The CourseMarker CBA System: Improvements over Ceilidh

COLIN HIGGINS (CORRESPONDING AUTHOR)

School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB UK, United Kingdom. E-mail: cah@cs.nott.ac.uk. Tel:+44 115 951 4213. Fax: +44 115 951 4254.

TAREK HEGAZY

School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB UK, United Kingdom. E-mail: tmh@cs.nott.ac.uk

PAVLOS SYMEONIDIS

School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB UK, United Kingdom. E-mail: pxs@cs.nott.ac.uk

ATHANASIOS TSINTSIFAS

School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB UK, United Kingdom. E-mail: azt@cs.nott.ac.uk

Abstract

This document reports on the results of re-designing and re-implementing the Ceilidh courseware system. It highlights the limitations identified in the thirteen years of Ceilidh's use at the University of Nottingham. It also illustrates how most of these limitations have been resolved by re-designing Ceilidh's architecture and improving various aspects of the marking and administrating processes. The new system, entitled CourseMarker (previously CourseMaster), offers enhanced functionality by adding useful features that have long been needed by Ceilidh's community. The paper concludes with an evaluation of the changes and a report on the experience of CourseMarker's use over the last four years. Finally, recent developments and future directions are discussed.

Keywords

Free Response Computer Based Assessment (CBA); Automatic Assessment of Programming Coursework.

1. Introduction

The first section of this paper gives a brief historical overview of the CBA systems that have been developed over the last 20 years. The following sections present the Ceilidh CBA system and its successor, CourseMarker, whilst comparing their architectures and discussing how the new design resolved the limitations of the old. Qualities such as scalability, performance, maintainability, extensibility, and usability are discussed. A

contrast on the marking and administration processes of the two systems is given which shows how improvements were made without compromising the old functionality.

CourseMarker was implemented in 1998 and tested for the first time as a replacement for Ceilidh in the academic year 1998-99. The first authored courses covered two Java programming modules containing 35 exercises in total. CourseMarker was made available to other academic institutions in February 2000 and is in use in about 20 academic institutions throughout the world where it supports the automation of coursework in classes that have as many as 1500 students.

2. Brief Historical Overview

The Ceilidh courseware system was one of the first CBA systems to provide functionality for the full lifecycle of programming courses. Ceilidh caters for the authoring of CBA coursework, the administration and management of modules and the presentation of information to the students. From its conception in 1988, Ceilidh had an important impact on the research and implementation of related CBA systems.

The Ceilidh system's success (Benford et al, 1993, Benford et al, 1995) suggests that the automatic assessment of computer programming exercises is both feasible and effective. The need for automating the assessment of programming courses has been perhaps greater than in any other field, primarily because computer science has attracted an increasing number of students. Kay et al express the opinions of many educators stating that: "*The time has come to devote effort and resources to developing robust, flexible, widely*

available tools for automatic program evaluation. We can no longer pretend to be able to assess student's work validly and reliably by manual means" (Kay et al, 1994).

Other CBA systems similar to Ceilidh soon followed such as ASSYST, RoboProof, BOSS and TRAKLA. Jackson and Usher describe the ASSYST system (Jackson 2000), a hybrid CBA system in which some of the marking processes are automated and some are performed by human markers. Jackson and Usher claim that a hybrid approach has advantages over fully automated assessment. By allowing fine-grained selection between automatic and manual assessment a teacher can benefit from the marking consistency and speed while maintaining full control over student results. The assessment techniques used in ASSYST are very similar to the Ceilidh system. (Jackson and Usher 1997).

Daly presented RoboProof, an on-line teaching web-based system (Daly 1999). RoboProof has been used to teach the syntax and structure of programming languages. Daly, after experimenting on a C++ course, reported results that demonstrate an improvement in learning and assessment. It allows any number of submissions and does not penalise for failures.

Joy and Luck presented a CBA system, entitled BOSS, that includes facilities for both automatic and manual marking. In Boss, a graphical user interface has been devised to ease the development of the marking scheme under the name "Electronic MarkSheets". Evaluating the Ceilidh system Joy and Luck state: "*One type of package in particular deserves some discussion because of their size and distinct approach. Systems such as Ceilidh are packages which provide a full programming environment, handling… submission and testing of assignments…"* Having evaluated Ceilidh, Joy and Luck

concluded that a major problem in Ceilidh is not supporting customisation of its functionality (Joy and Luck 1998).

Korhonen and Malmi have described TRAKLA, a system to aid the learning of data structures and algorithms. The system employs visualisation, animation and simulation for presenting concepts to the student. It also contains automatic assessment tools for formative evaluation. The architecture of TRAKLA's assessment component is similar to Ceilidh and this is acknowledged by Korhonen and Malmi in (Korhonen and Malmi 2000).

3. The Ceilidh Software System

3.1. Overview

Ceilidh provides for the full life cycle of Computer Based Assessment coursework. Typically the life cycle of an automatically assessed exercise includes the stages of its development, execution and administration. The Ceilidh project was developed in the mid-80s. Ceilidh's focus concentrated on mechanisms to facilitate the assessment of computer programming courses. Later, facilities to support other types of assessment were added. Ceilidh has been widely used throughout the academic world, serving both as a system with pre-prepared courses and as a suitable infrastructure for the research and evaluation of new ideas for the assessment of various subjects.

3.2. Architecture

Ceilidh's design objectives have been to support multiple courses, automatic feedback, multiple interfaces, remote learning and to allow for extensibility and portability (Foxley et al, 1998a). Ceilidh took an architectural approach based on three layers in order to separate the data from the various tools and interfaces. Figure 1 depicts the three layers as they relate to Ceilidh's users.

