2012

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

uRespond: A Classroom Response System on the iPad

Andrew Herrmann

A Capstone Project Proposal Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science / Department of Information Systems and Operations Management

University of North Carolina Wilmington
2012

Approved by

Advisory Committee

Dr. Ron Vetter	Dr. Ulku Yaylacicegi
Dr. Nathan Grove	Dr. Bryan Reinicke, Chair

Abstract

Personal Response Systems, or "clickers," are rapidly becoming a ubiquitous presence in university classrooms. Numerous studies in STEM disciplines have demonstrated the positive benefits that the use of such systems can have. Regrettably, in these theoretically abstract subjects an instructor's use of personal response systems is hindered by the inability to ask anything other than multiple-choice questions or questions requiring short alphanumeric responses. We developed a mobile application called uRespond to overcome the limiting types of questions of traditional clicker systems by allowing for free-form student input such as drawing, graphing, calculation, or structure creation and manipulation.

This paper describes some problems with university STEM education and how our personal response system addresses them. There is discussion of the system concept building, development methodology, technical details and results of preliminary user testing. Finally it concludes with a discussion of lessons learned and future work for the system.

Table of Contents

1 Introduction

There is growing concern that the United States is falling behind other countries in the teaching of students in the areas of science, technology, engineering, and mathematics (STEM). Colleges and universities are unable to keep up with the demand for students with degrees in these fields. In the K-12 education system, decades of lowering standards ineffective teaching methods and poor curriculum have led to a lack in the quality of students entering universities. Moreover, the quantity of people graduating with STEM related degrees has fallen due to challenges these students face in university.

Among the solutions proposed to improve teaching STEM subjects are better tools and innovative ways of teaching. Technology can provide teachers with ways to implement these solutions and utilize more active teaching techniques. Mobile technology provides a platform that students are comfortable with and offers educational opportunities regardless of time or place. Educational software can be tailored to an individual student's abilities, offer content in multi-media formats and provide instantaneous feedback.

One tool many professors are turning to is personal response systems or "clickers". Clickers are thin, small hand-held remote controls with a keypad that use infrared or radio signals to send responses to a receiver placed in front of the class which feeds into a computer controlled by the instructor. These responses are collected and presented to the instructor. In class, when the instructor poses a question and asks the students to respond, each student clicks an answer, which is picked up by the receivers and stored with the student's records by the software.

Numerous studies in STEM disciplines have pointed to the positive benefits clickers can have. Unfortunately, in these theoretically abstract subjects an instructor's use of personal response systems is hampered by the inability to ask anything other than multiple-choice questions or questions requiring short alphanumeric responses.

Dr. Nathaniel Grove of the University of North Carolina Wilmington (UNCW) Department of Chemistry and Biochemistry has proposed a solution through the development of uRespond, a suite of tools that will convert Apple's iPad into an interactive personal response system for use in the chemistry classroom. uRespond will overcome the limiting types of questions of traditional clicker systems by providing for the development of a wide range of drawing, graphing, calculation, and structure creation and manipulation skills. In addition, a question creation system will be integrated to allow teachers to create questions, view responses in real time and perform simple class management tasks.

2 Literature Review and Analysis

A strong STEM-based segment of the workforce is critical for the competiveness of America and the stability of its economy. People with STEM degrees enjoy lower unemployment rates and receive better pay over the course of their careers compared to the national average income [1, 2]. STEM occupations are vital because they generate the technological changes that shape other occupations and more significantly, innovation which fuels economic growth [1].

There has been no shortage of efforts to reconcile the issues in STEM education by both federal and state governments as well as by private organizations. A study

conducted by the Government Accountability Office found that approximately $3 billion had been spent on federal STEM education programs in 2004 [3]. In 2006, President George Bush introduced the American Competitiveness Initiative, a federal assistance program aimed at strengthening STEM education in high schools [3]. The Education Commission of the States reported in 2007 that some states had raised math and science requirements and enhanced teacher training in response to the STEM challenge [4]. Private foundations and industry leaders such as the Bill and Melinda Gates Foundation and Raytheon have put their support behind STEM education programs as well [4].

Despite the attention STEM education and competitiveness have received, American students continue to lag behind their counterparts in other countries. A test issued by the Program for International Student Assessment in 2009 showed that U.S. students ranked 23rd in science, and 31st in math among the 65 countries participating [5].

2.1 Shortage of STEM workers

The approaching retirement of a large segment of the existing STEM workforce combined with expected growth in STEM related industries point to an emergent need for more STEM field graduates [4, 6]. According to a report from the Center on Education and the Workforce, the postsecondary system will have produced 3 million fewer college graduates than demanded by the labor market by 2018. The shortage of STEM graduates has already had an impact on American businesses. According to a 2010 congressional report:

- 77% of global companies surveyed said they were planning to expand operations in China and India [7].
- America's cut of high-tech exports fell from 21% to 14% while China's rose from 7% to 20% [7].
- GE has now located the majority of its R&D personnel outside the United States [7].
- Manufacturing employment in the U.S. computer industry is now lower than when the first personal computer was built in 1975 [7].
- In the 2009 rankings of the Information Technology and Innovation Foundation the U.S. was in sixth place in global innovation-based competitiveness, but ranked 40th in the rate of change over the past decade [7].
- Almost one-third of U.S. manufacturing companies responding to a recent survey say they are suffering from some level of skills shortages [7].

2.2 Failures in STEM education

Trends in the overall supply and employment of STEM field workers are one concern for Americans. Equally important is preparation for STEM success. Many students never make it into STEM programs, because of inadequate preparation in math and science or poor teacher quality in their K-12 systems [4]. Of the high school graduates who took the ACT test in 2005, for example, only 41 percent achieved the College Readiness Benchmark in mathematics and 26 percent achieved that benchmark in science [8].

Furthermore, those who are academically qualified for college level studies in science and math fields often do not pursue those programs. They might be dissuaded by demanding curricula or strict grading practices within STEM majors [4]. A "sink or swim" mentality also discourages many students from pursuing degrees in the STEM areas [9].

Traditional ways of teaching STEM at universities have also frustrated students. These classes have historically relied on lecture-based teaching methods, treating the student as a passive observer [10, 11]. In a standard university-level class, an instructor would present material for the duration of the class, typically 50-75 minutes. This requires students to maintain focus for the length of the class although research has shown that the average attention span of an adult listening to speaker is only 10–30 minutes [5, 10]. It is likely that most students would lose concentration several times during the lecture, affecting their ability to retain information and understand the material.

2.3 Use of clickers to improve university STEM education

In recent decades, more student-centered models of learning have emerged – the most prevalent of which is constructivism. According to the constructivist model, knowledge is actively constructed by each learner [10, 12]. For this process to be meaningful, three key components must be present [11-14]. First, students must possess relevant prior knowledge upon which to anchor new knowledge. Second, the student must perceive this new knowledge as "relevant to other knowledge". Third, the learner

"must consciously and deliberately choose to relate new knowledge to knowledge the learner already knows in some nontrivial way" [14].

In efforts to promote and enhance student learning, there has been a growing interest in recent years to utilize constructivist teaching techniques in the STEM curriculum [15-22]. Clickers are being adopted at an increasing rate in science classrooms and lecture halls and provide instructors with a convenient means of actively engaging their students. Studies conducted in mathematics and other science disciplines over the course of the last fifteen years show that clickers can have a variety of positive impacts on student learning and affect [23-34]. The use of clickers in the classroom can lead to greater student engagement by enhancing discussion and interactivity among students and between the students and the instructor [26, 229, 31-34]. Students can develop an increased awareness of their own understanding of the subject matter [29, 30, 33]. Clickers also provide immediate feedback allowing instructors to assess students' understanding of material just presented and allow for responsive teaching practices based upon the results [26, 28, 29].

Clickers are especially beneficial in STEM class auditorium settings where student numbers are large and individual interactions with the professor are limited. Professors can determine if students understand important points or distinctions raised in class. They need not wait until homework is turned in or exams are completed to do so. Instead, they can receive feedback on a lecture during that same lecture and revisit or clarify the points he or she just made in class if required [26, 28, 29]. For students that have questions or wish to engage the professor, doing so in large enrollment courses can be intimidating. Victor Edmonds, an educator at UC Berkley, writes about students using

clickers: “Putting up your hand in class is pretty complex thing, kind of dangerous, socially and academically. But everyone is willing to give anonymous answers. Everyone is equally involved and the answers are honest. And fast.” [35]

2.4 Teaching Lewis structures

Many STEM students are required to take general chemistry courses to understand the relationship between molecular-level structure and the behavior of physical chemical and biological systems. The capacity to predict the properties of a substance from its molecular structure (and vice versa) is fundamental to such an understanding.

A key step toward understanding structure–property relationships is the ability to draw and manipulate the simple chemical structures known as Lewis structures. Lewis structures are illustrations that show the distribution of electrons in atoms of a molecule with chemical symbols, lines and dots (See figure 1)[36]. These diagrams can convey a great deal of structural information and can be used to explain a substance's physical and chemical properties.

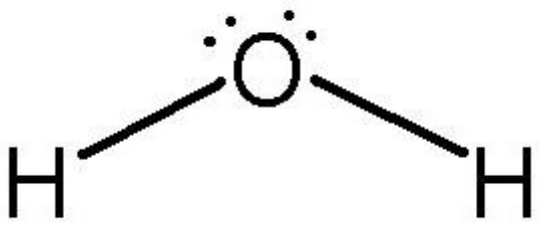


Figure 1: Lewis Structure of Water

2.5 Mobile devices in university classrooms

With the proliferation of mobile devices students are increasingly connected to technology. A survey conducted by Educause found that 10% of college students used a mobile device to access the Internet every day in 2008. That number rose to 43% by 2010 [37].

Mobile technologies provide an opportunity to create new and engaging educational activities on a medium students know well and feel comfortable using. Mobile devices are well suited for classroom use due to their small size, rich functionality and ability to communicate wirelessly. Research on the use of mobile devices in the classroom is scarce, however. Much of the research focuses on the role of mobile devices in distance learning [38-43] or their use to supplement and reinforce curriculum outside the classroom [44-45].

2.6 Computer aided learning and chemistry

Today, there are few schools in the United States that do not have computers available for student use, and do not use some form of computer-aided instruction on those computers [46]. A number of studies have reported that educational software can be successful in raising exam scores, improving student attitudes, and reducing the time needed to master course materials [46]. It also has been shown that students like the mode of presentation, that it is viewed as a positive experience, and that it is suitable for individual learning needs [46].

There is a variety of educational software geared specifically towards chemistry students. Virtual Lab is an application developed at Mari State Technical University,

which allows students to perform "virtual experiments" through a 3D interface [47]. Similar programs have been developed at Carnegie Mellon University, Brigham Young University and Oxford University [47]. Researchers at the Swiss Federal Institute of Technology created Augmented Chemistry, a system that uses a tangible user interface to create 3D molecular models [48].

A number of software developers have created applications for drawing Lewis structures as well. ChemPad is a pen-based application developed by Dana Tenneson at Brown University, where users can draw molecules on the computer and observe its 3D structure [49, 50]. Using a Tablet PC or interactive whiteboard, users can sketch these molecules quickly so they can be converted to 3D models that will help them visualize chemistry concepts. Programs like ChemDraw, MasteringChemistry and ACE Organic also provide interfaces for drawing Lewis structures [49].

2.7 uRespond

In 2009 Dr. Nathaniel Grove of UNCW's Chemistry Department submitted a proposal to the National Science Foundation (NSF) for the development of an interactive personal response system, uRespond. The proposal called for uRespond to include the student client, built on the iPad, a question authoring and class management tool and a central database for storing questions and answers.

The uRespond student client relies on the iPad's touch interface to draw molecular structures, graphs and other diagrams. When these notations are created with the traditional pencil-and-paper medium, they are static and only assist in initial concept and problem formulations. A laptop computer provides a dynamic medium for student

learning but makes expressing ideas less fluid and expressive because students are constrained by cumbersome input models, typically a keyboard and mouse. The iPad's combination of a computer's functionality with the expressive power of pencil and paper makes it an ideal platform for uRespond.

uRespond will greatly expand the types of questions professors can ask using personal response systems. Typical clicker systems only allow multiple-choice questions or questions that require a brief alphanumeric answer [28-31]. While these kinds of questions are useful to verify student understanding they cannot check for deep, conceptual understanding. Professors can better learn what the students know by requiring them to construct the answer. The feedback a professor receives from a graph, diagram or other student-constructed responses are much richer than a short alphanumeric responses.

One of the question types available in uRespond is the Chemical Structure Builder. This module consists of a canvas, a palette of atomic elements and tool bar for functional buttons. The user drags elements from the palette onto the canvas and uses a combination of touch gestures to create Lewis Structures. Whereas uRespond gives students full autonomy in creating Lewis Structures programs like ChemDraw will auto-complete many structures denying students the opportunity to make appropriate choices [49]. Web-based programs like Mastering Chemistry and ACE Organic constrain students by requiring them to build structures through a series of menus or buttons and often require placement of atoms on a grid [49]. These interfaces are unnatural and often have a substantial learning curve.

In addition to the Chemical Structure Builder module, uRespond will have six other question templates. These templates provide flexibility in the kinds of questions a professor could ask and are particularly useful for teaching chemistry classes as well as other STEM fields.

uRespond also expands on *ways* student can learn. Most STEM lecture classes focus on the acquisition of knowledge or "cognitive learning" for teaching students. Constructivist teaching models recommend using a variety of learning methods to actively engage the student's mind. One way students learn is through the physical or motor action known as "psychomotor learning" [51]. The uRespond system not only allows students to be cognitively active like other personal response systems, but to also practice psychomotor skills – such as graphing and structure drawing – that are not possible with currently available systems.

The uRespond system includes a web-based teacher client. The client will consist of a question authoring tool, a real-time feedback tool and a class management tool. The question authoring tool will allows teachers to create questions based on the seven question types and save them for presentation later. The real-time feedback component will allow professors to view the submissions of all of the students connected to the system. The instructor, whose display can be projected onto a screen, can use the student's own submissions anonymously as examples in subsequent class discussions. The class management tool will allow professors to upload a class roster, add, edit or delete students and take attendance. uRespond will also have a question bank available to all teachers through the Question Authoring Tool. The question bank will be populated

with approximately 150 questions of each question type that would be used in a typical two-semester organic chemistry course. These questions will be created by Dr. Grove.

3 Concept

3.1 System modeling

The uRespond project began with the May 21, 2009 National Science Foundation (NSF) proposal by professors Dr. Nathaniel Grove of UNCW and Dr. Melanie M. Cooper and Dr. Roy P. Pargas of Clemson University. The proposal detailed the need for the uRespond system in addition to the project timelines, costs and goals. The proposal was accepted by the NSF December of 2010.

Dr. Grove, the project lead, reached out to the Computer Science and Information Systems program at UNCW to form a team to develop the uRespond system. The team members would be made up of postgraduate students from the two universities involved. The development members consisted of myself, Ricardo Valea (later replaced by Rebecca Brown), Sam Bryfcynski and Josiah Hester. The project schedule would follow the timeline laid out in the NSF proposal. (See appendix 1)

Project stakeholders consisted of the principle investigators (PI) and Co-PIs of the uRespond proposal and the project sponsor, NSF. The project lead and the development members agreed upon the methods and frequency of communication. All communications from the development team would go through the project lead, Dr. Grove. Project status reports would be given in a weekly meeting with Dr. Grove and

through email as required. Communications between the UNCW and Clemson developers would occur at a monthly videoconference meeting via Skype and email as required. Dr. Grove would handle communications with the Co-PIs at Clemson and the NSF.

The project scope based on the high level requirements was finalized September 1, 2011. Dr. Grove and all members of the development team agreed to and signed the document. In it the division of responsibilities is defined: the UNCW developers will create the student client and Clemson developers will build the backend database. The two teams would collaborate on the teacher client to integrate the systems. Auto grading, a requirement of low priority for Dr. Grove, would not be included in this iteration of the system (see appendix 4).

3.1.1 Teacher Client Selection

Several turnkey solutions were researched for the teacher client component of uRespond. Many class management programs such as BlackBoard, Moodle and UNCW's own Entropy had rich functionality for class management and quiz administration but would require considerable effort to modify for the needs of uRespond.

The Chemistry and Computer Science Departments at Clemson had created a web-based quiz program called "BeSocratic" (http://besocratic.clemson.edu/index.html). BeSocratic included many of the functions required for the teacher client in uRespond:

- question authoring
- multiple question templates
- class management tools

BeSocratic also had a student client, which relied on the Silverlight framework. We briefly considered refactoring the BeSocratic student client to meet our needs but Silverlight is not compatible with iOS.

Since the developers at Clemson had intimate knowledge of BeSocratic they would know the best strategy for integrating it with the student client and could aid in the process. For these reasons Dr. Grove decided to use BeSocratic and modify it for the purposes of uRespond.

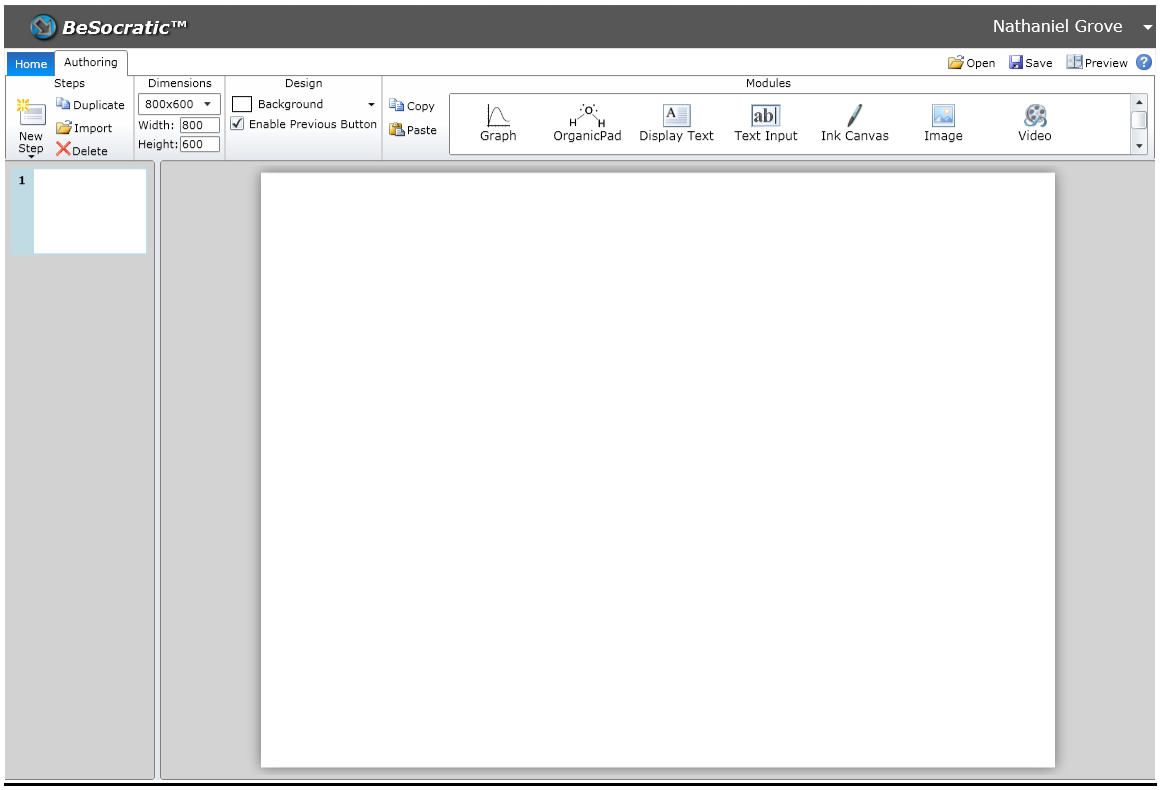


Figure 2: Question Authoring View of BeSocratic

3.2 Functional Requirements

In June of 2011 we conducted an interview with Dr. Grove to examine the functional requirements of uRespond not addressed in the NSF proposal. A list of detailed questions were prepared and discussed. Dr. Grove had previously designed a program called "OrganicPad" with his Co-PIs in 2009. Much of the user requirements came from his experience with developing the software. A set of diagrams was created based on the interview responses (See figures 2-3).

