Automated Assessment and Experiences of Teaching Programming

COLIN A. HIGGINS, GEOFFREY GRAY,
PAVLOS SYMEONIDIS, ATHANASIOS TSINTSIFAS
The University of Nottingham, Nottingham,UK

This article reports on the design, implementation, and usage of the CourseMarker (formerly known as CourseMaster) courseware Computer Based Assessment (CBA) system at the University of Nottingham. Students use CourseMarker to solve (programming) exercises and to submit their solutions. CourseMarker returns immediate results and feedback to the students. Educators author a variety of exercises that benefit the students while offering practical benefits. To date, both educators and students have been hampered by CBA software that has been constructed to assess text-based or multiple-choice answers only. Although there exist a few CBA systems with some capability to automatically assess programming coursework, none assess Java programs and none are as flexible, architecture-neutral, robust, or secure as the CourseMarker CBA system.

Categories and Subject Descriptors: K.3.1 [**Computers and Education**]: Computer Uses in Education

General Terms: Human Factors, Measurement

Additional Key Words and Phrases**:** Computer-based assessment, online assessment, formative and summative assessment

1. INTRODUCTION

1.1 Motivation

Assessment and feedback are key aspects of the educational process [Benford et al. 1994]. Not only do they benefit students, but also assist lecturers in recognizing course failings. This is particularly relevant when dealing with practical topics such as computer programming that cannot be learnt by heart; topics which include skills that can only be developed through constant practice. However, computer programs do not lend themselves easily to assessment by traditional methods such as marking printed solutions by hand [Arnow and Barshay 1999]. It is difficult to visually inspect program source code and determine whether it is syntactically and dynamically correct, let alone whether it will meet a given specification. Also, program development is rarely completed satisfactorily on the first attempt; it is a process of revision and refinement that consists of many levels of development and testing.

This process can be assisted by relevant feedback, but the difficulties involved in marking solutions efficiently by hand make it infeasible to do so, thus significantly reducing opportunities for feedback. The problems in marking are intensified when large

This research was supported by the Teaching and Learning Technology Programme, UK Government.
Authors' address: Learning Technology Research Group (LTRG), School of Computer Science and Information Technology, The University of Nottingham, UK; email: ltrg@cs.nott.ac.uk

numbers of students are taken into account, and despite a recent worldwide decline in numbers, large class sizes are still very common. Postgraduate students are often enrolled to assist in marking and feedback, but the turn-around is rarely as fast as the students would like, merely reducing rather than eliminating the problem. Marking by hand also introduces the problem of inconsistency. Markers, however experienced, always have their own interpretation of the marking scheme, which, depending on the marker, often results in the same solution getting different grades.

These problems led the authors to look for a new solution, which first emerged in the form of Ceilidh [Foxley et al. 1999], and then in its successor, CourseMarker [Foxley et al. 2001].

1.2 Background

Ceilidh was first developed in 1988, when it was found that shell scripts written to assist in submitting and marking students' programming exercise solutions could be combined into a more coherent and useful system. Ceilidh was based on an automatic assessment mechanism that could test and mark programs from several perspectives: check a program's dynamic correctness (i.e., see empirically if it met the specification) and the programming style and layout (i.e., the elegance of the solution), among others. The concept behind the automated assessment mechanism came from industry, where automated manufacturing and testing have long been standard.

The first version of Ceilidh was created to support a C programming course, but its extensible nature meant that other marking tools were written very quickly to allow it to mark everything from artificial intelligence programming in Prolog to Unix shell scripts. Copies of Ceilidh were taken by over 300 establishments worldwide and installed at many of them around the globe. Ceilidh, however, had several restrictions. It had no network support; students had to log onto clients residing on the same machine as the server, and lacked full X-Windows and/or PC-Windows graphical interfaces. In addition, the incorporation of enhancements over the years meant that it had grown uncontrollably and had become difficult to understand and maintain. Finally, it only ran on Unix platforms.

CourseMarker (formerly CourseMaster) was created as its replacement [Foxley et al. 2001]. It improves over Ceilidh in many ways: first, it is fully networked, allowing clients to be installed separately from servers; it has a window interface, which significantly simplifies the student's view of the system; and was developed in Java, which means it is portable and runs on Unix and Windows platforms. It was first used in September 1998; development is ongoing, with only a few minor fixes ever required.

1.3 Other Related Systems

Several other computer-based assessment systems exist. Most were developed after the release of CourseMarker, so limitations were not taken into account during its design. Most of these systems are also only semi-automatic regarding marking programs, and in this respect do not have the full capability of CourseMarker.

1.3.1 *BOSS.* BOSS [Luck and Joy 1999] is a system that caters for the submission and semi-automated testing of student coursework. BOSS can run the solutions against set test data, but only returns a mark of either 100% or 0%, which is returned with semi-automatic feedback to assist the student in correcting his or her program. It has a only limited mechanism for automated marking, as the process is based on the results of the automatic tests.

The real aim of BOSS is not to provide automatic assessment, but rather to assist the lecturer in obtaining a higher degree of accuracy and consistency in their marking. BOSS also provides extensive administrative and archiving functionality.

1.3.2 *CodeLab.* CodeLab, formerly called WebToTeach [Arnow and Barshey 1999], is a web-based system in which the students submit their answers via a software client. One of the major features of the system is that CodeLab's exercises tend to focus on short answers to questions, which develop as the student progresses. The exercises aim to cover all concepts within each technical topic (such as array declaration and assignment) before allowing students to move on.

1.3.3 *ASSYST.* The ASSYST system is a semi-automated assessment system that uses humans to mark C and Ada programs. The semi-automated approach offers the advantage of using humans where automation is not accurate or possible (e.g., for the evaluation of comments inside student code). Understanding comments is still beyond the ability of the most advanced artificial intelligence techniques [Jackson and Usher 1997].

However, the semi-automatic nature of this system, combined with the method of submitting solutions by e-mail, excludes the return of instant feedback to the students. It is argued that this is a critical omission for a CBA system.

1.3.4 *TRAKLA.* The TRAKLA CBA system helps in teaching data structures and algorithms by presenting individually tailored exercises of algorithm simulations to the students [Hyvönen and Malmi 1993]. It does not assess programming coursework in any computer language as such. Hyvönen and Malmi's work has shown that the development of an automatic assessment system for teaching data structures and algorithms is feasible and useful.

TRAKLA automatically assesses student solutions and presents them with feedback via email. It is argued that e-mail-based feedback is neither instant nor immediate, although certainly much better than no feedback at all.

1.3.5 *RoboProf.* RoboProf is a web-based CBA system that presents course notes and exercises to the students, accepts student submissions, tests submissions against test data, provides feedback to the students and archives the results [Daly 1999].

The feedback provided to the students by RoboProf is immediate but minimal. The feedback is mostly encompassed by correct/incorrect statements on the output of student code. The amount of feedback regarding the enhancement of student learning is considered limited.

RoboProf embeds a modular marking engine; but the current exercises are marked almost exclusively on output. RoboProf lacks marking tools that test and measure student code attributes such as size, speed, indentation, and commenting.

