CSbots: A case study in introducing educational technology to a classroom setting

Tom Lauwers Illah Nourbakhsh Emily Hamner

tlauwers@andrew.cmu.edu illah@ri.cmu.edu etf@andrew.cmu.edu

CMU-RI-TR-08-41

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

October 2008

Abstract

We present CSbots, an ongoing project to use robots as educational tools in the Introduction to Computer Science (CS1) course. Enrollments in Computer Science have dropped significantly, and part of the blame rests on an introductory course that has not kept pace with the progress of computational technology. We aim to use robotics to motivate students, provide a sense of relevance to the course work, and improve learning.

In our curriculum development process we use an iterative cycle composed of design, pilot, and evaluation phases. During our initial evaluation phase we assessed existing introduction to computer science courses by conducting a survey of educators and textbooks. We describe our robotic technology, software framework, and curriculum developed during our first design phase. Pilots were conducted at two community colleges in Fall 2007 with the revamped curricula and using the newly developed robotic platform. These pilots were evaluated using a number of qualitative and quantitative methods. The goals of the evaluation were to determine the impact of the robotic technology on student retention and interest in the course content, as well as to ensure that learning as measured by exams did not decline. The results of our evaluation and subsequent implications for our next design cycle are presented.

Contents

1 Background

In the context of steeply declining enrollments in Computer Science [1], we are focusing on developing curricular modules for introduction to Computer Science (CS1) classes in which robots are used as educational tools to motivate students about applications of Computing.

Robots, as physically manifested computing devices, inherently show students how computing algorithms can impact the real world; they provide a degree of relevance to assignments that is often missing. Even so, there are well-known weaknesses to using robots in computing courses [2]: Robots are typically too expensive for student ownership, and so students must work on robot programming assignments in labs with limited hours. Feedback is delayed due to the real-time nature of robotics, and so students must devote more time to tedious debugging, and less to developing solutions. The ingredients needed to overcome these limitations are achievable: New hardware technologies that enable high functioning robotics at a low cost; a software architecture that is student-friendly and can be peeled away to reveal layers of deepening complexity as students learn and become more sophisticated users; and a simulator to provide immediate debugging feedback.

We submit that the reason that robotics-based curricula have not succeeded more widely in CS1 classes is a lack of alignment between the features and capabilities of the robotic technology and the needs of the curriculum and students. Our goal is to develop a curricular framework with an accompanying robot whose technological features have been selected specifically to support the needs of the class, students, and curriculum.

1.1 Educational uses of robots

Robots have a twenty year history as educational tools; used at all grade levels, they are generally seen as motivating interest in science and technology activities. Although initially used mostly in out-of-school activities, they have recently gained a foothold in class environments.

1.1.1 Out-of-school

The best-known application of educational robotics is the robotics competition. These competitions have grown explosively at middle and high schools over the last decade, with current participation in the largest programs reaching 75,000 students. Although competitions vary along a number of characteristic, most require students to use a standard controller and a kit of parts, so as to place teams on the same footing regardless of financial strength. Although it is beyond the scope of this review to describe every robot-based tournament, it is worthwhile to consider the two largest such competitions; FIRST [3],[4] and Botball [5],[6]. The FIRST organization hosts three competitions, a high school league in which students receive a kit of parts and controller developed by Innovation First [7], the FIRST lego league in which elementary and middle school students build robots using the Lego Mindstorms kit, and the FIRST Vex challenge which uses a robotic assembly kit developed by Innovation First; Botball uses an advanced controller and legos for construction material. In all cases, the competition rules are revised annually, and students are given no more than two months from the announcement of the competition details to design, build, and test their robots. Tournaments occur on a local and national scale, and tend to mimick sporting events complete with mascots, cheering fans, and intense competition between teams.

Recently, recognition that robot competitions do not appeal equally to all students has led to the development of a second approach to attracting students to STEM (Science, Technology, Engineering, and Math) with robotics. In a yet-to-be published paper, Rusk et. al. [8] lists four key characteristics of this approach:

1. Focus on themes, not challenges. Themes should be broad enough to allow students to connect their personal interests to the design, while narrow enough to foster the feeling of a shared experience.
2. Combine art and engineering. Most students have had experience building for art and are familiar with craft materials; by incorporating these the technology becomes an addition to a process with which the students are comfortable.

3. Encourage storytelling. Children's play modes can be divided into two types, patterners and dramatists. Patterners are already attracted to robot competitions, to capture a new audience it is important to appeal to dramatists.

4. Organize exhibitions. The public display of end-products is an appealing feature of competitions, and should be retained by organizing public exhibitions.

Artbotics [9] is an example of a current program incorporating the above recommendations; in Artbotics high school and undergraduate students are tasked with creating interactive art exhibits.

1.1.2 Robots in Computer Science Education

Recently, and partially in response to the declining enrollment figures mentioned previously, robots have become a major tool in computer science education, notably in the first [2] and second [10] level courses (CS1 and CS2 respectively). At Carnegie Mellon, a student taught course [11] in which programming robot behaviors and creating robotic art are the main activities attracts nearly half its students from the humanities, business, and art schools. A high school summer course [12] in which students built a robot from a kit and then learned to program it to exhibit progressively more complex behaviors found increased student motivation to pursue further study in science and technology. A major study of the use of robots as educational tools at the US Air Force academy [2] found that students using the robots did not show improved learning or retention; however, the authors suggested that this was likely due to the necessity of having students work in closed, time-limited labs. Taking a cue from this study, a new Computer Science course initiative at Georgia Tech and Bryn Mawr proposes to provide every student with a personal robot [13].

2 Approach

Our approach to creating the CSbots program rests on three principles; feedback and evaluation driven iteration of design, deep partnerships with educators working in the Computer Science field, and alignment of the robot and curriculum with learning goals and audience of the CS1 class.

2.1 Iteration

We are engaging in an iterative design process with extensive participation from computer science educators at two and four year schools. The process is composed of the following steps:
Design. The design step involves the creation of a robotic platform and associated software, and the development of assignments and course outline.
Pilot. The pilot phase involves the pilot of the designed curriculum and technology in a CS1 course.
Evaluation. Learning, motivation, and retention are tracked with standard exams, weekly student surveys, and comparisons of drop-out rates. These data are used to inform the next design step.

2.2 Partnerships

As our design team has no experience teaching the CS1 class, we felt it greatly important to engage with educators of CS1. Our intention was to develop a partnership that was more than simply advisory, with a two-way sharing of domain knowledge and skill. Our partners would be involved from the start of the design process, allowing them to test and comment on early versions of the robot and software, while we would have access to their existing curricula and their crucial understanding of what works with students in the CS1 class. While the details of the robot design were left up to us, we worked together to design the interface of the software framework, and the curricular activities. Such a partnership takes time to both develop and maintain, and we are pleased to have found partners in the educator community to work with.

We are currently working with two educators teaching CS1 at community colleges: Don Smith at the Community College of Allegheny County in Pittsburgh, Pennsylvania, and David Patrick at Ohlone Community College in Fremont, California. These educators have become true design partners, suggesting changes to software and hardware, coming up with assignment ideas, and according us opportunities to pilot our curriculum.

2.3 Alignment

Alignment is traditionally a guiding principle of curriculum design; essentially it advises the course designer to align the learning goals, instruction, and assessment, so that each supports the other. An example of an unaligned class is one in which a teacher has given an exam that assesses materials or concepts that were not covered by prior lectures or exercises. Although in some cases classes may be intuitively aligned by their designers, it is a practice that can be performed in an intentional, explicit way [14].

Alignment is a powerful concept that can be extended to the design of educational technology. The features of a technological system provide a fourth design node, feeding back on and aligning with the three traditional nodes. In a very real sense, a robot can enable new instructional methods, new assessments, and even new learning goals; similarly, learning goals, instructional methods, and assessments all affect the technology's interactivity and design.

3 Initial Evaluation

To ensure that our designs were grounded in the realities of Computer Science education we engaged in extensive pre-design evaluations. These evaluations included a textbook survey that sought to discover unifying themes and learning objectives, a survey of educators to characterize curricular design constraints at different institutions as well as receptiveness to using robotics in the CS1 class, and an analysis of our partner educators' assignments and lectures.

3.1 Textbook Survey

In spring of 2006 we performed an analysis of ten major CS1 textbooks to help us develop a conception of the standard CS1 curriculum and discover the major learning objectives of the class. This analysis sought to answer a number of fundamental curricular questions:

- Does the textbook assume a specific Integrated Development Environment (IDE)?
- Does the textbook focus on the science of computing or is it a trade book focused on teaching students how to program?
- Are the problem sets cumulative or modular?
- Does the textbook require the use of additional hardware or software?
- What is the order and list of topics covered?

