Implementation of a Smart Lab for Teachers of Novice Programmers

Ali Alammary, Angela Carbone and Judy Sheard

Faculty of Information Technology
Monash University, Melbourne, Australia

Abstract

Communication between students and their instructors in the lab is a limited commodity. With limited access to the tutor, students can sometimes spend a long time trying to fix simple errors, continually revisiting and repeating the same errors. Instructors, on the other hand, find themselves explaining the same mistakes over and over again. It is often not clear to them how well individual students are progressing toward meeting the task objectives. This paper introduces a new implementation of Smart Classroom technology for introductory programming computer laboratories. The Smart Lab is intended to make the computer lab a better educational environment for both students and instructors. In the Smart Lab instructors are provided with information about each student’s progress as they perform programming tasks, enabling the instructors to readily respond to individual student’s problems and assess the overall progress of the class. Two different evaluation approaches were used to test the new implementation: an expert review session and a lab study. The evaluation found that the Smart Lab improved instructors understanding of their students’ problems enabling them to provide timely and appropriate feedback. It also provided instructors with better understanding of their students’ programming strategies and compilation behaviours.

Keywords: Smart Classroom, introductory programming, learning technology, tutoring, feedback.

1 Introduction

Learning programming is a difficult task for most students. Novice programming students experience many problems which contribute to high dropout and failure rates in introductory programming courses (Lister et al., 2004; McCracken et al., 2001). According to Gomes and Mendes (2007), these problems start in the early stages of learning when students are trying to understand and apply basic programming concepts, such as loops and control statements. A common argument used to explain the difficulties students face is that programming is a multifaceted skill. Learning to program requires knowledge about programming languages, programming tools and problem solving (Robins, Rountree and Rountree, 2003; Carbone, Mitchell and Hurst, 2009).

One way to assist students overcome difficulties in learning to program is by providing them with programming tasks to solve in the computer lab with support from a tutor. A tutor can inspect the students’ code, investigate the problem and then provide the students with the appropriate help. However, communication between students and their instructors in the lab is a limited commodity. As the number of students in the lab increases, the amount of time that instructor can devote to each student decreases. With limited access to the tutor, students can sometimes spend a long time trying to fix simple errors, continually revisiting and repeating the same errors. They may have difficulty understanding the compiler messages, making code correction a frustrating experience. Instructors, on the other hand, find themselves repeatedly explaining the same mistake. It is often not clear to them how well individual students are progressing toward meeting the task objectives. Furthermore, they are not easily able to assess the errors that students are most commonly encountering.

A number of tools have been developed to assist instructors in giving feedback to students about their work. All tools found in a search of the literature were designed to provide feedback about students’ assignments after they had completed and submitted their work, and not for their work on programming activities in lab classes. In view of this, we have developed a Smart Lab that is intended to improve the assistance instructors provide to their students in their lab classes. It allows tutors to better understand their students’ problems, programming strategies and compilation behaviours. As a result, students can be provided with appropriate help when they need it and common mistakes can be realised and explained.

This paper investigates a Smart Classroom technology for introductory programming computer laboratories, implemented as a Smart Lab. Section 2 details the Smart Lab development, with features of a new Smart Lab described in section 3. In section 4 details of the evaluation of Smart Lab are provided, followed by a discussion, conclusion and suggestions for future work.

2 Related work

As background to the development of the Smart Lab we have investigated two areas: i) implementations of Smart Lab technology ii) approaches to helping address the difficulties faced by students when working on a programming task.
2.1 Smart Lab technology

A Smart Lab can be defined as a lab equipped with tools designed to enhance instruction and learning (Di, Gang and Juhong, 2008). The idea behind the Smart Lab is to employ technology to provide students and teachers with tools to extend their ability to communicate effectively and to enable them to engage successfully with the curriculum. A Smart Lab can be implemented in different ways to achieve different goals. As an example, Tissenbaum and Slotta (2009) describe a lab which was developed in several layers using a variety of approaches and devices. The main goal of that implementation was to enable students to visualize problems and their solutions on a large display screen in the lab. This visualization was intended to help students connect and understand the relationships between these problems and their solutions.