2. COURSEMARKER ARCHITECTURE

2.1 Overview

While Ceilidh became a collection of extensions based on user suggestions received since its inception, the creation of CourseMarker was an opportunity to effectively re-engineer from scratch. Knowledge of object-oriented methods, frameworks, and design patterns were used in the design of CourseMarker. The benefits obtained from this include extensibility and maintainability, to a degree that Ceilidh could not aspire to. In an effort to reorganize Ceilidh's functionality in a more flexible way, the authors have identified dependencies, commonalities, and variations among Ceilidh's tools. All commonalities were abstracted into class hierarchies. Additionally, explicit points of extension and parameterization were defined. Finally, dependencies were decreased by separating the

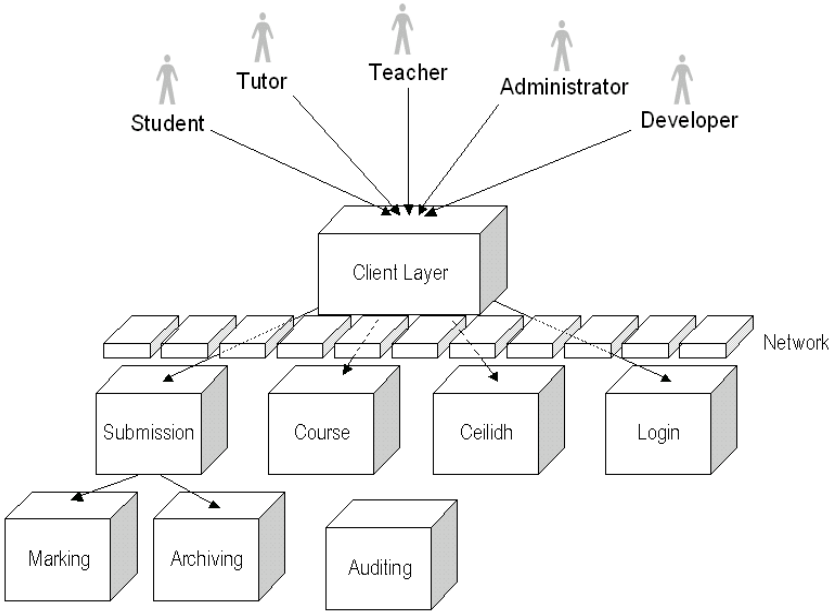


Fig. 1. A high-level view of CourseMarker's subsystems.

various responsibilities into seven logical parts (subsystems). The Marking Server is one of those subsystems; CourseMarker's subsystems are shown in Figure 1.

The initial architectural design of any system should take the programming language that will be used to implement it into consideration. We decided to develop CourseMarker using Java in order to satisfy the variety of deployment requirements made by academic institutions [Foxley et al. 2000]. Hence CourseMarker's design took into account the attributes offered to the developer by the Java programming language. Each subsystem has different responsibilities (described below) for the operation CourseMarker:

Archiving subsystem:. The archiving subsystem stores the students' marks and keeps records of the programs they submitted; the subsystem also keeps the students' transcripts, in case they are needed to settle disputes or check for plagiarism, as well as for educational research.

Audit subsystem:. This subsystem is responsible for logging. It provides the other servers with the mechanism to create their own logs, to track operations like user logons and submissions. This allows administrators to oversee the system and to assist if anything goes wrong.

Course subsystem:. This subsystem stores (and is responsible for) all files contained in a CourseMarker exercise. It also contains classes that provide a representation of the different types of files that could be encountered by the system (e.g., Jjava or C++ files). It is used by most of the other servers when trying to extract information from the files used by CourseMarker.

Login subsystem:. This subsystem performs the user authentication and the login process. A list of users is created for each CourseMarker module. The system can either use their current system password or, if required, passwords can be individually assigned

and validated by CourseMarker. If the login process is successful then the client GUI is loaded, but only after it has retrieved information about the courses that the student is taking.

Marking subsystem:. This subsystem will be discussed in detail in Section 3; it receives the submitted program, marks it, and returns the result to the user via the GUI. It then passes the program and a completed marking receipt file to the archiving server.

Submission subsystem:. This subsystem is in charge of the calls to the other servers. The submission server instructs the marking subsystem to start marking. When marking is finished, the mark is returned to the archiving subsystem. Each exercise has a property that decides whether marking or archiving will be supported. The server first retrieves the information about the specified exercise (e.g., the number of submissions available and the weight of the exercise). It then checks how many times the student in question has submitted the exercise and whether he or she was given any extra submissions. If the student is allowed to submit then it instructs the marking subsystem to perform marking.

Ceilidh subsystem:. The Ceilidh subsystem, named in honor of the old Ceilidh system, allows intercommunication of all CourseMarker subsystems. It also handles the communication of all subsystems with Java's RMI naming registry mechanism. Before the CourseMarker client commences initialization, it queries the Ceilidh subsystem to ensure that the rest of CourseMarker subsystems are already up and running.

The structure of the course information store in Ceilidh was not altered. CourseMarker can support a number of simultaneous courses; each course consists of a number of units representing chapters or concepts. A summary, course notes, and a number of exercises can be stored in each unit.

The most notable visible enhancement of CourseMarker was the development of a graphical user interface client for the students; a description of the web-based administration system available to users with different roles than "student" can be found in Foxley et al. [2001a].

2.2 Student Interface

The interface students were required to use in Ceilidh was purely text based; it was also difficult and dangerous to use because it had few user input error-handling safeguards. This situation improved greatly with the release of CourseMarker. The new Java Swing graphical user interface (GUI) gave the students not only enhanced functionality but also an easier to use format (see Figure 2). The interface is split into three areas. A row of function buttons that initiate a variety of actions appear along the top. The area on the left side is used as the course view frame. It displays the courses students are registered for and the units and exercises available for viewing and submission. Information is displayed on the right side, which can consist of course notes, question specifications, or even test data.

By prompting and confirming before initiating important tasks, like the submission of a solution or the set-up of an exercise (which would overwrite work already completed), the interface prevents students from performing accidental operations. Of greater significance to the students is the ability to access a graphical representation of their marks (see Figure 3). The collapsible tree structure design of the mark's representation allows students to see the sections in which they received lower marks. The course

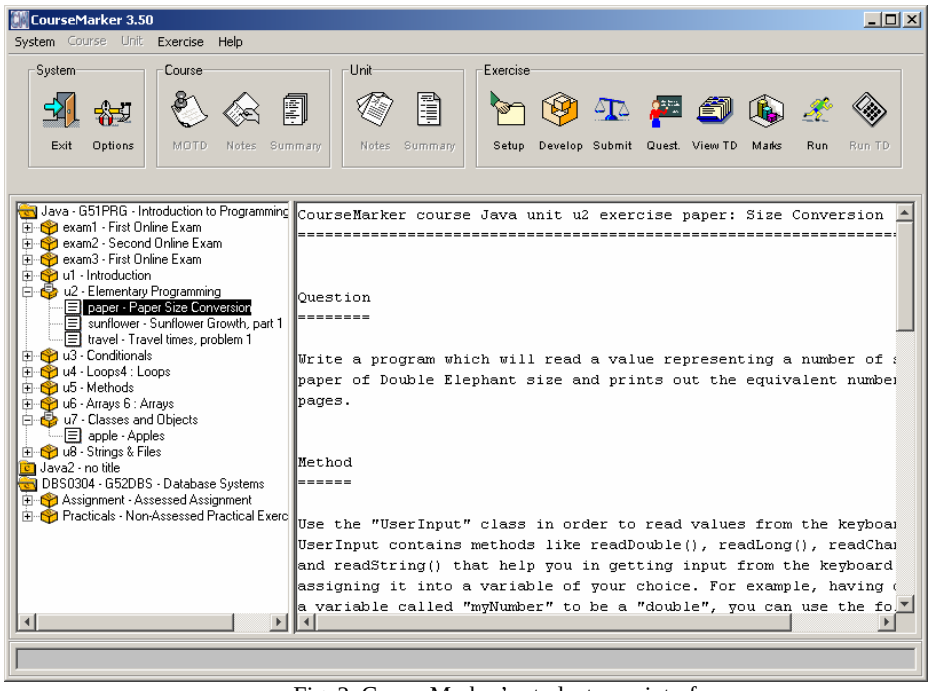


Fig. 2. CourseMarker's student user interface.

