CONTEXTUAL ANDROID EDUCATION

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
James Reed
December 2010

Committee Membership

TITLE:	Contextual Android Education
AUTHOR:	James Reed
DATE SUBMITTED:	December 2010
COMMITTEE CHAIR:	David Janzen
COMMITTEE MEMBER:	Alexander Dekhtyar
COMMITTEE MEMBER:	John Clements

Abstract

Contextual Android Education

James Reed

With advances in mobile phone hardware, the demand for mobile applications has risen drastically. This has resulted in mobile phones becoming a popular new medium for application development. However, the body of knowledge for contextual examples and tutorials leaves much to be desired. As of January 2010, California Polytechnic State University has offered a mobile development class that teaches students how to write applications for phones running Google's Android platform. This class aims at taking advantage of students' current interest in mobile applications to teach them about difficult computer science topics. As a corollary, the class hopes to foster and encourage a sense of independence and entrepreneurship through having students design, implement, and publish their own applications to the Android Application Marketplace. The main contribution of this thesis project comes in the form of a series of detailed educational laboratory exercises and a system for grading student submissions in an automated fashion. These labs are designed to supplement the Android documentation by providing contextual examples, activities, and tutorials. It is therefore the goal of this thesis project to aid in transforming the class of mobile development students into a group of successful, practicing, mobile developers.

Keywords: Android, Mobile Development, Software Engineering, Education

Contents

List of Figures

List of Tables

1 Introduction

With advances in mobile phone technology, like increased computing power, improved touch screen user interfaces, and faster wireless internet connections, the demand for both free and paid mobile applications has risen dramatically. This has resulted in mobile phones becoming a popular new medium for application development. There are many high powered platforms to choose from, such as Blackberry, iPhone, and Android. Most of these platforms offer an open marketplace for the sale and distribution of independently developed mobile applications. This makes it incredibly easy for independent developers to get their products into the hands of users. Awareness of this new market has not escaped the interest of computer science students who are often consumers of the applications themselves. With the demand of mobile applications, low barrier to entry into the market, and general interest from the computer science student population, mobile application development is an excellent skill for computer science students to learn. Most of these platforms offer a wide array of documentation for learning how to develop for their platforms; however, the body of knowledge for contextual examples and tutorials is drastically smaller in comparison. This can be troublesome for some college students attempting to break onto the mobile development scene on their own.

Since January 2010, California Polytechnic State University has offered a mobile development class that teaches students how to write applications for phones running on the Android platform. Computer Science (CS) students on the whole represent a younger demographic, which in turn represents a large customer base for mobile applications. While in some cases it can be difficult to capture the interest of CS students, this class aims to take advantage of students' current interest in mobile applications to teach them about difficult topics. In addition to providing the students with an in depth knowledge in mobile development practices and the skills necessary to be successful Android application developers, the class will incorporate aspects of Test Driven Development, Agile Processes, application profiling, and engineering for performance into the curriculum. With these skills, students will be able to independently create exceptional mobile applications. As a corollary, the class hopes to foster and encourage a sense of independence and entrepreneurship through having the students design, implement, and publish their own applications to the Android Application Marketplace.

The Android platform was chosen as the medium for the class, at least in part, because of its degree of openness. In short, the Android platform is an open source mobile device software stack coupled with a robust software development kit (SDK) based on the Java programming language that provides the tools and application programming interfaces (APIs) necessary to develop applications for the platform [28]. Additionally, this openness extends to the Android Application Marketplace as well. Submitted applications need only meet a handful of reasonable, clearly stated functional requirements and include a developer account, which can be obtained for a nominal fee. This level of openness makes it that much easier for students to become practicing mobile developers.

It is therefore the goal of this thesis project to aid in transforming the class of mobile development students into a group of successful, practicing, mobile developers. The main contribution of this thesis project comes in the form of a series of detailed educational laboratory exercises. These labs are designed to supplement the Android documentation by providing contextual examples, activities, and tutorials. They will be designed and used in coordination with in-class lectures as well, where the topics that will be covered in the labs shall be introduced and discussed. Each lab will have a number of learning objectives associated with it. The end goal of each lab is for every student to possess and be able to use the skills associated with each learning objective on their own. To be able to determine the success of these labs, we will incorporate a subjective survey that each student will complete at the end of the lab. These surveys will be designed with the intent of evaluating the effectiveness of the labs' ability to improve each student's understanding and application of the skills associated with it.

2 Background

As the work done for this thesis deals heavily with mobile application development and Google's Android platform, it's important to clearly define those terms. To start, we present the genre and current state of mobile application development from the perspective of entrepreneurial software developers. We then present the Android platform with respect to how it meets the needs of entrepreneurial mobile developers. By doing this, we hope to clearly summarize the scope and context of the problem this thesis is solving for and the tools used to solve it.

2.1 What is Mobile Development?

Mobile application development is the creation of software applications targeted specifically for mobile devices that may or may not rely on a cellular infrastructure[2]. Developers of mobile applications tend to target niche markets, in particular, those devices which tend to be most popular. Currently, the most popular of the mobile devices are cell phones. While it is true that there are many similarities between developing a mobile application and developing any other application, there are just as many differences, if not more. In the section that follows, I intend to highlight those differences in a way that helps describe what exactly developing a mobile application means.

2.1.1 Mobile Development and the Entrepreneur

There are a number of reasons why one might develop applications for mobile devices. However, most applications can be put into one of two categories. The first category consists of applications that are developed for the intent of personal consumption. One reason for developing such applications might be to achieve some sort of personal utilitarian goal; examples of such goals include education, hypothesis testing, and monetary gain. The second category consists of applications that are developed for the purposes of being distributed to and consumed by the masses. Reasons for developing applications for mass consumption include those for developing for personal consumption. The difference between the two categories lies in the intent for the application to be used by either one person or many people.

This paper focuses on developing applications intended for consumption by many, as this is generally the concern of entrepreneurial mobile application developers. While it is true that entrepreneurial activity can be undertaken with the intent of creating a mobile application for the personal consumption of the entrepreneur, this is not the focus of this paper. Therefore, for the purposes of this paper, we shall define the goal of the entrepreneurial mobile application developers to be that of getting their mobile applications into the hands of as many users as possible. In particular, it is the intent of this section to enumerate the obstacles that entrepreneurial mobile developers face in pursuit of their goal, as well as to show whether or not the Android platform helps entrepreneurs overcome these obstacles.

2.2 Fragmentation of the Mobile Market

One of the biggest problems with developing a mobile application is identifying the environment in which the application will run. In general, the environment is composed of a development platform on which the application is written, the operating system on which the platform was written, the cellular mobile device on which the operating system runs, and the cellular communications network on which the mobile phone operates [32]. The problem stems from the technical dependencies that exist between each environmental component that typically cause a mobile application to be tied to a relatively small combination of networks, devices, operating systems, and platforms[44].

Let us perform a depth first traversal of environmental component dependencies to understand the complexity. At the top level, we have a mobile application that strongly depends on the application programming interfaces (APIs) provided by its target platform. The platform then depends on the services offered by the operating system (OS) for which it was written. The OS heavily depends on the hardware and drivers provided by the mobile device on which the OS runs. Lastly, due to the incompatibility of networks, the hardware for a mobile device has to be developed for a specific network(s). In addition to this hierarchical structure of dependencies, it is often the case that dependencies reach across the hierarchical boundary. For example, a platform may provide an API for direct access to network or device specific functionality, such as camera or GPS service. This effectively ties the platform to the network and the device. Alternatively, the network or device might provide it's own API for these features. If an application uses these API it is then tied to the specific network GPS standard and device hardware[44]. It is because of these dependencies that consumers of mobile applications are divided into fragmented groups of environmental combinations which can only run a subset of all available mobile applications. Thus, the market for mobile applications is fragmented.

Now consider the problem that this causes for the mobile development entrepreneur. Any attempt to achieve widespread consumption of a mobile application then requires some combination of the following two strategies. The first strategy involves developing multiple versions of the application to run on different combinations of environmental components. This allows your application to run on multiple and distinct sections of the market. The second strategy involves developing for a particular combination that provides the greatest potential for user consumption. From the perspective of the entrepreneur, the optimum combination of environmental choices would be the one that targets the largest group of consumers. In any case, a single application cannot be developed that can be consumed by the entire mobile market. Instead, an application has only the potential to be consumed by the fragment of the market, or niche, for which it was developed. Furthermore, targeting development for a particular niche of the market affects decisions relating to production and distribution.

2.2.1 Effects on Production Decisions

In most cases, the target niche in the mobile market for a mobile application also dictates the technology stack on which the application will be written. Or at the very least, greatly reduces the possible combinations of environmental components. Furthermore, the choice of niche is often dictated by the users. That is to say, “operating systems, development tools and mobile networks may create a good environment for mobile application development, but only the user and the use create the business around applications”[32].

Take for example an application whose target audience is a demographic that is primarily using the iPhone. As of the date of writing this, the iPhone operates on a single network, runs a single operating system, and supports a single platform. In this example, targeting this niche of the mobile market dictates a single combination of environmental components for which the application must be developed. Furthermore, this particular example also dictates the tools that must be used to develop this application. iPhone development can only be performed using the Xcode integrated development environment (IDE), which comes with the iPhone software development kit (SDK), which can only be used with Intel-based Mac computers.[6]

While the previous example is the most extreme case, it illustrates the prevalent fact that targeting a mobile market niche limits the available technology choices related to development. In less extreme cases, such as targeting a BlackBerry audience, one still has the choice over development tools and the added benefit of BlackBerry devices being supported on multiple networks.[36] However, even in this case all decisions must align with the targeted mobile market niche. Ultimately, it is the targeted end users of an application and their combination of networks, devices, platforms, and operating systems that carry the most weight when it comes to technology choice.

2.2.2 Effects on Distribution Decisions

After development of a mobile application is complete, the application needs to be distributed to end users. For the purposes of this paper we define distribution within the mobile development context to be the systems and processes in place that allow developers to deploy and sell their applications to end users, as well as the means by which end users purchase, download, and install applications. There is a growing trend whereby the distribution of mobile applications are often controlled by some combination of mobile device manufacturer, platform developer, and/or mobile network operator. Distribution systems are therefore dependent on the targeted mobile market niche. Switching between target niches, or technology combinations, often means switching distribution systems as well. Examples of such distributions systems include Qualcomm's Brew platform, which operates in cooperation with network providers[30]. The Google Android platform, which operates its own distribution system called the Android Market[22]. The RIM BlackBerry platform, which operates its own distribution system called the App World storefront [33].

Controlling the distribution of mobile applications offers benefits to network operators, device manufactures, and platform providers. By controlling the dissemination of applications that operate on a particular network, device, or platform, the controlling party has the ability to inspect each application before it is consumed by the end user. This allows the controlling party to act as a gatekeeper, ensuring the security of their network, device, and/or platform. It also allows the the controlling party to mitigate problems with interoperability of applications across devices, networks, and platforms. Lastly, it allows the controlling party the ability to guarantee a certain level of quality in the applications that are released to end users[44]. Control of distribution is generally exercised through an application certification process in conjunction with the required use of a controlled distribution channel, such as a marketplace.

The certification process, as a mechanism to control distribution of applications, varies between the certifying parties. However, the processes generally incorporate one or more of the following steps as outlined by Tarnacha et. al[43]:

Developer Registration Requires the developer to register with the certification program and sign contracts specifying the legal and technical scope of the certification program. As is the case with BlackBerry[35] and iPhone[5] applications. Registration may or may not require a monetary fee.

Application Submission Requires the developer to submit the application to an online repository.

Authorized Certification Testing Requires the developer to choose an authorized testing lab or is assigned one to perform certification testing on their submitted application.

Application Testing Requires the submitted application to undergo testing. Testing is based on the certifying parties test plan and general guidelines.

Application Signing Requires that the submitted application be digitally signed with a unique application certificate. May be performed by developer, or may be required to be performed by a certificate/signing authority. As is the case with BlackBerry applications that desire to use certain restricted platform API[34]. Compare this with all Android applications, which all need to be signed, however Android applications may be self-signed[24].

This certification process is usually considered by developers as both an investment and a barrier to entry. This is due to the high costs associated with performance and functionality testing on various combinations of platforms devices and networks, compared to the typical revenue the application is expected to yield[44]. Therefore developers need to fully understand the certification process of a targeted niche. Failure to develop with the certification process in mind could lead to thousands of dollars or more on a product that never gets deployed. Perhaps the most stringent of certification processes is Apple's for its iPhone. Stories abound of the iPhone application certification process rejecting applications for apparently subjective and arbitrary reasons. As was the case for a Pulitzer prize winner's app, which merely displayed his Pulitzer-prize-winning cartoon editorials on the iPhone[41].

The certification process acts as a gatekeeper to the network operator, device manufacturer, and/or platform developers' distribution channel. An application that fails to meet certification guidelines can then be denied entrance to the distribution channel. In some cases, as with iPhone applications, there may be only one distribution channel. This ultimately blocks an application from being deployed at all. This has serious implications on attempting to develop two versions of a mobile application to target two niche markets. One can see that attempting to develop two applications might incur more than double the development time as an additional investment in certification may be required.

2.3 The Potential of Platforms

If you recall, the reason for the fragmented mobile market is due to the inability for an application written for one combination of network, device, OS, and platform to operate on another combination. However, the purpose of an application development platform is to remove the interdependency between applications and the OS, device, and network network specific API[43]. A platform is supposed to accomplish this by packaging up API from the different network, device, and OS layers into a single standard set of API. The platform then takes the form of an Adapter pattern, as described in Design Patterns by Gamma et. al[3], for the underlying hardware and software configuration. With a widely adopted standard application platform, a single application could be compatible with any hardware and software configuration that supports the platform, much like Java and Flash. So then why have we not achieved interoperability and why is the market still fragmented? There are four main factors that contribute to the inability for mobile application platforms to have a defragmenting effect on the market, which will be explained in the sections that follow. The first two factors affect open platforms, or platforms which do not impose restrictions on the types of devices and networks on which the platform can be used. The third factor affects proprietary platforms, or platforms that strictly enforce the types of networks and devices which are allowed to host the platform. The fourth and final factor that will be discussed affects both open and proprietary platforms alike.

2.3.1 Incomplete Adoption of Open Platforms

The first factor, and perhaps the most prevalent among open platforms, is incomplete adoption by the device manufactures of the platform specification[44]. The unrestricted nature of open platforms makes them extremely susceptible to two problems. Any device manufacturer can decide to implement the necessary portions of the open platform and host it on their hardware. It is because of this lack of restriction that open platforms are unable to police the device manufacturers and network operators that host the platform. This is not as big of a problem for proprietary platforms because they have much more control over the entities that choose to support them.

2.3.2 Multiple Versions of Open Platforms

The problem of incomplete adoption is exacerbated by the evolution of an open platform as well[44]. As a platform comes out with new versions, the older versions are still available on devices. This is not as big of a problem if new versions are written to be backward compatible with applications written for older versions. That is to say the new version of the platform can still run applications written for older platforms. Even with backward compatibility though, some devices will never be able to run applications written for the new versions of the platform. This is because providers have no way of demanding that device manufacturers provide implementations of these new versions on what has become obsolete hardware. As is the case with the problem of incomplete adoption, issues concerning multiple platform versions are not as prevalent on proprietary platforms because they have much more control over the entities that choose to support them.

2.3.3 Limited Reach of Proprietary Platforms

A third factor involves limited reach of proprietary platforms[44]. The problem is clear with device manufacturers who make use of their own proprietary platforms such as RIM's BlackBerry platform and Apple's iPhone platform. Since these platforms are only supported by their respective mobile devices, the diffusion of these platforms is directly equal to the market share of the devices.

However, these are not the only types of proprietary platforms. There are other proprietary platforms providers like BREW[29] and Flash[31] which allow other device manufacturers to support their platform. While the problem of limited reach is still a concern for these types of proprietary platforms, the cause is different and actually affects open platforms as well.

2.3.4 Competition Among Platforms

The fourth factor contributing to the inability of standard application development platforms to have a defragmenting affect on the market is the emergence of multiple standards[43]. As a single standard platform has yet to emerge from the pack, both proprietary and open platforms are experiencing limited reach in the market. Due to the incompatibility of applications written for different platforms, the market remains fragmented.

2.4 Android and the Entrepreneurial Developer

The intent of this section is to determine whether Google's Android platform is beneficial for the entrepreneurial developer. To determine this we need to evaluate all that comprises the Android platform. Evaluation is done by identifying the effects that the Android platform components have on the problems caused by market fragmentation and its potential as a standard mobile application development platform. For the Android platform to be considered beneficial, it should have a net positive affect on the problems facing the entrepreneurial developer as a whole.

2.4.1 What is Android?

Google describes the Android Mobile Development Platform as, "the first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone, but without the proprietary obstacles that have hindered mobile innovation"[42]. The Android platform, often referred to as just Android, is what some might think of as the first step towards a standard specification for mobile phones. Android provides[28]:

- A loose description of what the hardware (or phone) should be capable of in order to support the software stack.
- A Linux kernel providing the hardware interface, memory management, process control, etc.
- Open source libraries for application development.
- A run time to host and execute Android applications, which is the Dalvik Virtual Machine, like a JVM.
- An application framework for exposing system services to applications.
- A user interface.
- Some pre-installed applications (phone, messaging, maps, etc...).
- A software development kit.

In the years following its release by Google Inc., Android has quickly acquired widespread market adoption from major United States network operators like TMobile and Verizon, as well as device manufacturers such as HTC, LG, Samsung, Sony Ericsson, and Motorola. Each device manufacturer is responsible for compiling and creating the hardware drivers needed by the Linux kernel which their device will run. The kernel then makes use of the standard run-time host. Developers then write applications in Java using the open source libraries and application framework for getting system services. However, Android does not make use of the Java Virtual Machine(JVM) to run these applications. Instead, Android uses an optimized version of the JVM called the Dalvik Virtual Machine (DVM) that uses its own bytecode. The Java then gets compiled down into Dalvik bytecode[28]. In the end what you get is a standard set of APIs that a developer can use to write mobile applications, which will run on any mobile device and network that supports the Android platform.

Open Handset Alliance

Android was developed in cooperation with the Open Handset Alliance, which is a group of 71 technology and mobile companies. The goal of the Open Handset alliance is to"accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience"[1]. The group of companies are committed to deploying mobile devices and handsets that support and use the Android platform.

2.4.2 Effects on Production Related Decisions

Linux Operating System

The Android platform includes with it a version of the Linux Operating System. The consequences of this are that it prevents mobile developers from choosing their own operating system to develop for. While this removes some of the freedom from the developer, the affect is rather quite minimal. This is due to the fact that the purpose of a standard mobile application development platform is to remove dependencies between an application and the technologies on which it is built[43]. From the developers point of view, the application should never have to interact with the operating system. If an application really needs to make use of an operating system component that is not supported by the platform, then an API for that component should be added to the platform. Therefore, as long as the Android platform API provide broad enough coverage, then the developer will not even notice the operating system.

IDE's and Toolchain

Developing Android applications requires the use of specific tools that are needed to compile and sign the application files. However, the complete toolchain uses free and open source software. While at first this may seem to be a hindrance to require the use of certain applications, the applications are freely downloadable[26]. Additionally, the Android development platform provides the Android Development Toolkit, which is a plugin for Eclipse. The plugin allows you to download and integrate the complete toolchain necessary for developing Android based mobile applications into the Eclipse IDE[17]. Furthermore, the Android platform makes no requirements that you use Eclipse to develop[18]. Thus individual developers are free to use whatever IDE they choose. Therefore the net affect on IDE and Toolchain related decisions caused by the Android platform is a positive one, as Android provides a free and open solution to toolchain usage and allows for alternative IDEs to be used.

Technology Choices

Development of applications for the Android platform is more stringent on some core technology choices. In particular Android requires that developers code in Java. The Android platform then provides core library implementations that match most of the standard Java libraries[28]. This is more of a benefit for developers than it is a restriction since it doesn't require developers to learn a language which is only used by the development platform. For developers who are familiar with Java, this is a convenience. For developers who are not familiar with Java it is neither an inconvenience nor a convenience. This is because if the language had been an Android specific language then developers unfamiliar with Java would have had to use an unfamiliar language anyways.

The Android platform makes additional requirements on other technology choices like databases and graphics libraries. Android makes use of SQLite and OpenGL libraries. These libraries are then wrapped up in API that is exposed to developers through the platform[28]. This is a downside for developers in that they no longer have the choice to use the database or graphics library of their choice. However, this downside is offset by the standardizing affect that this has on applications. An application written for the Android platform is guaranteed to be interoperable with any device supporting the Android platform. Developers no longer have to worry about whether the host environment has support for a particular database or graphics library.

Application Priority

Something special about the Android platform is that all applications are considered equal. "Android does not differentiate between the phone's core applications and third-party applications"[1]. This benefit is almost entirely unique to Android; the idea that any application written for the platform can be replaced by another. This opens the market for developers to produce applications for mobile devices which were once exclusively developed by the platform or device manufacturers.

2.4.3 Effects on Distribution Related Decisions

Certification

The Android security policy is such that all applications must be digitally signed with a certificate[2]. This policy has less to do with controlling which applications are allowed to be installed and use certain APIs, as is the case with BlackBerry applications[40]. Instead, this serves as a form of author identification and is used to establish trust relationships between applications[25]. The important thing to note here is that the applications need not be signed by a certificate authority, and in fact it is commonplace to self sign. What this means for developers is that there is practically no investment to be made on certification efforts, as the tools needed to sign an application are freely distributed without so much as even requiring developer registration.

Distribution Channels

The main distribution channel for the Android platform is the Android Market. The Android Market acts as a content aggregator for Android applications. It allows developers to post and sell their applications as well as let users browse, download and purchase applications. There is a small up front investment in order to use the Android Market. Developers must register with the service using a Google account, agree to the market's terms of service, and pay a nominal fee of $20 USD[22]. An added benefit of the Android Market is that it guarantees compatibility of downloaded applications. Android Market does this by not displaying applications to users whose devices are not capable of running a particular application[39].

In addition to the Android Market, the Android platform makes no restriction on the source of applications. As such, a developer may use the services of an alternative content aggregator. The Android platform even allows users to self install applications, effectively bypassing mainstream distribution channels altogether. Such openness with respect to distribution enables the developer absolute freedom in their choice of distribution channels.