We used three sources to determine popular introductory CS texts. The first was a survey of web sites of CS1 courses at four-year universities across the United States. From a group of about 60 CS1 instructors, about 30 had accessible course web sites. The next measure of popularity was to look at the textbooks used at top universities such as MIT, Stanford, Harvard, etc. Lastly we emailed the CS1 instructors and the assistant dean of undergraduate education in the CS department at Carnegie Mellon University (CMU) to ask them which textbooks they thought were the most popular and widely used. Table 1 presents the top ten textbooks identified using these methods.

Textbook	**IDE**	**Problem Sets**	**Additional Resources**
Head First Java [15]	none	mostly modular	no
Java How To Program [16]	none	modular	no
Big Java [17]	BlueJ suggested	mostly modular	internet connection
Objects First with Java [18]	BlueJ required	cumulative	hooks to outside resources
Java Concepts [19]	BlueJ suggested	mostly modular	internet connection
C++ Programming [20]	none	mostly modular	no
Java Software Solutions [21]	none	modular	no
Computer Science [22]	none	modular	no
The Art and Science of C [23]	none	modular	additional libraries
Absolute Java [24]	TextPad suggested	modular	no

Table 1: Summary of key textbook attributes

Results. From table 1, only one book assumed that students were using a specific IDE, although several others suggested and even came with IDEs. All of the books were centered on teaching the skill of programming in a specific language, not on high-level computing concepts, and so they were all classified as trade books. Nine of the ten books had problem sets that were mostly modular; exercises from one chapter were not based on code written in a previous chapter. Most textbooks required no additional hardware or software, although two books assumed an internet connection was available, and one came with non-standard software libraries for use in some of the book assignments.

We analyzed the order of 6 topics that were covered in most of the textbooks and that have potential for self-contained robotics modules. Those 6 topics are:

V: Variables
I: Simple input/output (I/O) (screen I/O with characters or strings)
F: Flow (conditionals, loops, relational operators)
D: Arrays
E: Exceptions and errors
A: Advanced I/O (file I/O)

The most common ordering of these topics was: Variables, simple I/O, flow control, arrays, exceptions, and advanced I/O. The topics were found in this order in 4 of the 10 books. Variables and simple I/O were introduced first in all books but 1, with simple I/O coming before variables in only 3 cases. Flow control was the third topic in all but 2 books. The fourth topic was arrays in 6 of the books. In the other 4 books, arrays was the fifth topic. Exceptions came after arrays in all but two cases (once it was directly before arrays, and once exceptions was missing entirely). Advanced I/O was the last topic in 7 of the books. (See Table 2)

Number of Books	**Topic Order**	**Programming Languages**
4	V I F D E A	Java, Java, Java, Java
1	I V F D E A	Java
1	V I F E D A	C
1	I V F A D E	Java
1	V I F A D -	C
1	V I A F D E	C++
1	I F V D E A	Java

Table 2: Topic orderings. The topics are variables (V), simple I/O (I), flow control (F), arrays (D), exceptions and errors (E), and advanced I/O (A).

Implications and Conclusions. All of the books focused on programming skills and not computing concepts, and most presented a modular approach. This analysis implies that a broadly applicable curriculum should be composed of unrelated modules covering specific programming concepts. These modules could be picked up by educators and placed into their curriculum, regardless of their course schedule or textbook used. Based on our textbook analysis, modules with the following topic dependencies would likely fit into many CS1 courses:

Module 1: VI - Variables and simple I/O
Module 2: VI F - Flow control (with variables and simple I/O as prerequisites)
Module 3: VIF D - Arrays (with variables, simple I/O, and flow control as prerequisites)
Module 4: VIF E - Exceptions and error handling (with variables, simple I/O, and flow control as prerequisites)

3.2 Survey of Educators

We embarked on an extensive survey of educators at two- and four-year institutions [25] during the 2005-2006 school year. We were inspired by the Taulbee survey [26] and the McCauley and Manaris [27] studies, but our aims differed from these; instead of a broad-based, largely quantitative analysis of the state of Computer Science education, we were interested in analyzing the attitudes, opinions, and challenges faced specifically by CS1 educators. Our study sought to answer questions that were best asked in the context of a personal interview, and best analyzed through conceptual code-based qualitative metrics. The foundational questions that we sought to answer included:

- How do CS1 instructors feel about the effectiveness of their curricula, both in teaching students and in motivating them?
- To what degree are instructors able to make curricular changes?
- What are the typical dynamics and logistics of a CS1 course?
- What tools and programming languages do instructors currently use and what is their relative popularity?
- Are instructors interested in using robotics as a teaching tool?
- How do the classroom realities exposed by the previous questions inform methods for introducing new educational tools?

We interviewed 33 educators at four-year colleges and universities, and 4 educators at community colleges across the United States - their geographic distribution is mapped in figure 1. The educators came from a wide range of public and private, small and large institutions and were themselves varied with respect to age, gender, and professional standing. Our results provided tentative answers to the foundational questions and feed into an answer to our primary question: What methods and strategies should we employ in our attempt to introduce a new educational tool into the CS1 curriculum? Our conclusions can be split into those general to introducing a new educational tool and those specific to introducing robots.

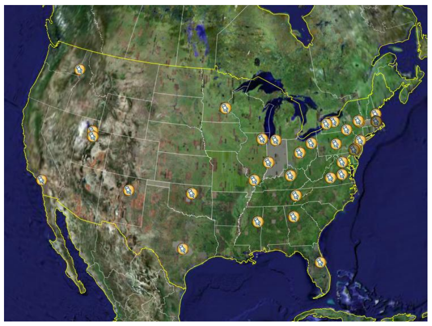


Figure 1: Distribution of Survey Respondents

General Conclusions

- A major curricular change is difficult to implement, requiring buy-in from the department faculty, and sometimes from other departments. Tools should therefore be introduced with a curriculum that is similar to that already in use.
- Tools should not be tied to a new or relatively rare programming language; not only does this require a major curricular change but few instructors are planning to change the programming language they use. Support for either Java or C++ is required for widespread adoption.
- Any new educational tool must integrate into the class such that it does not significantly increase student workloads.
- Given that most educators use textbooks as guides when embarking on a significantly different curriculum, it may be useful to either develop or adopt an accompanying textbook and lab manual for the educational tool.

Conclusions Specific to Using Robots in CS1

- Students must be able to work on their out-of-class assignments at home. This is not an impossible goal for a robotic tool; this challenge can be met with a number of methods; the use of a simulator, remote or tele-present access to physical robots, or by providing each student with a very low cost robot.
- Developing for Java provides access to the largest and most enthusiastic base of potential robotics adopters.
- Materials cost to students is widely perceived as more important than departmental costs, and so we do not believe that a robotics class should require students to purchase a robot unless it is in lieu of a similarly priced textbook.

The purpose of the educator survey was both to discover the realities of the group for whom we were developing the technology, and to ensure that members of this group were interested in our proposed technology. In these senses the survey was successful at identifying key logistical difficulties of introducing robots in CS1, as well as establishing that a significant fraction of educators were willing and able to try such an intervention.

3.3 Partner's Prior Curricula

In addition to the formal textbook and educator surveys, we also worked with our partner educators to detail the learning objectives, instructional activities, and assessments in their specific courses. This was important not just as a pre-design evaluation method, but also because it was our intention to pilot the revised course with our partners in such a way as to not change the learning goals. In the case of CCAC, our intentions were realized; we went through every exam, assignment, and prepared lecture used by our partner in the previous year. It was our goal to create a curriculum that would mirror this class at both the macro and micro levels - the learning goals of the class were to remain the same, as would the complexity and learning goals of the individual weekly assignments. Similarly, the assessments would cover the same material at the same point in the course schedule. By keeping constant much else of the class, it was easier to measure the effect of the robotics activities on the students. It also simplified the alignment process; as it was impossible to modify the learning goals or assessments, all that was necessary was to ensure that the instructional activities aligned with these two. At Ohlone college, we were unable to pilot a revised course, but were still able to access curricula from previous versions of their CS1 class.

4 Initial Design

We used the results of our initial evaluation and analysis to simultaneously design a curriculum, robot platform, and software for the CS1 classes of our partner educators.