The most common implementation of smart lab technology is for distance learning and is concerned with making distance education an exciting experience and as effective as face-to-face instruction. A notable example of this implementation is the work of Di et al. (2008) who presented what they called the blending-reality classroom. This system was designed to provide the instructors with intellectualized human-computer interaction technology for their teaching purposes, and also to provide the distance education students with an appropriate study environment.

In general, the different implementations of smart lab technology found in the literature were complex and contained many different components. They also required expensive hardware such as video cameras, servers and display screens. Of the smart lab technology reviewed, none were found to deal specifically with a computer programming lab, rather they were designed to work in any kind of lab or classroom.

2.2 Approaches to address the difficulties faced by programming students

Many studies have been conducted to address the difficulties faced by programming students when working on programming tasks (Carbone, 2007; Sheard et al., 2009). These studies can be classified into three areas: 1) identification of the difficulties encountered by novice programmers, 2) provision of automated help to students via their development platforms, and 3) provision of tools that help instructors monitor their students’ progress. The motivation for much of this work is to reduce the workload of instructors and to enhance students’ learning experiences. The following provides examples of studies from these three areas.

Many studies have investigated the difficulties encountered by novice programmers. For example, Jadud (2006) studied the ways in which novice programmers write their programs. He examined which errors were most often generated by novices and the time they would spend to fix these. Another work by Flowers, Carver and Jackson (2004) captured and explained syntax and semantic errors made by programming students. They determined the fifty most common programming errors and found that even the ‘stronger’ students continued to make these fifty errors late into the semester. The main reason for this, according to Flowers et al., is that the Java compiler error messages do not help students to understand and fix their errors. These studies have provided valuable information about the obstacles students face in writing programs and the approaches they take to overcome these.

Another body of work has focused on development of tools to provide automated help to students. An example is the web-based application WebToTeach developed by Arnow and Barshay (2002) that enables automated checking of students’ assignments. WebToTeach provides students with a list of programming tasks written by their tutor. Students write their answers to each exercise in a dedicated form and then wait for the system to provide feedback. If the answer passes a certain number of system checks, the student will be informed by the system that no extra work is needed. Otherwise, the answer will be rejected and the student will be provided with an explanation and hints to correct their submission. Another example is the tool Expresso, which is targeted at students studying introductory Java programming. Expresso provides students with easy to read error messages (Hristova et al., 2003). Expresso does not eliminate the need for compiler messages; rather it enhances the compiler functionalities by generating detailed and easy to read error messages and providing suggestions on how to fix the errors.

A further body of work has focused on development of tools to enable instructors to monitor their students’ progress. Spacco et al. (2006) developed a tool called Marmoset that provides instructors with detailed feedback on the development process of their students’ programming assignments. Snapshots of students’ code are captured and stored on a central repository each time students save their files. This process enables generation of development histories for each student. These histories offer a detailed perspective of students’ progress while doing a programming task. Another tool Retina (Murphy et al., 2009) does a similar job as Marmoset but focuses more on the compilation and run time errors. Retina collects data about students’ programming activities and stores these in a central database. It then processes this data to provide instructors with detailed information about students’ progress. Karam, Awad and Carbone (2010) developed a tool to analyse students’ code and extract what they termed actions. Each single statement or declaration in the program is considered as an action. After extracting the actions from the student code, the tool extracts actions from the solution code provided by the instructor. It finally compares the actions from the student’s code to those from the solution code, to generate a list with the missing and completed actions.

The tools that were found to assist instructors have been developed to investigate student code away from a class situation, for example, their assignment work. However, the number of assignments that students do in a semester is relatively small compared to the number of tasks they attempt in the programming lab sessions. Despite this, no tools were found that could be used in a teaching situation in a computer lab. Furthermore, except for the last mentioned tool, all the other tools rely on the code’s output and the compilation attempts to evaluate the students’ code. Such evaluation is insufficient to provide comprehensive insight into students’
programming behaviors and the different types of difficulties they experience.