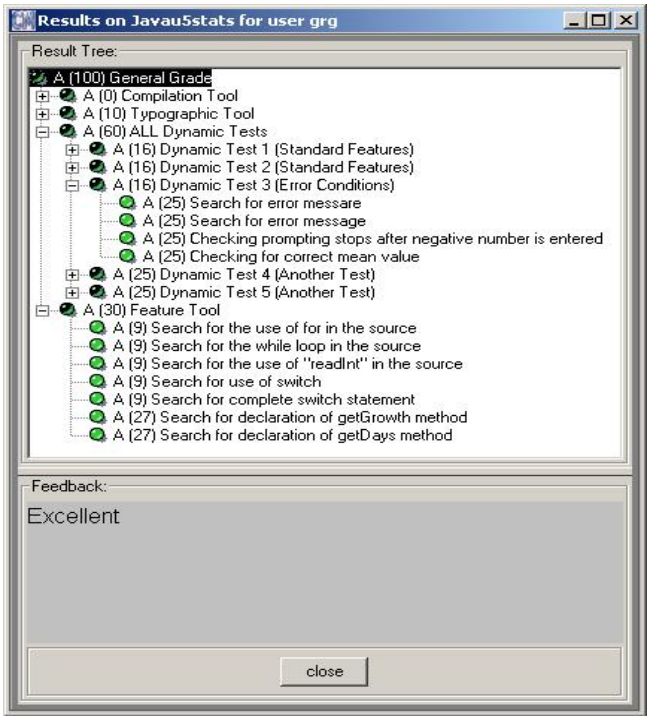


Fig. 3. The results screen.

designer can provide students with feedback as to why their marks are lower in a particular section.

3. MARKING SYSTEM REVIEW

3.1 Introduction

Initially Ceilidh had an open, maintainable, and upgradeable marking system [Benford et al. 1993]; but over the years a number of shortcomings were identified [Foxley et al. 1999]:

- Its marking system had internal dependencies and all changes propagated to many levels.
- The student feedback mechanism was limited, due to a limited expressiveness capability of the exercises.
- The marking system's architecture inflicted a penalty on performance, maintenance, and usability.

The authors' desire for a reliable, coherent, secure, feedback-rich, and extensible system led to the following architectural requirements:

(1) *Reliability and consistency* - to be a stable, crash-resistant, and always-on system.
(2) *Scalability* - to perform efficiently with large classes of students.
(3) *Maintainability* - to allow extensions and configurations.
(4) *Feedback-richness* - to provide expressive feedback to the students.

3.2 The Marking Process

Currently, CourseMarker can mark Java and C++ programs, but it also provides additional support for object-oriented design, graphical exercises, flowcharts, and logic circuits. When students login to CourseMarker, they must do the following:

- First, students must read the exercise question. The question is carefully worded so that it accurately describes the specification that a student's program has to follow, as well as the format of any input and output (keyboard/screen/files).
- Second, students need to obtain a skeleton solution along with any header files and/or testing tools that the teacher has provided. The skeleton may be totally empty; but it could also contain a semi-functional or a fully functional program, or a fully functional but flawed program to be debugged.
- Third, students follow a develop-compile-run cycle. They have to test their programs first before submitting them to CourseMarker.
- Finally, the students submit their solutions to CourseMarker for assessment. CourseMarker passes the programs to its marking subsystem (MS) for analysis. After the marking system has processed the students' solutions, the students receive a grade and feedback.
- CourseMarker permits the students to resubmit their solutions as many times as the exercise developer allows (usually between two and five at Nottingham).

3.3 Available Marking Tools

CourseMarker's marking subsystem assesses the students' submissions by running their solutions against a set of metric tools. These tools perform a variety of quality checks and provide feedback, which is then returned to the students. Table I illustrates the currently available tools developed for use in CourseMarker.

Table I. CourseMarker's Marking Tools

Tool	Description
Typographic tool	Checks student solutions for typographic layout.
Dynamic tool	Runs student solutions against test data.
Feature tool	Inspects student solutions for features specific to the exercise being marked.
Flowchart tool	Converts student flowcharts to BASIC programs, that can subsequently be marked with the dynamic tool.
Object-oriented tool	Checks student object-oriented diagrams for object-oriented analysis and design.
CircuitSim tool	Simulates logic circuits of all student solutions.

When the marking subsystem runs the typography tool, the student's solution (program) is checked for layout, indentation, choice and length of identifiers, and use of comments, as well as other tests. These typography parameters can be customized on a per exercise basis.

When the marking subsystem runs the dynamic tool, the student's program is run several times with various sets of test data by driving the student program with predetermined alphanumeric input strings. The dynamic tool verifies the correctness of the solution and its compliance with the specification. The marking subsystem gives the author of the exercise a wide range of options for the flow and control of dynamic tests. Output is checked by matching to regular expressions that define the desired output

When the marking subsystem runs the feature tool, the student's source code is checked for any special features (these are usually exercise-dependent). For example, an exercise that teaches the difference between an "if-then-else" statement and a "switch-case" statement would typically include such a feature test to ensure the appropriate use of each construct.

When the marking subsystem runs the flowchart tool, the student's flowchart diagram is converted into BASIC code, which is then fed to the dynamic tool. Subsequently, the dynamic tool runs the BASIC program and returns marks and feedback.

When the marking subsystem runs the object-oriented tool, the student's solution is tested for completeness, correctness, and accuracy. The student's design is also tested as to whether correct relationships exist between the various components. The student is penalized for the use of unnecessary classes that clutter the design.

When the marking subsystem runs the logical circuit tool, the student's solution will be simulated by "running" the circuit. Any faults that may be identified (for example, incorrect wiring between components or unnecessary use of components) will result in a lower mark.

The flowchart tool, object-oriented tool, and CircuitSim tool are run by an extension of CourseMarker's marking subsystem, called DATsys. DATsys is an object-oriented framework that caters for the design, authoring, and automated assessment of student diagram-based coursework. DATsys enables students to draw diagrams (e.g., flowcharts and electronic circuit diagrams) in its editor and have them marked against a given set of criteria [Higgins et al. 2002].