4.1 Curriculum

Week	Goals	Assignment
1	Learn how to compile and debug programs. Cover primitive data types.	Write a program that causes the robot to speak.
2	Use simple arithmetic operators and Boolean expressions and print variables. Use comment-based documentation.	Debug a provided program.
3	Learn about classes by using two examplars; the String class and the Robot class.	Make robot weather forecasters that say the forecast and move based on weather conditions.
4	Continue learning to use classes by covering more examples.	Move in a regular polygon from 3-10 sides.
5	Learn to allow input from screen and review of Boolean/logical expressions.	Move and turn from user input.
6	Use conditional statements like *if* and *switch*.	Have the robot react to a weather forecast based on a randomly generated mood.
7	Use looping statements like *for*, *while*, and *do-while*.	Wander around the room; if robot runs into something it should apologize and try to move around the object.
8	Learn to write non-main methods.	Create a generator of stories composed of randomly selected sentences.
9	Learn to write custom classes.	Write a simple simulator.
10	Create graphical programs with Swing, including handling button events.	Create a tele-op interface.
11	Learn to read text/string inputs from a graphical interface.	Improve the tele-op interface with a text input box; text placed in the box should be spoken by the robot.
12	Use arrays of primitive data types.	Script a series of motions on the robot by writing a program that reads bumper and button presses and then plays a sequence of motions based on those presses.
13	Use arrays of objects.	Write a program that tracks a color - each pixel is an object.
14	Develop a surface understanding of exceptions and exception handling.	Combine the wandering program with the tele-op program so that the robot is autonomous until it strikes something, then throws an exception and moves to tele-op control.
15	Review.	No assignment.

Table 3: Schedule for a CS1 Robotics course

Curriculum design was guided by several conclusions from our survey; educators teaching Java were most enthusiastic about using robots, most educators wanted a new curriculum to be linked to a textbook, and robots could be introduced into a class with minimal administrative paperwork so long as the programming language did not change and the course schedule was minimally affected. We focused on Java and chose a specific textbook, "'A Guide to Programming in Java"' [28], to link to our curriculum. We took care to ensure that each assignment developed was focused on teaching CS concepts; we used the prior assignments from our design partners as a baseline and analyzed these to ensure our assignments were similar conceptually. Table 3 contains the tentative schedule for a semester-long CS1 course with robot-based assignments.

Prior to testing our assignments in a pilot, we ensured that they were at an appropriate level for our intended audience; we had a high school student with one semester of Java experience complete solutions for each assignment. Although most assignments were found to be appropriate to the level of our intended learners, this early testing did result in several changes made to the later assignments to reduce tangential complexity and focus more heavily on core concepts.

4.2 Robot Platform

The end goal of our technology development process is to create a robot platform and accompanying software that has the correct sensing and acuating interactions for the introduction to computer science class while simultaneously being sufficiently low cost to be used in such classes. In line with these goals, during our first design cycle we focused on maximizing the number of testable sensing and actuation interactions included on the platform while keeping the cost per platform below $1000 so as to allow sizable initial pilots. This strategy allows us to test individual features during our pilots and determine an optimal feature set to guide our second robot design.

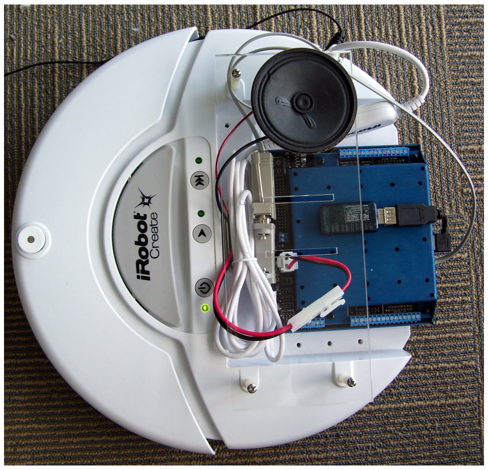


Figure 2: The first year robot platform.

The platform that we are using in the first design cycle is an iRobot Create[1] combined with a Qwerk controller[2] (figure 2). The iRobot Create is a robotic platform based on the popular Roomba line of robotic vacuum cleaners. The Qwerk, developed by Charmed Labs and the CREATE lab, contains circuitry for low level robot control, as well as firmware allowing instant teleoperation over a wifi network with the addition of a USB wireless device and USB webcam. The Qwerk is accompanied by the TeRK software environment[3], an extensive open-source software base for developers. Although not the lowest cost solution, the combined platform has the advantage of using off-the-shelf technology and maximizing feature richness. The combined platform has the following capabilities:

- A wireless tether, allowing programs to reside on a student's computer
- Bumpers for simple obstacle detection
- Encoders to track distance traveled and maintain equivalent wheel speeds
- Vision via a USB webcam
- Audio, including the playing of audio wav files and the generation of speech from text
- An LED array as well as a color and intensity variable LED
- Access to real-time RSS feeds, enabling programs to respond to internet events

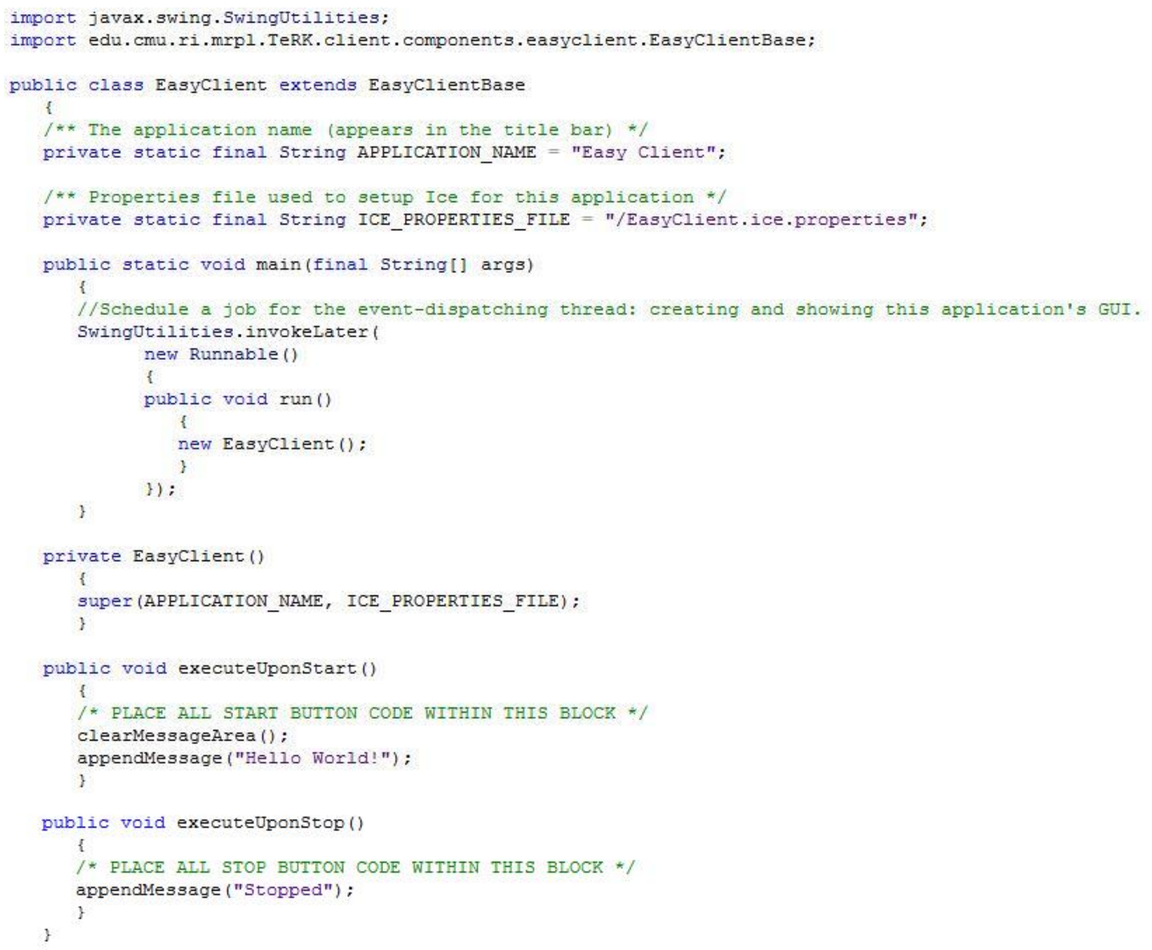
[1]http://www.irobot.com/sp.cfm?pageid=305
[2]http://www.charmedlabs.com/index.php?option=com_content&task=view&id=29
[3]http://www.terk.ri.cmu.edu/software/index.php

4.3 Software

The software environment we created was heavily influenced by the feedback we received from our partners as we were designing the environment and the constraints imposed by using the TeRK software environment. Unlike standard small Java programs which require little or no additional software libraries to run, our student-written Java programs would rest on top of several layers of increasingly advanced code; this code was necessary for connecting over the wireless network to the computer, for sending appropriately 'phrased' commands to the robot, and for supporting the graphical interface which was part of the software package.

