A Web Based Environment for Learning to Program

Nghi Truong, Peter Bancroft, Paul Roe

Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001

nk.truong@student.qut.edu.au, (p.bancroft,p.roe)@qut.edu.au

Abstract

The purpose of this paper is to describe in detail the current development status of the innovative Environment for Learning to Program (ELP) which provides an interactive web-based environment for teaching programming to the first year Information Technology students at Queensland University of Technology (QUT). ELP allows students to program at the early stages of their course without the need to familiarize themselves with a program development environment. Most importantly, it eliminates all the difficulties associated with installing and running a Java compiler. Using ELP, students learn and develop their problem solving skills by working with program template exercises on the web. ELP provides a learning environment which meets the diverse needs of students.

Keywords: computer programming, Java, web, tutoring system, XML, online learning

1   Introduction

Computer science educators are currently faced with a serious problem. Large numbers of students either fail or withdraw from the course because of their inability to assimilate basic concepts taught at the early stages.

In the Faculty of Information Technology at QUT, Software Development 1 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 Software Development 1. The subject is 13 weeks long and is delivered via a three-hour lecture and a one-hour tutorial per week with the emphasis on problem solving and programming skills. "Java: A framework for programming and problem solving" (Lambert and Osborne, 2002) is the recommended textbook for the subject. More than a thousand students take Software Development 1 each year and an average of 35% of students have failed over the past seven semesters. Clearly, it is the

responsibility of teachers to provide beginning students with a learning environment which does not hinder the learning process and increases the success rate in introductory programming courses.

ELP is an online, active, collaborative and constructive environment for learning to program, which is currently being developed at QUT to help Information Technology students to program successfully at an early stage in their learning and to assist teaching staff in the marking task. The project is motivated by the high failure rate among beginning programming students.

ELP directly addresses the QUT teaching and learning objectives. The university has embraced online teaching and delivered benefits to its students such as making it easier for them to work collaboratively, giving greater opportunity for feedback and practice, providing easier access to learning resources, encouraging reflective practices in learning and allowing students to progress at their own pace. The university is aiming to provide a learning environment which meet students' needs, giving greater flexibility in time and place of teaching and learning (QUT, 2002).

Most importantly, ELP enables the smooth integration of lecture notes, tutorial and practical exercises. With ELP, students will be able to learn to program and get feedback at any time, anywhere. Students are 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. They will learn by doing practice exercises, getting feedback and reflecting – all through the web. Students work using program templates that focus their attention on critical dimensions of the problem to be solved. Student learning will be guided towards key aspects of problem solving. Students will experience programming as a problem solving activity from the earliest stages of learning. They will be exposed to different approaches to problem solving, the consequent alternative solutions, and the relative merit of such approaches through a variety of feedback mechanisms.

The contribution of this paper is to describe a novel web based environment for helping beginning students to learn programming.

Section 2 of this paper will discuss some current problems of an introductory programming course and how ELP tackles them. The overview and the internal architecture of the current development status of ELP are described in detail in Sections 3 and 4. The result

of system evaluation is reported in Section 5. A review of related works is discussed in Section 6. Lastly, the future development plan of the system is mentioned in Section 7. The ELP system is available from http://elp.fit.qut.edu.au/ELP/ELP.html.

2 Current educational context and ELP design consideration

This section discusses the current problems in introductory programming courses and how the ELP system attempts to solve these problems.

2.1 Educational context

Computer science education currently faces three major problems. Firstly, a majority of novice programming students have difficulty in either starting their first program or constructing abstract programming knowledge. Secondly, the class size of introductory programming courses is continuously increasing which causes difficulty in providing an effective learning environment. Lastly, there is an increasing requirement to support multisite campuses and offshore delivery.

Learning to program is acknowledged to be difficult. One of the biggest difficulties is that programming languages are artificial (Moser, 1997). Programming knowledge cannot be directly transmitted from instructors to students to construct - it must be acquired actively by students (Ben-Ari, 2001). Listening to lecturers and reading reference materials are the primary means to construct knowledge but most importantly, repetition by practising will ensure that the knowledge is retained. Thus in order to succeed in a computer science course, students need to do a lot of practice. Unfortunately, they encounter many difficulties, which may make them lose confidence and delay starting to write their first program.

