Learning to Program Through the Web

Nghi Truong, Peter Bancroft and Paul Roe
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
+61 7 3864 9323
{n.truong, p.roe, p.bancroft}@qut.edu.au

ABSTRACT

Computer-based tutoring systems which assist students in solving introductory programming problems have significant potential for improving the quality of programming education and reducing the instructor's work load. The innovative Environment for Learning to Program (ELP) 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 gaps" of a partial computer program presented in a web page and to receive guidance in getting their programs to compile and run. Feedback on quality and correctness is provided through a program analysis framework. Students are given the opportunity to produce working programs at the early stages of their course without the need to familiarize themselves with a complex program development environment.

Categories and Subject Descriptors

K.3.1 [**Computer Uses in Education**]: Computer-assisted instruction.

General Terms

Documentation, Design, Experimentation, Human Factor, Languages.

Keywords

Computer programming, flexible delivery, web, tutoring system, online learning, feedback

1. INTRODUCTION

Programming skills can only be developed by extensive practice. Unfortunately, when attempting to write their first programs, novices may be hindered by difficult side issues such as learning how to use a program text editor, installing a language compiler, and knowing how to compile and debug the program using the error messages generated by the compiler. Lack of early success is one of the key issues leading to the low success rates and high attrition in introductory programming classes [7]. Students lose confidence and have a poor perception of programming. This is not helped by large class sizes and the need for multi-site and offshore delivery. Students in a big class with limited teaching resources are unlikely to get the individualized attention they need, leading to an unsatisfactory learning experience and increased possibility of failure [1].

When learning to program, it is essential that students are given the opportunity to practise in an environment where they can receive constructive and corrective feedback. Feedback is an especially important factor in the learning process when it is available on request. However, with large class sizes, it is difficult for teaching staff to synchronize their heavy schedules to provide additional help when the students need it.

The main contribution of this paper is to describe an innovative Environment for Learning to Program (ELP) which is designed to help students to program successfully at an early stage in their learning. ELP [9] is an online, active, collaborative and constructive web environment to support novice programmers. The system has been used in the Faculty of IT at QUT to provide tutorial support for Java and C# novice programmers for the past two years. It is available for trial from http://elp.fit.qut.edu.au/.

There are five distinguishing characteristics of the ELP system:

1. The system supports fill in the gap programming exercises and customized compilation error messages which reduce the complexity of writing programs. This allows students to focus on the problem to be solved and engages them more actively in the learning process.
2. The system is web based which eliminates the programming environment difficulties that students usually encounter. This also enables smooth integration of programming with lecture notes, tutorials and other web based content.
3. The ELP incorporates a program analysis tool which provides instant feedback for students about the quality and correctness of their programs.
4. The ELP supports configurable exercises i.e. the exercises and the environment can be configured for different stages of students learning.
5. The system allows tutors to provide additional feedback through annotations on students' programs

In this paper, we will describe the ELP system and the program analysis framework in more detail, before discussing the results of system evaluations. We also survey related work concerned with automated tutoring and conclude by briefly considering future extensions and improvements for the ELP and its program analysis framework.

2. THE ELP SYSTEM

This section describes the ELP in detail and considers how students and lecturers interact with the system.

2.1 System Overview

ELP is a web-based client-server system. It uses Java applets to enable programming exercises to be embedded in other web pages such as those served by a web Content Management System (CMS). In this way ELP can extend and coexist with an existing CMS without needing to change it. See Figure 1.

ELP currently supports Java, C# and C programming exercises. The system contains more than fifty Java and C# programming exercises which are used in three software development subjects at QUT. Additional exercises can be added through the author mode, Section 2.3. Students are presented with program templates as web pages. They complete an exercise and submit their answer to the server for compilation. The executable format of the exercise is downloaded and run on the student's own machine. With Java programming exercises, the resulting .class file of the exercise is packed together with other necessary libraries in a Java Archive (JAR) file; .exe files are used for C and C# exercises.