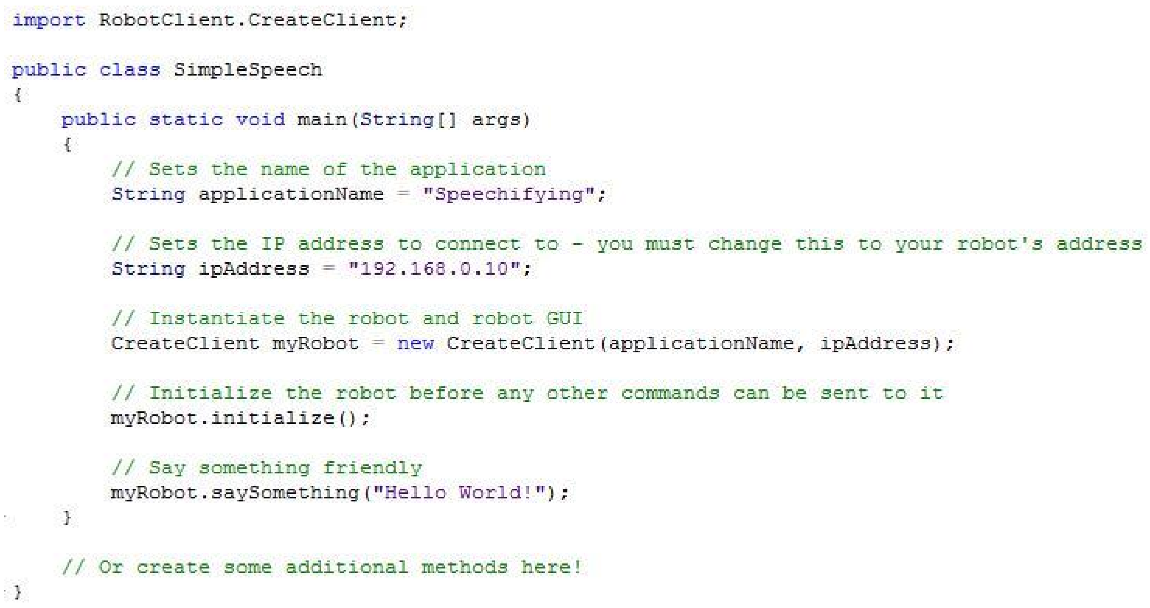
One of the major themes in our conversations with our partner educators as we were developing the code was to simplify as much as possible the top, or student visible level. Our first attempt to create a top-level file that students could edit is shown in figure 3; our partners objected to this file on the grounds that it was much too complicated and had too many unexplainable sections for beginning CS students. Several months of refinement followed, eventually producing the starter file in figure 4. Figure 4 shows a comparison between a program which causes the robot to say 'Hello World' and the standard 'Hello World' print program, and highlights the additional code required to run our version. The software libraries required were packaged into a single .jar file which could be used by a program with a single 'import' statement. The four lines of extra code in the program are fairly easy to explain to a novice user; one specifies the IP address of the robot to connect to (a number which is written on the robot's bumper in large bold print), one specifies the name of the program, one instantiates the robot object, and the last initializes the robot. Although students are required to use object oriented programming methods to call robot based methods before objects are theoretically explained, this did not prove to be a source of confusion, as *myRobot.saySomething("Hello");* and *System.out.println("Hello");* are equivalently complex and mysterious to the beginning programmer.

As we were developing the software environment, special attention was directed at ensuring that all method calls to the robot were relatively clear, such that reading the name of the method gives a proper indication of what that method will do. Students were provided with both a full Javadoc style listing of all robot class methods, as well as with a quick reference that covered the most important methods. The quick reference is available in this document as Appendix A.

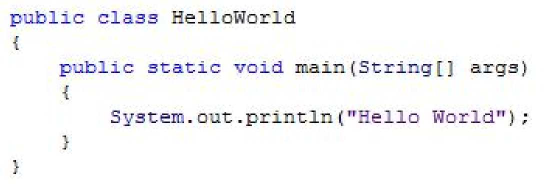


```java
import javax.swing.SwingUtilities;
import edu.cmu.ri.mrpl.TeRK.client.components.easyclient.EasyClientBase;

public class EasyClient extends EasyClientBase
   {
   /** The application name (appears in the title bar) */
   private static final String APPLICATION_NAME = "Easy Client";

   /** Properties file used to setup Ice for this application */
   private static final String ICE_PROPERTIES_FILE = "/EasyClient.ice.properties";

   public static void main(final String[] args)
      {
      //Schedule a job for the event-dispatching thread: creating and showing this application's GUI.
      SwingUtilities.invokeLater(
            new Runnable()
            {
            public void run()
               {
               new EasyClient();
               }
            });
      }

   private EasyClient()
      {
      super(APPLICATION_NAME, ICE_PROPERTIES_FILE);
      }

   public void executeUponStart()
      {
      /* PLACE ALL START BUTTON CODE WITHIN THIS BLOCK */
      clearMessageArea();
      appendMessage("Hello World!");
      }

   public void executeUponStop()
      {
      /* PLACE ALL STOP BUTTON CODE WITHIN THIS BLOCK */
      appendMessage("Stopped");
      }
   }
```