The processes a student undertakes to create a working program are edit, compile, debug and run. The most commonly used Java programming environment comprises the Java Development Kit (JDK), a text editor, and a console shell. Hence to produce their first Java program, students need to be able to understand and use all these tools, understand the Java language syntax and semantics; and know how to debug the program using the error messages generated by the compiler. Not surprisingly beginners encounter many difficulties in using the software tools for programming (Lewis and Watkins, 2001). The first hurdle facing many beginning Java programmers is to successfully install the JDK on their machine. An important step in installing the JDK is to modify the PATH and CLASSPATH system environment variables. This must be done correctly in order to successfully compile and run Java programs. Unfortunately, many students lack the knowledge to be able to modify the system environment variables which leads to an incomplete installation of the JDK.

A major challenge in computer science education is to be able to provide an effective learning environment for students while contending with ever increasing class sizes. Beginning programming students must actively construct knowledge assisted by guidance from teachers and feedback from others students (Ben-Ari, 2001). Based on this point of view, the best way to teach a student is through one-on-one interactions in a problem-solving manner. In fact, many instructors and students would agree with this view merely from personal experience and observation, without needing educational theory. However, with the steadily increasing number of students taking introductory programming courses, it becomes more and more difficult to implement this mode of instruction.

Lastly, there is the need for multisite and offshore delivery. With the evolution of technology, particularly the Internet, educational institutions can extend their reach beyond commuting students and teaching staff. Most Australian universities have more than one campus and many of them have a presence outside the country. A large number of students study in offshore mode and this figure will be continually increased. In Australia, there were a total of 95 607 university students in year 2000 and of these, 23 891 students (approximately 25%) studied in offshore campuses (Commonwealth of Australia, 2000).

2.2 ELP design consideration

ELP was designed to address the three problems outlined in Section 2.1 by providing a web user interface, program template exercises and server side compilation.

ELP incorporates a simple web based program editor. A web browser is considered to be the simplest and easiest interface to use when compared with existing Integrated Development Environments. With the chosen user interface, the system solves the problems previously identified, including difficulties that students encounter in learning how to use programming development tools, providing an effective learning environment and supporting multisite and offshore delivery. In addition, the web interface also increases the level of flexible delivery and overall accessibility of the system. Students and staff are able to use the ELP system anywhere at any time.

The study of Affleck and Smith (1999), has found that the main problem with beginning programmers is accessing prior knowledge and adopting an approach to study that goes beyond memorising explicit knowledge necessary to apply and transfer the domain concepts to new situations. We believe that using program template exercises is one of the best ways to stimulate prior knowledge from students and help them to apply concepts to new situations. ELP provides students with gap filling programming exercises. The code in the exercises is generally based on known concepts; the missing code contains the new concepts. The students need to understand the given code in order to complete an exercise. Furthermore, gap filling exercises reduce the complexity in writing programs.

Thus the approach builds up students' confidence. Students are motivated and engaged in their learning process more actively.

One of the most common difficulties which prevents students starting to program is to install and set up the compiler correctly. ELP allows students to compile their exercises on the server without any client side Java compiler installed in the student machine. Thus students are able to do practice exercise without having to set up the JDK at home.

3   The ELP

This section describes how students interact with the ELP.