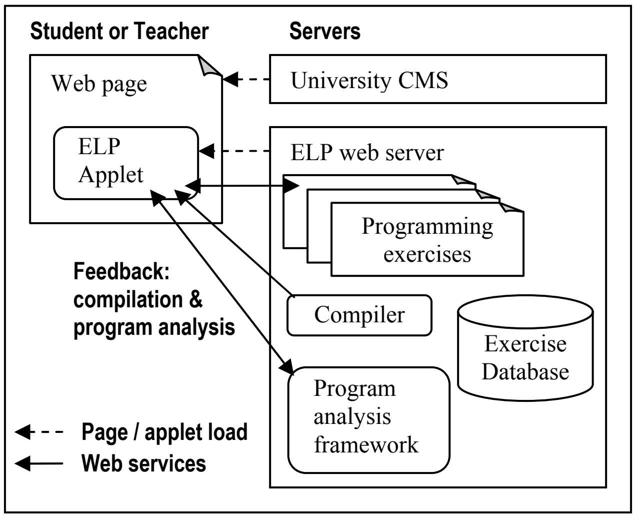


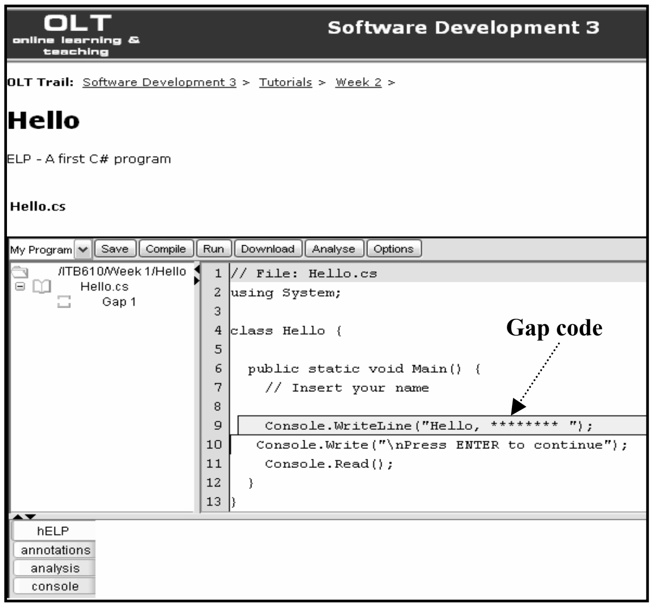
Figure 1. ELP Architecture

2.2 Student View of the ELP System

At QUT, students access the ELP through a local CMS, called Online Learning and Teaching (OLT) as shown in Figure 2. Programming exercises are embedded in the OLT's web pages. Each exercise comprises a separate Java applet which implements the ELP. The students log in and are authenticated for the subject by the ELP applet. Any contextual or support information for an exercise is contained in OLT pages. Thus ELP is used for programming activity only. This is fundamental in the design of the system as it means that the ELP can readily be integrated into any existing CMS.

An ELP exercise consists of a program with some missing lines of code which are required to be completed by the student. The missing lines (gaps) are writable fields in the ELP editor while the enclosing code is non-editable. The gaps usually contain helpful hints describing the missing code. A gap can be as small as an expression, or as large as a complete program. With different gap sizes, the system not only focuses on a small code segment but also fosters student's algorithm development skills. Exercises may contain more than one gap and may even incorporate multiple classes and multiple file programs.

The student completes a program by filling in all the writable fields of the exercise template. The exercise is sent to the server for compilation and the results are displayed on the console pane below the program text. If there are no errors, the student may download and run their executable. Alternatively they can save exercises locally and work off-line. Model solutions are released through the ELP at the lecturer's convenience. Additionally, the student can annotate their solutions with queries and at a later stage, a tutor provides an explanation by editing the annotations.



Hello

ELP - A first C# program

Hello.cs

```
// File: Hello.cs
using System;

class Hello {

  public static void Main() {
    // Insert your name

    Console.WriteLine("Hello, ******** ");
   Console.Write("\nPress ENTER to continue");
    Console.Read();
  }
}
```

Figure 2. An OLT web page with the ELP student applet embedded

2.3 Lecturer View of the ELP System

When a lecturer logs into the ELP, they are presented with an enhanced view of the editor applet with increased functionality for creation and management of exercises. The editor supports two modes: student mode and author mode. In author mode, programs can be fully developed in the editor or uploaded, with gaps being added later. They must designate the language of an exercise and set the release date of its sample solutions. Gaps can be created and manipulated. Exercises can be viewed either in hint view or solution view. In hint view, all gaps are occupied with helpful comments while in solution view the gaps are filled with examplar code. Lecturers can switch to student mode to view and test the exercise as a student. This rich authoring environment is the key to ELP's uptake among teaching staff.