Our work fills this gap by developing a Smart Lab implementation which incorporates and builds upon the code analysis tool of Karam, Awad and Carbone (2010). The Smart Lab is essentially a tool for use in a computer lab which provides instructors with detailed information about each student’s progress in writing computer programs.

3 Our Smart Lab solution

Our Smart Lab is a computer lab containing a set of networked workstations equipped with specially designed software. Code on student machines is captured, analysed and a summary displayed on the instructor’s machine. The instructor is thus able to monitor progress of each student in the class from one central point.

The Smart Lab was designed to be simple and easy to use. It does not require any extra hardware and can be easily installed on any workstation in a lab. It needs to be configured one time only in the lab. All configuration setting is stored in a single text file called BlueJSetting. To configure the workstation, this file can be easily copied to the C drive on the workstation. The architecture of the Smart Lab is shown in Figure 1. It analyses the compilation attempts and the code output, and then further analyses the code to provide deeper insights into the students’ problems and programming behaviour. There are two main components to the Smart Lab: the Code Transferer component, used to transmit real-time information about the students’ code and their programming activities to a central database, and the Analyser/Visualiser software component, that interacts with the central database to provide information to the instructor on the students’ progress and the problems they experience. Each of these is explained in detail in the sections that follow.

3.1 Code Transferer

The aim of the Code Transferer is to send real-time snapshots of students’ code and their programming activities to the central database. Currently, the Code Transferer is associated with the BlueJ IDE, but it can be easily modified to work with other IDEs. When the Code Transferer starts, it registers three types of listeners: Package Listener, Compile Listener and Class Listener.

Package listener does two main tasks: it captures a snapshot of the students’ code and detects the current class that students are working on at the moment and stores this information in the database. Compile Listener in turn captures and stores detailed information about both successful and unsuccessful compiling attempts. The Class Listener is used to detect the changes to the state of each of the classes that students are implementing. The state of the class changes when the class has been opened in the editor, compiled, uncompiled or renamed.
3.2 Analyzer/Visualiser

The Analyzer/Visualiser allows tutors to monitor their students’ progress in real-time when they are working on programming tasks in the lab. It also allows other teaching staff to access students’ captured data at a later stage to view the different strategies that students used to implement solutions, as well as the common difficulties they encountered. These features can help tutors to understand students’ needs and provide each of them with more appropriate and focused assistance. Lecturers can also utilise this feature to identify topics and concepts that students are experiencing problems with and tailor their instruction to those areas.

The Analyzer/Visualiser offers seven different features. These include:

1. Graphical representation of the lab setting
2. Common Missing Action box
3. Common Syntax Errors box
4. Highlighting which students require the most and the least help
5. Highlighting students who remain idle for extended periods
6. Detailed information about a particular student’s activities
7. Students’ final reports

Each of these features is explained in detail in the remainder of this section.

3.2.1 Graphical representation of the lab

Each student in the lab is represented by a label in the Analyzer/Visualiser’s main window (Figure 2). To enable easy mapping of each label to each student, the position of each label on the screen is related to the actual location of the student in the lab. Each label contains a summary of the student’s activities. This includes the current exercise that the student is working on, the number of successfully implemented actions, the number of actions not yet implemented, the number of compilation attempts of the current exercise and the result of the last compilation attempt.

The missing and implemented actions are identified by using the tool developed by Karam, Awad and Carbone (2010) which was described in the related work section. The graphical representation is designed to help the tutor readily observe the amount of progress that each student
3.2.1 has achieved in solving a programming exercise, understand the strategy that each student used to solve the programming exercise, and identify the exercise that took students the longest time to implement.

3.2.2 Common Missing Actions box
This box appears at the bottom left-hand corner of the Analyser/Visualiser main window and it shows the four most common missing actions that the students failed to implement (see Figure 2). This box is meant to help the tutor identify the actions that students find difficult to implement and to identify programming concepts that require additional clarification.

3.2.3 Common Syntactic Errors box
This box appears at the bottom right-hand corner of the Analyser/Visualiser main window and it highlights the four most common syntactic errors that the students made (see Figure 2). This box is meant to aid the identification of common syntactic errors that students make, thereby enabling the tutor to focus more on these errors and to advise the students how to avoid them.