3.1   System overview

ELP is a web based client server system. The student is presented with Java program template exercises as web pages. The completed exercise is submitted to the server for compilation. The resulting ".class" of the exercise is packed together with other necessary libraries in a JAR file and subsequently downloaded and run on the student's machine. Figure 1 gives an overview of the system.
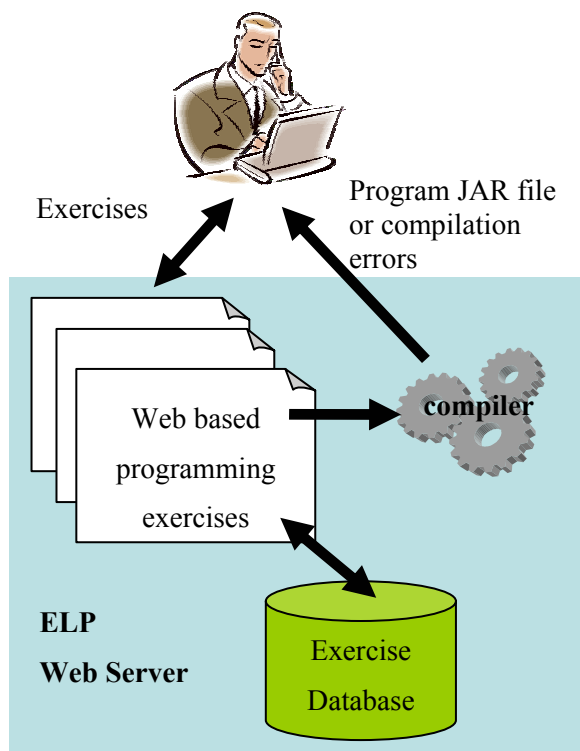

Figure 1: ELP system overview

3.2   Student view of the ELP system

The only system requirement of the ELP is that the Java Runtime Environment (JRE) must be installed on the student machine. For ease of use, ELP has established a mechanism to instruct students through the JRE installation process. An applet is included in the system home page to detect if JRE is installed on the student's machine. If it has not been installed, the student is redirected to a page where JRE is downloaded and installed automatically.

Currently, the ELP system requires Internet Explorer (IE) version 5 or above, but this will be relaxed in future releases. Students are requested to authenticate themselves in order to use the system. After logging in successfully, the student is presented with the topics index page which contains links to all the teaching topics in Software Development 1. When a topic is selected, the student is directed to an exercises page containing links to exercises relating to the chosen topic. Each exercise is displayed as a link together with a short description on the page. The student can navigate to the topics index, next topic's exercises and previous topic's exercises from the current topic's exercises page. The following diagram shows the document hierarchy in the ELP system.
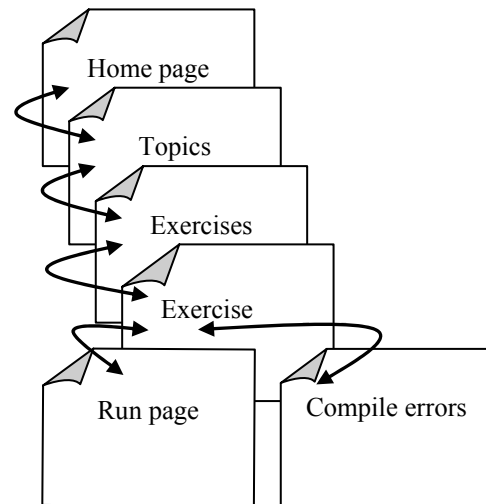

Figure 2: Document hierarchy

Each exercise page has a brief description about the exercise, learning objectives and learning outcomes. It also has a concise statement of a specific coding problem and a code template which consists of a program with one or more gaps. The code template is a sequence of one or more static text areas and writable fields. Those code sections which do not need modification are displayed as static text. A writable field is able to expand and shrink as required according to the range of input text. At the end of the page, there are three buttons: "save", "compile and save" and "reset". A screen shot of a typical exercise page is shown in Figure 3.

```
1    import TerminalIO.*;
2
3    public class KiloNaut {
4
5        KeyboardReader reader = new KeyboardReader();
6        ScreenWriter writer = new ScreenWriter();
7
8    public void run() {
9            // Let conversionFactor = the value of a numeric
10           // expression that relates nautical miles to kilometres
11           // Print "Please enter number of kilometres: "
12           // Read kilometres
13           // Let nauticalMiles = kilometres times conversionFactor
14           // Print "This is the same as "
15           // Print nauticalMiles
16           // Print " nautical miles "
17   }
18
19       public static void main(String[] args)
20       {
21           KiloNaut tpo = new KiloNaut();
22           tpo.run();
23       }
24   }
```

The student enters text

Figure 3: An example of an ELP exercise

In addition, each exercise page also has links to related information including general help about the ELP system, more detailed instructions about the specific coding problem and other related resources. Additional instructions for coding the problem may include structure charts, class diagrams, pseudo code and screen shots showing how the program is expected to look when it runs. Lecture notes, tutorial materials, links to related exercises and possible links to other information on the Internet are included in the related resources. A small evaluation question is attached to every exercise. The student is given three options: not helpful, helpful and very helpful to rate the exercise.

