ELP – A WEB ENVIRONMENT FOR LEARNING TO PROGRAM

Nghi Truong, Peter Bancroft & Paul Roe
Faculty of Information Technology
Queensland University of Technology, AUSTRALIA
nk.truong@student.qut.edu.au
p.bancroft@qut.edu.au
p.roe@qut.edu.au

Abstract

Computer-based tutoring systems that can coach students in solving introductory programming problems have significant potential for improving the quality of programming education and reducing the instructor's work load. The purpose of this paper is to describe the current development status of the innovative Environment for Learning to Program (ELP) which provides an interactive web-based environment for teaching programming to first year Information Technology students at Queensland University of Technology (QUT). ELP allows students to undertake programming exercises by "filling in the blanks" of a partial Java program presented in a web page. The system then compiles the completed program and returns the resulting class to students in Java ARchive format (JAR) if successful, otherwise helpful messages concerning compilation errors are returned.

Keywords

Information Technology, computer programming, Java, web, tutoring system, XML, assessment, feedback

Introduction

Learning to program is acknowledged to be difficult. Teaching introductory programming classes is also extremely difficult. They have high attrition and a low success rate (Morrison & Newman, 2001). Compounding difficulties include: increasing class sizes, the need for multi-site and offshore delivery. With the growing number of students currently taking introductory computer programming courses and limited teaching resources, students are unlikely to get individualized attention and may learn the material poorly (Anderson & Skawarecki, 1986).

In the Faculty of Information Technology at QUT, ITB410 is the first programming subject in the Bachelor of Information Technology degree. The course does not require previous experience with computers and it is a prerequisite for other subjects in the degree. Java is the chosen teaching language for ITB410 and the emphasis is on problem-solving and programming skills. The subject is 13 weeks long and is delivered via a two-hour lecture and a one-hour tutorial per week. "Java: A framework for programming and problem solving" (Lambert & Osborne, 2002) is the recommended textbook.

Some two thousand students take ITB410 each year. Many of them have found that the subject is extremely difficult and thus the subject has a high failure rate, usually around 30% of the class. One of the biggest difficulties is that programming languages are artificial. They contain a high level of abstraction (Moser, 1997). Students need to know the syntax of the language and be able to think abstractly in order to be able to program. With a subject such as computer programming, there is a great deal of implicit knowledge that is difficult or impossible for lecturers or instructors to make explicit and therefore this information cannot be directly transmitted for the student to construct knowledge. In order to gain the knowledge and become competent in a domain, students need to go beyond explicit information to construct experiential implicit knowledge (Affleck & Smith, 1999). Students need to do a lot of practice

to acquire the knowledge and unfortunately, they encounter many difficulties which may make them lose confidence and delay starting to write their first program.

ELP is an online, active, collaborative and constructive Environment for Learning to Program, which is currently being developed at QUT to help students to program successfully at an early stage in their learning. The project is motivated by high failure rates amongst beginning programming students. It addresses three of QUT teaching and learning priority areas. Firstly, it is available by flexible delivery – the environment is online. Secondly, it can provide timely formative assessment by presenting students with fine-grained online exercises and answers. Thirdly, the environment supports the generic capabilities of problem solving, time management and working collaboratively.

First year IT students, most of whom presently struggle with programming concepts, need on demand tutorial assistance. By using the ELP system, students will be able to access as much tuition as they need. They will not be limited to three contact hours per week, nor will they be limited to standard working hours. Students will experience programming as a problem solving activity from the earliest stages of learning. They will be exposed to a variety of approaches to problem solving, the consequent alternative solutions, and the relative merit of such approaches through various feedback mechanisms.

The ELP project provides a learning environment which meets the diverse needs of students, for example those who find attendance at tutorials difficult, those who require more time for tutorials or those who are not used to formal tutorial situations.

Furthermore, the environment also promotes constructivism learning theory which claims that knowledge is actively constructed by the student, not passively absorbed from textbooks and lectures (Ben-Ari, 2001). Constructivism has been widely applied in modern education. In computer science education, constructivism is essential as beginning programming students must actively build a mental model of language constructs to become proficient or accomplished programmers and designers (Astrachan, 1998).

In the remainder of this paper we will discuss some difficulties encountered by students when they start writing their first program, describe the ELP system in more detail and the outcomes of the project, survey related work in automated tutoring and briefly describe future work.

