

Virtual Clicker - A Tool for Classroom Interaction and Assessment

Nolan D. Glore

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Joseph G. Tront, Chair
Shelli B. Fowler
Lynn Abbott

December 2, 2011
Blacksburg, Virginia

Keywords: Tablet PC, Computer-aided instruction, Learner-centered instruction, Active learning, Classroom assessment

Virtual Clicker - A Tool for Classroom Interaction and Assessment

Nolan D. Glore

Abstract

Actively engaging students in the classroom and promoting their interaction, both amongst themselves and with the instructor, is an important aspect to student learning. Research has demonstrated that student learning improves when instructors make use of pedagogical techniques which promote active learning. Equally important is instructor feedback from activities such as in-class assessments. Studies have shown that when instructor feedback is given at the time a new topic is introduced, student performance is improved.

The focus of this thesis is the creation of a software program, Virtual Clicker, which addresses the need for active engagement, in-class feedback, and classroom interaction, even in large classrooms. When properly used it will allow for multi-directional feedback; teacher to student, student to teacher, and student to student. It also supports the use of digital ink for Tablet PCs in this interaction environment.

Table of Contents

List of Figures	vii
List of Code Snippets	ix
1 Introduction	1
2 Background	3
2.1 Pedagogy	3
2.1.1 Active Learning	4
2.1.2 Classroom Assessment	5
2.2 Related Work	6
2.2.1 Clickers	7
2.2.2 Tablet PC	7
2.2.3 Classtalk	9
2.2.4 ActiveClass	10
3 Feature Overview	11
3.1 Quizzes and Polls	12
3.2 Student Questions and Feedback	14
3.2.1 Student Status	15
3.2.2 Student Questions	16
3.3 Student Activity Monitor	17
4 Technical Overview	20
4.1 Technologies & Design Patterns	20

4.1.1	The .Net Framework	21
4.1.2	C#	22
4.1.3	WPF	22
4.1.4	MVVM	22
4.1.5	WCF	23
4.1.6	XML	24
4.1.7	Ink.....	26
4.2	Major Components	27
4.2.1	Common Code	28
4.2.2	Activity Tracker	29
4.2.3	Classroom Service	29
4.2.4	Quiz Creator Application	29
4.2.5	Quiz Grader Application	30
4.2.6	Instructor Classroom Application	30
4.2.7	Student Classroom Application	30
5	Quiz Document	31
6	Network Communication	35
6.1	Service Definition	36
6.1.1	Endpoint Address.....	37
6.1.2	Endpoint Binding.....	37
6.1.3	Endpoint Contract.....	38
6.1.4	Callback Contract	39
6.2	Service Discovery	40
7	Activity Monitor	44

8	Summary.....	48
8.1	Future Work.....	48
	References	50
	Appendix A: Developer’s Guide	57
	Code Project	57
	Version Control.....	59
	Logging	60
	Appendix B: User’s Guide	61
	Installation	61
	Installing .NET 4.0	62
	Installing Instructor Applications	62
	Installing Student Application.....	64
	Quiz Creator	66
	Starting Quiz Creator	66
	Adding Questions.....	67
	Deleting Questions.....	75
	Saving A Quiz	75
	Opening A Quiz	75
	Quiz Grader	76
	Starting Quiz Grader.....	77
	Opening Completed Quizzes	78
	Saving Graded Quizzes	79
	Exporting Grades.....	79
	Auto Grading.....	79

Navigation.....	80
Manually Grading.....	80
Instructor Host	81
Starting Instructor Host	81
Starting a Class.....	83
Stopping a Class	84
Giving a Quiz	84
Giving a Poll	85
Student Status	87
Student Activity.....	88
Student Questions	88
Student Client.....	90
Starting Student Client	90
Joining a Class	91
Leaving a Class	93
Setting Status.....	93
Taking Quizzes and Polls	94
Asking Questions.....	95
Voting on Questions.....	96
Conclusion	97

List of Figures

Figure 1 - Major components of Virtual Clicker application suite	28
Figure 2 - Quiz Document UML Class Diagram	32
Figure 3 - Service Announcement	41
Figure 4 - Discovery Probe	43
Figure 5 - Activity Tracker UML Class Diagram	45
Figure 6 - Instructor Applications Setup Wizzard	63
Figure 7 - Instructor Applications Install Location	63
Figure 8 - Instructor Applications Installation Complete	64
Figure 9 - Student Application Setup Wizzard	65
Figure 10 - Student Application Install Location	65
Figure 11 - Student Application Installation Complete	66
Figure 12 - Starting Quiz Creator	67
Figure 13 - Quiz Creator UI	67
Figure 14 - Quiz Creator Add Question	68
Figure 15 - Quiz Creator Added Question	68
Figure 16 - Default Multiple Choice Question	69
Figure 17 - Example Multiple Choice Question	70
Figure 18 - Default Multiple Answer Question	70
Figure 19 - Example Multiple Choice Question	71
Figure 20 - Default Essay Question	72
Figure 21 - Sample Essay Question	72
Figure 22 - Default Ink Question	73
Figure 23 - Quiz Creator Ribbon	74
Figure 24 - Example Ink Question	74
Figure 25 - Quiz Creator Application Menu	75

Figure 26 - Quiz File Extension.....	76
Figure 27 - Starting Quiz Grader	77
Figure 28 - Quiz Grader UI	78
Figure 29 - Completed Quizzes File Extension.....	79
Figure 31 - Starting Instructor Host	81
Figure 32 - Instructor Host First Run	82
Figure 33 - Instructor Host UI	83
Figure 34 - Instructor Host Start Classroom Session.....	84
Figure 35 - Give Quiz UI	85
Figure 36 - Give Poll UI	86
Figure 37 - Poll results UI.....	87
Figure 38 - Student Status	88
Figure 39 - Student Questions	89
Figure 40 - Save Questions	89
Figure 41 - Starting Instructor Host	90
Figure 42 - Student Client First Run	91
Figure 43 - Joining Broadcasted Classes.....	92
Figure 44 - Manually Joining Classes.....	93
Figure 45 - Setting Student Status	94
Figure 46 - Quiz Taker UI	95
Figure 47 - Asking Questions	96

List of Code Snippets

Code Snippet 1 - WCF Service Definition Sample	24
Code Snippet 2 - WCF Service Implementation Example	24
Code Snippet 3 - XML Sample	25
Code Snippet 4 - XML Serialization	26
Code Snippet 5 - InkCanvas WPF Sample	27
Code Snippet 6 - StrokeCollection Binding Sample	27
Code Snippet 7 - Endpoint Contract	38
Code Snippet 8 - Callback Contract	40
Code Snippet 9 - Service Discovery	42
Code Snippet 10 - Activity Tracker Native Methods	47

1 Introduction

Actively engaging students in the classroom and promoting their interaction, both amongst themselves and with the instructor, is an important aspect to student learning [1,2,3]. In class problem solving, group work, and discussion are good examples of this important teaching/learning process. At odds with this goal is increasing class sizes, especially in introductory courses. This can make it more difficult for instructors to obtain student feedback, resulting in lectures that lack interaction and fail to engage students [4]. There is also a perception that actively involving students in large classes is not appropriate. It often requires extra work, discouraging instructors from attempting classroom interaction [5].

Equally important to student learning is instructor feedback from activities such as in-class assessments. Research has demonstrated that when instructor feedback is given at the time a new topic is introduced, student performance is improved [6,7]. Classroom assessment also allows instructors to monitor students' learning, helping them to adapt lessons to focus on problem areas [8]. Unfortunately, assessment can be time-consuming and in-class assessment is particularly difficult in large classroom settings.

It's clear that engaging, assessing, and interacting with students is both important to student learning and also a difficult task for instructors. There have been multiple projects such as [9,10,11,12,13,14] which have attempted to solve one or more of these issues by introducing technology into the classroom. In general, these researchers have proven that technology can enable instructors to utilize pedagogical techniques that lead to increased student learning.

The focus of this thesis is the creation of a software program, Virtual Clicker, which addresses the need for active engagement, in-class feedback, and classroom interaction, even in large classrooms. When properly used it will allow for multi-directional feedback; teacher to student, student to teacher, and student to student. It also supports the use of digital ink for Tablet PCs

in this interaction environment. Pedagogical research and previous projects are used to support the design of each feature of this new tool i.e., functionality is added to the tool to improve sound pedagogical practices as opposed to simply inserting functions just because the programmer can.

2 Background

This chapter presents research on different pedagogical techniques that can help improve student learning. It then discusses research on current tools which can be used in the classroom to facilitate these techniques.

2.1 Pedagogy

In traditional classroom instruction, it is primarily the teacher's responsibility to convey course material, where the typical method used is lectures. This type of instruction is considered teacher-centered. An alternative to this, called student-centered instruction, shifts some of the responsibility to students. This is accomplished by replacing a portion of lectures with other techniques such as active learning experiences, assigning open-ended problems requiring critical or creative thinking, and using team-based learning activities [15,16]. Research has shown that using student-centered learning techniques results in increased motivation to learn, greater retention of knowledge, and a deeper understanding of the subjects taught [17,18].

Implementing these pedagogical techniques is not always easy. In fact, resistance from both students and other faculty is all but guaranteed. The problem is that while the promised benefits are real, they are neither immediate nor automatic. Students do not appreciate the increased responsibility and work required of them. Additionally, this new way of learning can leave students feeling threatened as they face the fear of the unknown.

The key to overcoming student resistance is communication. Instructors need to provide sufficient guidance along the way and should use positive reinforcement so students know they are on the right track. Instructors should also solicit feedback from students which can be used to reflect on the changes implemented and improve future instruction. It's also important for

instructors to understand the student-centered learning process and not get discouraged by inevitable setbacks [15,16].

It's important to mention that while technologies, such as the one presented here, can help teachers with student-centered instruction, technologies alone will not produce a student-centered classroom. Too often instructors will take hold of the latest teaching tool, believing it to be the magic bullet that will improve student learning. Instead they should begin from the view of putting pedagogy first and technology second [10].

2.1.1 Active Learning

Active learning is a term that refers to several methods of instruction but can be generalized as any instructional method that engages students in the learning process. An important aspect of this methodology consists of shifting the responsibility of learning onto the student. Types of active learning include: collaborative learning, in which students work together in small groups; cooperative learning, a more structured form of group work where students are assessed individually; and problem-based learning, which introduces real life problems at the beginning of an instruction cycle to provide context and motivation for the information that follows [2].

Just because typical lectures provide students with lots of information about a topic, this does not mean students are any more knowledgeable about it. Knowledge involves personal experience and requires learners to be able to comprehend and articulate what is being presented. The ways learners convert perceived information into knowledge can be grouped into two categories: active experimentation and reflective observation. Active experimentation involves actively engaging students while presenting information via discussions, examples, or application. Reflective observation involves introspection and observation of the information presented [19]. Neither type of learner benefits from traditional lecture because it neither engages active learners nor provides time for reflective learners to think about the information being presented. Therefore student participation should include opportunities for active

experimentation and reflective observation [20]. This means instructors should combine classroom discussion and problem solving with brief periods for reflection.

Another key area in which active learning can improve student education is by increasing attention span. Multiple studies suggest that when students are passive their concentration declines after 10-15 minutes [21,22]. This has a dramatic impact on retention of lecture material. Hartley and Davies found that immediately after a lecture students remembered 70% of information presented in the first ten minutes and only 20% of information presented in the last 10 minutes [23]. One implication is that instructors should use active learning activities to break up their lectures into 15 minute segments, keeping students engaged throughout the class period.

There are multiple studies that prove the effectiveness of active learning. Both [5] and [24] found significant improvement in exam scores when active learning techniques were used. Redish et al. showed that increases in learning are due to the nature of active engagement and not to extra time spent on a given topic [25]. Research also indicates active learning leads to better student attitudes and improvements in student's thinking and writing [17].

2.1.2 Classroom Assessment

Classroom assessment is beneficial to both instructor and student. It informs instructors about what students think and allows them to monitor students' understanding. It informs students about what and how their classmates think and improves their ability to monitor their own progress, helping them develop skills of self-evaluation [14,7,6].

Another crucial factor in student learning is addressing student misconceptions [1]. It is important that feedback be given frequently as new topics are introduced [6]. This can be accomplished through assessment, allowing students to revise any misunderstandings before moving on to new topics that are based on previously taught concepts. It also informs instructors, so they can modify lessons to fit the needs of their students.

Classroom assessment is a method that can be used to complement an active learning environment. Discussing responses with the class can create greater interaction in lectures [26,10]. This increases student knowledge and their involvement in their own learning, giving them more control over their education [7]. It also allows students to explore a variety of possible answers and thought processes, which can help illustrate how diversity of opinion is important [27,28]. In addition to facilitating active experimentation, classroom assessment leads to reflective observation. It does this by giving students time to review and consider new course content, apply their knowledge, and revise and improve their thinking [6,7,26].

Instructors that use classroom assessment have found students to report increased satisfaction, involvement, and understanding. Students also benefit from improved teaching because of instructors modifying and improving lessons based on assessment feedback [7]. Furthermore, research shows that final exam scores are improved when in-class assessment is used [6]. While exam scores are not necessarily the ultimate indicator of learning or understanding, the fact that there is some improvement shows that the in-class assessment does have some positive effect.

The only major disadvantage to using classroom assessment is that collecting and grading answers during class can be challenging. This can be a time consuming task; especially in sections with a large class size [6,14]. This is where technology can assist instructors, allowing them to distribute, collect, and grade questions with just a few clicks.

2.2 Related Work

This section covers previous research on various technologies that can be used to promote both active learning and classroom assessment. Both clickers and ClassTalk try to provide a solution for in-class assessment. ActiveClass is similar but also provides a way for students to ask questions and provide feedback to the instructor. ClassroomPresenter, DyKnow, and Microsoft Interactive Classroom leverage the Tablet PC, making it possible for instructors and students to share diagrams and drawings in real time during class.

2.2.1 Clickers

Clicker is a term used for a hand-held wireless transmitter used to submit responses to a receiver. They typically consist of a few buttons (e.g., a,b,c,d or 1,2,3,4) and can be used by instructors to ask a group of students multiple choice questions. Each student can use their clicker to individually answer the question which is collected by the receiver. Instructors can connect the receiver to a computer, allowing them to aggregate answers which can be shown to students in the form of a chart or histogram. Some statistical analysis may also be available to the instructor in an attempt to identify where deficiencies might lie.

Using clickers in the classroom supports active learning by increasing interaction and can be used to assess student learning. They can also be used to promote discussion among students after a histogram of answers is shown [9,10]. One disadvantage of this method is in the restriction of only asking multiple choice questions, which may limit their use. For example, in engineering classes instructors may want to test student's ability to fill out an electrical diagram or in a Japanese Kanji class instructors may want to ensure student's can draw the characters correctly. Additionally, Draper and Brown noted that typically 25-35% of students forgot to bring their clickers to class [10].

2.2.2 Tablet PC

Using Tablet PCs in the classroom can increase interaction and improve student learning, especially when paired with classroom software specifically designed for the Tablet PC. Its ability to support digital ink is helpful in classes that incorporate diagrams, drawings, and equations or when a topic is difficult to describe orally. For the instructor, these devices provide more flexibility than standard PowerPoint presentations, allowing them to adapt lectures to be more responsive to student interaction. For students, they improve the ability to organize, search, and review notes which helps improve learning [29].

2.2.2.1 Classroom Presenter

Classroom Presenter is a distributed Tablet PC based classroom interaction system. It allows instructors to convert a PowerPoint presentation into a slide deck that supports digital ink, enabling them to write on top of a slide. In class, instructors broadcast their deck and any ink annotations. This allows students to receive the deck and instructor's ink in real-time. Students can then add their own ink annotations, insert digital objects, or incorporate new slides into the individual student deck. Most importantly, Presenter also enables students to submit selected slides to the instructor; making it possible for instructors to display and discuss student solutions [11].

Classroom Presenter has been used in multiple classrooms at different institutions. Most research on its use has found it to support active learning, increase student participation and classroom discussion, and has shown it to have a positive impact on student learning [11,28,30]. Razmov and Anderson found that both attendance and exams scores were better in classes where Presenter was used [27].

2.2.2.2 DyKnow

DyKnow is an interactive education software system. It consists of two interoperable programs, DyKnow Vision and DyKnow Monitor, that can communicate together via a centralized server. It has several features that support active learning, collaboration, and classroom interaction. Students can use DyKnow to create an electronic notebook that supports digital ink for each class. These notebooks are a combination of a student's notes and an instructor's presentation which includes anything they write, type, or import. Instructors can use DyKnow to either create polls where students can respond to multiple choice questions or to request students to submit a solution to some problem. Instructors can then display these submissions to promote in class discussion or import one or more of these submissions into the presentation so students leave class with multiple solutions to the same problem. DyKnow also allows students to review their notes out of class and provides the ability to replay portions stroke-by-stroke; showing the steps taken to arrive at a given solution [12,31].

Research on DyKnow's use in the classroom has been positive. Surveys given to students after using DyKnow indicate they enjoy using it and it allows them to create better, more useful notes. Instructors feel it increases classroom interaction and facilitates quick feedback from students to faculty [32,33]. There has also been some indication of improved grades when DyKnow is used [12].

2.2.2.3 Microsoft Interactive Classroom

Microsoft Interactive Classroom is an add-in for PowerPoint and OneNote. It allows instructors to insert polling questions (e.g., multiple choice, yes/no, true/false) into presentations which students can answer and send back to the instructor. Answers are collated and presented to the instructor as a pie chart breakdown of the answers. It also allows instructors to share their slides plus any real-time ink annotations over the network which are captured by students using OneNote [34].

2.2.3 Classtalk

Classtalk is a classroom communication system created by Better Education, Inc. The system is somewhat similar to Clickers but provides more flexibility and options. It is comprised of multiple student devices networked to a central computer. Student devices are either a HP palm-top computer or a TI85 graphing calculator which enables up to four students to sign-on to a single device. Using the central computer, instructors can send tasks to the student devices and students can return the completed tasks back to the central computer. Tasks are created by the instructor either ahead of time or on an impromptu basis. They can be created in a variety of styles and are comprised of one or more questions, including any context needed to complete the task. Once tasks are completed, instructors can examine the responses, display the results in the form of a histogram, send feedback on the task to the students' devices, or store them for future analysis [14,35].

A study on the use of Classtalk over a two year period found it to be useful in promoting active learning and classroom assessment. It helped generate meaningful classroom discussions and gave students who are normally quiet, the confidence needed to voice their opinions in class. Students felt lectures were more enjoyable and helped them gain a better understanding of the subjects presented. Instructors indicated it was a useful classroom management tool and appreciated the immediate feedback it provided about everyone in the class [14].

2.2.4 ActiveClass

ActiveClass is an application for encouraging in-class participation using personal wireless devices such as Personal Digital Assistants (PDAs). It allows students to anonymously ask questions, answer polls, and send feedback to instructors. While using it, instructors have seen an increase and a broader range in student questions which they attribute to their anonymity. Additionally, it enables students to see others' questions, which they can rate. This has two effects. It reassures students that others have similar questions and enables instructors to focus their attention on the most misunderstood topics [13].

The background of which pedagogical techniques improve student learning described in this chapter were used to inform each decision made in regards to the Virtual Clicker feature set. Common pain points like large class sizes, improving classroom interaction, and making it easier to assess and obtain feedback from students was a primary focus. Each feature was designed to aid instructors in incorporating these techniques into the classroom. Research on previous tools was used to assist in determining what type of features are successful in helping instructors and students. It also revealed which type of features were of no use or actually hindered instruction and classroom interaction.

3 Feature Overview

In order to effectively teach complex subject matter, instructors need to be able to interact with their students and actively engage them using techniques such as problem solving, group work, and discussion. In-class assessment is a tool which instructors can use to incorporate both problem solving and group work into their lectures. They can also use results from these assessments to spur classroom discussion. Assessment is also important as it gives students the opportunity to reflect on what has been presented and allows instructors to tailor lessons based on student understanding. While engaging, assessing, and interacting with students is important to student learning, it's not always easy to accomplish and can become more difficult as class sizes increase.

The Virtual Clicker application suite was designed to address the need for active engagement, in-class feedback, and classroom interaction, even in large classrooms. It allows instructors to host a virtual class session which students can connect to. Once connected, the student's computer will listen for requests from the instructor and prompt students if any action needs to be taken. The connection also allows students to send information back to the instructor if needed.

Using Virtual Clicker, instructors can distribute virtual quizzes and polls to students who can answer the questions and submit them back to the instructor. The application also allows students to ask anonymous questions and allows instructors to monitor student status and activity levels. There are also two additional applications that can be used by the instructor to create quizzes ahead of class and quickly grade submitted quizzes after class.

During the creation of the Virtual Clicker application suite, the following goals were used to guide the features and design:

- Support in-class assessment, both formative and summative. It should automatically grade multiple choice type questions to allow for immediate feedback. Support for more open ended questions such as short answers and drawings should also be supported.
- Support anonymous polls that can be used to gauge student understanding or as a launching point for classroom discussion.
- Allow students to ask questions anonymously which can be seen and rated by other students.
- Allow students to provide feedback to the instructor about their current understanding of the topic being covered.
- Allow instructors to gauge student understanding by requesting feedback.
- Provide real-time information to instructors about student activity which can indicate focus on the lecture versus focus on note taking.
- Inform students when there is a shift of activity from note taking to paying attention to the instructor and vice versa.