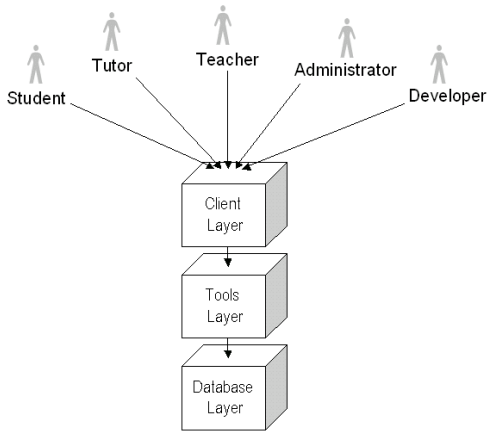


Figure 1: Ceilidh's three-layered architecture as it relates to its users

The database layer includes information stored for courses (such as authored material for notes and exercises), data archived for the year (such as user lists, submissions and marks) and many types of system properties and configurations. The structure of the information that constitutes the exercise material depends heavily on the type of assessment. For example, programming exercises have an entirely different organisation than multiple-choice questions. CBA courses can be developed for any domain that can take advantage

of Ceilidh's assessment provisions. Assessment feedback is part of the properties of an exercise and is described while configuring the assessment mechanism.

The tools layer consists of executable programs that understand the formats of the various files in the database layer and inter-communicate through well defined protocols (Benford et al., 1994). The objective of allowing extensibility has been met. Many tools have been developed since Ceilidh's initial release providing novel functionality and metrics in new domains (Foxley et al, 1998a). Its core version contains 70 tools, 51 of which are Unix shell scripts and 19 are other programs written in C.

The user interface layer accesses the tools layer in order to make the functionality listed in table 1 available. The tools layer eases the development of new views for Ceilidh users, and includes a command line interface (Foxley et al, 1996), a text menu and a web interface (Foxley et al, 1997). These interfaces not only reflect the responsibilities of each type of user but also the type of material being tested. For example, the student's menu on a Prolog-based programming exercise has different options than those of an essay exercise. Figures 2 and 3 depict Ceilidh's text and web interfaces. The design objectives have been successfully met except perhaps the ones that were related to portability. As PCs became more common, the need for portability rapidly increased.

```
CEILIDH system Course/unit menu:
lu          list unit titles           |     su    set unit code
lx          list unit exercise titles  |     sx    move to named exercise
vn          view notes on the screen   |     pn    print notes on beth
csum        read course summary        |     usum  read unit summary
vm          view all marks             |
clp         change printer             |     h     for more help
co          make a comment to teacher  |     q     quit
```

```
CEILIDH system exercise menu:
vq    view question on the screen     | pq     print question on beth
co    make a comment to teacher       | set    set up coursework
ep    edit program                    | cm     compile program
sub   submit for marking              |
h     for context help                | H      for general help
q     to return to calling menu       |
rex run solution executable           | rxt    run sol'n against test data
```

Figure 2: Student's view of Ceilidh's text menu interface

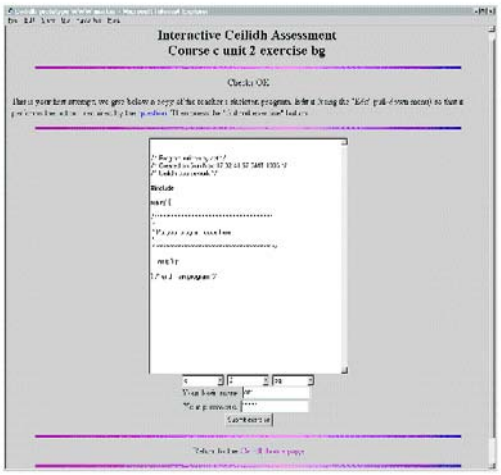


Figure 3: Student's view of Web interface

3.3. Advantages

The Ceilidh system was originally implemented to address the practical problems involved in the teaching of programming courses to large numbers of students. Ceilidh supports automation for the presentation of material, the administration of the course, and most importantly the assessment of student work.

For the presentation of information to the students, Ceilidh implements a hierarchical course structure with courses, units and exercises in which lecture notes, tutorials and other course related information can be organised and published. Ceilidh keeps students informed of their marks and returns feedback for submitted coursework.

For the administration and management of courses, Ceilidh assists in a number of ways. Firstly, it defines a secure and on-line process for the creation and submission of student coursework. Secondly, it supports customised views for its users, distributing the various responsibilities amongst them. The view of the teacher supports functions for monitoring individual and overall student progress as well as individual and overall exercise progress. Thirdly, Ceilidh handles the archiving of submitted student work, and manages the intercommunication between users. In addition, Ceilidh supports features for plagiarism detection.

For the automatic assessment of student work, Ceilidh has put forward a generic technique that can be applied to the development of both fixed and free response types of assessment. By introducing the concept of *marking tools*, Ceilidh supports the development of a variety of CBA exercises. Marking tools embody programs that examine a specific quality in the submitted coursework and return marks and feedback to the invoker. Marking tools have been implemented in Ceilidh for assessing courses in programming languages, multiple choice questionnaires, question/answer exercises, single sentence/word answers and essays/reports.

Although the presentation of information to the students is an important aspect of the learning process, Ceilidh has concentrated primarily on the automatic marking of student work and secondly on the administration of course modules.

3.4. Limitations

Ceilidh has been an open system, maintainable and upgradeable. However, upon closer inspection, Ceilidh had a number of short comings:

- The marking system had dependencies on many of Ceilidh's internal parts. Changes propagated to many levels.
- The limited expressiveness and configuration capability affected the quality of the marking process.
- The text-based interface was complex and difficult to learn for novice users.
- The architectural limitations inflicted a penalty on performance, maintenance, and usability.