3.2.4 Highlighting students who require the most and the least help
The Analyser/Visualiser changes the border colour of the labels that represent the students to red for the students that need most help and to green for the students who need the least help. This highlighting is meant to help tutors focus teaching efforts on the students in the lab according to the level of help they may need. Figure 2 indicates that students 2, 6 and 8 need the most help while students 5 and 9 need the least help. Note that the main window shows the missing actions for the current exercise only, while the application looks to the whole missing actions across all exercises to determine which students need the most or least help.

3.2.5 Highlighting students who remain idle for extended periods
The Analyser/Visualiser software changes the background colour of the labels to orange if a student has remained idle for more than two minutes. Highlighting the idle students is meant to provide tutors with more insight into students’ programming behaviours and they...
may then investigate the source of their inactivity. Figure 2 shows an example of this functionality; the system has changed the background colour of student 8 who remained idle for a period of time.

3.2.6 Detailed information about a particular student’s activities
Clicking on a particular student’s label in the main window opens a new window with detailed information about that student’s programming activities (see Figure 3). This window contains: the name of the current exercise the student is implementing, the last captured snapshot of code, a list of completed and missing actions and compilation attempts for that class. The window also allows the instructor to navigate forwards and backwards through the code snapshots by clicking the navigation buttons at the bottom of the window. It also has a drop-down list to allow tutors to navigate to the other exercises that the student has already attempted.

The information in this window sheds additional light on the student’s progress. It pinpoints the nature of the assistance that the student requires, tracks the amount of progress that a student is making in the task and aids understanding of the student’s programming strategy.

3.2.7 Student’s final report
The final report (see Figure 4) shows the amount of time the student spent on each exercise, the final analysis of the student code and detailed information on the compilation attempts. It provides an indication of the amount of effort that student has put into solving each programming task, the programming concepts that the student had problems with and his/her compilation behaviour.

4 Evaluation approach
The evaluation of the Smart Lab solution combined two evaluation techniques: an expert review session and a lab study. This approach of using two evaluation techniques is recommended by Krug, Burghardt and Edwards (2002) to provide a better assessment of computer systems rather than just using one technique.

4.1 Expert review
Four lecturers who teach programming units in the Faculty of Information Technology at our University were invited to evaluate the Smart Lab system in an
expert review session. Each lecturer had had many years of experience in teaching introductory programming units both in lecturing and tutoring in lab situations. During the session, each participant had the opportunity to take on the role as a tutor and use the Analyser/Visualiser software to monitor the programming activities of other participants who act as students working on programming exercises. At the end of the session each participant was interviewed on their experiences of using the Smart Lab system.

The evaluation questions were divided into three different sections, with each section focusing on a different part of the Smart Lab interface.

1. **Analyser/Visualiser main window.** Questions in this section were designed to investigate: 1) the usefulness of the information in the main window and whether it provided sufficient understanding of the programming activities of the other participants, 2) the ease of the mapping between the labels representing the other participants and their actual location in the lab, and 3) the helpfulness of highlighting the students who required the most and the least help.

2. **Detailed window** (see Figure 3). Questions in this section investigated the participants’ opinions about whether the information in the detailed window provided an understanding of the programming strategies and compilation behaviour of the other participants and whether it was helpful in spotting the different kinds of problems encountered by the participants performing the programming tasks.

3. **Students’ final reports** (see Figure 4). Questions in this section were designed to investigate whether the report information was helpful in identifying the amount of effort each participant put into solving the programming tasks and whether the report information provided an understanding of the programming strategies and compilation behaviour of the other participants.

4.2 **Lab Study**

Novice programming students in an introductory programming unit taught within a Masters program in the Faculty of Information Technology at our University were invited to participate in this study. The study was conducted in two laboratory classes each with a different tutor and different groups of students. In each session the tutor used the Smart Lab system to monitor their students’ programming activities. At the end of each laboratory session the tutors were interviewed about the system.