3.4 The Marking Mechanism

CourseMarker provides an expressive and extensive marking mechanism that is customized in a file called `mark.java` (see the Appendix). The `mark.java` file

contains Java code written by the exercise developer and can access CourseMarker's state or call external tools to help with the assessment. The `mark.java` files are exercise-specific and are loaded dynamically at runtime by the marking system. Dynamic loading is implemented via Java's class-loading mechanism. The `mark.java` files are mandatory.

For instance, a `mark.java` file for a programming exercise might first instruct the marking system to run an external tool in order to compile the student's code and then other internal CourseMarker tools to perform some typographic, feature, and dynamic execution tests. A `mark.java` file may also contain custom feedback ranges and feedback messages, which are associations between numeric marks and alphanumeric messages. For example, a student score of 70 -100 could be defined as Excellent (or A); 50 - 69 could be defined as Good (or B), and so on.

3.5 Feedback Detail and Grading Styles

CourseMarker's marking system provides mechanisms for on-demand feedback. When the marking process is complete, it presents the students with their results. These details are displayed using a "tree" GUI widget. Every CourseMarker marking tool returns feedback assembled in this tree to the student. For example, the typographic tool might inform the student that his solution has indentation problems or that the length of the identifiers is too long. At the discretion of the instructor, students may also receive comments on how to improve their solutions and links to further reading material.

However, a delicate balance must be maintained, which requires the attention of the author of the exercise: in their work the majority of students want the system to pinpoint the exact problem and the exact location in which it occurs, but experience has shown that ultra-detailed feedback can be detrimental to their learning experience. As a result, CourseMarker supports regulating the amount of feedback so that it matches the needs of the classroom. Consequently, the author of the exercise must judge how much feedback should be given, so that students will still have to think and try to determine how their work can be improved. Future versions of the CourseMarker system may include an intelligent mechanism that would decide how much feedback to provide the students, depending on their past grades and performance in the course [Yong and Higgins 2003].

Finally, the distribution of marks can be customized (e.g., whether a mark of 70 should be treated as an A or a B) as well as the association between numeric values, letters, colors, and shapes. The `mark.scale` file is used for this purpose (see the Appendix).

3.6 Exercise Properties

Exercise settings are parameterized using a file called `properties.txt`. Properties are used to specify behavioral aspects of an exercise's assessment in a controlled environment. Such properties might be the exercise's filename, the maximum number of allowable submissions, and whether the exercise is open or closed to students. Other properties might include the maximum size of a solution's output and the maximum CPU utilization time that should be allocated to run the solution. There is additional parameterization to address the configuration of other types of assessments such as diagramming and generic courses.

3.7 Scaling Marks

The ability to scale marks may be a necessity for academic institutions, as they often have to comply with institutional policies on the distribution of final marks. The `mark.scale` files contain information on scaling and how it should be done, if

required. Scaling is usually applied at the exercise level, but can also be applied to a whole course. CourseMarker's web interface facilities allow teachers to see both scaled and unscaled marks.

3.8 Reliability

The marking system was designed with reliability in mind; it is an important factor in the design of CBA systems [Foxley et al. 2000]. The marking system allows many students to submit concurrently without any difficulties. Should the marking system malfunction, the CourseMarker servers continue to function, and the difficulties, if any, will not have an adverse effect on other student submissions that may be running concurrently. The difficulties may be due to corrupted/invalid exercise data, faulty execution of student programs with bugs, and problems with the marking subsystem itself.

Since CourseMarker is written in Java, Java's built-in exception mechanism is utilized. When an anomaly occurs, exceptions are thrown and then caught in specific code segments. Thus, CourseMarker's development process was simplified while its robustness increased. A further benefit in using Java's exception-handling mechanism is the increased readability of the code, which in turn avoids bugs.

3.8.1 *Corrupted/Invalid Exercise Data.* In the unfortunate event that an exercise being marked contains corrupted and/or missing files, CourseMarker's marking system will recover from such a situation. It will first try to override the attempted, malformed, customized behavior and use default parameters. If this fails, the specific test will be aborted. In all cases, the event will appear on CourseMarker's logs and an informative, administrative alert message will appear on the CourseMarker servers' console.

3.8.2 *Problematic Student Programs*. CourseMarker assists (inexperienced) students and helps them learn programming, making faulty/buggy student program submissions inevitable. The marking system times each student's submission, so the marking process can be aborted when execution time exceeds a defined threshold for that exercise. Problems include student programs entering endless loops, crashing, waiting for superfluous input, and so on. When students receive feedback, they are notified of any problems so that they can make changes and possibly submit again.

3.8.3 *Problems of the Marking System.* Due to the complexity of large systems such as CourseMarker, errors (bugs) are difficult to avoid. If an error does occur in the marking system, the current student submission is aborted and the system will not be allowed *to enter* an unstable state. No problem with the marking system will affect the stability of the CourseMarker servers in any way, or any other concurrent student submissions for that matter. Furthermore, the event will be logged and a notification will appear in the CourseMarker servers' console.

3.9 Coherency

CBA systems like CourseMarker should provide a coherent and fair approach to students [Foxley et al. 2001a], and hence the marking system was designed to provide a high level of support to all students. Obviously, students will always get the same mark for identical submissions, as the marking system's tests will give the same results. Note that when different students submit an identical solution it is evidence of plagiarism. A plagiarism-detection tool is available that will pick up such an event, and after scanning all of the student programs and performing a variety of tests, will report any similarity between submitted work and inform the students' tutors. However, other externally available tools are recommended to assist plagiarism detection, as some of these are more sophisticated and thorough than CourseMarker's plagiarism tools.

3.10 Security

Security is a very important factor in CBA systems [Foxley et al. 1998]. A poorly guarded and insecure system can potentially put at risk the marks, the submissions, and the course content and be open to malicious practice by dishonest students.

CourseMarker was designed with security in mind: it has a password encryption mechanism (which ensures that nobody can acquire a username/password pair by using TCP/IP sniffing techniques), and a session key security feature (which ensures that CourseMarker clients' communication with the CourseMarker servers originates from a validated and non-tampered-with client). These aspects will not be discussed further in this article; the security provisions for the marking system will be the focus of attention here.

The first security check is performed on the students' source code. If enabled by the exercise developer, this check can identify potential security risks, such as "*.*", "unlink," "delete," and the use of network sockets/RMI. What happens next is also in the control of the developer. Typically, if the tests are positive, the marking process is curtailed and the student in question is informed of the situation.

The second-level security check comes in the form of a wrapper module. When dynamic execution of student programs takes place, the marking system runs each student's program inside a wrapper (a protected sand-box). This behavior needs proper CourseMarker server configuration and is not turned on by default. When fully configured, the marking system uses the Unix/Linux SUID/GUID facility or the Windows 2000/XP "runas" command. The wrapper has very restricted privileges; the student program's access in the system is prohibited, except to the program's own, designated, temporary subdirectory inside the `markingArea` directory. This is where, upon submission, all student programs are copied and executed independently of each other. Since the program is run in a controlled manner, no damage can be inflicted upon the CourseMarker servers themselves, or on the rest of the files on the machine that runs CourseMarker.

3.10.1 *Feedback-Richness*. CourseMarker returns feedback upon submission. This is a fundamental feature of CBA systems that unambiguously contributes pedagogic benefits. The CBA community has often described these benefits in the past [Charman and Elmes 1998]. On-demand feedback allows students to improve their learning by helping them concentrate their efforts on specific segments and aspects of their (problematic) submissions.

