, we aimed at investigating whether the proposed short course, illustrating the combination of software engineering and Android development, changed the interests of the participants with respect to Android. For this purpose a brief survey was administered anonymously to the students several weeks after the end of the course.

***Level 4 – "Results"***: the fourth level of the model refers to the analysis of the results and focuses on the effects on the environment resulting from the improved performance or advances in the level of knowledge of the trainees. Since the overall goal is to increase student's interest in Android development and emphasize the importance of considering software engineering theories and practices, an analysis of the results will be provided in the Discussion section.

Moreover, peer review was carried out using the Quality Matters standardized rubric (Quality Matters 2014) that is a faculty-centered peer review process for the evaluation of online and blended courses based on a set of 8 general and 41 specific standards. The rubric was completed by 4 instructors. This assessment corresponds to Question 4.

In the following paragraphs the results are analyzed using descriptive statistics. It should be noted, that because the short course was independently taught only in one setting, the results regarding the assessment at levels 1, 2, and 3 are presented for the group of students at Institute 1.

### 5.2 Results

#### 5.2.1 Students' Attitude towards the Course

In this section the results from the online questionnaire used for evaluating the course, as well as for retrieving information on students' background are presented.

One of the opening questions aimed at investigating whether students intent to be professionally involved with Android development, where 71% of them responded positively.

In the group of questions examining whether the course is relevant to the students' study objectives, the pattern of answers to all questions was similar to the one shown in Figure 4. In short, students tend to think that this kind of courses are valuable for their professional career, correspond to their expectations and improve their awareness and understanding of the subject. The popularity of an

Android related course comes as no surprise given that young people are daily flooded with innovations and success stories in the mobile computing market and consequently mobile computing has become one of the main attractors to computing studies.

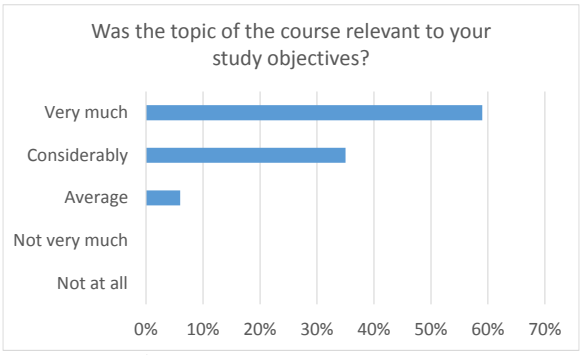


Fig. 4. Relevance of the proposed course to the student's study objectives.

To assess whether the course was placed in the right time of the students' curriculum, three questions investigated their prior knowledge level. Regarding the students' background on object-oriented programming, software engineering and Android development, students responded that they lack knowledge on Android development while the picture is slightly better for the other two, more classical academic subjects, namely object-oriented programming and software engineering (Figure 5). The findings confirm that Android development, despite its popularity has not yet found its way into computer science curricula, at least in the two involved institutes. It should be noted that the answers indicating a low familiarity with the aforementioned subjects came from the undergraduate students at Institute 1 where students have rather weaker programming skills.

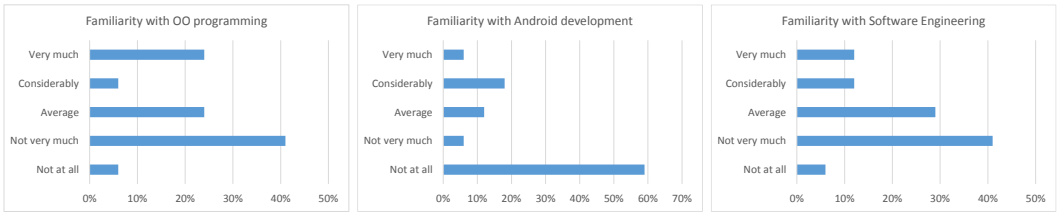


Fig. 5. Students background level on the course's topics.

For the questions investigating the novelty, interest and detail of the content as well as whether the use of a "live" Android application is helpful, the pattern of answers is similar to the one shown in Figure 6. Clearly students like courses where concepts are exemplified by real programming examples and the combination of state-of-the-art tools with programming activities.

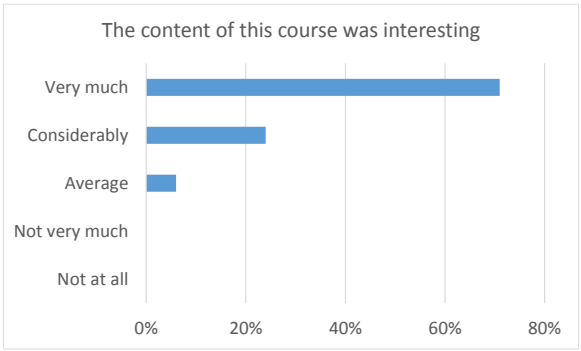


Fig. 6. Pattern of answers in questions investigating the delivery of novel and interesting content.