2.4.4 Android as a Defragmenting Force

Addressing Issue of Incomplete Adoption

The Open Handset Alliance formally states that it and its 71 members are committed to "making the initial version of the platform a commercial success"[1]. These members include all of the device manufacturers choosing to support the Android platform. As Dan Morrill, Google's Open Source and Compatibility Program Manager puts it, "OEMs are generally pretty motivated to ship compatible devices"[39]. Thus, the Android Developers Guide publicly claims that all versions of the Android platform have "the same API no matter what kind of device it is installed on."[7].

What happens when a particular device has no hardware that maps to a particular set of APIs. For example, if a device does not have GPS hardware how can it possible support the API? This particular problem is addressed by means of application filtering built into the architecture of the platform. In a manner similar to the way that the Android market prevents applications from being presented to mobile devices for which they are incompatible, the platform itself prevents mobile devices from being able to see the incompatible applications[7].

Addressing the Issue of Many Versions

If you recall, the issue of having multiple versions of a single platform distributed throughout the market is concerned with the incompatibility of new versions with applications written for older versions, or backwards compatibility. Dan Morrill claims that backwards compatibility is guaranteed, as "apps written properly for older versions also run on the newest versions"[39]. However, the Android Developers Guide still concedes that this might be a problem, and enumerates instances in which it has been. However, this problem is addressed again by the architectural filtering mechanisms outlined in the previous section regarding incomplete adoption[7]. In the best case scenario, allowing the word of Google's Open Source and Compatibility Program Manager to pass as law, backwards compatibility is guaranteed. In the worst case scenario, there exist mechanisms for preventing applications from being deployed and used on incompatible devices.

Addressing the Issue of Potential Market Reach

The Android platform seems to have done everything possible to overcome the potential of limited market reach. It has garnered the support of more than 10 different mobile operators and 18 mobile device manufacturers[1]. With more than 60 different mobile devices, selling 100,000 units a day, the Android platform definitely seems to be making a play for achieving widespread adoption[39]. Supporting this argument is data from NPD that has Android platform enabled devices with the second highest market share among smartphone sales in the U.S. for the first quarter of 2010, ahead of the iPhone and behind the BlackBerry[4]. At the very least, one can say that Android is trying its hardest to do everything possible to address the issue of potential market reach.

2.5 Wrapping Up

To summarize, the Android platform attempts to solve many problems introduced by mobile application development. While it imposes restrictions on technology choices like database support, graphics libraries, and operating system, these restrictions are negated by the standardizing affect the APIs that wrap these technologies have on applications. Ultimately, this boils down to sacrificing choice for interoperability and compatibility among mobile devices.

The platform excels in the realm of providing complete free and open source toolchains, while at the same time providing developers with the freedom to choose their own IDE. The excellence of the platform continues with its equality model for applications, as third party applications operate with the same priority as applications provided by the platform. The certification process is very developer-friendly, yet uncompromising when it comes to security. By choosing the Android platform, developers are provided with perhaps the most freedom among choices for distribution. Lastly, the Android platform addresses the problems of multiple versions and incomplete adoptions with an elegant mix of corporate cooperation, commitment to vision, and architectural platform safety measures. All of these features of the Android platform enable it to achieve widespread market adoption.

3 Related Work

3.1 Integrating Mobile Devices Into the Computer Science Curriculum

In this paper, Mahmoud presents his experience with and strategies for integrating mobile devices into the computer science curriculum[37]. He makes the case that mobile application development should be included as a core topic in undergraduate education. This case is argued not for the purposes of creating mobile development specialists, but rather for introducing students to mobile development from a software engineering perspective. One supporting argument is the fact that mobile devices and their application development provide a multidisciplinary educational breadth. Such education covers topics like programming, design, software engineering, human computer interaction, and many other computer science topics. Mahmoud then outlines his strategy.

Mahmoud recommends integrating mobile devices across the entire computer science curriculum. His claim is that development for mobile devices should be introduced as early as possible, to new and beginning students. This introduction should then extend into intermediate and advanced courses, such as databases, operating systems, and data structures. Where possible, this education should be included in the laboratory components of courses as well. Lastly, Mahmoud feels that mobile development should be incorporated into project-based courses like senior capstone projects.

Mobile devices are integrated into five of Mahmoud's computer science courses. Two of the courses are for first year students and introduce them to the mobile development paradigm. These courses use labs to reinforce the theory that students learn in class and require them to implement a few applications for BlackBerry devices. The next three courses include a project-based Distributed

Systems course and a two semester long Senior Capstone project. These project-based courses allow the students to design and develop mobile applications for BlackBerry devices as a means of fulfilling the course requirements.

Mahmoud is using this experience to design and develop an academic kit to help other universities integrate mobile devices into their computer science curriculum. Along the way, Mahmoud uses a series of surveys, questionnaires, and exit reviews to evaluate their work. The kit will be composed of modules which include teaching materials, lecture notes, labs, assignments, and tool guidelines. This work is being done for the Centre for Mobile Education Research, which is funded in part by RIM.

3.2 Smartphone Software Development Course Design Based on Android

Hu, Chen, and Lou present their design for a software development course for smartphones based on the Android platform[37]. They use their experience designing embedded systems and multicore programming classes to develop a class that teaches both the theory and practice of smartphone software development for Android. While designing the course, they used three principles to guide their decisions. First, the course should teach students a framework for smartphone development to help them develop on their own. Second, the content of the course should put an emphasis on practice over theory. Third, the learning objectives of the course should align with the needs of industry.

To honor their first design principle, Hu, Chen, and Lou, construct a syllabus that aims at taking a framework approach to learning objectives. They begin with an overview of smartphones and smartphone operating system before diving into the details of Android. By taking this approach students learn about smartphones in general before they learn about Android specifics, which will help them transition to learning to develop for other smartphones. The syllabus then covers detailed Android topics like architecture, kernel, run-time environment, application framework, and software development.

Hu, Chen, and Lou's careful design of laboratory exercises is consistent with their second design principle which requires them to focus on practice over theory. They create 64 credit hours worth of laboratory exercise that enable the students to master how to program for Android devices. These exercises are divided into lab assignments of three different types difficulty. There are 17 lab assignments for which the difficulty progressively increases from basic hands on labs, developing programming practice, all the way to comprehensive development requiring students to complete novel applications.

Lastly, Hu, Chen, and Lou innovate by incorporating new and exciting features into their course. They begin by bringing in engineers from industry to help teach the course, which allows them to align the learning objectives of the course with the needs of industry. Then they require students to participate in both online Android communities and physical technology communities. This teaches students how to positively contribute to a community. Lastly the course offers an application development contest which encourages students to organize themselves into teams for the purposes of creating a novel application.

4 Design Approach

This section identifies all of the major factors that influenced the design of the labs. The labs are being used for a course on mobile development and as such it is important that they incorporate elements that reinforce all that it stands for. Secondly, the labs are being used to teach students everything they need to know to be productive Android application developers. This covers a lot of information in a short amount of time and thus it is crucial for all that is essential to be covered. Lastly, the automated testing features provided in the Android application platform provides an excellent opportunity to both introduce students to the importance of testing and to develop an automated system for grading. The subsections that follow address all of these factors in more depth.

4.1 Mobile Development

Developing applications for mobile devices introduces a large number of challenges which developers of other applications most likely will not encounter. It is important that the labs introduce students to these challenges and offer them either solutions or strategies for dealing with them. In particular, the hardware for mobile devices is quite limited when compared to today's powerful desktops and laptops. Mobile devices offer significantly lower processing power, limited RAM, limited storage capacity, limited battery life, smaller and varying screens, and less reliable network connections [38]. While a lot of these used to be problems for developers in the past, they are most certainly new to young developers who have never had to develop a program for a processor whose clock speed is measured in megahertz. The labs should foster an environment for students to think critically about how their solutions are affected by these limitations.

Efficiency

The labs should teach students how to develop efficient applications for resource limited devices by making them aware of the performance and power costs associated with various programming practices. This includes informing students of things like the performance costs associated with instantiating objects and the various techniques for mitigating those costs. As an example the labs should introduce students to the idea of caching and reusing objects instead of instantiating new ones[15]. Additionally, the performance of an application has a direct link to power consumption of the device. Students should be taught that conservation of power has a direct relation to writing efficient code[19].

Responsiveness

Users expect their applications to be responsive, meaning that their applications do not hang or freeze[16]. While this is true of other applications as well, mobile applications have the added difficulty of accomplishing this feat with limited hardware resources. Thus, the labs should help the students to identify potentially slow operations and teach them how to make use of background threads to perform the necessary work. Additionally, the use of progress dialogues to show users that an application is still working properly can be beneficial as well.

Adaptive User Interfaces

The labs should introduce the students to the problems associated with designing user interfaces for mobile devices with varying screen sizes and limited space. There are many strategies for accommodating multiple screen sizes, especially those with limited space. Students should be introduced to as many of these as possible. Among these strategies are decomposing a UI into several screens to prevent overloading a screen and developing different UIs for different screen sizes[19].

Seamless Performance

Mobile devices are often used in a multitasking nature[38]. Meaning that a user might switch back and forth between a number of running applications at a time. It is important that this switch be performed as seamlessly as possible. The transition should be fast, not cause data to be lost, and the application should not tie up unused resources while it is not in use[19]. The labs need to teach the students how to efficiently save and restore the state of their applications. Additionally, students need to fully understand the lifecycle of their applications to perform these operations at the correct time.

4.2 Android Essentials

The Android application platform provides developers with a vast number of resources to use for developing their applications. Teaching every single piece of the platform is impossible, given that the course is only ten weeks long. Thus, only the most important aspects of the platform can be taught to the students. The goal is to provide them with enough depth of study of the core components along with some breadth of useful additional features that they can work successfully on their own. We identified a list of the application components, platform components, hardware, and libraries which we find to be the most important topics to teach the students in the allotted time.

Application Components

The Android application platform allows an application to share elements of itself with other applications. This requires the system be able to start or stop an application when its elements are needed by other applications. Thus, Android applications are composed of components that can be instantiated and run individually[9]. This feature is central to the Android platform and thus it is the most important topic that the labs need to teach. There are four different types of components and the labs need to represent each of them as completely as possible.

- **Activities:** Present a UI for a single cohesive task. Examples of this include choosing a contact from a list of contacts or composing an email to send to someone. Applications can be composed of many linked Activities or just one, but there is usually only one Activity that is displayed when the application starts up. It is important for students to understand the lifecycle of Activities so that they can seamlessly save and restore their state as the user switches between them and other applications.

- **Services:** These do not have a UI, but rather run in the background to perform some type of work. Examples of this include playing music while the user interacts with another application or fetching data from a network connection. Services are often used to create and start worker threads to perform the work that needs to be done. These are very useful for improving the responsiveness of applications and as such they should be included in the labs as well.

- **ContentProviders:** These are used to share an application's data with other applications. For example an email application might wish to query the phone application's list of contacts for an email address. The email application would interact with the phone application's ContentProvider to ask for this data. ContentProviders provide a layer of abstraction between the data and the manner in which it is stored. Storage can be implemented by an application by using a database, files, or however it sees fit.

- **BroadcastReceivers:** These listen and respond to systemwide messages. Applications have the ability to broadcast messages to the entire system via objects called Intents. Any application that listens for these messages must implement and register a BroadcastReceiver for the type of message they want to listen for. The BroadcastReceiver can then execute some code to respond to the messages. For example, the system might send out a message when an internet connection has been established. An email application could then have a BroadcastReceiver listening for this message which pushes any emails waiting to be sent and pulls any unreceived emails from a server.

Additional Platform Components

Aside from the main components that make up an application, the Android platform offers a number of other features that most, if not all, applications will make use of.

- **View Library:** Android's UI library is composed of a hierarchy of View and ViewGroup objects. The View class serves as the base of the hierarchy and is extended by all other classes. The ViewGroup class directly extends the View class and is used to group other View objects and control their layouts[27]. For example, a UI might use a TableLayout ViewGroup to display a grid of Button View objects. Understanding how to make use of and extend the View library to serve the needs of an application is imperative to implementing any type of UI for an application.

- **Resources:** Externalizing application resources like strings, values, UI layouts, and images is an important practice that is strongly encouraged by the Android application platform. The platform provides a system for including these resources into an application and accessing them at runtime. This allows developers to maintain these resources independently from their application code. Additionally, it enables applications to be reconfigured on the fly[10]. For example, lets say an application is pulling the text it displays from English String resources. If the user changes the language on their device from English to Spanish, the application can change the text it displays to use its Spanish String resources.

- **Intents:** Messages used to pass information between application components and to start other application components. Intents are data structures that contain a description of an operation to be performed or an event that has happened[20]. They can be broadcast by application components to tell other application components that something has happened, or they can be used to ask the system to perform some type of operation. For example, one Activity can explicitly start another Activity by broadcasting an Intent to do so. Alternatively, a Service can notify anyone that cares that it has finished downloading new emails by broadcasting an intent with that information.

Additional Hardware & Libraries

The following list of platform libraries and hardware features are commonly used among developers to implement their applications.

- **SQLite Databases:** The Android platform allows applications to create and maintain private SQLite databases[14]. These databases provide a robust, fast, and efficient solution for persisting application data locally on a mobile devices.
- **Internal & External Storage:** Android applications have permission to create and edit files on both the internal and external storage space of a mobile device[14]. The Android platform provides standard libraries for creating and editing these files, which rely heavily on the the java.io library.
- **Network Connections:** Making network connections to send and retrieve data is extremely important. As such, the Android platform provides a full implementation of the java.net library. Additionally, it provides its own android.net library to help with network access beyond the APIs provided in java.net[14].
- **Google Maps:** The Android platform supplies the Google Maps Library to allow developers to add mapping capabilities to their applications. This

library allows developers to create compelling map based functionality for their applications. In addition to being able to download, render, control, and display maps, the library allows developers to add their own information to maps by creating custom overlays[21].

- **Location Services:** Android applications have the ability to incorporate location based functionality by querying the device's LocationManager. If a device has a service that can provide the device's location it will do so. This service may make use of GPS hardware, network information, or some other means of obtaining location information. [21]
- **Camera:** If a mobile device has Camera hardware then an application may access it through the android.hardware library provided by the Android platform. Using this library, an application can adjust image capture settings, view camera previews, take pictures, and record video. [12]

4.3 Automated Grading

Each laboratory assignment, or lab for short, developed in this thesis requires that every participating student produce their own functional application, unique to that lab. Given a class of 30 students and seven different labs, the result is roughly 210 independent applications that each need to be graded. Complicating matters further, the seven different labs all have different acceptance criteria. Checking each of these 210 applications for seven different sets of acceptance criteria is a burdensome task that requires the allocation of time and resources which could be better spent elsewhere. This investment has to be repeated every time the class is offered.

4.3.1 Opportunity

Grading all of these applications is a tedious and repetitive task that is prone to human error. However, the same things that make grading a burden also make it a prime candidate for computer automation. Additionally, grading Android applications in particular lends itself to even further automation. This is due to the fact that the Android SDK provides a tightly integrated framework for testing. This testing framework is an extension of the popular JUnit Java testing framework. By using the testing framework to automate the grading process we develop a reliable and reusable alternative to manual grading. Investing time and resources to automate the grading process just once greatly saves in future grading investments every subsequent time the course is offered.

4.3.2 Grading Needs

The process of grading takes a student's implementation of a solution to a problem given by a lab and outputs a grade for that student. To determine the output, the grader needs to analyze the solution for correctness with respect to the specification outlined in the lab. While the output of a grade is the immediate need of a grader, there are requirements surrounding that need as well as additional objectives achieved through grading. Thus any automated solution to grading must have the following qualities:

- **Usable:** The solution should be easy to use and require little to no human interaction or oversight.
- **Reusable:** The solution should be reusable for future classes.
- **Accurate:** The solution should grade every submission as correctly as possible, at least providing better accuracy than manual grading.
- **Traceable:** The criteria used for determining grades should be able to be traced to specific requirements outlined in the labs.
- **Visible:** The grading results should be presented in an easily readable format.
- **Transparent:** The grading results should provide a detailed explanation of how a grade was calculated.
- **Insightful:** The grading results should provide insight into the quality of the assignments and the grading criteria.

5 Implementation

This section provides detailed implementation information for each component of the thesis. In particular, a general structure was used to develop all six labs. The details of the this structure are covered first and provide insight into the components that make up every lab. Next, each lab is addressed individually in the order in which they were developed and are assigned during the course. These individual lab sections provide insights into the educational contents of each lab. Lastly, the automated grading system developed for the course is laid out in detail. This includes a description of how the system functions, its architecture, and implementation details of all of its components.

5.1 Lab Structure

A common structure was used to govern the development of all six labs used for the class. In particular, each lab contains a detailed manual that instructs the students how to complete the lab, a completed application to reference as the solution to the lab, an application project skeleton the students use as a code base for their implementations, a means of distributing the lab, a means of collecting the lab, a procedure for grading the lab, and a survey to be completed by the students at the end of the lab. This section on the structure used for developing the labs briefly describes the process used to create each of these components and then provides more details on each of the individual components.

5.1.1 Development Process

The first step for developing a lab is to create a list of educational goals the lab will have the students achieve. This list of educational goals will then be used to develop an application whose implementation process serves as a vehicle for students to attain all of the educational goals. This completed application is then used as a solution to the lab. Next we make a copy of the application solution's source code and remove any implementation details relating to education goals. This leaves only method stubs, member variable declarations, and implementation details unrelated to the education goals. This copy of the application solution, referred to as a project skeleton, is distributed to the students and used as the code base for their implementations of the labs.

Using both the project skeleton and the application solution we then develop a detailed laboratory manual for the students to follow. This manual instructs students on how to complete the lab by filling in the missing implementation details in the project skeleton. The intent of the manual is to coach the students to implement an application that mimics the application solution for the lab. The hope is that by having the students implement functionality relating to the educational goals, they then achieve those goals.

The lab manual is then referenced as an application specification to develop a set of tests to use for grading. These tests identify pieces of functionality specified in the manual that relate to educational goals. The tests can then be used to determine whether a student has met an educational goal from the result of a test. Lastly, a grading rubric is developed for the lab, which assigns point values to all the tests and is used to translate test results into a grade.

5.1.2 Application Solutions

The application solutions provide the foundation for each lab. They are designed to incorporate all of the Android application platform components the students are supposed to learn. The knowledge that students with little to no experience programming for Android will also have to produce a similar application is taken into consideration while implementing and designing the application solutions. Also taken into consideration is the fact that implementation instructions will have to be written based off of the application solutions as well. This means that the source code should be written as modularly as possible to make instructions for filling in implementation details describable by referring to a particular method stub. Additionally, each application solution is used as the basis for grading. The tests used to grade the student submissions are written for the application solution. It is important that the application solutions be easily reproduced by the students and be as correct as possible.

5.1.3 Project Skeletons

A project skeleton is developed for each lab and distributed to every student. They are used as a code base by the students to complete the lab assignment. The project skeletons are copies of the application solution's Android project files with the most, if not all, of the implementation details removed. Method signatures with empty bodies, member variable declarations, and comments are left in the source code for the students to use. It is imperative that the project skeletons remain consistent with their respective application solutions. Meaning that if an application solution is refactored, its project skeleton should be refactored as well. This includes things such as resource file changes, variable or method renaming, and variable or method signature changes.

Students are required to use the project skeletons as the basis for their implementations instead of implementing their own solutions for a number of reasons. To begin with, project skeletons help teach more topics in a shorter amount of time because they allow unnecessary implementation details to be provided for the students. This includes things like setting up the projects themselves, designing the architecture of the application, and implementing supporting classes. While these are all important things for students to know how to do, they are not the learning objectives of the labs. Furthermore, since it is an upper division class, students are expected already have experience with these things. The students will get the experience with all of these things from the course project, which they must implement from scratch on their own.

The goal of the lab assignments is to provide tutorial style contextual examples of how to use the Android application platform components. Using project skeletons highly facilitates both the writing of the lab manuals and the students ability to follow along. They do this by defining a standardized set of methods and member variables to use for implementing the labs. These methods and variables can the be referenced by the lab manuals and easily identified by the students as they follow along. The lab manuals can tell the students to implement something using a particular Android application platform component, in a particular way, and specify the method and member variables their implementation should use.

Lastly, the use of skeleton projects are crucial to allowing student submissions to be graded in an automated fashion. Automated tests can be written for the application solutions and be used to accurately test student submissions since they are required to use the same classes, methods, and member variables in their implementations. Such tests could not be written if every student wrote their own application from scratch.

5.1.4 Laboratory Manuals

The lab manual is the most important component of every lab. Each lab manual serves as the specification that provides a detailed description of what the students are expected to learn, what they are expected to implement, and how they are expected to implement it. Each lab manual begins with an overview of the application that the students will be implementing. This is followed by a list of the educational goals that the lab hopes students attain via completion of the lab. The remainder of the lab manual is devoted to a detailed application specification which describes how to implement the lab assignment.

The application specification portion of the lab groups implementation details into sections, subsections, and subsubsections. These groupings attempt to organize the content in a manner that makes the most sense from both implementation and educational viewpoints. Implementation details corresponding to common educational goals are grouped into the same or neighboring sectional units. Additionally, the natural order of implementation details is respected by ordering sectional units in a similar manner.

Each sectional grouping of implementation details also includes a brief overview of the contents of that sectional unit. Screen shots of the application are displayed to provide the students with a visual description of what they are implementing and to aid in visualizing use case scenarios. Information is given on how these implementation details tie into the application as a whole, their relevance to educational goals, and what features of the Android application platform the students will be using. Information on features of the Android application platform are generally accompanied with a brief overview of their purpose, links to additional educational articles, and documentation on the Android developer website.

When implementation details are given in the application specification sections, clarity is the ultimate concern. The active voice is used to make clear and direct specifications of requirements. In some cases, examples, tips, and hints are provided as well. These come in the form of sample code, links to additional relevant documentation or educational articles, and mistakes or problems to watch out for.

5.1.5 Dispersal & Collection

The labs are designed to be distributed and collected entirely via the internet. Lab manuals are implemented as web pages on a publicly accessible website to enhance the learning experience. The skeleton projects are also distributed on this same site. Submissions are handed in via a website. By using web pages to disperse and collect the the labs the experience is much more graphical, interactive, reliable, and accessible.

In the lab manuals high resolution screen shots can be grouped together and displayed as use case scenarios. Additional links to external articles and documentation can be referenced in the lab manuals, allowing students to quickly access more detailed content. Updates and corrections to both the lab manuals and the project skeletons can be made from anywhere and the students will automatically see the updates on the site. Lastly, the students also have the ability to access the content and submit their implementations from anywhere. This helps enables the class to be taught online over the summer.