Figure 3: Initial skeleton file for software framework



```
import RobotClient.CreateClient;

public class SimpleSpeech
{
    public static void main(String[] args)
    {
        // Sets the name of the application
        String applicationName = "Speechifying";

        // Sets the IP address to connect to - you must change this to your robot's address
        String ipAddress = "192.168.0.10";

        // Instantiate the robot and robot GUI
        CreateClient myRobot = new CreateClient(applicationName, ipAddress);

        // Initialize the robot before any other commands can be sent to it
        myRobot.initialize();

        // Say something friendly
        myRobot.saySomething("Hello World!");
    }

    // Or create some additional methods here!
}
```



```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```

Figure 4: Programs which make the robot say hello world (top) and print Hello World to screen (bottom).

5 Pilots

After completing our curriculum and technology in the early summer of 2007, we prepared for two pilots, one with each of our partner educators. Prior to these full pilots, we ran two of the assignments in the summer 2007 course at CCAC.

5.1 CCAC Summer 2007

We conducted a brief week-long pilot during the regular CCAC Introduction to Computer Science course taught by our partner. During this period, two of the regular weekly assignments were tested with the students; specifically we gave the students assignments 7 and 12. Assignment 7 deals with looping and conditionals, a topic which the summer class had already covered, but which our partner suggested might be useful to repeat. Assignment 12 covers arrays, the topic that the class was in the process of covering during our trial.

5.1.1 Evaluation Methods

The goals of our evaluation of the summer pilot were to determine if the assignments we provided to the students were compelling, as well as to ensure that our survey materials were capturing the information we desired.

We surveyed students at the beginning of the course to determine their age, grade point average, prior programming knowledge, reasons for taking the class, and interest in using robots in the course. We also surveyed students at the completion of each robotics assignment to determine how interesting they thought the assignments were compared to other assignments, how confident they felt with the subject material covered by the assignment, and how easy they felt the assignment was. Lastly, we offered a very small amount of extra credit on assignment 12 for a reasonably difficult extension to the assignment to determine if students were interested enough in using the robot to do extra work with minimal grade-related motivation.

5.1.2 Evaluation Results

Our start survey had 16 respondents, while the surveys regarding the assignments had 9 and 12 respondents each; this disparity mostly reflects students dropping the course before our pilot began. The start survey provided validation of our questions regarding confidence with computing, prior programming knowledge, and reasons for taking the course, but only one notable and unexpected result; 75% of students thought programming a robot would be more interesting than programming a computer.

The results from our assignment surveys showed our approach to be promising and worth testing in a larger pilot in the fall. For both surveys, students were asked to rate on a 1-5 scale how confident they were with the concepts, how interested they were in the assignment, whether this was their favorite assignment of the class, and how easy or difficult the assignment was. Table 4 shows the averages for these results for assignments 7 and 12. As you can see from the table, both assignments got high marks for being interesting,

Assignment	**Confidence**	**Interest**	**Difficulty**	**Favorite so far**
7	4.22	4.22	2.67	7 yes, 2 blank, 0 no
12	3.58	4.08	4.08	7 yes, 2 blank, 3 no

Table 4: Confidence and Interest Ratings for Summer Pilot Robotics Assignments

and were considered the favorite assignments in the class by most students. Assignment 7 was listed as relatively easy, likely because the subject material was covered previously, assignment 12 shows that even with new subject matter, the students enjoyed the assignment. Ten of the twelve students in the class completed the assignment successfully; of the other two, one was unable to complete the assignment due to a medical emergency. Furthermore, six of the ten students who successfully completed assignment 12 finished the extra credit portion of the assignment.

5.2 Ohlone Fall 2007

The Ohlone pilot was conducted from early September through mid-December 2007. Fifteen students signed up for the class, but only four stayed in the class with a passing grade; although low, this is not an unusual retention rate for many computer science classes at both Ohlone and CCAC. The pilot was not framed as an introduction to computer science course, but as a robotics and AI course; even so, it provided us with an opportunity to test several of our activities, as well as our software framework and hardware platform. The class was held on Friday evenings for four hours, and so most of the students had day-time work and were interested in professional advancement. Students were only able to work with the robots during this limited class time.

Although we attempted to run a number of assignments in the course, technical problems prevented the robots from working reliably. After most of the semester had run its course, we discovered the source of the problem - the on-campus wireless network was occasionally and at random intervals throttling the wireless signals from our robots. This was an extremely difficult problem to diagnose, as the problem would come and go without warning, and so the robots would work for some time and then stop working. After a few weeks of these difficulties, our partner at Ohlone decided to de-emphasize the robotics portion of the class so as to save the educational value of the course.

A number of valuable lessons were learned through the experience of the Ohlone pilot. Firstly, our technology was vulnerable to issues beyond our immediate control; the Ohlone college wireless network was not configurable by our partner, and so it was difficult for him to experiment to discover the cause of the problem. Secondly, physical distance caused support to be much more limited in comparison to the support provided to CCAC. It is now clear that during the initial pilot the amount of support required was greater than our ability to provide that support.

5.3 CCAC Fall 2007

Figure 5: Students in the CCAC robot lab

The CCAC pilot was held from late August through early December 2007. Seventy-two students in four sections began the class; twenty-five students completed the course with a grade, and twenty-three of those passed. Three of the sections occured during the day, and consisted of bi-weekly lectures of two hours each. The fourth section was held in the evening for four hours once per week. The evening course attracted a different audience than the daytime sections - these students were more likely to be working and using the course for professional development, while students in the daytime sections were more likely to use the class for fulfilling degree requirements.

In addition to lecture times, students were able to access the robots for testing and demonstrating assignments; figure 5 shows some of the students in the robotics space. These open labs were staffed by the researcher, and were open from noon to six on Wednesdays and Thursdays - these times were arranged with the students beforehand and represented the optimal amount of available time for all students. As many of the evening section students were unavailable to work during the day time, the first hour of their scheduled lecture time was assigned to testing with the robots.

Students were shown the robot during the first couple of class sessions, but because of the logistics involved in arranging a lab space and finding a time to hold the lab, the first assignment which used the robot platform was the third week's assignment. It was also felt by the partner educator at the time that it might help the students to do a couple of traditional assignments before starting on the robot to reinforce some of the details of compiling programs and syntax.

About a week before each assignment was handed out, the principal researcher and partner educator would meet to discuss the upcoming assignment. We would use a previously developed assignment as a starting point and decide whether it was appropriate given our experience with the students to that point. In this way, we responded to our conceptions of students' abilities and aligned our upcoming assignments with those conceptions. Table 5 presents brief descriptions of the assignments that students were given; most of these assignments were entirely or heavily based on the previously developed curriculum; the assignments that were newly developed are designated with bold text.

Subject	**Description**
1. First program	Print a short statement to screen. This assignment did not use a robot.
2. Formatting output	Print a formatted block of text. This assignment did not use a robot.
3. Talking, dancing robots	Make the robot talk and make at minimum three distinct motions.
4. Conditional control structure	Have the robot sense bumper hits and condition based on those hits.
5. Looping control structure	Make the robot draw a polygon of 3-10 equal sides; the user inputs the number of sides.