However, it should be noted that students expressed concerns about the accessibility of the course and the allocated time frame (Figure 7). Indeed, considering the effort to practically apply all of the involved steps and ensure that the corresponding Android application compiled and executed successfully on each student's workstation requires more time than originally planned. The difficulty of the discussed content can be attributed partly to the weak background as well as on the limited time. Based on these observations the authors conclude that an increase of at least 30-40% of the course teaching hours would be reasonable (as already mentioned the course was delivered in 12 hours including theory, lab exercises, mentoring and setting up the environment).

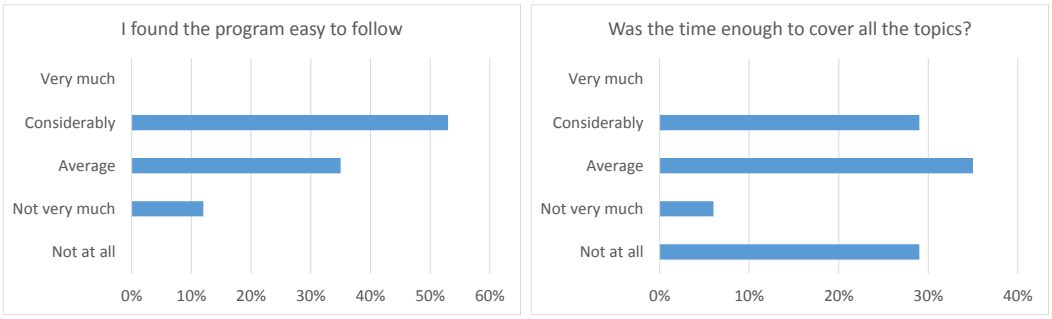


Fig. 7. Difficulty and adequacy of time.

The last set of questions aimed at identifying the particular fields which are found as more interesting by the students. None of the individual fields received a "Not very much" or "Not at all" answer, while the subjects can be ordered based on their popularity as follows: Java for Android development, Software Metrics, Object-Oriented Principles, Design Patterns, Collaborative Software Development, Unit Testing and Refactorings. Finally, 82% of the students responded that they would suggest such a course to their fellow students.

The findings from the open-ended questions asking for feedback, highlight the need to perform a smoother introduction to the basics of Android development and to devote more time to the individual steps, especially those that entail significant code writing activities. Provided that students can be split based on the prior courses which they have taken, the course could also be delivered in a more extended fashion by incorporating a full-fledged introduction into Java programming for Android development.

Although the aforementioned results cannot be considered as an in-depth and complete assessment of the pedagogical merits of the proposed course, they provide initial, promising indications that such a course could complement existing software engineering and object-oriented programming courses and stimulate the interest of students on proper software development practices in the world of Android applications.

#### 5.2.2 Assessment of Learning Outcomes

When teaching any academic course or unit, a major challenge lies in ensuring that there is an alignment between assessment and learning outcomes (Kennedy et al. 2006). Assessment tasks should mirror as directly as possible the learning outcomes since, from the perspective of students, the *assessment is the curriculum* (Biggs 2003).

In the context of this study summative assessment was used, as already mentioned, at the end of the course offered at Institute 1.

To determine the extent to which students have achieved the intended learning outcomes of the proposed short course, outcomes have been explicitly linked to assessment tasks (questions) as shown in Table 5. Such tables, although simple, can quickly reveal whether certain aspects of the course are under- or over-assessed.

**Table 5** Linking of Learning Outcomes to Assessment Tasks

| | **Learning Outcomes** | | |
|---|---|---|---|
| **Assessment Question** | (1) *Explain* the benefits from applying software engineering practices to Android | (2) *Apply* Design Principles and Patterns in Android applications | (3) *Identify* & *Resolve* design issues in Android applications |
| 1 | ✓ | | |
| 2 | ✓ | | |
| 3 | ✓ | | |
| 4 | ✓ | | |
| 5 | ✓ | | |
| 6 | | ✓ | |
| 7 | | ✓ | |
| 8 | | ✓ | |
| 9 | | ✓ | |
| 10 | | ✓ | |
| 11 | | | ✓ |
| 12 | | | ✓ |
| 13 | | | ✓ |
| 14 | | | ✓ |
| 15 | | | ✓ |

Since the learning outcomes in the proposed short course are mainly related to the ability of students to comprehend and apply (and to a lesser extent with the ability to analyze) and considering that the assessment consisted of multiple-choice items (due to the short-term nature of the course) the particular learning objectives have been measured by accuracy (i.e. correct vs. erroneous answers) (Carnegie Mellon 2014).