5.1.6 Grading

When students complete the lab assignments they are required to submit an archive file containing their entire Android application project to an online repository. This ensures that all source code and resource files are submitted. All submissions are then individually recompiled, tested, and assigned a grade. Each lab has its own suite of tests and a grading rubric used to translate a student's test results into a grade. A copy of the test results and grading rubric are then handed back to each student.

A test suite for a lab contains a set of acceptance tests that each determine whether a student's application has properly implemented some piece of functionality. Each acceptance test is derived from specific implementation details outlined in the lab manuals. When possible, these tests are written in an automated fashion. Automated tests make use of the Android Testing Framework and are quite similar to executing JUnit tests. More information about automated tests can be see in the Implementation section on Automated Grading. However, sometimes due to the highly interactive nature of some of the applications, it is not possible to write automated tests. In these cases, acceptance tests are implemented as a series of use case scenarios written in plain English that describe the expected behavior of the applications in a highly detailed manner.

Grading rubrics are developed independently for each lab as excel sheets. They assign point values to every test in their respective test suites. These point values are determined by taking into consideration both the difficulty of the implementation detail relative to the rest of the application and the significance the implementation detail has with respect to the educational goals. The rubrics are then copied and the test results are manually filled in for each student submission. After filling in a rubric for a student submission, the grade value is calculated automatically.

5.1.7 Student Surveys

Surveys are distributed to every student at the end of each lab manual. The student surveys for all labs are implemented as identical SurveyMonkey surveys and a link to each one is included at the bottom of each lab manual. The purpose of the surveys is to gather feedback from the students to determine whether the labs are meeting their performance goals, to identify any problems, and to identify any successes. This data is then used in conjunction with the grades from the labs to attempt to evaluate their performance. The data is used to evaluate each lab individually and all the labs as a whole.

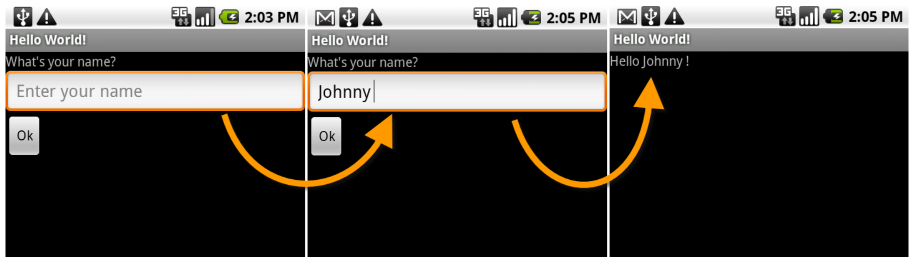


Figure 1: "Hello World!" Application Screen Shots

5.2 Lab 1

The goal of the first lab is to teach students the fundamentals of developing Android applications, from project creation to installation on a physical device. More specifically it is intended that students learn how to use the basic development tools to support the application development process, as well as the major components of an Android application itself. The lab accomplishes this by having the students set up their own development environment from scratch, develop a basic "Hello World!" application, run the application on the Android Emulator, and deploy it to a physical device.

5.2.1 Setting Up the Development Environment

Before one can develop for Android devices one has to have an environment to develop in. At a minimum this includes the Android Software Development Kit (SDK) and at least one SDK platform component. Additionally, Android also offers an IDE for developing in Eclipse. It is important that the students learn how to install and configure such an environment as you cannot develop without it.

- **Downloading & Installing the Android SDK** The students begin by downloading and installing the Android SDK distribution. The SDK has three different distributions; one for Windows, one for Mac, and one for

Linux. We do not place a requirement on the students to use a particular operating system. The SDK comes complete with a set of command line tools for building and deploying applications as well as a few useful activities.

- **Downloading & Installing the Eclipse Plugin** After installing the SDK students have to download a copy of the Eclipse IDE and install the Android Development Toolkit (ADT) which is a plugin for Eclipse. Students are encouraged to develop with the ADT plugin and are required to use it for at least the first lab. The ADT simplifies the development process with features like wizards for project creation, custom editors for android-specific files, and a basic user interface (UI) builder just to name a few.

- **Downloading & Installing Platform Components** Lastly, the student students must download the appropriate version of the application libraries that they will use. Just as there are different versions of the Android platform, there are different version of its SDK. Each version of the platform has a corresponding SDK component. Applications must be developed on the earliest SDK component corresponding to the earliest Android platform version it is designed to run on. The SDK, however, does not come with any of these components pre-installed as they are large files. Instead the SDK provides you with a management tool for choosing which components to download and update.

5.2.2 Creating an Android Project

The first step to creating an Android application is creating the project for it. Creating the project is a simple enough task by using either the SDK command line tools or the ADT project creation wizard. We have the students use the

ADT project creation wizard as we expected most, if not all, of the students to be developing in Eclipse using the ADT plugin. The students are provided with all of the details need to to properly complete the wizard so that the resulting project is consistent across the class. As an added benefit, the resulting project from the project creation wizard is a fully functional application that displays "Hello World!" on the screen and can be installed and run on an Android device or emulator.

5.2.3 Understanding Components of an Android Project

The application project the students create is very similar to other Java applications, however, there are some important differences. In particular, there is a directory structure for certain file types as well as important configuration and definition files that the build process uses. The good part about using the project creation wizard and the command line tools for creating a project is that these components are set up for you automatically.

However, changes to any of these components without fully understanding them could result in a failed build or problems with an application. Thus, the students are required to read through a brief overview of each of component in a tour like fashion. The students are asked to locate a particular component in their project, and then are given a description of the purpose of the component and the implications it has for the rest of the project.

5.2.4 Using the Android Emulator

Learning how to use the emulator is very important as the Android devices that the students are given has to be returned at the end of the class. In order for students to continue developing on there own after the end of the class they would have to either purchase their own device or make use of the emulator.

Aside from operating as a fully functional supplement to a physical device, the Android emulator is immensely useful for debugging purposes as well. Some debugging tools provided by the Android SDK do not work on the physical devices as they are not always granted root access to the device. These tools do work with the emulator. Thus the students have to start up an emulator and run their “Hello World!” application created by the project creation wizard on it.

5.2.5 Deploying an Application to an Android Device

It is always beneficial to test an application on a physical Android device if possible. In particular, the emulator tends to be much slower and can consume a lot of time when attempting to debug or execute test suites for an application. Additionally, it is possible that some bugs exist which only manifest themselves on certain Android devices. For these and other reasons the students are tasked with running their “Hello World!” application on a physical device with debugging enabled. Performing this involves making a simple modification to the application, enabling an option on the device, and installing some drivers on the computer used for development. Getting practice doing this is important as the driver installation can cause issues for some Linux users.

5.2.6 Creating a Simple User Interface

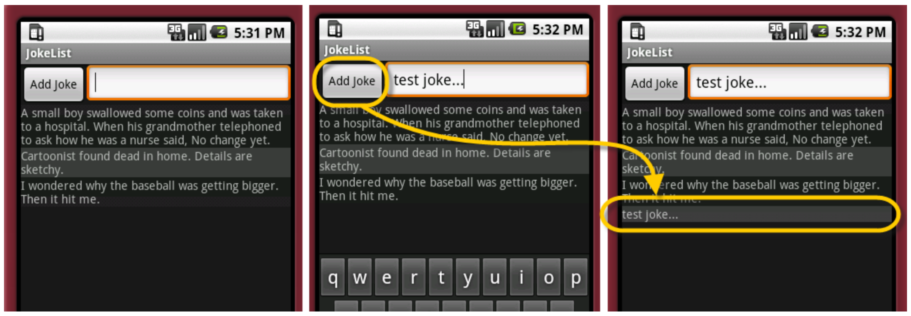
The purpose of this goal is to introduce students to the basic Android application components. This gets them to write some code and serves as an entry point for making future applications. The students are introduced to the most basic UI and application framework components. In particular the students extend their existing application by creating an additional screen that will be displayed on application startup. This screen then prompts the user to enter their name into a text box and hit a button. Then the original "Hello World!" screen is displayed along with the name the user entered on the previous screen.

- **Creating a Layout** The students start by defining the layout of their UI in an XML file in much the same way that one would make a basic HTML page. The students learn how to add XML resource files to their project and in which directory layout files should go. Then the students are introduced to basic UI elements as they add a text box and button to the layout file. They also assign each of these elements an ID so they can learn how to dynamically retrieve references to them later on in the lab.

- **Creating an Activity** With their layouts defined, the students are introduced to the Activity class, which serves as the Android application frameworks equivalent of a screen component. An Activity usually defines some sort of cohesive user interaction like selecting a contact phone number from a directory, or dialing a phone number. The students first display the layout that they created and then retrieve references to their button and text box UI elements using the IDs they assigned them. Then they add an event listener to the button so that when it is clicked the name that was entered into the text box is retrieved and forwarded to the original "Hello World" Activity. At this point, the students now know how to start other Activity classes and pass the data.

- **Editing the Manifest** The main application configuration file is named AndroidManifest.xml. This provides forward declarations to an Android device before the application ever runs all of the main application framework components, security declarations, and other application information. This serves as a contract between the application and the device so that a device will not run an application component or grant a security permission that isn't declared. The students learn about this file and some of its main purposes by adding a declaration to it for the Activity they created.

- **Calling Another Activity** Lastly, the students edit the original "Hello World!" Activity class by adding code to retrieve the name that was forwarded to it. Then they update the "Hello World!" greeting that gets displayed by replacing "World" with the name they just retrieved. At this point the students now know how to communicate data between two independent Activity components.



A small boy swallowed some coins and was taken to a hospital. When his grandmother telephoned to ask how he was a nurse said, No change yet.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

test joke...

Figure 2: "Joke List" Application Screen Shots

5.3 Lab 2

In the second lab students learn how to work with Android's user interface (UI) library by creating a "Joke List" application. The "Joke List" application allows a user to view and edit a list of jokes. The students learn about the View, ViewGroup, Layout, and Widget classes that are commonly used to build UIs. Then they use these classes to dynamically build their own UI. Additionally, the students are taught how to execute tests from Eclipse and learn in more depth about referencing data from resource files in their code and handling events generated by their UIs.

5.3.1 Executing Tests

In addition to using tests for automated grading, some tests are distributed with the labs. The purpose of this is to help ensure that students are implementing the labs correctly before handing them in. These tests also provided students with a source of instant feedback and will hopefully foster a sense of appreciation for testing in general. This lab has the students execute a single test class from Eclipse to ensure that they properly implement a class used by the "Joke List"application. The students are then told how to make use of the test results and track any test failures or errors.

5.3.2 Referencing Resource Data

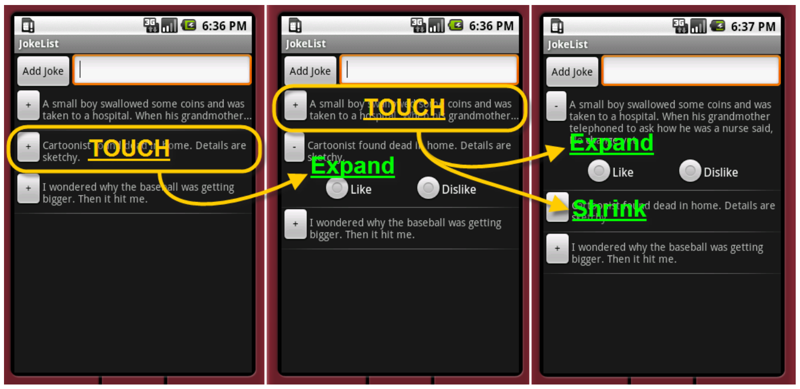
Externalizing resources such as images, strings, and UI layouts is an important feature of Android as it allows one to manage these resources independently from one's application. It is therefore important to teach students how to use Android's system for externalizing resources. This lab accomplishes this by demonstrating how the resource system works and then has the students make use of a set of string resources. Declaring and using layout resource files is a topic that is covered extensively in the third lab.

5.3.3 Declaring Dynamic Layouts

Declaring UI layouts statically in externalized XML resource files is the preferred method of defining UIs in Android. However, it is still necessary to be able to dynamically manipulate a UI at runtime. Furthermore, it allows students to practice working with different View classes and learn their interfaces in a programmatic paradigm they are familiar with. The students can then be taught how to declare and reference UIs as layout resources as a lesson separate from the one introducing them in the View classes. This separation hopefully makes the two topics easier to digest. For these reasons we have the students implement the "Joke List" application UI in their application code instead of in an externalized resource file.

5.3.4 Handling UI Events

Intercepting and processing events from the View objects that a user interacts with is imperative to creating a UI. Students learn about event driven programming by implementing event listeners and registering them with their associated View objects. They create and register listeners for touch and key events that allow users to add new jokes to the "Joke List" application.



A small boy swallowed some coins and was taken to a hospital. When his grandmother telephoned to ask how he was a nurse said,

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

Figure 3: “Advanced Joke List” Application Screen Shots

5.4 Lab 3

This lab is a continuation of Lab 2. It build on the students’ knowledge of the Android UI library and introduces new topics as well. First the students learn how to turn UIs into externalized resources by declaring them statically in XML layout resource files. Then the students learn how to extend the Android UI library by implementing their own custom View class that is used to display individual jokes. Next the students incorporate their custom joke View class into the “Joke List” application by making use more advanced UI classes. Lastly the students learn how to create context menus, options menus, and HTTP connections.

For this lab the students accomplish all of this by extending the “Joke List” application they created in the previous Lab. This version of the app provides a more polished interface and additional functionality. It allows the user to give ratings to jokes, delete jokes, filter the list of jokes, upload jokes to a server, and download jokes from a server.

5.4.1 Declaring Static Layouts

Declaring UI layouts statically in externalized XML resource files is the preferred method of defining UIs in Android. This allows UI layouts to be managed independently from an application, making it easier to supply different layouts for different devices, screen densities, and screen orientations. Students learn how to declare and inflate static layout resource files by converting the “Joke List” application’s dynamically defined layout from Lab 2 into an XML layout resource file. Then they implement the layout for a custom View component used to display individual jokes in an XML layout resource file. Examples of both can be see in figure 3.

5.4.2 Building Custom UI Components

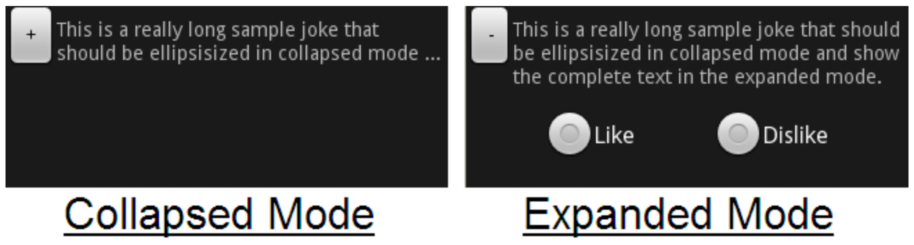

+
This is a really long sample joke that should be ellipsisized in collapsed mode ...
Collapsed Mode
-
This is a really long sample joke that should be ellipsisized in collapsed mode and show the complete text in the expanded mode.
Like
Dislike
Expanded Mode

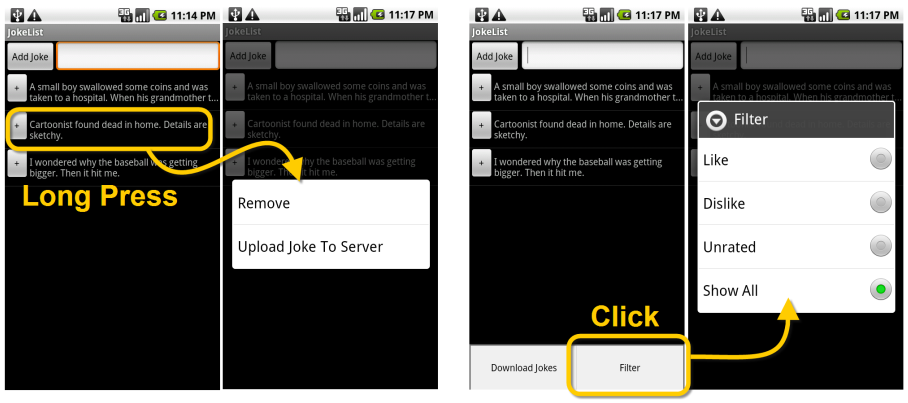
Figure 4: The custom JokeView UI component

Sometimes the standard Android UI library does not supply the functionality that is needed. In situations such as these it is common practice to define custom UI components. Students learn how to create custom UI components by extending and combining existing UI classes. This is accomplished by having the students create a custom JokeView class that encapsulates the logic to display and assign a rating to a Joke object. An example of the JokeView class can be seen in figure 4. The JokeView class has two states, an expanded and a collapsed state. The collapsed state displays the first two lines of the Joke, while the expanded state shows the full text and a group of radio buttons which can be used to apply a rating to the joke.

5.4.3 Using Adapters & AdapterViews

AdapterViews are View objects whose child Views are determined by an Adapter that binds to data of some data source [11]. The Adapter class follows the standard Adapter pattern by providing a View object when given a data object [3]. In the context of the "Joke List" application, the AdapterView is the scrollable list which contains a number of JokeView objects. These are provided to it by a custom JokeAdapter class designed to create JokeView objects for the array of Joke objects to which it is bound. This is a common patter for Android applications that need to display large amounts of similar data. Students learn about this pattern by making the "Joke List" application display JokeViews through the use of a ListView AdapterView and the custom JokeAdapter class that they implement as well.

5.4.4 Using Menus



A small boy swallowed some coins and was taken to a hospital. When his grandmother t...

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

Figure 5: "Joke List" Options Menu, Context Menu, and Filter Submenu

The Android application framework offers an easy-to-use programming interface for the creation of menus. There are three main types of menus [13]:

- **Options Menu:** This appears when the "Menu" button is pressed and generally offers options related to the Activity that generated it.
- **Context Menu:** This appears as a floating menu when a view is long-pressed and usually offers options pertaining to the view that was clicked
- **Submenu:** This floating list of options that appears as a result of clicking an item from another Options Menu or Context Menu.

The students learn how to create all three types of menus by adding functionality to the "Joke List" application that is initiated through these menus. First they create a Context Menu that appears when a JokeView is long-pressed, which allows users to delete or upload the selected joke to a remote server. Then the students add an Options Menu that allows users to download a list of jokes from a remote server or filter the Jokes that are displayed by their rating. The students also implement a Submenu for the joke rating filter options that is displayed when the filter Options Menu item is selected. Examples of each menu can be seen in figure 5.

5.4.5 Establishing HTTP Connections

HTTP connections are immensely useful for transferring data between a mobile device and a server. Such connections can be used by mobile devices for retrieving information as well as sending data back to a server. The students learn how to send and receive data via HTTP connections by implementing the functionality needed to share their jokes with the rest of the class. By making HTTP requests students connect with a server to upload their own jokes and download other students' jokes.

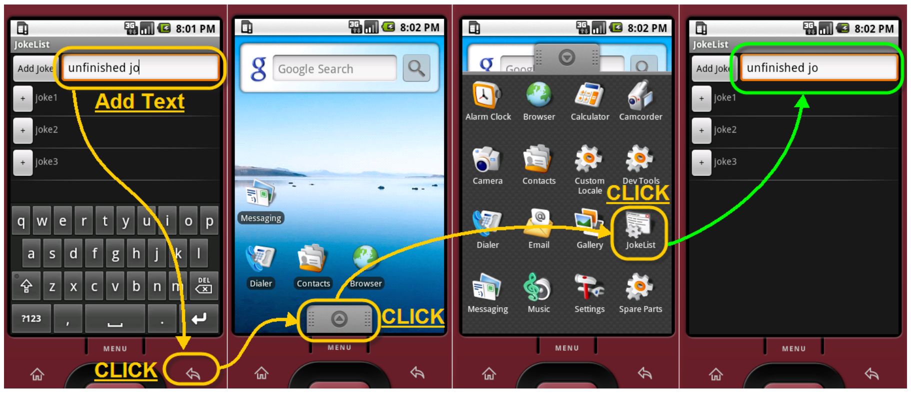


Figure 6: "Advanced Joke List" With State Persistence

5.5 Lab 4

This lab builds on Lab 3 by adding state persistence to the Advanced "Joke List" application. Students are given a solution to Lab 3 so they may work on Lab 4 even if they did not complete all of Lab 3. There are two kinds of persistent state that mainly exist, which include application data and internal state[23]. When the "Joke List" application is closed, all joke data and internal state is lost in the prior implementations of the application. For this implementation, students learn to persist joke data by using an SQLite database and preserve internal state by using a combination of Android application platform mechanisms.

5.5.1 Maintaining Internal State

Learning how to persist internal application state is important to creating a consistent user experience. It would frustrate users if all of an application's preferences and UI state were lost every time they changed screens. However, all internal state for an Activity, including things like application settings and preferences, is lost whenever it is destroyed, unless it is intentionally saved.

The Android application platform offers two mechanism for saving this state, Instance State and Shared Preferences.

- **Instance State** This is used to store Activity state for a single instance of an Activity. State is only made available to a particular Activity instance when it is recreated. When the Activity is explicitly closed by the user hitting the back button, this data is lost.

- **Shared Preferences** This is used to store application state globally for all components of an application. State is made available to all Activities of an application. State is persisted even when an application is explicitly closed by the user hitting the back button.

What makes this tricky is that there are many scenarios in which an Activity can be destroyed. The students must understand the lifecycle for an Activity in order to save and restore application state at the correct times. For example, anytime in which an Activity is not visible it is vulnerable to being destroyed. This includes explicitly closing an Activity, switching to another Activity, and changing orientation.

Students learn to how to correctly save and restore internal application state by using both Instance State and Shared Preferences. The students use Instance State to manually save and restore the text in the text field used for adding jokes by. By doing this, the text is only preserved for a single run of the application. Even in the event of an orientation change the text will remain in the text field; however, hitting the back button and re-opening the application causes the text field to be reset. The students also use Shared Preferences to allow the application to remember which filtering option the user chose. The filtering option is restored even when the application is explicitly closed and re-opened. It also makes this data available to other Activities.

5.5.2 Persisting Application Data

Learning how to persist application data is perhaps even more important to creating a consistent user experience than internal state. It would really frustrate users if all of an application's data were lost every time they changed screens. The Android application platform offers each application the ability to create its own private SQLite database. By using a database application data can be edited, saved, and restored across application component instances.

Students are first provided with a functioning database adapter implementation that they can use with their application. This comes in the form of a library jar file with hidden source code that the students can link their applications to. The database adapter provides public APIs that wrap all the functionality needed to set up a database of jokes, open and close the database, insert jokes, read jokes, update jokes, and delete jokes. Students first learn how to use an SQLite database for persisting data by updating their "Joke List" application to use these APIs. Students learn how to use the Android database in related classes like Cursors and CursorAdapters.