The remaining sections of this chapter briefly describe the main features of the Virtual Clicker application suite and discuss how each feature relates to the goals outlined above.

3.1 Quizzes and Polls

The quizzing and polling features of the Virtual Clicker application suite are aimed at supporting in-class assessment. As stated previously in section 2.1.2, classroom assessment is a useful practice that benefits both the instructor and students. In addition to surfacing student misconceptions, classroom assessment can be used to create an active learning environment. This can be accomplished by incorporating quizzes as part of a group activity or by using the results of the quiz to launch classroom discussions. The challenge is obtaining the results from in class assessment in a timely manner. This is where Virtual Clicker's quizzing and polling features come in.

There are four different types of quiz questions supported by the application: multiple choice, multiple answer, essay, and ink. Of these types, only multiple choice and multiple answer can be used in polls. Multiple choice questions can include from 2 to 26 different possible choices with only one correct answer. Multiple answer questions include the same number of choices but can have multiple correct answers. Essay questions are short answer questions in which students need to type in text. Ink questions are ink canvases that allow students to draw with electronic ink using their Tablet PC. Ink questions allow instructors to draw some initial ink in cases where students would need to complete a diagram. They also allow instructors to draw "private" ink. This is a special ink color that will only be seen by instructors and will be taken out before being distributed to students. This allows instructors to create an answer key for easy grading.

Instructors can create quizzes before class using the Quiz Creator application which is part of the Virtual Clicker suite. It enables instructors to quickly create or edit a quiz by adding and removing questions, editing question and answer text, and adding/editing digital ink on ink question panels. Once complete, quizzes can be saved to the file system as XML documents.

In the classroom, Virtual Clicker enables instructors to virtually distribute electronic quizzes and polls in a matter of seconds to a large classroom of students. When finished, students can submit their answers back to the instructor's PC where they are collected. Polls are submitted anonymously while quizzes include information about who completed it. Instructors are then able to display answers if they so desire or save them to be graded later.

If instructors choose to display the answers, multiple choice and answer questions are auto graded and shown as a distribution of answers. Essay answers are just shown as text and ink answers are shown as digital ink diagrams. This ability to quickly show answer distributions and student diagrams can benefit student learning in two ways. First, displaying answers can be used to spur classroom discussions. This has been shown to create greater interaction in lectures [26,10]. Second, by discussing the correct and incorrect answers, student misconceptions are immediately addressed which is important to student learning [1,6].

If instructors choose to store completed quizzes for later grading, they can save the full set of completed quizzes plus the answer key into a single XML file. The Quiz Grader application supports opening this file to allow grading of individual quizzes. It provides an auto grade feature which automatically scores all multiple choice and multiple answer questions. Scoring of the remaining questions can be accomplished by easily navigating between questions and students by using the arrow keys and typing in a score. Each question includes a side-by-side display of both the student's answer and answer key for each grading. For multiple choice and multiple answer questions, a question distribution chart is also shown. The scores can be saved back into the XML file or they can be exported to a comma separated value (CSV) file.

The quizzing and polling features of the Virtual Clicker application uphold the first goal of this thesis, supporting in-class assessment, both formative and summative. They also affirm the second goal that allows for anonymous polling. With these two features, instructors gain the ability to quickly gauge student understanding, address any student misconceptions, and increase classroom interaction by discussing the results.

3.2 Student Questions and Feedback

The student questions and student status features of the Virtual Clicker application suite are aimed at increasing student/teacher interaction and allowing instructors to quickly gauge student understanding. As stated previously in section 2.1, it is crucial for student misconceptions to be addressed at the time new topics are introduced. It is also important for students to take an active role in the learning process and interact with the instructor and other students. While classroom assessment is one technique instructors can use to obtain feedback on student comprehension, this section discusses some additional, more light-weight methods to quickly obtain feedback from students.

3.2.1 Student Status

The most important question instructors should be asking of their students is: *do you understand what was just presented*. The answer to this question is key to student learning outcomes. The results of such assessments should be used by instructors to adjust their lessons to ensure the answer is yes. Despite the efforts of many instructors to confirm student comprehension, it's not always easy to determine if students are grasping what is being presented. The issue commonly lies within the way the question is presented. Often methods such as, *raise your hand if you don't understand, or are there any questions*, leave room for uncertainty and can put pressure on students to perform in front of their peers. This can leave the uncertain or anxious student reluctant to communicate their insecurity. There may be hesitation on part of some students as they wait to see others raise their hand or wait to find out if the next topic will clarify their misunderstanding. Shy and less confident students will remain silent for fear they will be seen as less intelligent [30,36].

The student status feature allows instructors to ask students if they understand the information that was just presented. Students respond by setting their status to one of three colors: red, yellow, or green. White is also a possible color which indicates a student has not yet set their status since the last time the instructor inquired. Individual student status is displayed as a color square next to the student's name in the list of students that is part of the instructor's application. The overall class status is also displayed as a color square in the overall status pane in the instructor application. The overall status is calculated as the median value of all the set statuses (i.e., median of red, yellow, and green status, white status is excluded).

This feature provides instructors with a quick, reliable way to gauge student comprehension. What makes this method more reliable than asking for a show of hands is the fact that a student's response is anonymous to everyone but the instructor. This results in students being more honest about their understanding of the material [36].

In addition to gauging student understanding, instructors can use this feature to add a break in the lecture and give students time to reflect on what they have just learned before they

answer. Giving students this time to reflect has been shown to help student learning [19,20]. This feature could also be used in other clever ways. For example, it could be used when in-class group work is given to indicate if the groups are ready yet.

3.2.2 Student Questions

It's important for students to ask questions to ensure they fully understand a topic before moving on to the next. Unfortunately, research shows that many students avoid asking questions, especially in large classes. It also indicates that students in engineering majors are even more likely to avoid asking questions [36,37].

The student question feature lets students electronically ask questions which remain anonymous to everyone but the instructor. By making the questions anonymous, much of the apprehension felt by students when asking questions in class is removed. Another potential benefit of anonymous questions, observed by Ratto et al., is a broader range of questions from students who otherwise would have remained silent [13]. This increase in questions means instructors can better gauge student understanding and spot any trends for particular topics and concepts that are not fully understood. This provides instructors with the feedback needed to help them modify their lessons to meet student needs.

3.2.2.1 Question Ranking

The expected increase in questions resulting from making them anonymous and submitting them electronically can also pose a problem for instructors. With this increase, it's possible for instructors to become overwhelmed by the amount of questions asked, especially in large classes, leaving them struggling to decide which questions to answer. This can be an issue even in a traditional classroom where students raise their hands. It's not always possible for instructors to answer everyone's question, so how do they decide who to call on?

The question ranking feature allows other students to rank a question. When a student asks a question, it's not only sent to the instructor, it's also sent out anonymously to all the other

students. Other students can then vote for the question if they also want to hear the answer. These votes are collected by the instructor's application which sorts the list of questions by votes, causing questions with the most votes to bubble to the top of the list.

This feature helps instructors gain a better understanding of the common problem areas, allowing them to clarify any misunderstandings and giving them the option to modify the lecture for future classes. It can also help students to see that others have questions similar to theirs which can help build student confidence.

While the question ranking helps instructors focus on the most common problems first, it's not always possible to address all the important questions in class. To solve this, instructors can save the list of questions to a text file and address any remaining issues outside of the classroom.

The student questions feature supports the third overarching thesis goal of allowing anonymous questions which can be seen and rated by other students. Together, the student questions and student status features support the fourth and fifth goals of allowing instructors to request feedback to gauge student comprehension and for students to provide feedback on their understanding. With these two features, instructors gain the ability to quickly gauge student knowledge and address any student misconceptions. Students gain the ability to take a more active role in their learning by influencing the lecture via questions and conveying their comprehension.

3.3 Student Activity Monitor

The student activity monitor feature of the Virtual Clicker application suite is intended to allow students and teachers to be informed of shifts in student attention. As previously discussed in section 2.1.1, keeping students' attention has a large impact on retention of material. It's not always easy, especially in large classes, for instructors to assess if students are paying attention

or if they are taking notes when they should. Students can easily get distracted and potentially miss a pertinent discussion or fail to write down important notes.

The objective behind the student activity monitor feature is to be able to track student activity levels which can then be used to inform students and teachers. The original idea was to use pen strokes as a way to measure these levels. Unfortunately, due to limitations in the APIs exposed to developers by the Windows operating system, there is no way for an application to monitor pen input when it doesn't have focus. Since the design of the Virtual Clicker suite is to be used as a companion application for other Tablet PC apps, including apps for note taking, it's not possible for it to track just pen input.

Instead of using only pen input to track activity, a combination of pen, mouse, and keyboard input can be used. This is possible because, in the Windows operating system, pen input eventually gets treated as mouse input and Windows allows applications to globally monitor both keyboard and mouse input. In the end this method may be more beneficial since student activity is measured as overall computer interaction regardless of which input method is used and will account for students who prefer to take notes using the keyboard.

Using this method of tracking all input activity, the student's Virtual Clicker application measures how much activity occurs over set intervals. It then sends this information to the instructor's application which collects the information and calculates a class median activity level. This overall class activity level is displayed to the instructor in the overall status pane. It is also sent out to each Student application which will inform students whenever there is a big shift in activity levels (e.g., from low to high or high to low).

The student monitoring feature supports the sixth and seventh goals of providing information about student activity and shifts in activity to both instructors and students. It allows both students and teachers to be aware of shifts in student focus from their computers to other tasks. Instructors can use the information to adjust their lecture by allowing students to finish writing notes. Instructors can also modify their presentation if they notice when student attention starts to dwindle. Students may also benefit by being persuaded to take notes when

most of the class started taking them or to stop what they're doing to pay attention to the instructor since the class activity dropped.

The features discussed in this chapter are designed to meet the goals of this thesis. Together these features should enable instructors to actively engage with their students. This is accomplished by providing instructors with the ability to quickly distribute and grade quizzes and polls while making it easy to monitor student status, important feedback, and overall class activity. Additionally students can take a more active role in their learning by asking questions, monitoring and voting on questions asked by others, and updating their status to inform the instructor when they don't understand the current topic.

4 Technical Overview

The Virtual Clicker application suite is designed to run on the Windows operating system and can leverage digital ink if run on a Tablet PC. It is completely written in the C# language and developed using the .Net Framework. The user interface (UI) of all the applications use the Windows Presentation Foundation (WPF) and were written using the Extensible Application Markup Language (XAML). All the applications follow the Model View ViewModel (MVVM) design pattern which separates UI markup from application logic. The network communication between the instructor and student applications is built using the Windows Communication Foundation (WCF). All the document formats used throughout the applications are encoded using the Extensible Markup Language (XML). Even though the application suite was designed to run on Windows, the XML encoded documents and the network communications used enables other non-Windows applications to be developed in the future which can integrate with the existing applications. The remainder of this chapter gives a brief introduction to the technologies used to implement the Virtual Clicker suite of applications, followed by an overview of the applications and components.

4.1 Technologies & Design Patterns

This section discusses the technologies used to implement the Virtual Clicker suite of applications, including a short explanation of each technology and the reasons why they were chosen. The following are the goals which influenced the technologies to use:

- Minimize the amount of coding needed by leveraging existing libraries
- Use simple and accepted design practices to ensure reliable and maintainable code.

4.1.1 The .Net Framework

The .Net Framework is a software framework written by Microsoft that supports building and running desktop applications and Web services on any version of Windows with the framework installed. The framework is made up of two key components, the Common Language Runtime (CLR) and the framework class library. The CLR is the execution engine of the framework which manages memory and exception handling for the application. The framework class library supports several programming languages (including C#, F#, and VB .NET) and contains a large collection of reusable types that can be leveraged by applications. Another advantage to the library is that code written in one language can be leveraged by any other language the framework supports [38,39].

For the Virtual Clicker application suite, .NET Framework 4 was chosen, which was the latest version available during development. It was chosen because of its large class library which supports many of the features needed by the application, including networking and ink. Version 4 was chosen due to its new features in networking, the Managed Extensibility Framework (MEF), and the *IObservable* interface; all of which were heavily used throughout the Virtual Clicker application suite [40].

In addition to version 4 of the .NET Framework, the Reactive Extensions (Rx) library was leveraged. The Rx library is created by Microsoft and ties in with the .NET Framework but has not officially been released as part of the framework. Rx is designed to allow for reactive programming which makes it easy to express data flows and the propagation of change. It is considered a superset of .NET's Language Integrated Query (LINQ) extensions, which exposes asynchronous and event-based computations as push-based observable collections via the new *IObservable* interface [41]. These extensions are heavily used within the Virtual Clicker application suite's view model code to deal with property change events that occur because of users interacting with the application.

4.1.2 C#

C# is an object-oriented and type-safe programming language. It has its roots in the C family of languages and its syntax has similarities to that of C, C++, and Java. C# supports component-oriented programming which allows software to be designed and separated into self-contained and self-describing packages of functionality [42]. The C# language was chosen because of its support in the .NET Framework and its familiarity to the developer.

4.1.3 WPF

The Windows Presentation Foundation (WPF) is a resolution-independent and vector-based rendering engine that utilizes DirectX; taking advantage of modern graphics hardware. It is released as part of the .Net Framework starting with version 3. WPF offers separation of user interface design from application behavior. The appearance of the application is defined using the Extensible Application Markup Language (XAML). The behavior and application logic is written in one of .NET's supported languages. Connecting the UI and the behavior is done using either code-behind, where the UI objects are directly manipulated in code, or with data binding, which binds properties of UI objects to properties defined in code [43,44].

WPF was chosen for its ability to separate appearance specific markup from application behavior. This made it easy to define common UI that could be reused throughout the application.

4.1.4 MVVM

Model View ViewModel (MVVM) is a design pattern, which is a general reusable solution used to separate UI markup from application logic. It is similar to the model view controller (MVC) pattern which is commonly used in application design. MVVM was introduced as a standardized way to leverage WPF features to simplify UI creation and is based on the Presentation Model. The idea is to create a fully self-contained abstraction of the View, which represents all the

data, state, and behavior of the View but without any reference to the controls used to render it. This abstraction is called the ViewModel. This allows the View to become just a projection of the ViewModel's state and removes the need for any code within the View. So updating the UI becomes as simple as updating a property on the ViewModel which will update the View via WPF data binding [45,46,47,48].

The choice to use the MVVM pattern was made for multiple reasons. One of the primary motives is it helps to further separate the visual UI from the application behavior, making it even easier to use WPF features like binding and data templates. This allows for improved ability to design reusable UI components. Another reason is it enforces using data binding as the only way to connect the View to the ViewModel, which ensures loose coupling between the two and removes the need for any code that directly updates the UI.

4.1.5 WCF

Windows Communication Foundation (WCF) is a software development kit (SDK) in the .NET Framework for building and deploying service-oriented applications that run on Windows. It enables developers to expose CLR types as services and to consume services as CLR types. The WCF runtime provides all the plumbing needed to send data as asynchronous messages from one application end point to another; eliminating the need to write low-level networking code [49,50].

Code Snippet 1 is an example definition of a service that echoes back the text that it receives. Notice that this is just a standard C# interface definition with a method that accepts and returns a string. The key is the *ServiceContract* attribute, which indicates the interface is a WCF service definition, and the *OperationContract* attribute, which indicates the Echo method is an operation the service can perform.

Code Snippet 1 - WCF Service Definition Sample

```
[ServiceContract]
public interface IEchoService
{
    [OperationContract]
    string Echo(string text);
}
```

Code Snippet 2 is an example implementation of the service definition in Code Snippet 1. One just needs to create a class that inherits from the defined interface. Implementing the echo method is as simple as just returning the passed in string parameter.

Code Snippet 2 - WCF Service Implementation Example

```
public class EchoService : IEchoService
{
    public string Echo(string text)
    {
        return text;
    }
}
```

WCF was chosen because it greatly reduced the amount of code needed to communicate between the Virtual Clicker instructor and student applications. The ability to define a service as an Interface that passes CLR types back and forth between clients was much more straightforward than other approaches. It also produces much more reliable and maintainable code. Another feature of WCF, the WS-Discovery protocol, was also used; making it easy for clients to detect each other at runtime.

4.1.6 XML

Extensible Markup Language (XML) is a simple set of rules for encoding data as text which can be easily consumed by machines. XML was designed to enable the exchange of a wide variety of data between different programs and computers. It is considered a meta language, a language for describing other languages, which allows it to be used to define limitless different types of documents. The XML specification is an international standard produced by the World Wide Web Consortium (W3C) [51,52].

XML documents are made up of elements which can have attributes and can contain child elements or character data. Below is a simple example of an XML document that represents a message. In the Code Snippet 3 example, the first line is a declaration about the document type, indicating it is an XML document that follows version 1 of the XML specification and uses a utf-8 character encoding. The second line contains the *Message* element which is the root element of the document. There can only be one root element in a valid XML document. The *Message* element has multiple child elements: *From*, *To*, *Subject*, and *Body*. These child elements contain no child elements of their own, instead each contains character data relating to the element which can be consumed by programs.

Code Snippet 3 - XML Sample

```
<?xml version="1.0" encoding="utf-8"?>
<Message>
  <From>Jane Doe</From>
  <To>John Doe</To>
  <Subject>Reminder</Subject>
  <Body>Don't forget our plans for the weekend!</Body>
</Message>
```

There are multiple reasons XML was chosen to encode documents produced by the Virtual Clicker application suite. One factor is it's an international standard which is supported by numerous programming languages across multiple operating systems. This makes it possible to write other application that integrates with the Virtual Clicker application suite which can run on any platform. An even more important factor is how easy it is to read and write C# objects to and from XML documents.

Code Snippet 4 shows a C# class that when serialized would produce the same XML as seen in Code Snippet 3. Here the *Message* class has the *DataContract* attribute applied to it which indicates instances of this class can be serialized. When the class gets serialized, only properties which have the *DataMember* attribute applied to them will be included. So in this example, the *From*, *To*, *Subject*, and *Body* properties will get serialized but the *DateSent* property will not. To actually cause an instance of the class to be serialized to XML, a *DataContractSerializer* object must be created for the *Message* class. Then this object's *WriteObject* method should be called, passing in a stream to write the XML to and the *Message* object to be serialize.

Code Snippet 4 - XML Serialization

```
[DataContract]
public class Message
{
    [DataMember]
    public string From { get; set; }
    [DataMember]
    public string To { get; set; }
    [DataMember]
    public string Subject { get; set; }
    [DataMember]
    public string Body { get; set; }
    public string DateSent { get; set; }
}

...
Message msg = new Message();
msg.From = "Jane Doe";
msg.To = "John Doe";
msg.Subject = "Reminder";
msg.Body = "Don't forget our plans for the weekend!";

DataContractSerializer xmlSer = new DataContractSerializer(typeof(Message));
xmlSer.WriteObject(fileStream, msg);
...
```

4.1.7 Ink

Traditionally, in order to build applications for Tablet PCs that use digital ink, developers needed to use the Tablet PC SDK. With WPF, digital ink is included into the core of the framework. Adding support for ink in a WPF application is as simple as adding the *InkCanvas* control. As users draw on the control, the ink is stored as a collection of strokes. These strokes can then be serialized into a byte stream which can be saved and re-loaded later [44,53,54]. This greatly simplifies the programming needed to add ink support to an application, making its use identical to using any other control that accepts input.

The following code snippet is an example of XAML markup that defines an *InkCanvas*.

Code Snippet 5 - InkCanvas WPF Sample

```
<InkCanvas Background="Gray"
            Width="400"
            Height="200"
            Strokes="{Binding Path=Question.Ink}"/>
```

Notice the strokes property has a binding with a path equal to "Question.Ink". This means bind the strokes property of the *InkCanvas* to the Ink property of the current *Question* object. What this does is let WPF do all the work of collecting and removing strokes as the user interacts with the canvas and stores those strokes in the property indicated. On the C# side, the Ink property is just a *StrokeCollection* object that can be used to access the ink programmatically which allows the ink to be saved to a file. The following code snippet is an example of the property which the *InkCanvas* Strokes property binds to.

Code Snippet 6 - StrokeCollection Binding Sample

```
public class InkFormQuestion : Question
{
    ...
    public StrokeCollection Ink { get; set; }
    ...
}
```

4.2 Major Components

The Virtual Clicker application suite is made up of four different applications: the instructor in class application, the student in class application, the quiz creator, and the quiz grader. There is also three different dynamically linked libraries (DLLs) which contain stand alone functionality which is either used in more than one application or has the potential to be used in other applications in the future. Figure 1 is a block diagram of these seven components and indicates which applications leverage each DLL and how the instructor and student app communicate.