Table 6 lists the learning outcomes along with their respective measuring instruments, achievement targets, and results. Similarly to the standards and criteria for demonstrating excellence in various fields, the desired competency level for a learning outcome is considered as met if a minimum of 70% of the students score at least 70%. (As score for each learning outcome, the average score on the relevant MCQs is considered).

Descriptive statistics on the summative assessment results based on the final exam are shown in Figure 8. Average and median student scores are presented for each of the learning outcomes listed in Section 3.1.

**Table 6** Learning and Performance Summary for the proposed short course

| **Intended Learning outcome** | **Measuring Instrument** | **Achievement Target** | **Actual Results** | **Conclusion** |
|---|---|---|---|---|
| (1) *Explain* the benefits from applying software engineering practices to Android | MCQs | 70% of the students score at least 70% | 75% of students scored at least 70% | Met |
| (2) *Apply* Design Principles and Patterns in Android applications | MCQs | 70% of the students score at least 70% | 75% of students scored at least 70% | Met |
| (3) *Identify* & *Resolve* design issues in Android applications | MCQs | 70% of the students score at least 70% | 20% of students scored at least 70% | Not Met |

With regard to the intended learning outcomes, the first and second outcomes have been attained, partially validating the belief that such a short course can bridge existing knowledge on software engineering with contemporary programming platforms such as Android. The lowest scores (both average and median) have been achieved in questions related to the identification of design issues in Android applications by metrics and resolution by means of refactorings. As a result, the third and most ambitious learning outcome has not been met. This can be possibly attributed to the fact some of the concepts need longer exposition in actual Android projects in order to be really understood by students. Moreover, for Android related problems, the numerous implementation details behind Android development and its libraries pose difficulties to newcomers, and in agreement to the students' satisfaction survey (presented in Section 5.2.1) more time should be devoted, especially if students lack background in Java for Android applications.

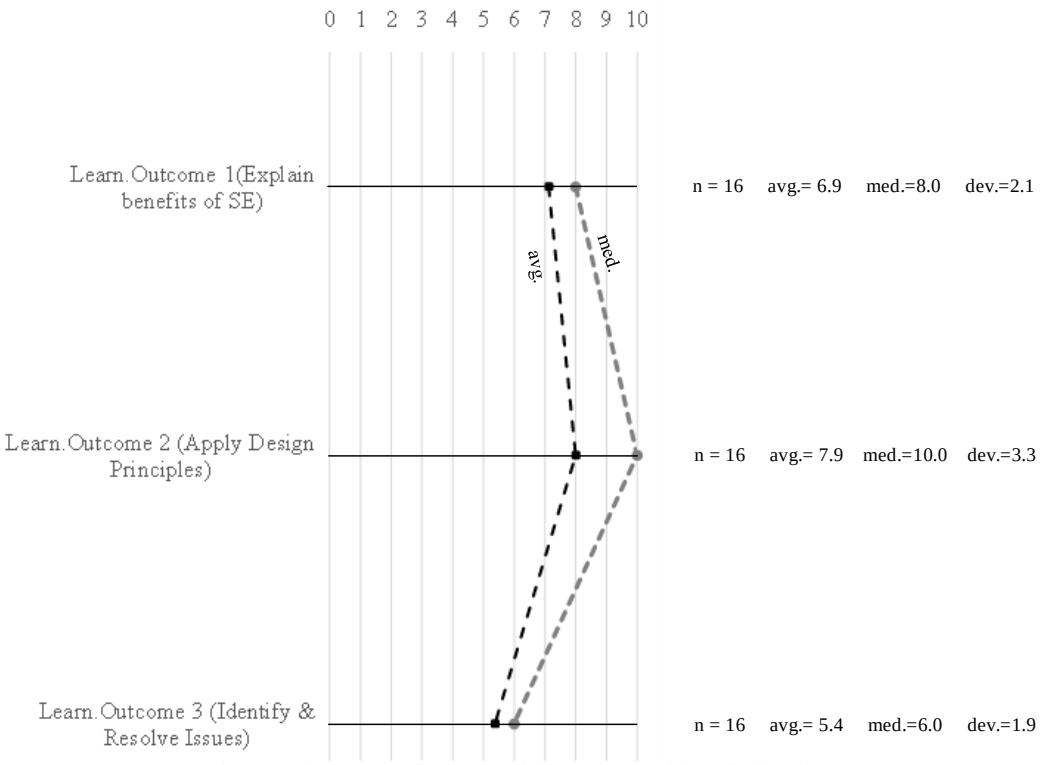


Fig. 8. Students' performance on final exam with respect to each intended learning outcome

**5.2.3 Assessment of changes in students' interests**

In this section the results regarding the change of the participants' interests with respect to Android are presented. Similarly to the assessment carried out by Ritzhaupt for a game development course (Ritzhaupt 2009), a brief survey was administered anonymously to the students several weeks after the end of the course containing the following three questions (answers have been recorded in the same scale as in the survey that took place at the end of the course):

Q1. Have you experimented with Android again since you took the relevant course in the college?