The features of highest importance were:

- Login/logout
- Send/receive activities
- Lewis structure builder module

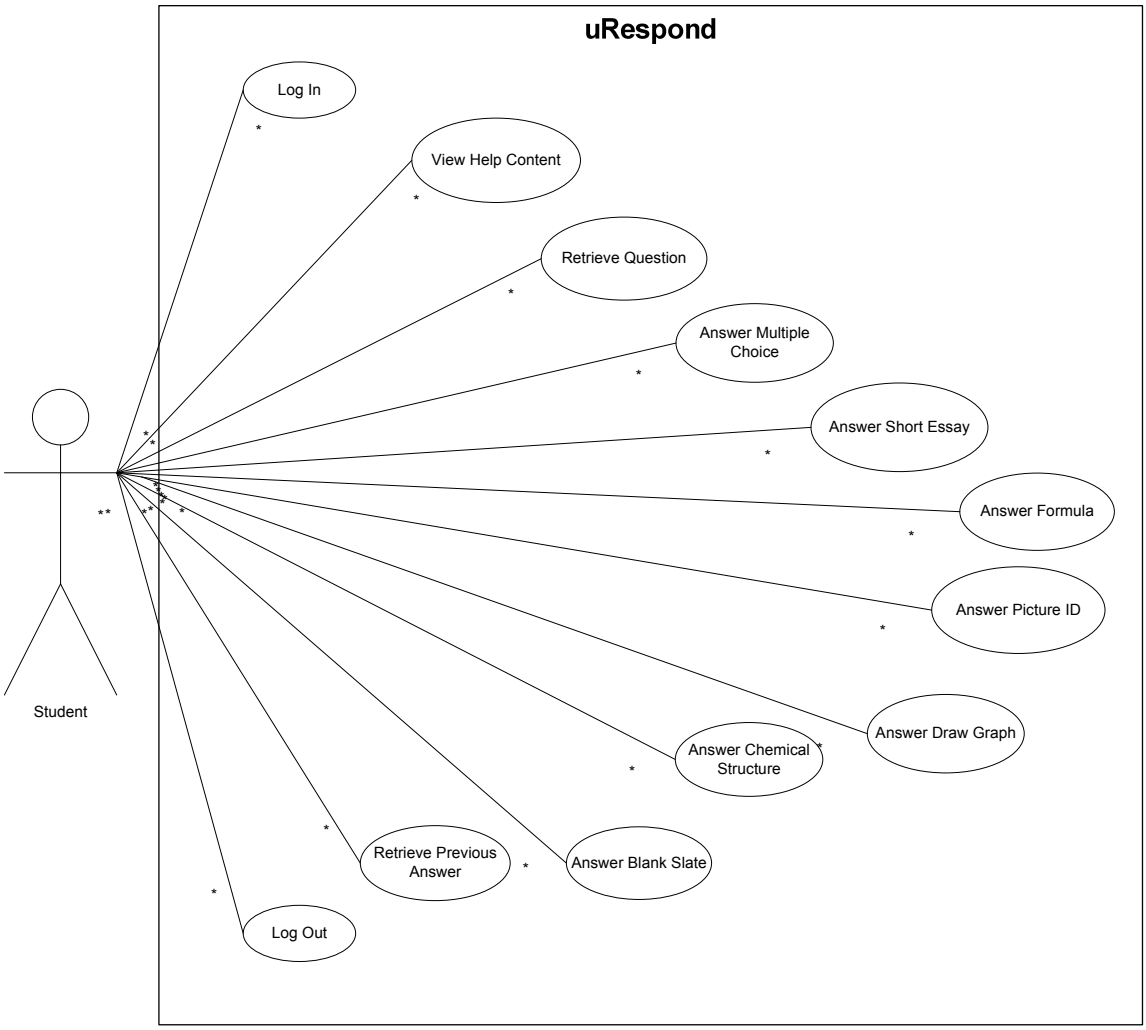


Figure 3: Student Use Cases

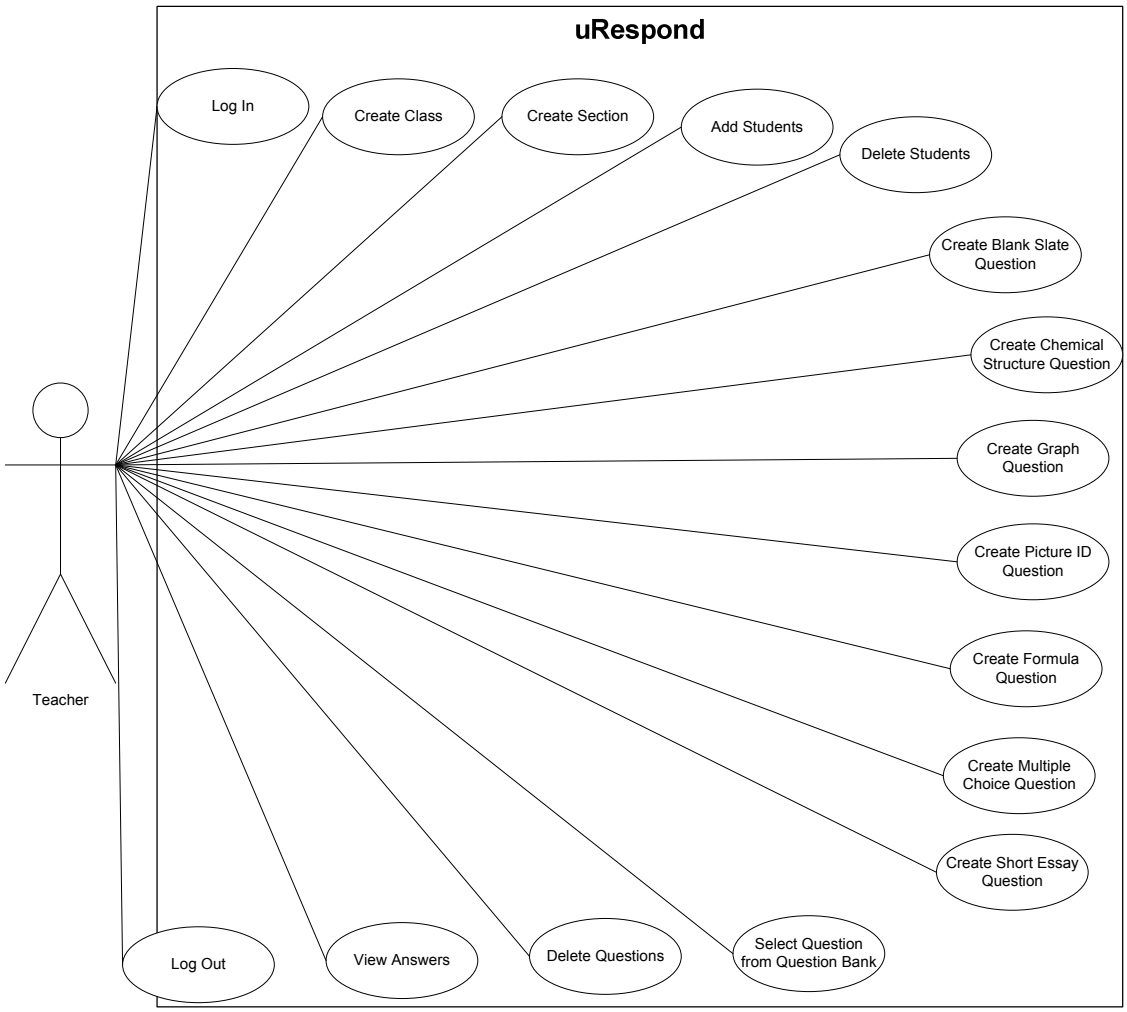


Figure 4: Teacher Use Cases

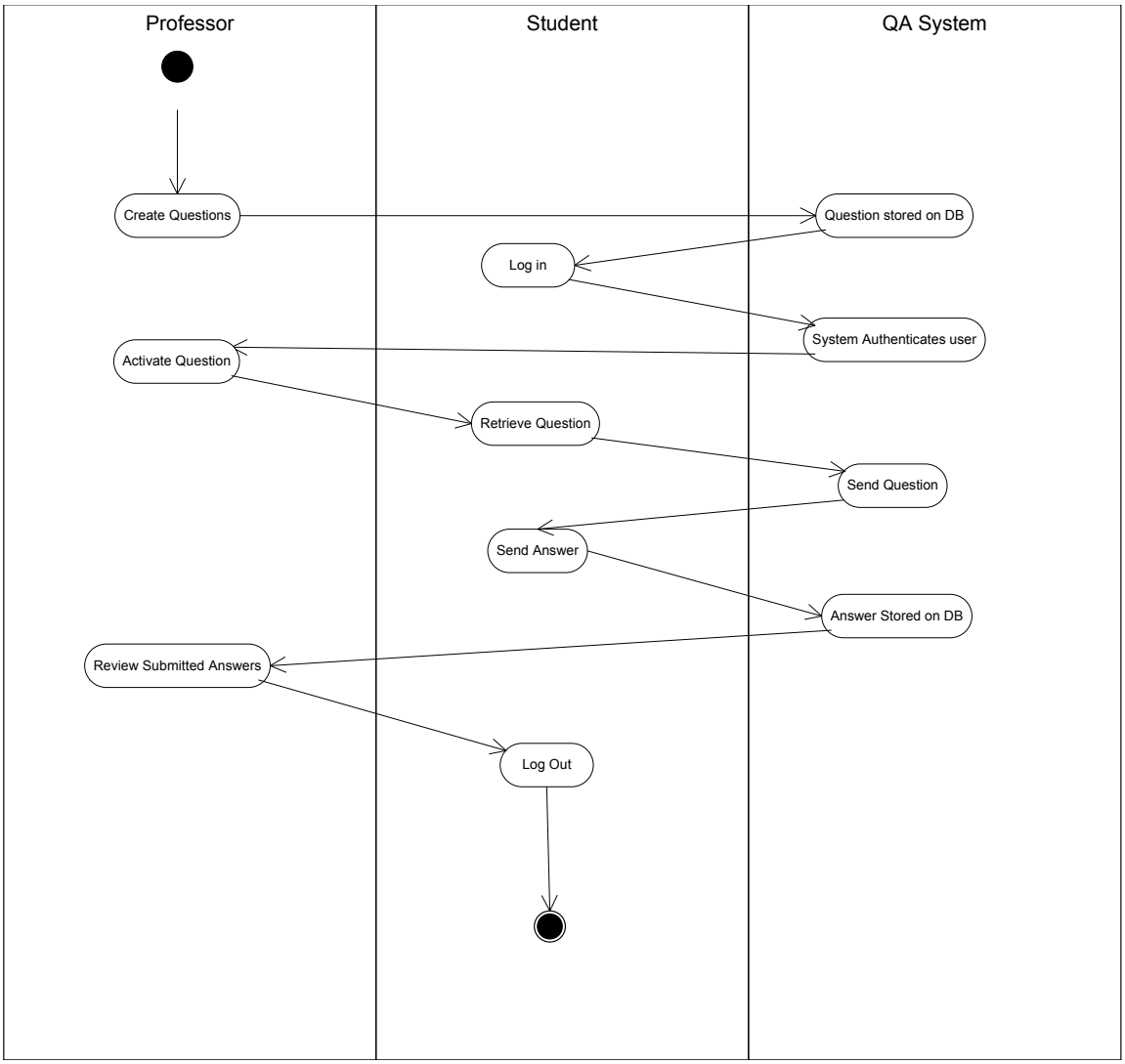


Figure 5: uRespond System Activity Diagram

3.2.1 Student Client Graphic User Interface

The graphic user interface (GUI) of uRespond was designed to maximize the workspace real estate on the screen. The application login, registration and navigation were contained in a small, unobtrusive bar across the top of the workspace. The buttons of particular question types appear at the bottom and left side of the screen leaving as much room as possible for the student to work. All of the question types were designed to be as intuitive as possible to reduce the learning curve for students. (See appendix 2)

3.2.1.1 Collaboration with Graphic Design Students

In June of 2011, the development team at UNCW reached out to Ned Irvine, a professor of graphic design at the university, to develop some of the icons and graphics for the student client of uRespond. Professor Irvine expressed interest teaming up to have his intermediate graphic design class develop the graphics. Having UNCW students design the graphics would be an ideal situation since university students are the target users for the iPad application.

The application needed to be visually appealing in order for students to want to use it. Additionally the graphics would help the students understand the functionality of uRespond. The icons and buttons needed to be fun and attractive, yet somewhat formal since the application would be used in an academic setting.

Ned Irvine and the graphic design students investigated making graphics for iOS devices. The graphic design students looked at design trends, image specifications and mobile device GUIs. Each student created a series of graphics and icons for uRespond.

The graphic design students and UNCW development team held bi-weekly meetings to present and critique the designs. The final designs were submitted December 8, 2011.

3.3 Technical Requirements

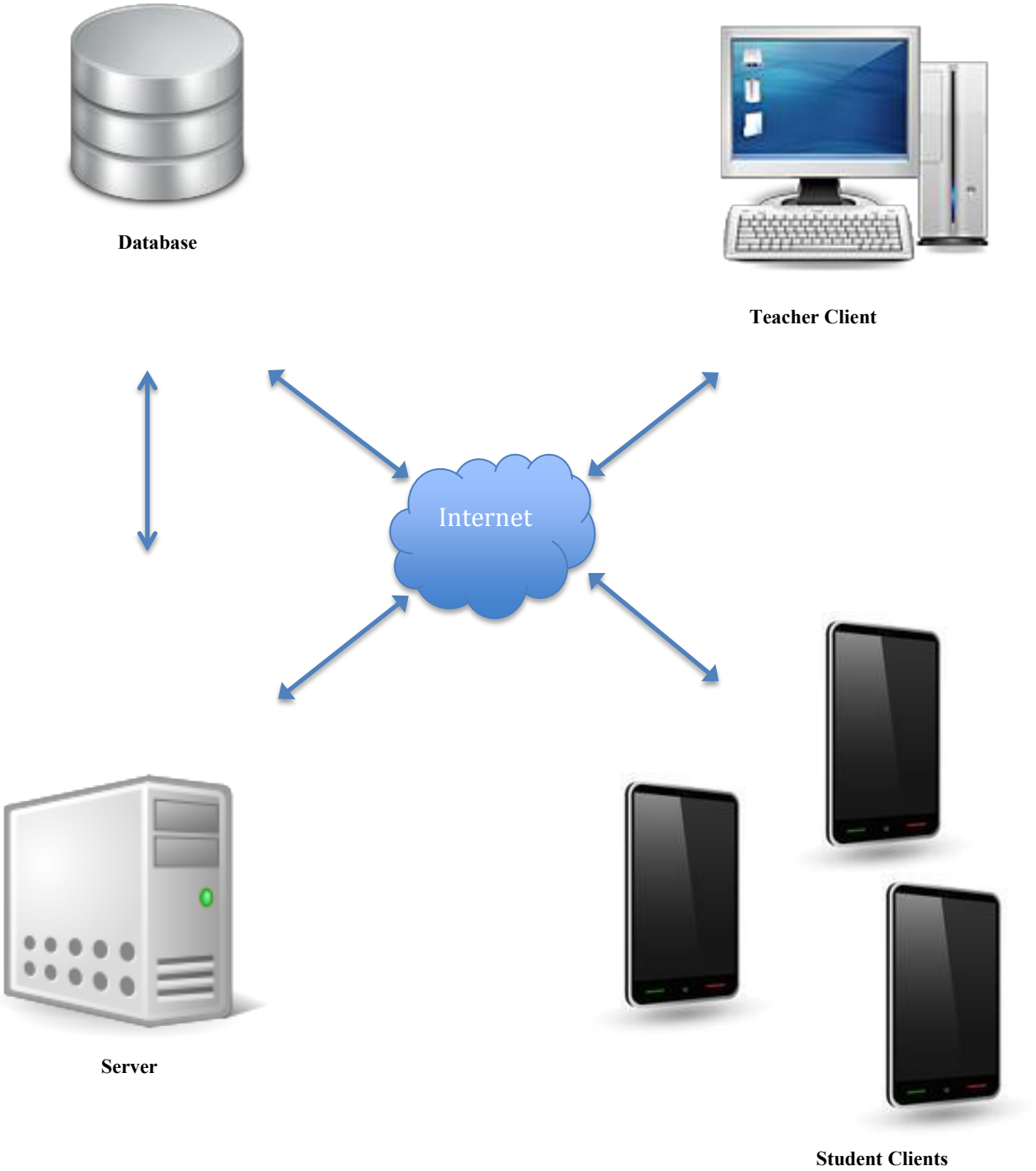
3.3.1 uRespond System Platforms

System Component	Teacher Client (BeSocratic)	Student Client	Database
OS	Server 2008 R2	iOS5	Microsoft SQL Server 2008 R2
Hardware	Dell PowerEdge x64 R610	iPad2	Dell PowerEdge x64 R610

3.3.1 uRespond Development Environments

Student Client	Teacher Client (BeSocratic)	Database	Student Client
Language	C#	SQL	Objective-C
Integrated Developer Environment (IDE)	Visual Studio 2010	Microsoft SQL Management Studio 2008	xCode
Additional Frameworks	Silverlight	REST	Sub-version

3.3.3 uRespond System Architecture



3.3.4 Database Communication

The set of questions an instructor uses in a class are collectively known as an "activity". The individual questions are called "activity steps" which are made of up various graphical and functional components called "modules". Each Activity is associated with a class roster (See Figure 6).

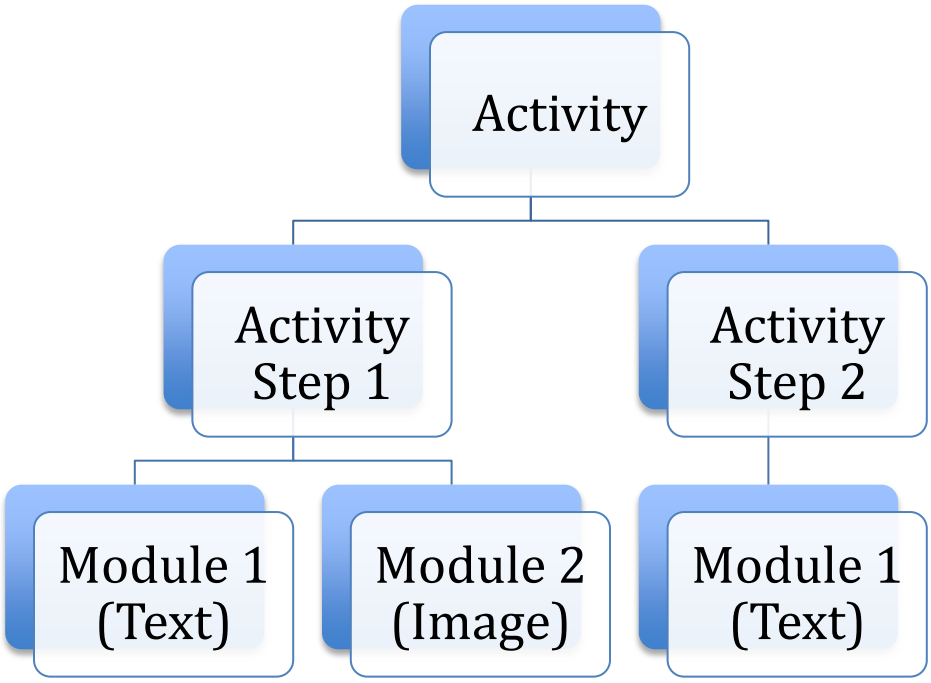


Figure 6: Example Activity

When a teacher begins a uRespond session he or she will start with a blank activity, that is there are no activity steps. The instructor can choose a question from the question bank, a prepared question he or she created or a blank question template and add it as an activity step to the activity. This activity step is then available to the students on their iPads.

When a student logs into uRespond on the iPad his or her username is associated with a roster. uRespond will query the data for an available activity for that student. The activity and first available activity step is sent to the device. The activity step type (which corresponds with the seven question types) and the modules associated with the step will determine what is displayed to the student (text for a question prompt, for example). When another activity step is available, the student will press a button to receive the next question.

Once an activity step is available to students, the teacher client will query the database for student submissions. These submissions will populate a list in the teacher client. An instructor can select a student's submission from the list to display and examine it further. (See appendix 3)

3.3.4.1 REST Web Service

The database communications with uRespond occur through RESTful requests. REST is an abbreviation for Representational State Transfer. It is a stateless architectural style used in web services [52]. The REST protocol relies on HTTP methods such as GET and POST to invoke remote functions and deliver response data [52].

We chose to use RESTful Web Service due to its limited overhead, simplicity and ease of development. The Clemson development team also had a REST web service implemented for BeSocratic.

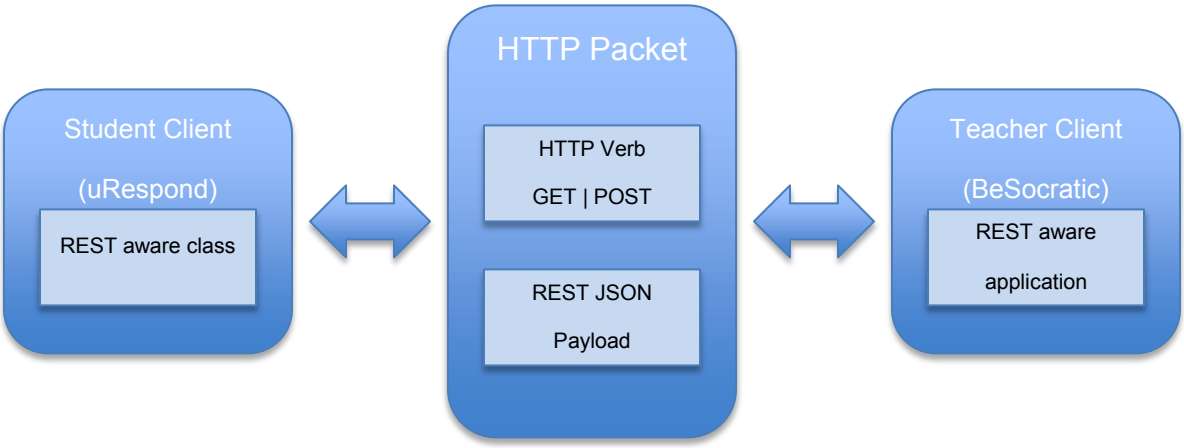


Figure 7: REST Web Service

3.3.4.1 Parsing the JSON payload

Each message exchanged between the teacher client and student client consisted of a nested JSON string. The string needed to be parsed to get the information relevant for uRespond. The string contained three "layers" (See Figure 8).

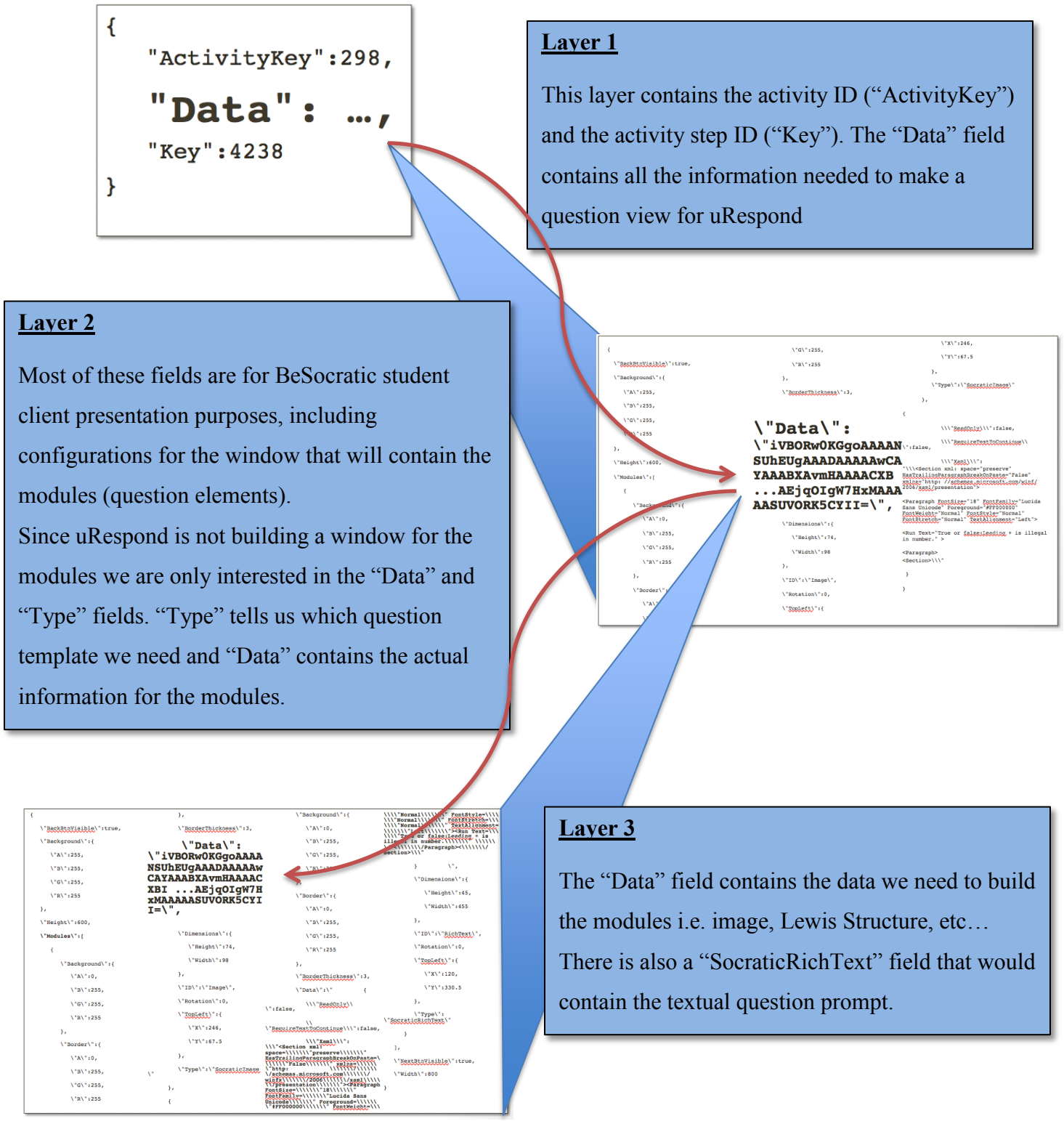


Figure 8: Parsing the JSON string

4: Development

As part of this capstone project, the following classes were developed (see figure 9). We implemented the core functions that were required of the testing prototype: application navigation, login/logout, chemical structure builder view and two additional question types (See appendix 6 for the full project UML diagram and appendix 7 for sample source code).