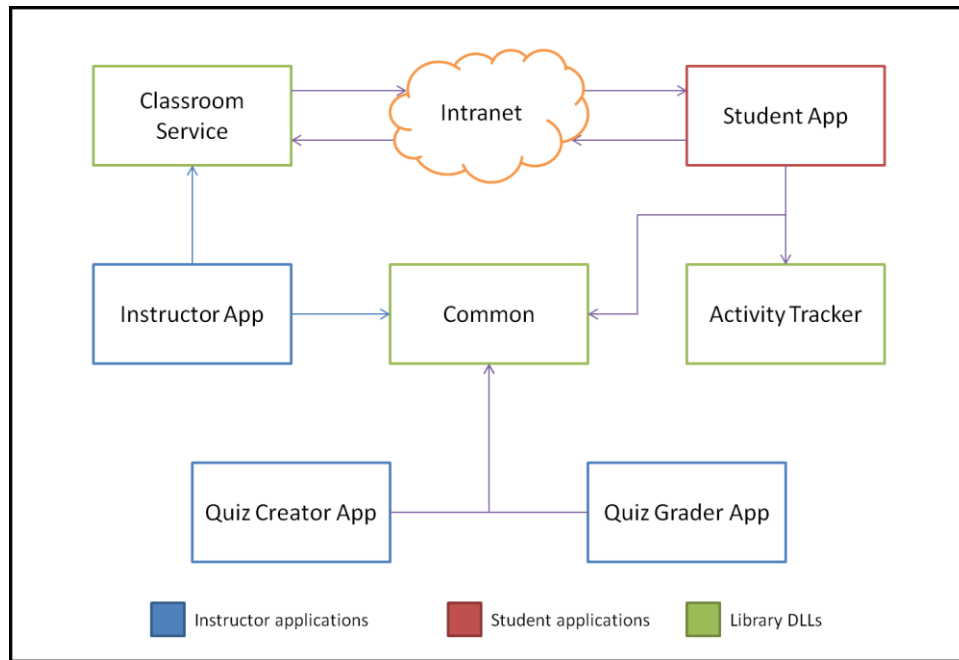


Figure 1 - Major components of Virtual Clicker application suite

4.2.1 Common Code

Since all of the applications share similar logic and have some UI in common, a large portion of the Virtual Clicker code base was broken out into a separate common DLL so the code could be shared between all the applications. One of the main parts of this DLL is the quiz object model. It contains all the code to represent a quiz and all the possible types of quiz questions including the logic to save and load these objects to and from XML documents.

Another major part is the common UI controls, views, and view models. This includes the UI for representing a quiz and the ink menu controls used for editing ink questions. One issue with having common UI to represent a quiz is for some applications the quiz needs to be fully editable but in others only the answers should be editable. To accomplish this, the quiz UI uses WPF control templates to display the proper UI based on mode.

The DLL also contains other common helper code, including code for application management, icons and images, logging, and MVVM binding. The advantage of breaking common components out into their own DLL is a reduction in code duplication which can lead to a more consistent experience across applications and reduce potential bugs caused by code being updated in one place but not another.

4.2.2 Activity Tracker

The activity tracker library provides the ability to track activity which is represented as the total active time or the percentage of active time over a given period. Active time is considered any time the user is interacting with the computer via keyboard, mouse, or pen. The library monitors interaction by registering system wide hooks with the operating system which allows it to track all input into the system. The Student Classroom application uses the activity tracker to monitor and periodically report activity levels to the instructor's application.

4.2.3 Classroom Service

The classroom service library is built using the WCF SDK. It contains the definitions of the interfaces used for communication between the Instructor Classroom application and the Student Classroom application. It also contains all the code for hosting the service which allows clients to connect. The Instructor Classroom application uses this library for hosting Virtual Clicker classroom sessions.

4.2.4 Quiz Creator Application

The quiz creator application is used for creating and editing quizzes. It enables instructors to easily add and remove questions to and from the quiz. It also allows editing question text and possible choices for multiple choice and multiple answer questions. For ink questions, it

provides basic tools like a pen, eraser, and highlighter and offers multiple ink colors, including an instructor only color which is hidden from students.

4.2.5 Quiz Grader Application

The quiz grader application can be used by instructors and teaching assistants to quickly grade submitted quizzes. It supports auto grading of multiple choice questions and side by side comparison of student submissions to the answer key, which is helpful for ink questions. It also provides an answer distribution chart for multiple choice and multiple answer questions. Users can quickly navigate between questions and submissions and can either save the graded quizzes back to the quiz collection file or they can export the grades to CSV.

4.2.6 Instructor Classroom Application

The Instructor Classroom application is used for hosting class sessions. It allows instructors to start a virtual classroom session so students can connect. Once a class session is started, the application displays information about connected students and student questions. For each student, an icon is displayed next to their name to indicate their status. There is also an overall status pane which indicates the median class status and the median class activity level. Using the application, instructors can start a quiz or a poll and request status from connected students.

4.2.7 Student Classroom Application

The Student Classroom application enables students to join class sessions hosted by instructors. It allows them to see and connect to any broadcasted classes or manually enter connection settings. Once connected, students can set their status, ask questions, or vote on questions from other students. The application will prompt students to complete quizzes and polls when an instructor starts them. They will also be informed of any shifts in classroom activity levels.

5 Quiz Document

The Virtual Clicker application suite had a need to define a document object model representing both quizzes and polls which would allow the documents to be modified in code. Additionally this object model would have to support passing these documents between instructor and student computers over the network and allow for these documents to be saved to and read from files. Once WCF was chosen for network communication, the logical choice was to use .NET's *DataContractSerializer* class [55]. This class provides methods for serializing and deserializing objects and is used by default in WCF for sending objects over the network. It also supports serializing and deserializing objects to and from XML which can be saved to a file.

In order to make classes compatible with the *DataContractSerializer*, they need to be marked with the *DataContract* attribute and all members of the class that should be serialized need to be marked with the *DataMember* attribute. This makes creating classes that can be saved to XML very easy. For more information on how *DataContractSerializer* works see section 4.1.6.

Figure 2 shows a class diagram of the quiz document object model. The *CompletedQuizzes* class represents a set of completed quizzes. It contains both the original quiz that was created by the instructor, including the answer key, and a collection of quizzes that were submitted by students. It also has methods for saving and loading this set of quizzes to and from an XML file.

The *Quiz* class is a model of a single quiz document. It can represent quizzes created by instructors which contain the answer key and quizzes that have been completed by a student. The class has properties for tracking the quiz author, name, instructions, who it was taken by, if it was graded and the score. It contains the collection of questions which make up the quiz. There are also methods for clearing answers and saving or loading the quiz to or from an XML file.

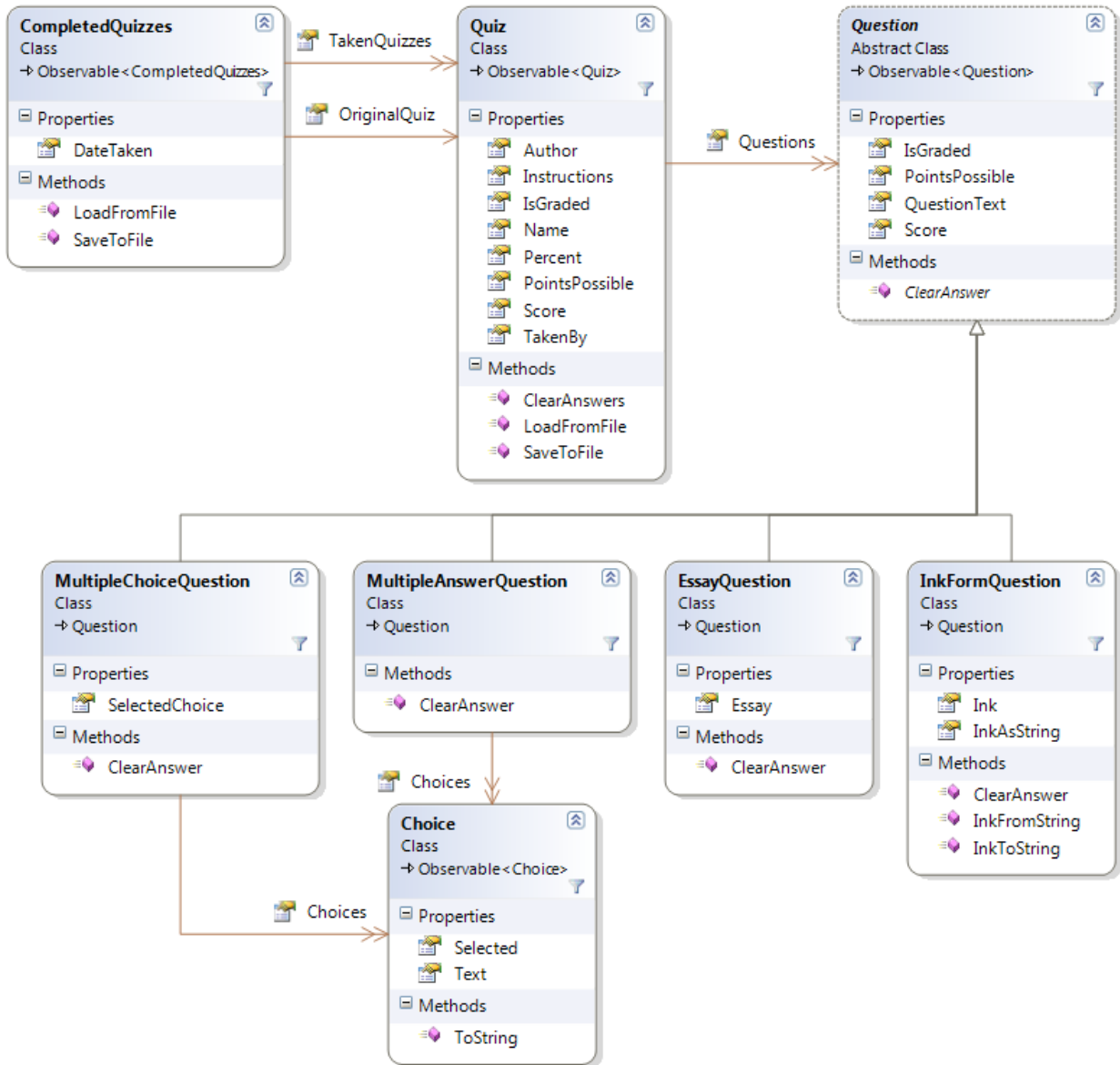


Figure 2 - Quiz Document UML Class Diagram

There are four classes which represent the different question types: *InkFormQuestion*, *EssayQuestion*, *MultipleAnswerQuestion*, and *MultipleChoiceQuestion*. They all extend from the abstract *Question* class which defines common properties like the question text, points possible, if the question is graded, and the score. The *Question* class also defines an abstract method, *ClearAnswer*, which is implemented by each subclass. This method is used to clear the

answers from an instructor created quiz before it is sent out over the network for students to complete.

The *InkFormQuestion* class represents questions that have a canvas for drawing digital ink. It has two properties which are used for tracking the drawn ink. The *Ink* property is of type *StrokeCollection* which represents a collection of pen strokes. The Virtual Clicker applications use this property to keep track of ink as it is drawn. This is accomplished by using WPF's data binding feature to bind the WPF *InkCanvas* control's *Strokes* property to the *InkFormQuestion*'s *Ink* property. This causes the *Ink* property to be updated with the current set of pen strokes every time a user adds or removes ink.

Unfortunately the *StrokeCollection* class can't be serialized to XML so the *InkAsString* property was added to support saving the ink to and from XML. This property is a Base64 encoded string representing the set of pen strokes in the *StrokeCollection* object. When serializing an *InkFormQuestion* object, just before serialization starts, the *InkToString* method is called to save the *StrokeCollection* object to a byte array in memory and then convert the byte array to a string using Base64 encoding. The resulting value is stored in the *InkAsString* property which will be serialized to XML. When deserializing an *InkFormQuestion*, first the *InkAsString* property will be deserialized from the XML. Then the *InkFromString* method is called to convert the string to a byte array using Base64 decoding and create a *StrokeCollection* object from the byte array.

The *EssayQuestion* class represents long answer type questions. It contains just a single property, *Essay*, which stores the answer text. Both the *MultipleAnswerQuestion* and *MultipleChoiceQuestion* classes represent questions that have multiple answers to choose from. The difference is the former supports multiple answers to be selected while the latter only supports one answer to be selected. Both classes have a *Choices* property which is a collection of *Choice* objects which represent the set of answers for the question. The *Choice* object has two properties: *Text*, which contains the answer text, and *Selected*, which indicates if the answer is selected or not. The *MultipleChoiceQuestion* class also has a *SelectedChoice* property which indicates the selected answer, since only one can be selected.

The quiz document object model defined here supports all the requirements of the Virtual Clicker application suite. It allows quizzes and polls to be easily manipulated in code and supports data binding to WCF controls so the object model is always up to date with user input. It also supports serialization so documents can be sent between computers over the network and they can be saved and opened to and from XML files.

6 Network Communication

One of the most important aspects of the Virtual Clicker application suite is its ability to transmit data between the Instructor and Student applications. Therefore it is important the network programming piece be done correctly since network communication plays a key role in the functionality of these applications. However, network programming can be difficult to get correct.

Originally the application's network programming was written using the .NET Framework's socket APIs [56]. The client-server model was used where the Student application is the client and the Instructor application is the server. The Transmission Control Protocol (TCP) was used to reliably transfer data between the two applications.

An API was written to try to encapsulate all the functionality needed for communication and to conceal all the low level networking code. Writing the application's programming on top of the socket API proved to be difficult. One reason is there are a lot of error conditions and different states that need to be handled in order to create a reliable API. A bigger issue was the large amount of code that needed to be written.

A large part of this networking API was written and then basic test apps were built on top of it to ensure it worked as expected. The API worked for the basic communication between the applications. Unfortunately, it proved to have issues and didn't properly handle all error conditions. Also, as the test apps were written, design flaws arose that required changes to the underlying design. This showed that it's not easy to write a reliable and flexible networking API.

Due to the issues that arose from using the socket APIs, research was performed to find a better alternative. This led to the use of the .NET Framework's WCF SDK, which abstracts the low level network programming and greatly reduces the amount of code that needs to be written. This led to much more reliable network communication.

6.1 Service Definition

WCF is intended to be used for building service-oriented applications. For the Virtual Clicker application suite, the Instructor application hosts the service which allows the Student application to connect. In this way the two apps can communicate in a manner similar to the client-server model.

In WCF, all communication with a service occurs through the endpoints defined by the service [57]. Endpoints are C# classes that can be configured in code or in an XML configuration file. They describe the way clients can connect and communicate with the service and are made up of four properties:

- An address that indicates where the endpoint can be found
- A binding, which is a C# class made up of a sequence of binding elements that each defines a step in the process of sending and receiving messages. Some of the common steps which can be configured include:
 - The transport protocol to use (e.g., TCP, UDP, HTTP)
 - The encoding to use (e.g., text or binary)
 - Any message security (e.g., SSL)
- A contract, which is a collection of operations that specify how clients can communicate with the endpoint. A contract will designate:
 - What operations can be called by a client
 - The format of the data exchanged
 - The type of input parameters and which ones are required for each operation
 - What type of response the client can expect for each operation
- A set of behaviors which specify implementation details of the endpoint and allow modification of WCF provided bindings. For example, behaviors can be used to:
 - Add discovery behavior which allows clients to find and obtain connection information about services running on the network
 - Add custom serialization/deserialization of data

- Add custom logging

Therefore the endpoint of the Instructor application's service defines how Student applications can connect to the service and how data is exchanged between the two applications.

6.1.1 Endpoint Address

The service address is made up of an IP address and a port number. The port number is determined at runtime using .NET's socket APIs to select an available port number between 1024 and 5000. By choosing the port number at runtime instead of hard coding it, more than one Instructor application can run on the same computer at once. The IP address is also determined at runtime by enumerating over all the network connections the PC has and selecting the first active connection.

Ideally the service would be hosted on all active network connections to ensure clients can connect regardless of which network they are on. Unfortunately, WCF doesn't support binding multiple addresses using the same endpoint binding type. So if a PC has multiple network cards, only the IP from one of them is selected. While this may block clients from automatically detecting the service, it shouldn't block them from connecting in most network settings and is mitigated by the fact that clients can connect using manual connection settings.

6.1.2 Endpoint Binding

The binding for the Virtual Clicker service uses WCF's built in *NetTcpBinding*. This binding uses TCP for transport, encodes data in binary, and uses SSL for security. It was chosen because it has the least network overhead and the best performance of all the built in network bindings. The down side to using the *NetTcpBinding* is that it's not interoperable with anything other than another WCF client. This is mitigated by the fact that a service can define multiple endpoints. So if the Instructor application ever needed to be interoperable with non .NET applications, another binding could be added.

6.1.3 Endpoint Contract

In WCF, a contract can be defined by creating an interface which contains methods which the service exposes as operations. The *ServiceContract* attribute needs to be applied to the interface to indicate it is defining a WCF endpoint contract. This attribute accepts multiple parameters which configure how the service deals with sessions and how the service can communicate back to the client if needed. The *OperationContract* attribute needs to be applied to each method which the service should expose. This attribute accepts multiple parameters which indicate if any methods must be called first or last.

Code Snippet 7 - Endpoint Contract

```
[ServiceContract(SessionMode = SessionMode.Required,
    CallbackContract = typeof(IVirtualClickerServiceCallback))]
public interface IVirtualClickerService
{
    [OperationContract(IsInitiating = true, IsTerminating = false)]
    Guid JoinClass(Student student);

    [OperationContract(IsInitiating = true, IsTerminating = false)]
    Guid JoinClassSecure(Student student, string password);

    [OperationContract(IsInitiating = false, IsTerminating = true)]
    void LeaveClass(Guid connectionId);

    [OperationContract(IsInitiating = false, IsTerminating = false)]
    void AskQuestion(Guid connectionId, StudentQuestion question);

    [OperationContract(IsInitiating = false, IsTerminating = false)]
    void SubmitQuiz(Guid connectionId, Guid quizId, Quiz quiz);

    [OperationContract(IsInitiating = false, IsTerminating = false)]
    void SetActivity(Guid connectionId, double activityLevel);

    [OperationContract(IsInitiating = false, IsTerminating = false)]
    void SetStatus(Guid connectionId, StudentStatus status);

    [OperationContract(IsInitiating = false, IsTerminating = false)]
    void VoteForQuestion(Guid connectionId, Guid questionId);
}
```

Code Snippet 7 contains the definition of the *IVirtualClickerService* interface which defines the Virtual Clicker service endpoint contracts and contains all the operations supported by the service. The *ServiceContract* attribute applied to the interface specifies two parameters: *SessionMode* and *CallbackContract*. The *SessionMode* parameter is set to *Required*, which

indicates the service needs to maintain session information for each client. The *CallbackContract* is set to be of the type *IVirtualClickerServiceCallback*, which indicates the *IVirtualClickerServiceCallback* interface defines the operations exposed by clients that the service can call.

Every method defined in the *IVirtualClickerService* interface has the *OperationContract* attribute applied to it, indicating each method is an operation the Virtual Clicker service exposes to clients. Two methods, *JoinClass* and *JoinClassSecure*, set the *IsInitiating* parameter of the *OperationContract* attribute to true. This indicates that one of these methods must be called before any other method can be called and that these methods can only be called once. The *LeaveClass* method sets the *IsTerminating* parameter of the *OperationContract* attribute to true. This indicates that no other methods can be called after this one and that the connection is closed after this method is called. The remaining methods set both *IsInitiating* and *IsTerminating* to false which indicates they can be called in any order after *JoinClass* or *JoinClassSecure* is called but before *LeaveClass* is called. By setting these two parameters, it is ensured that clients behave according to these rules which makes it easier to program the service logic since these error cases don't have to be handled explicitly by the programmer.

6.1.4 Callback Contract

The endpoint contract defined in section 6.1.3 defines how the Virtual Clicker Student application can communicate with the Instructor application. There is also a need for the Instructor application to be able to communicate back to the Student application. WCF supports this using duplex communication. This type of two-way communication requires connecting clients to expose a callback contract which is defined by the service.

A service defines a callback contract by creating an interface which contains methods the client should expose as operations. The *OperationContract* attribute needs to be applied to each method which should be exposed. The service also needs to define the interface as the callback contract by setting the *CallbackContract* property of the *ServiceContract* attribute which is

applied to the interface that defines the service's endpoint contract. This can be seen in the Virtual Clicker endpoint contract in Code Snippet 7.

Code Snippet 8 contains the definition of the Virtual Clicker callback contract. This contract allows the Instructor application to start and stop quizzes and polls, request student status, send information about student questions and activity, and stop the virtual classroom session.

Code Snippet 8 - Callback Contract

```
public interface IVirtualClickerServiceCallback
{
    [OperationContract]
    void StopClass();

    [OperationContract]
    void RequestStatus();

    [OperationContract]
    void StartQuiz(Guid quizId, Quiz quiz);

    [OperationContract]
    void StopQuiz(Guid quizId);

    [OperationContract]
    void StartPoll(Guid pollId, Quiz poll);

    [OperationContract]
    void StopPoll(Guid pollId);

    [OperationContract]
    void QuestionUpdate(List<StudentQuestion> questions);

    [OperationContract]
    void ActivityUpdate(double activityLevel);
}
```

6.2 Service Discovery

To make it easier for students to connect to hosted virtual classroom sessions and avoid the need to manually enter connection information, there needs to be a way for the Student application to find active sessions on the network and detect when new ones are started. The WCF's Discovery APIs offers ways this can be accomplished using the Web Services Dynamic Discovery (WS-Discovery) protocol. This protocol is a specification which defines a standard for

how clients can find services on the web at runtime and how services can broadcast their existence, all using multicast messaging [58].