Q2. Are you going to program on Android in the foreseeable future?

Q3. Are you looking for any jobs/projects relevant to Android?

The results are summarized in the bar charts of Figure 9.

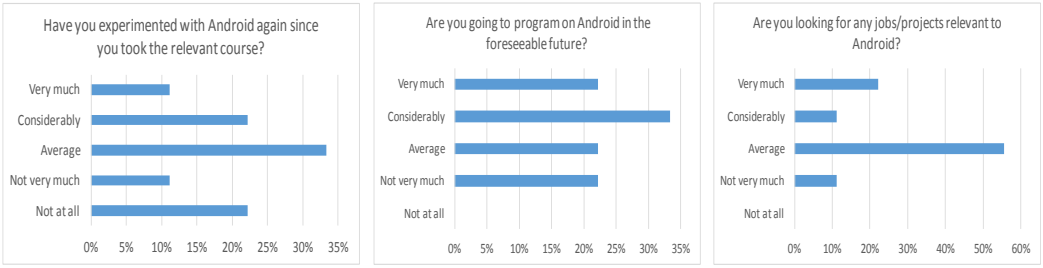


Fig. 9. Survey results on students' interest on Android

Based on students' responses, a significant engagement in Android development tasks occurred after the completion of the course which can be attributed to the attractiveness of Android applications and the relative ease in getting started with developing them. More striking is the renewed interest of students to experiment with Android in the foreseeable future, which can be acknowledged as one of the major achievements of the proposed course. Especially for students at Institute 1 (which is located in a country with limited opportunities for ICT careers), introducing an additional career opportunity, as reflected in the rightmost diagram in Figure 9, is a promising aspect for such a short course. These observations become even more important, considering the background knowledge level in Android which according to the results outlined in Figure 5, was essentially non-existent for this particular group of students, prior to the beginning of the course.

#### 5.2.4 Peer Review

Peer review of academic courses is another widely adopted approach employed for assessment. External reviewers can identify program strengths and weaknesses and offer significant feedback on whether the design and implementation of a course aligns with departmental goals and objectives (Fong 1987). For the evaluation of the proposed short course we employed the Quality Matters standardized rubric (Quality Matters 2014), which is a faculty-centered peer review process for the evaluation of online and blended courses based on a set of 8 general and 41 specific standards. The rubric has been completed by 4 instructors, of which three had significant experience in Vocational Pedagogy and thus can be considered as experts regarding the assessment of a course. All evaluators assigned a total score higher or equal to 81 (81, 84, 87, 88) meaning that the course would meet the corresponding standards. With regard to the eight general standards, the scores assigned by each of the four reviewers is shown in the radar chart of Figure 10. As it can be observed, all reviewers agreed that learning objectives, learner engagement, course technology and learner support have been fully addressed. On the other hand, there is room for improvement in the course overview, assessment, instructional material and accessibility.

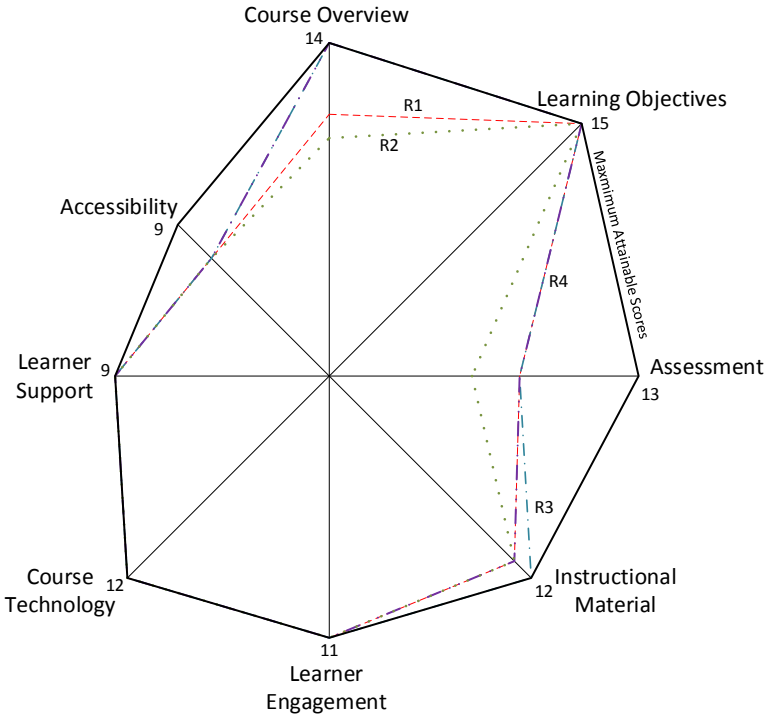


Fig. 10. Radar chart illustrating rubric scores along the 8 assessment standards

The main limitations which have been identified by all reviewers during this form of assessment are:

- the unavailability of multiple, varied instruments for students' assessment
- the fact that no distinction between required and optional material has been made clear
- the lack of equivalent alternatives to auditory and visual content

These limitations stem mainly from the fact that the proposed course is a short one embedded in a series of lectures rather than a self-standing full-semester course with regular and continuous assignments. Nevertheless, they can be easily addressed in future implementations by adopting a more classical course introduction and by embedding multiple ways for measuring students' progress.

## 6. DISCUSSION

As already mentioned above, the proposed short course has been delivered during the winter academic semester of 2013-2014 to three different groups of students. It goes without saying that an Android course is always welcome with enthusiasm by students given the popularity of the Android OS and the numerous success stories related to young entrepreneurs entering the global market with an exceptional Android application. The proposed course can be preceded by a brief introduction to Android development, depending on whether students have prior knowledge on mobile software development. Nevertheless, emphasis is not placed on the Android platform itself but rather on the fact that "classical" software engineering concepts also apply in this context.

### 6.1 Problems and Challenges

The major challenges during the preparation and implementation of the proposed course and the resulting lessons that have been learned are outlined next:

a) *Prior knowledge*: To ensure a smooth progression of the course the instructor should be aware in advance of the students' level of knowledge and skills in both software engineering and Java development. Although this is a theoretical prerequisite for any course, it is of vital importance in this

case, since the goal is to illustrate particular concepts on top of Android development and as a result it wouldn't be effective to deviate from this aim at every step, just in order to clarify the related Java programming or Android particularities. The optimum effectiveness can be obtained if the course is delivered right after a software engineering course, or at least a course related to Object-Oriented Software Development. Although key concepts such as the need to maintain software and the importance of software quality can be reminded during the course with a brief parallel lecture, the course will benefit the most if it can be considered as an extended software engineering lab exercise. According to Riley (2012) even writing simple applications from scratch in Android requires proficiency in object-orientation and these requirements with respect to the learning curve keep most students from learning Android on their own.

b) *Duration*: Although in theory the content of the proposed course is limited, it turned out that covering all of the introduced concepts and techniques along with the completion of the related activities requires no less than 12-16 teaching hours, depending on the students' background. Even when the course is delivered to postgraduate students, the multitude of involved technologies and the demonstration on an actual case study prolongs the time of each step.

c) *Infrastructure*: The course relies on a multitude of tools such as the Eclipse IDE for programming, Android SDT, tools for unit testing, metrics calculation, refactoring application, version control and possibly computer-aided software engineering (CASE) tools. This implies that in order for the course to progress in a smooth and timely fashion, the laboratories and workstations on which the course will take place have to be carefully prepared. Students could also be involved in the installation and configuration of the tools but this would incur significant time overheads, which can sometimes distract or disappoint students from the main learning goals. Due to the nature of the examined project it was often required to deliver one working version of the project to all students. To this end, it is particularly helpful to run the course in a lab where all workstations can be accessed from each other or from a central server.

d) *Synchronization*: The course is structured as a series of small incremental steps, exactly as it would occur during the actual development of a software project in a small team. For the students to remain focused the instructor should ensure that at any given time point all of the students are working on the same medium (i.e. version of the involved application) so that the discussed concepts and techniques will be illustrated without difficulty. Given that the students have often a completely different working pace from each other, this poses a serious challenge and requires continuous monitoring of the progress of each team. Once again, the students with the highest familiarization with Java confronted the fewest problems in this regard.

e) *Student participation*: As in any other programming course, student participation should be a key objective of the course. Students can actively participate either by asking questions, providing feedback on encountered problems, suggest alternative implementations and even develop a solution, implement it and share it with the rest of the students' groups. However, the multitude of introduced concepts, tools, libraries along with the compile, run-time or emulator errors that can arise in Android, hinders student's participation who find themselves in the process of "making their own project work". The instructor should be aware of this issue and resolve persistent problems so that students can take on the essence of the delivered material.