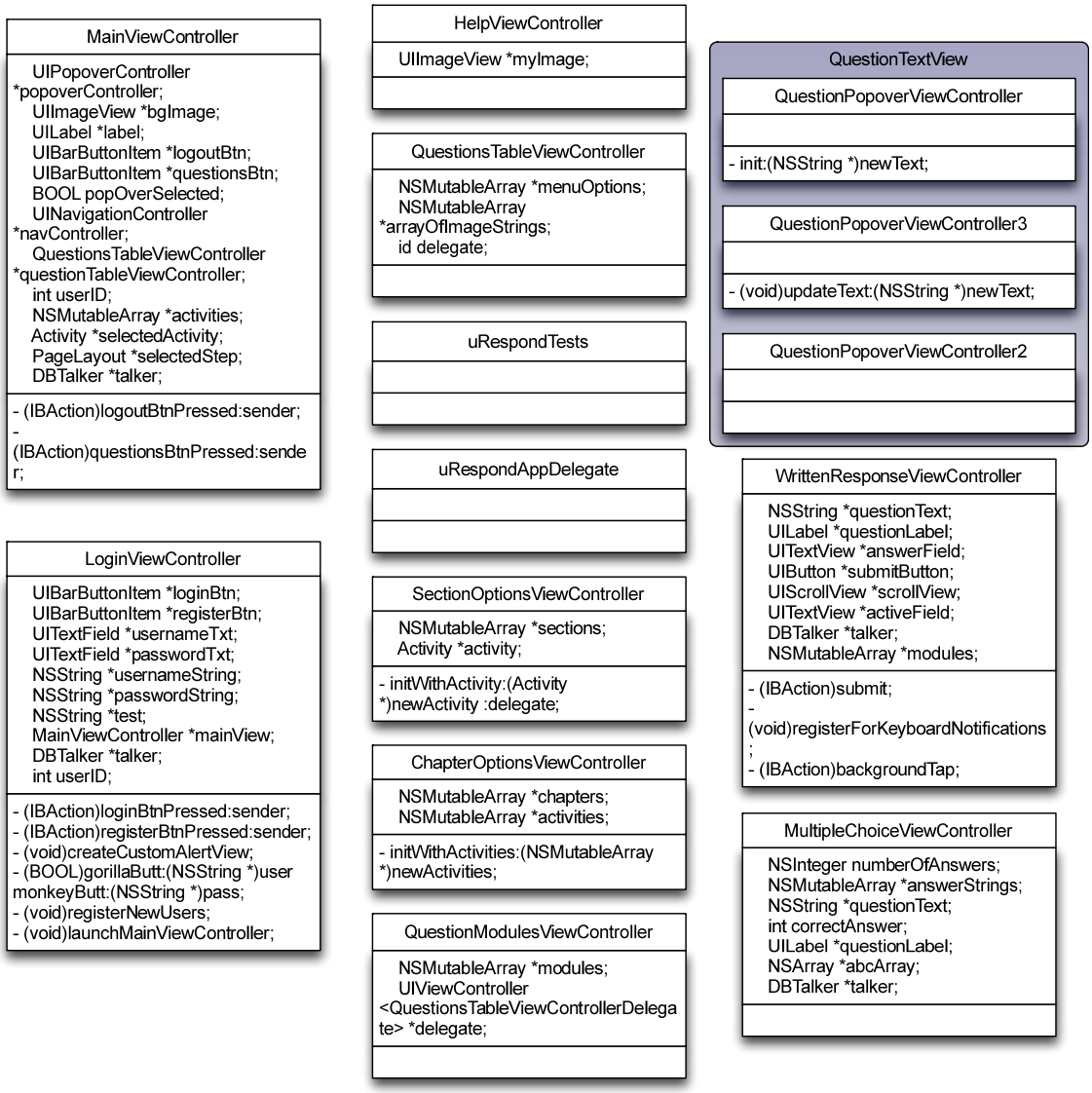


Figure 9: UML Diagram of Classes Developed

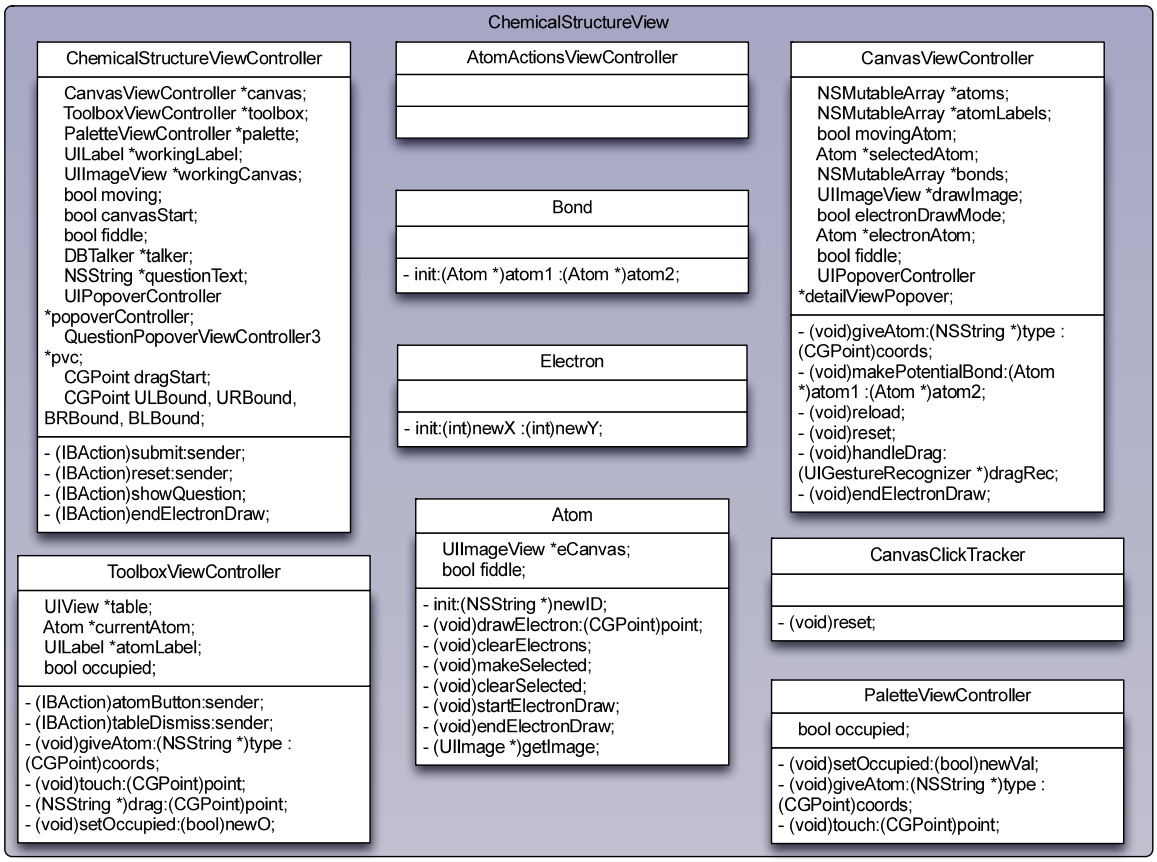


Figure 9 (continued): UML Diagram of Classes Developed

4.1 Development process

We used a spiral life cycle model for the development of uRespond. A working prototype was demonstrated to Dr. Grove at the end of July. Subsequent prototypes were

constructed and demonstrated on weekly iterations. Planning and analysis occurred during the weekly meetings held with Dr. Grove. Members of the UNCW development team tested each iteration of the application.

We decided to use the spiral life cycle model because it allowed us to focus on individual subsystems of uRespond. The incremental development also suited the modular question types featured in this project. In our initial iterations we concentrated on the core subsystems on which other subsystems relied. Progressing in this manner we could add additional subsystems in each subsequent iteration.

Dr. Grove and the UNCW development team looked at the requirements for uRespond and set a priority for components to be coded. The application classes were based on the use cases developed in the functional requirements gathering phase of the project (See Figure 2). We decided to code simpler classes first, leaving more complex components for later in the project. This approach would allow us to get acclimated with the tools and programming language early in the project. The coding priorities were revisited each week to determine the progress of the application

uRespond was developed on iMac machines with twenty-seven inch screens. The software developer kit (SDK) for iOS relies on the Macintosh OSX operating system, which does not run on other hardware. The large screens allowed for the coding tool and the iPad simulator to run in actual size on one view. The project was coded in Apple's XCode, an integrated development environment (IDE) that was designed to create iOS applications. We decided to develop in XCode and Objective-C because Apple strongly recommends developers use these and we were most familiar with these platforms.

Xcode includes many tools helpful for development of iOS applications. Interface Builder is a WYSIWYG (What-You-See-Is-What-You-Get) editor that allows developers to design application user interfaces [53]. There is also an Instrument analysis tool that assists in tracking application performance in areas such as CPU usage and memory [53]. This tool can be used with either the integrated iOS simulator or a connected iOS device [58].

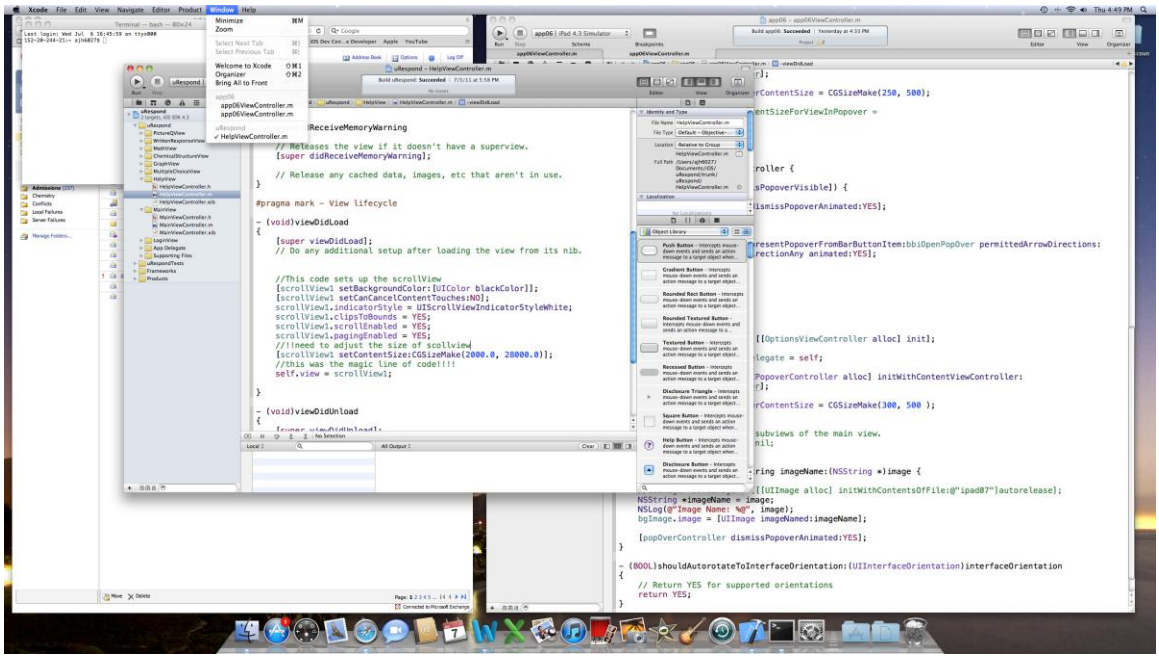


Figure 10: Xcode IDE

4.2 Version control

Version control is important for any software development project, as it offers a central repository of the source code as well as a way of tracking changes to it [54].

Version control systems allow developers to collaborate on a project by giving that person access to the latest version, usually in the IDE they are working in.

We implemented a version control system before development of uRespond began. Xcode supports the system and comes with an integrated interface with the code repository (See Figure 11). With multiple developers working on the project, it was essential that file names and directory structures be consistent for all team members.

For the uRespond project we used Subversion, an open source version control system that manages files and directories, and the changes made to them, over time [54]. Subversion handles the integration of new code introduced to the project and presents conflicts between new and old code to allow the developer to clarify which version to use. Multiple versions or "branches" of the code can be maintained to allow reverting when needed.

The system also served as the communication medium for the team. We made a requirement that a user enter a brief and meaningful comment when he/she committed code to the central repository. These comments let other developers know what changes have been made, who performed them and when.

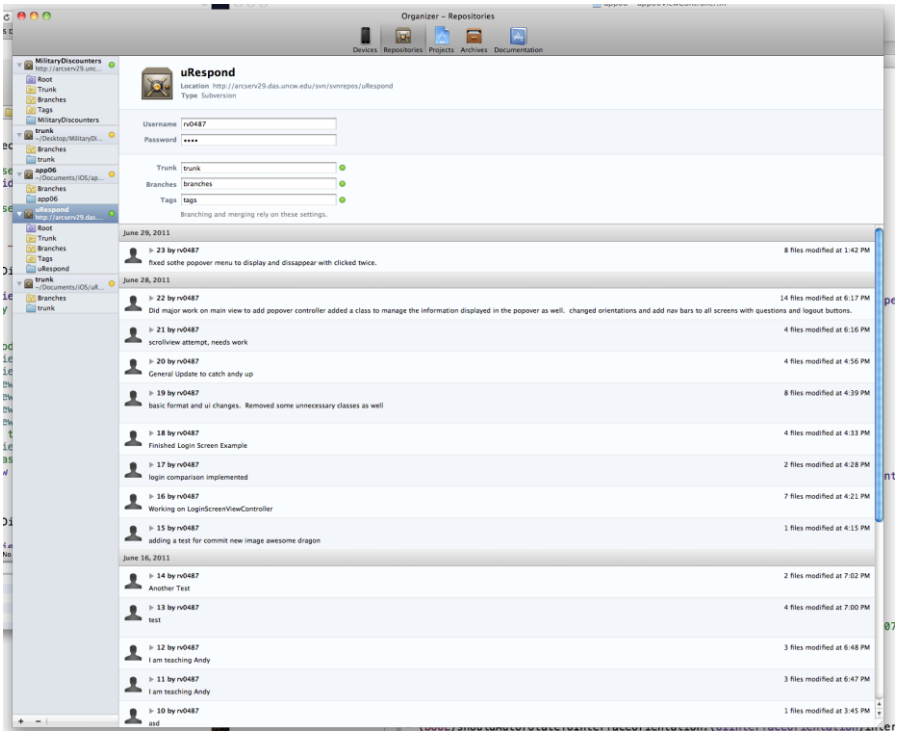


Figure 11: Subversion interface in Xcode

4.3 Distributed teams

As the uRespond application project progressed, it became necessary to coordinate with the Clemson team to integrate uRespond with the BeSocratic system. There are several challenges when dealing with teams that are distributed geographically, including delayed feedback, constrained communication and reduced shared project awareness [55]. We utilized several technologies and modes of communication to overcome these challenges and ensure all members were current.

It is important for distributed teams to spend more time on up-front requirements and written communications in order to build a strong understanding of the project [55]. The Clemson team sent us a copy of their database schema and database files for the

purpose of replicating the database on our server. We shared requirement documents and screen shots of uRespond with the Clemson team. We discussed and documented the database communications as well as the format of the JSON strings that the applications would pass. We agreed to use Dropbox as a central repository for files and code.

Regular communication was essential for the coordination of the integration. The two teams frequently exchanged emails to communicate questions about the project as well as to share files. Weekly videoconferences were held via Skype to present progress, discuss detailed points of the integration and show code we were writing. The two teams also physically met once to discuss the project. The in person meeting was especially helpful as we were able to discuss in greater detail and more efficiently than through videoconferencing or email. We were able to explain concepts more naturally and to illustrate issues by drawing on a white board.

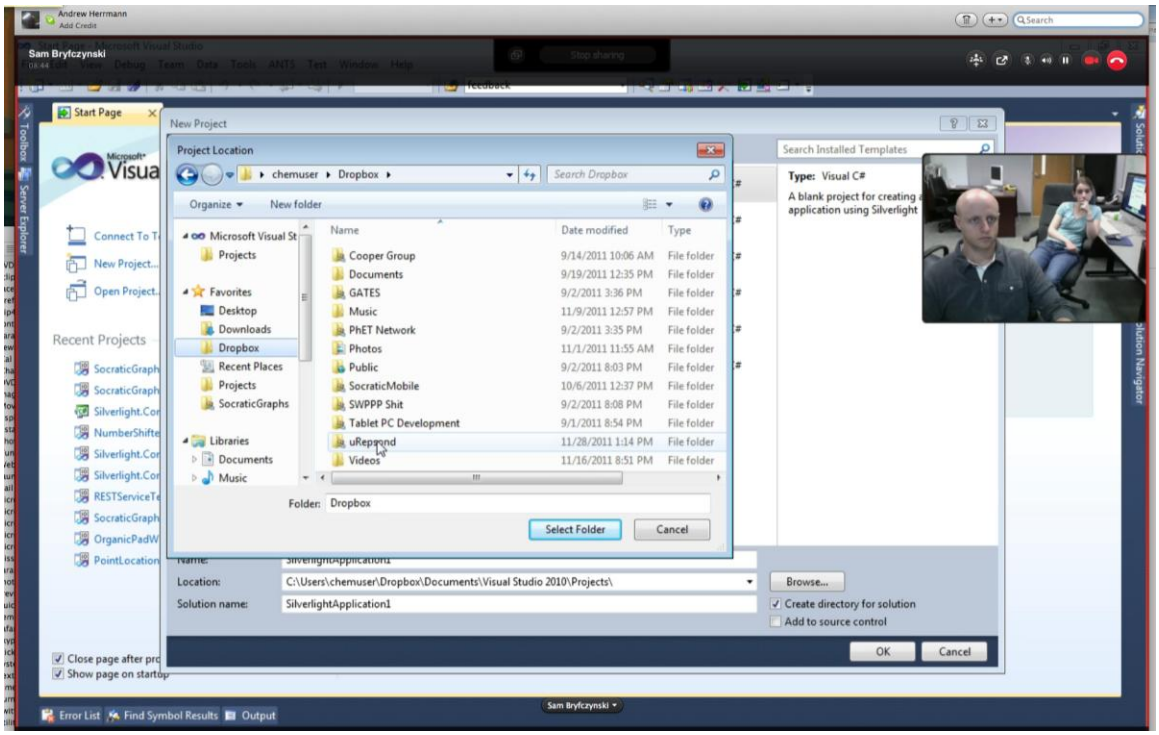


Figure 12: Screen-share during videoconference

4.4 Platform change

In November of 2011, the project leads at Clemson and UNCW reevaluated the direction of the student client. Clemson had been developing a mobile app since the summer of 2011 that replicated much of the functionality of uRespond. The project leads from both institutions felt there was a duplication of efforts and integrating the two applications would be the best way to progress forward.

After reviewing the functionality of the two applications, it was decided that the Clemson mobile application, BeSocratic Mobile, had all the functionality of uRespond with the exceptions of the chemical structure builder and image plotter question types. These two question templates would be integrated into the BeSocratic Mobile application.

The Clemson development team had been developing their app on the Appcelerator's Titanium platform. The Titanium SDK uses a JavaScript API to compile JavaScript code to a different language (Objective-C for iOS, Java for Android) so a developer can deploy an application to multiple targets [56]. There is an interpreter in-place to analyze JavaScript at run-time, which allows the application to be dynamic [56]. Continuing development in Titanium would allow for smoother integration into other platforms in the future.

Pre-written Objective-C code is integrated in a Titanium project by putting that native Objective-C code into a "module" then referencing that from the Titanium project. Although native code can be used in Titanium to write application for iOS, not all native iOS functions are supported. Code has to be refactored to conform to the Titanium

modular format. For example, some of the iOS gesture recognizers were not supported in Ttianium, pictures are reference differently, and user interface files cannot be packaged into the modules.

Despite these challenges, we were successful in integrating code into BeSocratic Mobile.

5 Testing

We conducted a usability test on April 10th, 2012, during a review session with Dr. Grove's general chemistry class. The purpose of the test was to gather users' impression of the design and functionality of the application. The survey responders consisted of twenty-nine students in the class that volunteered to evaluate the system. We felt these opinions were important because similar students would be the end users when the application is deployed later. The questions asked revolved around the user interface, glitches, responsiveness, ease of use and over impression of the system (see Appendix 7). This portion of the project was reviewed and approved by UNCW's IRB.

Generally, we found students had a positive experience using the application. An overwhelming majority of users agreed that the app was easy to navigate and the layout and font were appropriate (See figure 13). Overall respondents highly rated the programs responsiveness and ease of use (see figure 14). One student wrote, "I really think this is a fun/effective way to learn & teach". Another student expressed, "Personally, I found the app to be easy to use, and fun."

One issue we discovered was that a number of students faced a bug, which caused the system to break. Students were unable to dismiss the periodic table in the chemical

structure builder view. We are currently investigating the cause of this. Several students offered suggestions on improving the system. “Questions need to be bigger, answers smaller”, and “It would be nice if the subscripts were actually below the element [in the text of questions]” were two recommendations were received.