3.10.2 *Extensibility and Customizability*. For a CBA system, extensibility is not usually considered a requirement. CBA systems are usually designed to adhere to their original specification only. However, the authors designed CourseMarker with extensibility in mind, so that additional features can be seamlessly bolted on at a later stage, along with a variety of new types of courses and exercises. The original functionality of the marking system can also be modified to suit new environments.

CourseMarker's marking process can be extended in two major ways: by customizing the exercises and the marking system itself. If a project type is already supported by CourseMarker (e.g., Java programming, object-oriented design), then all the customizations and extensions that an exercise developer may need can be done by either customizing the exercise's marking files and/or by customizing the mark.java file. However, if a radically new project type that is unsupported is required (e.g., marking musical notation), then the exercise creator has to customize the marking system itself.

Customizing exercises. When a new exercise is to be created (or an existing one is to be modified), most of the developer's time will be spent in writing the questions and

model solutions, as well as editing the exercise's marking files to make them CourseMarker compliant. Finally, the developer will want to test the exercise.

For the vast majority of exercises, all that is required is the editing of various text files associated with the aforementioned tasks. Given that CourseMarker's exercise files are all ASCII, their editing requires little more than a common text editor. The only exception is the mark.java file; but while it only requires a text editor in order to be edited, it requires a Java compiler to be compiled. The selection of metric tests to be performed on the students' programs, along with their parameters, only requires simple text input, and does not require any alterations or programming of code into the marking system. Techniques for authoring and modifying exercises are detailed in a previously written technical report [Symeonidis 1998].

Customizing the Marking Server. If new functionality (or a major alteration to the existing one) is required, it will have to be built around new marking tools. Typically, this requires design and implementation of new classes (or alterations of existing classes), depending on the severity of the modifications.

CourseMarker's internal state can be queried within the marking system, hence novel marking strategies can be authored that perform a variety of new tasks. For example, in order to establish whether a student is improving, a marking strategy may be devised that compares the student's current submission with all the work that he or she has submitted during the course. If there was no improvement, CourseMarker would suggest to the student that he or she seek advice from the tutor. More information on customizing the marking system can be found in Symeonidis [2001].

4. COURSEMARKER IN PRACTICE

4.1 Introduction

CourseMarker's ability to ease the burden on lecturers is wasted if the appropriate support infrastructure is missing. A clearly defined content structure for the course is required in order to set exercises appropriate to the students' current knowledge. The following section highlights how we at Nottingham plan the course and provide the students with the support required for the smooth running of a year-long programming module.

4.2 Course Content

Course content has not undergone any major changes since the release of CourseMarker although minor updates are implemented every year as necessary. A major change occurred the year of CourseMarker's release, when the computer science department at the University of Nottingham changed its curriculum to use Java, an object-oriented language, instead of C, a procedural language. The course could have been taught in a variety of ways, one of which was to explain classes and objects in the early stages and then continue with the procedural aspects of programming. However, since many of the students had never programmed before arriving at the university, an alternative approach was taken, which was to teach procedural programming first and then take an object-orientated approach.

The current course is spread over a full academic year consisting of two semesters. The first two-thirds of the first semester is spent teaching Java as if it were a procedural language. CourseMarker's ability to split courses into units means that each unit can be based solely around a new programming concept, from the very basic use of variables, to methods, strings, and file reading/writing. Students receive between one and four lectures for each unit, depending on its difficulty. They are given weekly exercises based on the current unit being taught.

By the end of the first semester, students have covered the basics of programming, such as variables, conditionals, loops, input/output, and arrays. Students are also given an introduction on the use and design of classes and objects. Comprehensive instruction in object-orientation is provided in the second semester, and object-oriented topics like inheritance and polymorphism are covered. In addition, students are required to create their first graphical user interface program in the second semester. Depending on time constraints, there are other topics that are also covered; including threaded programs and networking.

4.3 Exercises

Any number of exercises may be assigned each week. (At the University of Nottingham two exercises are normally considered a good balance between time and difficulty.) The types of question sets can also vary, as CourseMarker can provide several different types of exercises. The design of the marking server enables course administrators to insert new marking tools. CourseMarker is capable of marking most programming languages by simply modifying its configuration and not its source code. The only limitation is that all programs need to be command-line-driven.

Besides programming, CourseMarker can mark diagrams via the DATsys extension. CourseMarker can also mark multimodal questions, which include multiple choice and short-answer questions [Yong and Higgins 2003].

4.3.1 *Assessment.* The exact distribution of assessment types (traditional exams, weekly exercises, major exercises, online exams, multiple-choice questions, written reports, debugging exercises) and how much each contributes to the overall student grade has altered over the years, as teaching staff have gained experience and student expectations have altered. The system currently in place is described below.

Students are assessed using a mixture of weekly exercises and more "formal" assessments. Each week, two exercises are (usually) set that reflect the current concepts being taught. The first is an introductory exercise and the second is a longer and more advanced exercise, which involves implementing and combining knowledge from the previous weeks. Some exercises are based on debugging a faulty program. The Learning Technology Research group has published research on the importance of mastering debugging skills [Ahmadzadeh et al. 2005]. It has been observed that the majority of good debuggers are good programmers, while less than half of good programmers are good debuggers. The assessment tests the students' ability to meet the specification outlined in the question. CourseMarker checks the source code for the concepts that the students are currently learning and ensures that their program solutions are typographically correct.

There are four different types of "formal" assessment in the course at Nottingham. They are either more substantial pieces of work or done under exam conditions.

The first type is a variation on the weekly exercise concept, except students are given extra time to complete the task, due to its considerable size and complexity. In addition to writing a program, students are required to write a short report on their program explaining its functionality and implementation. The report is used to verify that the students understand the task and that the work is their own, as an outstanding program is rarely accompanied by a poor report. There are two of these assessments in the first semester.

The second type of formal assessment takes the form of a programming exam taken in "real time." It requires that students solve exercises (under time constraints) similar to the weekly ones, but that are slightly easier than the weekly ones because they are meant

to be completed within a lab session. These exercises are based on previous ones, so there is very little new material, if any, needed for the solution. These short exams do not take place on the regular CourseMarker server, but in a dedicated exam server. This allows the computer lab to disconnect from the outside world by disabling its network connection; access is also restricted to most software on the lab machines. Students can open CourseMarker and the Java editor only, thus significantly reducing any opportunities for cheating. As the class size is usually too large to cater for all the students simultaneously, the short exam is performed in two sittings, directly one after the other. When the first session has finished, students are sent out of the lab and out of the building via the back doors. Meanwhile, the second group is brought in through the front, preventing the two student groups from meeting and exchanging information.

The third formal assessment is in the form of multiple-choice questions (MCQs), which can all be answered reasonably quickly. The questions are mainly based on recently learned concepts, and (again) are taken under exam conditions.

Finally, the standard weekly exercises contribute toward raising the grade for the course to encourage students to attempt all the exercises.

4.3.2 *Weighting.* The ability to weight marks awarded is important. It would be wrong for an exercise that prints "Hello World!" to be worth as much as an online exam. CourseMarker enables the designer of an exercise to set its weight during its authoring process. As the weeks progress, the weights of the exercises increase along with the size and difficulty of the programs. Currently, the weekly exercises are worth only 5% of a 20-credit module and the weights per exercise refer to a proportion of that; the formal exercises are worth more; each MCQ assessment is worth 5% of the total.