The Discovery API supports two different modes: Managed and Ad-Hoc. In Managed mode clients can query a centralized server which maintains information about available services. In Ad-Hoc mode, there is no centralized server. Instead multicast messages are used to find services [59]. The Virtual Clicker applications use Ad-Hoc mode.

There are two aspects to WCF Discovery that need to be implemented: service announcement and service discovery. Service announcement works by having the Virtual Clicker service send out a multicast service announcement message, as seen in Figure 3. This message contains information about the service and how clients can connect. Virtual Clicker clients need to listen for service announcements which are sent from a Virtual Clicker service.

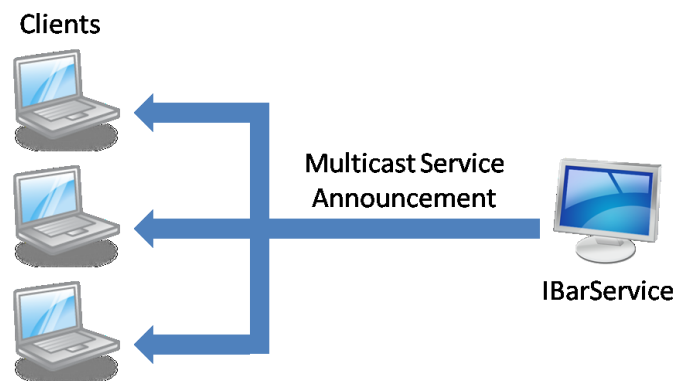


Figure 3 - Service Announcement

Code Snippet 1 shows how a service announcement is added to the Virtual Clicker service. First an *EndpointDiscoveryBehavior* is created and a custom XML extension is added to it which contains information about the hosted virtual classroom session (e.g., class name, instructor's name). Second a *ServiceDiscoveryBehavior* is created which has a *UdpAnnouncementEndpoint*. Finally both these are added to the service's behaviors. This causes the service to send out a multicast message at startup which contains connection information and the custom XML that was added to the *EndpointDiscoveryBehavior* object.

Code Snippet 9 - Service Discovery

```
/* ----- Service Code ----- */

// Create a discovery behavior to pass class information
EndpointDiscoveryBehavior discoveryBehavior = new EndpointDiscoveryBehavior();
XElement discoveryExtension = new XElement("Root");
discoveryExtension.Add(new XElement("ClassName", className));
discoveryExtension.Add(new XElement("Instructor", instructor));
discoveryExtension.Add(new XElement("Password", password != null));
discoveryBehavior.Extensions.Add(discoveryExtension);
service.Description.Endpoints[0].Behaviors.Add(discoveryBehavior);

// Add the service announcement behavior to the service
ServiceDiscoveryBehavior discovery = new ServiceDiscoveryBehavior();
discovery.AnnouncementEndpoints.Add(new UdpAnnouncementEndpoint());
service.Description.Behaviors.Add(discovery);

// Add an endpoint to listen for discovery messages
service.AddServiceEndpoint(new UdpDiscoveryEndpoint());

/* ----- Client Code ----- */

DiscoveryClient discoveryClient = new DiscoveryClient(new UdpDiscoveryEndpoint());
discoveryClient.Find(new FindCriteria(typeof(IVirtualClickerService)));
```

Since a service announcement is only sent when a class session is started, clients need a way to find existing class sessions. This is accomplished using service discovery. This works by having the client send out a multicast probe on startup, as seen in Figure 4, which indicates it's looking for any Virtual Clicker services. The Virtual Clicker service is configured to listen for these probe messages and when one is received, it responds directly to the client, sending information about the service and how clients can connect.

To enable the service to listen for discovery probes, the service needs to add a WCF provided *UdpDiscoveryEndpoint* object to its list of endpoints, as shown in Code Snippet 9. The client needs create a *DiscoveryClient* object and call its *Find* method, passing in the type of service to find. In this case the client passes the interface, *IVirtualClickerService*, which defines the Virtual Clicker service.

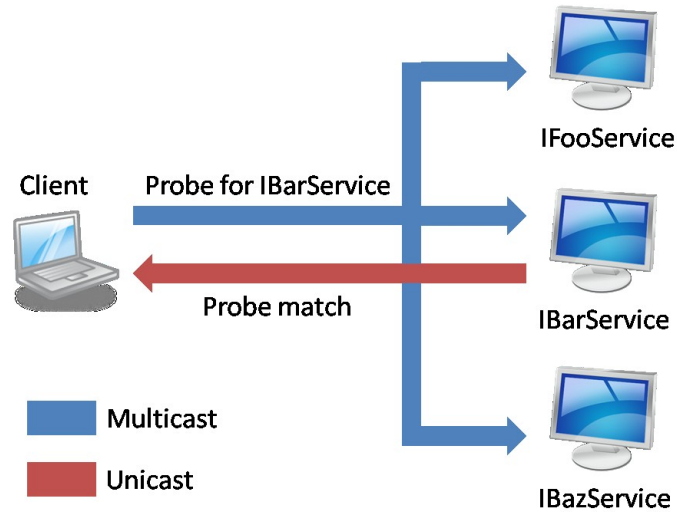


Figure 4 - Discovery Probe

The one downside to using the Ad-Hoc mode of the Discovery API is there is no guarantee discovery messages will be received by clients who are not connected to the same network router as the service. This is because it is common practice to configure network routers to not forward multicast messages across different networks. This issue is mitigated by the fact the Student application allows connection information to be entered manually.

Using WCF to handle all the network communication between the Virtual Clicker applications simplified the network programming and greatly reduced the amount of code needed. This resulted in a much more reliable product and a more maintainable code base. Also, by using industry standards, all the network communication can be made fully interoperable with other non-Microsoft clients in the future.

7 Activity Monitor

Two of the goals of the Virtual Clicker application suite are aimed at providing both instructors and students with real-time feedback on student activity levels. The original idea was to use pen strokes to indicate how active a student is. Due to limitations in the Windows API set this isn't possible. Instead, durations of pen, mouse, and keyboard input is tracked to be able to report a student's activity as a percentage of the time spent actively interacting with their computer.

The *ActivityTracker* class is what provides the ability to track the amount of time a user is active. Active time is considered to be the amount of time a user takes to type a word, drag their mouse, or create an ink stroke. The time spent typing a word is calculated by monitoring consecutive key presses, other than the space, period, or enter keys, which occur within a timed threshold. Since the Windows API set doesn't allow distinguishing between mouse and pen input at the global input level, mouse and pen input are treated the same. So the time spent creating a stroke is calculated as the time between a mouse left button down event until a mouse left button up event, which maps to the start and stop of a pen stroke. After some testing of typical keyboard/pen input while taking notes and typical mouse usage, the increase in the percentage of active time due to mouse was minuscule in comparison.

Figure 5 shows a class diagram of the classes used to monitor activity. The *ActivityTracker* sets up everything needed to do the monitoring and keeps track of the total active time. It has properties for the time tracking started, the active time, and the active time as a percentage of total time. It provides a *Reset* method to clear the collected activities and restart tracking. It also fires an event every time a new activity time is detected. The class also implements the *IDisposable* interface, which is used in the common C# design pattern for cleaning up unmanaged resources that aren't handled for the programmer by the .NET runtime [60]. This is needed since unmanaged low level system hooks are used to monitor user input and these

need to be properly released. The *ActivityTracker* class uses two other classes to monitor user input: *WordTracker* and *StrokeTracker*.

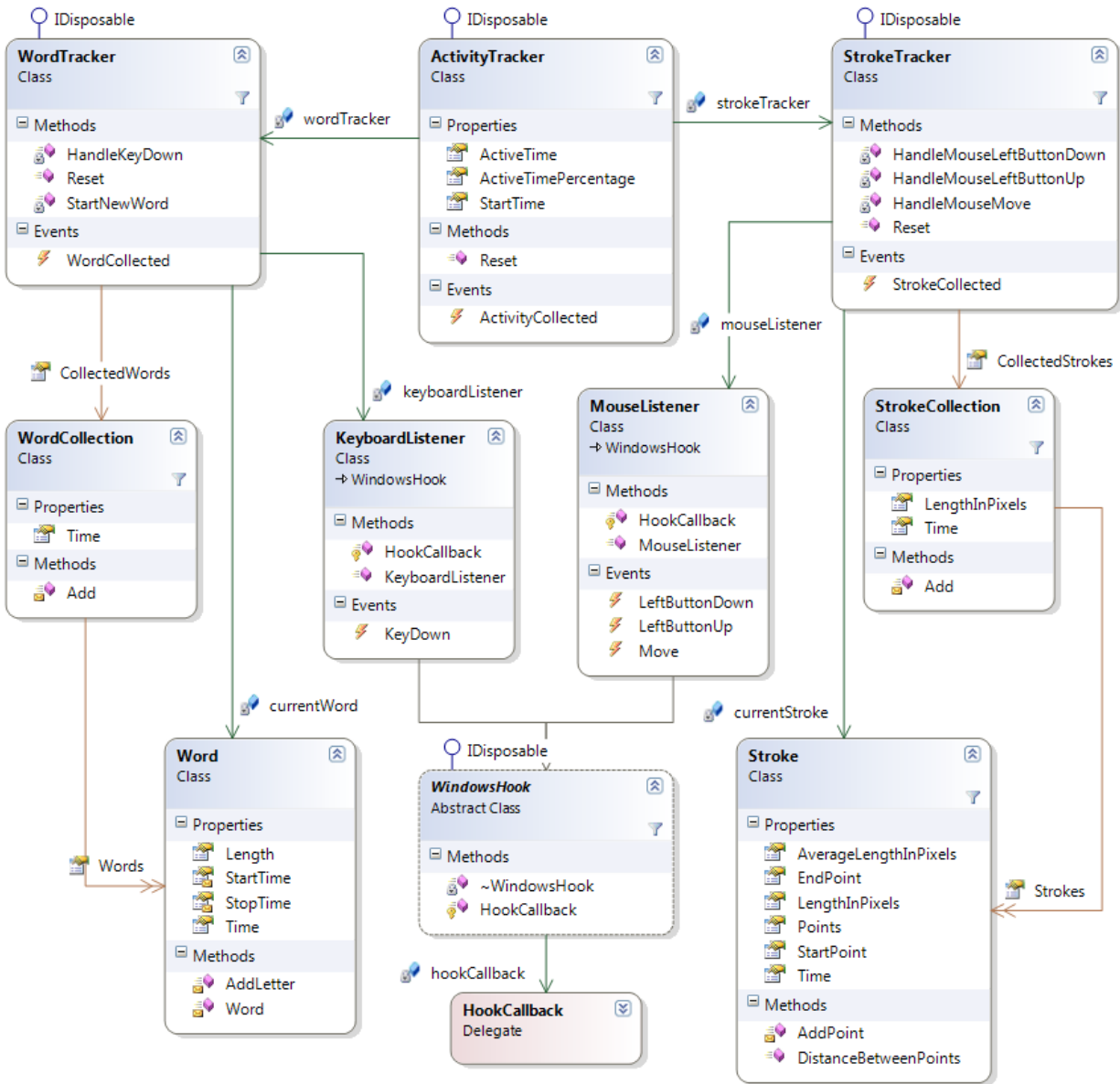


Figure 5 - Activity Tracker UML Class Diagram

The *WordTracker* class is used for monitoring keyboard input and tracks the time users spend typing single words. It does this by measuring the time span between the detection of special characters: space, return, and period. It also considers a word to be finished if a user pauses for

more than a certain time period, currently two seconds. This prevents counting the time a users spends contemplating what to type as active time. The class exposes one event which is fired when a new word is entered and contains information about the time spent entering the word.

The *StrokeTracker* class is used for monitoring pen and mouse input and tracks the time users spend creating ink strokes. It does this by measuring the time span between the detection of pen down and pen up events. The class exposes one event which is fired when a new stroke is entered and contains information about the time spent creating the stroke.

The *KeyboardListener* and *MouseListener* classes are used by the *WordTracker* and *StrokeTracker* respectively. They provide the ability to listen to the low level system input in real-time. Both classes extend from the *WindowsHook* class. This class provides the base code to create Windows hooks which allow an application to listen to all low level system messages before they reach the target window, regardless of which application has focus [61].

Using Windows hooks is the only way to monitor input across the system but the .NET framework doesn't support using them directly. To get around this limitation, the *WindowsHook* class uses Platform Invocation Services (PInvoke). PInvoke allows managed .NET code to call unmanaged native functions that are implemented in a DLL [62]. To use PInvoke, the unmanaged methods that need to be used must be redefined in C# with some special attributes to indicate which DLL contains the method. Also all the enums and structs that are used as method parameters need to be redefined in C#.

Code Snippet 10 contains an example of a redefined native structure and the redefined native methods which are used to create Windows hooks. The methods are defined inside the *NativeMethods* class. The *WindowsHook* class uses these methods to handle the logic of adding and removing hooks. Since the *KeyboardListener* and *MouseListener* classes extend from the *WindowsHook* class, they don't have to handle this logic. They just need to implement the *HookCallback* method which gets called for every system message. In this method, they check if they are interested in the system message, and if so, they fire the appropriate event. For example, the *KeyboardListener* checks for the key down system messages and when this message is found it fires its *KeyDown* event, indicating which key was pressed.

Code Snippet 10 - Activity Tracker Native Methods

```
internal class NativeMethods
{
    #region DllImports
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
    public static extern IntPtr GetModuleHandle(string moduleName);

    [DllImport("user32.dll", SetLastError = true)]
    public static extern IntPtr SetWindowsHookEx(HookType hookType, HookCallback
hookCallback, IntPtr hMod, uint threadId);

    [DllImport("user32.dll", SetLastError = true)]
    [return: MarshalAs(UnmanagedType.Bool)]
    public static extern bool UnhookWindowsHookEx(IntPtr hookHandle);

    [DllImport("user32.dll")]
    public static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode, UIntPtr
wParam, IntPtr lParam);
    #endregion
}

// One example of an unmanaged struct redefined in C#
[StructLayout(LayoutKind.Sequential)]
internal struct MouseLLHookStruct
{
    public MousePoint Point;
    public uint MouseData;
    public uint Flags;
    public uint Time;
    public UIntPtr ExtraInfo;
}
```

The Virtual Clicker's activity monitoring code provides the ability to track the time a user spends on typing in words and drawing digital ink. This time can be represented as total time spent or the percentage of time spent from a given starting point. This is accomplished using low level Windows system hooks which allows listening to user input regardless of which application the user is utilizing.

8 Summary

It has been shown that active learning and classroom assessment techniques, when used correctly, can significantly improve student learning outcomes. Though these assessments can be effective, not all instructors are willing to apply them. Reasons for this resistance are, time constraints, both in and out of the classroom, and the inability of instructors and students to shift to this new teaching model. While technology alone cannot create an active learning environment, it can aid instructors in overcoming some of the barriers.

The Virtual Clicker suite of applications was designed with these challenges in mind. The features it provides can aid in instructor student interactions. It does this by making it easier for instructors to quickly assess student understanding and by allowing students to inform the instructor of their comprehension of the current topic via questions and status.

Unfortunately there has been no opportunity to conduct a study of the Virtual Clicker application being used in the classroom to date. While past research has proven many of the application's features to be effective, it's expected that the unique combination and implementation details of the Virtual Clicker application suite will prove to be more effective and user friendly. There is little doubt that assessing the multi-faceted aspects of the activity monitoring feature will be of interest to educators and students.

8.1 Future Work

The Virtual Clicker suite has been heavily tested via simulations using test clients and test hosts but has yet to be used in the classroom. Therefore the obvious next step is to conduct a trial run using the suite in an actual class session. This will potentially expose logical and system level bugs that need to be fixed.

The activity monitor feature is another area that will need to be examined and tested for effectiveness. There are currently multiple thresholds in place that are used when calculating stroke, word, and overall activity rates. The current values have been chosen after some testing was conducted of typical keyboard, mouse, and pen usage by a handful of subjects as they performed common computer activities. When the application suite is deployed for trial runs, all the different activity metrics should be logged and analyzed so these thresholds can be updated if needed.

References

- [1] John Bransford, Ann Brown, and Rodney Cocking, *How People Learn: Brain, Mind, Experience, and School*. Washington D.C.: National Academy Press, 2000.
- [2] Michael Prince, "Does Active Learning Work?," *Journal of Engineering Education*, vol. 93, no. 3, pp. 223-231, 2004.
- [3] Polly Fassinger, "Understanding classroom interaction: Students' and professors' contributions to students' silence," *Journal of Higher Education*, vol. 66, no. 1, pp. 82-96, 1995.
- [4] Donald Bligh, *What's the use of lectures?* San Francisco: Jossey-Bass, 2000.
- [5] Jeffery J. McConnell, "Active learning and its use in computer science," *ACM SIGCSE Bulletin*, vol. 28, no. SI, pp. 52-54, 1996.
- [6] Graham Gibbs and Claire Simpson, "Conditions Under Which Assessment Supports Students' Learning," *Learning and Teaching in Higher Education*, no. 1, 2004-05.
- [7] Mimi Steadman, "Using Classroom Assessment to Change Both Teaching and Learning," *New Directions for Teaching and Learning*, no. 75, pp. 23-35, 1998.
- [8] Robert J. Dufresne, William J. Gerace, Jose P. Mestre, and William J. Leonard, "ASK-IT/A2L: Assessing Student Knowledge with Instructional Technology," University of Massachusetts Physics Education Research Group, UMPERG Technical Report PERG-2000#09-SEP#11-28pp, 2000.
- [9] Ernst Wit, "Who wants to be. The Use of a Personal Response System in Statistics

- Teaching," *MSOR Connections*, vol. 3, no. 2, pp. 14-20, May 2003.
- [10] S. W. Draper and M. I. Brown, "Increasing interactivity in lectures using an electronic voting system," *Journal of Computer Assisted Learning*, vol. 20, no. 2, pp. 81-94, April 2004.
- [11] Richard Anderson et al., "Classroom Presenter: A Classroom Interaction System for Active and Collaborative Learning," in *The Impact of Tablet PCs and Pen-based Technology on Education*, Dave A. Berque, Jane C. Prey, and Robert H. Reed, Eds.: Purdue University Press, 2006, pp. 21-30.
- [12] Dave Berque, "Pushing Forty (Courses per Semester): Pen-Computing and DyKnow Tools at DePauw University," in *The Impact of Tablet PCs and Pen-based Technology on Education*, Dave A. Berque, Jane C. Prey, and Robert H. Reed, Eds.: Purdue University Press, 2006, pp. 31-39.
- [13] Matt R. Ratto, Benjamin R. Shapiro, Tan Minh Truong, and William G. Griswold, "The ActiveClass Project: Experiments in Encouraging Classroom Participation," in *Computer Supported Collaborative Learning*, Kluwer, 2003.
- [14] Robert J. Dufresne, William J. Gerace, William J. Leonard, and Jose P., Wenk, Laura Mestre, "Classtalk: A classroom communication system for active learning," *JOURNAL OF COMPUTING IN HIGHER EDUCATION*, vol. 7, no. 2, pp. 3-47, 1996.
- [15] Maryellen Weimer, *Learner-Centered Teaching: Five Key Changes to Practice.*: Jossey-Bass, 2002.
- [16] Richard M. Felder and Rebecca Brent, "Navigating The Bumpy Road to Student-Centered Instruction," *College Teaching*, vol. 44, no. 2, pp. 43-47, 1996.
- [17] Charles C Bonwell and James A. Eison, "Active Learning: Creating Excitement in the Classroom," George Washington University, Washington DC, ASHEERIC Higher Education Report 1991.

- [18] Chet Meyers and Thomas B. Jones, *Promoting Active Learning: Strategies for the College Classroom.*: Jossey-Bass, 1993.
- [19] David A. Koib, *Experiential Learning: Experience as the Source of Learning and Development.*: Prentice Hall, 1984.
- [20] Richard M. Felder and Linda K. Silverman, "Learning and Teaching Styles in Engineering Education," *Engineering Education*, vol. 78, no. 7, pp. 674-681, 1988.
- [21] Philip C. Wankat, *The Effective, Efficient Professor: Teaching Scholarship and Service*, 1st ed.: Allyn & Bacon, 2002.
- [22] John Stuart and R. J. Rutherford, "Medical Student Concentration During Lectures," *Lancet*, vol. 312, no. 8088, pp. 514-516, September 1978.
- [23] James Hartley and Ivor K. Davies, "Note-Taking: A Critical Review," *Programmed Learning and Educational Technology*, vol. 15, pp. 207-224, August 1978.
- [24] Richard Hake, "Interactive-engagement vs. traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses," *American Journal of Physics*, vol. 66, no. 1, pp. 64-74, 1998.
- [25] Edward F. Redish, Saul Jeffery M., and Steinberg Richard N., "On the Effectiveness of Active-Engagement Microcomputer-Based Laboratories," *American Journal of Physics*, vol. 65, pp. 45-54, 1997.
- [26] Kathleen A. Marrs and Gregor Novak, "Just-in-Time Teaching in Biology: Creating an Active Learner Classroom Using the Internet," *American Society for Cell Biology*, vol. 3, pp. 59-61, 2004.
- [27] Kimberle Koile and David Singer, "Improving learning in CS1 via tablet-PC-based in-class assessment," in *International Computing Education Research* , Canterbury, 2006, pp. 24-26.