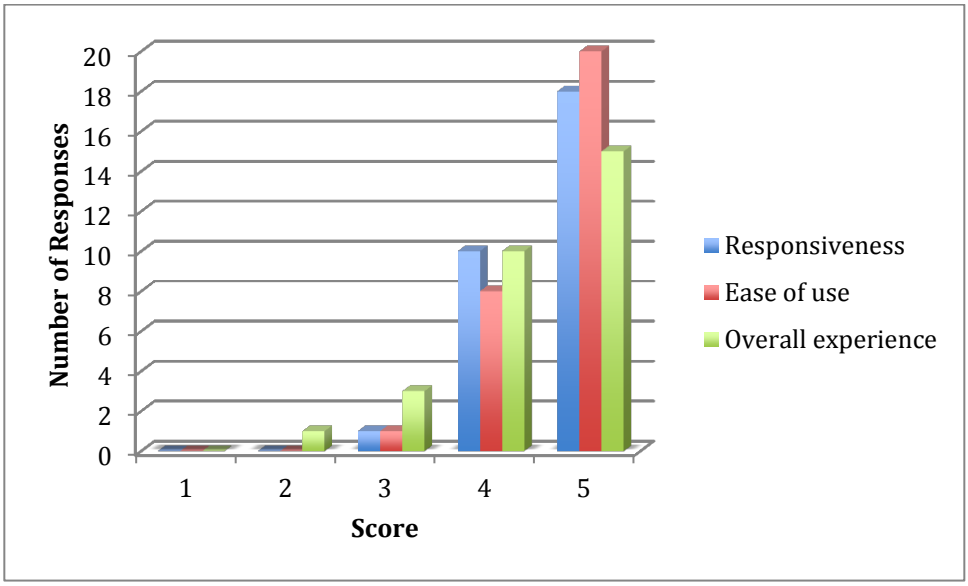


Figure 13: Student Survey Results

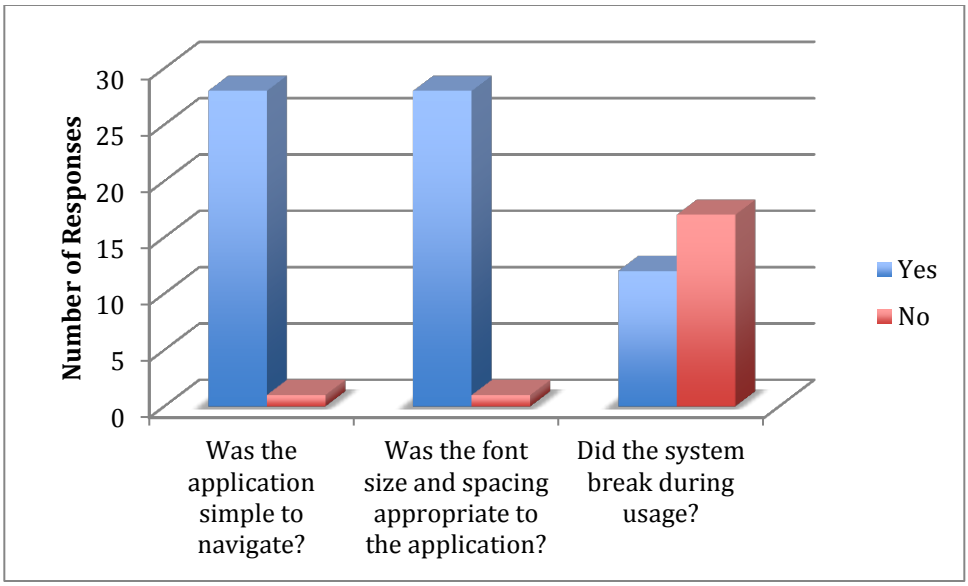


Figure 14: Student Survey Results (Continued)

6 Future Work

6.1 Deployment

The NSF grant proposal for uRespond specifies a three-year schedule. Several members of the development teams at Clemson and UNCW will continue to develop the system and deploy it in phases in the fall of 2012. All of the core functionality has been developed and only minor additions are expected from this point.

The app will initially be used for the general chemistry course at UNCW. The Chemistry Department has purchased 20 iPads for use in the classroom and several dozen more will be ordered soon. These iPads will need to be provisioned for testing through Apple's website. The app can then be tested more rigorously in classroom settings with multiple users over the course of a semester.

The app will need to be submitted to Apple's approval process. This process can take several weeks in a best-case scenario. If the app is rejected changes to the program must be made and the app needs to be re-submitted.

6.2 Data Visualization

The teacher client will have the option of looking at individual responses or a visualization of all of the responses. Student answers need to be grouped and presented in a meaningful way for uRespond to be useful for professors in a large auditorium setting. We have looked into ways of grouping and clustering data to create a visual representation of student responses.

We have explored a variety of graphs from simple bar charts to more complex concept maps. The representations need to allow the professor to understand quickly what the students know as a class. These visualizations need to be tested in classroom situations to find the most beneficial method of presenting the data for instructors.

6.2 Platform Expansion

The mobile device market is young and dynamic. Committing to one platform now could restrict the accessibility of the application later. The iPad is currently the most prevalent tablet device but competing platforms such as the Kindle Fire are growing in popularity. The much anticipated Windows 8 OS is to be released soon and it will offer users another platform on an assortment of devices.

Moving the application to the Appcelerator Titanium platform will allow for integration with Android and HTML 5. Other platform will be considered as they emerge.

6.3 Handwriting Recognition

It is conceivable that even more sophisticated forms of analysis could be performed on student responses if the student client could recognize the letters, numbers or symbols that students drew. Handwriting recognition would permit students to continue submitting free-form response while reducing variation in answers simplifying the grouping process for visualization and analysis.

6.4 Student Activity Archive

Students could potentially use uRespond as a study guide for tests. Adding an archive of activities would allow students to review the materials covered in class and prepare for upcoming exams. The archive would store not only the questions asked but also the individual responses allowing students to become aware of their learning process.

6.5 Auto-Grading

One particularly beneficial feature for instructors would be automatic evaluation of student responses. Knowing the number of students that submitted a correct or incorrect answer would be valuable to a professor during a lecture. This information

could also be used in creating more detailed visualizations in the teacher client. The system could be used to administer and grade quizzes as well.

One disadvantage with Auto-grading is that a correct answer must be given to compare against, limiting the system's usefulness in creating activities on the fly.

6.6 Application Icons

The graphics developed with Ned Irvine and the graphic design students are not currently used in the teacher or student client. With the integration into BeSocratic Mobile, the uRespond identity was lost. In the immediate future, Dr. Grove plans to replace icons in the teacher client with visuals created for this project. Eventually, the images will be incorporated into the student client as well. There are plans to rebrand BeSocratic with the graphics developed by the UNCW students, in the long term.

7 Lessons Learned

7.1 Challenges with developing software

We found Objective-C to be one of the more difficult languages we have worked with. The rules for memory management in the language take some practice to master. It is a particularly verbose language, but fortunately Xcode helps with its auto-completion feature.

Collaborative tools and increased communication are vital when working with people remotely. We used tools to screen-share or see demos on a regular basis. Despite

the tools the distance made communicating more effort intensive. It was difficult to convey ideas at times and there was lag time when communicating by email. We were fortunate to avoid some issues facing distributed teams such as time zone, language barrier, and the difficulties of developing software versus just integrating.

7.2 Handling changes in development environment

Keeping up with developing for the current version of mobile platforms can be difficult. We upgraded the iPads we used to test uRespond to iOS 5.0 on October 26, 2011. Afterward we were unable to build the application to the iPad device. We learned that the version of Xcode needed to be updated to 4.2. However, the version of the OSX were using, Snow Leopard did not support Xcode 4.2. We purchased the most current version of OSX, Lion, installed it and upgraded to XCode 4.2 on November 2, 2011.

Upon upgrading to XCode 4.2 the Subversion integration ceased working. We were still able to submit and update code but we needed to perform these tasks in the terminal. We were unable to fix the Subversion interface in Xcode for the rest of the project.

7.3 Research vs. Industry

Developing software in a research environment can be different from developing in an industry setting. Industry often puts an emphasis on process and documentation where research is concerned with the results. While we followed the standard process for

the software development life cycle, we needed to change out methodology to adapt to the needs of the project.

Considerable effort went into documenting and clarifying the scope of the uRespond project. The NSF grant proposal laid down a basic framework for the application, specifying the iPad platform and some of the key functionality. We spent several weeks documenting the rest of the user and technical requirements through interviews and research. When we reached a clear vision of the project, we drafted a scope document, which all parties signed.

With well-defined goals and objectives set, we began coding the application. The development was going smoothly and we were progressing ahead of schedule. We communicated frequently with Dr. Grove and the Clemson development team about the development of the application.

The uRespond project was initiated before the BeSocratic project in 2009. Dr. Grove had limited involvement in developing the BeSocratic project and little knowledge of its direction. Initially, the BeSocratic Mobile and uRespond projects progressed in parallel, often with the two teams ignorant of each other's work. Through discussions between the two teams, it became increasingly clear that our project was similar to the one being developed by the Clemson team. The project leads decided to combine the projects.

Unfortunately, the project stalled for two months as it was unclear which platform we would use going forward. Although BeSocratic Mobile's project goal was different from uRespond the functionality was comparable. The purpose of both applications was student learning through free-form input. uRespond focused on providing professors with

live feedback during class whereas BeSocratic Mobile captured how students interact with the modules and relied on post-analysis processing to examine which strategies were effective or ineffective for teaching. As discussed earlier, the project leads deliberated and chose the Titanium platform.

Clear hierarchies are typical in business settings but project member relationships in research can be ambiguous. This led to some confusion in roles played by members between the two universities and the overall decision-making process. The flow of information sometimes came from our project lead and other times from developers on the other team. There was no authoritative voice on the technical partnership leading to different levels of project rigor and process documentation.

Although the two teams decided at the project's inception that UNCW would handle the development of the student client, cooperating on both the teacher client and student client seemed to be the better method of developing the system. Often a similar applications developed in industry would be considered competitors but in research it's possible to collaborate to combine the two, offering a better outcome for all involved.

8 Conclusions

My contributions to this project are as follows:

- documented requirements
- installed and configured Subversion
- coded or contributed to all classes shown in figure 9
- provisioned devices for testing

- managed communications with Clemson Team
- headed collaboration with graphic design class
- directed application testing
- prepared IRB review forms
- installed testing database on UNCW server

Several aspects of this project went particularly well. Some of the tools we used helped it go more smoothly. I would recommend Subversion, or any other version control software, to anyone working on a project with multiple developers. We had a developer leave several weeks into the project. With Subversion, the next developer came in and all the code and update comments were available to allow her to quickly learn the project. Skype also helped with communication with the Clemson developers. The screen share functionality of the tool allowed us to show and view code in real time as well as illustrate comments with drawing tools. The weekly status meeting with our project was especially beneficial to receive feedback on progress and clarification on upcoming work. Another positive point of the project was the team cohesiveness. Good teamwork and a total buy-in into the project helped us work well together and avoid conflicts that can arise in teams.

Some things didn't work well with this project. I installed Trac to help manage bugs and monitor progress with the project. Our project lead found our weekly status meetings more useful for monitoring progress and we used a white board to track bugs. I think the tool would be more useful to a project manager working with three or more developers. The tool wasn't suitable for our purposes and was rarely used. Unclear goals

also hindered the project. By beginning development before we fully understood the goal we ran into issues later in the project and ultimately did not utilize much of the code we wrote for uRespond. Repurposing the code we wrote in Objective-C for Titanium didn't go well either. We had to refactor the code to fit into Titanium's platform. Debugging was awkward and bugs were sometime difficult to reproduce. Rewriting the code in JavaScript may have been the better way of integrating the projects

Knowing what I know now I would do the project differently. First of all I would communicate more with the Clemson team. Secondly, I would gather both teams' requirements and design a system that satisfies all parties' needs. Finally, developing for a single platform would be best for the initial release of the system.

Mobile cross-platform technologies are new, buggy and, from our experiences, difficult to work with. iOS and Android are too different in their design ideas to be covered successfully by a common API. Android would be the best platform to use because it is easier to create an open-source project on this platform compared to iOS. Android applications can be installed on directly on Android devices easily and for free. A paid developer's license is required to submit apps to Apple's App Store for distribution or to provision devices for testing apps. It would be simpler to set up an online community of uRespond Android developers that could straightforwardly download the code, modify it, and test it on their own device without having to go through a third party. Additionally, Android tablets are generally cheaper, have the same functionality as iPads and come in a variety of hardware configurations.

After deploying uRespond and proving its benefits in the classroom, the app could be introduced to Android developers interested in open source projects. This would

increase the apps support, future proof the project and add improvements to the program. If the project gained momentum other platforms could be developed either through the open source community or through hiring developers with money from revenue streams created by the apps popularity.

The future potential of this project is great. This system's value is two-fold: it helps students learn and it helps instructors teach. As more professors and students see this application in use more institutions will want to adopt it. It is inevitable that more platforms will need to be developed for the student client as the apps popularity grows. For the reasons I mentioned previously, a native Android app would be better than refactoring code in Titanium for Android deployment. We spent equal amounts of time integrating our Objective-c code with Titanium as we did with initially developing it. An open-source Android development could lead to a wider community involvement in the project and additional platform development.

Acknowledgements:

- Dr. Nathaniel Grove
- Rebecca Brown
- Ned Irvine
- Sam Bryfcynski
- Josiah Hester
- Ricardo Valea

References:

1. Bodzash, D. (2010, September). America on 'perilous path' as students fall further behind in science, math. *Examiner.com*. Retrieved from http://www.examiner.com/article/america-on-perilous-path-as-students-fall-further-behind-science-math.

2. Terrell, N. (2007). STEM Occupations. *Occupational Outlook Quarterly*. 26-33.

3. Kuenzi, J. (2008). Science, Technology, Engineering, and Mathematics (STEM) Education: Background, Federal Policy, and Legislative Action. *CRS Report for Congress*.

4. U.S. Department of Labor, Employment and Training Administration by Jobs for the Future. (April 2007). The STEM Workforce Challenge: the Role of the Public Workforce System in a National Solution for a Competitive Science, Technology, Engineering, and Mathematics (STEM) Workforce.

5. Dillon, S. (2010, December 7). Top Test Scores From Shanghai Stun Educators. *The New York Times*. pp. A1

6. Carnevale, P., Smith, N., & Strohl, J. (2010) Help Wanted: Projections of Jobs and Education Requirements through 2018. *The Georgetown University Center on Education and the Workforce*.

7. Members of 2005 Rising Above the Gathering Storm Committee. (2010) Rising Above the Gathering Storm, Revisited: Rapidly Approaching Category 5. *The*

National Academies Press.

8. ACT. (2006). Developing the STEM Education Pipeline. 2006.

9. Boundaoui, Assia. (May 2011) Why would-be engineers end up as English majors. *CNN.com*. http://edition.cnn.com/2011/US/05/17/education.stem.graduation/index.html

10. Bodner, G. M. (1986). *Journal of Chemical Education.* 63, 873.

11. Bretz, S. L. (2001). *Journal of Chemical Education.* 78, 1107.

12. Ausubel, D. P. (1968). *Education Psychology: A Cognitive View*. New York: Holt, Rinehart, and Winston.

13. Novak, J. D. (1977). *A Theory of Education*. Ithaca: Cornell University.

14. Novak, J. D. (1998). *Learning, Creating, and Using Knowledge: ConceptMaps as Facilitative Tools in Schools and Corporations*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

15. Cooper, M. (1995). Cooperative learning: an approach for large enrollment classes. *J. Chem. Educ ,* (72), 162-164.

16. Campbell, D. & Smith, K. (1997). *New paradigms for college teaching*. Edina, MN: International Book Co.

17. Kogut, L. (1997). Using cooperative learning to enhance performance in general Chemistry. *Journal of Chemical Education* , (74), 720-722.

18. Nurrenbern, S. & Robinson, W. (1997). Cooperative learning: a bibliography. *J. Chem. Educ.* ,(74), 623-624.

19. Farrell, J.; Moog, R.; and Spencer, J. (1999). A guided inquiry general chemistry Course. *Journal of Chemical Education*, (76), 570-574.

20. Paulson, D. (1999). Active learning and cooperative learning in the organic chemistry lecture class. *Journal of Chemical Education*, (76), 1136-1140.

21. Hanson, D. & Wolfskill, T. (2000). Process workshops – a new model for instruction. *J. Chem. Educ.*, (77), 120-130.

22. Eybe, H. and Schmidt, H. (2004). Group discussions as a tool for investigating students' concepts. *Chem. Educ. Res. Pract*., (5), 265-280.

23. Kennedy, G. E. and Cutts, Q. I. (2005) The association between students' use of an electronic voting system and their learning outcomes. *J. Comp. Assist. Learn*. , 21, 260-268.

24. Fies, C. and Marshall, J. (2006). The C3 framework: Evaluating classroom response system interactions in university classrooms. Journal of Science Education and Technology 2008, 17, 483-499.

25. Fies, C. and Marshall, J.(2006). Classroom response systems: A review of the literature. Journal of Science Education and Technology, 15, 101-109.

26 Bullock, D. W.; LaBella, V. P.; Clingan, T.; Ding, Z.; Stewart, G.; and Thibado, P. M. (2002). Enhancing the student-instructor interaction frequency. *Phys. Teach*., 40, 30-36.

27. Burnstein, R. A. and Lederman, L. M. (2001). Using wireless keypads in lecture classes Phys. Teach., 39, 8-11.

28. Davies, S. M. (2003). Observations in classrooms using a network of handheld devices. J. *Comp. Assist. Learn*., 19, 298-307.

29. Dufresne, R. J.; Wenk, L.; Mestre, J. P.; Gerace, W. J.; and Leonard, W. J. (1996). Classtalk: A classroom communication system for active learning. *J. Comp. High. Educ.*, 7, 3-47.

30. Ganger, A. C. and Jackson, M. (2003). Wireless handheld computers in the preclinical undergraduate curriculum. *Med. Educ. Online*, 8.

31. Nicol, D. J. and Boyle, J. T. (2003). Peer instruction versus class-wide discussion in large classes: A comparison of two interactive methods in the wired classroom. *Stud. High. Educ*., 28, 458-473.

32. Paschal, C. B. (2002). Formative assessment in physiology teaching using a wireless classroom communication system. *Adv. Phys. Educ*., 26, 299-308.

33. Poulis, J.; Massen, C.; Robens, E.; and Gilbert, M. (1998). Physics lecturing with audiencepaced feedback. *Amer. J. Phys*., 66, 439-441.

34. Reay, N. W.; Bao, L.; Pengfei, L.; Warnakulasooriya, R.; and Baugh, G. (2005) Toward an effective use of voting machines in physics lectures. Amer. J. Phys., 73, 554-558.

35. TurningPoint Student Response System (2004). http://campustechnology.com/articles/2004/08/turningpoint-student-response-system.aspx.

36. IUPAC. (2011). Gold Book. http://goldbook.iupac.org/L03513.html.

37. Keller, J. (2011). As the web goes mobile, colleges fail to keep up. *Educause*. http://chronicle.com/article/Colleges-Search-for-Their/126016/

38. Trifonova, A. and Ronchetti, M.. (2003).Where is Mobile Learning Going?. *In Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2003*, 1794-1801.

39. Wains, S. and Mahmood, W. (2008). Integrating m-learning with e-learning. *In Proceedings of the 9th ACM SIGITE conference on Information technology education (SIGITE '08)*. ACM, New York, NY: ACM, 31-38.

40. Yong Liu, Y., Hu, F. and Li, H. (2009). Understanding learners' perspectives on m-learning: results from a survey. *In Proceedings of the 2009 Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship (EATIS '09)*. ACM, New York, NY: ACM, 6, 3.

41. Martin,S., M. Gil,R.; Cristobal,E; Diaz,G.; Castro, M.; Peire,J.; Milev,M.; Mileva,N. (2009). Middleware for the development of context-aware applications inside m-Learning: Connecting e-learning to the mobile world. *Computing in the Global Information Technology, 2009*. *Fourth International Multi-Conference IEEE*, Cannes, La Bocca. P. 217-222.

42. Ana Maria Fermoso Garcia, Alberto Pedreo Esteban, Reusing (2011). *Educational Contents in M-learning. Advanced Learning Technologies, IEEE International Conference* on, pp. 448-449,

43. Georgiev, Tsvetozar, Evgenia Georgieva, and Angel Smrikarov. (2004). M-Learning - a New Stage of E-Learning. 1-5.

44. Carole Salis, Fabrizio Murgia, Andrea Mameli (2007). *Proceedings of the 2007 Euro American conference on Telematics and Information Systems, EATIS 2007*, Pages: 32-32.

45. Maria Virvou and Eythimios Alepis. (January 2005). Mobile educational features in authoring tools for personalised tutoring. *Comput. Educ*. 44, 1, 53-68.

46. Collins, D., Deck, A., & McCrickard, M. (2008). Computer Aided Instruction: A Study of Student Evaluations and Academic Performance. *Journal of College Teaching & Learning*, 5.11, 49-58

47. Mikhail Morozov, Andrey Tanakov, Alexey Gerasimov, Dmitry Bystrov, Eduard Cvirco, (2004). Virtual Chemistry Laboratory for School Education. *Advanced Learning Technologies, IEEE International Conference on, pp. 605-608, Fourth IEEE International Conference on Advanced Learning Technologies*

(ICALT'04),.

48. Fjeld, M, and B M Voegtli. (2002). Augmented Chemistry: an interactive educational workbench. *Proceedings International Symposium on Mixed and Augmented Reality* Mmi: 259-321.

49. Melanie M. Cooper, Nathaniel P. Grove, Roy Pargas, Sam P. Bryfczynski and Todd Gatlin. (2009). *OrganicPad*: an interactive freehand drawing application for drawing Lewis structures and the development of skills in organic chemistry. *Chem. Educ. Res. Pract*., 10, 296-301.

50. *ChemDraw*, (2008), CambridgeSoft: Cambridge, MA.

51. Melanie M. Cooper, Nathaniel P. Grove, Roy Pargas. (2009). iRespond: iPhone/iPod Touch as interactive personal response system. *NSF Proposal*.