Students attend three MCQ exams during the first semester; they are also given two online exams, with each one worth 5% of the total. There are two large weekly exercises with complementary reports worth 7.5% and 12.5%. There is a similar scheme for the second semester, in which there are fewer components but each one contributes a greater proportion to the grade. Hence students are able take on larger case studies and learn to produce bigger program solutions.

4.3.3 *Course Evolution.* Many changes have been made over the years to the way in which the course is assessed. Initially, a collection of weekly exercises accounted for the full total of the mark. This worked effectively but opened up the system to abuse through plagiarism. The course was then altered to incorporate a written exam at the end of the year on all concepts covered, especially the more advanced topics from the second semester. However, programming is a practical, hands-on subject and does not translate well to a paper-based exam [Mason and Woit 1998]. So the idea of an exam was abandoned in favor of more frequent tests. The weekly exercises were complimented by larger exercises (with reports) and multiple-choice questions worth a substantial proportion of the mark. This improved the way marks were obtained by the students. But it was important to see how well students performed under timed conditions and without their notes or help from lab demonstrators. Thus, the online exams were also introduced: marks awarded for the weekly exercises could be significantly reduced and students could then be encouraged to work together, minimizing the effects of plagiarism.

Changes to the way in which the weekly exercises are monitored have been made recently. For the last two years students have been using a modified Java compiler, which has been collecting their compilation errors and storing them in a MySQL database [Ahmadzadeh et al. 2005]. With this information it is possible to discern the mistakes that students regularly make and the stage of the course at which they were made, and thus alter the weekly questions appropriately. Currently, the simple introductory exercise

is to debug a partly written program. The database stores the kinds of bugs experienced by the students and their frequency. The details on the most commonly occurring bugs are then extracted from the database and similar errors are written into the debugging problem. Being able to quickly determine which parts of the course are causing the most difficulty to the current group of students allows the teacher to target revision lectures and materials appropriately -- a powerful improvement in the education of weaker students.

4.3.4 *Plagiarism.* Combating plagiarism is a major challenge for many educational establishments [Higgins et al. 2002; Culwin and Lancaster 2001]. Plagiarism is not limited to certain types of courses, it is a recurring problem faced by nearly all subject areas. However, the inbuilt reusability of modern programming languages allows students to easily and efficiently copy large sections of each other's code. Following a worrying recent trend, students seem less deterred by the penalties for plagiarism than previously. Hence Computer Science at the University of Nottingham decided to adopt an alternative strategy. Nottingham's programming course deters students from plagiarizing in the first place.

At the beginning of the course, students are informed that they are permitted to assist each other when working on a solution for any of the regular weekly exercises. But any large exercise must be solely the work of the individual student. Allowing students to work together has benefits, e.g., mutual support and motivation, but may make plagiarism more attractive. This problem is tackled by producing, at regular intervals, "similarity lists" of students with similar solutions by using external plagiarism software. These lists are then published on the web and students who have plagiarized are encouraged to admit having done so.

To counteract the effect that student collaboration might have on marks, the weight of the exercises is altered accordingly. The combined worth of all weekly exercises is reduced to 5% of the course total, whereas the larger exercises are worth at least 7.5% each. If two students appear on several "similarity lists," their work can be cross-checked to establish whether they are regularly (over) collaborating.

Over the years the public display of the lists has decreased the amount of plagiarism detected; the more frequently the lists appear, the greater the reduction.

4.4 Support

Support for students has significantly increased and improved over the years. So have the sources from which help can be obtained. Most of the assistance that students require is provided at the compulsory labs, which students attend for two hours per week. Support is also available outside regular lab times.

4.4.1 *Laboratory Sessions.* The labs are structured so that computers are grouped together and color-coded. Each student is given a group color at the beginning of the year. Students are assigned to groups arbitrarily, rather than allowing them to form groups on their own. Students are then required to sit in the assigned area of the lab for the duration of the course. A lab demonstrator, usually a PhD student with programming experience in the relevant language, is also designated for each color group. Each demonstrator has approximately 10 to 15 students in his or her group, allowing rapport between demonstrators and students to grow and enabling demonstrators to tailor their advice according to each student's learning style. Dealing with the same demonstrator for the duration of the course helps students gain the confidence necessary to ask questions. It also allows demonstrators to get to know the students and their strengths and weaknesses, providing another possible check on plagiarism. Although a

demonstrator is designated for each group, students are encouraged to work together both in and out of labs, making it more likely that they will work through solutions themselves rather than seeking help. This also gives students invaluable teamwork experience.

The number of demonstrators has changed dramatically over the years. The first year CourseMarker was released there were only four demonstrators for the entire group of students (a ratio of about 35 to 1). This made it difficult to obtain help in a timely way. The number of demonstrators gradually increased to 14 (with the ratio improving only about 20 to 1 due to an increase in the number of students), but the demonstrators' knowledge of their domain was sometimes limited. Since then the number of demonstrators has been reduced to 10, with an additional two "floaters" who answer questions whenever a group seems to be struggling and the group's demonstrator is disproportionately busy (with decreasing student numbers, there is a very good current ratio of 12 to 1). To become a demonstrator, the candidate must prove competence in both the subject matter and teaching methods. The test for all candidates is to correct a faulty student solution and to explain why it is wrong.

The demonstrators' job continues after the lab. Demonstrators are also required to hold problem classes every week. Unlike tutorials, the structure of the classes is flexible and allows the students' current problems to be discussed and explained via examples. Although attendance is not compulsory, it is high, indicating that students themselves will actively seek help if the means to obtain it exists.

Alongside labs and problem classes, there is a help desk run by teaching and lab assistants about four days per week, which offers students one-to-one help on any course-related problem.

4.4.2 *Online Help.* A substantial amount of help is available online. First, there is the comprehensive course website that contains all the information needed for the students to complete the course. This includes everything from lab allocations to a timetable for deadlines to lecture slides. Weekly questions along with any skeleton files that may be provided by CourseMarker are also supplied, allowing students to develop their solutions at home without the need to use the computer room. The website also includes a collection of frequently asked questions (FAQs), covering topics such as the usage of CourseMarker, the way typography-marking is performed, and so on.

Additionally, there is an email address that forwards student questions to the course administrator. The queries range from problems with the system to question-specific clarifications. While it is sometimes abused by lazy students, it is also a valuable resource for those in charge of the course. This kind of feedback informs the course administrators as to whether the exercise questions are being clearly written or whether a change in style is required. It also reveals the areas in the course that students are having problems with, as this can be judged by the quantity of emails arriving in a given week on any particular topic.

Working alongside the administrator's email help system is another one, Question Master, which underwent preliminary trials in 2003-4 [Al-Yahya et al. 2005]. Question Master is a dialogue reuse system that allows for interaction and discussion between students and instructors. Machine learning and information retrieval techniques are implemented in Question Master's discussion forum. Thus the system is able to learn from past answers, and will attempt to answer questions correctly with no intervention from the instructor.