- [28] Valentin Razmov and Richard Anderson, "Pedagogical techniques supported by the use of student devices in teaching software engineering," in *SIGCSE*, Houston, 06, pp. 344-348.
- [29] Joseph G. Tront and Glenda R. Scales, "Keynote Address Implementing a Large-Scale Tablet PC Deployment," in *The Impact of Tablet PCs and Pen-based Technology on Education*, Jane C. Prey, Robert H. Reed, and Dave A. Berque, Eds.: Purdue University Press, 2007, pp. 1-10.
- [30] Beth Simon, Ruth Anderson, Crystal Hoyer, and Jonathan Su, "Preliminary experiences with a Tablet PC based system to support active learning in computer science courses," in *ITICSE*, Leeds, 2004, pp. 213-217.
- [31] DyKnow. (2011, May) DyKnow. [Online]. <http://www.dyknow.com/>
- [32] Mary Dixon, Kerry Pannell, and Michele Villinski, "From "Chalk and Talk" to Animate and Collaborate: DyKnow-Mite Applications of Pen-Based Instruction in Economics," in *The Impact Of Tablet Pcs And Pen-Based Technology On Education: Vignettes, Evaluations, And Future Directions*, Dave A. Berque, Jane C. Prey, and Robert H. Reed, Eds.: Purdue University Press, 2006, pp. 49-55.
- [33] Scott M. Thede, "Using DyKnow Vision Software and Pen-Enabled Computers to Increase Class Participation," in *The Impact Of Tablet Pcs And Pen-Based Technology On Education: Vignettes, Evaluations, And Future Directions*, Dave A. Berque, Jane C. Prey, and Robert H. Reed, Eds.: Purdue University Press, 2006, pp. 179-186.
- [34] Microsoft. (2011, November) Microsoft Education Labs. [Online]. <http://www.educationlabs.com/projects/ic/Pages/default.aspx>
- [35] Better Education, Inc. (2011, May) Classtalk. [Online]. <http://www.bedu.com/classtalk.html>
- [36] John Waite Bowers, "Classroom Communication Apprehension: A Survey," *Communication Education*, vol. 35, pp. 372-378, October 1986.

- [37] Brenda Timmerman, Robert Lingard, and Michael G. Barnes, "Active Learning with Upper Division Computer Science Students," in *31st ASEE/IEEE Frontiers in Education Conference*, Reno, 2001, pp. 19-23.
- [38] Microsoft. (2011, April) MSDN.Net Development. [Online]. <http://msdn.microsoft.com/en-us/library/ff361664.aspx>
- [39] Jeffrey Richter, *Applied Microsoft .Net Framework Programming*, 1st ed., Sally Stickney, Ed.: Microsoft Press, 2002.
- [40] Microsoft. (2011, April) MSDN.NET Framework 4. [Online]. <http://msdn.microsoft.com/en-us/library/w0x726c2.aspx>
- [41] Microsoft. (2011, April) MSDN The Reactive Extensions. [Online]. <http://msdn.microsoft.com/en-us/data/gg577609>
- [42] Microsoft. (2011, April) C# Language Specification 3.0. [Online]. <http://download.microsoft.com/download/3/8/8/388e7205-bc10-4226-b2a8-75351c669b09/CSharp%20Language%20Specification.doc>
- [43] Microsoft. (2011, May) MSDN Windows Presentation Foundation. [Online]. <http://msdn.microsoft.com/en-us/library/ms754130.aspx>
- [44] Charles Petzold, *Applications = Code + Markup*, 1st ed.: Microsoft Press, 2006.
- [45] Josh Smith. (2009, February) WPF Apps With The Model-View-ViewModel Design Pattern. [Online]. <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>
- [46] John Gossman. (2005, October) Introduction to Model/View/ViewModel pattern for building WPF apps. [Online]. <http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx>

- [47] Martin Fowler. (2004, July) Presentation Model. [Online].
<http://martinfowler.com/eaDev/PresentationModel.html>
- [48] Josh Smith, *Advanced MVVM*, 1st ed.: Josh Smith, 2010.
- [49] Juval Lowy, *Programming WCF Services*, 1st ed.: O'Reilly Media, Inc, 2007.
- [50] Microsoft. (2011, May) MSDN Windows Communication Foundation. [Online].
<http://msdn.microsoft.com/en-us/library/dd456779.aspx>
- [51] W3C. (2011, October) Extensible Markup Language. [Online]. <http://www.w3.org/XML/>
- [52] W3C. (2010, October) Extensible Markup Language 1.0 Specification. [Online].
<http://www.w3.org/TR/2008/REC-xml-20081126/>
- [53] Microsoft. (2011, May) MSDN Digital Ink. [Online]. <http://msdn.microsoft.com/en-us/library/ms752707.aspx>
- [54] Rob Jarrett and Su Philip, *Building Tablet PC Applications*.: Microsoft Press, 2003.
- [55] Microsoft. (2011, July) MSDN System.Runtime.Serialization Namespace. [Online].
<http://msdn.microsoft.com/en-us/library/kd1dc9w5.aspx>
- [56] Microsoft. (2011, October) MSDN - Sockets. [Online]. <http://msdn.microsoft.com/en-us/library/sb27wehh.aspx>
- [57] Microsoft. (2001, July) MSDN WCF Endpoints. [Online]. <http://msdn.microsoft.com/en-us/library/ms733107.aspx>
- [58] OASIS. (2011, October) Web Services Dynamic Discovery. [Online]. <http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.pdf>
- [59] Microsoft. (2011, July) MSDN WCF Discovery. [Online]. <http://msdn.microsoft.com/en->

[us/library/dd456782.aspx](http://msdn.microsoft.com/en-us/library/dd456782.aspx)

[60] Microsoft. (2011, October) MSDN: IDispose. [Online]. <http://msdn.microsoft.com/en-us/library/fs2xkftw.aspx>

[61] Microsoft. (2011, July) MSDN: Hooks. [Online]. [http://msdn.microsoft.com/en-us/library/ms632589\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms632589(v=vs.85).aspx)

[62] Microsoft. (2011, July) MSDN: Platform Invoke Tutorial. [Online]. [http://msdn.microsoft.com/en-us/library/aa288468\(v=VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa288468(v=VS.71).aspx)

[63] Microsoft. (2011, July) Visual Studio. [Online]. <http://msdn.microsoft.com/en-us/vstudio/aa718325>

[64] Microsoft. (2011, July) MSDN: System Events and Mouse Messages. [Online]. [http://msdn.microsoft.com/en-gb/library/ms703320\(VS.85\).aspx](http://msdn.microsoft.com/en-gb/library/ms703320(VS.85).aspx)

Appendix A: Developer's Guide

This appendix describes the layout of the source code into different projects, how its source version control works, and how to obtain helpful logging information. The complete Virtual Clicker suite was written in XAML and C# using Visual Studio 2010, which can be downloaded from the Microsoft Visual Studio website [63]. The root directory of the project contains:

- Nine folders for Visual Studio projects, these all start with VirtualClicker.
- A folder named 'DesignImages' which contains all the images and icons used.
- A folder named 'dlls' which contains the DLLs for the Reactive Extensions library.
- A folder named '.git' which contains source control information.
- A text file named .gitignore which has source control settings.
- A text file named 'bugs.txt' which has a list of current bugs.
- A Visual Studio solution file named 'VirtualClicker.sln' which will open up all the projects.

Code Project

This is a list of all the different projects that make up the Virtual Clicker suite code base.

VirtualClicker.Common

This project builds into a DLL class library and contains all the common code which is shared between all the different applications. The code in this project is broken down into the following different namespaces:

- **AppManagement** - the base classes all applications extend from and all the single instance management code.

- **Commands** - all the code needed for command binding with WPF when the MVVM design pattern is used.
- **ComponentModel** - common interfaces, all the observable object and observable property code used for property change tracking need when using the MVVM design pattern.
- **Converters** - all the data converters used within the XAML code.
- **Diagnostics** - all the application logging code.
- **Extensions** - all the common extension methods.
- **Images** - all the common application images used.
- **Models** - all the data models including the quiz/question models.
- **Resources** - common style sheets.
- **UIElements** - custom UI controls.
- **Validation** - input validation rules.
- **ViewModels** - base view model all others extend from and view models for common views.
- **Views** - common XAML view code for quiz, questions, and ink ribbon.

VirtualClicker.Service

This project builds into a DLL class library. It defines the interface for the host service and the callback interface for clients. It also has the host service implementation.

VirtualClicker.Host

This project builds into a WPF application. It contains all the instructor host application code including the XAML for all its Views and the corresponding ViewModels.

VirtualClicker.Client

This project builds into a WPF application. It contains all the student client application code including the XAML for all its Views and the corresponding ViewModels. It also contains an

implementation of the service callback interface, code for finding and detecting hosted services, and code for connecting and communicating with a hosted service.

VirtualClicker.TestClient

This project builds into a command line application. It implements the service callback interface and allows simulating multiple clients to allow testing of the different features.

VirtualClicker.QuizCreator

This project builds into a WPF application. It contains all the quiz creator application code including the XAML for all its Views and the corresponding ViewModels.

VirtualClicker.QuizGrader

This project builds into a WPF application. It contains all the quiz grader application code including the XAML for all its Views and the corresponding ViewModels.

VirtualClicker.StudentSetup

This is a setup and deployment project that builds an MSI file for installing the Student client application.

VirtualClicker.InstructorSetup

This is a setup and deployment project that builds an MSI file for installing the Instructor application, the quiz creator application, and the quiz grader application.

Version Control

Keeping version control in a project of this size is important so all changes can be tracked and reverted in case a change causes a regression in functionality. For this project Git was chosen for version control. It was chosen because 1) it was free and 2) no servers are needed, allowing

all the version history to be stored in a folder within the parent solution folder. This allows the code, including all revision history to be easily handed off as just a ZIP file.

To view the revision history of the Virtual Clicker application suite, first install Git and then open the code repository by pointing either the GUI or command line Git tool at the root folder of the project. Git can be downloaded from <http://git-scm.com/>.

Logging

All the applications in the Virtual Clicker suite log useful information, including any errors or exceptions encountered. Logs are saved under %appdata%\VirtualClicker\[AppName].log, where %appdata% is the Windows environment variable that points to a users roaming application data folder and [AppName] is the application's name (e.g., VirtualClicker.Host). There isn't much overhead caused by this logging but logging can be turned off at compile time by removing the "TRACE" constant from the Visual Studio projects. This is accomplished by opening the properties for a project, selecting the build tab, choosing the "Release" configuration, and un-checking the "Define TRACE constant" check box.

Appendix B: User's Guide

This appendix is the User Guide for the Virtual Clicker application suite which was designed to address the need for active engagement, in-class feedback, and classroom interaction, even in large classrooms. It allows instructors to host a virtual class session which students can connect to. Once students are connected, instructors can distribute virtual quizzes and polls to students who can answer the questions and submit them back to the instructor. The application suite also allows students to ask anonymous questions and allows instructors to monitor student status and activity levels.

There are four applications included in the suite: Quiz Creator, Quiz Grader, Instructor host application, and Student client application. The first two are designed to be used by instructors and teaching assistants to create quizzes ahead of class and quickly grade submitted quizzes after class. The third, Instructor host application, is intended to be used in the classroom by instructors for hosting virtual classroom sessions. The fourth, Student client application, allows students to connect to hosted virtual classroom sessions.

This appendix illustrates, in a step-by-step fashion, how users can install and utilize each of the four applications in the suite.

Installation

The four applications of the suite are broken up into two groups which can be installed using a single Windows Installer file, also called an MSI file. The first group contains the applications which are intended to be used by instructors. These are the Quiz Creator, Quiz Grader, and Instructor host applications. The second group contains the one application intended to be used by students, which is the Student client application. Both groups can be installed on the

same computer if desired. Also, note that both installations require .NET version 4.0 Client Profile to be installed first.

Installing .NET 4.0

The .NET Framework 4.0 Client Profile can be installed through Windows Update. It is released as a recommended update on Windows Vista and Windows 7. It is released as an optional update for Windows XP. Alternatively, a standalone installer can be downloaded from [here](#).

Installing Instructor Applications

The following steps will install the three instructor applications: Quiz Creator, Quiz Grader, and Instructor host. The .NET framework version 4.0 must be installed first otherwise installation will fail.

1. Run the "*VirtualClickerInstructorSetup.msi*" setup file by double clicking on it.
2. Once the setup wizard dialog appears as seen in Figure 6. Click the "Next" button to continue.

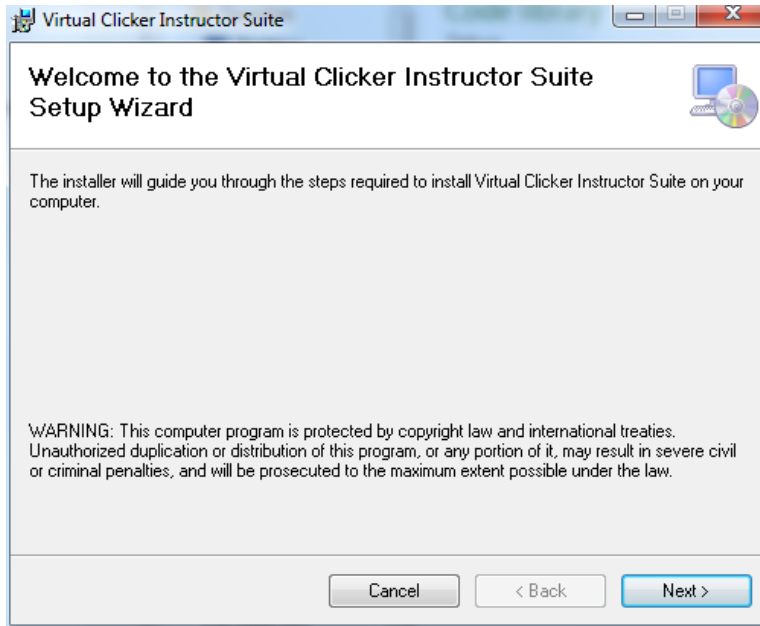


Figure 6 - Instructor Applications Setup Wizard

3. The next step is to choose the location to install the instructor applications as seen in Figure 7. The default location is "C:\Program Files\Virginia Tech\Virtual Clicker Instructor Suite\". After setting the location or using the default, click the "Next" button.

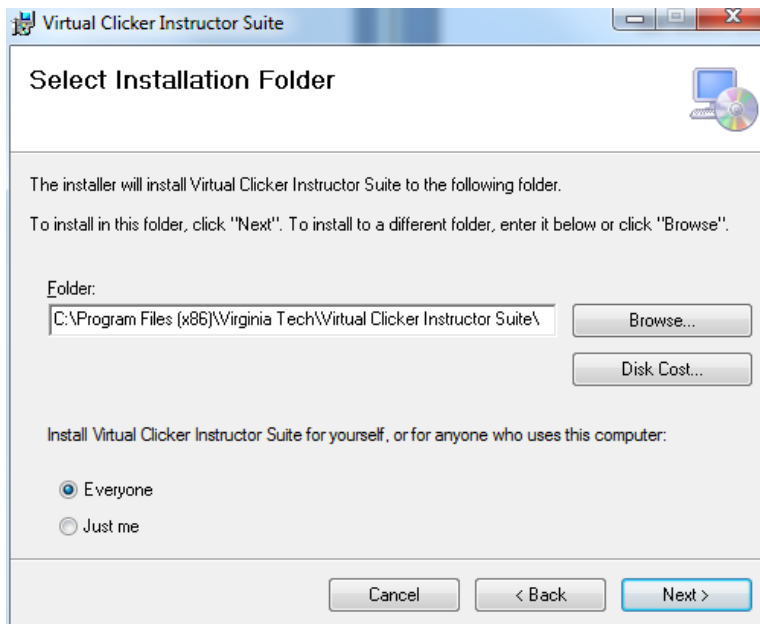


Figure 7 - Instructor Applications Install Location

4. The next step is to confirm the install. Click the "Next" button to start installation.

5. The installation should take a short time and should display a progress bar. Once finished an installation complete dialog will be displayed, as seen in Figure 8. Click the "Close" button to dismiss the dialog. The application suite is now installed.

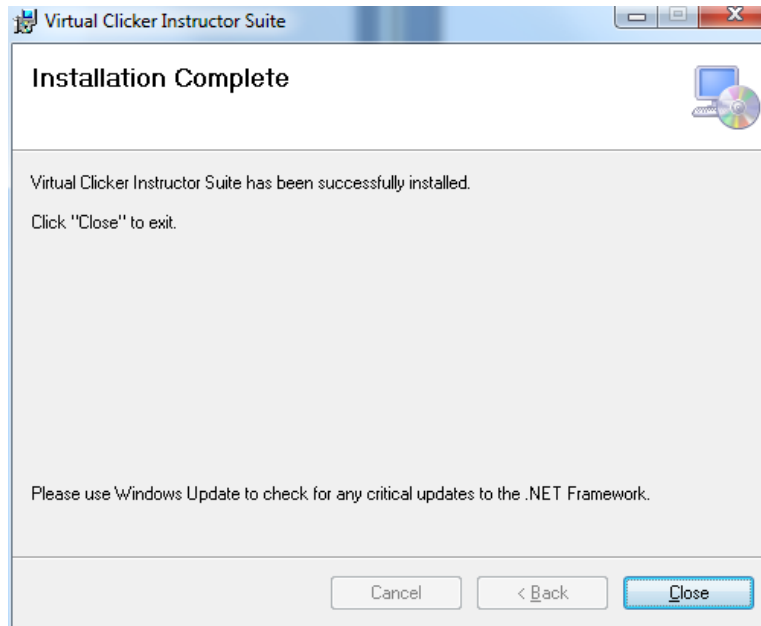


Figure 8 - Instructor Applications Installation Complete

Installing Student Application

The following steps will install the Student client application. The .NET framework version 4.0 must be installed first otherwise installation will fail.

1. Run the " *VirtualClickerStudentttSetup.msi*" setup file by double clicking on it.
2. Once the setup wizard dialog appears as seen in Figure 9. Click the "Next" button to continue.

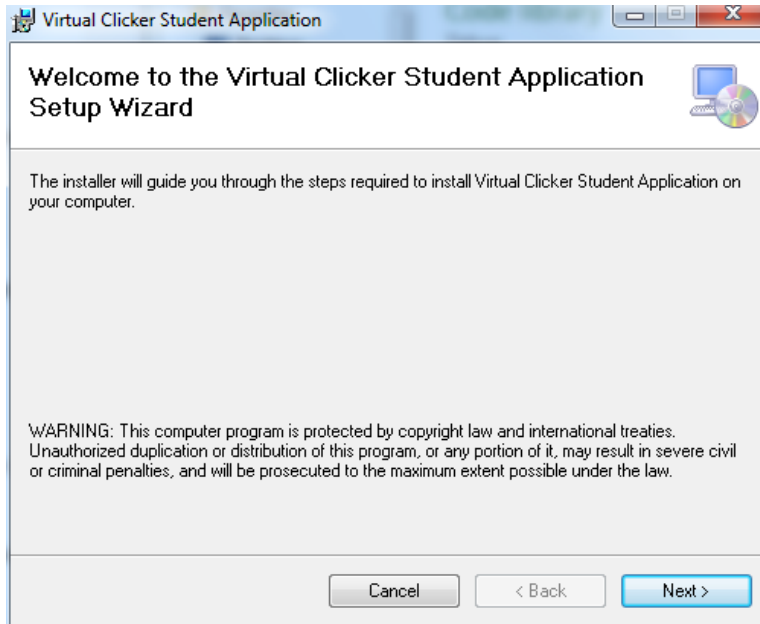


Figure 9 - Student Application Setup Wizard

3. The next step is to choose the location to install the instructor applications as seen in Figure 10. The default location is " C:\Program Files\Virginia Tech\Virtual Clicker Student Application\ ". After setting the location or using the default, click the "Next" button.