3. THE ANALYSIS FRAMEWORK

This section gives an overview of the program analysis framework and then describes in detail each of its components.

3.1 Framework Overview

The program analysis framework analyzes students' programs and provides feedback on the quality and correctness of their solutions. It can also be used to assist instructors in the marking task. The framework is configurable and extensible but the novel aspect is its ability to analyze fill in the gap exercises. The framework can perform static analysis [10] and dynamic analysis of programs as shown in Figure 3. Static analysis is the process of analyzing source code without executing it for the purpose of evaluating the quality of students' programs. It comprises two components: software engineering metrics analysis and structural similarity analysis. Dynamic analysis involves executing a student's solution through a set of test data to estimate correctness by using black box and white box tests.

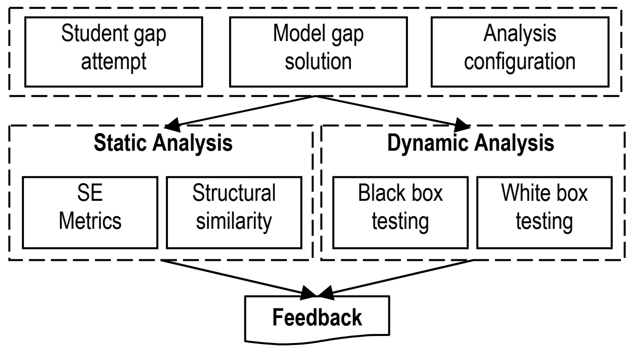


Figure 3. An overview of the program analysis framework

3.2 The Static Analysis

Static analysis is provided as a set of functions so that instructors can specify which analyses should be carried out for each gap in an exercise. Thus the framework is configurable. These functions make use of dynamic loading at run time so that other functions can be easily plugged in, if required. They are grouped into two distinct categories: software engineering metrics analysis and structural similarity analysis, described in Sections 3.2.1 and 3.2.2 respectively.

Only well formed gaps are analyzed by the static analysis to ensure that there is enough information about the context. Examples of well formed gaps are a statement or block of statements, a method or a complete class. Since only small programs are analyzed, our main conjecture is that a program's structure reflects its quality. As a result of that, the static analysis only focuses on the structure and quality of code.

3.2.1 Software Engineering Metrics Analysis

Software metrics is a well-known quantitative approach used to measure software complexity. This analysis is based on the software complexity metric and good programming practice guide lines. Cyclomatic complexity, which measures the number of linearly-independent paths through a program module is adopted because it provides useful information about the structure of a program. Currently, there are eight functions provided in this analysis which are showed in Table 1.

Table 1. Functions provided

Check	Description
Program Statistics	Count the total number of variables, statements and expressions in a gap.
Cyclomatic Complexity	Count the number of logic decisions in a gap.
Unused Parameters	Check if there are any unused parameters in a method.
Redundant Logic Expression	Detect redundant logical expressions e.g. "x==true".
Unused Variables	Check if there are any unreferenced variables.
Magic Numbers	Ensure student solutions do not have hard coded numbers or string literals.
Access Modifiers	Ensure variables and methods have the correct modifiers.
Switch Statements	Ensure that all switch statements have a default case and that each case has a break.

3.2.2 Structural Similarity Analysis

The purpose of this analysis is to refine the result of the software engineering metrics analysis and to check how the structure of the student solution compares with model solutions. In the analysis, the student solution and a model solution are both transformed into an abstract pseudocode form which represents just the algorithmic structure of the programs. The abstract representations of the student solution and model solution are compared to identify differences. Feedback to both students and instructors indicates the similarity of the student and model solutions. The techniques that are adopted in this analysis only work for fairly simple introductory programs.

By comparing student solutions with model solutions, the framework is able to identify high complexity areas in the student code, such as lengthy methods. Unmatched areas between student solutions and model solutions can be used to provide better feedback to students if their solutions ultimately result in an incorrect output in the dynamic analysis stage. Thus structural similarity analysis closes the gap between static analysis and dynamic analysis which exists in earlier related research.

3.3 The Dynamic Analysis

Black box and white box tests are incorporated to verify the correctness of students' solutions and detect at which gap errors occur and feedback can then be provided to the student. The implementations of these tests are discussed in Section 3.3.1 and 3.3.2 respectively.