To overcome Ceilidh's limitations a redesign was deemed necessary. This would allow the system to support more students and a variety of platforms. It would also satisfy a wider range of requests in terms of modifications and updates. The main concerns motivating this extensive change were to resolve Ceilidh's architectural limitations, to improve maintainability, allow functional extensions and diverse configurations and increase usability for all its users.

4. The CourseMarker System

4.1. Overview

The fundamental motivation in designing CourseMarker has been an aspiration to address Ceilidh's limitations. The goal was to make CourseMarker more flexible in sustaining changes than its predecessor. An attempt to restructure the useful parts was made aiming to increase scalability, performance, maintainability, extensibility, and usability.

Both Ceilidh and CourseMarker support five types of users with respective responsibilities: Students, Tutors, Teachers, Developers, and Administrators. They define four levels for the presentation of the course material: System, Course, Unit and Exercise. Users inherit available functionality. Teachers can do everything that tutors can do, tutors can do everything that students can do and so on. However, the two systems differ in the available functionality that they provide to their users. As table 1 illustrates, CourseMarker preserved most of Ceilidh's functionality and added new features.

System levels / User types	System	Course / Unit	Exercise
Student	⊗ - View system documents ○ - Set environment properties ○ - Customize system view ⊗ - View system MOTD* × - Register to a course	- View course documents: ⊗ - View course notes ⊗ - View course summary ⊗ - View course MOTD ⊗ - View total course work ⊗ - View all student's marks - View unit documents: ⊗ - View unit notes. ⊗ - View unit summary.	⊗ - Set up exercise ⊗ - View exercise question ⊗ - View exercise skeleton ⊗ - Develop exercise ⊗ - Submit exercise ○ - View analytical mark and feedback ⊗ - View exercise test data (TD) ⊗ - View marks: - Execute solution: ⊗ - Interactively ⊗ - With TD × - With teacher's TD × - Execute teacher's solution with student TD ⊗ - Enquire by e-mail
Tutor		⊗ - View students' course marks ⊗ - View student statistics ⊗ - Check course state × - Register student to a course ⊗ - View student's work	⊗ - View student solutions × - Mark solutions (manually or automatically) ⊗ - View statistics ⊗ - View missing / submitted students ⊗ - View tutor help ⊗ - View plagiarism results × - Find tutees for a specific tutor
Teacher	⊗ - Add / delete students, tutors, teachers × - List tutor's tutees × - View audit trails	⊗ - Edit course MOTD ⊗ - Check / edit course properties ⊗ - Edit unit properties	⊗ - Edit tutor help ⊗ - Edit exercise question ⊗ - Change exercise properties ⊗ - Expunge student's work ⊗ - Set borderline ⊗ - Search for plagiarism
Developer		⊗ - Create / add courses ⊗ - Create / add units	○ - Copy / delete exercises × - Author exercises
Administrator	⊗ - Edit system MOTD ⊗ - Add / delete Administrators ○ - View error log ⊗ - Register students	⊗ - Edit course documents ⊗ - Install new courses	○ - Give extensions ○ - Copy / delete exercises × - Author exercises

×: Ceilidh ○: CourseMarker ⊗: Both Ceilidh and CourseMarker * MOTD: Message of the day.

Table 1: Users' responsibilities at different levels of system use – Ceilidh vs. CourseMarker

4.2. Architecture

In order to satisfy the deployment requirements arising from the wide range of computing configurations that exist in academic institutions, it has been decided to develop CourseMarker using the Java language. Java facilitates platform independence and offers a convenient distribution mechanism with its the Remote Method Invocation (RMI) mechanism. RMI is effective and relatively easy to use as it makes the network transparent to the programmer. The latter feature has been especially sought as in the early stages of

development the inherent complexities of building distributed systems in heterogeneous environments were identified.

In an effort to reorganise Ceilidh's functionality in a more flexible way, the dependencies, commonalities and variations between the tools and data layer have been identified. The commonalities have been abstracted into class hierarchies, explicit points of extension were defined and the dependencies were decreased by separating the various responsibilities between seven logical parts. Those parts were implemented as servers that operate as remote objects. Every server manages an associated filestore. Figure 4 depicts a high level overview of the relationships between the types of users and the remote objects. Table 2 lists CourseMarker's servers (remote objects) and summarises their responsibilities. Every server is decoupled from the others so that it can operate as independently as possible.

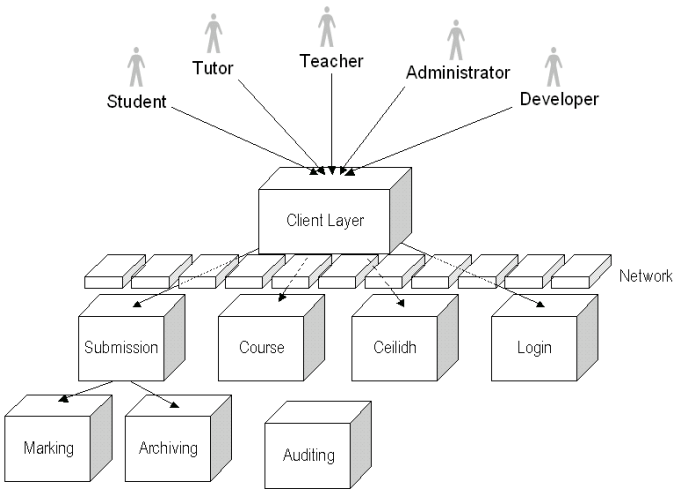


Figure 4: A high level view of CourseMarker main parts

Server Name	Function
Login	Authorises users to perform the requested tasks
Course	Manages course material and responds to requests
Submission	Decides between accepting and rejecting a submission.
Marking	Marks the submission and produces analytical feedback
Archiving	Archives student work and issues unique receipts
Auditing	Logs audit trails for various configurations
Ceilidh	Provides course related information to its clients

Table 2: CourseMarker's servers and their responsibilities

Interface Name	Description
Submission	Contains all the files of a student's project in addition to student and security related information
MarkingResult	Represents a single or composite assessment result for a student's solution. It carries detailed feedback for all assessment criteria.
Receipt	Confirms the completion of a submission. It is issued by the Archiving-Server and is sent to the client.
CourseModule	Embodies a whole course, unit or exercise. Every level has its own structure and properties
Project	Represents the type of assessment. CourseMarker provides for various programming, diagramming, and essay types of projects.