### 6.2 Compliance to Curriculum Guidelines

Regarding the curriculum guidelines for software engineering (LeBlanc and Sobel 2004), we believe that the proposed course adheres to most of them. Next, we discuss the ones that focus on individual courses and those that seemed more relevant to the challenges that have been faced during the design and implementation of this course. The 1st guideline according to which *"Curriculum designers and instructors must have sufficient relevant knowledge and experience and understand the character of software engineering"* is particularly relevant in this case, because of the rapid evolution of Android related technology. Student questions in such a course can vary to a large extent requiring from the instructor solid knowledge on object-orientation and software engineering (e.g. students have asked about the difference between *overloading* and *overriding*, or questions like "*can we employ serialization for object persistence in Android applications?"*). Although we are not in a position to judge whether the delivered course adhere to this principle, the covered topics are standard areas within software engineering so that the instructor can assess whether he masters them or not.

The 3rd guideline emphasizes the need to *"... strike an appropriate balance between coverage of material, and flexibility to allow for innovation"*. Given that the proposed course is structured around a 'live' application, it can be easily adapted to address particular technologies related to Android development by incorporating advanced functionality.

The 7th guideline stresses the importance of recognizing both the computing and engineering dimensions of software engineering. A 'pure' course on Android development will unavoidably entangle itself around the particularities of Android libraries and limitations. The introduction of software engineering concepts highlights the kind of tradeoffs that an engineer should address in a real life setting by weighting software qualities against the required cost and effort of improving them.

The fact that the proposed course can be organized as a series of lab exercises that can be undertaken by groups of students enables the application of the 8th guideline regarding the need to include training on certain personal skills. Effective communication is encouraged, especially because of the explicit requirement for collaborative software development by means of a version control system. The accompanying content management system can be also of great help to this direction.

Of particular importance is curriculum guideline 11, which places emphasis on the enduring knowledge of software engineering and not on the details of the involved technologies. Many of the trivial technical aspects (e.g. setting up the work environment in Eclipse, setting up initial repository and GIT setup etc.) have been omitted on purpose, since they do not address the above guideline. As it has been explained, prior experience on Java programming and familiarization with Android SDT is a key factor which affects both the effectiveness and the timeliness of such as course. Nevertheless, the students are introduced to a good number of state-of-the-art tools in accordance to the 12th guideline ("*The curriculum must be taught so that students gain experience using appropriate and up-to-date tools, even though tool details are not the focus of the learning*").

The course is clearly in agreement with curriculum guideline 14 emphasizing the need to have a *"significant real-world basis"*. The developed Android application is far from a 'hit' mobile app, but it can be extended based on students' interest(s). Such extensions can be easily integrated into the course in the form of group assignments. The incorporation of individual changes is facilitated by the use of a collaborative software development platform and policy. For the same reason, it can be claimed that the course addressed also guideline 16 as it attempts to motivate students by using interesting, concrete and convincing examples.

The motivation for introducing this particular course acknowledges the need to combine software engineering with mobile application development as the two are often studied in isolation. Curriculum guideline 18 claims that *"important efficiencies and synergies can be achieved"* by acquiring several types of knowledge at the same time. This synergistic teaching and learning approach could be adopted in case the proposed course is extended so that during the same teaching hours both software engineering, Java programming and Android development can be learned.

### 6.3 Evaluation Results

This section discusses the main findings of the evaluation that has been performed, with respect to the four questions that have been formulated in Section 5.1.

*How do students evaluate the proposed course?* From the student's responses to the questionnaire that has been delivered it can be concluded that students are definitely interested in the proposed course although it is highly probable that the main attractor is Android. Even so, employing a highly popular platform to convey software engineering concepts can yield a successful, in terms of students' perception, course. Nevertheless, students feel that more time should be available for digesting the involved technologies. It should be emphasized that a questionnaire inquiring students' perception of the course unavoidably reveals that students' behavior is monitored. As a result the Hawthorn effect (Roethlisberger and Dickson, 1939) might have affected students' responses since they have been aware that they are part of an experiment. However, the fact that students frankly stated their weaknesses in the examined topics and pointed out the inadequacy of the devoted time frame, partially mitigates this threat.

*What are students' achievements with respect to the intended learning outcomes?* The intended learning outcomes for this course are limited and coarse-grained due to the short duration. Students scored well

on multiple choice questions related to the most theoretical aspects of the delivered content (first and second outcome) but achieved lower scores with respect to the practical skill of identifying and resolving design issues in Android applications (third outcome). From the results it can be concluded that learning outcomes related to understanding and application according to the revised Bloom's taxonomy (Krathwohl 2002) have been met but it is rather ambitious to turn inexperienced students into programming experts who can identify and resolve (i.e. analyze) problems in Android applications in a period of a few days. Combined with the results of the other types of evaluation it becomes evident that the content of the proposed course should either be delivered to more experienced students or span across a longer period of time.