Once the students understand how to use the database adapter and its APIs they learn how to implement their own. Students are tasked with implementing their own version of the database adapter using the exact same APIs. They are provided with a skeleton version of the database adapter source code they have been using, but with the implementation removed. This leaves only the method stubs and member variable declarations. The students then fill in the implementation details themselves and link against their own version of the database adapter. By doing this, the students learn how to use a database to persist data, as well as how to implement one themselves.

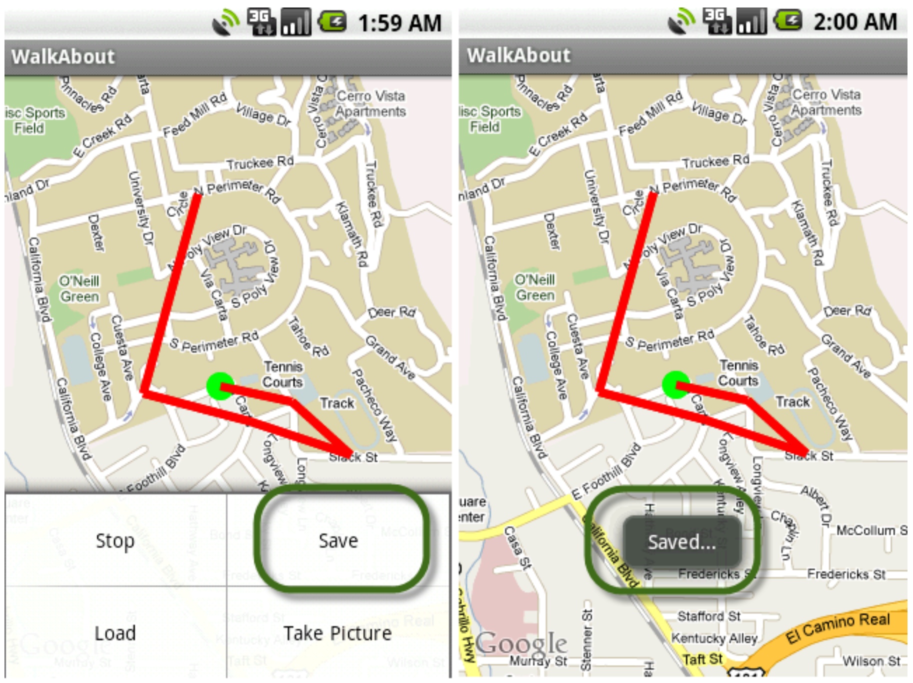


Figure 7: "WalkAbout" Application Screen Shots

5.6 Lab 5

For this lab, students develop a new GPS recording application called WalkAbout. The purpose of the application is to allow users to record their GPS location information as they travel. While the application records the user's GPS data, it displays it back to the user in the form of a path drawn on top of a Google Map. While recording data, the user can launch a camera Activity that will capture and store pictures on an SD-card. When finished recording, the application gives the user the option of storing the current GPS data as a private application file to be loaded and displayed at a later time.

5.6.1 Using Google Maps & Overlays

The Android platform comes with a Google Maps library for integrating Google Maps into applications. This library allows developers to include and manipulate Google Maps into their applications. Students learn how to display maps by using the MapView and MapActivity classes in the "Walk About" application. These classes encapsulate all the viewing and gesture logic necessary for handling panning, zooming, and touching objects on a map.

The Google Maps library also provides an Overlay class which allows developers to overlay additional content and interactions on a map. The MyLocationOverlay class provided by the Google Maps library is one such example; it can draw a beacon and a compass on the map to display the device's current position and heading respectively. Students learn how to use overlays by first adding a MyLocationOverlay to the "Walk About" application and then implement their own overlay to display a path. The path overlay the students create draws a red path on a map for a given list of latitude and longitude points.

5.6.2 Using GPS

The Android platform provides services and APIs for determining the current location of the device. Using this information, developers can enrich their applications by providing them with location awareness. Students learn how to interact with location-based services by adding functionality to enable and monitor the GPS LocationProvider in the "Walk About" application. The students then record changes in location as the user's path. This path information is then used draw the user's path onto the "Walk About" MapActivity's MapView object.

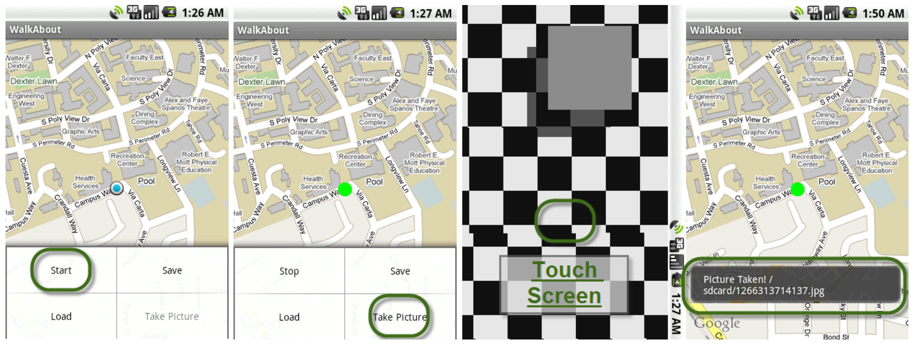


Figure 8: "WalkAbout" Camera Activity

5.6.3 Using a Camera

The Android platform allows applications direct access to the camera hardware. Integrating a camera into an application allows developers to create novel applications. Students learn how to make use of the camera related APIs by implementing a camera Activity that allows users to take pictures from the "Walk About" application. The camera activity was borrowed from the Google API Demos application [8] and displays a full screen preview of what the camera is looking at. The students extend this Activity class by capturing a picture when the user touches the screen and saving it to the SD-card. Screen shots of this use case are shown in figure 8.

5.6.4 Working With Files

The Android platform makes use of the java.io library for file manipulation, allowing developers to create, write, read and delete local files. File manipulation provides developers with an alternative to database storage for their data. Android allows files to be stored in two locations [14]:

- **Internal Storage** These files become private to the application that created them by default. These files reside in the internal memory of the device and are removed when the application is uninstalled. Developers have the option to override this privacy mechanism, allowing them to be shared with other applications.

- **External Storage** These files are world-readable and may be edited by the user. External can come in the form of removable media, such as an SD-card, or additional internal memory.

Students learn how to create and edit both internal storage and external storage files. They learn how to use internal storage by adding functionality to the "Walk About" application to save and load a users path as an internal file. Then they learn how to use external storage by saving a the picture taken from the camera Activity as an external file on an SD-Card.

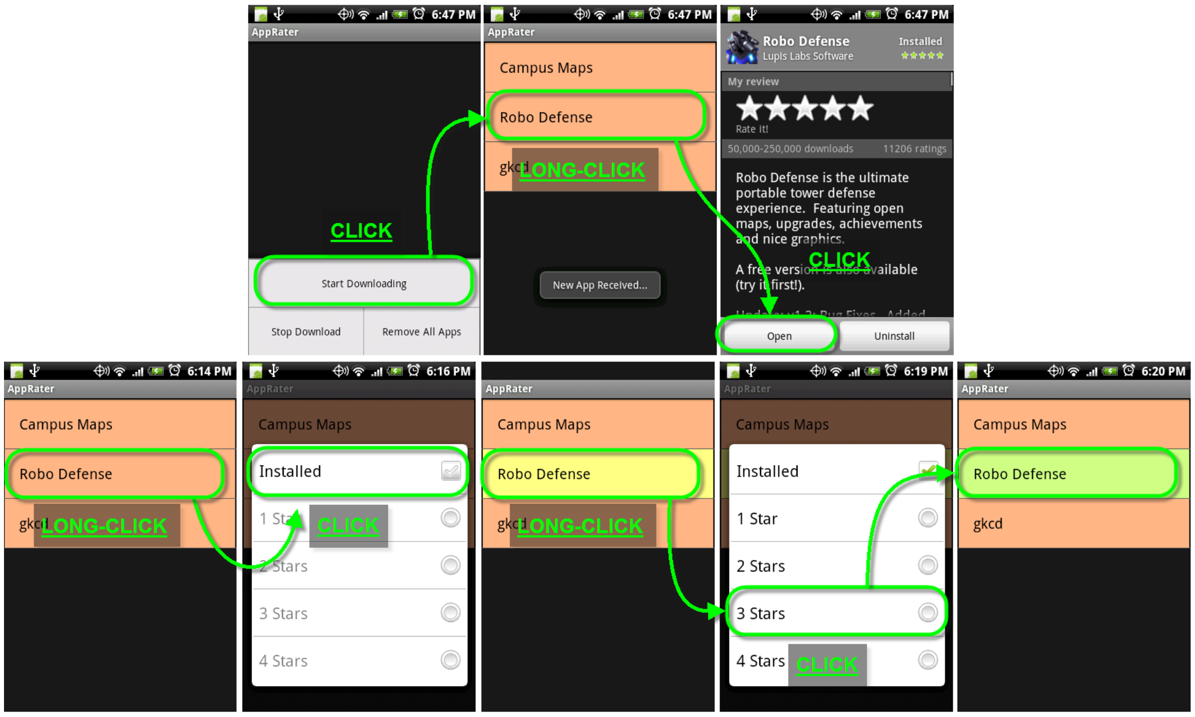


Figure 9: "App Rater" Application Screen Shots

5.7 Lab 6

For this lab, students develop a new application named AppRater that suggests other applications for users to download and try. The purpose of the application is to share fun and interesting applications with other users. The users can then rate the applications. It's a simple application whose base implementation is extended and used to run an application development contest put on by this course at the end of each quarter.

The application makes use of additional Android application platform components such as ContentProviders, Services, and BroadastReceivers. A ContentProvider serves as the interface for data persistence. A Service is used to download new applications in a background thread. A BroadastReceiver is used to pass messages between the Service and Activity components.

5.7.1 Using ContentProviders

The Android platform has a component called ContentProvider which serves as a means of providing content to other components. The ContentProvider provides an interface for manipulating its content while abstracting away the manner in which the content is stored. Interacting with a ContentProvider is very similar to interacting with a database adapter. One can execute queries which return Cursors, delete data, update data, and insert data. The “App Rater” application uses a ContentProvider to maintain its list of applications to rate.

Using ContentProviders offers two main benefits that database adapters do not. Take for example the “Joke List” application from Lab 4 that persisted data in a joke database through the use of a database adapter. This joke database is only visible to the JokeList application. However, if a ContentProvider is used to wrap the database, other applications could potentially interact with it as well. Additionally, ContentProviders abstract away the underlying manner in which the data they serve is persisted. This allows for different persistence implementations to be swapped out while maintaining the same interface.

Students are first provided with a functioning ContentProvider that serves their “App Rater” Activity with suggested applications to display and rate. This comes in the form of a library jar file that the students can link their applications to with the source code hidden. The Students first learn how to use a ContentProvider to retrieve and persist data by implementing their “App Rater” application to use this ContentProvider.

Once the students understand how to use a ContentProvider, they learn how to implement their own. Students are tasked with implementing their own version of the ContentProvider. They are provided with a skeleton version of the ContentProvider source code they have been using, but with the implementation removed. This leaves only the method stubs and member variable declarations. The students then fill in the implementation details themselves and link against their own version of the ContentProvider. By doing this the students learn how to use a ContentProvider to persist data, as well as how to implement one themselves.

5.7.2 Using Services

The Android platform has a component called Services, which run in the background and do not have UIs. Any time there is code which needs to be run regularly but does not need a user interface it can probably be implemented as a Service. Services can be started from an application's currently visible Activities or can be awoken by System Notifications even when an application's Activities are all closed. The "App Rater" application uses a Service to download new applications to rate and add them through its ContentProvider.

Students are first provided with a functioning Service that refreshes the "App Rater" application with new applications to rate. This comes in the form of a library jar file that the students can link their applications to with the source code hidden. The students first learn how to start and stop a Service to perform work for them by implementing their "App Rater" application to use this Service.

Once the students understand how to use a Service they learn how to implement their own. Students are tasked with implementing their own version of the Service. They are provided with a skeleton version of the Service source code they have been using, but with the implementation removed. This leaves only the method stubs and member variable declarations. The students then fill in the implementation details themselves and link against their own version of the Service. By doing this, the students learn how to use a Service to persist data, as well as how to implement one themselves.

5.7.3 Broadcasting Intents

Intents are used by the Android application platform as a system-level message passing system. They can be used to start application components, or they can be used to send messages between components. In order to listen for a message, one has to implement a BroadcastReceiver. The “App Rater” application’s download Service broadcasts an Intent after it successfully downloads and saves a new app through the ContentProvider. When the “App Rater” Activity is running it uses a BroadcastReceiver to listen for the Intent broadcasted by the Service, so that it can notify the user that a new app was downloaded.

Students learn how to listen for Intent broadcasts as well as how to broadcast Intents themselves. They begin by implementing a BroadcastReceiver to listen for the Intents broadcasted by the “App Rater” download Service provided to them. Once implemented, they make their “App Rater” Activity use this BroadcastReceiver. Then while implementing their own download Service, they make it broadcast the correct Intent.

5.8 Automated Grading

The solution for automated grading makes heavy use of the Android testing framework to minimize the amount of human involvement. The Android testing framework is an extension to the Junit testing framework. It provides additional testing classes that set up an interface with Android application framework components so they can be tested. Using this framework we programmatically determine whether every submitted application has been correctly implemented according to the specification outlined in its lab. This is done by implementing three separate components.

The first is composed of a set of test suites that each define the acceptance criteria for a particular lab, known as the lab tests. The second component is a standalone application, known as the Grader, that runs a single test suite for a lab against each student application submitted for that lab. The Grader then outputs the test results from each student application while grading. The third component, which comes in the form of a set of grading rubrics, is then used to manually transform the test results into a grade.

5.8.1 Lab Tests

For each lab, a suite of tests is created. These tests make up the acceptance criteria used to determine whether a submitted application meets the specification set forth by its laboratory assignment. Test suites are created for labs after their application solutions and application skeleton projects are created. This is done to ensure that names and signatures for classes, methods, member variables, and other resources have been solidified. The test suites are comprised of both unit and acceptance tests.

The acceptance tests serve as course grained tests to ensure a piece of functionality is implemented correctly from the user interaction level. These tests interact with and test the UI. They cause buttons to be clicked and carry out an intended use case or user story. These take the place of application demonstrations that would be otherwise performed manually by a human grader.

Unit tests serve as fine grained tests to ensure that a piece of functionality is implemented as it is intended. These tests make sure the methods and classes used for a piece of functionality are doing what the lab specification said they should be doing. Most importantly, they ensure that the applications are using the features of the Android application framework they are supposed to use and that they are being used correctly. This helps catch applications that are faking functionality which not be caught by the acceptance tests alone. The unit tests take the place of code reading that would be otherwise performed manually by a human grader.

During the creation of the lab tests, unforeseen problems arise which generate creative solutions. These problems include testing applications when tests are inherently implementation dependent, gaining access to private and protected member variables and methods for testing, ensuring that tests work for partial implementations, and keeping track of lab specification requirements and the tests that check for them. The sections that follow summarize these problems and their solutions.

Lab Implementation Dependencies

In order to test a method for a particular class, one at the very least needs to know the name of the class and the signature of the method. The same follows for member variables as well. It follows then that testing in general is an implementation dependent activity. In fact, without the source code tests it will not even compile. This begs the question: "How do you write tests for student implementations that have yet to be implemented?" In short, the answer is: By providing the students with application project skeletons and requiring the use of predetermined names and signatures of classes, methods, member variables, and other resources.

As outlined earlier in the the Lab Structure Implementation section, project skeletons are created and distributed to the students for each lab. The skeleton projects are essentially copies of the application solutions with the implementation details removed. The project skeletons define the classes to be used, the signatures of their methods and member variables, as well as the names of resources and the IDs. The lab specifications each contain explicit instructions that names and signatures of classes, methods, variables, and resources are not to be changed and should be used as specified.

By forcing all application submissions to make use of the skeleton projects based on the solution applications, we are able to resolve most implementation dependencies. Tests can then be written for the solution application with the reasonable expectations that they should work for the student application submissions as well. However, the assumption that students will follow the requirement not to change names or signatures and will use specific resource IDs does not mean that it is the reality. Tests are therefore written to ensure that these requirements are in fact honored.

Java Friend Functionality

While writing tests for the lab it is often the case that a private or protected member variable or method will need to be referenced. For whatever reason, this method or variable is necessary to either cause some sort of action or check that an action occurs correctly. In these situations, we are left with a few options.

The first options is to make the needed member variables and methods publicly accessible. This can be done by adding unnecessary accessor and mutator methods for the needed member variables or by making the needed methods public. This option seems to be a poor one in that it is in contrast with the spirit of encapsulation. Had those variables and methods needed to be publicly accessible for the sake of the application, they would have been. However, the architecture for the application deems that this is not necessary. Making variables and methods publicly accessible solely for the sake of being able to test the application seems like a poor compromise.

The second option involves multiple inheritance through extension. By having the test classes extend the application class under test and making the desired member variables and methods protected we can grant the tests access to them. While making variables and methods protected solely for the sake of testing is not optimal, it is still better than making them public. The only problem with this solution is that all test classes have to extend an Android testing framework class and Java only provides support for multiple inheritance through interfaces and not through extension. Thus, the test classes cannot extend another class. A workaround for this would involve creating a separate test class that extends the class under test and provides public wrapper methods for accessing the desired methods and variables. However, in addition to writing the tests and the application, this requires writing more unnecessary code just to provide access to needed variables and methods. Thus, we search again for another simpler solution.

The third and most elegant of options involves the friend class feature of the C++ language. With this feature, a class can be declared as a friend of another class thereby granting the friend class access to all private and protected methods and member variables. This option will not break encapsulation, but rather create an exception for our classes that are performing testing. Additionally, this option will require adding only a single line of code to each class we would like to test. The only problem with this option is that it is not a feature of the Java language.

The last option, and the one that we end up taking, makes use of a feature in Java called reflection to gain access to private and protected variables and methods. Using Java's reflection mechanism, one can query about and retrieve references to an objects public, private, and protected member data, methods, and constructors. For testing purposes, we rely heavily on this feature to retrieve otherwise hidden methods and variables.

We end up creating a static utility class named FriendClass that provides our test classes with two methods. The first is for retrieving references to hidden member variables from an object called **retrieveHiddenMember**. This method takes, as arguments, the name and type of the hidden member variable, as well as the object from which to retrieve it. The method returns a reference to the desired member variable allowing the test class to read or write to it.

The second method used for invoking an object's hidden methods is called **invokeHiddenMethods**. The methods takes, as arguments, the name, return type, and list of arguments for the hidden method, as well as the object on which to invoke the method. The method returns whatever the invoked method returns or null if the method is void. The test can then read or write to the value returned.

Both methods throw various exceptions if the variable or method requested does not exist. The tests explicitly check for these exceptions and rely on them to ensure that the names and types of member variables have not been changed by the students. An example of both of these methods can be seen in **Listing 1**. In this manner, we are able to mimic the benefits of the friend class feature of C++.

```
// Class under test
Class A {
   private Variable mVar;
   private String method(String string) {
     // do something to string
     return string;
   }
}

// Testing class
Class TestA {
 public void test {
   A a = new A();
   Variable var = null;
   String str = null;
   try {
     var = FriendClass.retrieveHiddenMember("mVar",var,a);
   } catch (FriendClassExceptoin exc) {
     fail("Could not retrieve mVar from A:" + exc.getMessage());
   }
   assertEquals("failure message", 2, mVar.getValue());
   ...
   try {
     ret = FriendClass.invokeHiddenMethod("method",a,str," t ");
   } catch (FriendClassExceptoin exc) {
     fail("Could not invoke method on A:" + exc.getMessage());
   }
   assertEquals("failure message", "new string", str);
   ...
 }
}
```

Listing 1: FreindClass Sample Code

Forward Compatible Tests

The labs have a tendency to guide students to grow their implementations as opposed to implementing the whole thing all at once. For example, in the beginning sections of a lab it is not uncommon to have students implement Functionality X in a simplistic or incomplete fashion. The lab might then have the students go back and reimplement Functionality X more completely or by using more advanced features. This is done for any number of reasons, ranging from showing the students multiple ways to do something to getting their application into an executable state so they can test it. The problem with this comes when you need to write a test for Functionality X.

Writing tests for things like Functionality X, that might be in one of more possible implemented states, involves considering the ultimate motive for the tests, which is grading. To get full credit for Functionality X the student needs to have the functionality present and have implemented it in the manner the specification expects. While there are multiple possible implementations, the fact remains that they are a series of intermediate implementations that end up in a final implementation state that is expected at the end of the lab. The strategy we chose is to give credit for both achieving the functionality via an accepted implementation and having the implementation expected at the end of the lab. We forgo giving credit to each intermediate implementation as the submitted applications can only have one implementation and the final implementation is generally the more important one.

With this in mind we are able to group tests into two separate test methods. The test first method incorporates acceptance tests to perform implementation independent checks for functionality and unit tests to ensure implementation is performed in one of the specified ways. The second test method includes only the unit tests designed to check for the final implementation, or the implementation expected at the end of the lab. By placing the code to check for the final implementation in a separate method and calling it from the two test methods, no code needs to be duplicated. By having these two separate methods we can credit the students appropriately and ensure that the same tests will work for an implementation in any stage of the lab.

There is a flaw in this strategy in that it cannot give credit or check for intermediate and alternative implementations. In some situations the tests should check for this, since the labs have students perform alternative implementations to show them how to do things in different ways using different application platform components. For these types of situations, this problem can be remedied by using the Strategy pattern to decompose intermediate and alternative implementations into their own classes or methods. Then tests can be written and credit given for each implementation instead of just the final implementation.

Requirements & Testing Traceability

Most labs have a reasonably large set of requirements defined in their specifications. This in turn generates a reasonably large set of tests to check that those requirements had been met. One problem that we run into is being able to trace from individual tests to requirements and back. For this we rely heavily on mimicking the natural structure of the labs in the tests themselves. Each lab specification is decomposed into sections, subsections, and subsubsections. In general, the lowest level defines some atomic piece of cohesive functionality. As you move up levels, functionality is combined together to define some larger piece of functionality.

Each lab test suite is composed of independent Android testing framework classes. Each test class is intended to test a single cohesive piece of functionality defined at either a section, subsection, or subsubsection level. The test classes are in turn composed of a set of test methods. Each test method is designed to test one or more requirements outlined in the section of the lab specification targeted by its test class. By structuring the lab specifications and lab test suites in this manner it is much easier trace requirements to tests and vice versa.