Table 3: CourseMarker's basic objects for communication between the servers

CourseMarker's remote objects inter-communicate using a set of interfaces that are exchanged between all parts. Table 3 lists the interfaces and explains their meaning.

CourseMarker has effectively replaced Ceilidh's 70 tools with an object-oriented architecture that comprises of 28 interfaces and 263 classes contained in 23 packages. Six packages contain client-related classes, four contain common classes between clients and servers and the remaining packages contain server related classes.

It should be noted that RMI is currently not used for communication between servers as the physical segmentation of servers has not yet been necessary. However the design allows this trivially.

4.3. Increasing Software Quality

Performance, scalability, maintainability and usability have been Ceilidh's weakest points. CourseMarker approached these limitations by considering them as vital requirements during the design stage.

4.3.1. Performance and Scalability

Good performance is indispensable to any system that has to be scalable. The decision of implementing CourseMarker in Java caused some initial concern as, at the time, programs developed in Java executed much slower than those developed in native code. However, these concerns quickly disappeared after some preliminary prototyping and the latest advances in HotSpot compilers (SUN Microsystems, 1999). CourseMarker can almost achieve the performance of natively compiled code.

CourseMarker outperforms Ceilidh for two reasons: Firstly, CourseMarker has considerably reduced the number of spawned processes executing in the Operating System (OS). Ceilidh's tools layer consists of externally invoked OS based programs. For every option Ceilidh's users choose, the OS must execute processes that interact by exchanging data through UNIX pipes and files. Context switching between processes takes much more time than switching between threads. CourseMarker represents Ceilidh's tools as internal objects on a single process and uses threads as the token of its concurrency.

Secondly, by offloading the client tasks to the client side, CourseMarker has succeed in relieving the servers from the burden of executing student-spawned processes such as compilations, simulations, visualisations and in general other external tools. On a typical programming exercise in Ceilidh, a student usually compiles their program many times before submitting. By moving those processes to the client, CourseMarker decreases significantly the amount of resources it would have needed otherwise on the server machine.

CourseMarker's implementation has been further optimised using profiling and optimisation tools. Even better performance can be achieved by upgrading the hardware platform underneath the system, or by using two or more processors in Symetric MultiProcessing mode (SMP). CourseMarker can take advantage of servers with multiple CPUs thus significantly increasing performance without the need of re-compilation or re-configuration.

The future performance of a system can be directly linked to its potential for large-scale use.

CourseMarker scales better than Ceilidh for an additional reason. The design decision of maintaining independency between CourseMarker's remote objects, allows the separation of the servers into multiple machines. In this configuration, every task takes only a time-slice from every machine that participates in the distribution. The speed and resource need of RMI for the intercommunication of the servers does reduce performance slightly. However, if a greater need arises it is possible to add load-balancing and to partition multiple CourseMarker servers into clusters.

4.3.2. Maintainability and Extensibility

Maintainability is an important quality for software but is often overlooked in favour of short-term objectives. This leads to software often being more expensive in its maintenance stage than in its initial development.

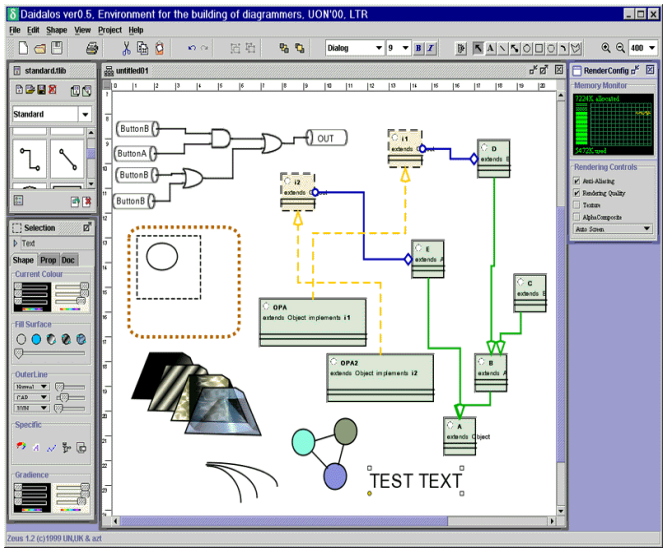
CourseMarker has a considerably higher degree of maintainability than Ceilidh for four reasons. Firstly, its object-oriented architecture explicitly exposes design hotspots in anticipation of future changes. Design hotspots allow modifications through sub-classing and run-time configuration. Secondly, encapsulation and modularity are much better catered for in the object-oriented paradigm. Thirdly, Java's platform independence ensures that there are no platform specific segments of code. Finally, in contrast with Ceilidh, CourseMarker's source code demonstrates improved readability and assiduous documentation.

Extensibility has been one of the most important objectives from the early stages of redevelopment. Facilitating experimentation and research for assessment has been one of Ceilidh's objectives. Various types of courses have been developed over the years, the highest proportion of which were authored by Ceilidh's community (Foxley et al, 1998a).

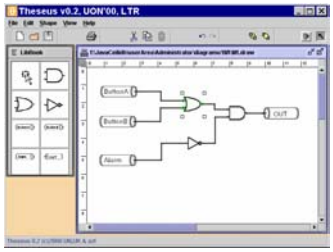
CourseMarker increases Ceilidh's extensibility with three significant enhancements. Firstly, it employs the idea of describing the exercise's marking process in Java, which subsequently allows for an increased amount of customisation to take place if required. Java's control structures can be used to fine tune the marking of an exercise. Additionally, this marking can acquire information directly from CourseMarker's internal state.