The journey of writing a first program

Learning to program is a difficult process. Programming is not a single skill but a multi-layered hierarchy of skills, many layers of which will be active at the same time. Furthermore, not all programming knowledge can be transmitted by the instructors for the student to construct new knowledge - programming cannot be learnt without doing a lot of practice. The processes a student undertakes to create a working program are edit, compile, debug and run. Hence to produce their first Java program, students need to be able to use a program text editor, successfully install the Java compiler, know the Java language syntax, be able to compile a program and know how to debug the program using the error messages generated by the compiler. More often than not, beginners encounter many difficulties in using the software tools that comprise the supporting development system (Lewis & Watkins, 2001). Figure 1 illustrates common obstacles that beginners encounter when they start writing their first program.

A simple and efficient Integrated Development Environment (IDE) for beginner Java programmers needs to be intuitive to use, and small enough to carry on a single diskette. It needs to make the Java Development Kit (JDK) easy to use so students are studying Java, not spending large amount of time in learning how to use the IDE (Higginbotham, 2001). In ITB410, we use Program File Editor as the editor (PFE), and JDK as the compiler. PFE tends to be used by many Java instructors as it has the ability to launch the Java compiler and the interpreter to run the program. However, PFE's lack of syntax highlighting and a fully integrated compiler are two drawbacks (Lewis & Watkins, 2001).

Despite the fact that installation instructions are repeatedly handed out from the first week of the course, the majority of our students have problems installing the JDK on their own computers because they have to modify two system environment variables, PATH and CLASSPATH before they can compile and run programs. In week six of the subject, there are always students who still cannot manage to install the JDK. Since the subject does not require any previous experience with computers, there are students who are not

familiar with the concept of an environment variable. There are always a few students who manage to set these variables wrongly in spite of the detailed guidelines. Setting CLASSPATH wrongly may cause a more severe problem – the JDK may run an older version of a program after it has been updated which often results in anxiety for students.

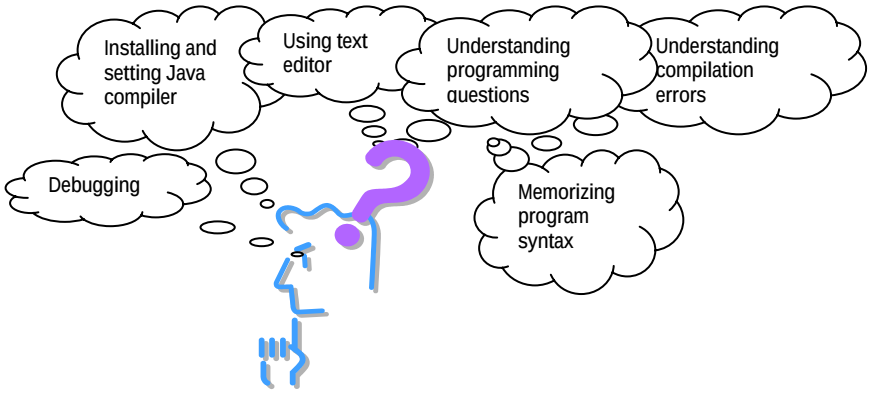

Figure 1: Difficulties when writing a first program

Most importantly, students have to learn some of the Java language syntax to be able to write any program. According to many instructors, Java is not designed as a teaching language but as an application development language (Andreae et al., 2000; Biddle & Tempero, 1998). Just to write a very simple program, a student needs to know many abstract concepts. To compound these difficulties, the student will be faced with unfriendly compilation error messages provided by compiler. Debugging a program is similarly as complex as writing a first program. It requires a variety of skills including an ability to coordinate information derived from sources such as error messages, the program plan, the program specification and the actual code.

Consequently, with all of the above problems, the result may be a serious delay in a student's progress. Lack of early success is a key issue leading to high failure rates, high level of anxiety, loss of confidence, and poor perception of programming. Thus students' learning is often hindered by the need to understand and use software tools which are ancillary to the core process of learning to program.

ELP

Students interact with ELP from a web browser. Most first year students are familiar with web browsers so there is no problem being able to use the system. Students will be presented with online interactive programming problems and exercises. Typically, students will work with program templates that focus their attentions on critical dimensions of the problem to be solved; and then receive timely appropriate, and directed feedback on their solutions to the exercises presented. It has been proved by many constructivists that programming templates promote knowledge integration, and they are a useful way to develop student problem solving skills (Hadjerrouit, 1998).