*How does the course affect students' interest on Android development?* Based on the results of the corresponding assessment and considering the prior knowledge on the subject, it can be safely concluded that a short-course with the proposed content can cultivate student's interest on mobile application software development. However, also in this type of post-course assessment, one should consider the Hawthorne effect when interpreting the feedback since students' responses might have been biased.

*How do instructors evaluate the course?* Based on the results from the Quality Matters rubric, the proposed course would meet the required standards. Opportunities for improvement have been identified regarding the course's overview, assessment, instructional material and accessibility.

## 7. CONCLUSIONS

Android development courses have turned into 'high demand' courses in higher education institutes around the world given the popularity of mobile devices running the Android OS and the bright prospects for mobile application developers in the IT market. Android software systems are becoming increasingly complex and are evolving projects calling for the application of all established best practices in software engineering. However, there is a clear lack of academic experience in the design and implementation of courses that aim at illustrating the application of software engineering concepts and tools in the context of Android programming. In this paper, we have introduced a short course aiming at bridging the gap between Android development and state-of-the-art software engineering content, which has been delivered to three groups of students in two countries.

The proposed course is structured as a series of focused enhancements in a base Android application where a different software engineering concept, technique or tool is introduced at each step. The implementation of the course reveals both the benefits that can result from the fusion of Android and software engineering concepts as well as the challenges in such a project. The course is supported by an accompanying wiki page and the content as well as the underlying application can be easily adapted to meet the requirements of any curriculum. Findings from four different types of evaluation suggest that such a course can be both appealing and valuable to computer science students. However, care should be taken in properly adjusting the course material with students' prior knowledge.

## REFERENCES

ACM. (2013). Curriculum Guidelines for Undergraduate Programs in Computer Science, Retrieved December 15, 2014, from www.acm.org/education/CS2013-final-report.pdf.

Akopian, D., Melkonyan, A., Golgani, S., Yuen, T. & Saygin, C. (2013). A Template-Based Short Course Concept on Android Application Development. *J. Inf. Technol. Educ. Innov. Pract*. 12 (2013), 13–28.

Biggs, J. (2003). Aligning teaching and assessing to course objectives. *Teaching and Learning in Higher Education: New Trends and Innovations*. University of Aveiro, April 2003.

Biggs, J. & Tang, C. (2011). *Teaching for Quality Learning at University*. 4 edition. Maidenhead: Open University Press.

Carnegie Mellon. (2014). Aligning Assessment with Objectives. Retrieved August 18, 2014 from http://www.cmu.edu/teaching/assessment/howto/basics/objectives.html.

Dewey, J. (1938). *Education and experience*. Macmillan, New York, NY.

European Commission (2012). European Vacancy and Recruitment Report. Retrieved December 15, 2014 from http://ec.europa.eu/social/main.jsp?catId=738&langId=en&pubId=7267&type=2&furtherPubs=yes

Fong, B. (1987). *The External Examiners Approach to Assessment*. Washington, DC, USA: Association of American Colleges.

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional, Boston, MA.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Reading, MA.

Gartner. (2013). Gartner Says Worldwide PC, Tablet and Mobile Phone Shipments to Grow 4.5 Percent in 2013 as Lower-Priced Devices Drive Growth. Retrieved January 16, 2014 from http://www.gartner.com/newsroom/id/2610015.

Ghezzi, C., Jazayeri, M. & Mandrioli, D. (2002). *Fundamentals of software engineering*. Prentice Hall PTR, Upper Saddle River, NJ.

git. 2014. Git. Retrieved February 21, 2014 from http://git-scm.com/.

GitHub. (2014). GitHub. Retrieved February 21, 2014 from https://github.com/.

Heckman, S., Horton, T. B. & Sherriff, M. (2011). Teaching second-level Java and software engineering with Android. In *Proceedings of 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T'11)*. IEEE Press, New York, NY, 540–542, DOI: http://dx.doi.org/ 10.1109/CSEET.2011.5876144

Herbert, N., de Salas, K., Lewis, I., Cameron-Jones, M., Chinthammit, W., Dermoudy, J., Ellis, L. & Springer, M. (2013, January). Identifying career outcomes as the first step in ICT curricula development. In *Proceedings of the Fifteenth Australasian Computing Education Conference*-Volume 136 (pp. 31-40). Australian Computer Society, Inc..

Hu, W., Chen, T., Shi, Q. & Lou, X. (2010). Smartphone Software Development Course Design Based on Android. In Proceedings of the 10th IEEE Conference on Computer and Information Technology (CIT'10). IEEE Press, New York, NY, 2180–2184. DOI: http://dx.doi.org/ 10.1109/CIT.2010.374

JUnit. (2014). JUnit. Retrieved March 2, 2014 from http://junit.org/.