Secondly, CourseMarker provides extensive facilities to inter-operate with other programs. It features a generic type of course, which can be parameterised to create user-defined course types. In such exercise types, a custom environment can be invoked for the students and a custom project can be easily defined.

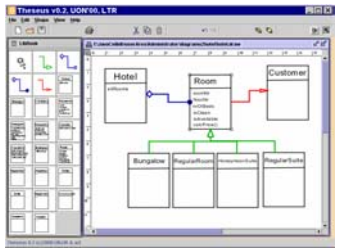
Thirdly, CourseMarker includes a recently developed facility that allows the authoring of generic diagram-based exercises. In this type of course, the exercise author can define the specifics for the domain, the exercise and its assessment, as well as creating the domain's editor. The student's environment is automatically generated and it allows the student to graphically design the exercise's solution. Figure 5 illustrates the authoring environment and two editors that were authored for coursework in object oriented and circuit design.



(a) The authoring environment for diagram-based exercises

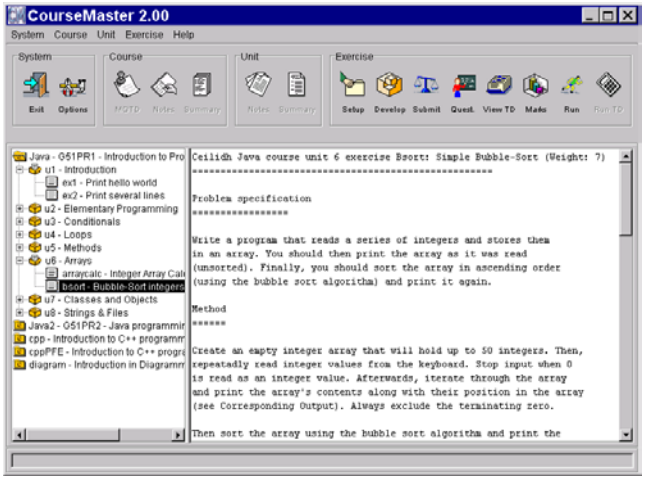


(b) A circuit design exercise

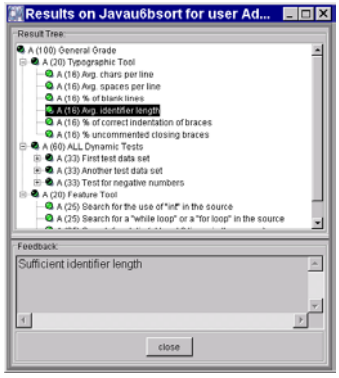


(c) An OO design exercise

Figure 5: CourseMarker's diagram-based assessment



(a) The student view for a course



(b) Exercise results for a student

Figure 6: CourseMarker's student interface

4.3.3 Usability

CourseMarker improves on Ceilidh's usability for the student. A number of CourseMarker clients have been developed. Prior to release at Nottingham, an initial text-based client was developed for testing purposes. A GUI client running on top of Java's Abstract Windowing Toolkit (AWT) was the first client that students used at Nottingham in 1998. In 1999, a new client was developed, using Java's JFC (Java Foundation Classes) toolkit. The new client requires more processing power than the AWT, but is more flexible and intuitive (see figure 6).

Informal discussion with users of the Ceilidh system indicated the need for improvements to the interface. In comparison to Ceilidh, CourseMarker provides the students with a superior interface based on these discussions. Figure 6(a) illustrates CourseMarker's standard JFC view for students. The tree component on the lower left represents the available courses, units and exercises and allows browsing through the course's material. The options to the student are context dependant and sensitive to the state of the exercise. For example, students that haven't compiled their programs can't select the option to submit. Options appear as menu items, toolbar buttons and shortcuts. Students can customise their view in a number of functional and presentational ways. The area where the notes are presented at bottom right can render either text or HTML documents.

CourseMarker also improved Ceilidh's usability in respect to its deployment and configuration. CourseMarker's installation is a much easier process than that of Ceilidh's. Ceilidh uses shell-scripts that require an in-depth knowledge of paths, environmental variables and external tools. In contrast, CourseMarker employs a graphical installation

wizard that guides the user through the installation sequence. However, the configuration of CourseMarker requires some basic networking knowledge, such as configuring TCP/IP addresses and ports, and some minimum Java knowledge, such as setting the CLASSPATH variable correctly.

5. The Marking Process

Ceilidh can assess student work of various types. It can check multiple-choice questionnaires, simple text/numeric value entries, essays and computer programs in a variety of languages. Courses for assessing programming material have been written for most computer programming languages taught in academia such as C, C++, SML, Pascal, SQL, FORTRAN, Modula 2, Prolog, Z specifications and UNIX shell script.

CourseMarker maintains the ability to mark courses already written for Ceilidh, although some of the existing courses have yet to be ported. It also provides additional support for Object Oriented (OO) languages and design. To satisfy new requirements set by the assessment of languages such as Java, CourseMarker made improvements to the functionality of the marking tools and marking scheme, and to the execution of the assessment mechanism.

At the exercise level, both systems require the students to:

- Read the question to the exercise. The question accurately specifies what the student's program/solution has to do, and the formats of any input and output files and data.

- Obtain a skeleton solution along with any header files and/or testing tools. This can be anything from an empty document to a fully functioning solution at the discretion of the teacher.

- Develop a satisfactory solution.

- Submit their solution for assessment and receive a grade and comments on their work.

Both Ceilidh and CourseMarker allow the students to resubmit their solution for as many times as the exercise developer specifies.