From the constructivism point of view, learning must be active. The student must construct knowledge assisted by guidance from the teacher and feedback from others students. It also emphasizes that the teacher must guide the student based on the understanding of each student's currently existing cognitive structures (Ben-Ari, 2001). Based on this view of teaching, the best way to teach a student is through one-on-one interactions in a problem-solving setting. In fact, many instructors and students would agree with this view merely from personal experience and observation, without needing educational theory. However, with the number of students taking introductory programming courses increasing steadily, it becomes impossible to implement this mode of instruction. The development of web technology means that online learning offers an alternative way to provide one-on-one interaction. On line learning enables students to progress at their own pace. A beginning computer science student has no effective model of a computer

that they can use to make viable construction of knowledge based upon sensory experiences such as reading, listening to lecturers and working with a computer. Therefore, it is crucial that the correct answer to a programming problem is easily and immediately accessible to beginners otherwise misconceptions may occur (Ben-Ari, 2001).

As can be seen from Figure 2, students are presented with online interactive programming problems and exercises. Feedback is generated from several sources - student peers, Peer Assisted Study Scheme leaders, duty tutors, tutors and unit coordinators. Automatic analysis and feedback will be supported. Small beginner programs are amenable to this; for example, they might respond: “Your program has the right structure and passes the following tests, but not this test”. This approach will emphasise problem-solving aspect of programming and enable finer grained assessment to be used. In general, students will learn by doing, getting feedback, and reflecting – all through the web. Learning will be improved by providing early and frequent feedback.