Kennedy, D., Áine, H. & Norma, R. (2006). Writing and Using Learning Outcomes: A Practical Guide. In EUA Bologna Handbook – Making Bologna Work. Berlin: Raabe Verlag.

Keller, J. M. (1983). Motivational design of instruction. In C. Riegeluth (ed.), Instructional Design Theories and Models. Hillsdale, NJ: Erlbaum, 383–434.

Kirkpatrick, D. L. & Kirkpatrick, J. D. (2006). *Evaluating Training Programs: The Four Levels*. 3rd edition. San Francisco, CA: Berrett-Koehler Publishers.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into practice*, 41(4), 212-218.

Lai, Y. C. & Ng, E. M. W. (2011). Using wikis to develop student teachers' learning, teaching, and assessment capabilities. *Internet High. Educ*. 14 (2011), 15–26.

LeBlanc, R. & Sobel, A. (2004). Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. IEEE Computer Society and ACM.

Liskov, B. H. & Wing, J. M. (1994). A behavioral notion of subtyping. A*CM Trans Program Lang. Syst.* 16 (1994), 1811–1841.

Mahmoud, Q. H. (2008). Integrating mobile devices into the computer science curriculum. In *Proceedings of the 38th Annual Conference on Frontiers in Education Conference (FIE'08)*. S3E-17 - S3E-22. DOI: http://dx.doi.org/10.1109/FIE.2008.4720686

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, Upped Saddle River, NJ.

Mednieks, Z., Dornin, L., Meike, G.B. & Nakamura, M. (2012). *Programming Android: Java Programming for the New Generation of Mobile Devices*. O'Reilly Media, Sebastopol, CA.

Mulligan, M. & Card, D. (2014). Sizing the EU app economy. Retrieved January 10, 2014 from http://eurapp.eu/sites/default/files/Sizing%20the%20EU%20App%20Economy.pdf.

Parker, K. R. & Chao, J. T. (2007). Wiki as a teaching tool. *Interdisciplinary Journal of Knowledge and Learning Objects*, 3 (2007), 57–72.

Persico, D. (1996). Courseware Validation: A Case Study. *Journal of Computer Assisted Learning* 12(4): 232–244.

Petkovic, D., Thompson, G. & Todtenhoefer, R. (2006). Teaching Practical Software Engineering and Global Software Engineering: Evaluation and Comparison. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, (ITICSE '06)*. ACM, New York, NY, 294–298. DOI: http://dx.doi.org/ 10.1145/1140124.1140202

Quality Matters. (2014). Quality Matters Overview. Retrieved May 15, 2014 from https://www.qualitymatters.org/applying-Rubric15/download/QM_Overview_for%20Current%20Subscribers_AE2013.pdf.

Riley, D., (2012). Using mobile phone programming to teach Java and advanced programming to computer scientists. In Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12). ACM, New York, NY, 541–546. DOI: http://dx.doi.org/ 10.1145/2157136.2157292

Ritzhaupt, A. D. (2009). Creating a Game Development Course with Limited Resources: An Evaluation Study. *ACM Transactions on Computing Education* 9(1): 3:1–3:16.

Roethlisberger, F. J. & Dickson, W. J. (1939). Management and the worker. Cambridge, MA: Harvard University Press.

Schwartz, L., Clark, S., Cossarin, M. & Rudolph, J. (2004). Educational Wikis: features and selection criteria. *Int. Rev. Res. Open Distance Learn*, 5, 1 (2004).

SFIA Foundation, Skills Framework for the Information Age. http://www.sfia-online.org/, Accessed June 26, 2015.

Spinellis, D. (2003). *Code reading: the open source perspective*. Addison-Wesley Professional, Boston, MA.

Tenenberg, J. D. (1995). Using cooperative learning in the undergraduate computer science classroom. *Journal of Computing in Small Colleges.* 11, 2 (1995), 38-49.

Tsantalis, N. & Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. *IEEE Trans. Softw. Eng*. 35, 3 (2009), 347–367.

Victor, H. (2013). Android's Google Play beats App Store with over 1 million apps, now officially largest. Retrieved January 16, 2014 from http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680.

Wasserman, A. I. (2010). Software engineering issues for mobile application development. In *Proceedings of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FoSER '10)*. ACM Press, New York, NY, 397-400. DOI: http://dx.doi.org/ 10.1145/1882362.1882443

Whitney, L. (2012). Android reclaims 61 percent of all U.S. smartphone sales | Internet & Media - CNET News. Retrieved January 16, 2014 from http://news.cnet.com/8301-1023_3-57429192-93/android-reclaims-61-percent-of-all-u.s-smartphone-sales/.