3.3.1 Black Box Testing

Black box testing is the process of testing a program against its specification with testers having no knowledge of the implementation of the program. It is designed to evaluate the correctness of students' solutions. Black box testing is carried out by executing students' programs through a set of test data; gap outputs are captured and sent back to the server to check for correctness. Special print statements are inserted at the start and end of each gap so that the individual gap outputs can be extracted and analyzed separately.

The most important feature of the black box testing is that teaching staff can specify what needs to match between the student's program output and the model solution output by providing a set of filter, matcher and normalizer functions. Thus the framework is configurable. Filters can be used to extract all the important keywords that are related to the correctness of a program. The normalizer allows comparison of the output of a student's solution with those of the model solution while ignoring insignificant differences such as spaces or tab. Currently, the system provides two filter, two matcher and two normalizer functions. Filter functions include number and keyword. Matcher functions are: exact match and match without regard to order. Space or tab and new line character are provided as normalizer functions. These filter, normalizer and matcher functions make use of dynamic loading at run time thus additional filters and matchers can plug-in the system later on – the framework is extensible.

3.3.2 White Box Testing

Knowledge of a program's implementation is required for white box testing. The main purpose of this test is to discover any possible logic errors which are not revealed in black box testing or to detect gaps that have logic errors which lead to incorrect outputs. White box testing is carried out by inserting a student's gap solutions, one at a time into a test harness program. The outputs of each gap are compared with the outputs produced by the corresponding gap solution.

The framework supports two types of test harness program: a default test harness which is the exercise model solution and a customized one which is provided by teaching staff. White box testing requires that each gap be wrapped in adaptor code to provide more information. We identified two types of gap behavior: gaps in which the states of variables are changed and gaps which only modify the output of the program. In order to accommodate these gap types, the framework incorporates three different types of adapter code each providing a mechanism to carry out white box testing. These mechanisms include: variable state dumps, program assertions and print. Variable state dumps and program assertions can be used to check the states of variables before and after gap execution while the print method is used to check those programs whose gaps only modify the output. The state dump mechanism provides a set of functions to record the type and value of a variable at runtime. The program assertion mechanism can be used to test certain conditions that need to be met either before or after gap execution. The print mechanism provides markers to distinguish the output produced by each gap from the program outputs. The gap outputs are then extracted and analyzed for changes.

4. EVALUATION

The ELP system is being continuously evaluated and improved to meet students' needs. Two evaluations were conducted to discover whether the system makes learning to program easier. The first evaluation was for a laboratory-based introductory Java class; the program analysis framework was not available at that time. Thirty first year students participated at the end of week 5 of their course. These students found themselves having difficulty with Java programming and some had not even been able to install the JDK at home. The students experienced the ELP system for two hours and then filled in an online feedback form. Overall, all the students enjoyed using ELP and agreed that the system makes compiling and writing a program so much easier with the gap type exercises and customized compilation error messages.

The second evaluation took place in a second year unit, Software Development 3. C# is used as the teaching language; the static analysis of the program analysis framework had been integrated in this evaluation. Students were required to do tutorial exercises on the first 5 weeks of the unit using the ELP system. Evaluation forms were handed out at the end of week 5. Forty six students participated and 29 of them (more than 63%) gave positive feedback about the ELP system. Sixty percent of the students agreed that fill in the gap programming exercises are a good way to learn to program for "not so good" programmers. Most students acknowledged that the anytime, anywhere characteristic of the system is the most useful feature since many of them do not have visual studio at home or have trouble in using it.

Students were also required to comment on the program analysis framework. Sixty three percent of the students responded that it was useful in providing them feedback about how close their solution compare to the model solution and help them to think about other alternative solutions when doing the exercise. However, the main limitation of the structural similarity of the program analysis framework is that only limited numbers of model solutions of an exercise are available. The following table summarizes the second evaluation result.

Table 2. The second ELP evaluation results

The ELP		**Program Analysis Framework**		
Positive	Negative	Positive	Negative	Not used
63%	37%	63%	17%	20%

5. RELATED WORK

Systems that have been developed to help novice programmers can be grouped into four categories: programming environments, debugging aids, intelligent tutoring systems and intelligent programming environments [4]. Programming environments, especially web based are the main focus in this paper. These systems allow students to experiment with specific features of programming languages and are used in program construction, compilation, testing and debugging. CodeSaw [6], InSTEP [8], WenToTeach [2], CourseMaster [3], WWW for C++ [5] are similar to the ELP system.