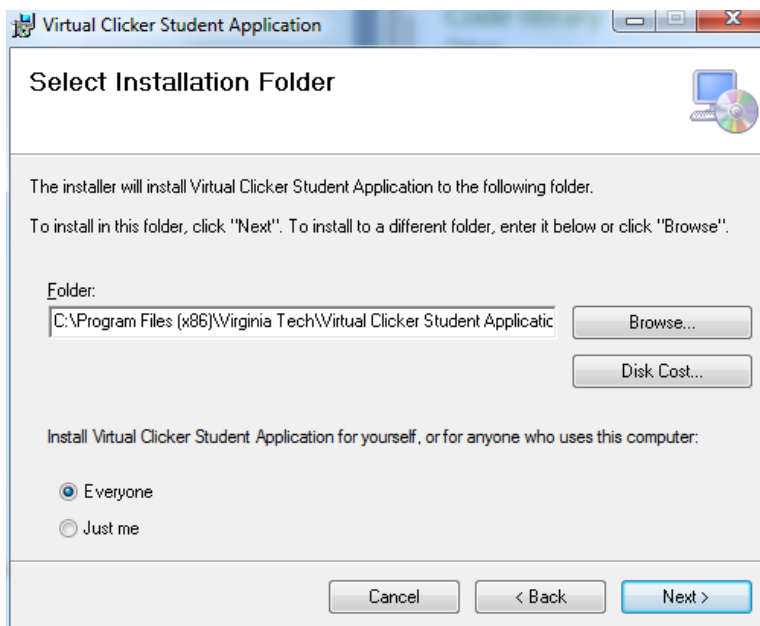


Figure 10 - Student Application Install Location

4. The next step is to confirm the install. Click the "Next" button to start installation.

5. The installation should take a short time and should display a progress bar. Once finished an installation complete dialog will be displayed, as seen in Figure 11. Click the "Close" button to dismiss the dialog. The application suite is now installed.

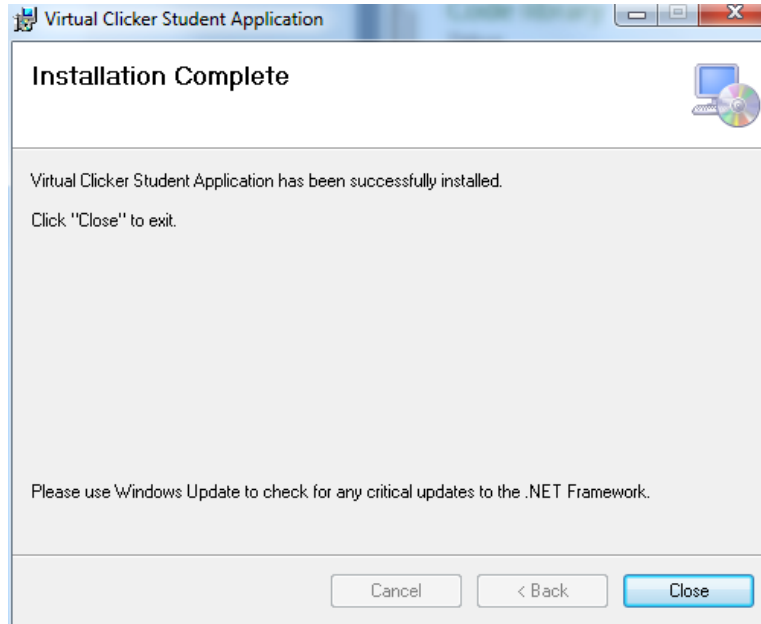


Figure 11 - Student Application Installation Complete

Quiz Creator

This section of the appendix provides detailed instructions on how to use the Quiz Creator application. This application allows instructors and teaching assistants to create virtual quizzes ahead of a classroom session.

Starting Quiz Creator

The Quiz Creator can be launched from the start menu by choosing: Start → All Programs → Virtual Clicker → Virtual Clicker Quiz Creator. Figure 12 shows the Quiz Creator highlighted in the start menu.

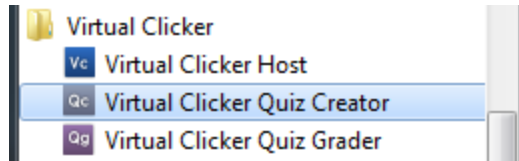


Figure 12 - Starting Quiz Creator

Once the Quiz Creator is launched, it opens up to a quiz document which has no questions, shown in Figure 13. It has a ribbon control instead of a standard toolbar because the larger buttons of the ribbon make it easier to use on Tablet PCs. The body of the application contains the quiz document which is a vertically scrollable region which contains information about the quiz (e.g., quiz name, instructor name, instructions) and any questions added.

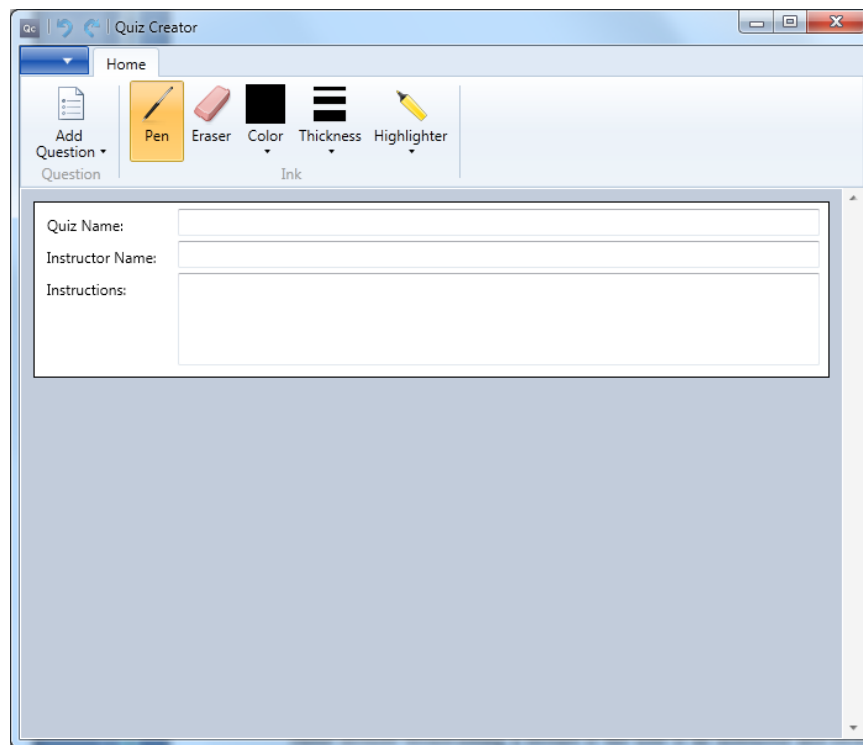


Figure 13 - Quiz Creator UI

Adding Questions

To add questions to the quiz:

1. Click the "Add Question" dropdown button in the ribbon.
2. From the dropdown, select the type of question to add. As seen in Figure 14, the available types are: Multiple Choice Question, Multiple Answer Question, Essay Question, and Ink Form Question.

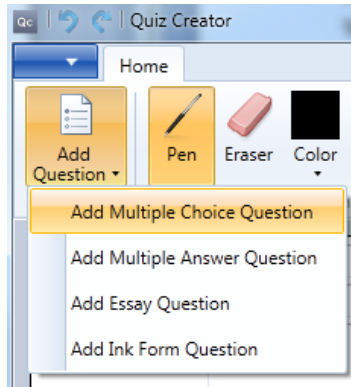


Figure 14 - Quiz Creator Add Question

3. This causes a new question to get added to the end of the quiz document, as seen in Figure 15.

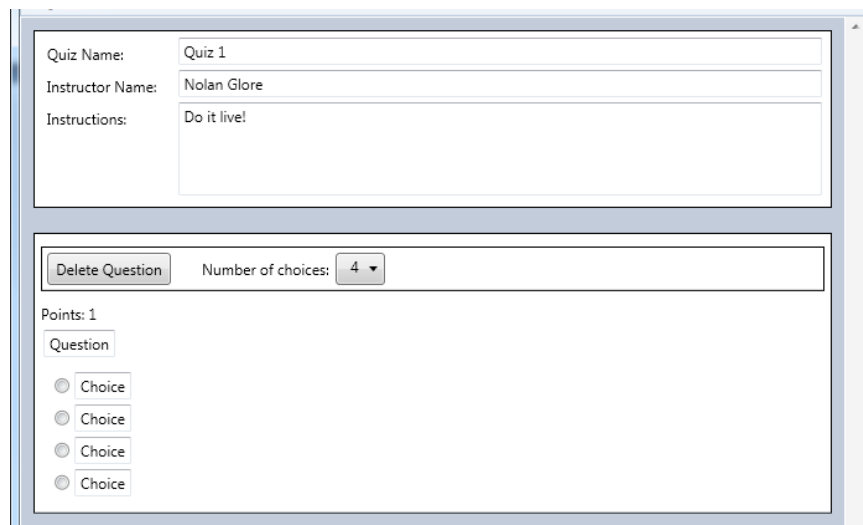


Figure 15 - Quiz Creator Added Question

Multiple Choice Questions

Multiple choice questions contain a question and anywhere from 2 to 26 choices where only one choice is the correct answer. This type of question has 1 point possible. When graded, a student gets the 1 point if they select the correct answer. When a new multiple choice question is added, it is displayed with some defaults, as seen in Figure 16.

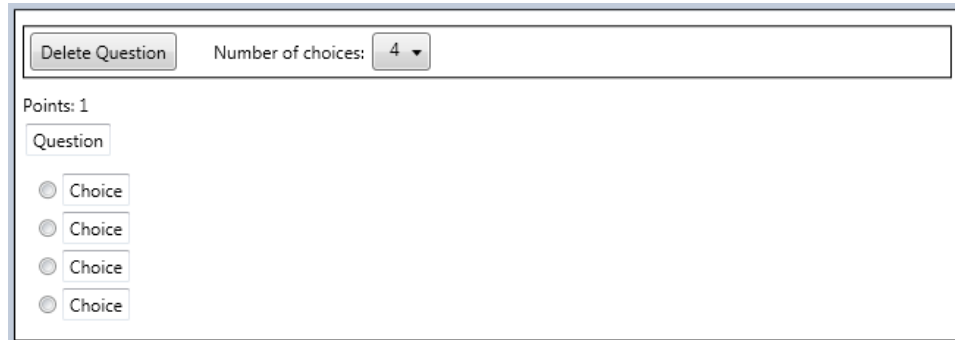
The image shows a screenshot of a question editor interface. At the top, there is a 'Delete Question' button and a 'Number of choices:' label followed by a dropdown menu set to '4'. Below this, the 'Points:' field is set to '1'. The 'Question' text box contains the word 'Question'. Underneath, there are four radio buttons, each followed by a text box containing the word 'Choice'.

Figure 16 - Default Multiple Choice Question

Follow these steps to customize a multiple choice question:

1. When a multiple choice question gets added it has 4 choices by default. To change this, select the "Number of choices" drop down, located at the top of the question, and select a number between 2 and 26.
2. The first text box is the question text. By default this contains the word "Question". Replace this with a question of your choice. In the example seen in Figure 17, "In the equation $2x + 3 = 4$, solve for x " was entered.
3. Under the question text are X, where X is the number chosen in step 1, radio buttons with text boxes next to them. These represent the possible choices and are filled with the word "Choice" by default. Replace these with the possible choices for the question and then select the correct choice by clicking on the radio box next to the correct choice text box. In the example seen in Figure 17, the third choice, .5, is selected as the correct choice.

A screenshot of a question editor interface. At the top, there is a 'Delete Question' button and a 'Number of choices:' dropdown menu set to '5'. Below this, the 'Points:' field is set to '1'. The question text is 'In the equation $2x + 3 = 4$, solve for x.' Below the question, there are five radio button options: '4', '10', '.5', '1.5', and '8'. The option '.5' is selected.

Figure 17 - Example Multiple Choice Question

Multiple Answer Questions

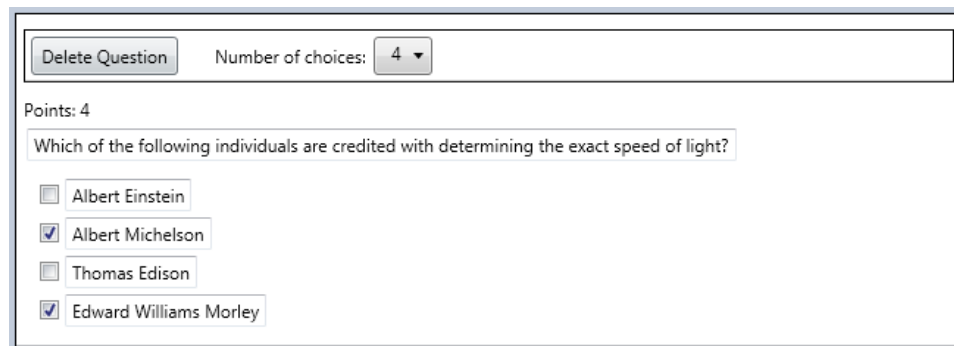
Multiple answer questions contain a question and anywhere from 2 to 26 answers where it's possible that more than one is correct. This type of question has X points possible, where X is the number of possible answers. When graded, a student gets 1 point for each answer they correctly indicated as either correct or incorrect. When a new multiple answer question is added, it is displayed with some defaults, as seen in Figure 18.

A screenshot of a question editor interface for a multiple answer question. At the top, there is a 'Delete Question' button and a 'Number of choices:' dropdown menu set to '4'. Below this, the 'Points:' field is set to '4'. The question text is 'Question'. Below the question, there are four checkbox options, each labeled 'Choice'.

Figure 18 - Default Multiple Answer Question

Follow these steps to customize a multiple answer question:

1. When a multiple answer question gets added it has 4 answer by default. To change this, select the "Number of choices" drop down, located at the top of the question, and select a number between 2 and 26.
2. The first text box is the question text. By default this contains the word "Question". Replace this with a question of your choice. In the example seen in Figure 19, "Which of the following individuals are credited with determining the exact speed of light" was entered.
3. Under the question text are X, where X is the number chosen in step 1, check boxes with text boxes next to them. These represent the possible answers and are filled with the word "Choice" by default. Replace these with the possible answers for the question and then select all the correct answers by clicking on the check boxes next to the correct answers text boxes. In the example seen in Figure 19, the second answer, Albert Michelson, and the fourth answer, Edward Williams Morley, are selected as correct answers.



Delete Question Number of choices: 4

Points: 4

Which of the following individuals are credited with determining the exact speed of light?

Albert Einstein

Albert Michelson

Thomas Edison

Edward Williams Morley

Figure 19 - Example Multiple Choice Question

Essay Question

Essay questions contain a question and area for students to type in their answer. The possible points for this type of question is configurable from 1 to 999. When a new essay question is added, it is displayed with some defaults, as seen in Figure 20.

The screenshot shows a web form for creating an essay question. At the top is a button labeled "Delete Question". Below it is a "Points:" label followed by a text input field containing the number "1". Underneath is a "Question" label followed by a large, empty text area for entering the question text.

Figure 20 - Default Essay Question

Follow these steps to customize an essay question:

1. When an essay question gets added, its points possible defaults to 1. This can be changed to any value between 1 and 999 by updating the value in the first text box.
2. The second text box is the question text. By default this contains the word "Question". Replace this with a question of your choice. In the example seen in Figure 21, "Based on the model we have been describing in class, how could you account for the difference between a conductor and a resistor... Explain" was entered.
3. The third text box represents the answer. This text box can be used to enter any notes about what the correct answer should contain which can help in the grading process.

The screenshot shows the same form as Figure 20 but with customized content. The "Delete Question" button is at the top. The "Points:" label is followed by a text input field containing the number "5". The "Question" label is followed by a text area containing the text: "Based on the model we have been describing in class, how could you account for the difference between a conductor and a resistor... Explain." Below this is a larger text area containing the text: "Instructor notes about what the correct answer should contain can go here to help in the grading process".

Figure 21 - Sample Essay Question

Ink Question

Ink questions contain a question and area for students to draw digital ink. The possible points for this type of question is configurable from 1 to 999. When a new ink question is added, it is displayed with some defaults and a blank drawing canvas, as seen in Figure 22.

The image shows a software interface for creating an ink question. At the top, there is a button labeled "Delete Question". Below this, there is a "Points:" label followed by a text input box containing the number "1". Underneath that is a "Question" label followed by a larger text input box containing the word "Question". The bottom half of the interface is a large, solid gray rectangular area representing a blank drawing canvas.

Figure 22 - Default Ink Question

Follow these steps to customize an ink question:

1. When an ink question gets added, its points possible defaults to 1. This can be changed to any value between 1 and 999 by updating the value in the first text box.
2. The second text box is the question text. By default this contains the word "Question". Replace this with a question of your choice. In the example seen in Figure 24, "Finish the cube" was entered.
3. Finally there is a drawing canvas that supports digital ink. A base drawing can be added to this canvas which students have to complete. The application's ribbon has controls to configure editing the drawing, as seen in Figure 23. It supports switching the input method between a pen, eraser, and highlighter. It also supports changing the input color and thickness.

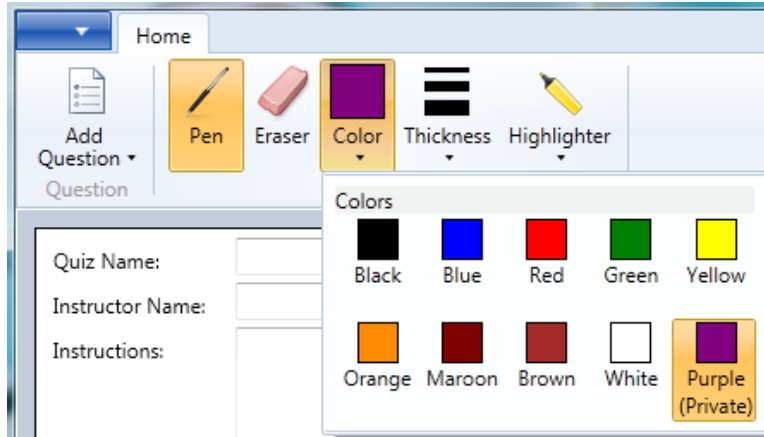


Figure 23 - Quiz Creator Ribbon

4. Private instructor only ink can also be added by choosing the input color purple, as seen in Figure 23. This ink will be removed from the drawing before the quiz is sent to students.

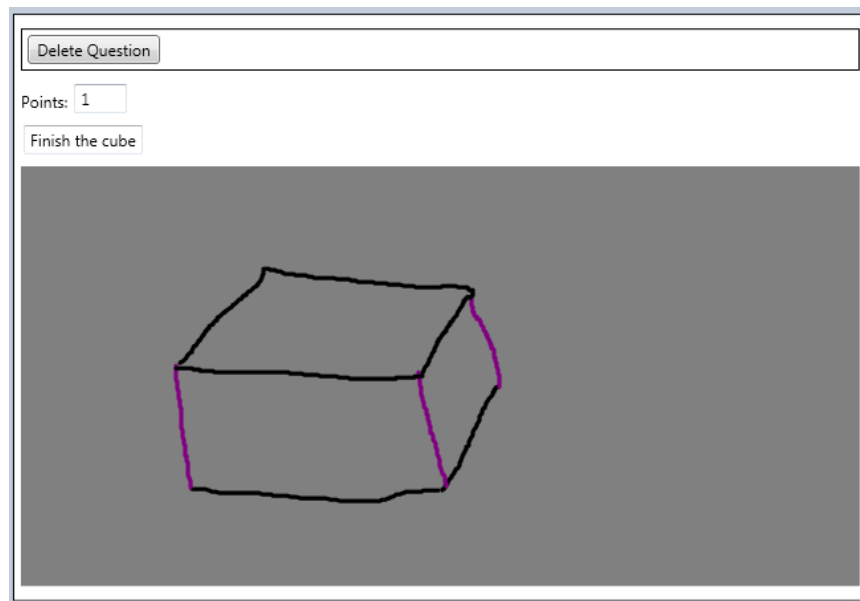


Figure 24 - Example Ink Question

Deleting Questions

Any question can be deleted by clicking the "Delete Question" button found at the top of every question.

Saving A Quiz

Follow these steps to save a quiz:

1. Click on the blue button with the down arrow that is to the left of the "Home" ribbon tab. This can be seen in Figure 25.
2. Click on the "Save" button to save any changes to the current document or click on the "Save As" button to save changes to a new document.

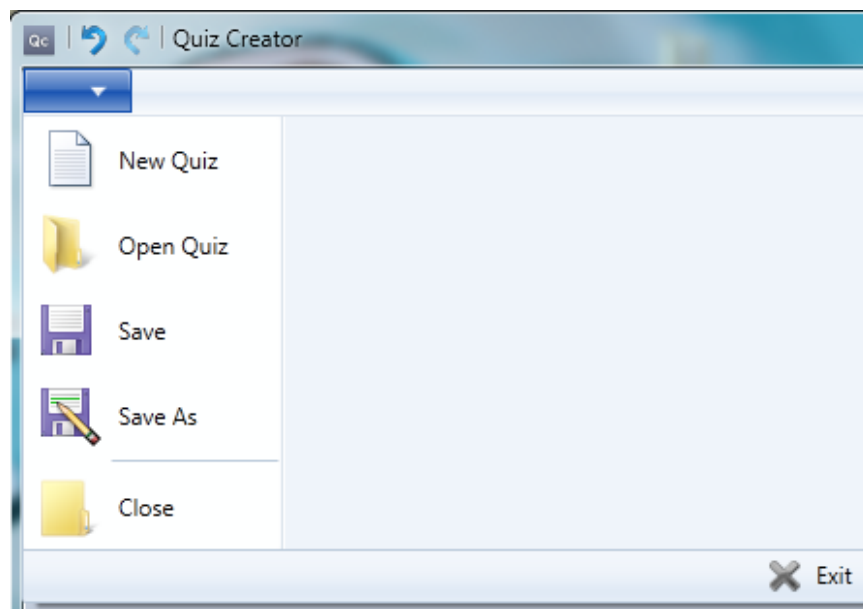


Figure 25 - Quiz Creator Application Menu

Opening A Quiz

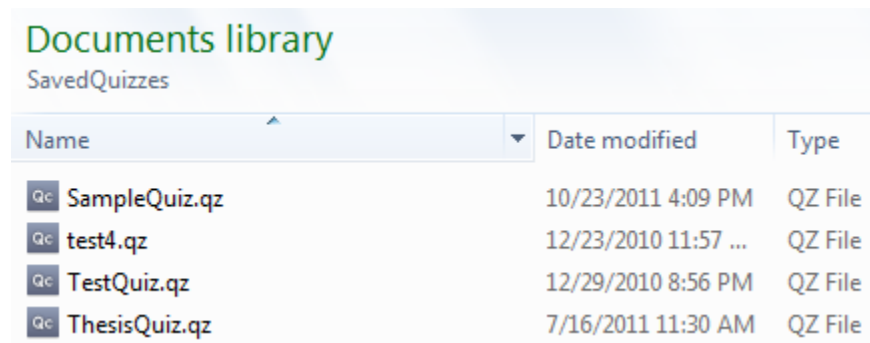
A quiz can be opened in two ways:

Method 1:

1. Click on the blue button with the down arrow that is to the left of the "Home" ribbon tab. This can be seen in Figure 25.
2. Click on the "Open Quiz" button and navigate to the quiz to open.