5.1. Marking Tools for the Assessment of Programming

The assessment of programming courses in Ceilidh was originally carried out by configuring a set of metric tools that performed various quality checks upon submission of a student program. As table 4 illustrates available metric tools included typographic layout, dynamic execution, program features, program complexity, program structure and dynamic efficiency (Foxley and Zin, 1991). However, as the last few years in the teaching of programming have been characterised by a transition from performance issues to design issues, the last three metrics have become less relevant and hence redundant.

Tool	Description
`Typographic`	Checks text for solution's typographic features

`Dynamic`	Tests solution's behaviour via interaction (e.g. runs it against test data)
`Feature`	Inspects for features specific to the exercise
`Complexity`	Statically analyses the occurrence of programming constructs
`Structure`	Enquires for weaknesses in lexical structure
`Efficiency`	Profiles the solution against a model solution

Table 4: Ceilidh's basic tools for the assessment of programming-related courses

Regarding typography, both systems check the program layout, indentation, choice and length of identifiers and usage of comments. All typography parameters can be customised on a per exercise basis. In addition, CourseMarker provides a way to parameterise the typographic feedback students receive with a much finer granularity.

Regarding dynamic execution, both systems run the student's program several times with different test data sets. This verifies whether the student's program is correct and satisfies the specification. In contrast with Ceilidh, CourseMarker gives a wide range of options concerning the flow, control and order in which the dynamic tests are executed.

Regarding program features, the student's source code is checked for special features that are exercise dependant. For example, an exercise set on a unit that teaches the difference between "switch/case" and "if" statements would typically include a feature test to ensure

the appropriate use of each construct. Both systems take the same approach towards the execution and configuration of the program features mechanism.

A detailed view of the available metric tools in Ceilidh is documented in (Foxley and Zin, 1991).

5.2. The Marking Scheme

A crucial modification that was requested by the Ceilidh community was to improve the expressiveness of the marking process. Ceilidh allowed exercise authors to describe the marking process as a sequence of invocations defined in a file, called the mark-action file. Each line of the mark-action file required two elements, the name of the marking tool to be invoked and the highest grade that this tool could contribute to the overall mark. Although this mechanism simplified the authoring of exercises, it did not provide the necessary control structures to allow fine-grained customisations for the assessment of each exercise and detailed settings for feedback to the students.

CourseMarker provides the developer with the means to almost limitlessly customise the marking process by using the idea of a marking scheme. A marking scheme is expressed as a Java program and is a fundamental property of every exercise. Upon student submission the marking scheme of the respective exercise is linked to CourseMarker at run-time and is executed. The author of the marking scheme has access to CourseMarker's state and marking tools while being able to use programming control structures. The marking scheme can call any other external tools to help with the assessment.

5.3. Feedback Detail and Grading Styles

The return of immediate feedback upon submission is a fundamental feature contributing to pedagogic benefits that the CAA published community has often described (Charman and Elmes, 1998). Both Ceilidh and CourseMarker provide mechanisms for immediate feedback.

In Ceilidh, feedback to the student is limited to a mark that is composed of the results obtained by the marking tools participating in the marking process. It does not directly justify the loss of marks to the students. Also no explanation is given in order for the student to improve their mark.

CourseMarker provides a much improved feedback mechanism and presents the students with their results in detail. Figure 6.b illustrates a CourseMarker's GUI client displaying this information using a tree component. The students can navigate the tree and easily identify the reasons for their lost marks. Comments on how to improve their solution or links with further reading material can be given. Experience has shown that overly detailed feedback can be detrimental to the student's learning experience. In CourseMarker, the amount of feedback can be regulated to match the needs of the classroom. In addition, the exercise author can choose whether the student's mark is to be displayed in a numeric or in an alphabetic scale. The association between numeric values, letters, colours and shapes can be customised.

5.4. Exercise Parameterisation

In both systems, every exercise is customised using a property file. Each property is used to specify a behavioural aspect of the exercise's assessment in a controlled environment. For example, an exercise's skeleton filename, the maximum allowed number of submissions and its availability status (e.g. open/closed/late) are all examples of such properties.

Exercise properties in the Ceilidh system are inherited from parent levels such as the unit, course and system levels. Therefore, properties like the maximum allowed number of submissions can be specified for a whole unit and can be overridden for a specific exercise. Although it was anticipated that this mechanism would decrease the unnecessary repetition of properties, we observed that it actually hinders readability and maintainability.

CourseMarker omits this inheritance mechanism for exercises. Each exercise has a separate property file that contains the complete parameterisation for that exercise. CourseMarker introduced additional properties for the assessment of programming courses such as the maximum allowed number of lines that a student program is able to output and the maximum CPU time allocated for the running of a student's solution. Further parameterisation can be given to address the configuration of other types of assessment such as diagramming and application-specific courses.

5.5 Marks Scaling

Academic institutions often have to comply with (departmental) policies concerning the distribution of the final marks. Marks scaling is needed to convert the reported marks to an accepted scale.

Both systems provide mechanisms to scale the student marks. Scale files are used in order to specify whether there should be any scaling and how this should be performed. In CourseMarker, mark scaling can be applied at the course or exercise level. This provides an additional level of fine-tuning that gives better control over the students' end results.

For various reasons including legal and moral issues the students must be able to see both of their marks. This information is provided by facilities in both systems.

6. Systems Administration Enhancements

Ceilidh and CourseMarker provide support for administering various aspects of a course. For example, the adding and removing of users, courses, units and exercises, the opening and closing of exercises, and the monitoring of students are important features that are supported in both systems.