As an added measure, both naming conventions and comments are used to make it clear which requirement a particular test is testing for. In general, test classes are named after the lab section they target and test methods are named after the requirements they test. Both test classes and methods include comments identifying specific section numbers and requirement descriptions that are being tested. Lastly, the grading rubric for each lab provides a course grained traceability matrix that maps each lab specification section, subsection, and subsubsection to the test classes that test them. The grading rubric will be discussed in more depth after the the next section.

5.8.2 The Grader

The Grader is implemented as a standalone Java application designed to execute a test suite on a set of submitted applications in a batch processing fashion. The Grader is executed via the command line and is passed a number of required and optional parameters. At the least, the Grader needs to know the location of the root directory for the test suite project, the directory containing all of the application project submissions, and the full qualified name of the root package for the application. Optionally, the grader can be provided with parameters to adjust granularity of logging output.

When the Grader is started, it first rebuilds the test suite to ensure that it is signed with the local debug certificate. It then rebuilds each application submission using the local debug certificate and executes the lab test suite on it as well any test suite included by the student in their application submission. The results of the build and tests for each application submission are output to a combined test results file. Additionally, the verbose output of the build and tests run for each application submission are output to an individual test results file. Lastly, every test error and failure generated by the build or test suite execution on any application submission are output to a test errors and failures file. Each output file is placed into a directory created by the Grader so as to make the files easy to retrieve.

These features allow the grader to be easily started or scripted to run automatically. Each output file serves a special purpose to help make the automated grading more accurate, traceable, visible, transparent, and insightful. The purpose and benefits of each file are explained in more depth in the subsections that follow. When the Grader is finished, the directory containing all the output files can then be retrieved and used to generate grades and refine the test suites.

Combined Test Results

For each application the results of its build and test suite execution are added to a combined test results file. The combined test results file is a comma separated values (CSV) file containing the results from one application per line. The data recorded for each application includes:

- **Student Name:** The name pulled from the directory containing the application submission.
- **Build Result Code:** The code identifying successful build or the cause of failure which can be looked up in the grader source for code for an explanation.
- **Number of Student Tests:** The total number of student supplied tests run.

- **Number of Student Failures:** The total number of student supplied tests failed.
- **Number of Student Errors:** The total number of student supplied tests that had errors.
- **Student Test Result Code:** The code identifying successful student supplied test suite execution or the cause of failure, which can be looked up in the grader source for code for an explanation.
- **Number of Lab Tests:** The total number of lab tests run.
- **Number of Lab Failures:** The total number of lab tests failed.
- **Number of Lab Errors:** The total number of lab tests that had errors.
- **Lab Test Result Code:** The code identifying successful lab test suite execution or the cause of failure, which can be looked up in the grader source for code for an explanation.

This file allows the person overseeing the grading to quickly identify any application submissions that produced build or test execution errors which might need to be fixed and rerun. Additionally, the file allows the same person to view how the results of each application submission compare to each other. If every submission is failing or passing most of the tests, one might be able to deduce that there is a problem with the tests. Alternatively, there might be a problem with the lab specification. This file helps improve course grained analysis of the results of the grader by making key metrics much more visible. This allows for problems to be quickly identified and remedied, which improves the accuracy of the grading process and quality of the laboratory specifications.

Individual Test Results

For each application, the verbose output of its build and test suite execution are saved to an individual results file. Each file is titled using the name pulled from the directory containing the application submission. This file provides the person overseeing the grading with a way to investigate any build problems associated with an application submission. Additionally, it provides the same person the ability to see which tests failed or produced errors. This last bit of information is combined with the grading rubrics to generate a grade for an application submission.

The individual results file is also intended to be given back to the student who submitted the application for which the file was generated. This allows a students to see the detailed results of their build as well as the detailed results of the tests run on their application. With this information, a student can then attempt to correct any build problems, test failures, or test errors and resubmit their application if permitted. This file makes the grading process more transparent for students. It also makes fine grained analysis of the results from a single application submission much more visible and insightful by enabling the overseer the ability to review the grading and building process.

Test Errors & Failures

For every run of the Grader, a single file is created and filled with every test error and failure that is generated during the grading of all application submissions. This test errors and failures file is a CSV file containing the data about a single error or failure on each line. The data recorded for each error or failure includes:

- **Error/Failure and Method:** This dentifies whether this is a test error or failure and the test method that caused it.

- **Error/Failure Type and Message:** This identifies the type of assertion or exception that caused the failure or error and the message associated with it.
- **Location:** This is the stack trace line identifying the source of the failure or error. It includes the fully qualified class and method name as well as file and line number.
- **Student Name:** This is the name pulled from the directory containing the application submission.

The test errors and failures file is intended to provide the person overseeing the grading process with information on which tests are causing the most failures and errors and why. By sorting the failures and errors by the appropriate data points, the overseer can see which test methods are causing the most problems and investigate whether those tests need to be refactored. This file helps improve course grained analysis of the results of the grader by making common failures and errors more visible. This allows for problems to be quickly identified and remedied, which improves the accuracy of the grading process and quality of the laboratory specifications.

Project Rebuilding

In order to execute a test suite on an application, both the test suite and the application need to be signed by the same certificate. One solution to this problem is to require all the students to use the same certificate. However, this option proved to be too burdensome to manage as it required the dispersal and management of a single certificate, as well as having the students update their development environment to make use of the certificate. Instead, we chose the more robust option of having the Grader rebuild each submitted application and test suite using the same key.

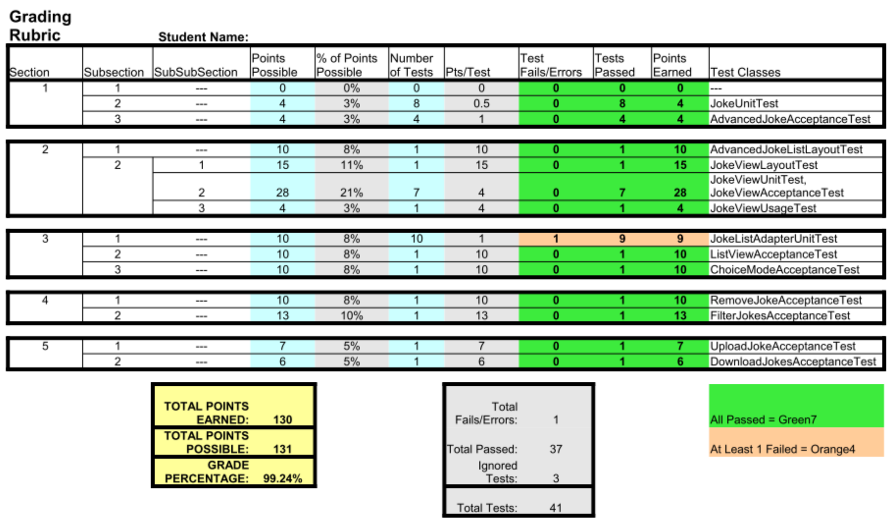
The Grader makes use of the Android Toolkit distributed with the SDK, Apache Ant, and the local debug.keystore certificate to rebuild the applications. The Grader uses the “android update package” command to generate the build file for each project and then executes its “ant debug” target. The build output logs and results of each build are incorporated into both the combined test results file and the individual test results files. This allows both the students and whoever is overseeing the grading process to see if there are build errors for any particular submission.

Grader Dependencies

The Grader makes use of both the Android Toolkit provided with the SDK and Apache Ant. Thus the Grader requires that the directory containing these dependencies be present in the path environment variable. The Grader requires that all application submissions come in the form of a complete Android application project. Additionally, the root directory of each application submission must bear the submitting student’s name and all application submissions must reside in the same directory.

5.8.3 Grading Rubrics

Grading rubrics for each lab are developed to translate the results of a lab test suite into a grade. The rubrics exist as Excel sheets, which are copied and filled out for each student. An example of the rubric used for the third lab can be seen in figure 10. Rubrics are created by first listing each of the lowest levels of sectioning, including sections, subsection, and subsubsections in their own row. Then each of these section rows is assigned a number of points. The relative


Grading Rubric **Student Name:**

Section	Subsection	SubSubSection	Points Possible	% of Points Possible	Number of Tests	Pts/Test	Test Fails/Errors	Tests Passed	Points Earned	Test Classes
1	1	---	0	0%	0	0	0	0	0	---
	2	---	4	3%	8	0.5	0	8	4	JokeUnitTest
	3	---	4	3%	4	1	0	4	4	AdvancedJokeAcceptanceTest
2	1	---	10	8%	1	10	0	1	10	AdvancedJokeListLayoutTest
	2	1	15	11%	1	15	0	1	15	JokeViewLayoutTest
		2	28	21%	7	4	0	7	28	JokeViewUnitTest, JokeViewAcceptanceTest
		3	4	3%	1	4	0	1	4	JokeViewUsageTest
3	1	---	10	8%	10	1	1	9	9	JokeListAdapterUnitTest
	2	---	10	8%	1	10	0	1	10	ListViewAcceptanceTest
	3	---	10	8%	1	10	0	1	10	ChoiceModeAcceptanceTest
4	1	---	10	8%	1	10	0	1	10	RemoveJokeAcceptanceTest
	2	---	13	10%	1	13	0	1	13	FilterJokesAcceptanceTest
5	1	---	7	5%	1	7	0	1	7	UploadJokeAcceptanceTest
	2	---	6	5%	1	6	0	1	6	DownloadJokesAcceptanceTest

TOTAL POINTS EARNED:	130
TOTAL POINTS POSSIBLE:	131
GRADE PERCENTAGE:	99.24%

Total Fails/Errors:	1
Total Passed:	37
Ignored Tests:	3
Total Tests:	41

🟩	All Passed = Green7
🟧	At Least 1 Failed = Orange4

Figure 10: Example Grading Rubric

percentage of total points for each section row is displayed as well, to aid in this task. Then the total number of tests are counted and entered for each section row. Additionally, the number of points per test and a list of the test classes for a section row are entered.

Translation from testing results to grades is performed manually by a person. The person makes a copy of the rubric for the student they are grading and opens their individual test results file as well. The person then counts the number of tests that failed or produced errors from the list of test classes for a particular section row. This number is then entered into that section row's "Test Failures/Errors" column. The remaining "Tests Passed" and "Points Earned" columns are calculated automatically by the spreadsheet. If any tests failed or caused errors for a section row its color is changed from green to orange. This process is repeated for each of the section rows listed in the rubric.

While the test results are being entered, the spreadsheet automatically calculates the total points earned, total points possible, and grade percentage.

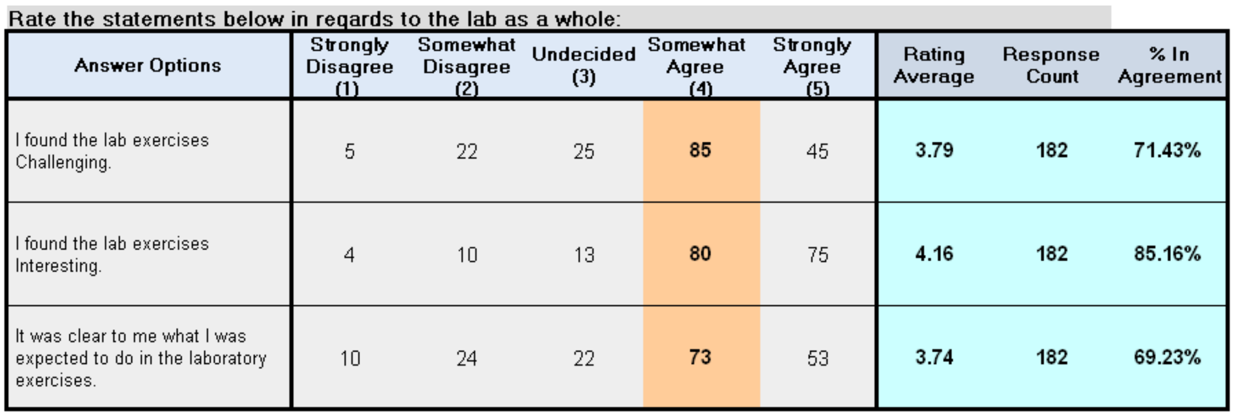
Additionally, the spreadsheet also sums the total number tests, tests passed, and test failures or errors to help ensure that the test results are entered properly. In some cases, there exist tests in the test suites that are not being used for the grading process. A count of the ignored tests are entered into the lab rubric before it is copied and used for grade translation.

After grades have been completed for each student, copies of the rubrics can be distributed to the students along with copies of their individual test results file. All test failures and errors are highlighted in orange and list the test classes which generated them. Students can then investigate the cause of these failures using their individual test results file. In this manner the rubrics provide students with a clear understanding of how their grade is calculated as well as additional transparency to the grading process.

6 Evaluation

The evaluation of this project is performed in three parts. The first part evaluates the labs as a whole by leveraging an aggregate analysis of the student responses collected from surveys, student performance on the labs, and student performance in other aspects of the related Mobile Development course. The second part provides an in depth evaluation of each of the labs by using student responses from their corresponding surveys. The final part of this evaluation section analyzes the performance of the system for automated grading.

The data used to perform all three evaluations comes from two separate instances of the course. The first instance of the course was taught in the winter quarter of 2010, from January to March. This course was taught as a traditional, in-person class composed of 33 students. All student lab submissions for the winter quarter were graded less stringently via individual demonstrations. The second instance of the course was taught in the summer quarter of 2010, from June to August. This course was taught as an online class composed of 22 students where student instruction and work was performed remotely. The first two student lab submissions for the summer quarter were graded using the same method as the winter quarter. All subsequent lab submissions were graded using the method for automated grading described in previous sections. In both the winter and summer instances, the courses were offered to third and fourth year undergraduate students and graduate level masters students as an elective at California Polytechnic State University at San Luis Obispo. Students in both instances of the courses were expected to have considerable programming experience with Java and some experience with software engineering practices.



Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	5	22	25	**85**	45	**3.79**	**182**	**71.43%**
I found the lab exercises Interesting.	4	10	13	**80**	75	**4.16**	**182**	**85.16%**
It was clear to me what I was expected to do in the laboratory exercises.	10	24	22	**73**	53	**3.74**	**182**	**69.23%**

Table 1: Student Ratings of the Labs on Difficulty, Interest, and Clarity

6.1 Aggregate Lab Analysis

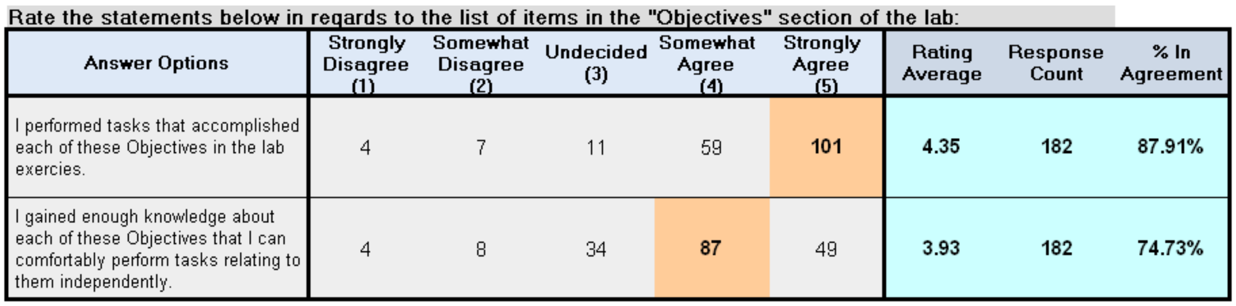
In the section that follows we analyze the labs as a whole. We aggregate the data from all of the individual labs in an attempt to evaluate their overall performance. This includes an aggregate analysis of the lab survey results, followed by an aggregate analysis of student performance on the labs.

6.1.1 Aggregate Survey Results

For this analysis we aggregate the lab survey results of all the labs from both the winter and summer courses. We then take that data and attempt to answer eight different questions about the quality of the labs.

Are the labs challenging?

On the whole, students thought the labs were reasonably challenging. As table 1 shows, most ratings are centered around "Somewhat Agree" with 71% of responses agreeing to some degree that the labs were challenging.


Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	4	7	11	59	**101**	**4.35**	**182**	**87.91%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	4	8	34	**87**	49	**3.93**	**182**	**74.73%**

Table 2: Student Ratings of the Labs on Ability to Teach Learning Objectives

Are the labs interesting?

In general the labs were able to capture the interest of the students. Table 1 shows 85% agree to some degree with the statement about finding the labs interesting.

Are the labs clearly written?

The students thought the labs were somewhat clearly written. Table 1 shows 69% agree to some degree with the statement about finding the labs clearly written and the average rating is closer to “Somewhat Agree” than “undecided.”

Do the labs teach their learning objectives?

For the most part, the students are in agreement with the statement that the lab had them engage in activities that are related to its stated learning objectives. As table 2 shows 87% of students agree with the statement, with majority of them strongly agreeing.

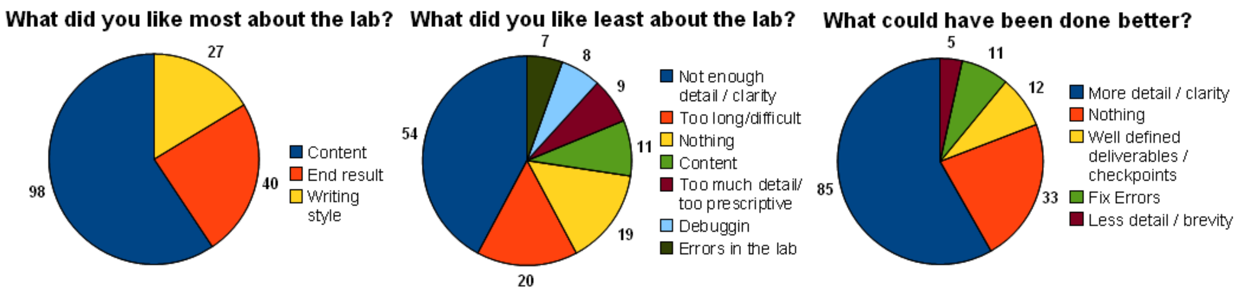


Figure 11: Student Opinions of the Labs

Do the labs prepare students to work independently?

The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not all that high. The majority of ratings in agreement with the statement are only somewhat in agreement, which can be see in table 2.

What did students like most about the labs?

The leftmost pie chart in figure 11 has the majority of comments identifying three things that students like most about the labs. The most repeated comments consist of students stating that they enjoyed the content the labs covered. The second most repeated comments make mention of the satisfaction that students receive from implementing the applications. Lastly, students commented frequently on the tutorial writing style of the instructions.

What did students like least about the labs?

The center pie chart in figure 11 has the students identifying a wide range of things they do not like about the labs, however, three of the answers are repeated the most. The most repeated comment is that the instructions are either not detailed or not clear enough. The second most repeated answer is that students feel the labs are too long. Lastly, some of the students feel that there wasn't anything wrong with some of the labs.

What did students think needs improvement?

The right most pie chart in figure 11 has the majority of students identifying three suggestions for improving the labs. The most repeated suggestion is to add more detail to or edit the current details of the instructions to improve their clarity. The second most suggested improvement is to change nothing. Lastly, the students feel that the labs needed a clearer list of deliverables, possibly coming in the form of checkpoints placed throughout the labs.

6.1.2 Student Performance

For this analysis we compare the grades students received for each of the labs from both the winter and summer courses. We then take that data and attempt to determine the difficulty of the labs and identify correlations between average performance on labs and performance in other areas of the course. It is important to note that when analyzing the difficulty of the labs we take into consideration the differences in grading procedures between the winter and summer courses. As described in prior sections, the labs for the winter students were graded less stringently while the majority of the labs for the summer students were graded much more rigorously using the method for automated grading. Thus, the data from the winter and summer quarters is separated for the following analysis.

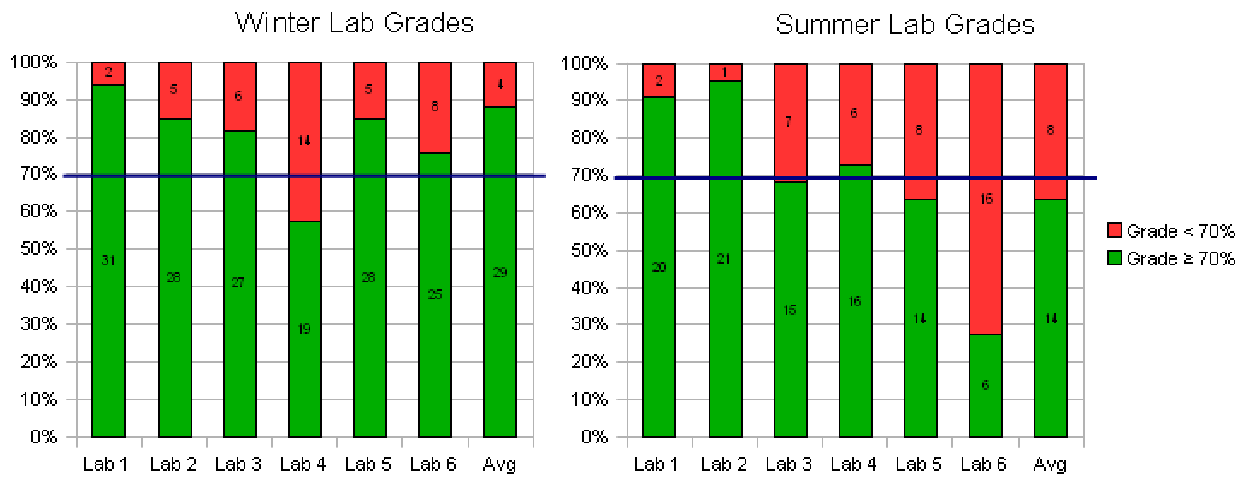


Figure 12: Distribution of Average Student Lab Grades

Were the labs adequately difficult?

We define a lab to be adequately difficult if 70% of the student submissions receive a grade of 70% or better. This metric tells us that the lab has enough difficulty such that a minority of students receive a grade worse than a C, while the majority of students are able to obtain a C or better grade. Labs in which more than 70% of the students get a grade of 70% or better indicate that the lab in question could be too simple, while the inverse statement indicates that the lab in question could be too difficult.

Figure 12 shows that 87% of students in the winter class and 63% of the students in the summer class received an average lab grade of 70% or better. It is our feeling that the grades from the winter class have a slight tendency to be overinflated due to the less stringent grading techniques, while the grades for the summer class better reflect actual performance. Taking into account the difference in grading techniques, we feel that the labs as a whole have a tendency to be slightly too difficult. We now attempt to explain some of the anomalies in the data:

- **Lab 1 grades:** The grades for the first lab for both the winter and summer classes are sufficiently higher than the others. This can be explained by the introductory nature of the lab and indicates that additional exercises could be added to the end of this lab.