Method 2:

1. Use Windows Explorer to navigate to the folder containing the quiz to open.
2. Double click on the quiz file. This will cause the Quiz Creator to open with the quiz document loaded. This works because the .qz file extension is registered with Windows to launch the Quiz Creator app by default. This causes the Quiz Creator logo to be displayed for .qz files, as seen in Figure 26.







Name	Date modified	Type
 SampleQuiz.qz	10/23/2011 4:09 PM	QZ File
 test4.qz	12/23/2010 11:57 ...	QZ File
 TestQuiz.qz	12/29/2010 8:56 PM	QZ File
 ThesisQuiz.qz	7/16/2011 11:30 AM	QZ File

Figure 26 - Quiz File Extension

Quiz Grader

The Quiz Grader application allows for quick and easy grading of completed quizzes. When instructors stop a quiz in the Instructor Host application, all submitted quizzes are automatically saved to a .qzs file which contains all the submitted quizzes and the answer key.

Starting Quiz Grader

The Instructor Host application can be launched from the start menu by choosing: Start → All Programs → Virtual Clicker → Virtual Clicker Quiz Grader. Figure 27 shows the Quiz Grader highlighted in the start menu.

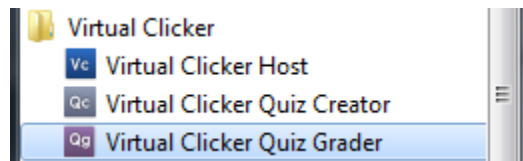


Figure 27 - Starting Quiz Grader

Figure 28 breaks down the major components of the Quiz Grader UI. There are three toolbars which are used in grading quizzes. The file toolbar allows opening completed quizzes, saving graded quizzes, exporting grades, and auto grading. The navigation toolbar is used for navigating left and right between the questions of a quiz and up and down between the students who submitted a quiz. The third toolbar has an editable text box for entering the score of the current question for the selected student.

The student list displays all the students who have submitted a quiz. Next to each student name is their total points scored and the percentage of scored points out of total points possible. Student names are highlighted in red when there are still questions that have yet to be scored.

The current question pane displays a single question for the selected student. Just below this pane is the answer key, which displays the instructor provided answer for the current question. Below the student list is the answer distribution pane which displays a pie chart showing the answer distribution across all students for the current question. This is only displayed for multiple choice and multiple answer question types.

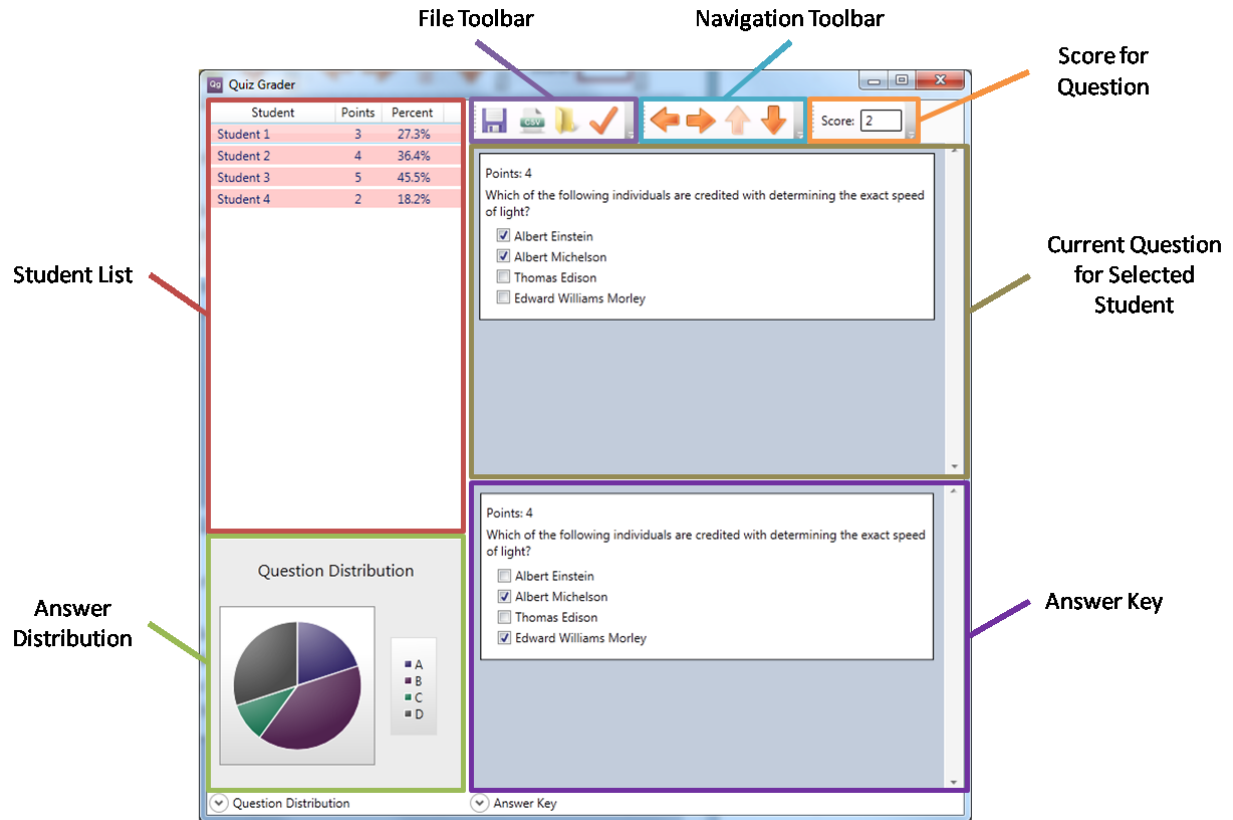


Figure 28 - Quiz Grader UI

Opening Completed Quizzes

Completed quizzes can be opened in two ways:

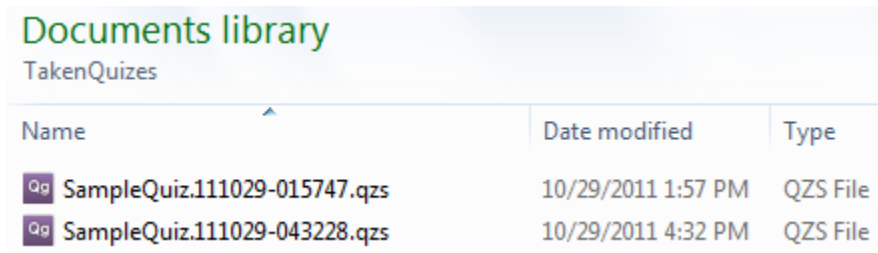
Method 1:

1. Click on the "Open" button from the file toolbar and select the completed quizzes file to open.

Method 2:

1. Use Windows Explorer to navigate to the folder containing the quiz to open.
2. Double click on the completed quiz file. This will cause the Quiz Grader to open with the completed quizzes document loaded. This works because the .qzs file extension is

registered with Windows to launch the Quiz Grader app by default. This causes the Quiz Grader logo to be displayed for .qzs files, as seen in Figure 29.



The screenshot shows a Windows 'Documents library' window with a folder named 'TakenQuizzes'. Inside the folder, there is a table listing two files. Each file has a purple icon with 'Qq' on it, indicating it is a QZS File. The table columns are 'Name', 'Date modified', and 'Type'.

Name	Date modified	Type
SampleQuiz.111029-015747.qzs	10/29/2011 1:57 PM	QZS File
SampleQuiz.111029-043228.qzs	10/29/2011 4:32 PM	QZS File

Figure 29 - Completed Quizzes File Extension

Saving Graded Quizzes

To save graded quizzes back to the .qzs file, click on the "Save" button in the file toolbar. This saves any entered scores back into the file which allows instructors to grade quizzes across multiple launches of the application.

Exporting Grades

The Quiz Grader application allows exporting grades to a comma separated value (CSV) file which can be opened in Excel. To export grades, click on the "Export Grades" button in the file toolbar and choose where to save the CSV file.

Auto Grading

The quiz Grader application supports auto grading multiple choice and multiple answer questions. To perform an auto grade on these types of questions, click the "Auto Grade" in the file toolbar. The auto grader will score multiple choice questions with one point if the student chooses the correct answer. For multiple answer questions, it will give one point for each answer that was properly indicated as correct or incorrect (e.g., if there were 4 possible

answers, where first two were correct, and the student indicated the middle two where correct, the score would be 2 out of 4 points).

Navigation

The Quiz Grader only shows one question at a time. The instructor must navigate between the list of students and the questions to grade all the quizzes.

Navigation Between Students

There are three ways to navigate between students:

- Select a student from the student list.
- Use the up and down arrows in the navigation toolbar to move to the previous and next student in the list.
- Use the up and down keyboard arrows to move to the previous and next students in the list.

Navigation Between Questions

There are two ways to navigate between questions:

- Use the left and right arrows in the navigation toolbar to move to the previous and next questions in the quiz.
- Use the left and right keyboard arrows to move to the previous and next question in the quiz.

Manually Grading

To modify an auto graded score or to grade essay and ink questions, simply enter an integer value into the text box in the score toolbar.

Instructor Host

This section of the appendix provides detailed instructions on how to use the Instructor application. This application allows instructors to host virtual classroom sessions which students can connect to. This allows instructors to send quizzes and polls to students, monitor student status and activity, and receive student questions.

Starting Instructor Host

The Instructor Host application can be launched from the start menu by choosing: Start → All Programs → Virtual Clicker → Virtual Clicker Host. Figure 30 shows the Instructor Host highlighted in the start menu.

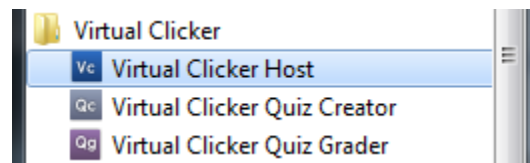


Figure 30 - Starting Instructor Host

The very first time the application is launched, the instructor will be prompted for their name as seen in Figure 31. The application stores this information which is used when broadcasting hosted classroom information.

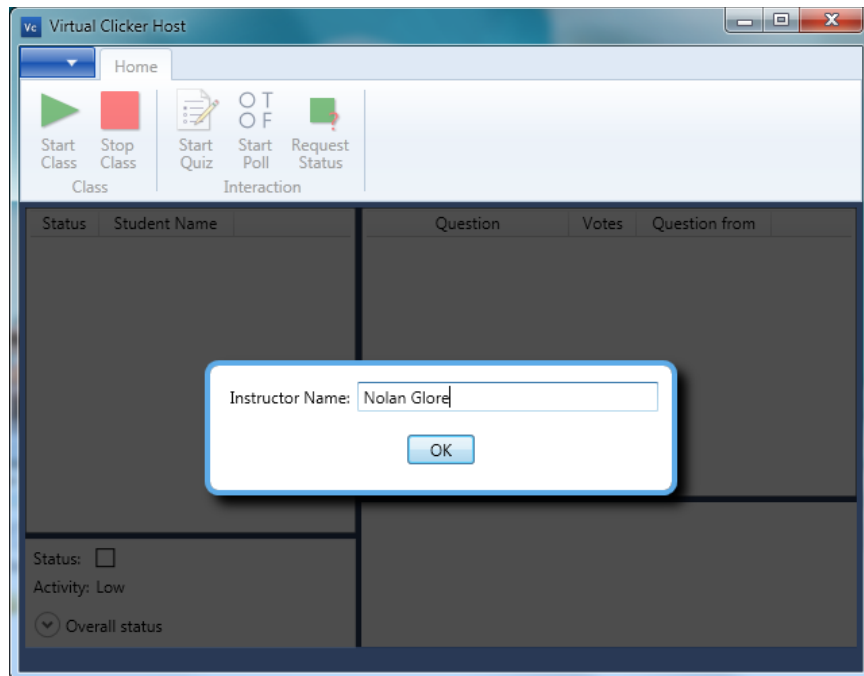


Figure 31 - Instructor Host First Run

Figure 32 breaks down the main UI components of the application. It has a ribbon control instead of a standard toolbar because the larger buttons of the ribbon make it easier to use on Tablet PCs. The ribbon has buttons for common commands, including: start class, stop class, start quiz, start poll, and request status. There is a student list which displays all the students connected to the hosted classroom session and indicates the status of each student. Below this list is the overall class status pane which displays the overall status of the class and the overall student activity level. To the right of the student list is the question list. This is a list of questions received from students which is sorted by the number of votes the question has received. Below this list is the full text of the selected question.

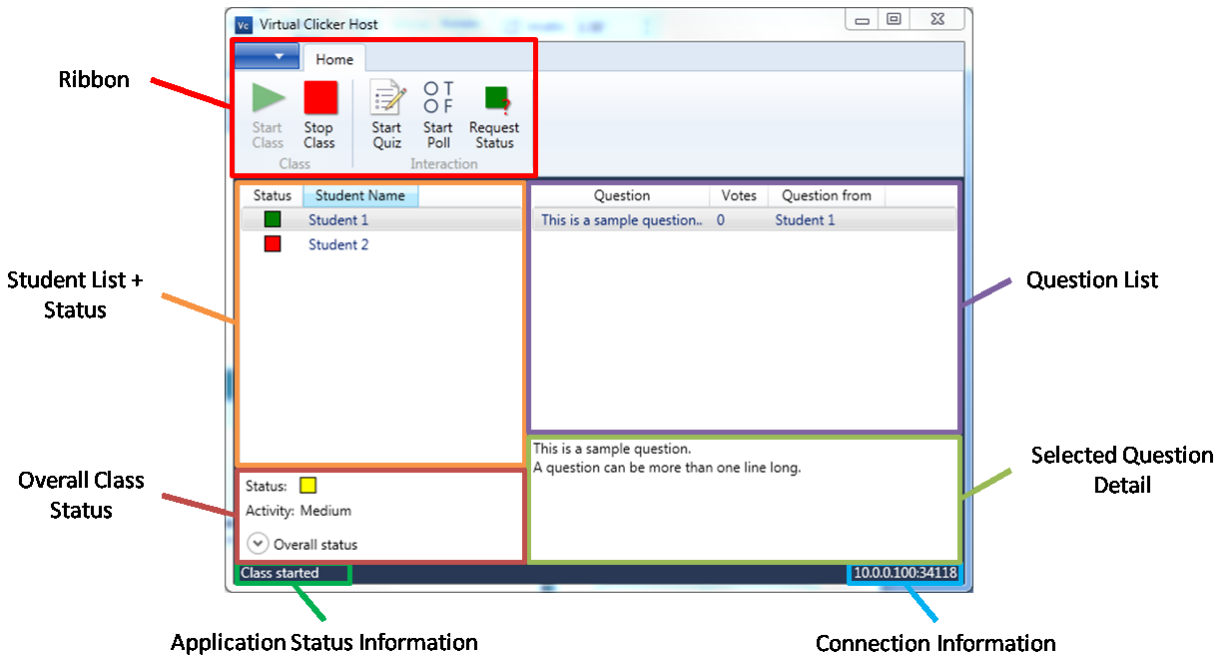


Figure 32 - Instructor Host UI

Starting a Class

Follow these steps to start a classroom session:

1. Click the "Start Class" button in the application's ribbon. This will cause the class settings dialog to appear as seen in Figure 33.
2. Enter a class name or choose one from the drop down (this drop down will be populated from previously entered class names).
3. Change the instructor name if desired (this will be auto populated from the name given the first time this application was run).
4. If the class session should be password protected (this will require that the students know the password to connect), select the check box next to "Password Required" and enter in a password.
5. Click the "Start Class" button. This will cause the classroom session to be started. Students should be able to auto detect the session but if they can't the connection information is displayed in the bottom right hand corner of the Instructor application as

seen in Figure 32. This information is displayed in the form X.X.X.X:YYYY, where the part before the colon is the IP address of the host and the part after is the port number the session is running on.

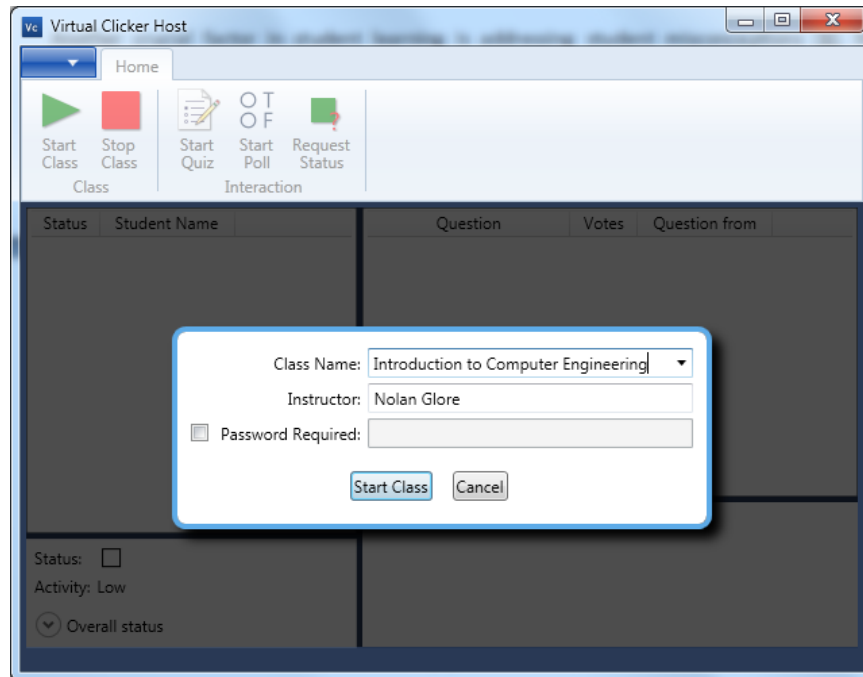


Figure 33 - Instructor Host Start Classroom Session

Stopping a Class

To stop a classroom session, click the "Stop Class" button in the application's ribbon. This will cause all students to disconnect. Closing the application will also cause a classroom session to end.

Giving a Quiz

Follow these steps to give a quiz:

1. Click the "Start Quiz" button in the application's ribbon. This will cause the give quiz window, shown in Figure 34, to open.
2. Click the "Browse" button and select the quiz document to send to students.

3. The give quiz window has a timer which can be used by instructors to time how long students have to finish the quiz. By default the timer is set to five minutes. Use the "+" and "-" buttons to increase or decrease the allotted time.
4. Click the "Start Quiz" button. This will cause the quiz document to be sent to every student and the timer will start counting down. As students start submitting completed quizzes, the received indicator just below the timer will display how many quizzes have been submitted out of the total number of students.
5. Click the "Stop Quiz" button to stop the quiz. Once this is clicked, students will no longer be able to submit completed quizzes and all quizzes collected at that point will be saved to a single file under "%userprofile%\documents\VirtualClicker\TakenQuizzes". The name of the file will be in the format "QuizFileName-date-time.qzs".

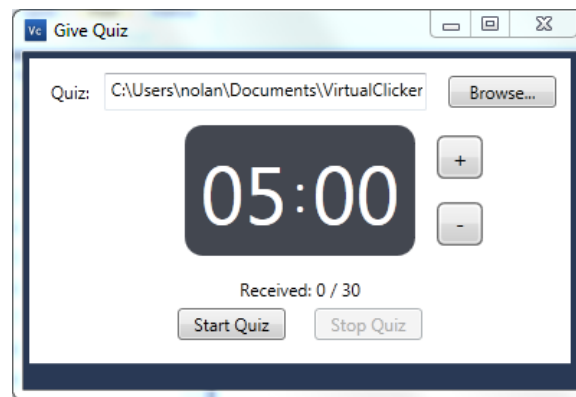


Figure 34 - Give Quiz UI

Giving a Poll

Follow these steps to give a quiz:

1. Click the "Start Poll" button in the application's ribbon. This will cause the give poll window, shown in Figure 35 - Give Poll UI, to open.