If it is the student's first attempt at an exercise, each writable field in the template is occupied by a comment that describes the code that should be placed in the field. If the exercise has been done before, the student's previous work is loaded into each writable field. Exercises are organized in increasing level of difficulty. Some exercises have no gaps; the student simply compiles and runs the exercise. This allows the student to experience how ELP works and to see some complete programs. A gap can be as small as an expression, or as large as a complete Java program. There may be a single or multiple gaps in an exercise. The program source code of the exercise is displayed together with the line number for ease of locating errors.

The student completes a program by filling in all the writable fields of the exercise template then clicks the "compile and save" button. The exercise is sent to the server for compilation. 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.

If there are no errors in compiling the exercise, the student is presented with a result page. The completed exercise is downloaded and executed automatically on the student's machine by clicking the "run" button provided on the page. The result page contains links to view the model solutions provided by the instructors or tutors and to download the student's completed source code for the exercise as well as the links to the next and previous exercise and to the topics index page.

If there are syntax errors, compiler error messages are returned to the student on an error page. By clicking the "return to code" button, the student can go back and fix the errors in the code they have just written. The error messages are simultaneously displayed in a separate window when the student returns to the code.

Currently, ELP supports two types of exercises, console and Java Swing. These correspond to the two libraries, TerminalIO and BreezySwing used in the required textbook.

4   The architecture

The ELP is developed using servlet technology which extends the capabilities of servers that host applications access via a request-response

programming model. Apache Tomcat is used as a web container and it currently operates on a Linux server.

In ELP, all study material such as the topics index of a subject, the exercises and exercise information are stored in eXtensible Markup Language (XML) format. This is one of the characteristics that makes ELP different from other teaching and learning tools. With the use of XML, information in the ELP has a handy structured representation and high level of reuse.

The rest of this section will describe in depth the internal architecture and implementation of ELP. Figure 4 shows the flowchart of the ELP system.
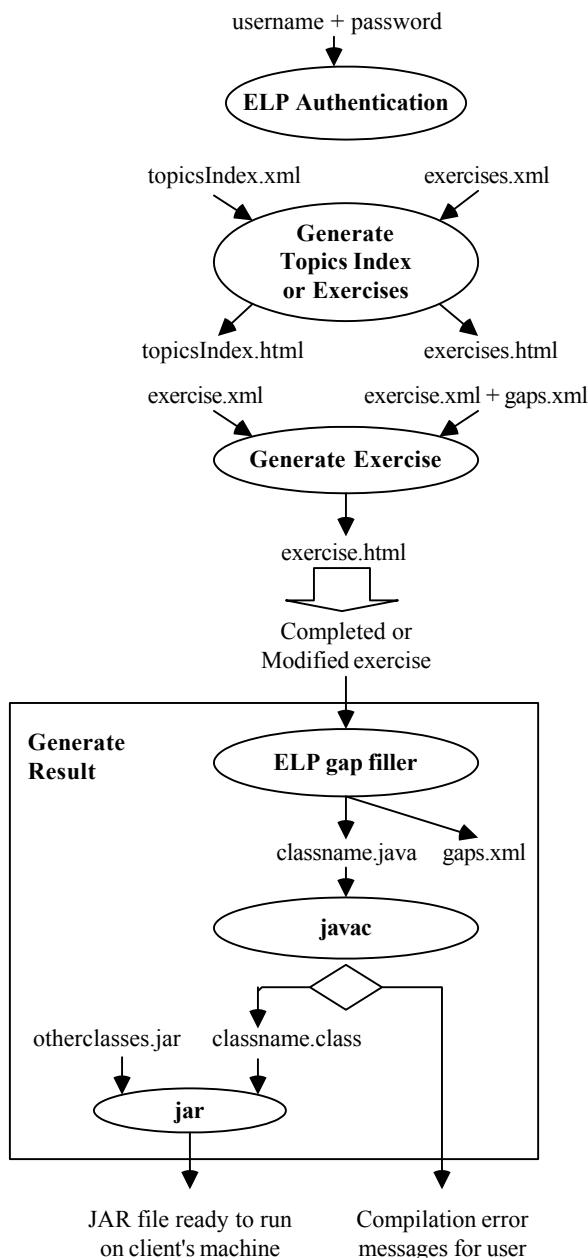