- **Lab 1 & 2 summer grades:** The grades for the first two labs of the summer class are much higher than the rest of the labs. This is because these two labs were graded using the same less stringent method that was used for the winter class, which has a noted tendency to produce higher grades.

- **Lab 4 grades:** The difference in performance between winter and summer classes for the fourth lab have switched. The summer class outperformed the winter class. This is attributed in part to two separate problems. The first problem is the winter grading procedure's poor ability to identify partial credit because of its heavy reliance on demonstrations. This seems to be isolated to lab four because of its considerable length and the "all or nothing" nature of its implementation. Meaning that failing to complete a significant portion of the lab would lead to an inoperable application. Since the winter quarter grading procedure was unable to apply adequate partial credit this yielded lower scores.

  The second problem stems from a serious bug in the first release of the lab manual and project skeleton for the winter quarter. This bug caused a large number of students to get stuck. While a bug fix was released, the damage had already been done in the form of confusion, loss of productivity, and lowered confidence. Thus, we believe the summer quarter's higher grades in lab four can be accredited partly to the summer students not having to encounter the aforementioned bug.

- **Lab 6 grades:** The winter class drastically outperformed the summer class on the sixth lab. This is attributed to a flaw in the winter grading procedure and its poor ability to detect false positives, or faked results. It would be expected that the winter course would have grades more similar to the summer course due to the extreme difficulty students had with the lab.

- **Higher scores in winter:** The winter class had higher average scores than the summer class on five out of six of the labs. Aside from the differences in grading we feel that there were two other factors that may have played a role. The first contributing factor comes from the fact that the winter class was the first instance of the course. These students can be seen as early adopters of the course. It is the motivation for early adoption, which might include a higher than average interest in the topic, that may have contributed in part to higher scores. The second contributing factor comes from the nature of offering a summer course where roughly half of the participating students were engaged in extra curricular activities like internships and full or part time jobs. These activities may have caused some of the summer students to expend less effort on the labs.

Do lab grades correlate to final exam grades?

Although there is some slight correlation between final exam grades and average lab grades, we cannot rely on lab grades as a predictor of final exam scores. As figure 13 shows, the R^2 is too low. However, it is interesting to note that the more stringent grading process used to evaluate the labs in the summer class significantly improved the R^2 value of the correlation.

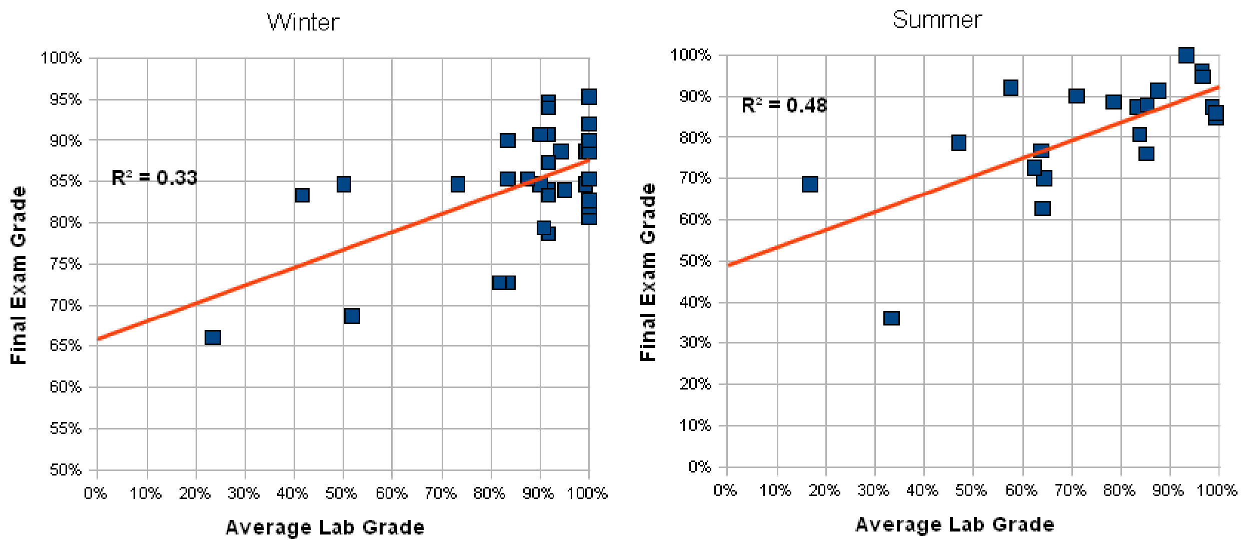


Figure 13: Correlation Between Student Final Exam and Lab Grades

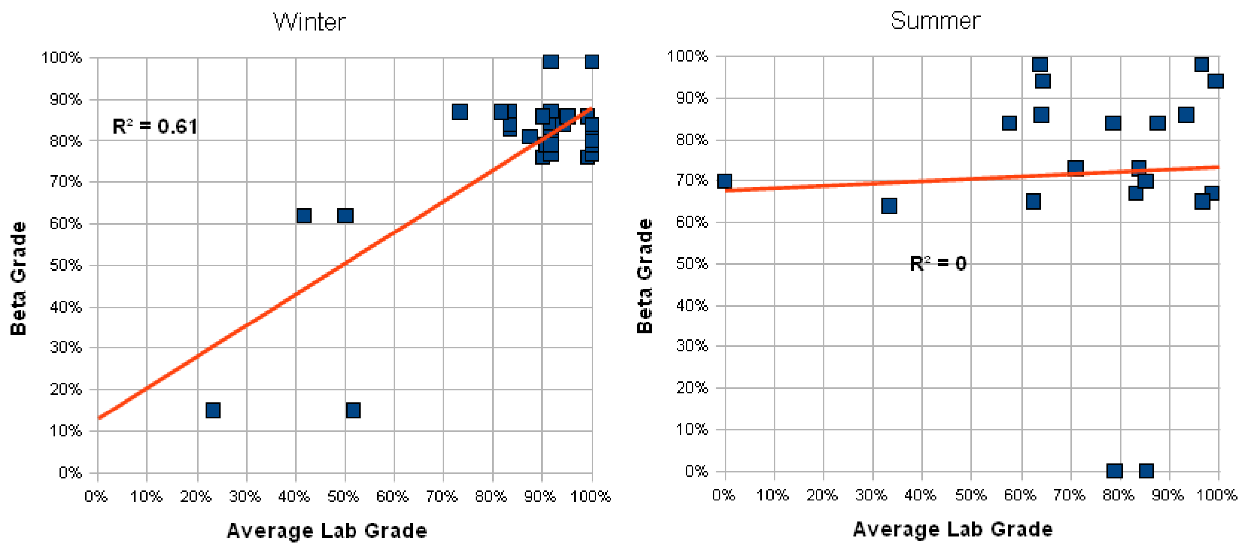
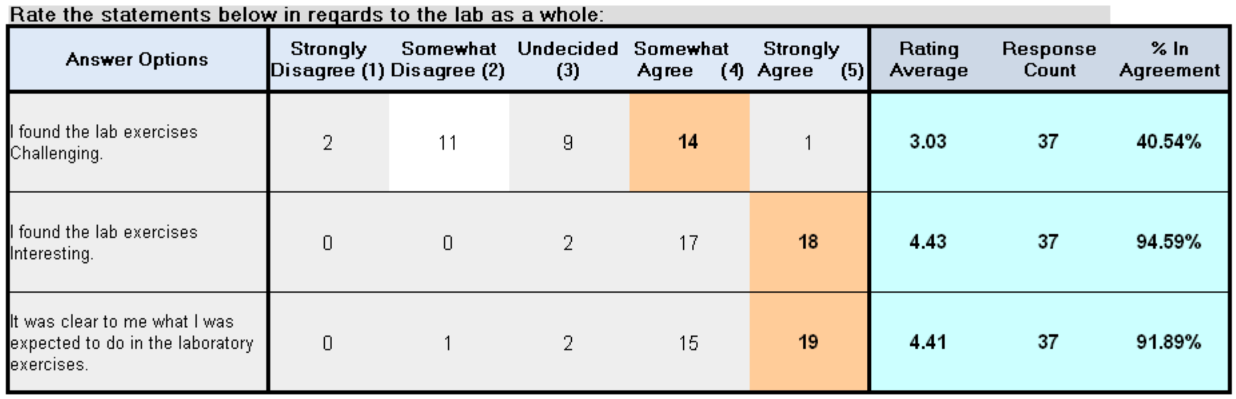


Figure 14: Correlation Between Student Final Project and Lab Grades

Do lab grades correlate to project grades?

It turns out that we cannot rely on lab grades as a predictor of the quality of the class projects. As figure 14 shows, the R^2 is too low. The drastic difference between the correlations for the winter and summer classes is most easily explained by observing the difference in the density of the data points between the two classes along both axes. The winter class has a much more dense distribution along both axes, which significantly contributes to its higher R^2 value.

The higher density along the x-axis, or average lab grade, is attributed to the differences in grading procedures. While the higher density long the y-axis, or project grade, we attribute to the difference in class types. The winter class was a standard "in person" class, while the summer class was an "online class". We believe that the "in person" nature of winter class fostered a more productive and creative environment as it guaranteed the students time to work together in close proximity to each other. It is these two contributing factors that caused the drastic difference in R^2 values.



Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	2	11	9	**14**	1	**3.03**	**37**	**40.54%**
I found the lab exercises Interesting.	0	0	2	17	**18**	**4.43**	**37**	**94.59%**
It was clear to me what I was expected to do in the laboratory exercises.	0	1	2	15	**19**	**4.41**	**37**	**91.89%**

Table 3: Student Ratings of Lab 1 on Difficulty, Interest, and Clarity

6.2 Individual Lab Evaluations

In the section that follows we analyze each of the labs individually. We review the survey results separately for each lab. From these results we to attempt to gain insight from the students responses about the quality of the labs.

6.2.1 Lab 1 Analysis

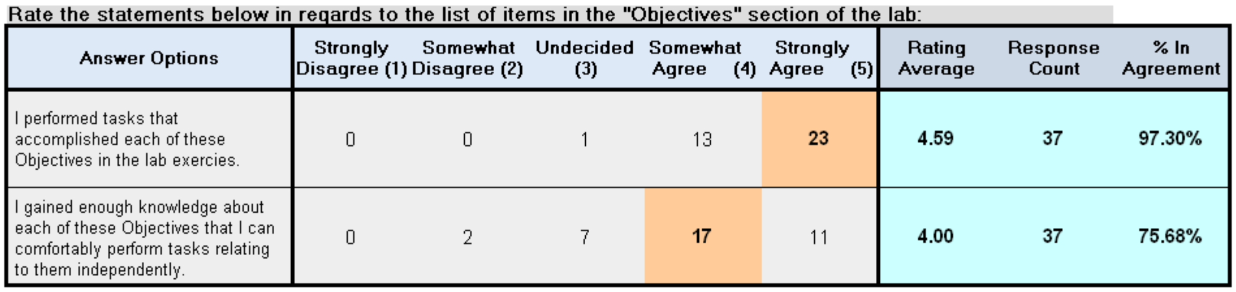
Survey Results

- **Is the lab challenging?**

  Overall, the students thought the lab was not challenging while at the same time not too easy. As table 3 shows, most ratings are centered around "Undecided" with only three students voting at the the extremes of "Strongly Agree" and "Strongly Disagree".

- **Is the lab interesting?**

  In general, the lab was able to capture the interest of the students. Table 3 shows 94% agree to some degree with the statement about finding the



Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	0	0	1	13	**23**	**4.59**	**37**	**97.30%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	0	2	7	**17**	11	**4.00**	**37**	**75.68%**

Table 4: Student Ratings of Lab 1 on Ability to Teach Learning Objectives

labs interesting, while not a single person out of 37 disagree with the statement.

- **Is the lab clearly written?**

  The students as a whole felt the lab made it clear to them what they were expected to do. Table 3 shows 91% agree to some degree with the statement about finding the labs clearly written, while only a single person out of 37 disagree with the statement.

- **Does the lab teach its learning objectives?**

  In general, the students are in agreement with the statement that the lab had them engage in activities that were related to its stated learning objectives. As table 4 shows, 97% of students agree with the statement, with majority of them strongly agreeing.

- **Does the lab prepare students to work independently?**

  The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not high. The majority of students only somewhat agree with this statement, which can be see in table 4.

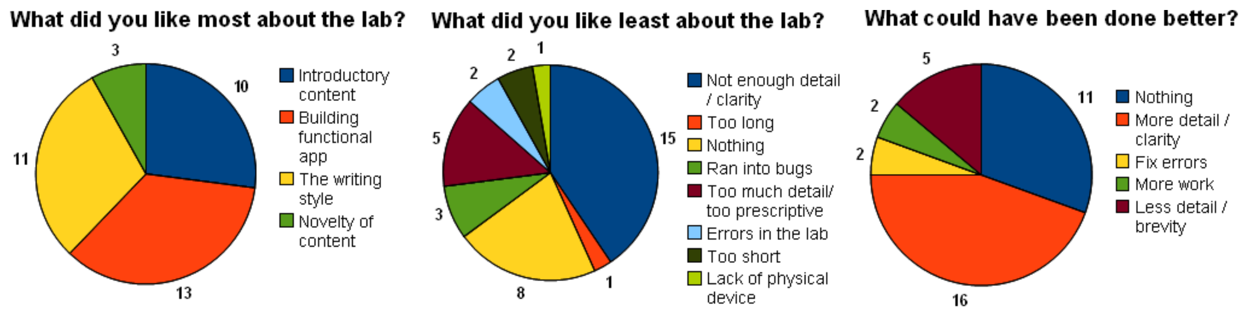


Figure 15: Student Opinions of Lab 1

- **What did students like most about the lab?**

  The leftmost pie chart in figure 15 has the majority of students identifying three things they like most about the first lab. The most liked feature of the lab is that it has students implement and deploy a functioning application that they can use. The second most liked feature is the step-by-step, tutorial, writing style of the instructions. Lastly, the students like the introductory nature of the content covered by the lab.

- **What did students like least about the lab?**

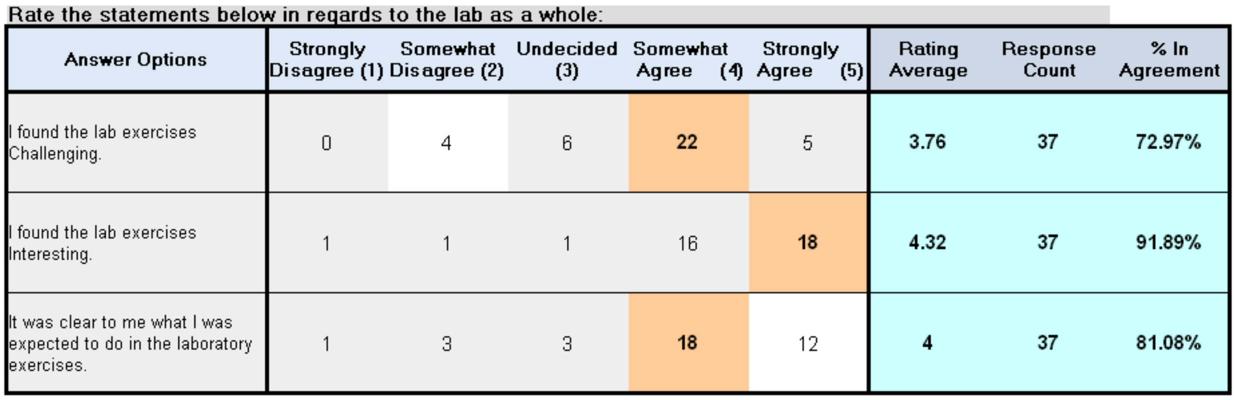
  The center pie chart in figure 15 has the students identifying a wide range of things they do not like about the first lab, however, three of the answers are repeated the most. The most disliked feature of the lab is that some of the instructions are either not detailed or not clear enough. The second most repeated answer is that students do not dislike anything about the lab. Lastly, some of the students find that the instructions provide too much detail and are too prescriptive.

- **What did students think needs improvement?**

  The rightmost pie chart in figure 15 has the majority of students identifying three suggestions for improving the first lab. The most repeated suggestion is to add more detail to or edit the current details of some instructions to improve their clarity. The second most suggested improvement is to change nothing. Lastly, the students feel that the instructions need to reduce the overall level of detail to improve on the brevity of instructions.

Conclusions

To summarize, the students' opinions of the lab are generally good. Students feel that it was both interesting and clear to them what they were expected to do. However, the lab is not very challenging and could therefore stand to be made slightly more difficult. The lab succeeds in engaging the students in activities related to its stated learning objectives. Moreover, the students in general feel relatively confident in their ability to perform similar tasks independently. The students enjoy that the lab had them implement a fully functional application. However, they feel that the lab needs to improve on the level of detail of its instructions by focusing on clarity and brevity.



Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	0	4	6	**22**	5	**3.76**	**37**	**72.97%**
I found the lab exercises Interesting.	1	1	1	16	**18**	**4.32**	**37**	**91.89%**
It was clear to me what I was expected to do in the laboratory exercises.	1	3	3	**18**	12	**4**	**37**	**81.08%**

Table 5: Student Ratings of Lab 2 on Difficulty, Interest, and Clarity

6.2.2 Lab 2 Analysis

Survey Results

- **Is the lab challenging?**

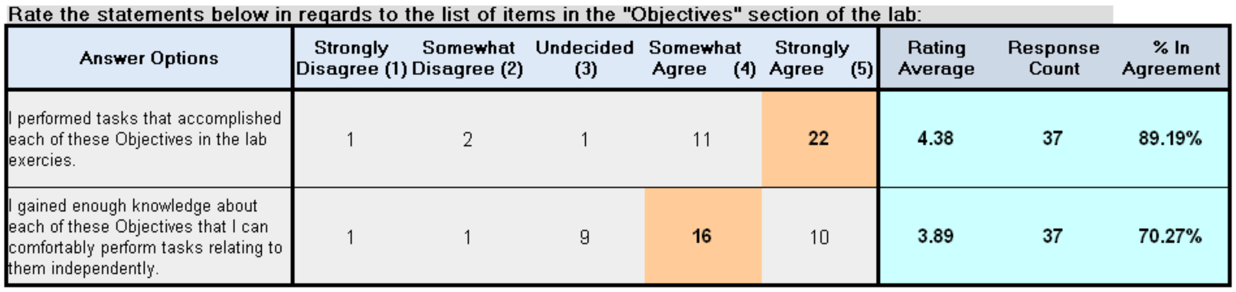
  Overall, the students thought the lab was somewhat challenging. As table 5 shows most ratings are centered around "Somewhat Agree" with only four out of 37 disagreeing with the statement that the lab was challenging.

- **Is the lab interesting?**

  In general, the lab was able to capture the interest of the students. Table 5 shows 91% agree to some degree with the statement about finding the labs interesting, while only two people out of 37 disagree with the statement.

- **Is the lab clearly written?**

  The students as a whole feel the lab made it reasonably clear to them what they were expected to do. Table 5 shows 81% agree to some degree with the statement about finding the labs clearly written, while four people out of 37 disagree with the statement.



Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	1	2	1	11	**22**	**4.38**	**37**	**89.19%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	1	1	9	**16**	10	**3.89**	**37**	**70.27%**

Table 6: Student Ratings of Lab 2 on Ability to Teach Learning Objectives

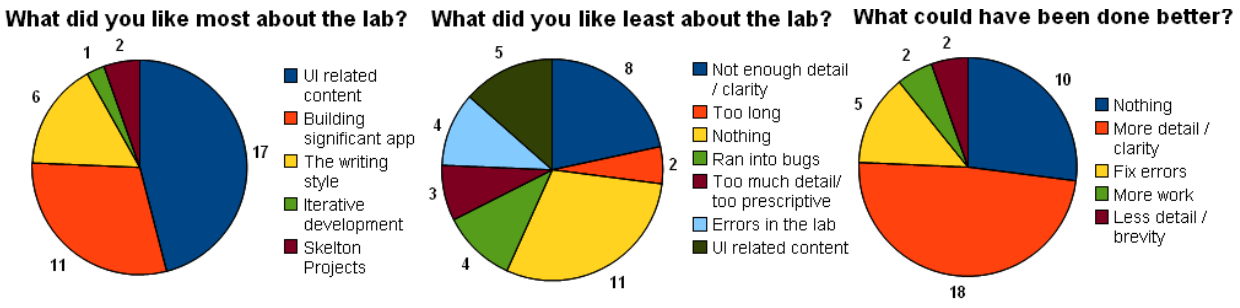


Figure 16: Student Opinions of Lab 2

- **Does the lab teach its learning objectives?**

  In general, the students are in agreement with the statement that the lab had them engage in activities that were related to it's stated learning objectives. As table 6 shows the average rating is roughly centered between somewhat and strongly agreeing, with majority of them strongly agreeing.

- **Does the lab prepare students to work independently?**

  The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not high. The majority of students only somewhat agree with this statement, which can be see in table 6.

- **What did students like most about the lab?**

  The leftmost pie chart in figure 16 has the majority of students identifying three things they like most about the lab. The most liked feature of the lab is that it dealt with UI related topics. The second most liked feature is that the lab had the students create an application of significant substance. Lastly, the students like the step-by-step, tutorial, writing style of the instructions.

- **What did students like least about the lab?**

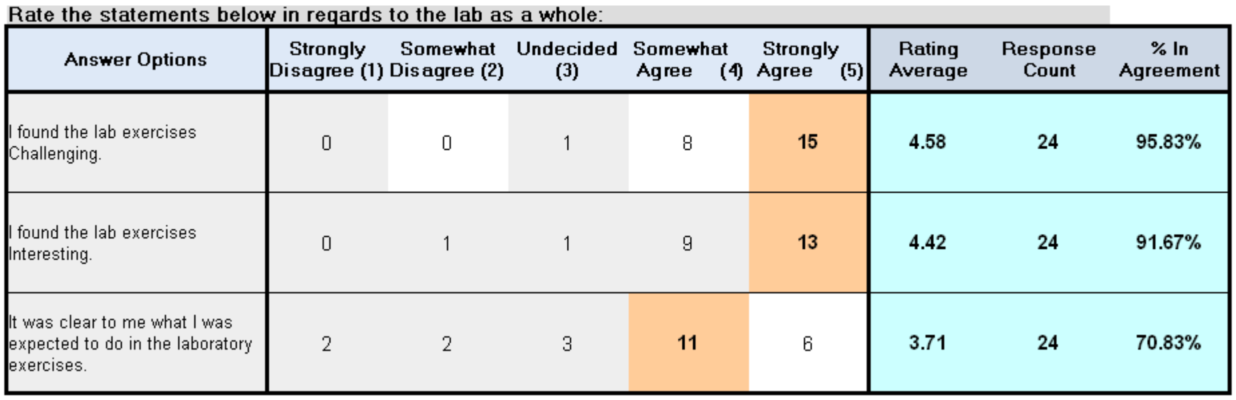
  The center pie chart in figure 16 has the students identifying a wide range of things they do not like about the lab, however, three of the answers are repeated the most. The majority of students state that there is not anything in the lab they did not like. The second most repeated answer is that some of the instructions are either not detailed or not clear enough. Lastly, some of the students highlight UI related activities they did not like.