52. Alessi, P. (2011). iPhone and iPad Database Application Programming. Indianapolis: Wiley Publishing, 375.

53. Xcode. (2011). https://developer.apple.com/technologies/tools/

54. Yeates, Stuart. (2012 March 18). What is version control? Why is it important for due diligence?. *OSS Watch*.

55. Guck, Randy. (2012, March 8). Managing Distributed Software Development. *Sticky Minds*. http://www.stickyminds.com/sitewide.asp?ObjectId=6002&Function=edetail&ObjectType=ART

56. Whinnery, K. (2010). Titanium Guides Project: JS Environment. *Appcelerator Developer Blog*. http://developer.appcelerator.com/blog/2010/12/titanium-guides-project-js-environment.html

57. Hall, R. H.; Collier, H. L.; Thomas, M. L.; and Hilgers, M. G. (2005). A student response system for increasing engagement, motivation, and learning in high enrollment lectures. *Proceedings of the 11th Americas Conference on Information Systems*, Omaha, NE, 1-7.

58. King, D. B. & Joshi, S. (2008). Gender differences in the use and effectiveness of personal response devices. *Journal of Science Education and Technology*, (17), 544-552.

59. Woelk, K. (2008). Optimizing the use of personal response devices (clickers) in large enrollment introductory courses. Journal of Chemical Education, 85, 1400-1405.

60. MacArthur, J. R. and Jones, L. L. (2008). A review of literature reports of clickers

applicable to college chemistry classrooms. *Chem. Educ. Res. Pract*., 9, 187-195

61. Audience Response Systems: Turning Technologies. http://www.turningtechnologies.com/

62. i>Clicker. (2011). http://www.iclicker.com/dnn/.

63. eInstruction. (2011). http:/www.einstruction.com/

64. H-ITT Classroom Response System. (2011). http:/www.h-itt.com/

65. Heward, W. L. (2003). Guided notes – improving the effectiveness of your lectures. *Factsheet published by the US Department of Education*.

66. Lammers, W. J. and Murphy, J. J. (2002). A profile of teaching techniques used in the university classroom: A descriptive profile of a US public university. *Act. Learn. High. Educ.*, 3, 54-67.

67. Huxham, M. (2005). Learning in lectures – do 'interactive windows' help? *Act. Learn. High. Educ*., 6, 17-31.

68. Report to the President on ensuring American leadership in advanced manufacturing. (2011). *Executive Office of the President, President's Council of Advisors on Science and Technology*.

Appendicies

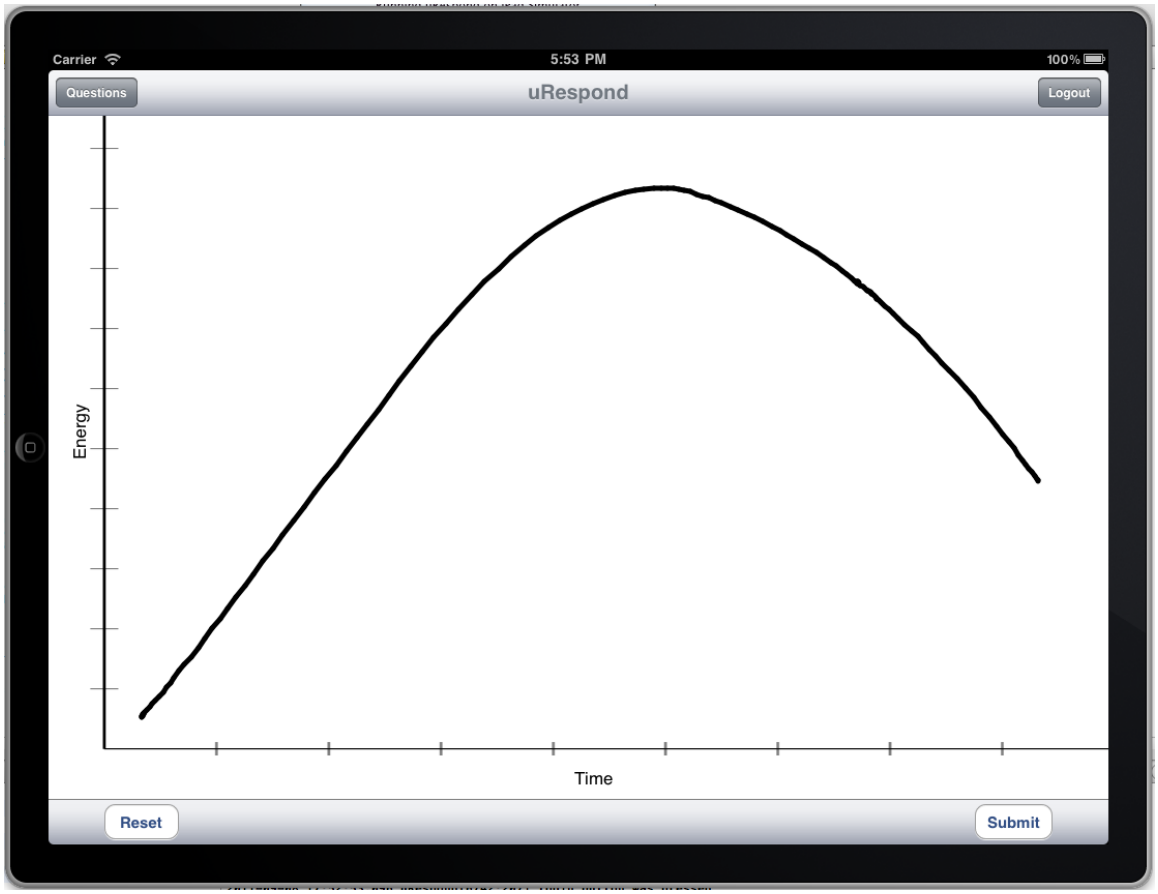
Appendix 1: Project Timeline in NSF Proposal

Timetable

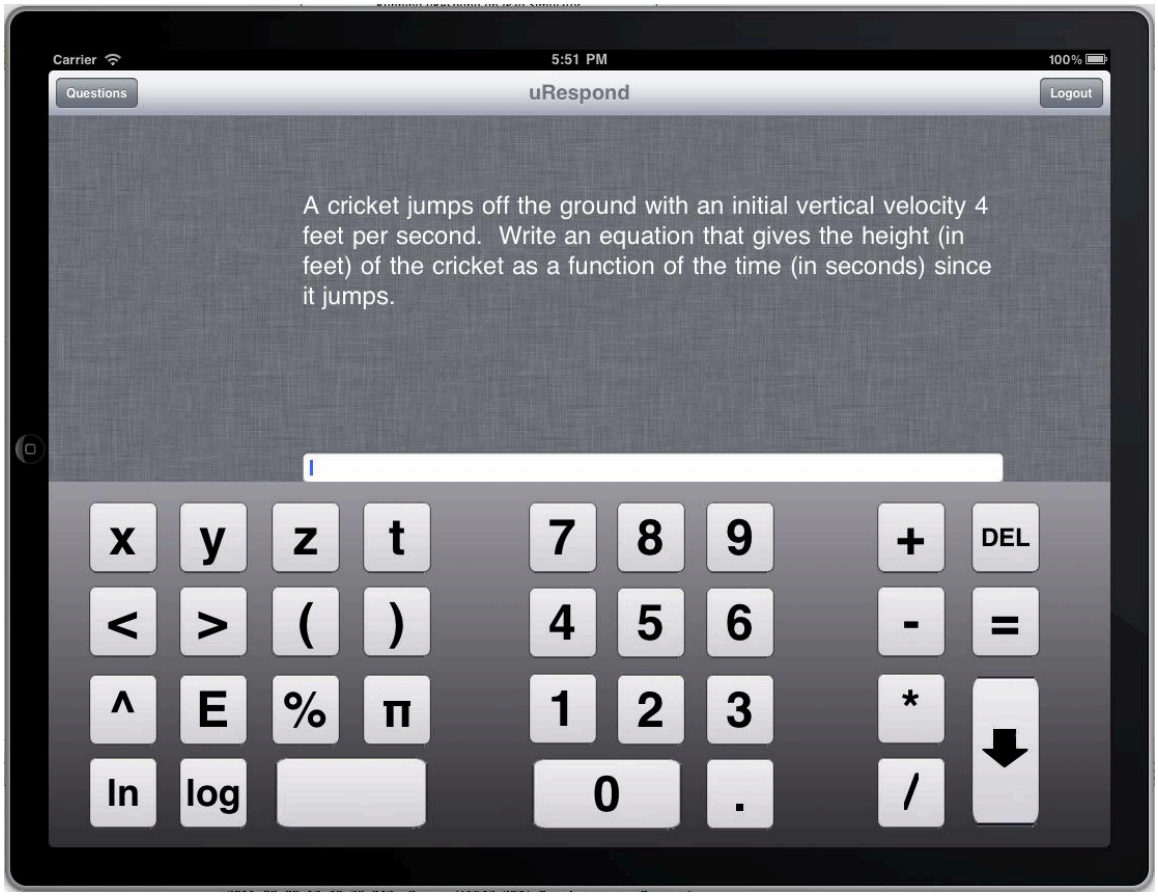
January-December	Timeline of Activities
2010 JFMAMJJASOND	Software Development
xxxxxxxxxxxx xxxxx xxx	1. Software development and refinement 2. Begin pilot-testing with students 3. Develop 1st year intermediate report
2011 JFMAMJJASOND	Software Refinement and Development of Organic Materials
xxxxxxxxxxxx xxxxxxxxxxx	1. Software development and refinement 2. Continued student testing
xxxxxxx xxx	2. Development of organic chemistry-themed *iRespond* questions 3. Develop 2nd year intermediate report
2012 JFMAMJJASOND	Software Refinement and Development and Refinement of Organic Materials
xxxxxxxxx xxxxxxxxx xxx xxx xxxxxx	1. Software Refinement 2. Development/Refinement of organic chemistry-themed *iRespond* questions 3. *iRespond* Workshop 4. 3rd year intermediate report 5. Final report

Appendix 2: Question Type GUIs

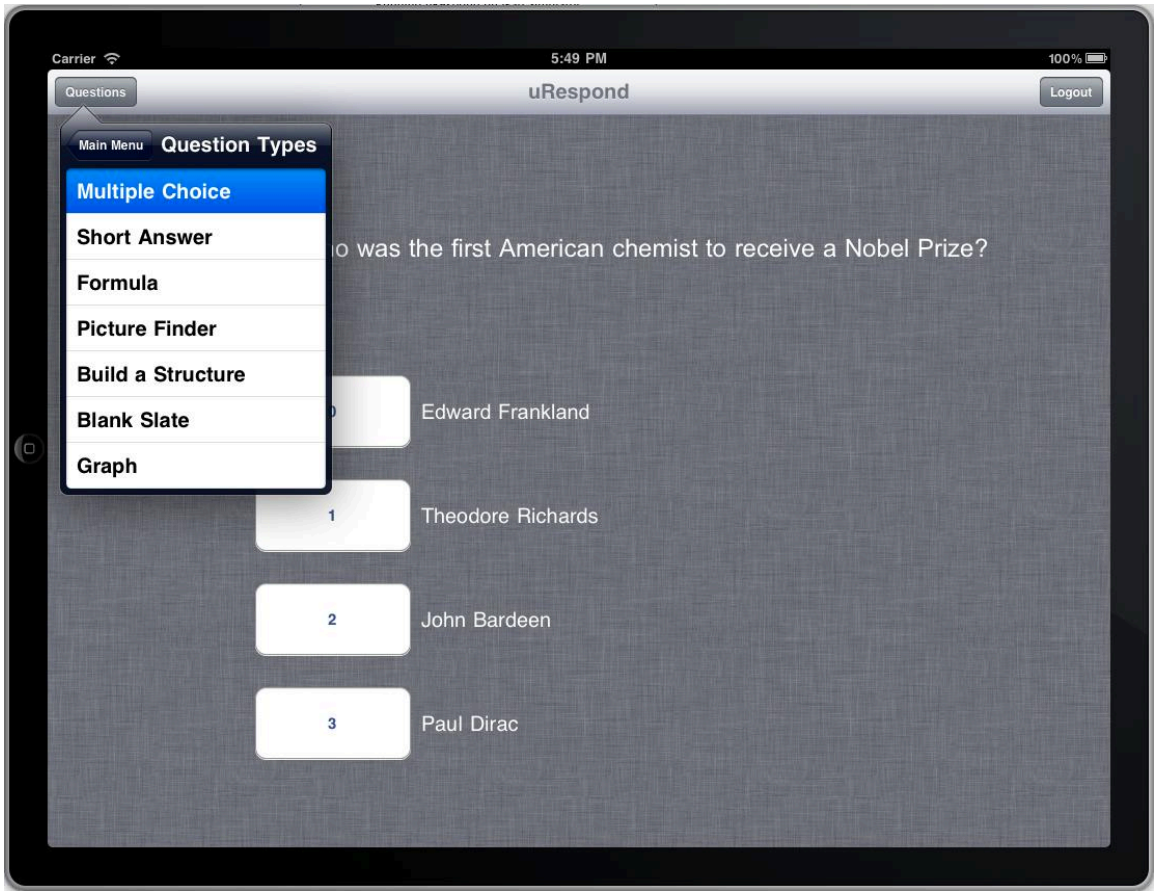
Image Plotter	An image on which students place a dot.
Receives	String – question prompt Blob - image
Sends	String – x y coordinates of dot
Functions	Red dot can be moved around the screen Reset button puts dot back at horizontal and vertical center Submit button sends x y coordinates to DB Prompt button will bring up question text in a new window (To Do)



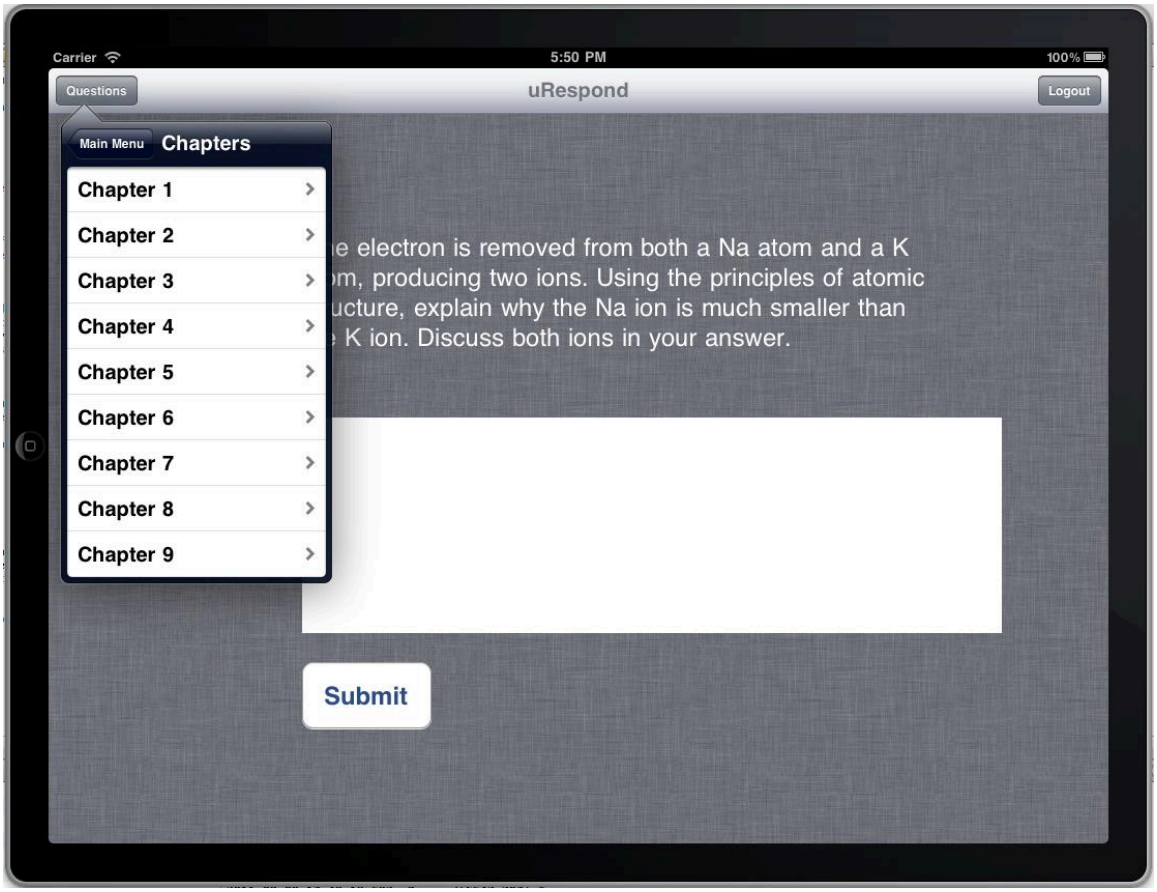
Graph	An X-Y graph that allows students to draw lines or dots.
Receives	String – question prompt
Sends	blob – image of graph drawn
Functions	Area above the x axis and to the right of the y axis is an “ink canvas” in which students can freely draw a line Reset button clears the “ink canvas” Submit button sends image to DB Prompt button will bring up question text in a new window (To Do)



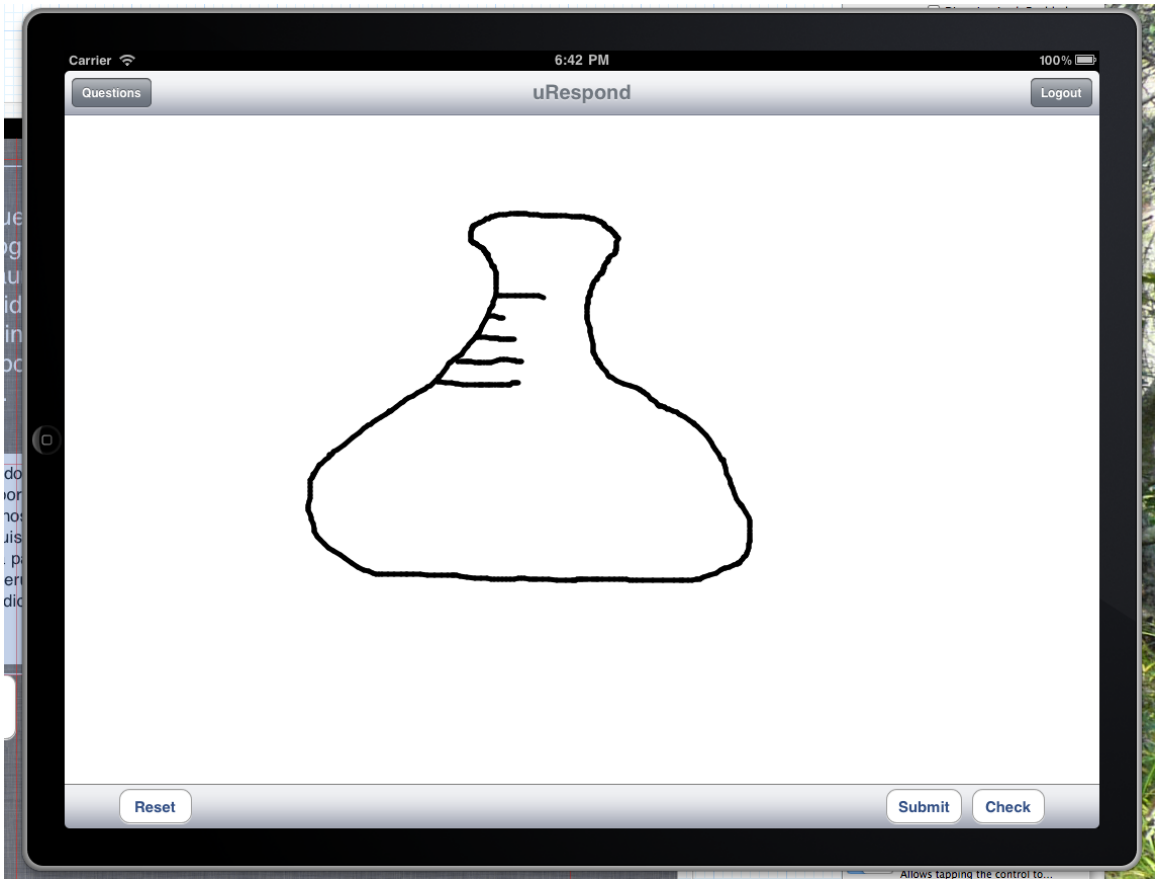
Math/Formula	An input box in which students can put a formula or other math notations.
Receives	String – question prompt
Sends	String – formula answer
Functions	Touching in the answer text box calls the custom keyboard. Buttons on the keyboard do not calculate but simply put the character in the text box (except DEL which deletes a character and the down arrow which dismisses the keyboard) Submit button sends string answer to DB



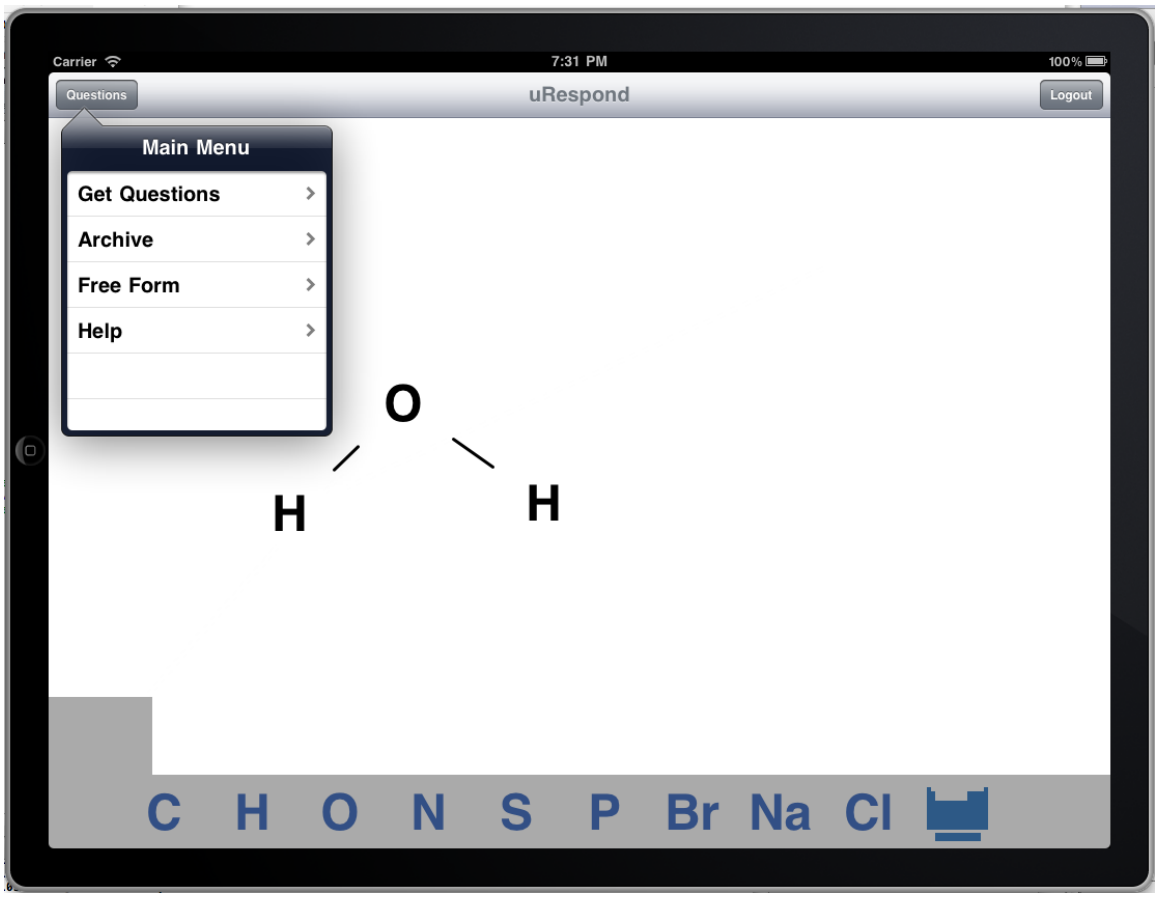
"Multiple Choice"	A simple multiple choice template similar to other clicker systems.
Receives	String – question prompt String – possible answers
Sends	String – answer
Functions	Can have between 2 and 6 possible answers Pressing a button sends the answer to the DB