Figure 4: ELP system flowchart

4.1 ELP authentication

This servlet is responsible for authenticating students and for creating a home directory on the server for first time users. The servlet communicates with the university's authentication database to validate the student's username and password. Figure 5 depicts the directory structure created for all users.
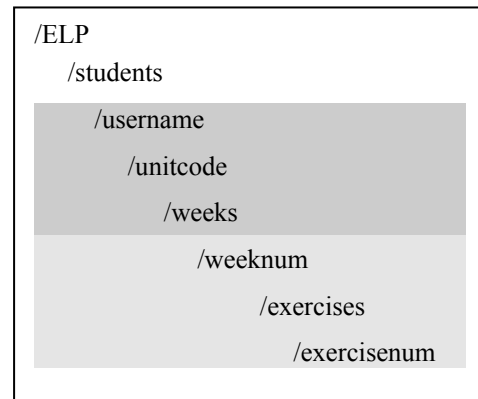

Figure 5: A student directories example

In the structure, both ELP and students are system directories. The username, unitcode and weeks directories are created by the authentication servlet. The remaining directories are created on the fly by the result servlet when the students attempt an exercise. A list of the units in which the student is current enrolled is obtained from the university database after successfully validating the student. Each current enrolled unit is given a separate directory on the server.

A HTTP session object is assigned for each authorised student to keep track of the exercise that the student is currently doing. The student's username is set as the key attribute of the session object to differentiate among multiple student's sessions.

4.2 Generate Topic Index and Exercises

The process to generate the topics index and the exercises themselves is identical - read the XML file, apply a pre-defined HTML template to construct the page and send it back to the student. The only difference is that the topicsIndex.xml file is parsed and processed to generate the topicsIndex.html page whereas the exercises.xml is processed to generate the exercises.html page.

4.3 Generate Exercise

Each exercise on the system is stored in an XML document (exercise.xml), located in a specific directory and has a unique id. The XML text for the simple exercise from Figure 3 is given in appendix A.

The <gap> tag in the XML file is displayed as an expandable, writable field in the exercise page.

There is a text file resident in the server to maintain a list of all exercises in the system including exercise ID, the week number and the number of the exercise. When an exercise is selected, the exercise id is passed to the GenerateExercise servlet together with the request. The servlet will look up the text file to find out the location of the exercise.xml file of the selected exercise. It then processes the file and generates the exercise.html dynamically. The current exercise information is set as another attribute in the session object.

If the exercise had been previously attempted, there would be a gaps.xml file, which stores the student's input and which resides in the student current exercise directory. This file is processed together with the exercise.xml to restore the student's previous work to all the writable fields in the exercise page.

As described in section 3, there are three main functionalities in the exercise page: "save", "compile and save" and "reset". When the "compile and save" button is clicked, the "Generate Result" servlet is invoked. When the student clicks the "save" button, all the student's input is stored in the gaps.xml file. This process is executed by a small and simple servlet. When the student clicks the "reset" button, the GenerateExercise servlet is reinvoked. However, it only reads the exercise.xml file to generate the page without checking the existence of the gaps.xml file - all the writable fields are then re-occupied by instructors' hints.

4.4 Generate Result

This servlet is responsible for building, compiling, and making a JAR file for a completed Java programming exercise.

When invoked, the servlet gets the student's input in all the writable fields of the template and stores it in the gaps.xml file. It then builds a complete Java program from the static text of the code together with the input. The Java program is saved in the exercisenum directory shown in Figure 5.

A process is created to invoke javac, the java compiler. Depending on the success of the compilation process, either the result page or error page is returned to the student. If there are no compilation errors, the resulting class files and all necessary libraries are packed into a JAR file and ready to run. A result page is generated which links to the JAR file.