6. Methods	The robot draws a shape, but each shape is a different method in the program.
7. Classes - Robot Tango	The program instantiates two robot objects representing different robots, and has to choreograph a dance between the two.
8. Classes - Story Teller	Students are provided with a main program which calls a class that they need to write. The class has to provide story snippets that the main program assembles into stories the robot tells/acts out.
9. Arrays - Record a Dance	Students create a program that uses bumper hits to store a sequence of directions. Playback of the sequence is also programmed by the students.
10. Applets	Students create a simple applet with graphics that appropriately resize as the applet window is stretched. This assignment did not use the robot.
11. GUIs	Students develop a simple interface to drive the robot around remotely.

Table 5: Listing of Assignments used in CCAC pilot

5.3.1 Evaluation methods

The CCAC pilot was formally evaluated along a number of metrics to inform our second-stage design; we were especially interested in student motivation and retention, but also assessed learning. We tracked retention of students after every assignment and compared it to the retention of students in courses offered by the same educator from Fall 2003 to Fall 2006. The retention data is very rich - it shows exactly which week in the class individual students stopped completing assignments, and so we can compare the piloted course to older courses not just on a gross scale, but on a fine-grained time-scale. Additionally, we tracked student interest in our assignments with short surveys that were completed after each assignment, and we also compared interest in CS as measured by pre and post surveys. To assess learning, we compared performance on traditional CS exams of the students in the pilot course to performance by the students in four previous fall semesters; these exams are changed superficially year over year to prevent cheating, but cover the same content. As the conceptual progression of the curriculum is the same as in the previous non-pilot courses, the exams are well-aligned with what students have learned.

5.3.2 Evaluation Results

We present the results from the evaluation and analysis of the pre/post surveys, post-assignment surveys, and comparisons between pilot and prior year retention rates and grade performance.

Pre-Post survey results We asked students to fill out surveys at the beginning and end of the course to gauge student prior experience, their reasons for taking the class, and their interest in and confidence with computers and programming.

To gauge prior experience, we asked students to state which programming language they knew best, and if they had one, to rate their experience level with that language on a 1-5 scale (from novice to expert). Table 6 presents the results for pre and post surveys. Although 40 students took the pre survey and 20 the post survey, only 10 students took both; therefore we present comparisons between both the larger set, which includes data in the pre survey from students who dropped the class, as well as comparisons between the ten students who took both surveys. Not surprisingly, many students became more familiar with Java and no students answered 'none' for the post survey. Experience ratings did not increase because many students with no experience (and thus no rating) became beginners after taking the class.

Language	**All Pre** (n=41)	**All Post** (n=20)	**Matched Pre** (n=10)	**Matched Post** (n=10)
Java	17.07%	55.00%	20.00%	70.00%
C/C++	14.63%	15.00%	20.00%	20.00%
Other	17.07%	25.00%	20.00%	10.00%
None	51.22%	0.00%	40.00%	0.00%
Experience Rating	2.35	2.58	2.17	2.1

Table 6: Percentage of students who reported each category as their best-known programming language

We asked students to choose from a list of reasons why they were taking the CS1 class; students could select multiple reasons. The reasons were: Required for degree, plan to transfer credit to a 4 year school, want to program as a career, professional advancement, interested in programming and other. Table 7 presents the reasons students gave for taking the class; there was no significant difference between pre and post surveys, but this data provides some insight into the diversity of needs the incoming students of a community college CS1 class have. It is interesting that the reasons changed somewhat among the matched set - this may represent a shifting focus over the course of the semester to more degree based and academic goals.

We gave students a number of statements intended to measure their interest in and confidence with computers and programming. Students were asked to rate the statements on a scale from 1 to 5 with 1 being strongly disagree, and 5 being strongly agree. Table 8 presents the results of these surveys. In both the matched and unmatched sets some of the measures of confidence and interest increased, although not by statistically significant amounts.

Reason	**All Pre** (n=41)	**All Post** (n=20)	**Matched Pre** (n=10)	**Matched Post** (n=10)
Degree	41.46%	40.00%	30.00%	50.00%
Transfer	34.15%	40.00%	20.00%	50.00%
Career	31.71%	25.00%	50.00%	20.00%
Advancement	21.95%	40.00%	30.00%	30.00%
Interest	48.78%	50.00%	50.00%	50.00%
Other	12.20%	25.00%	10.00%	20.00%

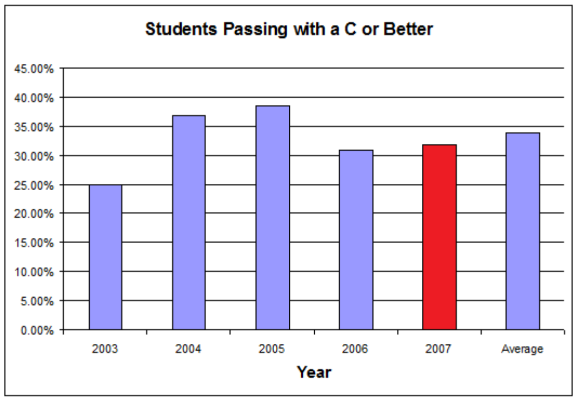
Table 7: Student reasons for taking course

Statement	**All Pre** (n=41)	**All Post** (n=20)	**Matched Pre** (n=10)	**Matched Post** (n=10)
I am familiar with computers.	4.67	4.47	4.90	4.80
I am familiar with computer programming.	3.00	3.73	3.44	3.50
I am confident I could program a computer if I needed to.	2.95	3.21	2.60	3.10
Programming is easy for me.	3.08	3.37	3.22	3.10
I like computer programming.	3.68	4.21	3.80	4.10
I would like to study computer programming as my major.	3.20	3.72	3.33	3.80
I would like to program computers as a career.	3.37	3.74	3.56	3.80

Table 8: Student ratings of interest in and confidence with computing

Retention Rates We compared the retention rates of the fall 2007 course to courses taught by our partner in fall semester 2003-2006. We compared the overall retention rate, and found no significant difference between our pilot and prior years (figure 6). We also examined the retention rate at the week-by-week level, by determining when a student last completed an assignment for non-zero credit. Note that this is different than when a student officially drops a course, but in some ways is more accurate, as it represents when the student stopped trying in the class. At the more detailed level, we also saw no real differences in the pattern of drops between the pilot and prior years. However, two important patterns are visible in the detailed data: First, on average about one-third of all students drop out before assignment 3, that is, before any robotics assignments had been handed out in the pilot. Second, there is usually a small spike in drops in the middle of the course (between assignments 4 and 7) - this spike is caused by students dropping after receiving poor results on their first exam.

Interest We measured student interest in the robot assignments provided to them during the pilot both by comparing their assignment completion rates to prior years and by directly asking them in post-assignment surveys. Student reports of the assignments via these surveys was generally favorable. Figure 7a shows the percentage of students who, after completing an assignment, rated that assignment as their favorite assignment to that point in the class. One would expect that the trend line in this figure would drop over time, as students have more assignments to choose from. Instead we see a flat to marginally increasing trend line, indicating that students who did not drop the course stayed engaged throughout. Although this response can be explained partially by recency effects, in the post survey a number of students wrote



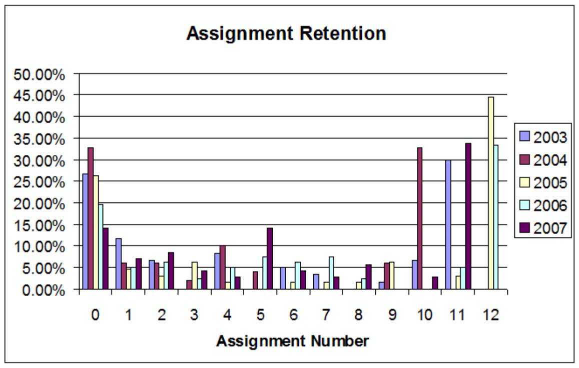
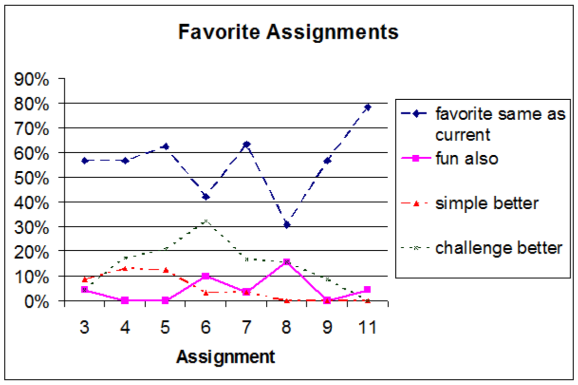


Figure 6: Overall (left) and week-by-week (right) retention rates in the Fall semester CS1 course at CCAC

that assignments grew increasingly interesting, and that each new assignment surpassed the last in interest. As students were able to give an open-ended answer to our post-assignment survey questions, they would



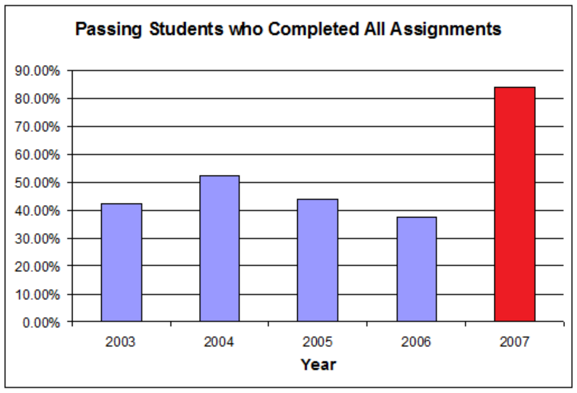


Figure 7: (left) Percent of students who listed the current assignment as their favorite to date (right) Percent of passing students completing all course assignments

occasionally provide us with a reason behind their answer. Three responses were frequently expressed and are also shown in figure 7a. The first response was termed as 'fun also', and codes for responses in which the student stated that while the current assignment was not their favorite, it was still enjoyable. This response was expressed often for assignments six and eight, which garnered lower overall 'favorite' ratings. The second response was 'simple better', which codes responses where the student said they liked an earlier assignment better because it was simpler or easier. These responses dropped off towards the middle of the semester, likely because students who expressed this response dropped the class. The third response was 'challenge better', where students said that they liked this assignment best because it was more complex, had more value for learning programming, was a better challenge, etc. These responses peaked in the middle of the semester, likely because it is during the middle of the semester that students have to start understanding deeper, more challenging programming concepts.