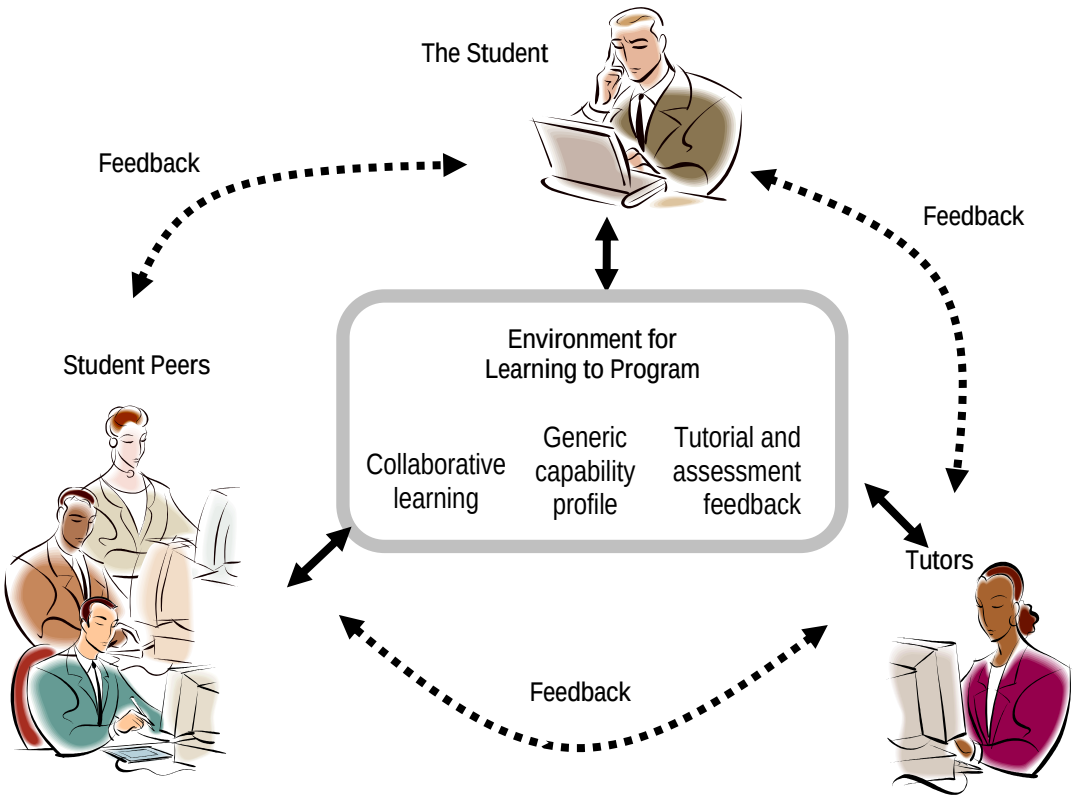

Figure 2: ELP

It is important to point out that all the information of the system is stored in Extensible Markup Language (XML). This is one of the characteristics that make ELP system different from other teaching and learning tool systems. XML is the universal format for structured documents and data on the Web (W3C, 2002). HTML documents are used to display a wide variety type of information across the network from simple text to complex multimedia with sound and animation. However, HTML tags are mostly for formatting objects. They do not give information about the content of a web page, and thus make it hard for the information to be reused in another context (Fitzpatrick, 2001). On the other hand, XML is designed to describe data. It provides a way to separate information from content. By doing so, it allows information to be reused in different ways by different applications. Furthermore, the authors of the XML can define their own tags and their own document structures (Usdin & Graham, 1998).

In the ELP system, the study guide of ITB410, the weekly tutorials, and the exercises are marked up using XML. Since XML allows users to define their own tags, and separates the formatting information from

the content, all the above documents have high level structures and user readability. An example of the exercise markup is shown in Figure 3. Other advantages of using XML include a high level of information reuse and flexible display.

```
<?xml version="1.0"?>
  <Exercise xml:space="preserve" indent="yes" id = "23" name="DisplayName" number="3" week="2"
level="introductory" type="Terminal">
<objective>This exercise will let you practice displaying text output in a Java program.</objective>
<outcome>You will be able to use the println method of the ScreenWriter class.</outcome>
<shorttext>Exercise: You will create a program that displays your name in a terminal window.</shorttext>
<questiontext>
Below, you will see an almost-complete Java program.
&lt;li&gt;Your job is to fix the program so that it displays your name.
&lt;li&gt;To help you get started, we have inserted a default value in the gap.
&lt;li&gt;writer.println( &amp;quot;Hi, my name is Lawrence&amp;quot; );
&lt;li&gt;Then change the message to print your own name.
&lt;li&gt;Click the "Submit" button, and follow the sequence of screens&lt;/li&gt;
</questiontext>
<code>
<class name="DisplayName">
import TerminalIO.*;

public class DisplayName {

   KeyboardReader reader = new KeyboardReader();
   ScreenWriter writer = new ScreenWriter();

   public void run() {
      <gap xml:space="preserve" indent ="yes">
<solution>
      writer.println( "Hi, my name is Lawrence" );
</solution>
<hint>
      writer.println( "Hi, my name is Lawrence" );
</hint>
</gap>
   }
   public static void main(String[] args)  {
      DisplayName tpo = new DisplayName();
      tpo.run();
   }
}
</class>
</code>
</Exercise>
```

Figure 3: XML exercise example

Table 1 lists all functionalities supported by the current ELP system.

Functions	**Description**
View study guide	Displays the subject study guide
View weekly tutorial	Displays all tutorial questions for a week
View exercise	Displays an exercise
Save	Saves a student's current work to a XML file and stores it in the student's directory in the server
Save and compile	Saves student work and compiles the completed program
Reset	Reloads all writable fields in the exercise page with instructors comments
View result	Enables a student to run the completed exercise by clicking on the Run button
View error	Views compilation errors if exist

Download work	Downloads the completed exercise source code
View solution	Views suggested solutions for all the gaps in the exercise

Table 1: Functionality supported by current ELP

Students are required to have the Java Runtime Environment (JRE) in order to use ELP. In the home page of the system, there is an applet designed to detect whether JRE is installed on the student machine. If it has not been installed, the student is redirected to a page where JRE can be downloaded and installed. The installation process of JRE is much simpler that JDK since it does not require setting environment variables and can be done automatically. Currently, the ELP system can only be viewed by Microsoft Internet Explorer with JRE supported. Students are requested to authenticate themselves in order to use the system. After entering their QUT username and password, the student is presented with a page that contains a list of weekly exercises organized according to the subject study guide, generated dynamically from an XML file. If this is the first time a student logs into the system, a directory structure is created for the student in the server. The directory structure is created dynamically according to the student's progress. This allows instructors to monitor the student's progress, check the student's work and identify common errors, thus adjusting teaching material to fit the student's need. From a constructivist point of view, lecturers cannot ignore students' preconceptions. When misconception occurs, lecturers need to know which logical model a student is using, then elicit the model and guide the student in its modification.