The bulletin board attached to the course allows students to post questions about the course and about the coursework. Although an experienced programmer monitors the

bulletin board, it gives students the chance to answer each other's questions, encouraging mutual support and increasing efficiency.

5. RESULTS

CourseMarker's Java course (referred to as PR1 in the first semester and PR2 in the second semester) has been taught since 1998. The course contains more than 35 exercises in total. CourseMarker's diagrammatics course, started in 2000, covers object-oriented design and digital circuits, and currently contains three exercises. The CourseMarker development team is busy adding more content across other graphical domains. CourseMarker's C++ course was completed in 2003. It has yet to be used in Nottingham, but has been supplied to other establishments that use CourseMarker. The latest addition to CourseMarker's list is a database systems course; it has made use of CourseMarker since September 2003, and the addition of new content is ongoing.

Since Ceilidh has such an extensive exercise base, it is easy to convert C, Z, and Prolog courses from Ceilidh to CourseMarker (a conversion tool has already been developed).

Data gathered during the six years that CourseMarker and Java ran at the University of Nottingham show that students became good programmers and achieved better marks as a result of CourseMarker's detailed on-demand feedback, coupled with multiple submissions. The authors also attribute this success to carefully designed exercises and improved support, both of which have evolved with experience. The vast majority of students (94%) have consistently improved their grades by utilizing the two extra submissions normally provided per exercise. The average improvement in marks from the first to third submission was 63%. Figure 4 highlights how the students improved their marks with each submission. The marks shown relate to five of the later exercises, where students should already have a working knowledge of the programming language in question. Students were not required to use all submissions, for example only 18 students used their 3rd submission for Exercise 8 part 3. By comparing the first submission and exercise averages, the benefits provided by multiple submissions are clear to all.

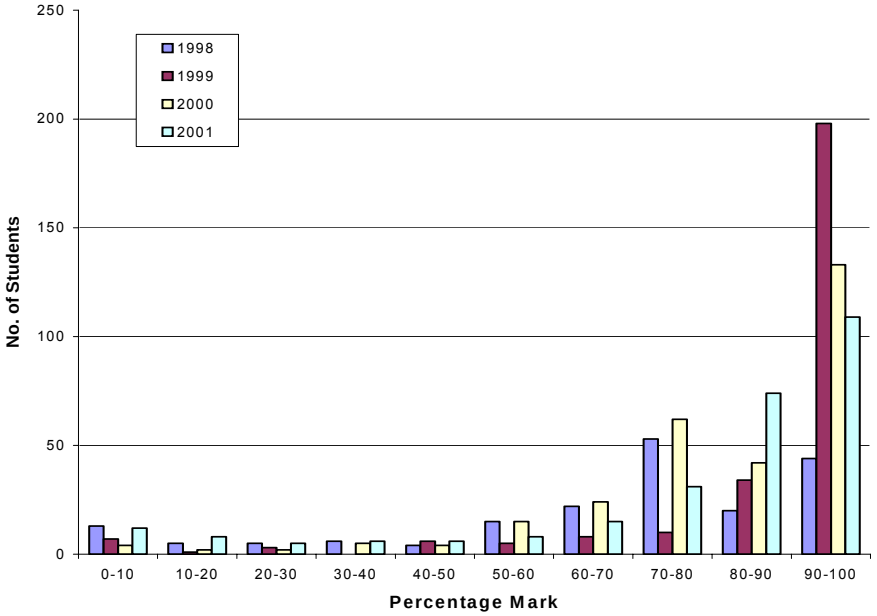
Over the years, various questionnaires have been given to the students in order to monitor progress and student attitudes to CourseMarker. 83% of students felt that the number of available submissions was enough to obtain good marks; 80% of students reported that the additional submissions motivated them into putting more effort to achieve better grades. The average grade for the PR1 course is 78% and for the PR2 course 74%. In both PR1 and PR2, the average grades have been decreasing over the years as more challenging exercises were authored (see Figures 5 and 6). Continuous tweaking and enhancement of the CourseMarker system has given exercise authors more freedom to develop good exercises. The ratio of student passes to failures is very high,

Exercise	Ex 5	Ex 7	Ex 8 part 1	Ex 8 part 2	Ex 8 part 3
1st submission average	69.02	78.23	66.98	66.12	44.47
2nd submission average	76.08	81.19	81.78	82.18	55.58
3rd submission average	79.99	90.27	89.09	86.08	78.56
4th submission average	81.09	-	-	-	-
Student submissions	129	104	122	104	83
Exercise average	83.79	85.86	87.40	80.92	56.12

Fig. 4. Changes in marks over several submissions (2004-2005).

and has improved with the evolution of CourseMarker and the support provided by the system.

Currently 92% of students passed PR1 and 93% PR2. Questionnaire results suggest that students prefer to be assessed on a weekly basis rather than being confronted with a final exam.



Fig, 5. First semester marks from 1998 – 2001.

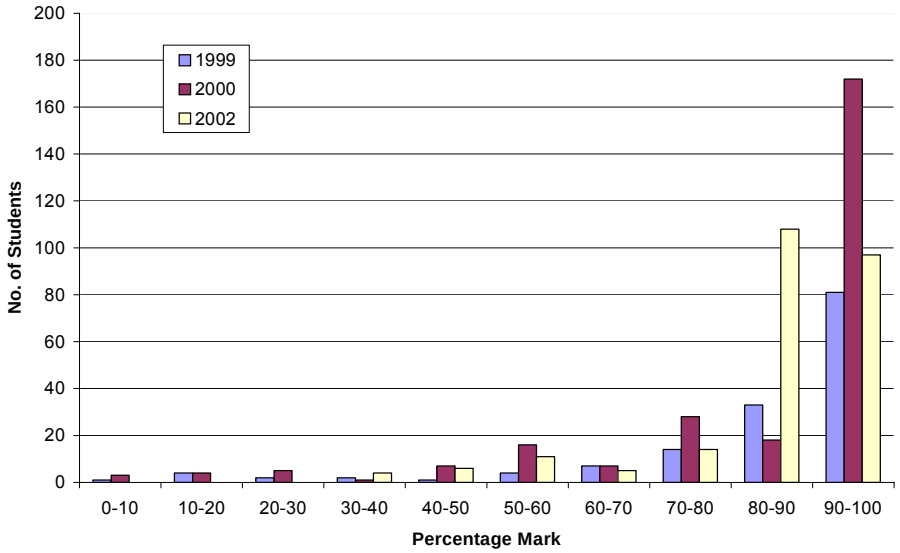


Fig. 6. Second semester marks from 1999 – 2002.

To test the efficacy of the CourseMarker system further, the 2003 student cohort taking PR1 were divided into two groups. One group was introduced to CourseMarker and were told to use it for their coursework, while the other group was told to do their coursework manually and to e-mail their solutions to their respective lab assistants. The lab assistants would then mark the students' work by hand. Some students chose to join a specific group, whereas others, who did not care whether their work would be marked by a human or a machine, were allocated into one of the two groups randomly. Comparing the results of both groups shows that CourseMarker marks at least as well as humans do, provides on-demand, impartial feedback, and as a bonus saves hundreds of marking hours for the academic staff.

6. FURTHER WORK

As described earlier, it is easy to extend the marking system so that it can accommodate more functionality than currently available. Future versions of the marking system may embed technologies such as object-oriented heuristics and refactoring strategies that will enable teachers and exercise authors to further enhance the existing CourseMarker exercises and help students improve their programming skills.