CodeSaw is an online digital workspace which is designed to provide students with a dynamic environment to view, compile and run example programs without the need to install a program development environment or having to type in code. Currently CodeSaw supports for Java, C, C++ and other programming languages.

InSTEP is designed to help students learn looping in C, C++ and Java. The system compares students' program outputs and the expected outputs to check for correctness. If the outputs are not the same, they are first analyzed for a set of common errors before analyzing the solution code itself. Students are provided with hints on how to fix the problem.

WebToTeach is an automatic homework checking tool for computer science classes. It supports Java, C, C++ and other programming languages. The system 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 provided with immediate feedback.

CourseMaster is designed for delivering course based programming. It provides functions for automatic assessment, administration of marks, solutions, course materials and detects plagiarism. A student is able to develop a program, submit it to the server for marking or evaluation and get instant feedback.

Hitz and Kogeler developed a web based interactive C++ to help first year business informatics students. It supports fill in the gap programming exercises. Students are able to edit, compile and run their programs. Compilation error messages are displayed to students as close as possible to line where the error occurs.

Table 3. System comparison

	Fill in the gap	Web based	Configurable exercises	Customized compilation messages	Annotation feedback	Static Analysis	Dynamic Analysis
CodeSaw		✓					
InSTEP		✓				✓	✓
WebToTeach	✓	✓					✓
CourseMaster		✓				✓	✓
WWW for C++	✓	✓					
ELP	✓	✓	✓	✓	✓	✓	✓

6. CONCLUSION AND FUTURE WORK

The ELP system offers a useful multi-language, web based tool for teaching novice programmers. The strength of the 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 a compiler or learning how to use a program editor. It supports "fill in the gap" style exercises, which reduce the complexity for students in writing their own programs. Most importantly, the system enables the smooth integration of lecture notes, tutorial and practical exercises.

The program analysis framework provides static and dynamic analyses for students to check the quality and correctness of their solutions, providing hints about how to fix a problem and alternative solutions. In addition, it also assists teaching staff in the marking process. The ability to analyze fill in the gap programming exercises and the system's configurability and extensibility are key features of the framework. In the future, the ELP system will be able to provide or suggest exercises for students adaptively. ELP is available for trial from http://elp.fit.qut.edu.au/ and in the future an open source distribution will be made publicly available.

7. REFERENCES

[1] Anderson, J.R. and Skwarecki, E., The automated tutoring of introductory computer programming. *Communications of the ACM, 29*, 9 (September 1986), 842-849.

[2] Arnow, D. and Barshay, O. WebToTeach: An Interactive Focused Programming Exercise System. In *Proceedings of the twenty-ninth ASEE/IEEE Frontiers in Education* (San Juan, Puerto Rico, 1999). IEEE Press, 1999, 12a9/39 - 12a9/44.

[3] CourseMaster. Retrieved April, 2000, from http://www.cs.nott.ac.uk/CourseMaster/cm_com/index.html.

[4] Deek, F.P. and McHugh, J.A., A survey and critical analysis of Tools for Learning Programming. *Computer Science Education, 8*, 2 (August 1998), 130-178.

[5] Hitz, M. and Kogeler, S. Teaching C++ on the WWW. In *Proceedings of the 2nd conference on Integrating Technology into Computer Science Education* (Uppsala, Sweden, 1997). ACM Press, 1997, 11-13.

[6] Liqwid Krystal, CodeSaw. Retrieved June, 2002, from www.codesaw.com.

[7] Morrison, M. and Newman, T.S. A study of the impact on student background and preparedness on outcomes in CS1. In *Proceedings of the thirty-second SIGCSE Technical Symposium on Computer Science Education* (Charlotte, North Carolina, US, 2001). ACM Press, 2001, 179-183.

[8] Odekirk-Hash, E. Providing Automatic Feedback To Novice Programmers. MA Thesis, University of Utah, Utah, 2001.

[9] Truong, N., Bancroft, P., and Roe, P. A Web Based Environment for Learning to Program. In *Proceedings of the twenty-sixth Australasian Computer Science Conference (ACSC2003)* (Adelaide, Australia, 2003). ACM Press, 2003, 255-264.

[10] Truong, N., Roe, P., and Bancroft, P. Static Analysis of Students' Java Programs. In *Proceedings of the Sixth Australasian Computing Education (ACE2004)* (Dunedin, New Zealand, 2004). ACM Press, 2004, 317-325.