Short Answer	An input box in which students can enter single word or short essay responses.
Receives	String – question prompt
Sends	String – Answer
Functions	Selecting the answer text box calls the iPad keyboard The question prompt and answer text box are scrollable, being able to move up and down Submit button sends text in answer box



Blank Slate	A white canvas on which students can draw freely.
Receives	String – question prompt
Sends	Blob – image drawn on canvas
Functions	Reset button clears the “ink canvas” Submit button sends blob image to DB Prompt button will bring up question text in a new window (To Do)



Chemical Structure	An interface on which users can create Lewis Structures.
Receives	String – question prompt
Sends	Blob – image of chemical structure
Functions	Students select an element from the pallet, which puts it in the staging area on the left. The periodic table icon calls a view with the periodic table. When a user selects an element it is sent to the staging area. Students will drag the element from the staging area to the canvas to activate it. Once an element is in the staging area can tap between atoms to create bonds. Reset button clears the canvas Submit button blob image to DB Prompt button will bring up question text in a new window (To Do)

Appendix 3: uRespond Graphics

Blank Slate

Math/ Formula

Image Plotter

Short Answer

Chemical Structure

Multiple Choice

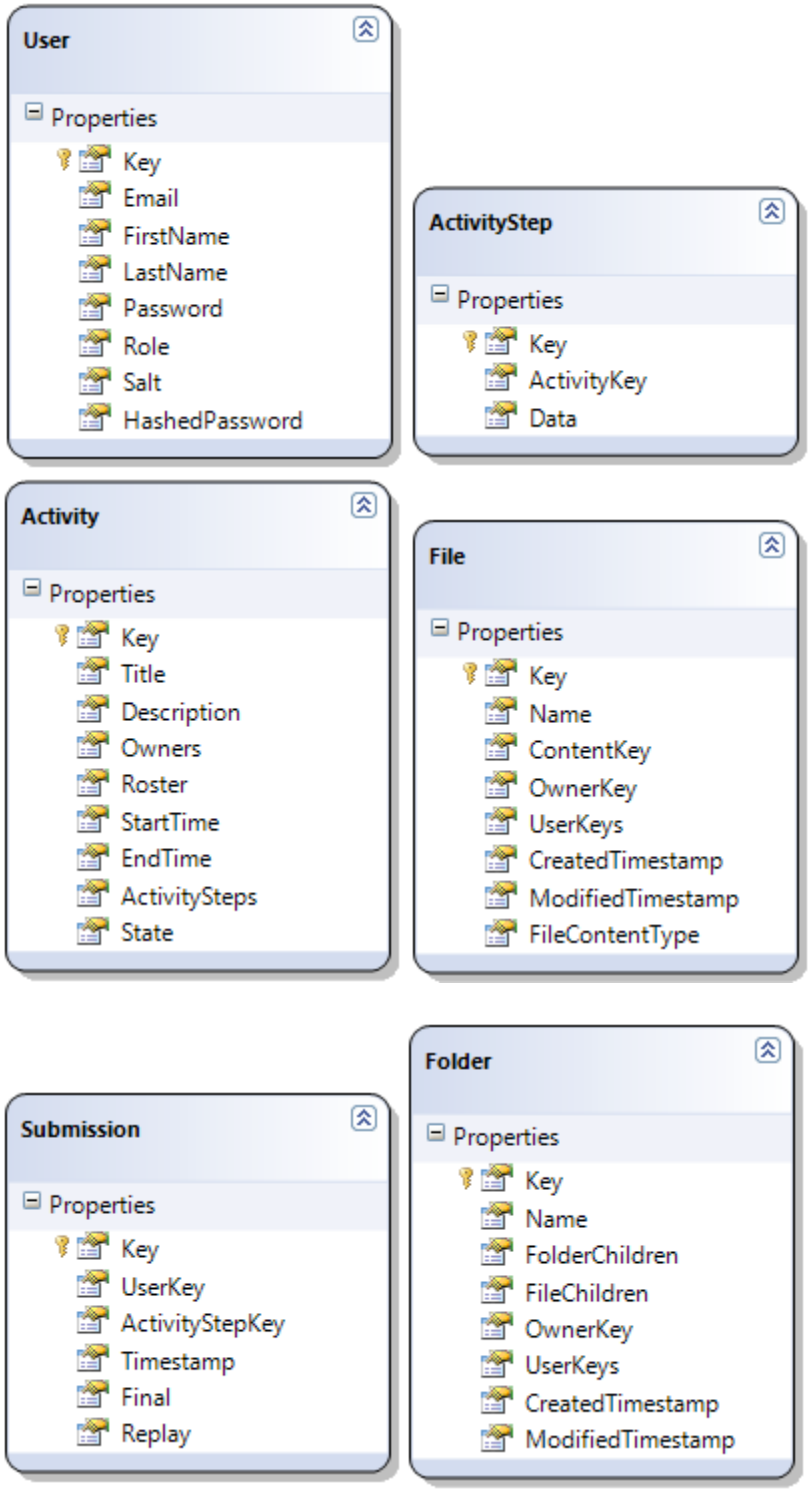
Graph

Background Image

Application Icon

Login Screen Image

Appendix 4: BeSocratic Database Schema



BeSocratic Database Table Descriptions

User Table		
Key	int	A unique primary key for a user
Email	String	The is the user's email address. Should also be unique
FirstName	String	User's First Name
LastName	String	User's Last Name
Password	String	[No longer used. Need to remove.] Unencrypted password
Role	int	0=student, 1=instructor, 2=admin
Salt	Byte[]	A random byte array used for security
HashedPassword	Byte[]	The user's actual password hashed with SHA-256

Activity		
Key	int	A unique primary key for the activity
Title	String	The visible title for this activity. This is what students see
Description	String	A description of the activity. This is to help instructors keep track of their activities
Owners	String	[Deprecated with the use of folders]And comma separated list of emails. Used for sharing activities
Roster	String	A comma separated list of emails. This dictates who is able to complete the activity.
StartTime	String	Timestamp for the time when this Activity becomes *Active*
EndTime	String	Timestamp for the time when this Activity is no longer *Active*
ActivitySteps	String	Comma separated list of integers. Each integer is an ActivityStep

		key
State		[Deprecated]

Activity Step		
Key	int	A unique primary key for an activity step
ActivityKey	int	The key of the activity this step is part of
Data	String	A json object containing all of the layout and module information for this activity step

Submission		
Key	int	A unique primary key for the submission
UserKey	int	The user who made the submission
ActivityStepKey	int	The activity step corresponding to the submission
Timestamp	String	The time of the submission
Final	String	A data snapshot of the activity step's final state
Replay	String	A data recording of the activity step's replay

Folder		
Key	int	A unique primary key for the folder
Name	String	Folder name
FolderChildren	String	Comma separated list of integers. Each integer is a key for a folder. This gives a hierarchical structure for our file system
FileChildren	String	Comma separated list of integers. Each integer is a key for a file.
OwnerKey	String	A user key for the creator of the folder
UserKeys	String	A comma separated list of integers. Each integer is a user key. These users will be able to see the contents of the folder. This gives BeSocratic collaborative functionality
CreatedTimestamp	String	The time the folder was created
ModifiedTimestamp	String	The last time the folder was modified

File		
Key	int	A unique primary key for the file
Name	String	File name
ContentKey	int	A generic key that points to the file contents in some table. Should be used with FileContentType. Right now this only points to an activity table entry
OwnerKey	String	A user key for the creator of the folder
UserKeys	String	A comma separated list of integers. Each integer is a user

		key. These users will be able to see the file. This gives BeSocratic collaborative functionality
CreatedTimestamp	String	The time the file was created
ModifiedTimestamp	String	The last time the file was modified
FileContentType	int	An identifier for the files data type. Like an extension. Currently, it is not really used since the only data stored is activities.

Appendix 5: Scope Document

Scope Statement (Version 1.0)

Project Title: uRespond Remote Answering System
Date: 9/1/2011 **Prepared by:** Andrew Herrmann

Project Justification:

Despite nearly four decades of literature describing "improved" ways of teaching Lewis structures, the creation of valid representations remains an elusive objective for many chemistry students.

The ability to predict the properties of a material from its molecular structure (and vice versa) is central to such an understanding. However, the links between molecular-level structure and properties are complex and require thoughtful study, practice, and skill. Without a robust understanding of these principles, students presented with the chemical structures of large biomolecules, for example, can only respond with surface-level learning and memorization. To effectively teach Lewis structures students need a chance at representational creation in an environment that allows them to make mistakes and receive substantive feedback. Creating such an environment would be especially beneficial for large classes.

Product Characteristics and Requirements:

1. The system must run on an iPad
2. The system must allow the user to receive questions and submit answers via a database over a WiFi internet connection
3. The system must include 7 question modules:
   - Short Essay
   - Formula
   - Multiple Choice
   - Picture ID
   - Graph Drawing
   - Chemical Structure Building
   - Blank Slate
4. The system must:
   - Allow students to register with the uRespond system
   - Allow students to review past questions, submitted answers and correct answers
   - Allow teachers simple class management (upload a CSV file of a class roster, add/delete students, etc…)
   - Allow teachers to create questions based on the seven question modules

and store them in a database
- Allow teachers to view student responses in real time
- Allow teachers to display particular responses anonymously
- Allow teachers to view past student responses

Following items must be considered out of scope:

1. Student response analysis, auto-grading

Summary of Project Deliverables

Project management-related deliverables: scope statement, WBS, schedule, status reports, final project presentation, final project report, lessons-learned report, and any other documents required to manage the project.

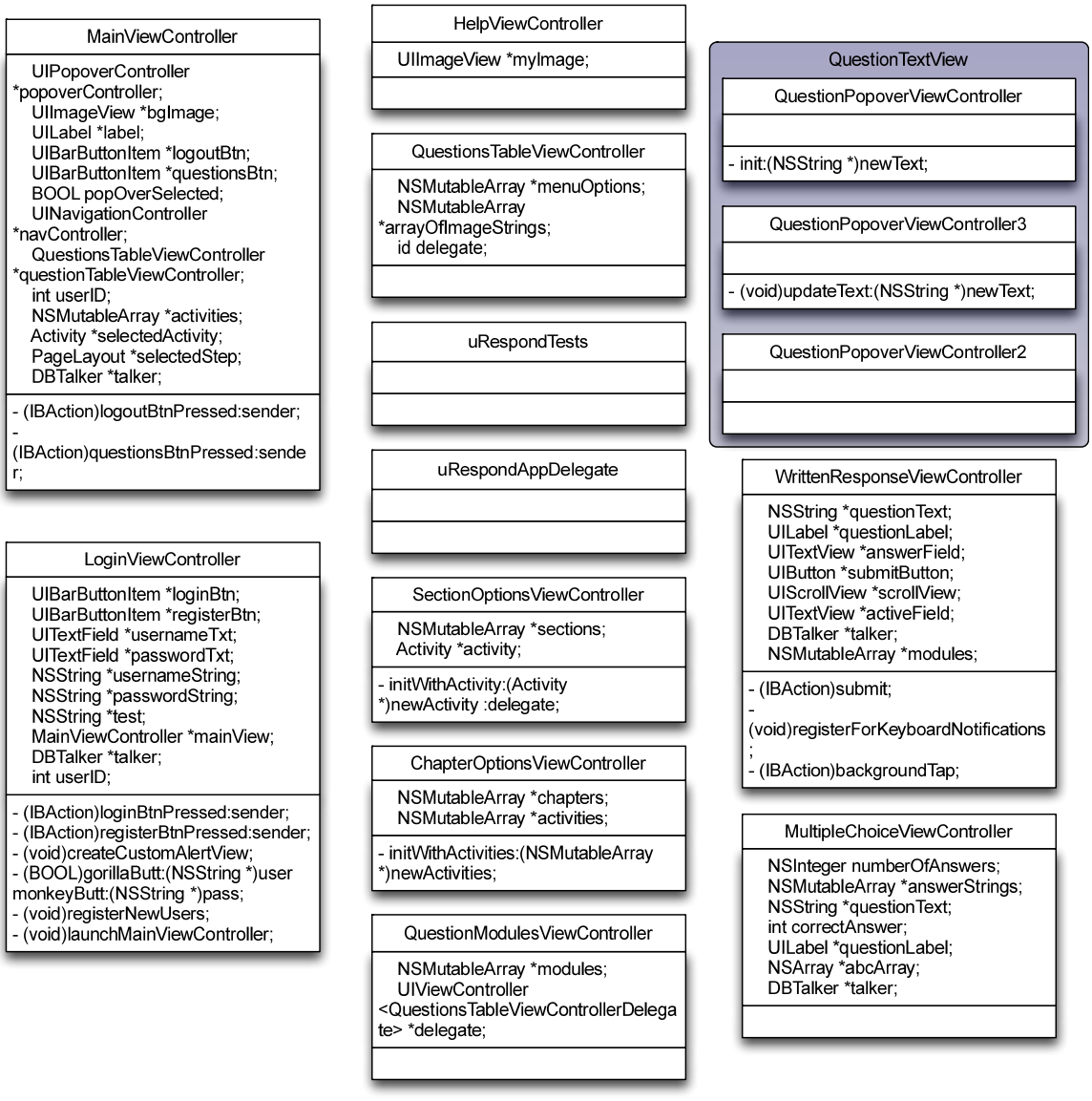
Product-related deliverables: research reports, design documents, software code, hardware, etc.

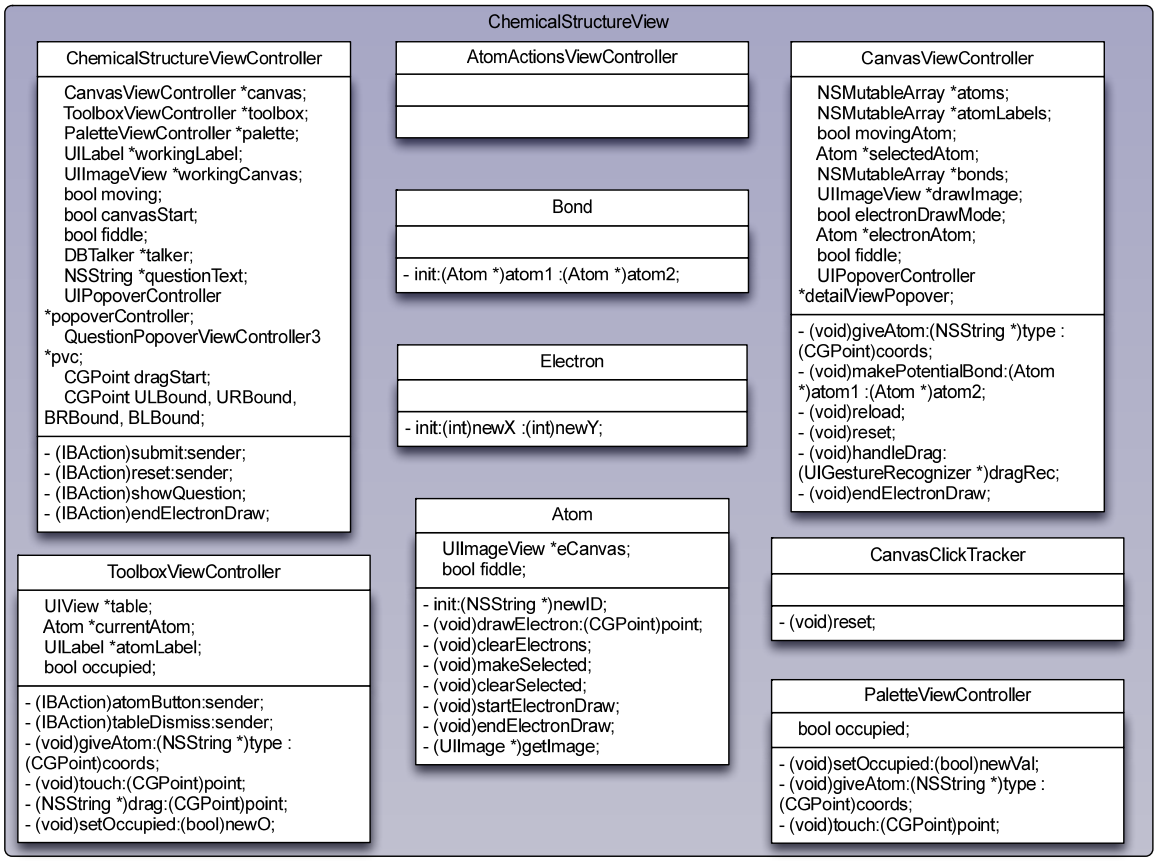
1. Design documents
2. iPad application & source code
3. Database schema and configuration create on server at Clemson
4. Modification of Question Creation/Class Management System (BeSocratic)
    a. Feature to see student responses in real time

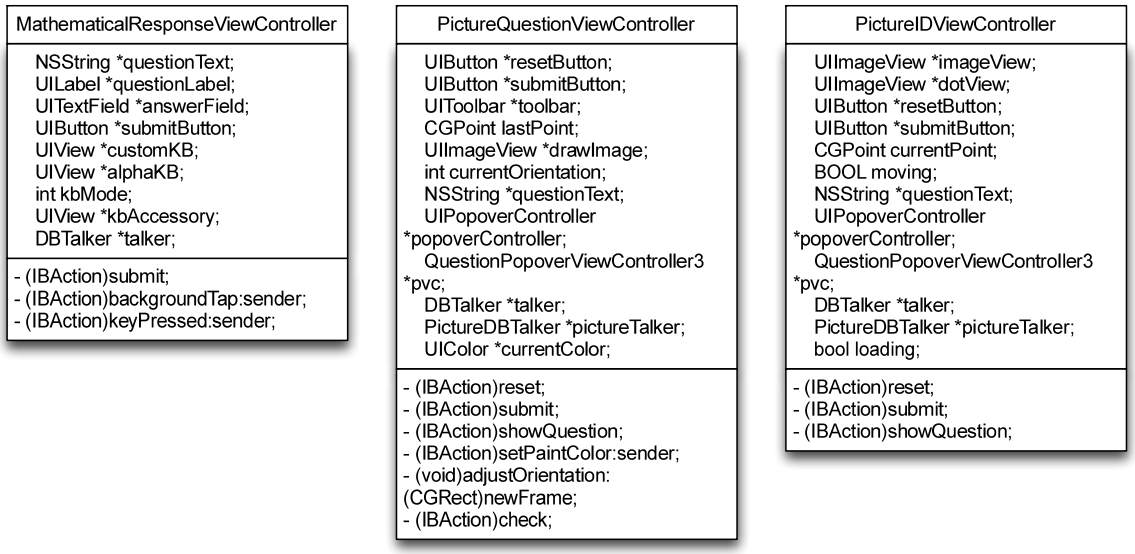
Project Success Criteria:

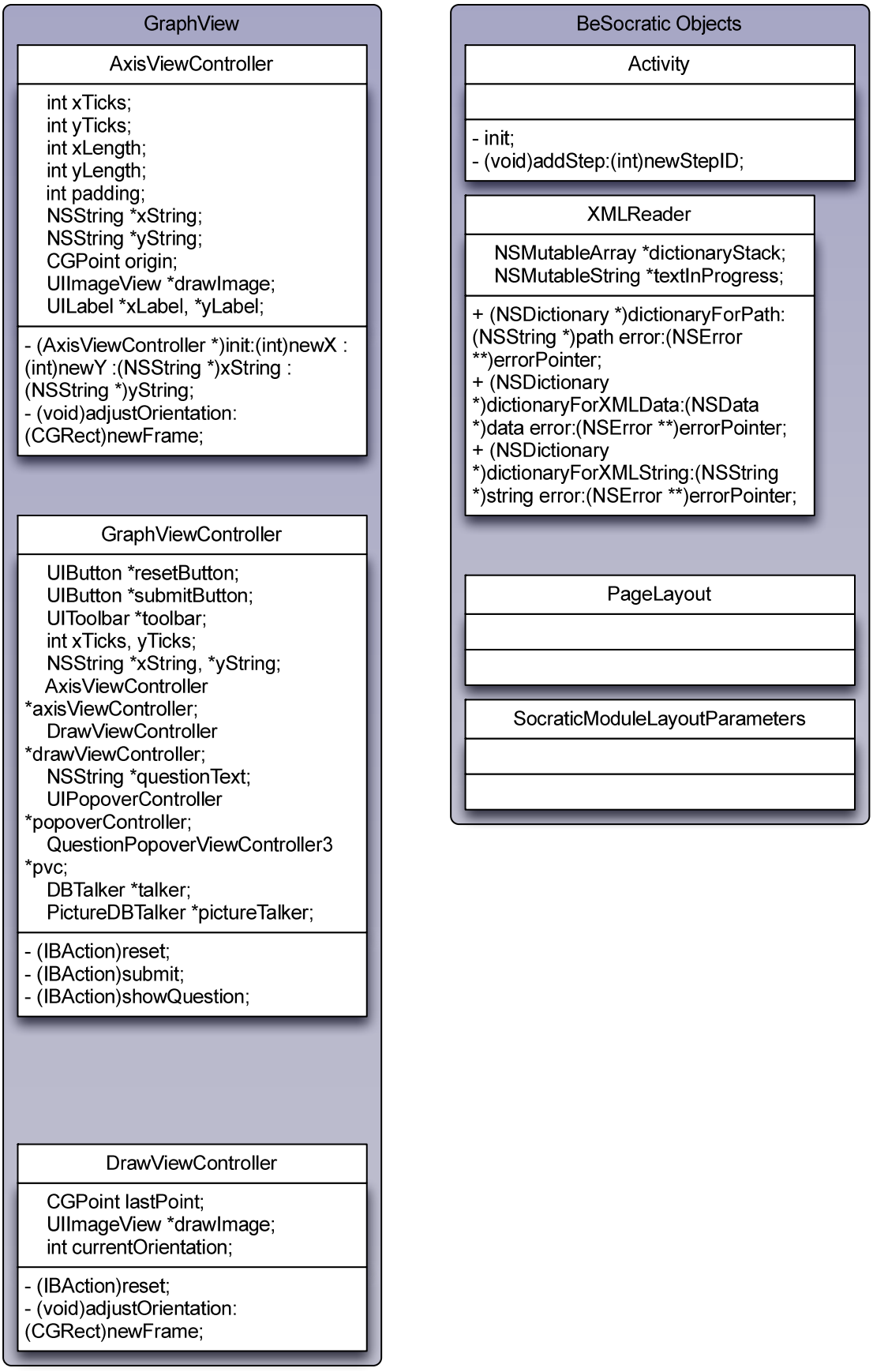
- A robust functional version of uRespond finished by 5/1/2012
- Relatively few bugs in the system
- Allow teachers to create questions and students to view and respond to them via the iPad application.