- **What did students think needs improvement?**

  The right most pie chart in figure 16 has the majority of students identifying three suggestions for improving the lab. The most repeated suggestion is to add more detail to or edit the current details of some instructions to improve their clarity. The second most suggested improvement is to change nothing. Lastly, a number of students point out errors in the lab that need to be corrected.

Conclusions

To summarize, the students' opinions of the lab are generally good. Students feel that it was both interesting and relatively clear to them what they were expected to do while at the same time reasonably challenging. The lab succeeded in engaging the students in activities related to its stated learning objectives. While the students in general feel somewhat confident in their ability to perform similar tasks independently, the lab can be reworked with an emphasis on improving this score. The students enjoy that the lab has them implement a more significant application. However, they feel that the lab needs to improve on the level of detail of its instructions by focusing on clarity and brevity.


Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	0	0	1	8	**15**	**4.58**	**24**	**95.83%**
I found the lab exercises Interesting.	0	1	1	9	**13**	**4.42**	**24**	**91.67%**
It was clear to me what I was expected to do in the laboratory exercises.	2	2	3	**11**	6	**3.71**	**24**	**70.83%**

Table 7: Student Ratings of Lab 3 on Difficulty, Interest, and Clarity

6.2.3 Lab 3 Analysis

Survey Results

- **Is the lab challenging?**

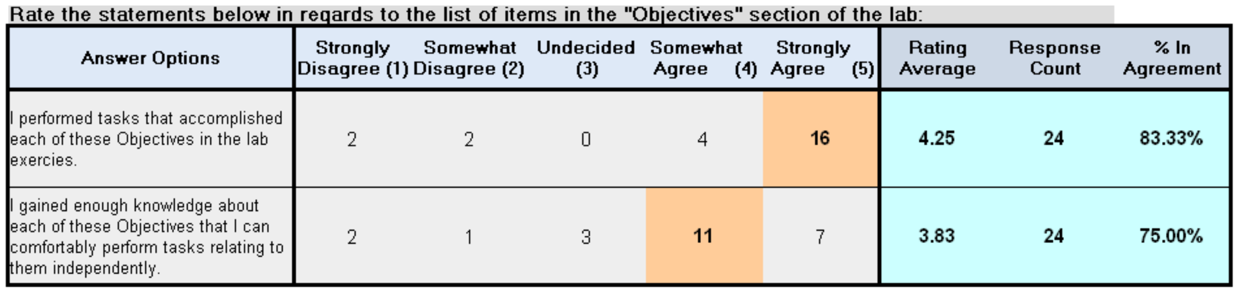
  The students almost unanimously agree the lab is challenging. As table 7 shows, most students strongly agree that the lab is challenging while only a single student out of 24 declined to agree and is instead undecided in their opinion.

- **Is the lab interesting?**

  In general, the lab was able to capture the interest of the students. Table 7 shows 91% agree to some degree with the statement about finding the labs interesting, while only single person out of 24 disagrees with the statement.

- **Is the lab clearly written?**

  The majority of students feel the lab made it reasonably clear to them what they were expected to do. Table 7 shows 70% agree to some degree with the statement about finding the labs clearly written, while the average student rating is slightly less then "Somewhat" agreeable to this statement.



Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	2	2	0	4	**16**	**4.25**	**24**	**83.33%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	2	1	3	**11**	7	**3.83**	**24**	**75.00%**

Table 8: Student Ratings of Lab 3 on Ability to Teach Learning Objectives

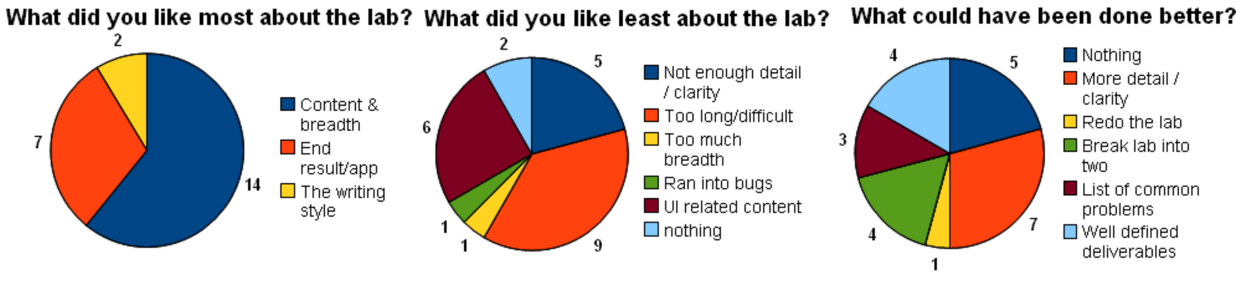


Figure 17: Student Opinions of Lab 3

- **Does the lab teach its learning objectives?**

  In general, the students are in agreement with the statement that the lab had them engage in activities that are related to its stated learning objectives. As table 8 shows, the average rating is greater than somewhat agreeing, with majority of them strongly agreeing.

- **Does the lab prepare students to work independently?**

  The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not high. The majority of students only somewhat agree with this statement, which can be see in table 8.

- **What did students like most about the lab?**

  The leftmost pie chart in figure 17 shows the majority of students identifying three things they like most about the lab. The most liked feature of the lab is the breadth of content it covered. The second most liked feature is the satisfaction of creating a more advanced application. Lastly, the students like the step-by-step, tutorial, writing style of the instructions.

- **What did students like least about the lab?**

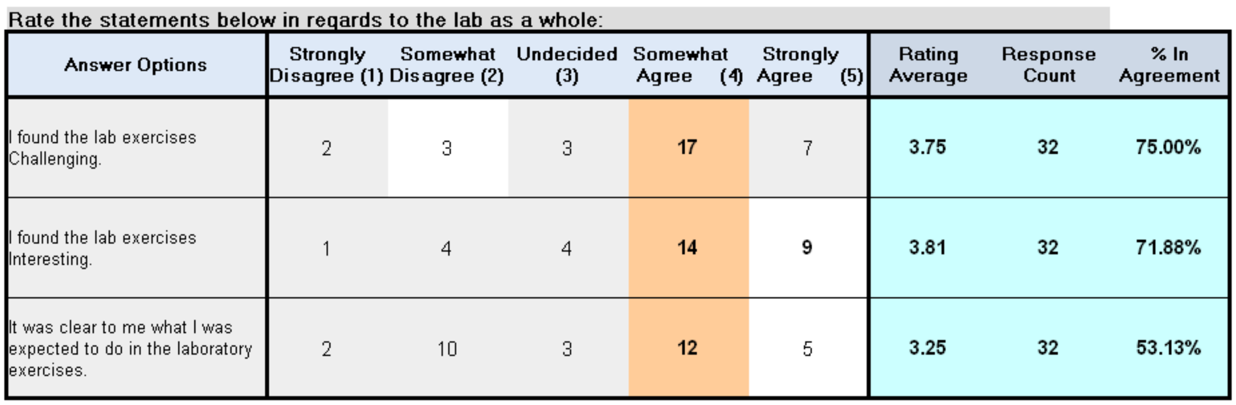
  The center pie chart in figure 17 has the students identifying a wide range of things they do not like about the lab, however, three of the answers are repeated the most. The most common complaint is that the lab is far too long and more difficult in comparison to the previous labs. The second most repeated complaints highlight specific activities related to using certain UI components. Lastly, students found that some of the instructions are either not detailed or not clear enough.

- **What did students think needs improvement?**

  The rightmost pie chart in figure 17 has the majority of students identifying four suggestions for improving the lab. The most repeated suggestion is to add more detail to or edit the current details of some instructions to improve their clarity. The second most suggested improvement is to change nothing. Next, the students feel that the lab is long enough that it should be broken into two separate labs. Lastly, the students feel that a clear list of deliverables or a correctly implemented version of the application should be provided to help the students understand how to implement their applications correctly.

Conclusions

To summarize, the students' opinions of the lab are mediocre. Students feel that it is both interesting and challenging. However, the lack of clarity in this lab most assuredly needs improvement. The lab succeeds in engaging the students in activities related to its stated learning objectives. While the students in general feel somewhat confident in there ability to perform similar tasks independently, the lab can be reworked with an emphasis on improving this score. The students enjoy the satisfaction of implementing an application that makes use of more advanced features. However, they feel that the lab needs to be reworked to adjust the length and provide a clearer list of deliverables.



Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	2	3	3	**17**	7	**3.75**	**32**	**75.00%**
I found the lab exercises Interesting.	1	4	4	**14**	**9**	**3.81**	**32**	**71.88%**
It was clear to me what I was expected to do in the laboratory exercises.	2	10	3	**12**	5	**3.25**	**32**	**53.13%**

Table 9: Student Ratings of Lab 4 on Difficulty, Interest, and Clarity

6.2.4 Lab 4 Analysis

Survey Results

- **Is the lab challenging?**

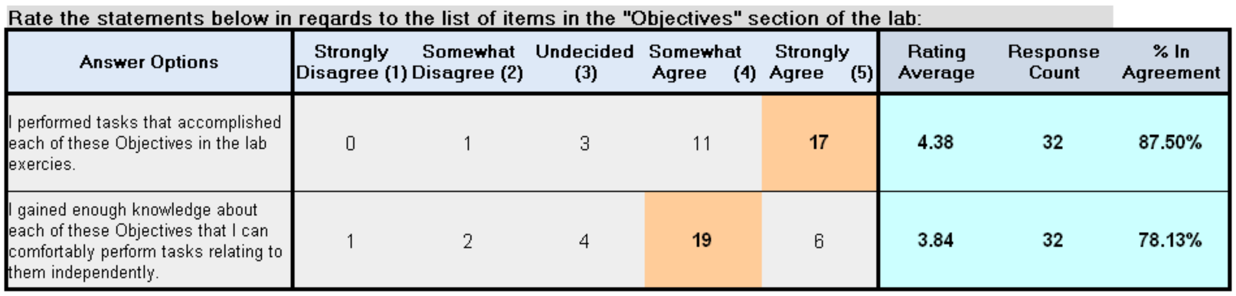
  Overall the students think the lab is somewhat challenging. As table 9 shows, most ratings are centered around "Somewhat Agree" with only five out of 32 disagreeing with the statement that the lab is challenging.

- **Is the lab interesting?**

  The majority of students think the lab is somewhat interesting. Table 9 shows 71% agree to some degree with the statement about finding the labs interesting.

- **Is the lab clearly written?**

  While the majority of students feel the lab made it reasonably clear to them what they were expected to do, there are almost as many people who somewhat disagree with this statement. Table 9 shows only 53% agree to some degree with the statement about finding the labs clearly written, while the average student rating is centered around"Undecided".



Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	0	1	3	11	**17**	**4.38**	**32**	**87.50%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	1	2	4	**19**	6	**3.84**	**32**	**78.13%**

Table 10: Student Ratings of Lab 4 on Ability to Teach Learning Objectives

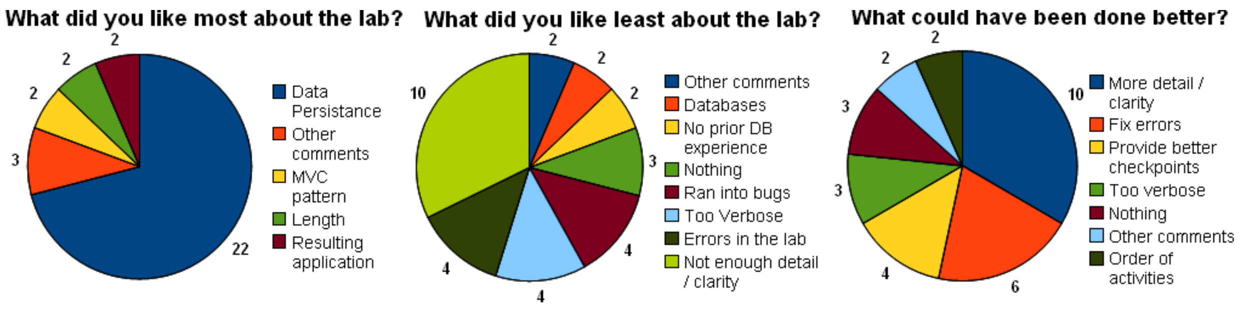


Figure 18: Student Opinions of Lab 4

- **Does the lab teach its learning objectives?**

  In general, the students are in agreement with the statement that the lab had them engage in activities that are related to its stated learning objectives of the lab. As table 10 shows, 87% of students agree with the statement, with majority of them strongly agreeing.

- **Does the lab prepare students to work independently?**

  The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not high. The majority of students only somewhat agree with this statement, which can be see in table 10.

- **What did students like most about the lab?**

  The leftmost pie chart in figure 18 has the majority of students identifying four things they like most about the lab. The most liked feature of the lab is the fact that it covers Data Persistence topics like SQLite Databases and Shared Preferences. This response is repeated most often by students. A handful of students voted for other favorite features like content relating to the MVC pattern, the resulting application, and the reduced length of lab four in comparison to lab three.

- **What did students like least about the lab?**

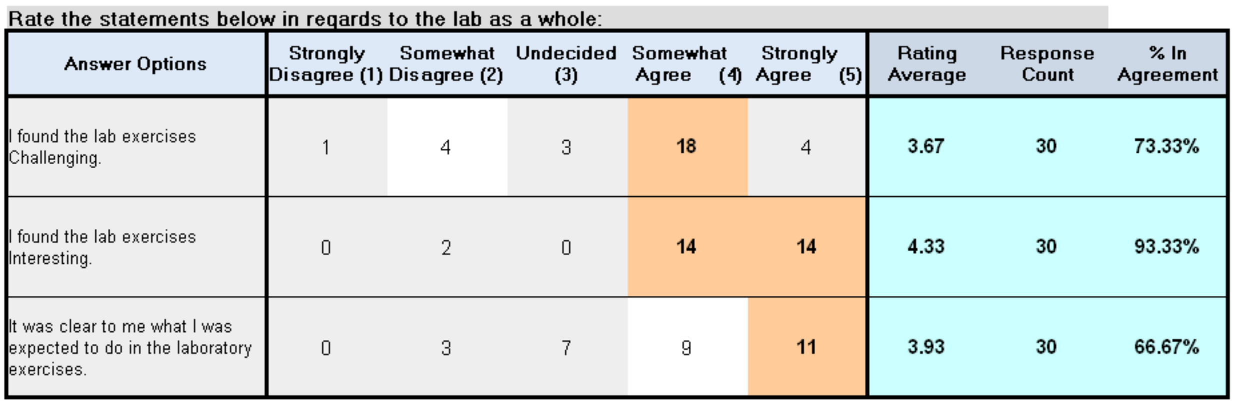
  The center pie chart in figure 18 has the students identifying a wide range of things they do not like about the lab, however, four of the answers are repeated the most. By far, the students dislike the level of detail and clarity the instructions had the most. The students also commented that the labs are too verbose, contain errors that caused problems for them, and that they had difficulty debugging their applications.

- **What did students think needs improvement?**

  The rightmost pie chart in figure 18 has the majority of students identifying three suggestions for improving the lab. The most repeated suggestion is to add more detail to or edit the current details of some instructions to improve their clarity. The second most suggested improvement is to fix a few errors in the lab that caused problems for a number of students. Lastly, the students feel that improving the quality and frequency of testing checkpoints in the lab may help ensure that they are implementing their applications correctly.

Conclusions

To summarize, the students' opinions of the lab are not that great. While students have interest in the topic of Data Persistence, the clarity of the labs instructions made it difficult for a large percentage of the students to correctly implement their applications. The lab does succeed in engaging students in activities related to its stated learning objectives. Moreover, the students in general feel relatively confident in their ability to perform similar tasks independently. The lab most assuredly needs to improve the quality of its instructions with an emphasis on clarity and correct a few errors. Lastly, the students feel that supplying more test cases for them to use as checkpoints in the lab may help them feel more confident in the correctness of their application.



Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	1	4	3	**18**	4	**3.67**	**30**	**73.33%**
I found the lab exercises Interesting.	0	2	0	**14**	**14**	**4.33**	**30**	**93.33%**
It was clear to me what I was expected to do in the laboratory exercises.	0	3	7	9	**11**	**3.93**	**30**	**66.67%**

Table 11: Student Ratings of Lab 5 on Difficulty, Interest, and Clarity

6.2.5 Lab 5 Analysis

Survey Results

- **Is the lab challenging?**

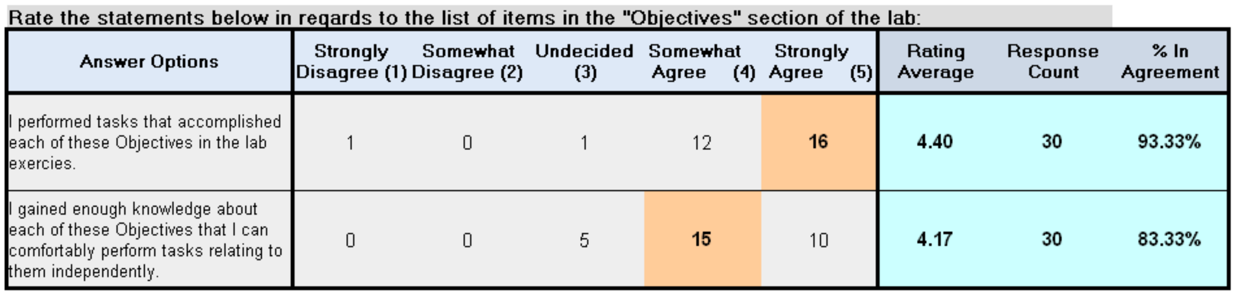
  Overall the students think the lab was somewhat challenging. As table 11 shows, most ratings are centered around "Somewhat Agree" with only five out of 30 disagreeing with the statement that the lab is challenging.

- **Is the lab interesting?**

  The students as a whole feel the lab was interesting. Table 11 shows 93% agree to some degree with the statement about finding the labs interesting, while only two people out of 30 disagree with the statement.

- **Is the lab clearly written?**

  While the majority of students feel the lab made it reasonably clear to them what they were expected to do there are a good number of students who were still at least somewhat confused. Table 11 shows only 66% agree to some degree with the statement about finding the labs clearly



Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	1	0	1	12	**16**	**4.40**	**30**	**93.33%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	0	0	5	**15**	10	**4.17**	**30**	**83.33%**

Table 12: Student Ratings of Lab 5 on Ability to Teach Learning Objectives

written, while the average student rating is slightly less than "Somewhat" agreeable.

- **Does the lab teach its learning objectives?**

  In general, the students firmly agree with the statement that the lab had them engage in activities that are related to it's stated learning objectives of the lab. As table 12 shows, 93% of students agree with the statement, with majority of them strongly agreeing.

- **Does the lab prepare students to work independently?**

  The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not high. The majority of students only somewhat agree with this statement while not a single person disagrees, which can be see in table 12.

- **What did students like most about the lab?**

  The leftmost pie chart in figure 19 shows the majority of students identifying three things they like most about the lab. Students most like that they learned how to use the Google Maps APIs. The second most liked feature is that the students got to interact with the GPS and camera hardware.

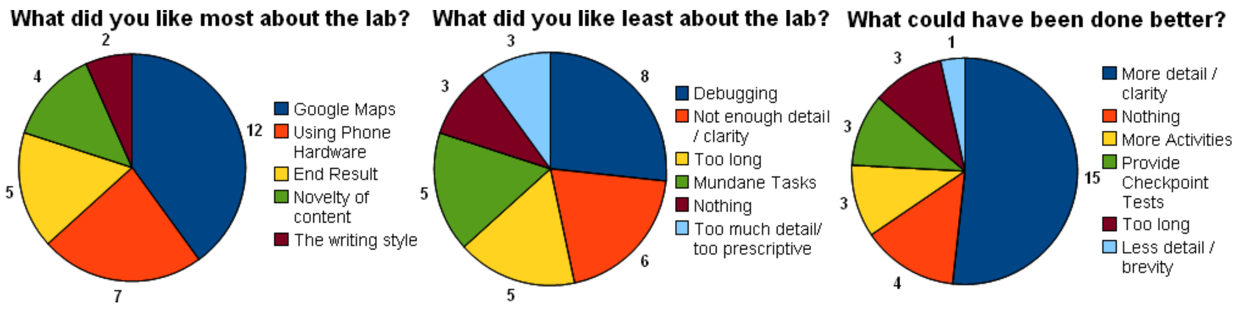


Figure 19: Student Opinions of Lab 5

Lastly, the students are greatly pleased with the application the lab had them implement.

- **What did students like least about the lab?**

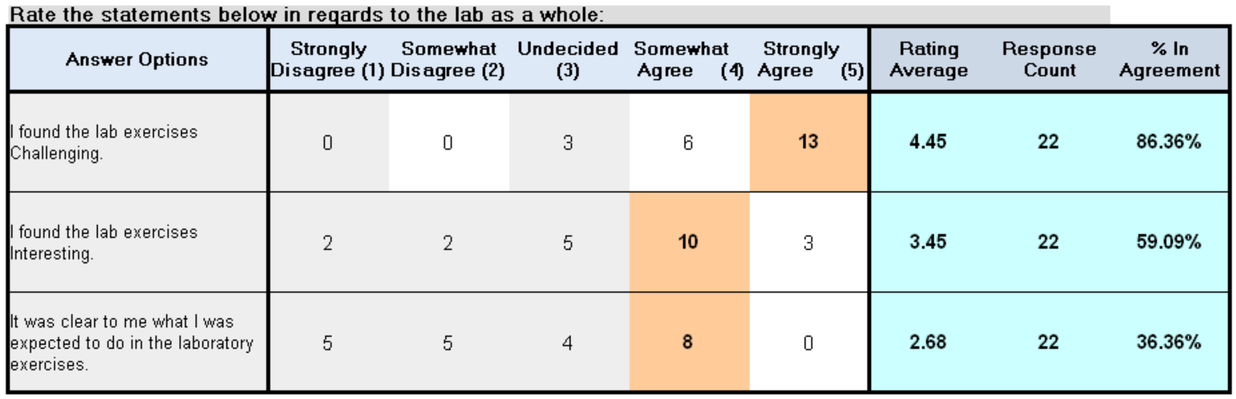
  The center pie chart in figure 19 has the students identifying a wide range of things they do not like about the lab, however, four of the answers are repeated the most. By far, the students dislike the effort required to debug and test their applications' interactions with the GPS hardware. Next, the students are still unsatisfied with the level of detail and clarity the instructions had. Lastly, students mentioned that the lab is too long and it had them perform mundane tasks like file I/O.