We also compared student interest in the assignments to prior years by comparing the percentage of passing students who completed all of the assignments in the class. The grading structure both in the pilot and in prior years de-emphasized assignments, with each assignment making up only 2 to 2.5 % of the total grade; as such, it is very easy to pass the class while missing an assignment, and so to some extent students complete the assignments because they recognize their value as aids to learning. As figure 7b shows, the percentage of students completing all assignments in our pilot was significantly higher than in prior years (p=0.003 for 2003, p=0.044 for 2004, p=0.002 for 2005, p=0.000 for 2006).

Confidence We measured student confidence in their performance on the robotics assignments by asking them what grade they thought they would receive on the assignment they had just completed. Figure 8 shows the results on an assignment by assignment basis. Generally student confidence improved slightly over time; this may be due to stronger students staying in the course while weaker ones drop out. There is a strong dip in confidence for assignment six; assignment six was widely seen as a difficult assignment and marks the introduction of the programming concept of non-main methods.

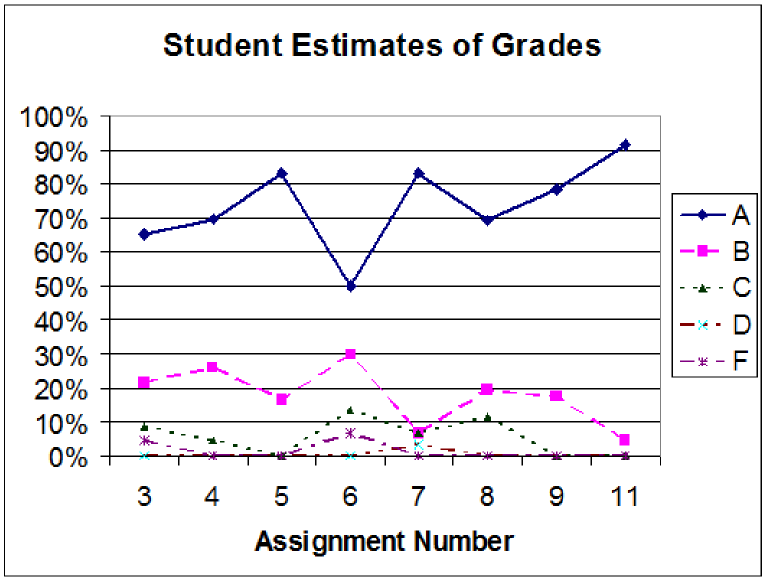


Figure 8: Student estimates of their grade on each assignment

Frustration We attempted to deduce potential 'sticking points' that would make the pilot assignments less engaging or more difficult by asking students to tell us the most difficult part of the just-completed assignment. We coded these responses into six categories; the percentage of expression of each is shown in figure 9. The codes were:

- **conceptual** - Code for responses that gave a programming or CS concept as the answer.
- **compile+syntax+IDE** - Code for responses where students had trouble with syntax, using the IDE, or figuring out compiler error messages.
- **aesthetic** - Code for responses where the students responded that it was difficult thinking of non-programming creative elements; how the robot should dance, what it should say, etc.
- **lab times+testing** - Code for responses dealing with the logistics of the lab setup - generally that it was difficult getting to the lab during the open hours, and that there weren't that many hours for testing.
- **framework** - Code for responses dealing with bugs or usability issues in the robot software framework created for the pilot.
- **nothing difficult** - Code for responses stating that nothing was difficult about the assignment.

Viewing figure 9, it becomes apparent that conceptual issues were often the most difficult part of an assignment. From a pedagogical point of view, we feel this is the desired result; students should spend the majority of their time in a computing class struggling with computing concepts. We were also sensitive to any negative impacts of the robot on the student's ability to complete their assignments; both the lab times+testing and framework codes represent difficulties that would not occur in a non-robot CS1 class. Fortunately, these responses were expressed fairly rarely, and combined make up less than 20% of reported difficulties. Also encouraging is the lack of responses indicating a problem with robot hardware; reflecting the fairly robust and failure-free operation of our robot platform.

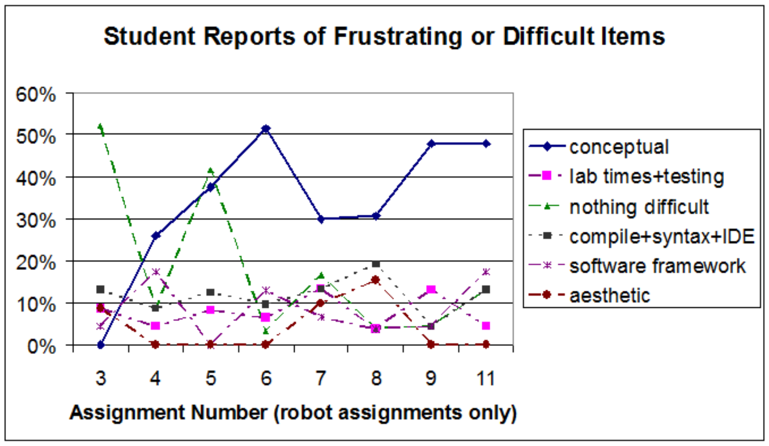


Figure 9: Student sources of frustration

Relevance We asked students if they saw any relationships or links between the program written for the assignment and the operation of software, computers, or computational devices. We were aiming to create assignments that would be relevant to students' lives so as to be engaging and motivating. Figure 10 shows the percentage of students who linked the assignment content to an item in the real world. Students increasingly saw relationships between their assignments and the real world; this may be due to the increasing complexity of later assignments.

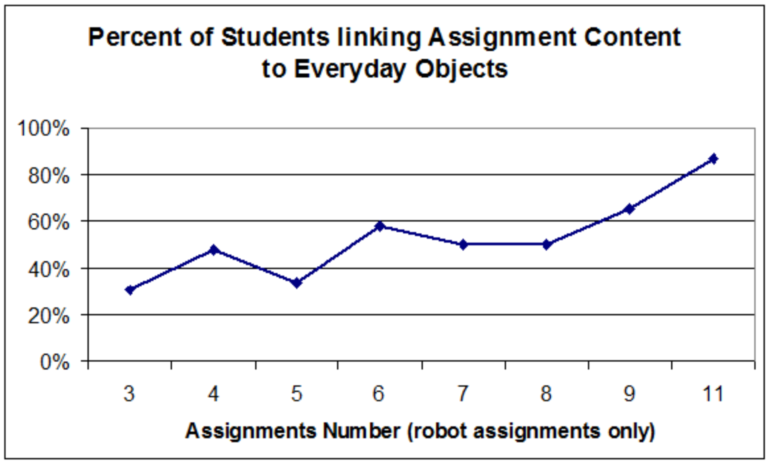


Figure 10: Percent of students who linked assignment content to the external world

Grades We compared the grades of students in our pilot to prior years to ensure that our students' learning was not hindered by the robotics assignments. The grade structure of the class was such that 75% of the grade consisted of exam grades. Exams were modified only superficially (to prevent cheating) between our year and prior years; as such, we consider the performance on these exams and subsequent performance in the class as an adequate measure of comparative student learning. Figure 11 details the average grades in the 2007 pilot and prior years. Students in the pilot performed significantly better (p¡0.01) than the five year average, and significantly better than three of the four previous years (p¡0.05 in all three cases).

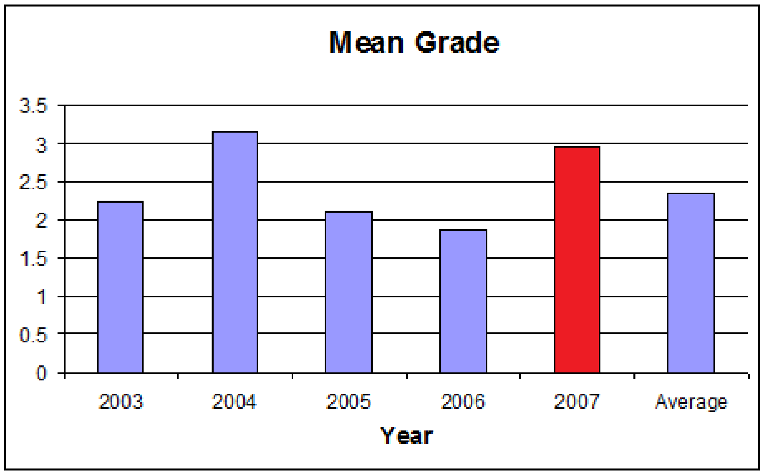


Figure 11: Mean grades of passing students, 2003-2007

6 Next Steps

Our results to date and experiences with the existing pilots have validated our approach of using robots in the CS classroom and pointed to a large number of improvements we could make to our design and evaluation. We are now in the process of redesigning our robot platform, software, and curriculum and are also starting to plan the evaluation of further pilots. We are planning to pilot our redesign, and to use the results of these future pilots to continue iterating on our design. Our aim is to scale up the number of pilots each time we iterate on the design; thereby creating a design with the optimal feature set for more and different contexts and over time and iterations approaching a readily disseminable curriculum and technology.

6.1 Planned pilots

We are planning a number of pilots to take place in the Spring of 2009. We are again working with our partner at CCAC to use robots in his CS1 sections. We have identified a new partner at Stanford who is interested in testing out two or three of our curricular activities during a regular CS1 course; essentially testing the ability of our design to be 'dropped in' to a regular curriculum. Despite the problems of the first pilot, we plan to work with Ohlone college again to offer another course. Through a related program we have partnered with high school CS1 teachers and are jointly developing curricula and activities with them appropriate to the high school level. We plan to use these contacts to test new activities and the new robot platform at the high school level. Finally, we are beginning to search for additional partners, with a major objective being to offer a CS1 robotics course at Carnegie Mellon.

6.2 Changes to design

Of the three components of our design (the curriculum, the robot, and the software framework), we foresee the greatest changes occuring to the design of the robot. From our survey of educators as well as prior attempts to use robots in CS1 [2], we suspected that a major problem with our initial design was that it was not possible to provide each student with a robot to test with. Our experience running the CCAC pilot has solidified this notion, and persuaded us that it would be virtually impossible to introduce robotics in the CS1 course without ensuring that students have unconditional access to those robots. This implies that the robot must be owned or controlled by the student, so that she can take it home and test with it at any time. Making the robot student owned sets up two core design goals for our platform; one, it must be very low cost, in the same range as a textbook ($50-$100), and two, it must be very robust, both in terms of surviving being carried in a backpack, and in terms of working with minimal setup on a home computer. At the same time, we need to ensure that the platform's feature richness is such that we can still use the popular prior curricular activities.

Robot Platform We are currently working on the prototype design for our new robot platform, and have made several design decisions aimed at reducing cost. Our greatest conceptual breakthrough when designing the robot was realizing that when used in the computer science class, the robot is a way of adding interaction modalities to computer programs; instead of just screen or file input and output, robots take real sensor inputs, and can emote through sound, movement, and light. None of these modalities is diminished if the robot is physically tethered to the computer, and so we are running the robot entirely with power provided through the USB port. Although this requires that the robot be tethered to the computer at all times, it simplifies the system in a number of ways: There is never a delay from charging a battery, robot-computer communication occurs over a very robust connection instead of over a prone-to-fail wireless network. We have also identified crucial capabilities that the robot must have in order to meet the needs of our curriculum, specifically:

- Sensing obstacles
- Playing sounds and speaking (through the computer speakers)

- Moving a set distance or at a set velocity
- Emoting through multi-color LEDs

We are also including a number of no-cost and very low-cost features because they expand the capabilities of our robot:

- Access to internet data through RSS feeds
- Vision through use a USB webcam
- Light-level and temperature sensing