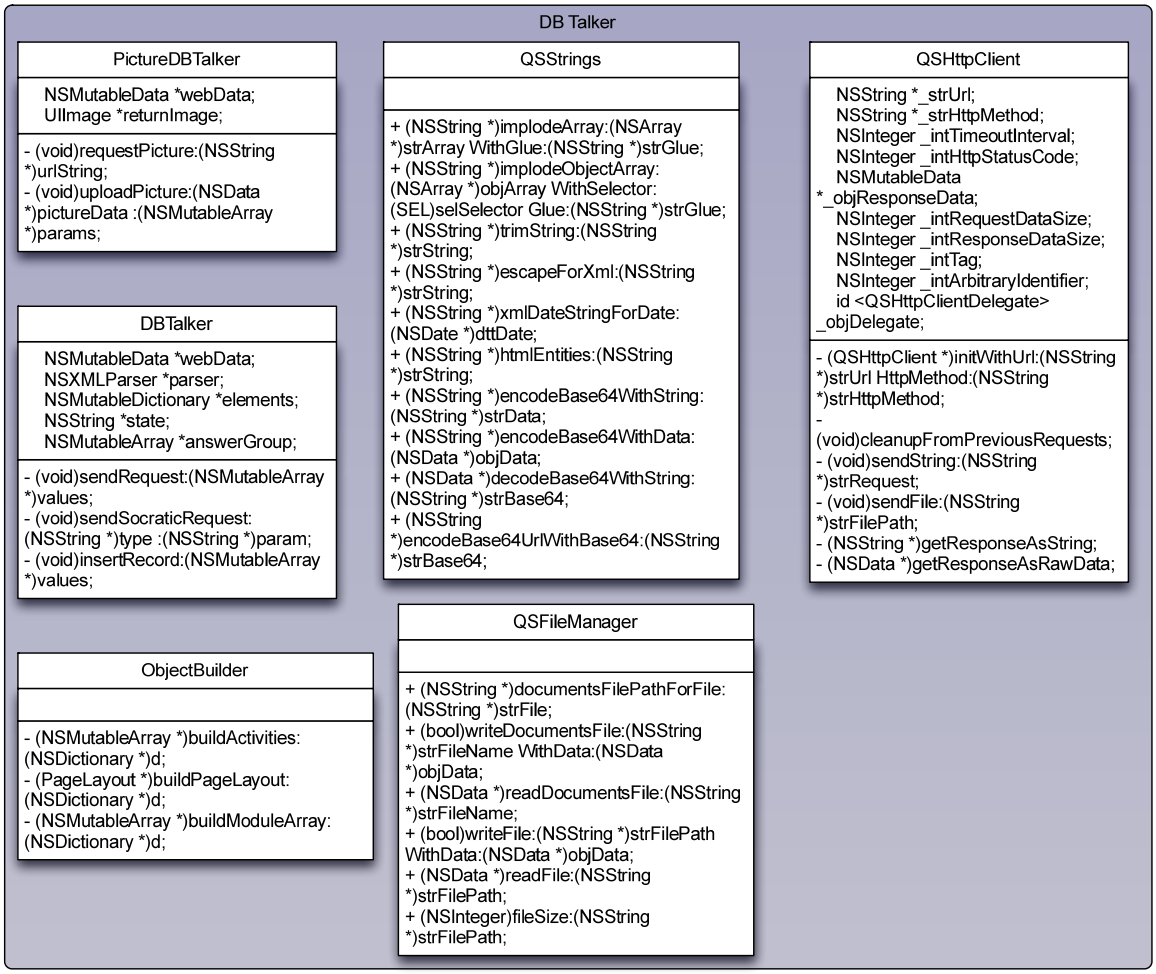
Appendix 6: uRespond UML Diagram

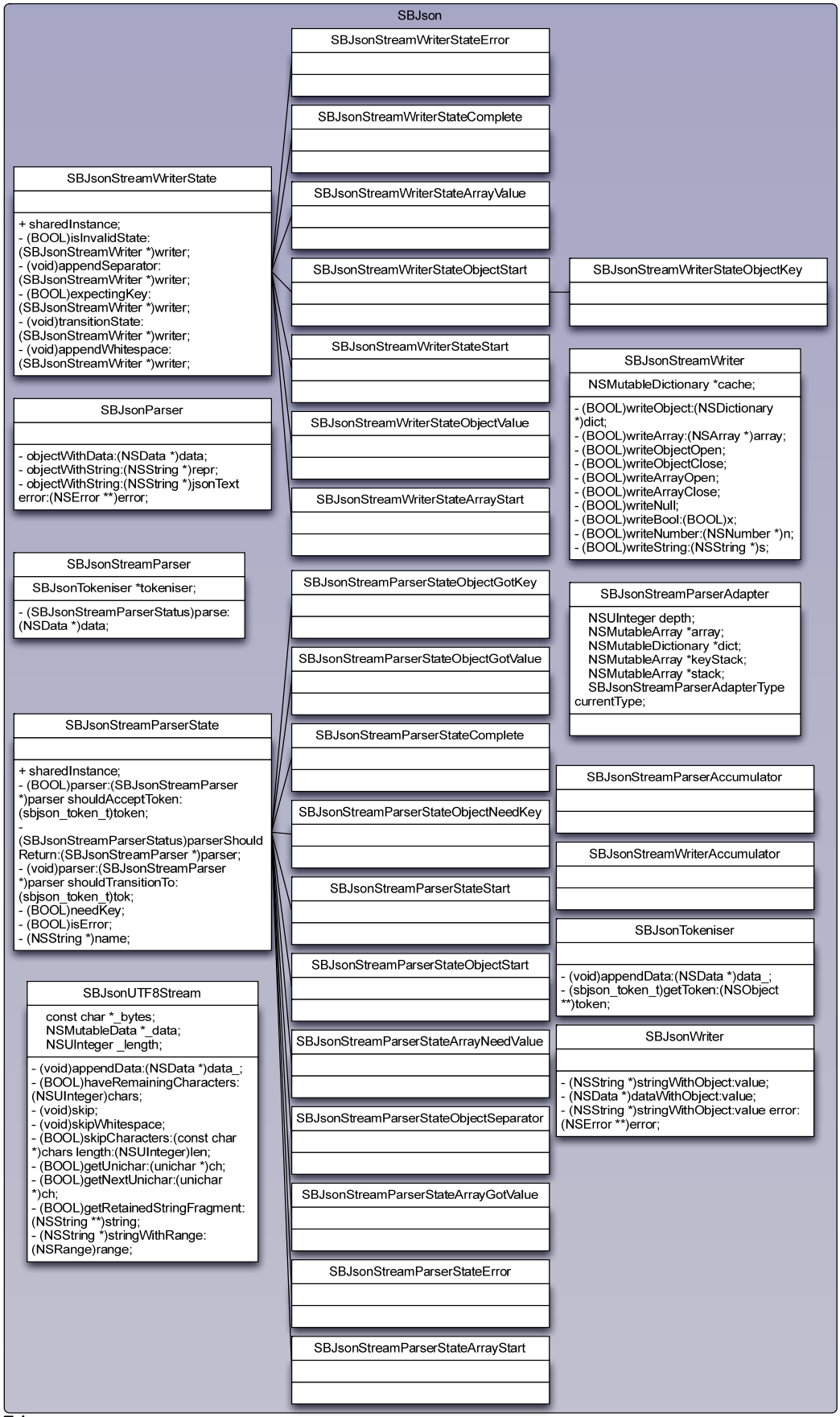



ChemicalStructureView
ChemicalStructureViewController
CanvasViewController *canvas;
ToolboxViewController *toolbox;
PaletteViewController *palette;
UILabel *workingLabel;
UIImageView *workingCanvas;
bool moving;
bool canvasStart;
bool fiddle;
DBTalker *talker;
NSString *questionText;
UIPopoverController *popoverController;
QuestionPopoverViewController3 *pvc;
CGPoint dragStart;
CGPoint ULBound, URBound, BRBound, BLBound;
- (IBAction)submit:sender;
- (IBAction)reset:sender;
- (IBAction)showQuestion;
- (IBAction)endElectronDraw;
AtomActionsViewController
Bond
- init:(Atom *)atom1 :(Atom *)atom2;
Electron
- init:(int)newX :(int)newY;
CanvasViewController
NSMutableArray *atoms;
NSMutableArray *atomLabels;
bool movingAtom;
Atom *selectedAtom;
NSMutableArray *bonds;
UIImageView *drawImage;
bool electronDrawMode;
Atom *electronAtom;
bool fiddle;
UIPopoverController *detailViewPopover;
- (void)giveAtom:(NSString *)type : (CGPoint)coords;
- (void)makePotentialBond:(Atom *)atom1 :(Atom *)atom2;
- (void)reload;
- (void)reset;
- (void)handleDrag: (UIGestureRecognizer *)dragRec;
- (void)endElectronDraw;
ToolboxViewController
UIView *table;
Atom *currentAtom;
UILabel *atomLabel;
bool occupied;
- (IBAction)atomButton:sender;
- (IBAction)tableDismiss:sender;
- (void)giveAtom:(NSString *)type : (CGPoint)coords;
- (void)touch:(CGPoint)point;
- (NSString *)drag:(CGPoint)point;
- (void)setOccupied:(bool)newO;
Atom
UIImageView *eCanvas;
bool fiddle;
- init:(NSString *)newID;
- (void)drawElectron:(CGPoint)point;
- (void)clearElectrons;
- (void)makeSelected;
- (void)clearSelected;
- (void)startElectronDraw;
- (void)endElectronDraw;
- (UIImage *)getImage;
CanvasClickTracker
- (void)reset;
PaletteViewController
bool occupied;
- (void)setOccupied:(bool)newVal;
- (void)giveAtom:(NSString *)type : (CGPoint)coords;
- (void)touch:(CGPoint)point;


MathematicalResponseViewController
NSString *questionText;
UILabel *questionLabel;
UITextField *answerField;
UIButton *submitButton;
UIView *customKB;
UIView *alphaKB;
int kbMode;
UIView *kbAccessory;
DBTalker *talker;
- (IBAction)submit;
- (IBAction)backgroundTap:sender;
- (IBAction)keyPressed:sender;
PictureQuestionViewController
UIButton *resetButton;
UIButton *submitButton;
UIToolbar *toolbar;
CGPoint lastPoint;
UIImageView *drawImage;
int currentOrientation;
NSString *questionText;
UIPopoverController *popoverController;
QuestionPopoverViewController3 *pvc;
DBTalker *talker;
PictureDBTalker *pictureTalker;
UIColor *currentColor;
- (IBAction)reset;
- (IBAction)submit;
- (IBAction)showQuestion;
- (IBAction)setPaintColor:sender;
- (void)adjustOrientation: (CGRect)newFrame;
- (IBAction)check;
PictureIDViewController
UIImageView *imageView;
UIImageView *dotView;
UIButton *resetButton;
UIButton *submitButton;
CGPoint currentPoint;
BOOL moving;
NSString *questionText;
UIPopoverController *popoverController;
QuestionPopoverViewController3 *pvc;
DBTalker *talker;
PictureDBTalker *pictureTalker;
bool loading;
- (IBAction)reset;
- (IBAction)submit;
- (IBAction)showQuestion;


GraphView
AxisViewController
int xTicks;
int yTicks;
int xLength;
int yLength;
int padding;
NSString *xString;
NSString *yString;
CGPoint origin;
UIImageView *drawImage;
UILabel *xLabel, *yLabel;
- (AxisViewController *)init:(int)newX :(int)newY :(NSString *)xString :(NSString *)yString;
- (void)adjustOrientation:(CGRect)newFrame;
GraphViewController
UIButton *resetButton;
UIButton *submitButton;
UIToolbar *toolbar;
int xTicks, yTicks;
NSString *xString, *yString;
AxisViewController *axisViewController;
DrawViewController *drawViewController;
NSString *questionText;
UIPopoverController *popoverController;
QuestionPopoverViewController3 *pvc;
DBTalker *talker;
PictureDBTalker *pictureTalker;
- (IBAction)reset;
- (IBAction)submit;
- (IBAction)showQuestion;
DrawViewController
CGPoint lastPoint;
UIImageView *drawImage;
int currentOrientation;
- (IBAction)reset;
- (void)adjustOrientation:(CGRect)newFrame;
BeSocratic Objects
Activity
- init;
- (void)addStep:(int)newStepID;
XMLReader
NSMutableArray *dictionaryStack;
NSMutableString *textInProgress;
+ (NSDictionary *)dictionaryForPath:(NSString *)path error:(NSError **)errorPointer;
+ (NSDictionary *)dictionaryForXMLData:(NSData *)data error:(NSError **)errorPointer;
+ (NSDictionary *)dictionaryForXMLString:(NSString *)string error:(NSError **)errorPointer;
PageLayout
SocraticModuleLayoutParameters


DB Talker
PictureDBTalker
NSMutableData *webData;
UIImage *returnImage;
- (void)requestPicture:(NSString *)urlString;
- (void)uploadPicture:(NSData *)pictureData :(NSMutableArray *)params;
QSStrings
+ (NSString *)implodeArray:(NSArray *)strArray WithGlue:(NSString *)strGlue;
+ (NSString *)implodeObjectArray:(NSArray *)objArray WithSelector:(SEL)selSelector Glue:(NSString *)strGlue;
+ (NSString *)trimString:(NSString *)strString;
+ (NSString *)escapeForXml:(NSString *)strString;
+ (NSString *)xmlDateStringForDate:(NSDate *)dttDate;
+ (NSString *)htmlEntities:(NSString *)strString;
+ (NSString *)encodeBase64WithString:(NSString *)strData;
+ (NSString *)encodeBase64WithData:(NSData *)objData;
+ (NSData *)decodeBase64WithString:(NSString *)strBase64;
+ (NSString *)encodeBase64UrlWithBase64:(NSString *)strBase64;
QSHttpClient
NSString *_strUrl;
NSString *_strHttpMethod;
NSInteger _intTimeoutInterval;
NSInteger _intHttpStatusCode;
NSMutableData *_objResponseData;
NSInteger _intRequestDataSize;
NSInteger _intResponseDataSize;
NSInteger _intTag;
NSInteger _intArbitraryIdentifier;
id <QSHttpClientDelegate> _objDelegate;
- (QSHttpClient *)initWithUrl:(NSString *)strUrl HttpMethod:(NSString *)strHttpMethod;
- (void)cleanupFromPreviousRequests;
- (void)sendString:(NSString *)strRequest;
- (void)sendFile:(NSString *)strFilePath;
- (NSString *)getResponseAsString;
- (NSData *)getResponseAsRawData;
DBTalker
NSMutableData *webData;
NSXMLParser *parser;
NSMutableDictionary *elements;
NSString *state;
NSMutableArray *answerGroup;
- (void)sendRequest:(NSMutableArray *)values;
- (void)sendSocraticRequest:(NSString *)type :(NSString *)param;
- (void)insertRecord:(NSMutableArray *)values;
QSFileManager
+ (NSString *)documentsFilePathForFile:(NSString *)strFile;
+ (bool)writeDocumentsFile:(NSString *)strFileName WithData:(NSData *)objData;
+ (NSData *)readDocumentsFile:(NSString *)strFileName;
+ (bool)writeFile:(NSString *)strFilePath WithData:(NSData *)objData;
+ (NSData *)readFile:(NSString *)strFilePath;
+ (NSInteger)fileSize:(NSString *)strFilePath;
ObjectBuilder
- (NSMutableArray *)buildActivities:(NSDictionary *)d;
- (PageLayout *)buildPageLayout:(NSDictionary *)d;
- (NSMutableArray *)buildModuleArray:(NSDictionary *)d;


SBJson
SBJsonStreamWriterStateError
SBJsonStreamWriterStateComplete
SBJsonStreamWriterStateArrayValue
SBJsonStreamWriterStateObjectStart
SBJsonStreamWriterStateObjectKey
SBJsonStreamWriterStateStart
SBJsonStreamWriterStateObjectValue
SBJsonStreamWriterStateArrayStart
SBJsonStreamWriterState
+ sharedInstance;
- (BOOL)isInvalidState:
(SBJsonStreamWriter *)writer;
- (void)appendSeparator:
(SBJsonStreamWriter *)writer;
- (BOOL)expectingKey:
(SBJsonStreamWriter *)writer;
- (void)transitionState:
(SBJsonStreamWriter *)writer;
- (void)appendWhitespace:
(SBJsonStreamWriter *)writer;
SBJsonStreamWriter
NSMutableDictionary *cache;
- (BOOL)writeObject:(NSDictionary
*)dict;
- (BOOL)writeArray:(NSArray *)array;
- (BOOL)writeObjectOpen;
- (BOOL)writeObjectClose;
- (BOOL)writeArrayOpen;
- (BOOL)writeArrayClose;
- (BOOL)writeNull;
- (BOOL)writeBool:(BOOL)x;
- (BOOL)writeNumber:(NSNumber *)n;
- (BOOL)writeString:(NSString *)s;
SBJsonParser
- objectWithData:(NSData *)data;
- objectWithString:(NSString *)repr;
- objectWithString:(NSString *)jsonText
error:(NSError **)error;
SBJsonStreamParser
SBJsonTokeniser *tokeniser;
- (SBJsonStreamParserStatus)parse:
(NSData *)data;
SBJsonStreamParserStateObjectGotKey
SBJsonStreamParserStateObjectGotValue
SBJsonStreamParserStateComplete
SBJsonStreamParserStateObjectNeedKey
SBJsonStreamParserStateStart
SBJsonStreamParserStateObjectStart
SBJsonStreamParserStateArrayNeedValue
SBJsonStreamParserStateObjectSeparator
SBJsonStreamParserStateArrayGotValue
SBJsonStreamParserStateError
SBJsonStreamParserStateArrayStart
SBJsonStreamParserAdapter
NSUInteger depth;
NSMutableArray *array;
NSMutableDictionary *dict;
NSMutableArray *keyStack;
NSMutableArray *stack;
SBJsonStreamParserAdapterType
currentType;
SBJsonStreamParserState
+ sharedInstance;
- (BOOL)parser:(SBJsonStreamParser
*)parser shouldAcceptToken:
(sbjson_token_t)token;
-
(SBJsonStreamParserStatus)parserShould
Return:(SBJsonStreamParser *)parser;
- (void)parser:(SBJsonStreamParser
*)parser shouldTransitionTo:
(sbjson_token_t)tok;
- (BOOL)needKey;
- (BOOL)isError;
- (NSString *)name;
SBJsonStreamParserAccumulator
SBJsonStreamWriterAccumulator
SBJsonTokeniser
- (void)appendData:(NSData *)data_;
- (sbjson_token_t)getToken:(NSObject
**)token;
SBJsonUTF8Stream
const char *_bytes;
NSMutableData *_data;
NSUInteger _length;
- (void)appendData:(NSData *)data_;
- (BOOL)haveRemainingCharacters:
(NSUInteger)chars;
- (void)skip;
- (void)skipWhitespace;
- (BOOL)skipCharacters:(const char
*)chars length:(NSUInteger)len;
- (BOOL)getUnichar:(unichar *)ch;
- (BOOL)getNextUnichar:(unichar
*)ch;
- (BOOL)getRetainedStringFragment:
(NSString **)string;
- (NSString *)stringWithRange:
(NSRange)range;
SBJsonWriter
- (NSString *)stringWithObject:value;
- (NSData *)dataWithObject:value;
- (NSString *)stringWithObject:value error:
(NSError **)error;