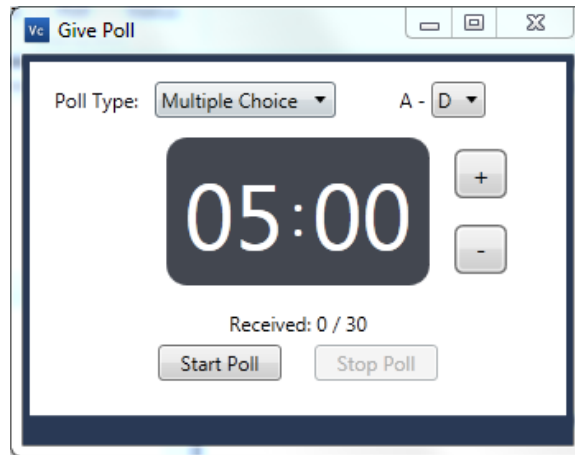


Figure 35 - Give Poll UI

2. Choose the type of poll to give by selecting an option in the "Poll Type" dropdown. The options are true/false, multiple choice, and multiple answer.
3. If multiple choice or multiple answer type is selected, select the number of possible answers by selecting a value from B to Z from the letter dropdown. There is no place to actually enter the question or the choices, so instructors need to communicate this in some other way (e.g., PowerPoint slide, white board).
4. The give poll window has a timer which can be used by instructors to time how long students have to finish the quiz. By default the timer is set to five minutes. Use the "+" and "-" buttons to increase or decrease the allotted time.
5. Click the "Start Poll" button. This will cause a single quiz document to be generated and sent to every student and the timer will start counting down. As students start submitting completed polls, the received indicator just below the timer will display how many quizzes have been submitted out of the total number of students.
6. Click the "Stop Poll" button to stop the poll. Once this is clicked student will no longer be able to submit answers to the poll. This will also cause the give poll window to display the results of the poll in the form of a pie chart, shown in Figure 36.

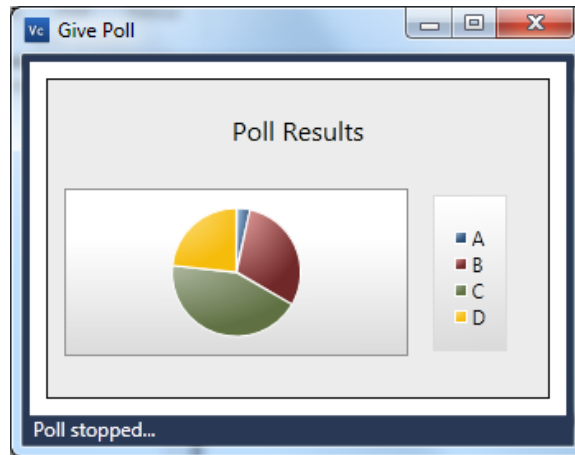


Figure 36 - Poll results UI

Student Status

The Virtual Clicker's student status feature provides a quick, reliable way for instructors to ask students if they understand the information presented. A key aspect that makes this status more reliable is the fact that a student's response is anonymous to everyone but the instructor. Student status is represented as one of four colors: white, which indicates a student has not set their status, red, yellow, or green. This status is shown next to each student's name in the student list, shown in Figure 37. There is also an overall class status which is shown in the overall status pane. This overall class status is calculated by taking the median of the red/yellow/green status values for each student.

Instructors have the ability to request status from students at any time by clicking on the "Request Status" button in the application's ribbon. This causes the status for each student to be reset to none/white and students will be prompted to set their status.

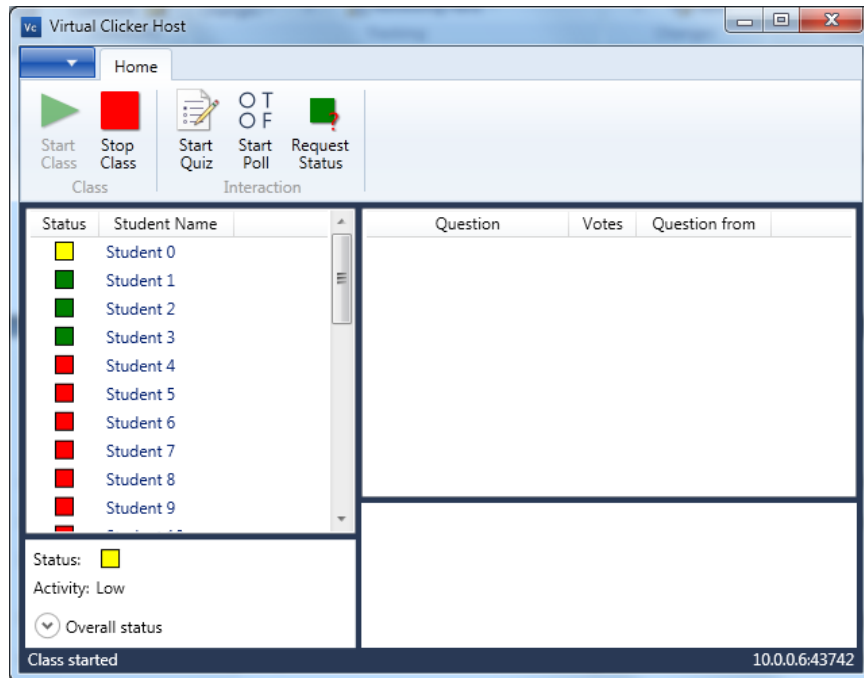


Figure 37 - Student Status

Student Activity

Each Student client application tracks activity levels of pen and keyboard input and periodically reports these levels to the Instructor Host application. The activity levels collected from each student are averaged and this average is used to determine if the class activity is low, medium, or high. This overall class activity level is displayed in the overall class status pane.

Student Questions

The Virtual Clicker application suite allows students to submit questions to the instructor electronically. When a question is submitted it shows up in the student question list, shown in Figure 38. The question is also distributed anonymously to all the other students in the class which allows other students to vote for the question if they would also like to hear the answer.

The question list shows a snippet of the question, the number of votes the question received, and who asked the question. The list is sorted so the question with the most votes is at the top

of the list. To view the full text of a question, select it from the list and the full text will be displayed in the question detail pane which is below the list.

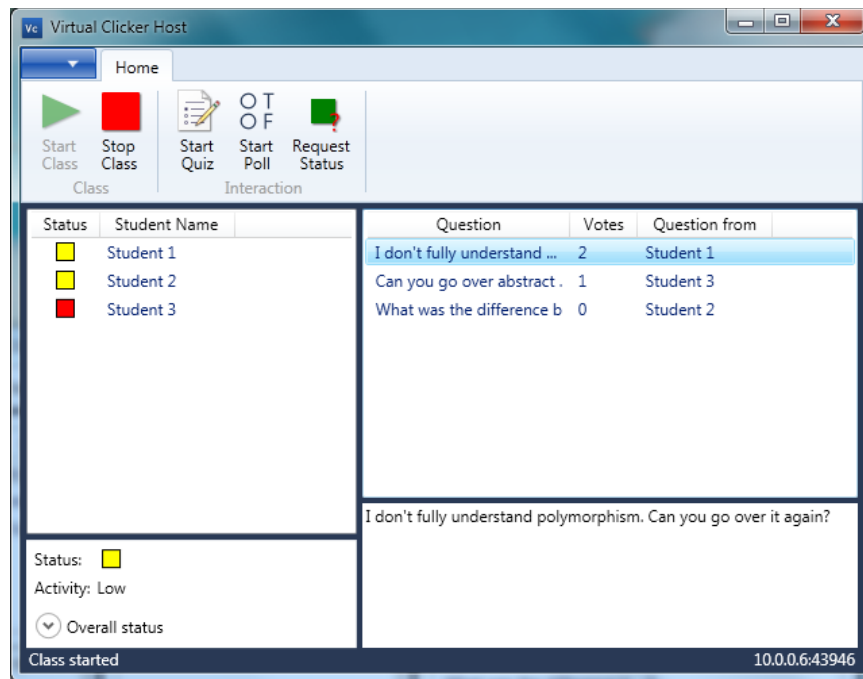


Figure 38 - Student Questions

Saving Student Questions

It's not always possible to answer every question during class. To save the list of received questions:

1. Click on the blue button with the down arrow that is to the left of the "Home" ribbon tab. This can be seen in Figure 39.
2. Click on the "Save Questions" button and choose a text file to save the questions to.

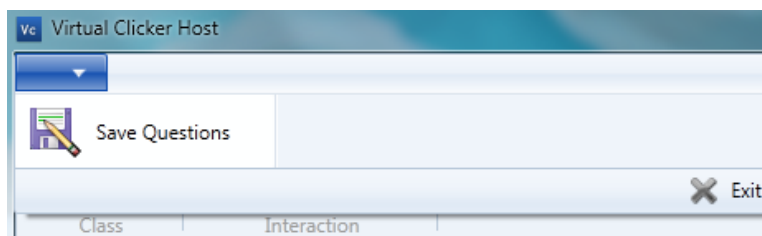


Figure 39 - Save Questions

Student Client

This section of the appendix provides detailed instructions on how to use the Student application. This application allows students to connect to virtual classroom sessions hosted by instructors. It allows students to take quizzes and polls, set their status, ask questions, and vote on questions from others.

Starting Student Client

The Instructor Host application can be launched from the start menu by choosing: Start → All Programs → Virtual Clicker → Virtual Clicker Student Client. Figure 40 shows the Student Client highlighted in the start menu.

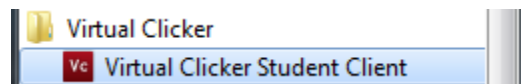


Figure 40 - Starting Instructor Host

The very first time the application is launched, the student will be prompted for their name as shown in Figure 41. The application stores this information which is sent to the Instructor application for displaying in the student list when connecting to hosted classroom sessions.

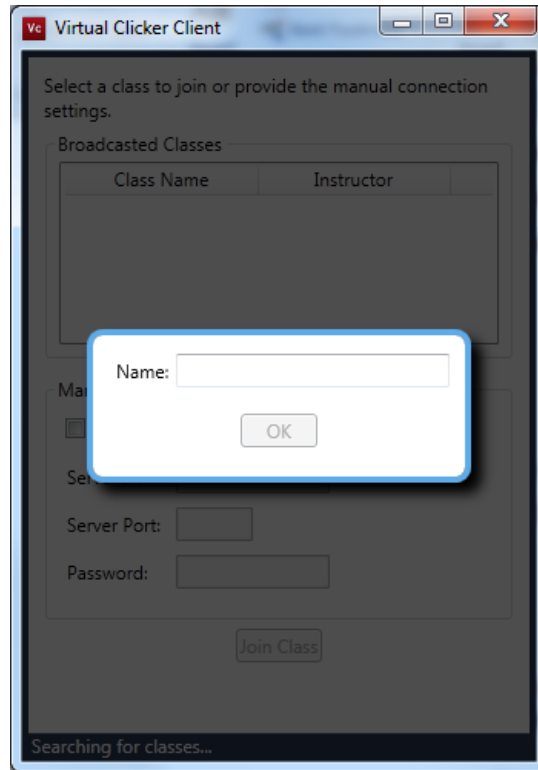


Figure 41 - Student Client First Run

Joining a Class

When the Student Client application starts it searches for any currently hosted virtual classroom sessions. This search takes about 20 seconds to complete. Once it's done any classes found will be displayed in the broadcasted classes list. Students can join a classroom session by either choosing one from the list of classes found or by manually entering connection settings.

Joining a Broadcasted Class

To join a classroom session that was found and displayed in the list of broadcasted classes:

1. Select the class to join from the broadcasted classes list as shown in Figure 42.
2. Click the "Join Class" button. This will take a few seconds and the application will be updated to the in class UI.
3. If the classroom session requires a password, the user will be prompted to enter it.

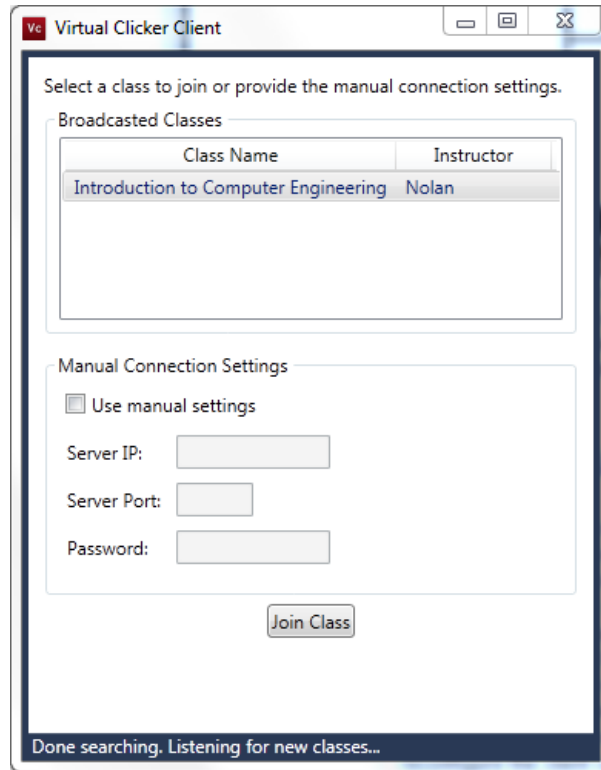


Figure 42 - Joining Broadcasted Classes

Joining a Class Manually

To join a class by manually entering connection settings:

1. Select the "Use manual setting" check box as shown in Figure 43.
2. Enter in the IP address of the PC hosting the virtual classroom session. This value can be obtained from the connection information area of the Instructor Host application.
3. Enter the port number of the hosted virtual classroom session. This value can be obtained from the connection information area of the Instructor Host application.
4. Enter the password for the classroom session if required.
5. Click the "Join Class" button. This will take a few seconds and the application will be updated to the in class UI.

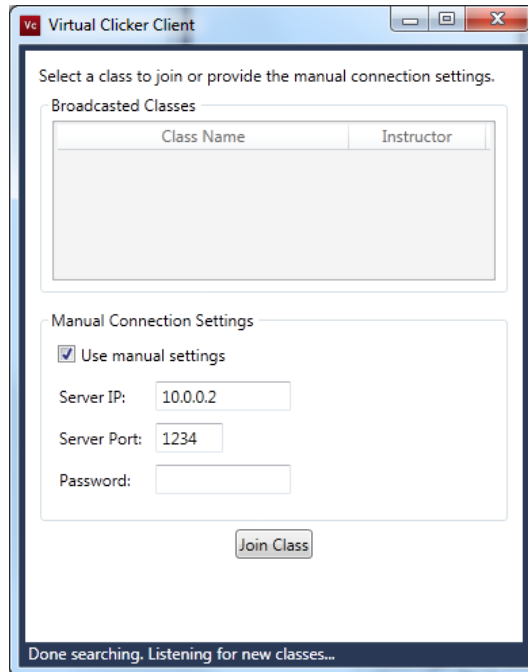


Figure 43 - Manually Joining Classes

Leaving a Class

To leave a class, click the "Leave Class" button in the application's toolbar. Closing the application will also cause the student to leave the classroom session.

Setting Status

To set student status, click the "Status" drop down in the application's toolbar and choose between none, red, yellow, and green, as shown in Figure 44. The change in status will be sent to the instructor.

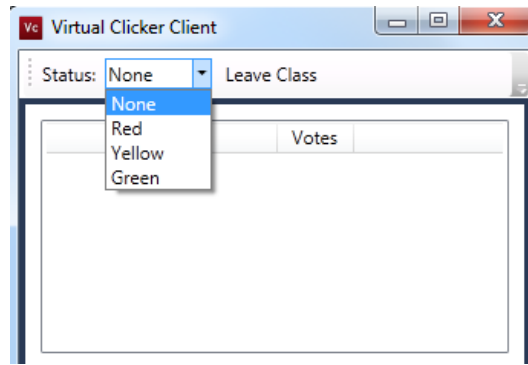


Figure 44 - Setting Student Status

Taking Quizzes and Polls

When an instructor starts a quiz or a poll, it gets sent to all connected clients. When the Student application receives the quiz, it displays the quiz taker window, shown in Figure 45. This window contains the quiz document which allows students to answer questions and a ribbon which contains buttons for submitting the quiz and ink editing controls.

To take and submit both quizzes and polls:

1. Complete multiple choice questions by clicking on the radio button next to the correct answer.
2. Complete multiple answer questions by clicking on the check boxes next to each correct answer.
3. Complete essay questions by typing in the empty text box.
4. Complete ink questions by completing the drawing on the canvas. Use the Ink control buttons in the ribbon to switch between the pen, highlighter, and eraser input modes and to control the ink color and thickness.
5. To send the completed quiz to the instructor, click the "Submit" button in the ribbon.
Note: if the instructor has already stopped the quiz, the "Submit" button will be disabled.

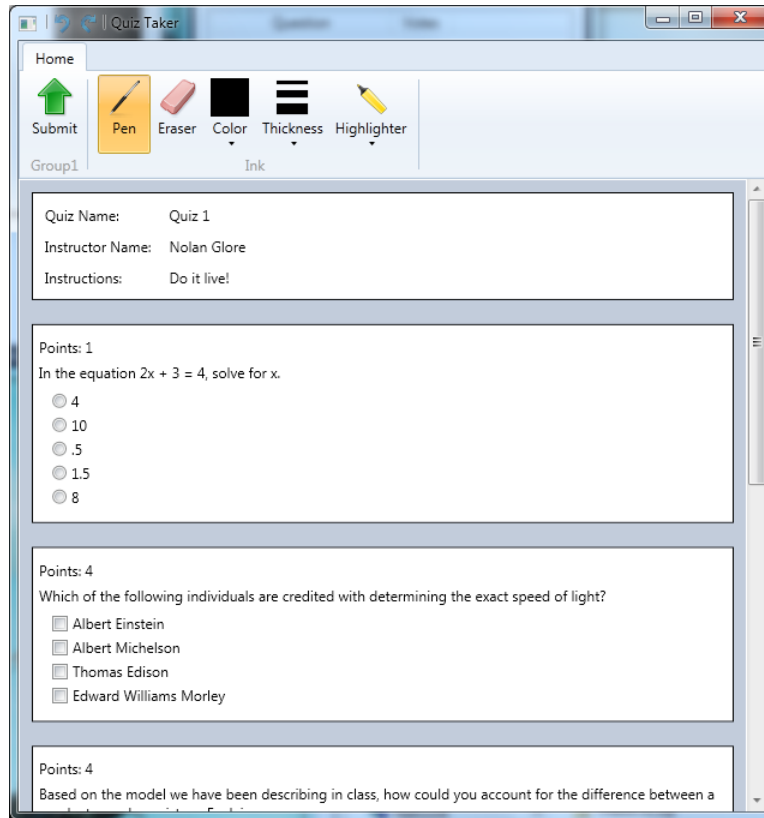


Figure 45 - Quiz Taker UI

Asking Questions

The Student application allows students to submit questions to the instructor electronically. To ask a question, type the question into the text box at the bottom of the application and click the "Ask Question" button, as seen in Figure 46.

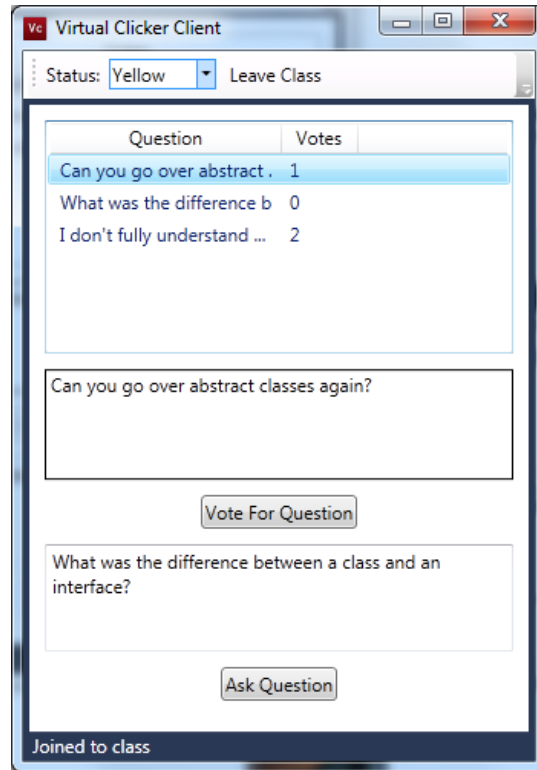


Figure 46 - Asking Questions

Voting on Questions

Once the Instructor Host application receives student questions, it sends them anonymously to all the students joined to the classroom session. These questions show up in the question list as seen in Figure 46 which displays a snippet of the question and the number of votes the question has received. The questions are sorted so the most recent ones are on top. To see all of the question's text, select it from the list and it will be displayed in the text box below the list. Students can vote for a given question if they also want to hear the answer. This helps instructors answer the questions that most students care about.

To vote for a question:

1. Select the question from the list.
2. Click the "Vote For Question" button. Note: A question can only be voted for once by each student.

Conclusion

The installers for the Virtual Clicker Instructor application suite and Student application can be downloaded from <http://filebox.ece.vt.edu/~jgtront/virtualclicker/index.html>.