The administration of Ceilidh and CourseMarker is very manageable. Ceilidh provides administrative tools in the form of shell scripts that, upon execution, support the tasks listed in Table 2. CourseMarker has a web administration facility that performs similarly to Ceilidh's shell scripts. It uses a combination of dynamically generated HTML pages and

CGI scripts. The available facilities include cohort student statistics, changing of exercise properties and viewing missing/submitted students.

CourseMarker's web facilities include a feature that suggests to the teacher potential exercises that satisfy course-based criteria. In addition, a web-based wizard allows the exercise developer to create new exercises for CourseMarker. The wizard guides the developer through the authoring process by presenting them with the sequence of steps that have to be completed. An additional mechanism allows the exercise developer to create a new exercise by modifying an existing one with similar features.

CourseMarker has a secure remote server console client. The client allows an administrator to connect to the CourseMarker servers and perform tasks such as monitoring and shutting down the system.

6.1. Security

Security is paramount in systems that support automatic assessment for summative purposes. One of the major security risks is posed when assessing programming executables. An astute student could devise malicious code in order to gain unauthorised access and cause damage.

Both Ceilidh and CourseMarker feature a number of security mechanisms (Foxley et al, 1998b) in order to ensure safe and trouble-free execution. Ceilidh's security is based on the SUID and GUID security that Unix systems provide. There is no provision for any additional security measures.

Security has been a primary design concern for CourseMarker. For programming courses, CourseMarker uses the SUID and GUID security mechanisms when running under Unix. Alternatively, the "runas" command can be used when running under Windows 2000. CourseMarker also uses encryption for the sensitive information exchanged between servers.

Security mechanisms work in conjunction with any other restrictions and/or privileges an administrator may assign to the students.

6.1.1. Auditing

Both Ceilidh and CourseMarker include auditing facilities. All actions of the various subsystems are logged; for example, submissions, marking, login and archiving. CourseMarker additionally can log its operations. The detail of the level of the auditing process can be configured at startup or at runtime. The logging trails created by the auditing subsystem can be monitored either by examining the log files or online using CourseMarker's remote server console tool. CourseMarker's auditing facilities allow for simultaneous screen, file and network output of the system's trails.

6.1.2. Authorisation-Login

Students are allowed to use CourseMarker only if their username has been added to the login list. The administrator is responsible for maintaining user lists. CourseMarker provides two forms of authentication. The default way is to store the user's password in the login file. An alternative way requires the setting up of a POP3 server that will authenticate the users on behalf of CourseMarker.

For each successful login, a unique session key is generated for authorisation purposes. Each key is assigned to a CourseMarker client and is validated on every transaction for the lifetime of that login.

6.1.3. Password Encryption

The passwords that are transmitted between the clients and the servers may pass through potentially insecure networks. CourseMarker uses the DES password encryption algorithm to overcome this problem and minimise risks of security breaches through network packet interception.

6.2. Daily Administrative Tasks

6.2.1. Monitoring and Management

Monitoring of the system can be performed using either the web facilities provided and/or the remote server console tool. The web facilities help the administrator to add and delete users, view error logs, edit course documents, create and install new courses units and exercises, gather exercise metrics and grant extensions to students.

System statistics and debugging facilities are available only on the administration console tool. The system statistics displays, amongst other information, the number of submissions processed, assessed and archived and the number of users currently logged in. The administration console tool also allows the reloading of the course directory structures, should a change occur. This is needed as CourseMarker caches the directory structures in memory to increase performance.

6.2.2. Archiving

Ceilidh archives only the most recent submission of a student. By contrast, CourseMarker archives all students' submissions. It is also possible to revert to a previous submission if there is sufficient reason to do so, for example, if requested by a tutor.

All submissions are date and time stamped. Student receipt files are generated on every submission, and both binary and ASCII versions of those files are kept. The binary versions are used internally in CourseMarker, while the ASCII versions are used by the web tool.

6.2.3. Plagiarism Detection

With the emergence of the Internet, academic institutions are increasingly concerned with the submission of plagiarised material. Operating system based security measures have to be taken into account in order to deter students from copying from each other. However, there is no means of guaranteeing that students will not share their work with others.

A solution to this problem was first introduced in Ceilidh in the form of a plagiarism detection tool. The tool compares all the student solutions with each other and reports evidence of plagiarism based on the similarities of student work. When used for programming projects, the tool can detect comment and variable alterations as well as syntactical variations (e.g. the transformation of a "for" loop to a "while" loop, or the modification of a "switch" statement to multiple "if-else" statements).

7. Evaluation

The decision to re-implement Ceilidh in Java led to an increase in the system's performance and scalability. At the University of Nottingham, CourseMarker has been used to assess more than 1000 student programs per week. Although the load has peaked before deadlines, CourseMarker has had relatively few problems and has run reliably for over four years. Reliability is also indicated by responses from other universities that have obtained CourseMarker. This feedback shows that the system is very reliable at high loads. For example, the National University in Singapore (NUS) has run CourseMarker sometimes marking more than 3000 assignments per week.

7.1. Reported usage from the University of Nottingham

Two Java courses (PR1 and PR2 in 1st and 2nd semester respectively) have been initially authored under CourseMarker. The University of Nottingham has been teaching these two courses to their first year undergraduate students since 1998. The courses contain more than 35 exercises in total. A diagrammatics course that features object oriented design and digital circuit design has also been developed. This course has been taught at the University of Nottingham during the academic period of 1999-2000. Additional course material covering more graphical domains is currently being added to the diagrammatics course. Furthermore, a C++ course has been recently developed. Ceilidh's extensive exercise base can be used as a source of exercise material. It is easy to convert Ceilidh's C and C++ exercises to CourseMarker by following the appropriate guidelines or by using an administration tool recently developed for this purpose.