When the student makes a "view exercise" request, the ELP system checks the student's directory to see if the exercise was previously attempted. If not, the exercise is loaded from the server and presented to the student's web browser. Each exercise consists of a description of a programming problem and a template for a program. Each writable field in the template is occupied by a comment that describes the code that should be placed in the field. Those code sections which do not need modification are displayed as static text. The program source code of the exercise is displayed together with the line number for ease of locating errors. In the first few weeks the exercises have no gaps. Students simply compile and run the exercise. If the exercise has been attempted before, the student's previous work is loaded into each writable field accordingly. Thus, the student can make further changes to the exercise. If the reset button is clicked, all writable fields in the exercises will be reoccupied by a comment that describes the code that should be placed in the field. The student can save their current work by clicking on the save button. A screenshot of a typical exercise page is shown in Figure 4.

The student completes a program by filling in a form in a Web browser and then submits it to the server for compilation. If there are no compilation errors, the .class file of the exercise will be packaged together with other external classes in one .jar file. The student is able to run the program by clicking on the Run button on the result page. If there are syntax errors, compiler error messages are returned to the student.

The exercise seen by the student in Figure 4 is derived from marked up Java code, using XML. The XML text for this simple exercise is that previously given in Figure 3. As shown in the example, each exercise has a unique ID, which is stored in a text file resident in the server. This text file maintains a list of all exercises in the system, the week number and the number of the exercise. The <gap> tag in the XML file is displayed as an input field in the exercise page (the gap for missing code). Students are required to fill in one or more input lines.

Figure 4: An ELP first exercise

When a student successfully completes an exercise, they can view the model solution from the instructor or tutor. Although the system is not yet be able to suggest alternate solutions, the model solution can help to answer some common questions from students such as “Is my solution correct?” and “What other alternative solutions are there?”. In the case where the students fails to complete the exercise, easy access to the correct solution will help them develop logical thinking and eliminate some misconceptions, although the student may not fully understand why at the time.

Many teaching and learning tools use a Java applet as a mean to illustrate concept. Java applets have played an important role in education (Kamthan, 1999). Applets and other alternatives were investigated for ELP; for simplicity, performance and security, JAR (Java archive) files were used. These enable all files necessary for running an exercise to be packed in one archive which can be downloaded and run by students. Students can click on a hyperlink to a JAR file, and download and run their program as easily as if it were an executable program on their own machine.

Evaluation

A laboratory-based evaluation was conducted among 12 students to evaluate the usefulness of the system. Students were divided into three groups: six Peer Assisted Study Scheme leaders, three tutors, and three students who had just finished the ITB410 subject. Students had a chance to experience the system for 30 minutes. After that, they were required to fill in a feedback form.

Most of students found the system easy to use and agreed it would help them to learn programming at an early stage. Students who had just finished the subject indicated that the system would help them in thinking abstractly and develop their problem solving skills. They mentioned sometimes they could not understand questions in the textbook. However, with the code skeleton template in the ELP, they are given a better idea about what they are expected to do rather than trying to read and understand the text of a question.

However, the tutors and the PASS leaders raised two questions. Firstly, there will be a transition after using the ELP system to the real Java Development environment, since the system is not designed for advanced students. Secondly, since ELP presents gap exercises and students are not able to edit the skeleton, there is the possibility that the compilation errors generated by JDK might incorrectly point to a line number that the student cannot edit.

The ELP system will soon be introduced to small groups of students, around 20-30, in catch up classes. In ITB410, we hold catch up classes on week 5 in every semester to help students who either have found themselves having difficulty with Java programming or in fact have not even been successful in installing the JDK at home.

Furthermore, since it is a teaching and learning tool, evaluation is an on going process. We have integrated QUT's Web Online Feedback System (Nulty, Bancroft, Brewster, & Smith, 1998) , WOLF, into every exercise in the ELP system so that a student can give feedback about the content. WOLF is an online service which allows users to create snap questionnaires on the web and to receive responses directly and anonymously by email.