We expect that by taking advantage of a computer's built-in capabilities (sound, vision, and power), we can drastically reduce the cost of our robot platform and bring it in-line with the price range required for a successful personal robot.

Software Framework Although we will need to redesign the lower levels of our software package to reflect the change in communication with the robot from wireless to USB, we will make few changes to the API level that students deal with directly. We felt that our software framework was easily picked up and used by students, and while we noted several minor improvements thanks to student feedback, we will keep the basic functionality the same.

Curriculum Changes to the curricular activities and assignments will be made in conjunction with our current and future partners and will be based on the classroom context. We understand that an assignment that works at CCAC may not work at Stanford and vice versa, and so as we embark on the next phase of pilots, we will build up a large and diverse set of curricular activities.

6.3 Changes to evaluation plan

Our aim for the next set of evaluations is to provide further evidence of the success of our approach, while continuing to find areas of improvement for further iterations. We will likely use similar evaluation methods: Pre/post and in-class surveys coupled with observation where the primary researcher is near enough to observe the classroom. In addition to this evaluation, we are also planning a concurrent control group at at least one of our partner sites. This control group would receive the same curriculum as the pilot but with non-robot activities. The control and pilot will be taught by the same educator, and students would be randomly assigned between this group and the pilot group. Lastly, we plan to track individual students with greater fidelity in the next round of pilots; our inability to track students during this evaluation cycle meant that it was very difficult for us to characterize actions such as dropping out of the class on an individual scale.

7 Conclusion

We have presented the first completed stage of an iterative process to design a robot platform, software environment, and curriculum for introductory computer science education. By partnering with educators in the field and conducting significant initial evaluations of the classroom realities of CS1 education, we formed a foundation for our initial robot and curriculum design work. Furthermore, these initial evaluations enabled us to develop a long term strategy for creating a technology that would be aligned with the needs of CS1 students and learning goals of the CS1 class. Our recent pilots have provided us with a number of lessons, both organizational and research related, that will be used during the next design-pilot-evaluate stage.

A Appendix - Popular Robot Software Control Methods

void saySomething(String text)
Uses the Text To Speech library to synthesize audio from the text variable, and play that audio on the Qwerk using the playSound command. Note that this program blocks for the amount of time that it takes to synthesize the audio - for example, synthesizing the sentence ”Hello world” may block further program execution for two seconds.

void writeToTextBox(String message)
Writes the message string to the GUI's textbox. The message is always preceded by a timestamp.

int getTextFieldValueAsInt()
Returns the contents of the GUIs editable text field as an int. Returns 0 if the text field is empty or the value cannot be converted to an integer.

String getTextFieldValueAsString()
Returns the contents of the GUIs editable text field as a String.

boolean sleepUnlessStop(int ms)
Sleeps the program for a given number of milliseconds. If the GUIs Stop button is pressed, this method immediately exits without sleeping.

void moveMotors(int leftMotor, int rightMotor)
Sets the velocities of the Create's left and right motors in millimeters per second (25.4 millimeters = 1 inch). The maximum velocity is 500 mm/s. To drive a wheel backwards, send a negative velocity.

void moveMotors(int leftMotor, int rightMotor, int runningTime)
Exactly like moveMotors, but with a third argument, runningTime, which sets how many milli-seconds the motors should run at the velocities given. After runningTime has elapsed, the motor velocities are both set to 0. Note that this method blocks any further program operation until runningTime has elapsed.

void stopMoving()
Stops the Create's motors by sending 0 mm/s velocities.

void moveDistance(int travelDistance)
Causes the Create to move in a straight line by the number of millimeters specified by travelDistance. Note that your robot must travel by the number of millimeters specified in order for further commands to be accepted if it gets stuck against a wall you will need to manually rescue it.

void moveAngle(int travelAngle)
Cause the Create to rotate in place by the number of degrees specified by travelAngle.

void turnLeft()
Cause the Create to rotate 90 degrees left.

void turnRight()
Cause the Create to rotate 90 degrees right.

boolean bumpLeft()
Returns true if the left bumper is bumped.

boolean bumpRight()
Returns true if the right bumper is bumped.

boolean bumpCenter()
Returns true if both left and right bumpers are bumped.

boolean playButtonPressed()
True if the play button on the Create is pressed, false otherwise

boolean advanceButtonPressed()
True if the advance button on the Create is pressed, false otherwise

void setAllLEDs(boolean play, boolean advance, int color, int intensity)
Set the states of the play, advanced, and power LEDs on the Create with a single method.
play - State to set the LED located next to the play button ('¿') to - true is on, false is off
advance - State to set the LED located next to the advance button ('¿¿—') to - true is on, false is off
color - Set the color of the power LED - 0 = green, 255 = red, and values between mix green and red
intensity - Set the intensity of the power LED, valid values range from 0 (off) to 255 (completely on)

References

[1] J. Vegso, “Interest in cs as a major drops among incoming freshmen,” *Computing Research News*, May 2005.

[2] M. L. Fagin, B., “Measuring the effectiveness of robots in teaching computer science,” in *SIGCSE*, 2003, pp. 307–311.

[3] “For inspiration and recognition of science and technology (first).” [Online]. Available: http://www.usfirst.org/

[4] M. Yim, M. D. Chow, and W. L. Dunbar, “Eat, sleep, robotics,” pp. 245–292, 2000.

[5] “Botball.” [Online]. Available: http://www.botball.org/

[6] D. P. Miller and C. Stein, “so that’s what pi is for! and other educational epiphanies from hands-on robotics,” pp. 219–243, 2000.

[7] “Innovation first inc.” [Online]. Available: http://www.innovationfirst.com/

[8] R. M. B. R. Rusk, N. and M. Pezalla-Granlund, “New pathways into robotics: Strategies for broadening participation,” 2007. [Online]. Available: http://llk.media.mit.edu/papers/NewPathwaystoRobotics-06-07.pdf

[9] H. J. Kim, D. Coluntino, F. G. Martin, L. Silka, and H. A. Yanco, “Artbotics: community-based collaborative art and technology education,” in *SIGGRAPH ’07: ACM SIGGRAPH 2007 educators program.* New York, NY, USA: ACM, 2007, p. 6.

[10] M. McNally, “Walking the grid: Robotics in cs2,” in *Proceedings of the 8th Austalian conference on Computing education*, 2006, pp. 151–155.

[11] e. a. Shamlian S., “Fun with robots: A student-taught undergraduate robotics course,” in *ICRA*, 2006, pp. 369–374.

[12] I. Nourbakhsh, K. Crowley, K. Wilkinson, and E. Hamner, “The educational impact of the robotic autonomy mobile robotics course,” Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-29, August 2003.

[13] J. M. Douglas Blank, Deepak Kumar and L. Meeden, “Advanced robotics projects for undergraduate students,” in *Symposium on Robots and Robot Venues: Resources for AI Education*, 2007.

[14] J. M. Grant Wiggins, *Understanding by Design, Expanded 2nd Edition.* Prentice Hall, 2005.

[15] K. Sierra and B. Bates, *Head First Java. 2nd edition.* OReilly, 2005.

[16] H. Deitel and P. Deitel, *Java How to Program. 6th edition.* Pearson Education, Inc., 2005.

[17] C. Horstmann, *Big Java, 2nd edition.* John Wiley & Sons, Inc., 2006.

[18] D. J. Barnes and M. Kolling, *Objects First with Java. 2nd edition.* Pearson Education, 2005.

[19] C. Horstmann, *Java Concepts. 4th edition.* John Wiley & Sons, Inc., 2005.

[20] D. Malik, *C++ Programming: From Problem Analysis to Program Design. 2nd edition.* Course Technology, Thomson Learning, 2004.

[21] J. Lewis and W. Loftus, *Java Software Solutions: Foundations of Program Design. 4th edition.* Addison Wesley, Pearson Education, Inc., 2005.

[22] B. A. Forouzan and R. F. Gilberg, *Computer Science: A Structured Programming Approach Using C. 2nd edition.* Brooks/Cole, 2001.

[23] E. Roberts, *The Art and Science of C: A Library-Based Introduction to Computer Science.* Addison Wesley, 1995.

[24] W. Savitch, *Absolute Java. 2nd edition.* Addison Wesley, 2006.

[25] T. Lauwers and I. Nourbakhsh, "Informing curricular design by surveying cs1 educators," in *Proceedings of the 4th International Symposium on Autonomous Minirobots for Research and Edutainment*, 2007.

[26] S. Zweben, "2003-2004 taulbee survey: Record ph.d. production on the horizon; undergraduate enrollments continue in decline," *Computing Research News*, May 2005.

[27] R. McCauley and B. Manaris, "Computer science education at the start of the 21st century - a survey of accredited programs," in *Proceedings of 32nd ASEE/IEEE Frontiers in Education Conference*, 2002.

[28] B. Brown, *A Guide to Programming in Java; 2nd edition.* Lawrenceville Press, 2007.