- **What did students think needs improvement?**

  The rightmost pie chart in figure 19 has the majority of students identifying three suggestions for improving the lab. The most suggested improvement is to add more detail to or edit the current details of some instructions to improve their clarity. The second most repeated comment is that the lab is fine the way it is. Lastly, the students feel that providing tests to use as checkpoints at certain places in the lab would greatly help their development efforts.

Conclusions

To summarize, the students' opinions of the lab are reasonably good. Students feel that it is both interesting and challenging. However, the lack of clarity in this lab most assuredly needs improvement. The lab succeeded in engaging the students in activities related to its stated learning objectives. While the students in general feel somewhat confident in their ability to perform similar tasks independently, the lab can be reworked with an emphasis on improving this score. The lab needs to improve the quality of its instructions with an emphasis on clarity and brevity. Lastly, the students feel that supplying test cases to use as checkpoints would help them implement the lab correctly.



Rate the statements below in regards to the lab as a whole:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I found the lab exercises Challenging.	0	0	3	6	**13**	**4.45**	**22**	**86.36%**
I found the lab exercises Interesting.	2	2	5	**10**	3	**3.45**	**22**	**59.09%**
It was clear to me what I was expected to do in the laboratory exercises.	5	5	4	**8**	0	**2.68**	**22**	**36.36%**

Table 13: Student Ratings of Lab 6 on Difficulty, Interest, and Clarity

6.2.6 Lab 6 Analysis

Survey Results

- **Is the lab challenging?**

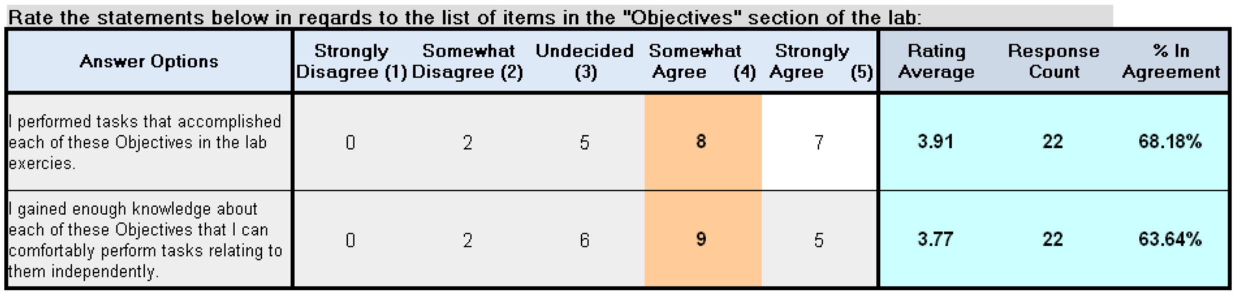
  Overall the students think the lab is challenging. As table 13 shows, most ratings are centered between "Somewhat" and "Strongly" agree with not a single person disagreeing with the statement that the lab is challenging.

- **Is the lab interesting?**

  While there is some interest in the lab, the students do not find this lab particularly interesting. Table 13 shows only 59% agree to some degree with the statement about finding the labs interesting, while the average rating is centered between "Undecided" and "Somewhat Agree."

- **Is the lab clearly written?**

  The majority of students do not feel the lab made it reasonably clear to them what they were expected to do. There are a good number of students who were at least somewhat confused. Table 13 shows only 36% agree to some degree with the statement about finding the labs clearly



Rate the statements below in regards to the list of items in the "Objectives" section of the lab:

Answer Options	Strongly Disagree (1)	Somewhat Disagree (2)	Undecided (3)	Somewhat Agree (4)	Strongly Agree (5)	Rating Average	Response Count	% In Agreement
I performed tasks that accomplished each of these Objectives in the lab exercies.	0	2	5	**8**	7	**3.91**	**22**	**68.18%**
I gained enough knowledge about each of these Objectives that I can comfortably perform tasks relating to them independently.	0	2	6	**9**	5	**3.77**	**22**	**63.64%**

Table 14: Student Ratings of Lab 6 on Ability to Teach Learning Objectives

written, while the average student rating is centered between "Somewhat Disagree" and "Undecided."

- **Does the lab teach its learning objectives?**

  The students tend towards agreement with the statement that the lab had them engage in activities that were related to its stated learning objectives of the lab. Although table 14 shows the average rating slightly less than "Somewhat Agree," the majority of students agree with the statement to some degree.

- **Does the lab prepare students to work independently?**

  The majority of students agree that they feel capable of performing similar tasks on their own. However, their confidence in this ability is not high. The average rating is slightly less than "Somewhat Agree," which can be see in table 14.

- **What did students like most about the lab?**

  The leftmost pie chart in figure 20 shows the majority of students identifying three things they like most about the lab. Students most like that they learned how to work with Services and ContentProviders. The second most liked feature is that this lab does not hold the students' hands

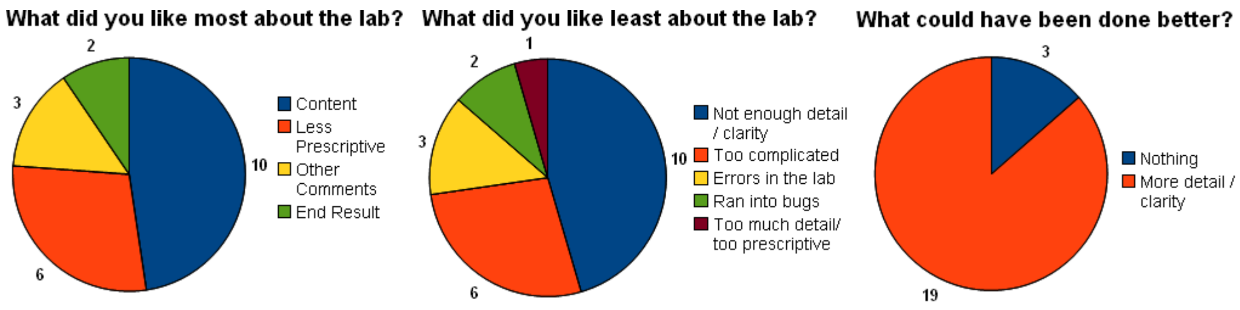


Figure 20: Student Opinions of Lab 6

as much as the previous labs. Lastly, the students are greatly pleased with the application the lab had them implement.

- **What did students like least about the lab?**

  The center pie chart in figure 20 has the students identifying a wide range of things they do not like about the lab, however, three of the answers are repeated the most. By far, the students dislike the level of detail and clarity the instructions have. Next, the students feel the implementation of this lab is too complicated. Lastly, students mention a few errors in the lab that caused problems for them.

- **What did students think needs improvement?**

  The rightmost pie chart in figure 20 has the majority of students identifying two suggestions for improving the lab. The most suggested improvement is to add more detail to or edit the current details of some instructions to improve their clarity. Aside from the previous improvement, the rest of the students feel nothing needs to be improved.

Conclusions

To summarize, the students' opinions of the lab are mediocre. Students feel that it is not that interesting, but it is still challenging. Additionally, the lack of clarity in this lab's instructions need some serious improvement. The confusion students feel might have had some affect on the lower rating the lab received for being able to have the students engage in activities related its learning objectives. However, the students in general feel somewhat confident in there ability to perform similar tasks independently. Although, the lab can still be reworked with an emphasis on improving this score.

6.3 Automated Grading Analysis

In the section that follows, we evaluate the system for automated grading with respect to the needs outlined in its design section. We use both subjective anecdotal evidence and objective statistical evidence to perform this evaluation. Anecdotal evidence comes from experiences using the system for automated grading. Statistical evidence comes from comparing the grades students received for each of the labs from both the winter and summer courses. We can then ascribe conclusions from this comparison, at least in part, to the differences in grading techniques between the two classes. As described in prior sections, the labs for the winter students were graded less stringently while the majority of the labs for the summer students were graded much more rigorously using the system for automated grading.

Usability:

With regards to the need for a solution that is easy to use and requires little to no human interaction or oversight, our system of automated grading needs work. The Grader application can be executed easily from the command line on a directory of student submissions, however the submissions generally come as an archived file. The user of the Grader application must then manually unzip all of the student submissions. Additionally, manual oversight is needed to translate an individuals test results into a formal grade using a rubric. This interaction is a tedious process that could most definitely be automated as well. While our system for automated grading removes the most tedious process of evaluating a submission, there is still a lot of interaction that could be automated as well.

Reusability:

With regards to the need for a solution that is reusable in future offerings of the course, our system is acceptable. Since the automated tests and rubrics are tied directly to the lab manuals and skeleton projects, the entire system can be reused wherever the lab materials are reused. What prevents our system from being more successful is the degree to which it is tied directly to the lab manuals and skeleton projects. Any sort of change to a lab manual or skeleton project will at the very least require the related tests and rubric to be checked to ensure the change is properly reflected their as well. In worst case scenarios, existing tests and rubrics may have to be refactored and additional tests may have to be written. The tightly coupled nature of the tests to their corresponding lab materials thus requires our system for automated grading to be maintained along with the labs materials themselves.

Accuracy:

With regards to the need for a solution that provides student grades that are as accurate as possible, our system excels. The fine grained nature of the automated tests allows us to verify the correctness of a student submission that is far more accurate than the previous system for grading. To start with, our solution can identify partial credit in the event that a student has not completed a lab. Without partial credit students would receive lower lab grades for partial implementations. Additionally, our solution can identify false positives, or situations in which the observed behavior of an application is correct, but the underlying implementation is incorrect. The presence of false positives causes a student to receive a grade higher than they deserve. The previous system for grading would require a submission's source code to be analyzed by a human

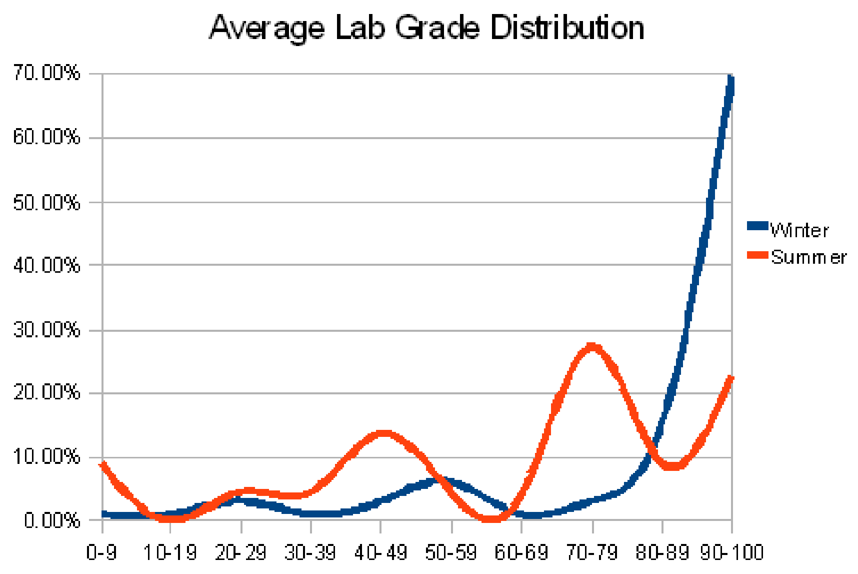


Figure 21: Comparison of Average Lab Grade Distributions

grader to catch either grading problems, which is an error-prone process. While it is possible that these two types of grading problems could cancel each other out to have no effect on the lab grades, our intuition is that false positives are a larger issue. Thus, we feel that the grades produced for labs using the previous manual grading process tend to be overinflated.

Since using our system for automated grading we have seen marked improvements in the average grade distribution for labs evaluated by our automated grading system. As you can see from figure 21, the distribution of average lab grades has become more normalized and has produced a lower mean average lab grade. The distribution and mean produced by the automated grading system is more consistent with what we would expect. Furthermore, the lowered mean average lab grade reinforces our opinion that false negatives are a larger issue than partial credit. We must also acknowledge here that the differences in the grade distributions may also have some correlation with the differences between the two classes from which the grades were taken from. These differences are outlined in the Student Performance section of the Aggregate Lab Analysis.

Traceability:

With regards to the need for a solution that clearly traces evaluation criteria to the requirements set forth in the lab manuals, our solution excels at a coarse grained level. The rubrics act as a sort of traceability matrix, mapping lab manual sections to automated tests. The rubrics do not identify the individual requirements set forth in the lab manuals that are under test in the corresponding automated test classes. However, the test classes are thoroughly commented and clearly state what they are testing. The rubrics could be improved to trace requirements under test at a fine grained level, but this would create more maintenance work in the event that the labs need to be refactored.

Visibility:

With regards to the need for a solution that presents grades results in a format that is easily readable, our solution is acceptable. Visibility is achieved through three features of our solution, a combined test results file, an individual test results file, and a grading rubric file.

- **Combined Test Results:** At first glance this file is difficult to read as it is a plain comma separated values file containing mostly error output from the Java console. However, it is not crucial for this file to be easily readable as it is used rarely and only to help identify problems at a high level. In most cases, the person overseeing the grading process may only glance at this file to see the relative frequency of errors with respect to the entire class. With this in mind, an effort could be made to pre-sort the data so that common errors are grouped and the most repeated errors appear at the top.

- **Individual Test Results:** While this file may not look very appealing, it does its job well by displaying pertinent information from the results of executing a test suite on a student submission. Both students and the person overseeing the grading process can clearly see which tests if any failed or caused errors in a student submission. Efforts could be made to improve the format of the output to be more visually appealing. However, all the pertinent information is present and ordered in a logical manner.
- **Grading Rubric:** This file succeeds in presenting both students and the person overseeing the grading process with all the necessary information in a visually appealing manner. Viewers of the file can easily identify test failures and grades. While improvements could be made through usability testing, the current state of the grading rubrics are more than acceptable.

Transparency:

With regards to the need for a solution that provides students with a clear explanation of how their grade was calculated, our solution excels. The grading rubrics provide a obvious break down of how a grade is calculated. They clearly display how much each section of the lab contributes to the overall grade. Then they identify the automated test used to evaluate each of those sections of the lab as well as the results of those tests. After viewing the grading rubric, a student can view their individual test results file to see in detail the output of any test failures. Armed with this information, a student can then investigate the cause of these failures in their submission. In the event that the failure is caused by one of the tests and not by the submission, the student then has recourse to refute a test failure.

Insightfulness:

With regards to the need for a solution that provides insight into the quality of the labs themselves, our solution is acceptable. Through the combined test results file we can identify commonly thrown errors and test failures from all the student submissions for a particular lab. This creates a feedback loop where the labs can fixed or modified, assigned to the students, graded, and then fixed or modified again as indicated by the combined test results file. Additionally, since our grading process is more standardized, consistent, and accurate for every student submission the grades become a more reliable indicator for the quality of the assignments and the students' understanding of the subject matter.

7 Conclusion

7.1 Labs

The labs created for this thesis teach students about a broad range of Android topics with enough depth to enable them to develop Android applications on their own. Based on the student responses taken from surveys, students think the labs are interesting, clearly written, and challenging. Students also feel that the labs accomplish their stated learning objectives and prepared them to work independently. These findings are supported by student performance on the labs. However, the labs need to be refactored with an emphasis on brevity and clarity. In the end, the labs accomplished everything they intended to accomplish.

7.2 Automated Grading

Using both subjective anecdotal evidence and objective statistical evidence, we are unable to conclude whether or not the System for Automated Grading is a successful endeavor. However, we feel that the it had a net benefit. The System for Automated Grading provides a reusable and highly accurate solution which clearly traces evaluation criteria to the their defining requirements set forth in the lab specifications. The output generated by the System for Automated Grading displays grades in a manner that is easy to read and understand, while at the same time creates a feedback loop to further refine itself and the labs. The two areas where the System for Automated Grading needs improvement is in its reliance on a manually intensive process for translating test results into student grades and the highly coupled relationship between lab manuals, automated tests, and grading rubrics. In situations where there are lots of student submissions to grade that require an accurate and rigorous testing process then our solution is worth the time spent on maintenance and manual interaction. In situations where there a few submissions to grade with a small set of acceptance criteria then our system is not worth the overhead.

References

[1] Open Handset Alliance. Open handset alliance. Internet, May 2010. http://www.openhandsetalliance.com/index.html.

[2] W. Enck, M. Ongtang, and P. McDaniel. Understanding android security. *Security & Privacy, IEEE*, 7(1):50–57, Jan.-Feb. 2009.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[4] The NPD Group. Android shakes up us smartphone market. Internet, May 2010. http://www.npd.com/press/releases/press_100510.htm.

[5] Apple Inc. Distribute your app – iphone developer program – apple developer. `http://developer.apple.com/programs/iphone/distribute.html`, 2009.

[6] Apple Inc. iphone development guide. `http://developer.apple.com/iphone/library/documentation/Xcode/Conceptual/iphone_development/000-Introduction/introduction.html`, March 2010.

[7] Google Inc. Android compatibility. Internet, June 2010. http://developer.android.com/guide/practices/compatibility.html.

[8] Google Inc. Api demos. Internet, August 2010. http://developer.android.com/resources/samples/ApiDemos/index.html.

[9] Google Inc. Application fundamentals. Internet, August 2010. http://developer.android.com/guide/topics/fundamentals.html.

[10] Google Inc. Application resources. Internet, August 2010. http://developer.android.com/guide/topics/resources/index.html.

[11] Google Inc. Binding to data with adapterview. Internet, August 2010. http://developer.android.com/guide/topics/ui/binding.html.

[12] Google Inc. Camera. Internet, August 2010. http://developer.android.com/reference/android/hardware/Camera.html.

[13] Google Inc. Creating menus. Internet, August 2010. http://developer.android.com/guide/topics/ui/menus.html.

[14] Google Inc. Data storage. Internet, August 2010. http://developer.android.com/guide/topics/data/data-storage.html.

[15] Google Inc. Designing for performance. Internet, August 2010. http://developer.android.com/guide/practices/design/performance.html.

[16] Google Inc. Designing for responsiveness. Internet, August 2010. http://developer.android.com/guide/practices/design/responsiveness.html.

[17] Google Inc. Developing in eclipse, with adt. Internet, June 2010. http://developer.android.com/guide/developing/eclipse-adt.html.

[18] Google Inc. Developing in other ides. Internet, June 2010. http://developer.android.com/guide/developing/other-ide.html.

[19] Google Inc. Do conserve the device battery. Internet, August 2010. http://developer.android.com/guide/practices/design/seamlessness.html.

[20] Google Inc. Intents and intent filters. Internet, August 2010. http://developer.android.com/guide/topics/intents/intents-filters.html.

[21] Google Inc. Location and maps. Internet, August 2010. http://developer.android.com/guide/topics/location/index.html.

[22] Google Inc. Publishing you applications. `http://developer.android.com/guide/publishing/publishing.html`, June 2010.

[23] Google Inc. Saving persistent state. Internet, August 2010. http://developer.android.com/reference/android/app/Activity.html#SavingPersistentState.

[24] Google Inc. Signing your applicatin. `http://developer.android.com/guide/publishing/app-signing.html`, June 2010.

[25] Google Inc. Signing your applications. Internet, June 2010. http://developer.android.com/guide/publishing/app-signing.html.

[26] Google Inc. Tools overview. Internet, June 2010. http://developer.android.com/guide/developing/tools/index.html.

[27] Google Inc. User interface. Internet, August 2010. http://developer.android.com/guide/topics/ui/index.html.

[28] Google Inc. What is android? Internet, June 2010. http://developer.android.com/guide/basics/what-is-android.html.

[29] QUALCOMM Inc. About brew. `http://brew.qualcomm.com/brew/en/about/about_brew.html`, 2010.

[30] QUALCOMM Inc. Developer home. `http://brew.qualcomm.com/brew/en/developer/overview.html`, 2010.

[31] Adobe Systems Incorporated. Adobe – flash lite. `http://www.adobe.com/products/flashlite/`, 2010.

[32] Jari Karvonen and Juhani Warsta. Mobile multimedia services development: value chain perspective. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 171–178, New York, NY, USA, 2004. ACM.

[33] Research In Motion Limited. Blackberry app world storefront feature and technical overview. `http://docs.blackberry.com/en/developers/deliverables/7299/BlackBerry_App_World-1.0-US.pdf`, 2009.

[34] Research In Motion Limited. Java code signing keys. `http://na.blackberry.com/eng/developers/javaappdev/codekeys.jsp`, 2009.

[35] Research In Motion Limited. Vendor portal for blackberry app world^{TM}. `https://appworld.blackberry.com/isvportal/home/login.seam?pageIndex=1&cid=40385`, 2009.

[36] Research In Motion Limited. Compare blackberry smartphones. `http://na.blackberry.com/eng/devices/compare/`, 2010.

[37] Q.H. Mahmoud. Integrating mobile devices into the computer science curriculum. pages S3E–17 –S3E–22, oct. 2008.

[38] Reto Meier. *Professional Android Applicaton Development*. Wiley Publishing, Inc., Indianapolis, IN, USA, 2009.

[39] Dan Morrill. On android compatibility. Internet, June 2010. http://android-developers.blogspot.com/2010/05/on-android-compatibility.html.

[40] J. O'Connor. Attack surface analysis of blackberry devices. *White Paper: Symantec Security Response*, 2007.

[41] Rob Pegoraro. Apple rejects pulitzer winner's iphone app because it 'ridicules public figures'. `http://voices.washingtonpost.com/fasterforward/2010/04/apple_refuses_pulitzer_winners.html`, April 2010.

[42] Andy Rubin. Where's my gphone? The Official Google Blog. Google Inc., November 2007. http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html.

[43] A. Tarnacha and C. Maitland. Standards competition and the role of intermediation in the supply of mobile applications. In *34th Annual Research Conference on Communication, Information and Internet Policy*, September 2006.

[44] Ankur Tarnacha and Carleen F. Maitland. Entrepreneurship in mobile application development. In *ICEC '06: Proceedings of the 8th international conference on Electronic commerce*, pages 589–593, New York, NY, USA, 2006. ACM.