Appendix 7: Sample of Code Written

```
//
//  MainViewController.m
//  uRespond
//
//  Created by Andrew Herrmann on 6/6/11.
//  Copyright 2011 University of North Carolina Wilmington. All rights reserved.
//

#import "MainViewController.h"
#import "QuestionsTableViewController.h"
#import "ChapterOptionsViewController.h"
#import "DBTalker.h"
#import "Activity.h"
#import <sqlite3.h>

#import "SocraticModuleLayoutParameters.h"

@implementation MainViewController


@synthesize questionTableViewController;
@synthesize logoutBtn, questionsBtn, label, popoverController, bgImage, delegate;



- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)dealloc
{
    [label release];
    [questionsBtn release];
    [super dealloc];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)populateSteps {
```

```
    NSLog(@"mv: Populate Steps");
    for(Activity *activity in activities){
        NSLog(@"mv: activity %d", activity.ActivityID);
        NSLog(@"mv: step count %d", [activity.steps count]);
        for(PageLayout *p in activity.steps){
            [talker sendSocraticRequest:@"step" :[NSString stringWithFormat:@"%d", p.key]];
        }
    }
}

- (Activity *)activityWithKey:(int)key {
    for(Activity *a in activities){
        if(a.ActivityID == key) return a;
    }
    return nil;
}

- (PageLayout *)stepWithKey:(int)key {
    for(Activity *a in activities){
        for(PageLayout *p in a.steps) {
            if(p.key == key) return p;
        }
    }
    return nil;
}

- (void)getResponse:(NSMutableDictionary *)elements {
    if([elements objectForKey:@"activities"] != nil){
        activities = [elements objectForKey:@"activities"];
        [self populateSteps];
    }
    else if([elements objectForKey:@"step"] != nil){
        NSLog(@"MainView got step response");
        PageLayout *pageIn = [elements objectForKey:@"step"];
        int inKey = pageIn.key;
        NSLog(@"mv: storing step %d", pageIn.key);
  //      PageLayout *store = [self stepWithKey:inKey];
  //      store = pageIn;
        [self stepWithKey:inKey].Modules = [[NSMutableArray alloc] initWithArray:pageIn.Modules];
        SocraticModuleLayoutParameters *smlp = [[self stepWithKey:inKey].Modules
objectAtIndex:0];
//        NSLog(@"data in 0 module: %@", smlp.Data);
    }
}

#pragma mark - View lifecycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    // Do any additional setup after loading the view from its nib.

    //Uncomment here if running on device, need writable db
    //[self createWritableDatabase];
```

```
    questionTableViewController = [[QuestionsTableViewController alloc] init];

    questionTableViewController.delegate = self;
    questionTableViewController.navigationItem.title = @"Main Menu";
    navController =
    [[UINavigationController alloc] initWithRootViewController:questionTableViewController];

    popoverController = [[UIPopoverController alloc]
                         initWithContentViewController:navController];
    popoverController.delegate = self;
    popoverController.popoverContentSize = CGSizeMake(250, 250);

    popOverSelected = true;

    [questionTableViewController release];
    //[navController release];

    userID = 430;//[delegate getUserID];

    talker = [[DBTalker alloc] init];
    [talker setDelegate:self];
    [talker sendSocraticRequest:@"activity" :[NSString stringWithFormat:@"%d", userID]];

}

- (void)createWritableDatabase{
    BOOL success;

    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSError *error;
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *writableDBPath = [documentsDirectory
stringByAppendingPathComponent:@"data.sqlite3"];
    success = [fileManager fileExistsAtPath:writableDBPath];

    if (success) return;

    NSLog(@"no database, creating");
    // The writable database does not exist, so copy the default to the appropriate location.
    NSString *defaultDBPath = [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"data.sqlite3"];
    success = [fileManager copyItemAtPath:defaultDBPath toPath:writableDBPath error:&error];
    if (!success) {
        NSAssert1(0, @"Failed to create writable database file with message '%@'.", [error
localizedDescription]);
    }

    //create appropriate tables and insert necessary data
    sqlite3 *database;
    if (sqlite3_open([writableDBPath UTF8String], &database)
        != SQLITE_OK) {
        sqlite3_close(database);
        NSAssert(0, @"Failed to open database");
    }
```

```
    sqlite3_close(database);
}



- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


// Only supporting one view
//- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
//{
//    // Return YES for supported orientations
//            return YES;
//}

#pragma marks - IBAction Methods

-(IBAction) logoutBtnPressed:(id)sender
{
    //[[self.view superview] addSubview:newView.view];
    [self.view removeFromSuperview];
    //[self.view addSubview: newView.view];
}


//Show popover when button is pressed

-(IBAction) questionsBtnPressed:(id)sender
{
    NSLog(@"Questions was Pressed");


    if ([popoverController isPopoverVisible])
    {

        [popoverController dismissPopoverAnimated:YES];

    }
    else

    {
        NSLog(@"NavController value: %@", navController.visibleViewController);

        QuestionsTableViewController *qtvc = (QuestionsTableViewController
*)navController.visibleViewController;
        qtvc.qpDelegate = self;
        NSLog(@"main view setting activities");
        [qtvc setActivities:activities];

        NSLog(@"main view presenting popover");
```

```
        [popoverController presentPopoverFromBarButtonItem:questionsBtn
                                  permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];


            if (popOverSelected == false) {



            }
    }

}



//---called when the user clicks outside the popover view---
- (BOOL)popoverControllerShouldDismissPopover:(UIPopoverController *)popoverController {

    NSLog(@"popover about to be dismissed");
    //popOverSelected = false;
    return YES;
}

//---called when the popover view is dismissed---
- (void)popoverControllerDidDismissPopover:
(UIPopoverController *)popoverController {

    NSLog(@"popover dismissed");
    //popOverSelected = false;

    //this broke after upgrading to iOS5
    //[navController popToRootViewControllerAnimated:YES];
   }

-(void)didTap:(NSString *)string imageName:(NSString*)image {


    bgImage.image = [UIImage imageNamed:image];

    label.text = string;

   // [self.popoverController dismissPopoverAnimated:YES];
}

-(void)changeView:(NSString *)string{
    NSLog(@"String: %@", string);
    string = @"login_screen";
    Class className = NSClassFromString(string);
    UIViewController *newView = [[className alloc] initWithNibName:string bundle:nil];
    newView.view.frame = CGRectMake(0.0, 44.0, 1024, 724.0);


    [self.view addSubview: newView.view];

    //[self.navigationController setNavigationBarHidden:NO animated: YES];
```

```
}

- (void)selectActivity:(Activity *)newActivity {
    selectedActivity = newActivity;
}

- (void)showQuestion:(PageLayout *)selected {
    //
    //to do: use Template field to determine type of view controller to show (hardcoded for now
    //       because Sam has yet to add Template field)
    //

    NSLog(@"showQuestion %d", selected.key);
    //NSLog(@"selected %@", [[selected.Modules objectAtIndex:0] description]);

    //NSString *type = @"WrittenResponseViewController";
    NSString *type = @"PictureIDViewController";

    Class className = NSClassFromString(type);

    UIViewController *newView = [[className alloc] initWithQuestionData:selected];
    newView.view.frame = CGRectMake(0.0, 44.0, 1024, 724.0);
 //   [newView setQuestionData:selected];

    [self.view addSubview:newView.view];
}

@end


//
//  SectionOptionsViewController.m
//  uRespond
//
//  Created by Andrew Herrmann on 7/25/11.
//  Copyright 2011 University of North Carolina Wilmington. All rights reserved.
//

#import "SectionOptionsViewController.h"


@implementation SectionOptionsViewController
@synthesize qpDelegate;
- (id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (id)initWithActivity:(Activity *)newActivity:(id<QuestionPasserDelegate>)newDelegate {
    if((self = [super init])){
        activity = newActivity;
        qpDelegate = newDelegate;
```

```
    }
    return self;
}

- (void)setActivities:(NSMutableArray *)activities {  //main view may call this, thinking it's talking to
a QuestionsTableViewController
}

- (void)dealloc
{
    [super dealloc];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

#pragma mark - View lifecycle

- (void)viewDidLoad
{
    NSLog(@"RICKY03");
    [super viewDidLoad];

    self.title = @"Questions";

    if(activity == nil) NSLog(@"SectionView activity null");

        sections = [[NSMutableArray alloc] init];

    for(PageLayout *p in activity.steps){
        [sections addObject:[NSString stringWithFormat:@"Question %d", p.key]];
    }



     self.clearsSelectionOnViewWillAppear = NO;


     self.navigationItem.rightBarButtonItem = self.editButtonItem;
}

- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)viewWillAppear:(BOOL)animated
{
    NSLog(@"RICKY02");
```

```
    [self.tableView reloadData];
    [super viewWillAppear:animated];
    CGSize size = CGSizeMake(250, [sections count] * 44); // size of view in popover
    self.contentSizeForViewInPopover = size;
}

- (void)viewDidAppear:(BOOL)animated
{
    NSLog(@"RICKY01");
    [super viewDidAppear:animated];
}

- (void)viewWillDisappear:(BOOL)animated
{

    [super viewWillDisappear:animated];
}

- (void)viewDidDisappear:(BOOL)animated
{
    [super viewDidDisappear:animated];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
            return YES;
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{

    // Return the number of rows in the section.
    return [sections count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell...
```

```
    cell.textLabel.text = [sections objectAtIndex:indexPath.row];


    return cell;
}


#pragma mark - Table view delegate

- (PageLayout *)stepWithKey:(int)key {
    for(PageLayout *p in activity.steps){
        if(p.key == key) return p;
    }
    return nil;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{


    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    NSString *pid = [cell.textLabel.text substringFromIndex:9];

    PageLayout *selected = [self stepWithKey:[pid intValue]];
    NSLog(@"pid %d", selected.key);
    NSLog(@"count %d", [selected.Modules count]);
    if(selected == nil) NSLog(@"selected null");
    if(self.qpDelegate == nil) NSLog(@"delegate null");

    [self.qpDelegate showQuestion:selected];
}

@end


//
//  AtomActionsViewController.m
//  uRespond
//
//  Created by Andrew Herrmann on 9/8/11.
//  Copyright 2011 University of North Carolina Wilmington. All rights reserved.
//

#import "AtomActionsViewController.h"


@implementation AtomActionsViewController
@synthesize delegate;


- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
```

```
}

#pragma mark - View lifecycle

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.
}

- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
            return NO;
}
- (NSInteger)tableView:(UITableView *)aTableView numberOfRowsInSection:(NSInteger)section
{

    // Two sections, one for each detail view controller.
    return 4;
}


- (UITableViewCell *)tableView:(UITableView *)aTableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"AtomActionsViewControllerCellIdentifier";

    // Dequeue or create a cell of the appropriate type.
    UITableViewCell *cell = [self.tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
    }

    // Set appropriate labels for the cells.
    switch(indexPath.row){
        case 0:
            cell.textLabel.text = @"Add Electrons";
            break;
        case 1:
            cell.textLabel.text = @"Clear Electrons";
            break;
        case 2:
            cell.textLabel.text = @"Delete Atom";
            break;
        default:
            cell.textLabel.text = @"Dismiss Menu";
    }
```

```
    return cell;
}



#pragma mark -
#pragma mark Table view selection

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSUInteger row = indexPath.row;

    [delegate didTap:row];


}


#pragma mark -
#pragma mark Memory management

- (void)dealloc {
    [super dealloc];
}

@end




//
//  WrittenResponseViewController.m
//  uRespond
//
//  Created by Andrew Herrmann on 6/6/11.
//  Copyright 2011 University of North Carolina Wilmington. All rights reserved.
//

//Controls the question module which asks the student a short-answer question, allowing a
paragraph-length (or so) response.

#import "WrittenResponseViewController.h"
#import <sqlite3.h>
#import "DBTalker.h"
#import "SocraticModuleLayoutParameters.h"
#import "XMLReader.h"

#define FILE_NAME @"data.sqlite3"

@implementation WrittenResponseViewController

@synthesize questionLabel, answerField, submitButton, scrollView, questionData, delegate;
```

```
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (id)initWithQuestionData:(PageLayout *)newQData {
    if((self = [super init])){
        questionData = newQData;
    }
    return self;
}

- (void)submitAnswer:(NSString *)sub {
    NSMutableArray *values = [[NSMutableArray alloc] initWithObjects: nil];
    [values addObject:@"sa"]; //type
    [values addObject:@"1"];  //sid
    [values addObject:@"1"];  //qid
    [values addObject:sub];  //answer
    [talker insertRecord:values];
    [values release];
}

- (IBAction)submit{
    [self submitAnswer:answerField.text];
}

- (IBAction)backgroundTap{
    //A tap on the background of this view dismisses the keyboard.
    [answerField endEditing:true];
}

- (NSString *)dataFilePath {
    //return path to writeable database
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    return [documentsDirectory stringByAppendingPathComponent:FILE_NAME];
}

- (void)storeQuestionData {
    //for testing purposes only
    //convert sample data to storable form and put in database
    sqlite3 *database;
    if (sqlite3_open([[self dataFilePath] UTF8String], &database)
        != SQLITE_OK) {
        sqlite3_close(database);
        NSAssert(0, @"Failed to open database");
    }

    NSString *update = @"update saquestions set questiontext=? where qid=1";
    sqlite3_stmt *statement;
    if(sqlite3_prepare_v2(database, [update UTF8String], -1, &statement, NULL) != SQLITE_OK)
```

```
        NSAssert1(0, @"Error while creating update statement. '%s'", sqlite3_errmsg(database));
    sqlite3_bind_text(statement, 1, "One electron is removed from both a Na atom and a K atom,
producing two ions. Using the principles of atomic structure, explain why the Na ion is much
smaller than the K ion. Discuss both ions in your answer.", -1, NULL);

    if(SQLITE_DONE != sqlite3_step(statement))
        NSAssert1(0, @"Error while updating. '%s'", sqlite3_errmsg(database));

    sqlite3_close(database);
}

- (void)loadQuestionData {
    //load the question data (just question text) from database and displays it
    //This method should get the real questionID rather than hardcoding 1...
    sqlite3 *database;
    if (sqlite3_open([[self dataFilePath] UTF8String], &database)
        != SQLITE_OK) {
        sqlite3_close(database);
        NSAssert(0, @"Failed to open database");
    }

    NSString *getQuestionData = [[NSString alloc] initWithFormat:@"select questiontext from
saquestions where qid=1"];
    sqlite3_stmt *statement;
    if(sqlite3_prepare_v2(database, [getQuestionData UTF8String], -1, &statement, nil) ==
SQLITE_OK){
        while(sqlite3_step(statement) == SQLITE_ROW){
            questionText = [NSString stringWithUTF8String:sqlite3_column_text(statement, 0)];
        }
    }
    [getQuestionData release];
    questionLabel.text = questionText;
    sqlite3_close(database);
}

- (void)loadTalker {

    NSMutableArray *values = [[NSMutableArray alloc] initWithObjects: nil];
    [values addObject:@"questiontext"];
    [values addObject:@"saquestions"];
    [values addObject:@"qid"];
    [values addObject:@"1"];
    [values addObject:@"text"];
    [talker sendRequest:values];
    [values release];
}
- (void)getResponse:(NSMutableDictionary *)elements{
    questionText = [elements objectForKey:@"questiontext"];
    questionLabel.text = questionText;
}

- (void)dealloc
{
    [questionLabel release];
    [answerField release];
    [submitButton release];
```

```
    [super dealloc];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

#pragma mark - View lifecycle

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.

    talker = [[DBTalker alloc] init];
    [talker setDelegate:self];

    questionLabel.text = @"Please wait, loading question...";

    answerField.text = nil;
//    [self storeQuestionData];
//    [self loadQuestionData];
//    [self loadTalker];

    NSLog(@"questiondata %@", [questionData description]);

    for(SocraticModuleLayoutParameters *smlp in questionData.Modules){
        NSString *dataString = smlp.Data;
        NSRange runText = [dataString rangeOfString:@"<Run Text=\\\""];
        int start = runText.location+runText.length;
        NSString *remainder = [dataString substringFromIndex:start];
        NSRange chaff = [remainder rangeOfString:@"\\\""];
        NSString *text = [remainder substringToIndex:chaff.location];
        NSLog(@"chaff %d, %d", chaff.location, chaff.length);
        NSLog(@"questionText %@", text);
        questionLabel.text = text;
    }

    [self registerForKeyboardNotifications];
    activeField = answerField;

}

// Call this method somewhere in your view controller setup code.
- (void)registerForKeyboardNotifications
{
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(keyboardWasShown:)
                                                 name:UIKeyboardDidShowNotification object:nil];

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(keyboardWillBeHidden:)
```

```
                                    name:UIKeyboardWillHideNotification object:nil];

}

// Called when the UIKeyboardDidShowNotification is sent.
- (void)keyboardWasShown:(NSNotification*)aNotification
{
    //Move the answer field up (over the question text) so it can be seen while typing

    NSDictionary* info = [aNotification userInfo];
    CGSize kbSize = [[info objectForKey:UIKeyboardFrameBeginUserInfoKey] CGRectValue].size;

    UIEdgeInsets contentInsets = UIEdgeInsetsMake(0.0, 0.0, kbSize.height, 0.0);
    scrollView.contentInset = contentInsets;
    scrollView.scrollIndicatorInsets = contentInsets;

    }

// Called when the UIKeyboardWillHideNotification is sent
- (void)keyboardWillBeHidden:(NSNotification*)aNotification
{
    //move answer field back
    UIEdgeInsets contentInsets = UIEdgeInsetsZero;
    scrollView.contentInset = contentInsets;
    scrollView.scrollIndicatorInsets = contentInsets;
}


- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
    self.questionLabel = nil;
    self.answerField = nil;
    self.submitButton = nil;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
            return YES;
}

@end



//
//  MulitpleChoiceViewController.m
//  uRespond
//
//  Created by Andrew Herrmann on 6/6/11.
//  Copyright 2011 University of North Carolina Wilmington. All rights reserved.
//
```

```
//Controls the question module that presents the student with a multiple-choice question
//(maximum answer choices = 6)

#import "MultipleChoiceViewController.h"
#import <sqlite3.h>
#import "DBTalker.h"

#define FILE_NAME @"data.sqlite3"


@implementation MultipleChoiceViewController

- (NSString *)dataFilePath {
    //return path to writeable database
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    return [documentsDirectory stringByAppendingPathComponent:FILE_NAME];
}

- (void)dealloc
{
    [talker release];
    [questionText release];
    [answerStrings release];
    [abcArray release];
    [super dealloc];
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)loadData {
    NSLog(@"loading data %d", numberOfAnswers);
    NSString *s;
    for( int i = 0; i < numberOfAnswers ; i++ ) {
        //Create evenly-spaced buttons/labels for answer choices

        NSLog(@"Adding button %d", i);

        //create the button
        UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
        [button setTag:i];
        [button addTarget:self action:@selector(buttonClicked:)
forControlEvents:UIControlEventTouchUpInside];

        //set the position of the button
        if (i == 0) {
            button.frame = CGRectMake(200 , 250 , 150, (110 - (numberOfAnswers*10)));

            //create the label
```

```
        UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(360, 250, 600, (110 -
(numberOfAnswers*10)))];
        label.text = @"Answer";
        label.font = [UIFont fontWithName:@"Arial" size:(24.0 - numberOfAnswers)];
        label.numberOfLines = 0;
        label.backgroundColor = [UIColor clearColor];
        label.textColor = [UIColor whiteColor];
        label.text = [answerStrings objectAtIndex:i];
        [self.view addSubview:label];
        [label release];



        //  label.frame = CGRectMake(210, 250, 100, 80);

    }else {
        button.frame = CGRectMake(200, 250 + i*(110 - (numberOfAnswers*10) +
(120/numberOfAnswers)), 150, (110 - (numberOfAnswers*10)));

        //create the label
        UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(360, 250 +  i*(110 -
(numberOfAnswers*10) + (120/numberOfAnswers)), 600, (110 - (numberOfAnswers*10)))];
        label.text = @"Answer";
        label.font = [UIFont fontWithName:@"Helvetica" size:(24.0 - numberOfAnswers)];
        label.numberOfLines = 0;
        label.backgroundColor = [UIColor clearColor];
        label.textColor = [UIColor whiteColor];
        label.text = [answerStrings objectAtIndex:i];
        [self.view addSubview:label];
        [label release];


        //label.frame = CGRectMake(210, 250 + i*(110 - (numberOfAnswers*10) +
(120/numberOfAnswers)), 50, 8);
    }


    //set the button's title
    s = [[NSString alloc] initWithFormat:@"%@", [abcArray objectAtIndex:i]];

    [button setTitle: s
          forState:UIControlStateNormal];
    button.tag = i;

    //set the label properties


    [self.view addSubview:button];
    [s release];

  }

}

- (void)loadTalker {
  //DBTalker *talker = [[DBTalker alloc] init];
```

```
    //[talker setDelegate:self];
    NSMutableArray *values = [[NSMutableArray alloc] initWithObjects: nil];
    [values addObject:@"questiontext, answergroup, rightanswer"];
    [values addObject:@"mcquestions"];
    [values addObject:@"qid"];
    [values addObject:@"1"];
    [values addObject:@"mc"];
    [talker sendRequest:values];
    [values release];
}
- (void)getResponse:(NSMutableDictionary *)elements {
    questionText = [elements objectForKey:@"questiontext"];
    questionLabel.text = questionText;
    answerStrings = [elements objectForKey:@"answergroup"];
    numberOfAnswers = [answerStrings count];
    correctAnswer = [[elements objectForKey:@"rightanswer"] intValue];
    [self loadData];
}

- (void)submitAnswer:(int)aid {
    NSMutableArray *values = [[NSMutableArray alloc] initWithObjects: nil];
    [values addObject:@"mc"]; //type
    [values addObject:@"1"];  //sid
    [values addObject:@"1"];  //qid
    [values addObject:[NSString stringWithFormat:@"%d", aid]];  //answer
    [talker insertRecord:values];
    [values release];
}

- (void)buttonClicked:(id)sender {
    NSString *check;
    if([sender tag] == correctAnswer) check = @"Right";
    else check = @"Wrong";
    NSLog(@"%@ answer selected.", check);
    [self submitAnswer:[sender tag]];
}

#pragma mark - View lifecycle

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.

    answerStrings = [[NSMutableArray alloc] init];
    talker = [[DBTalker alloc] init];
    [talker setDelegate:self];

    //[self storeQuestionData];
    //[self loadQuestionData];
    //[self setNumberOfAnswers];
    //[self setAnswerStrings];

    abcArray = [[NSArray alloc] initWithObjects:@"A", @"B", @"C", @"D", @"E", @"F", nil]; //max
6 answers
```

```
    [self loadTalker];

    //NSString *s;
    //self.view = [[UIView alloc] initWithFrame:[[UIScreen mainScreen] applicationFrame]];

    questionLabel = [[UILabel alloc] initWithFrame:CGRectMake(241, 20, 676, 217)];
    questionLabel.text = @"Please wait, loading question...";
    questionLabel.font = [UIFont fontWithName:@"Arial" size:24];
    questionLabel.backgroundColor = [UIColor clearColor];
    questionLabel.textColor = [UIColor whiteColor];
    [self.view addSubview:questionLabel];
    //[questionLabel release];

}

- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
//    questionLabel = nil;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    // Return YES for supported orientations
            return YES;
}

@end
```

Appendix 8: Usability Questionaire

Directions: Please provide your opinion below. Do not put your name on this sheet. Your participation is voluntary; you can stop at any time or refuse to answer any question and will not be treated any differently by the researchers.

Q1. Was the application simple to navigate?

a) Yes

b) No

Q2. If the answer to the above question is 'No', please give your recommendation.

Q3. Was the font size and spacing appropriate to the application?

a) Yes

b) No

Q4. If the answer to the above question is 'No', please give your recommendation.

Q5. Did the system break during usage?

a) Yes

b) No

Q6. If the answer to the previous question is 'Yes', please give a brief description of the issue(s) faced and what you were doing at the time of system break?

Q7. How would you rate the responsiveness of the application on a scale of 1 to 5 where 1 is slow and 5 is fast?

a) 1 b) 2 c) 3 d) 4 e) 5

Q8. How easy was the application to use, where 1 is difficult and 5 is easy?

a) 1 b) 2 c) 3 d) 4 e) 5

Q9. How would you rate your experience using the application on a scale of 1 to 5 where 1 is poor and 5 is excellent?

a) 1 b) 2 c) 3 d) 4 e) 5

If you have additional comments, please leave them below.