About 50 universities worldwide have taken copies of CourseMarker for trial and approximately 15 purchased the system for actual use. It is hoped that the further evolution of CourseMarker's marking system will increase the number of institutions interested in using it (it is available at CourseMarker.com). Research by the Learning Technology Research group at Nottingham is currently expanding on a number of issues, including the following:

- support for multiple-choice and other types of questionnaires [Yong and Higgins 2003];
- conversion of all Ceilidh courses to CourseMarker;
- design and implementation of a web CourseMarker client for the students, to allow them to use CourseMarker from any web-enabled computer;
- creating GUI-based Java programs;
- design and implementation of marking metrics for a database systems course and the implementation of a CourseMarker course and exercises for it; and
- a multimedia framework that links to CourseMarker in order to provide multimedia content to the students.

7. CONCLUSION

An overview of the CourseMarker CBA system and how it addresses the needs of the classroom at Nottingham University is presented in this article. Ceilidh, and now CourseMarker, have proven to be invaluable to the University of Nottingham and to other academic institutions. The CourseMarker system demonstrates a considerable number of improvements over the older Ceilidh system. CourseMarker's architecture and implementation satisfy the objectives of improving on the old functionality of Ceilidh, while also providing better feedback to users and increasing performance, scalability, maintainability, extensibility, and usability.

The design and implementation of CourseMarker's marking system is described in detail; the power and expressiveness of marking schemes written in Java are discussed; we also show that additional functionality can be accommodated in the future.

The changes made to the assessment and administration processes have proven to be successful, as the test results from the usability studies conducted at the University of

Nottingham show. This success is illustrated in the data provided in the results section. As the expressiveness of the marking process and the ease in managing courses are improved, automatic computer-based assessment with CourseMarker becomes ever more accessible to academic institutions.

APPENDIX

A complete printout of the set of exercise files for exercise javau6bsort can be found on the CourseMarker website: www.coursemarker.com.

BIBLIOGRAPHY

ARNOW, D. AND BARSHAY, O. 1999a. On-line programming examinations using WebToTeach. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education* (Krakow, Poland, June 27 – 30). 21–24.

ARNOW, D. AND BARSHEY, O. 1999b. WebToTeach: An interactive focused programming exercise system. In *Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference* (Puerto Rico, Nov.).

AHMADZADEH, M., ELLIMAN, D., AND HIGGINS, C. 2005. An analysis of patterns of debugging among novice computer science students. In *Proceedings of the ITiCSE 2005 Conference* (Lisbon)

AL-YAHYA, M., HIGGINS, C., BRAILSFORD, T., AND ASHMAN, H. 2005. Question Master: An open model for reusing dialogue in learning environments. In *Proceedings of the CAL'05 Conference on Virtual Learning.*

BENFORD, S.D., BURKE, E.K., FOXLEY, E., GUTTERIDGE, N. H., AND MOHD ZIN, A. 1993. Experiences with the Ceilidh system. In *Proceedings of the 1st International Conference on Computer Based Learning in Science* (Vienna).

BENFORD, S.D., BURKE, E.K., FOXLEY, E., GUTTERIDGE, N.H., HIGGINS, C., AND MOHD ZIN, A. 1994. Software support for automated assessment and administration. *J. Res. Comput.Edu*. (1994).

CHARMAN, D. AND ELMES, A. 1998. *Computer Based Assessment: A Guide to Good Practice.* Vol. I. University of Plymouth, 1998

CULWIN, F. AND LANCASTER, T. 2001. Plagiarism issues for higher education. *Inf. Security 21*, 2, 36 – 41.

DALY, C. 1999. RoboProf and an introductory computer programming course. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education* (Krakow, June 27 – 30). 155 – 158.

FOXLEY, E., HIGGINS, C., HEGAZY, T., SYMEONIDIS, P., AND TSINTSIFAS, A. 2001a. The CourseMaster CBA system: Improvements over Ceilid. In *Proceedings of the Fifth International Computer Assisted Assessment Conference* (Loughborough University, UK, July 2 – 4). 189 – 201.

FOXLEY, E., HIGGINS, C., SYMEONIDIS, P., AND TSINTSIFAS, A. 2001b. The CourseMaster automated assessment system – A next generation Ceilidh. In *Proceedings of the Workshop on Computer Assisted Assessment to Support the ICS Disciplines* (University of Warwick, April 5 – 6).

FOXLEY, E., HIGGINS, C., TSINTSIFAS, A., AND SYMEONIDIS, P. 2000. The Ceilidh-CourseMaster system, An introduction. In *Proceedings of the. 4th Java in the Curriculum Conference* (South Bank University, UK, Jan. 24).

FOXLEY, E., TSINTSIFAS, A., HIGGINS, C., AND SYMEONIDIS P. 1999. Ceilidh, A system for the automatic evaluation of students' programming work. In *Proceedings of the CBLISS 99 Conference* (University of Twente, Holland, July 2 – 7).

FOXLEY, E., HIGGINS, C., SYMEONIDIS, P., AND TSINTSIFAS, A. 1998. Security issues under Ceilidh's WWW interface. In *Proceedings of the ICCE'98 Conference* (Beijing, Oct.14 – 17).235 – 240.

HIGGINS, C., SYMEONIDIS, P., AND TSINTSIFAS, A. 2002a. Diagram-based CBA using DATsys and CourseMaster. In *Proceedings of the International Conference on Computers in Education* (ICCE2002, Auckland, New Zealand, Dec. 3 – 6).

HIGGINS, C., TSINTSIFAS, A., AND SYMEONIDIS, P. 2002b. CourseMaster marking programs and diagrams. In *Proceedings of the Dealing with Plagiarism in ICS Education Conference* (Warwick, April 11–12).

HYVÖNEN, J. AND MALMI, L. 1993. TRAKLA – A system for teaching algorithms using email and a graphical editor. In *Proceedings of the HYPERMEDIA Conference* (Vaasa). 141–147.

JACKSON, D. AND USHER, M. 1997. Grading student programs using ASSYST, In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education* (San Jose, CA, Feb. 27–March 1). 335–339.

LUCK, M. AND JOY, M.S. 1999. A secure on-line submission system. *Softw. – Pract.Exper.29*, 8, 721–740

MASON, D. AND WOIT, D. 1998. Integrating technology into computer science examinations. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education* (Atlanta, GA, Feb. 26–March 1). 1401–1404.

SUN MICROSYSTEMS. 1999. The JAVA HotSpot performance engine architecture. White paper. http://java.sun.com/products/hotspot/whitepaper.html, 1999.

SYMEONIDIS, P. 2001. An in-depth review of CourseMaster marking subsystem. Tech. Rep., LTR Group, University of Nottingham, UK.

SYMEONIDIS, P. 1998. Creating an exercise using CourseMarker (formerly JavaCeilidh). Tech. Rep., LTR Group, University of Nottingham, UK.

YONG, C. AND HIGGINS, C. 2003. Automatically creating personalised exercises based on student profiles. In *Proceedings of the ITiCSE 2003 Conference* (Thessaloniki, Greece, June 30–July 2).

Received January 2005; revised November 2005; accepted December 2005