5 Evaluation

A laboratory-based evaluation was conducted among 30 students at the end of week 5, in catch up classes to evaluate the usefulness of the system. In Software Development 1, catch up classes are held in the middle of 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. The students experienced the system for two hours. After that, they were requested to fill in a Web Online Feedback Form, WOLF (Nulty et al., 1998), an online service which allows users to create snap questionnaires on the web and to receive responses directly and anonymously by email.

The students were asked the following questions:

1. The ELP system makes it easier for me to write Java programs.

A. I agree

B. I disagree

C. I neither agree nor disagree

2. Using the ELP system helps me to understand Java programs

A. I agree

B. I disagree

C. I neither agree nor disagree

3. I would like to continue to use ELP for the rest of this unit

A. Yes

B. No

4. How do you feel about the ELP system?

At the end of each question, the students were able to give additional comments on the system.

Twelve students completed the evaluation form. All twelve students mentioned that they are currently having problems in learning the unit. Two students in the group had never written a Java program as they thought they could not do it. One student in the group had problems in running the compiler in the command line and as a consequence, the student had never been able to compile a program.

The result of the evaluation is tabulated in the following table:

	Agree (Yes)	Disagree (No)	neither agree nor disagree
Question 1	11	0	1
Question 2	12	0	0
Question 3	12	0	0

Table 1: ELP evaluation result

Overall, the students enjoyed using the system. All students agreed that the system makes compiling and writing a program so much easier. They also stated that they believe that using the ELP is a good way to learn programming. The students agreed that programming exercises become easier with the gap type exercises so that they can concentrate on the main parts of the problem. A student suggested that the system would be

best used in laboratory tutorials because students can ask tutors if they have any questions. The student who chose "neither agree nor disagree" as the answer in question 1 stated "it depends on the amount of time that I had" as the reason.

Since it is a teaching and learning tool, evaluation is an on going process to develop a system which meets students' needs. A small WOLF question is attached to every exercise. Students can rate the helpfulness of an exercise by choosing one of the following three options: not helpful, helpful and very helpful. When a student clicks on the send button, an email is sent to the exercise's author. The email simply contains the exercise number and the student feedback. The student can only submit their feedback once every time they do an exercise. The student's identification is not shown on the email feedback for confidential purposes. An example of how the student can rate the usefulness of the exercise is show in Figure 6.

Figure 6: ELP and WOLF integration

6   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, not many systems provide a complete environment for learning programming with the emphasis on problem solving skills rather than the programming language features (Deek and McHugh, 1998).

A survey was conducted by Deek and McHugh (1998), to analyse and uncover the limitations of existing tools for learning programming. The survey analysed 28 systems in four categories: programming environment, debugging aids, intelligent tutoring systems and intelligent programming environments. The study revealed that most of the existing systems have four main limitations which stopped them achieving their

goals. Despite the fact that lack of problem solving skills is the main central difficulty of novice programmers (Mayer, 1981, Deek and McHugh, 1998), most of the current tools do not have problem solving and software engineering frameworks. They are basically an environment for coding and testing. Novice programmers more often are required to develop a complete solution. Thus, they may be overpowered by the need to simultaneously understand the problem, apply design methodologies and implement details. In addition, existing tools do not have sufficient testing and evaluation and are not integrated into the curriculum. There is no evidence in the literature of a system which has become an integral part of the learning process and which is used regularly by students and teachers to enhance the learning environment. Researchers admit a number of factors were not addressed including: no attempt was made to consider the curriculum content; overall curriculum objectives were ignored in favour of immediate results. Finally, existing tools all tend to have a complicated interface. Students typically need additional tutorial support to overcome basic operational difficulties.

All four main problems of previous tools are addressed in the ELP system. ELP address the first drawback in existing tools by providing students with program template exercises. As mentioned in section 2.2, this type of exercise maximizes a student's knowledge integration - thus ELP develops student's problem solving skills step by step. ELP overcomes the lack of testing evaluation by continuously introducing the system to students and teaching staff on a semester basis during the 2 year development period, thereby obtaining feedback to develop the system in a way which meets the needs of students and teaching staff. To ensure ELP will become an integral part of the course, the focus of each exercise is closely related to the study guide of the subjects. Exercises are selected from the textbook with only partially completed code. Students only fill in the code section which is the main focus of the exercise. More importantly, ELP is a web and server based application. Thus, no additional practical session is required in order to use the system.