The interview questions were divided into three sections:

1. The purpose of this section was to solicit the tutor’s reactions to the Smart Lab system and gather their opinions of the positive and negative aspects of the Smart Lab system.
2. This section explored the tutor’s experiences while using the Smart Lab system and with each of the Smart Lab features.

3. In this section the tutor was asked to rate the usefulness of the system. He/she was then asked to discuss the benefits of the system and suggestions for improvement.

5 **Results and discussion**

In the expert review session, all four lecturers reported that the Analyser/Visualiser software provided them with useful information about the programming activities of the other participants who were playing the role of students. They all also agreed that it helped them to identify the participants who were not progressing on their programming task and to understand their compilation behaviour and programming strategies. Three lecturers claimed that it was easy for them to do mental mapping between the labels representing the other participants and their actual location in the lab, and to identify the different kinds of problems they were facing.

In the lab study, the tutors stated that the graphical representation of the lab helped them identify the students’ programming activities. They agreed that the missing actions and syntax error boxes were useful in spotting the common problems that students encountered. The colour change in the labels and their borders also helped the tutor identify which students needed the most help. However, a couple reported that analysis of the missing and completed actions did not accurately reflect the students’ actual actions. This identified a problem with the Karam, Awad and Carbone (2010) tool used for the analysis.

The tutors also agreed that the students’ detailed window helped them to understand the students’ programming strategies and the problems encountered by each student. They found the final report useful in helping them to identify the amount of effort that individual students put into each task.

On a 10 point scale, where 1 = useless and 10 = very useful, the tutors both gave the system a 9 rating. One commented that “we might have rated it even higher if we had been able to use it for longer time”.

Overall, the experts and tutors found the Smart Lab system a valuable tool to helping instructors identify the common problems that the participants encountered with their code. They also found that the system enabled them gain a better understanding of the programming difficulties that the participants were having with their code. The tutors from the study lab commented that each student’s problems were easy to identify.

The Smart Lab system also seemed to provide enough information to the instructor to effectively help the students when the need arose. Both experts and tutors agreed that, based on the information provided by the system, they could better prioritize their time to help those students in need of most help. They were also able to provide the right type of help.

The experts and tutors agreed that the system is potentially very useful for gaining understanding of the students’ programming strategies and compilation behaviour. They found the detailed information that appeared when clicking on a particular helpful them in fully understanding the students’ programming and compiling strategies.
Finally, both experts and tutors agreed that the information in the final report was of great help in discovering the progress made by each student towards completing the exercise.

6 Future work and conclusion

This study produced the first implementation of a smart classroom technology for the introductory programming units’ labs. The implementation of the Smart Lab system was successful in achieving its intended aim of providing instructors with specific, timely and detailed information about their students’ performance on programming tasks during the lab class sessions. The Smart Lab facilitated the communication between the tutors and their students in the lab, and improved the overall effectiveness of the lab session in achieving the learning objectives. Results indicate that it is a useful aid for laboratory instructors by helping them:

- find the common problems that students faced with their code;
- address their students’ difficulties when the need arose;
- understand their students’ programming strategies and compiling behaviours;
- identify the progress made by each student in solving the programming exercises.

This study has opened up new research areas for further improvements and future work. The Smart Lab system could be extended in three different ways:

1. by allowing the tutor to provide online help and feedback directly to the students in their development platform. This saves the need to always go to the students’ machines or disturb the whole class by explaining a problem on the board, as it the case with the normal programming labs.

2. by providing automated help to the students about the errors they make and the actions they miss. The feedback could come in the form of hints. These hints would be generated as easy to read messages, based on the actions students have not yet implemented and their syntax errors.

3. by discovering whether use of such monitoring tool actually has an impact on students’ behaviour. The fact that there is someone looking over your shoulder could be inhibiting. The impact might even be positive - take a little more time to think about each error rather than repeating the tinker/compile cycle every fifteen seconds.

7 References

Arnow, D. & Barshay, O. (2002): WebToTeach: an interactive focused programming exercise system. 29th ASEWIEEE Frontiers in Education Conference San Juan, Puerto Rico, 12A9, IEEE.