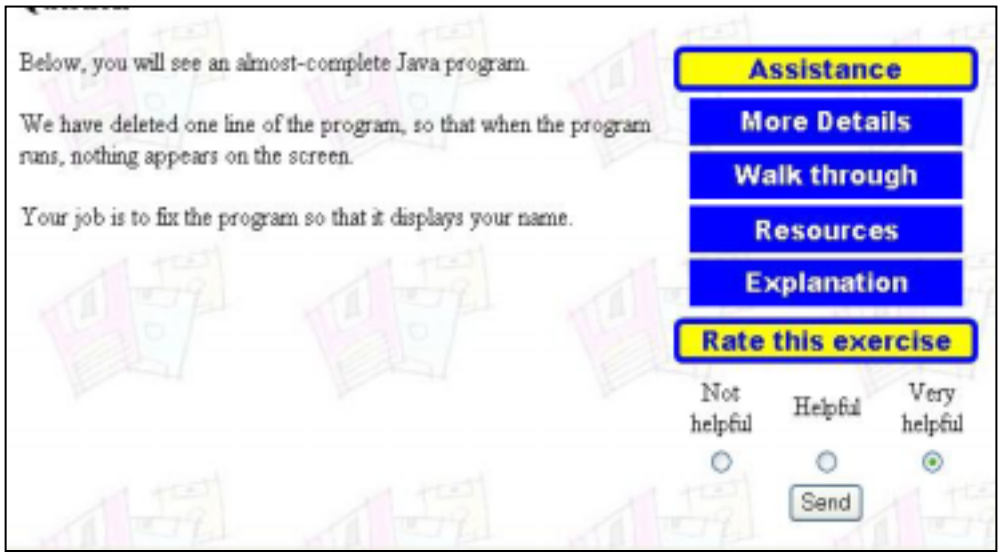

Figure 5: ELP and WOLF integration

Related Work

Computer-based tutoring systems that can coach students in learning introductory programming have long been an issue for discussion. Much research has been carried out to develop tools, which support learning programming and problem solving. However, there is still a lack of systems that provide a complete environment for learning programming and which stress the use of programming as a tool for solving problems. Emphasis must be on the issue of programming methods and problem solving, rather than the programming language features as an end to themselves. CourseMaster and InSTEP are quite similar to the ELP system.

CourseMaster (CourseMaster, 2000) is a client-sever system for delivering course based programming. It provides functions for automatic assessment of students' work, administration of the resulting marks, solutions and course materials. It is also able to detect plagiarism in students' programs. A student is able to develop a program, submit it to the server for marking or evaluation and get instant feedback. CourseMaster is a complete re-implementation after 10 years of experience using the Ceilidh (Foxley, 1999) system in many institutions. There are two main differences when comparing CourseMaster with ELP. Firstly, ELP provides a web-based interface for students to do exercises whereas CourseMaster requires students to provide the IDE that they want to work with. As a result of that, students have to learn how to use the IDE first. As mentioned earlier, learning how to use the IDE is one of the difficulties that beginning students encounter. It is considered to be one of the drawbacks of CourseMaster (Lewis & Watkins, 2001). Secondly, students are required to write a complete program from the beginning in CourseMaster rather than just filling in gaps.

InSTEP (Odekirk-Hash, 2001) is a web-based tutoring system . It is designed to help students learn looping in C, C++ and Java. Students complete a program exercise by filling in a web browser form and submitting it for evaluation. InSTEP examines the students' solutions, runs the students' programs and scans for mistakes in their solutions. If there is a mistake, it tries to determine the source of the mistake and give the student hints on how to fix the problem. Only student's input fields are parsed to detect errors. Thus, InSTEP can misdiagnose a student's problem because it does not understand program code. Another drawback of the InSTEP system is that it cannot analyse an arbitrary programming exercise. In contrast, ELP supports full Java language syntax. It is designed as a learning environment so it is supported for all course based materials. ELP marks up programs with XML as does CourseMaster whereas InSTEP does not.

Conclusion and Future Work

The strength of the ELP system is that it can provide an anytime, anywhere, easy access learning environment for students. It eliminates some of the unnecessary problems confronting students, such as installing the JDK and setting environment variables to compile and run Java programs. ELP supports "fill in the gap" style exercises, which reduces the complexity for students in writing their own programs. The basic requirements of the system are an Internet connection and a web browser which supports the Java Runtime Environment.

As mentioned earlier, this paper describes the current development status of the ELP system. In future work, the ELP system will be able to analyse students programs for code structures and complexity and evaluate the correctness. Currently, the automatic analysis of student program is being implemented. The analysis includes both static and dynamic analyses. Static analysis does not require execution of the student program whereas dynamic analysis executes the program via a set of test cases. The results of the analysis will be returned to the student instantly and it will also be saved in the student's directory. The ELP system will also be able to provide or suggest exercises for students adaptively. We also believe that collaboration work benefits problem solving efforts, even for relatively non-complex and small problems. It enhances confidence in the solution and enjoyment of the process. Thus, in later version of the ELP system, students will be able to do an exercise either alone or in a group. Lastly, we are looking forward to expanding the system so that it can also be used to support other advanced Java programming subjects in the course.