BlueJ, CourseMaster, WebToTeach and CodeSaw are systems which quite similar to the ELP system.

BlueJ (Kolling, 2002) is designed and implemented at Monash University, Melbourne Australia for teaching object-orientation to beginners. The application runs on top of the Java 2 Platform, Standard Edition (J2SE) and uses the standard compiler and virtual machine. BlueJ visualises Java abstract concepts by using Unified Modelling Language (UML) diagrams. It also provides an interactive interface that allows students to create and test objects, methods, method call and fields. It has the ability to run applications and applets. It provides a simple debugger. BlueJ is different from ELP in three aspects. Firstly, the teaching aim of BlueJ is different from that of the ELP. ELP is used to teach students Java syntax and problem solving skills whereas BlueJ was designed to teach object-oriented concepts. Secondly, the ELP system requirements are

more easily satisfied by beginning programmers than for BlueJ. Although ELP requires the installation of JRE on students' machines, the installation process is much simpler since it does not involve setting environment variables. Moreover, the ELP also provides a simple and easy to use programming environment, the web browser. Lastly, BlueJ requires the student to write a complete object oriented program. In contrast, the ELP uses program template exercises. Students only have to focus on the critical dimension of the problem to be solved. Students are able to build up their problem solving skills step by step with the increasing level of difficulty of exercises in the ELP system.

Course Master (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 Integrated Development Environment (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 and Watkins, 2001). Secondly, students are required to write a complete program from the beginning in CourseMaster rather than just filling in gaps.

WebToTeach (Arnod and Barshay, 1999) is a web based automatic homework checking tool for computer science classes. It was developed at Brooklyn College and is currently used in ten computer science courses. WebToTeach supports Java, C, C++, Fortran, Ada and Pascal and incorporates various types of exercises including writing a code fragment, writing data for a test suite, writing a complete single source program and writing several source files. Students are given either a single text area or multiple text areas on the web browser to provide the solution depending on the exercise type. Upon submitting the solution for the exercise, students are told immediately whether it is correct or not. In the case of failure the student is given information about the cause of failure. WebToTeach also provides facilities for instructors to forgive lateness, view the first and last correct submission of a student, directly send mail to individual students, broadcast mail to students, set and edit the message of the day, and get statistics on homework completions. The ELP system is different from the WebToTeach system in four ways. Firstly, ELP is developed to provide an environment for students to learn rather than for checking assignments. Secondly, ELP

provides a nicer program editor. Thirdly, source code is displayed together with line number, which makes the debugging tasks much easier. Finally, with WebToTeach, students do not have access to either the executable program or the returned result of the exercise whereas in ELP, the JAR file can be downloaded and run on the student machine. Lastly, information in ELP is marked up using XML. By doing so, the information has a handy structure and is easy to reuse.

CodeSaw (Liqwid Krystal, 2002) is an online digital workspace which is designed to enable sample programs within a technology book to be compiled and run without the need for installing a development environment or having to type in code. Currently CodeSaw supports instantaneous learning for C, C++, Java, Perl, Python, Ruby, XML, HTML, PHP, SQL, PL/SQL, and JavaScript. Users log in to CodeSaw and download code samples which they can view, compile, run, and edit.

Many teaching and learning tools use Java applets as a means to illustrate a concept. Java applets have played an important role in education. Applets and other alternatives were investigated for ELP and two problems were encountered with their use. Firstly, the browser caches an applet and it is difficult to clear the cache. When the student updates the exercise that they have just successfully compiled, the cache has not yet expired so the result sent back is the old version of the exercise. Secondly, with the use of applets students will not be able to run file input-output exercises on their own machine due to the fact that an arbitrary applet does not have enough privilege to do so unless it is a signed applet. The process to sign an applet is not easy. For simplicity, performance and security, JAR files were decided upon for ELP. These enable all files necessary for running an exercise to be packed into one archive which can be downloaded and students can run their program on their own machine as easily as if it were an executable.

Not so long ago students learnt to write programs by logging on or submitting jobs to a mainframe computer. Later there was a move to PCs and client side programming. ELP represents a hybrid between the two approaches - programs are compiled on the server but run locally. Furthermore, ELP represents an e-learning environment where the learning process involves students submitting exercises to the server for compilation and receiving the result class interactively.

7 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,

Internet Explorer version 5 or later, and the Sun Java Plugin.

As mentioned earlier, this paper describes the current development status of the ELP system. In future work, the ELP system will expand in four major aspects. Firstly, the system will support automated analysis of student programs and provide feedback. Secondly, email facility will be provided at every exercise. If a student is struggling to write a piece code, the student could email the offending code fragment to a duty tutor. A simple email will be sent to the student to notify that help has been given. Thirdly, a simple exercise authoring tool will be developed for teaching staff. Currently, the topics index, exercises and exercise XML templates are given to lecturers and instructors. Lecturers and instructors developing the new material then have to copy all the documents to the server manually. 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.

8   Acknowledgments

9   References

Affleck, G. and Smith, T. (1999) In *ASCILITE* Online, Brisbane, Australia.

Arnod, D. and Barshay, O. (1999) In *IEEE Frontiers in Education Conference*Department of Computer and Information Sciences, Brooklyn College.

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

Commonwealth of Australia (2000), Vol. 2002.

CourseMaster (2000), Vol. 2002 School of Computer Science & IT, The University of Nottingham, UK.

Deek, F. and McHugh, J. (1998) *Computer Science Education,* **8,** 130-178.

Foxley, E. (1999), Vol. 2002.

Kolling, M. (2002), Vol. 2002.

Lambert, K. and Osborne, M. (2002) *Java: A Framework for Programming and Problem Solving,* Brooks/Cole.

Lewis, S. and Watkins, M. (2001) In *5th Java in the Computing Curriculum Conference (JICC 5)*South Bank University, UK.

Mayer, R. (1981) *ACM Computing Survey,* **13,** 121-141.

Moser, R. (1997) In *Annual Joint Conference Integrating Technology into Computer Science Education*ACM Press   New York, NY, USA, Uppsala, Sweden, pp. 114-116.

Nulty, D., Bancroft, P., Brewster, S. and Smith, D. (1998), Vol. 2002.

QUT (2002), Vol. 2002 QUT.

Appendix A: XML exercise example

```xml
<?xml version="1.0"?>
  <Exercise xml:space="preserve" indent="yes" id = "27" name="KiloNaut" number="7" week="2" level="introductory"
type="Terminal">
<objective>This exercise will:
            1.    Let you practice using Java's arithmetic operators
            2.    Reinforce the tasks of reading and writing numeric data in a simple Java program
</objective>
<outcome> You will be able to
            1.    Write Java expression involving multiplication and division operators
            2.    Use the println method of the ScreenWriter class to get numeric input from the user
            3.    Use the ELP system to successfully edit, compile and debug a simple Java program.
</outcome>
<shorttext>Write a program that converts kilometres to nautical miles. </shorttext>
<questiontext>
In this question, you are required to write a program that  converts kilometres to nautical miles. The user will be asked
to enter the number of kilometres. Your program must take that value, convert it to the corresponding number of nautical miles, and
display the result on the screen.
</questiontext>
<code>
<class name="KiloNaut">
import TerminalIO.*;

public class KiloNaut {

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

    public void run() {
<gap xml:space="preserve" indent ="yes">
<solution>
            double conversionFactor = 90.0 * 60.0 / 10000.0;
            writer.print( "Please enter number of kilometres: " );
            double kilometres = reader.readDouble();
            double nauticalMiles = kilometres * conversionFactor;
            writer.print( "That is the same as " );
            writer.print( nauticalMiles );
            writer.println( " nautical miles." );
</solution>
<hint>
        // Let conversionFactor = the value of a numeric expression that relates nautical miles to kilometres
        // Print "Please enter number of kilometres: "
        // Read kilometres
        // Let nauticalMiles = kilometres times conversionFactor
        // Print "This is the same as "
        // Print nauticalMiles
        // Print " nautical miles"
</hint>
</gap>
    }

    public static void main(String[] args)
    {
        KiloNaut tpo = new KiloNaut();
        tpo.run();
    }
}
</class>
</code>
</Exercise>
```