References

Affleck, G., & Smith, T. (1999). *Identifying a need for web-based course support*. Paper presented at ASCILITE 99, Brisbane, Australia.

Anderson, J. R., & Skawarecki, E. (1986, September 1986). The automated tutoring of introductory computer programming. *Communications of the ACM*, 29, 842-849.

Andreae, P., Biddle, R., Dobbie, G., Gale, A., Miller, L., & Tempero, E. (2000). Experience Teaching CS1 with Java. *Journal of Computer Science Education*, 14(1--2), 19-28.

Astrachan, O. (1998). *Concrete Teaching: Hooks and Props as Instructional Technology*. Paper presented at the Annual Joint Conference Integrating Technology into Computer Science Education, Dublin City University, Ireland.

Ben-Ari, M. (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics & Science Teaching*, 20(1), 24-73.

Biddle, R., & Tempero, E. (1998). *Java pitfalls for beginners. SIGCSE Bulletin*, 30(2), 45-82.

CourseMaster. (2000, April 3, 2000). CourseMaster [Online]. School of Computer Science & IT, The University of Nottingham, UK., Available: http://www.cs.nott.ac.uk/CourseMaster/cm_com/index.html [2002].

Fitzpatrick, C. (2001, 28-30 August). *An XML Approach to Creating Web-Based Learning Environments*. Paper presented at the 2nd Annual Conference of the LTSN Centre for Information and Computer Sciences, University of North London.

Foxley, E. (1999, Unknown). *Ceilidh on the World Wide Web* [Online]. Available: http://www.cs.nott.ac.uk/~ceilidh/ [1st Jan, 2002].

Hadjerrouit, S. (1998). *A Constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum*. Paper presented at the Annual Joint Conference Integrating Technology into Computer Science Education, Dublin City University, Ireland.

Higginbotham, T. F. (2001, Monday 22nd). *The Requirements for a User-Friendly Java IDE*. Paper presented at the Java & the Internet in the Computing Curriculum Conference, South Bank University, London.
Kamthan, P. (1999, 3rd December 2001). *Java Applets in Education* [Online]. irt.org. Available: http://tech.irt.org/articles/js151/ [1st July, 2002].
Lambert, K. A., & Osborne, M. (2002). *Java: A Framework for Programming and Problem Solving* (2nd ed.): Brooks/Cole.
Lewis, S. F., & Watkins, M. (2001, 22nd January). *Using Java tools to teach Java, the integration of Bluej and CourseMaster for delivery over the Internet.* Paper presented at the 5th Java in the Computing Curriculum Conference (JICC 5), South Bank University, UK.
Morrison, M., & Newman, T. S. (2001). *A study of the impact on student background and preparedness on outcomes in CS1*. Paper presented at the thirty second SIGCSE technical symposium on Computer Science Education, Charlotte, North Carolina, United States.
Moser, R. (1997). *A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it.* Paper presented at the Annual Joint Conference Integrating Technology into Computer Science Education, Uppsala, Sweden.
Nulty, D., Bancroft, P., Brewster, S., & Smith, D. (1998). *Web Online Feedback System* [Online]. Available: http://www.fit.qut.edu.au/wolf/ [24 July, 2002].
Odekirk-Hash, E. (2001). *Providing Automatic Feedback To Novice Programmers.* Unpublished MA, The University of Utah, Utah.
Usdin, T., & Graham, T. (1998). *XML: not a silver bullet, but a great pipe wrench.* StandardView, 6(3), 125-132.
W3C. (2002, 19/06). *XML* [Online]. Available: http://www.w3.org/XML/ [20 July, 2002].

Acknowledgements

The authors wish to thank the Queensland University of Technology Teaching and Learning Committee for the funding that allowed the ELP system to be implemented.

Copyright © 2002 Nghi Truong, Peter Bancroft, Paul Roe.

The author(s) assign to ASCILITE and educational non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The author(s) also grant a non-exclusive licence to ASCILITE to publish this document in full on the World Wide Web (prime sites and mirrors) and in printed form within the ASCILITE 2002 conference proceedings. Any other usage is prohibited without the express permission of the author(s).