Usage data has been gathered during the four year period of using CourseMarker at the University of Nottingham. Analysis of this data suggests that CourseMarker's feedback mechanism, along with the revised exercises and teaching material, guides the students into achieving better grades. Figure 7 illustrates the number of submissions that the students have used in order to solve the exercises in PR1 and PR2. Although some students used only one submission, the majority of the students (94% and 95% respectively) have been consistently improving their marks by learning from their mistakes and using an additional 2 submissions to revise their solutions. Further analysis of the data indicates that the improvement in marks from the 1st to 3rd submission is 43% for PR1 and 83.4% for PR2. (A few students that had requested extensions for various reasons, mostly medical, were given "late" submissions.) Questionnaire results show that 82.7% of the students felt that the number of allowed submissions was adequate for their task. On the questionnaire, students were asked to provide a value in the range 1 to 3 that indicates whether they put more effort into solving CourseMarker exercises than hand-marked ones (where 1 was less effort and 3 was more effort). The students reported that it was the additional submissions that CourseMarker provided which motivated them into putting more effort, with an average response value of 2.4.

CourseMarker has been handling an increasing number of students each year. Figure 8 depicts that during the early period of using CourseMarker, students were achieving disproportionately good grades relative to hand marking. However, careful tuning of the marking subsystem and the exercises over the last four years has yielded more conforming results, without increasing the difficulty of the exercises. The average mark over time for

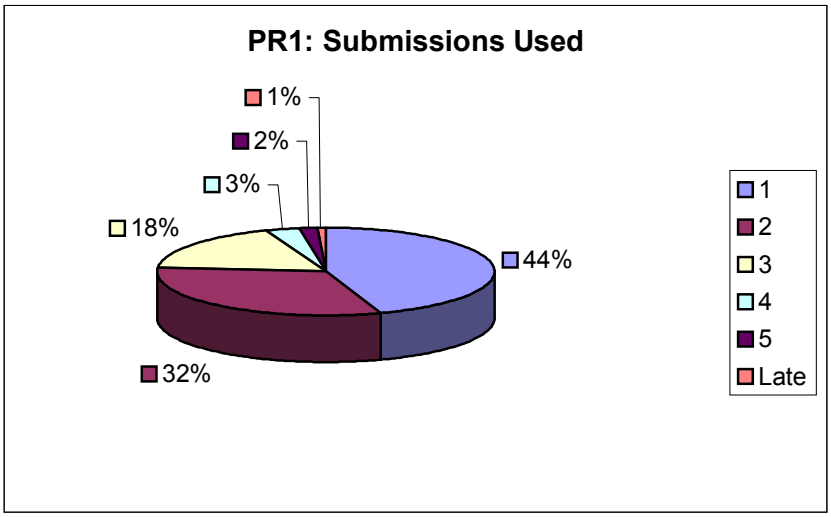
PR1 is 78% and for PR2 is 73.6%. A further subsequent scaling process ensures that the marks are consistent with the rest of the university's curriculum.

The student pass/fail ratio is high which may be due to the introductory nature of the courses. Figure 9 shows that for the four years of data gathered so far, the average pass/fail ratio is at 92% and 93% for both courses respectively. Questionnaire results indicate that the students unanimously agree that they prefer to be assessed on a regular basis rather than being confronted with a final exam at the end of the semester.

While the CourseMarker servers run reliably on a continuous basis, due to building access restrictions the students use their CourseMarker clients and submit their solutions only during office hours. Figures 10 and 11 display the number of student submissions along with their achieved marks for PR1 and PR2 respectively. From the results it can be seen that most students prefer to submit during the 14:00 to 17:00 hour period. Higher grades have been achieved during the same hours. This was expected, as during those hours the Computer Science Department of the University of Nottingham runs labs in order to help first year undergraduate students with their coursework.

There are a number of lab assistants at the disposal of the students, should they have a question regarding programming. The assistants are now allocated to a specific group of students. Questionnaire results suggest that the students seek help from the lab assistants on an almost exclusive basis instead of asking their fellow students or their lecturers. Recently, the cohort of students has been split into two. Half of the students were instructed to use CourseMarker in order to solve their coursework, while the rest were

instructed to e-mail their solutions to their respective lab assistants for traditional hand marking. A preliminary comparison of the marking results indicate that CourseMarker's marks are on a par with marks given by human markers, while saving hundreds of marking hours for the academic staff.



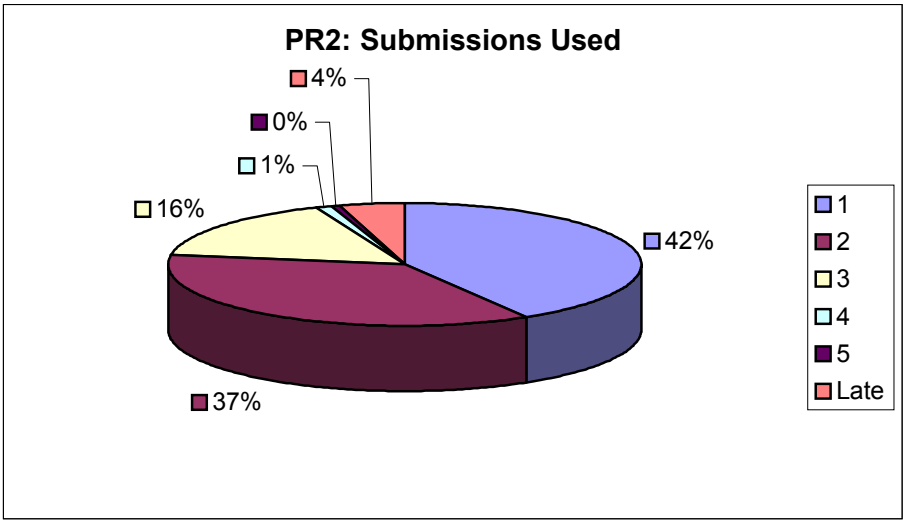
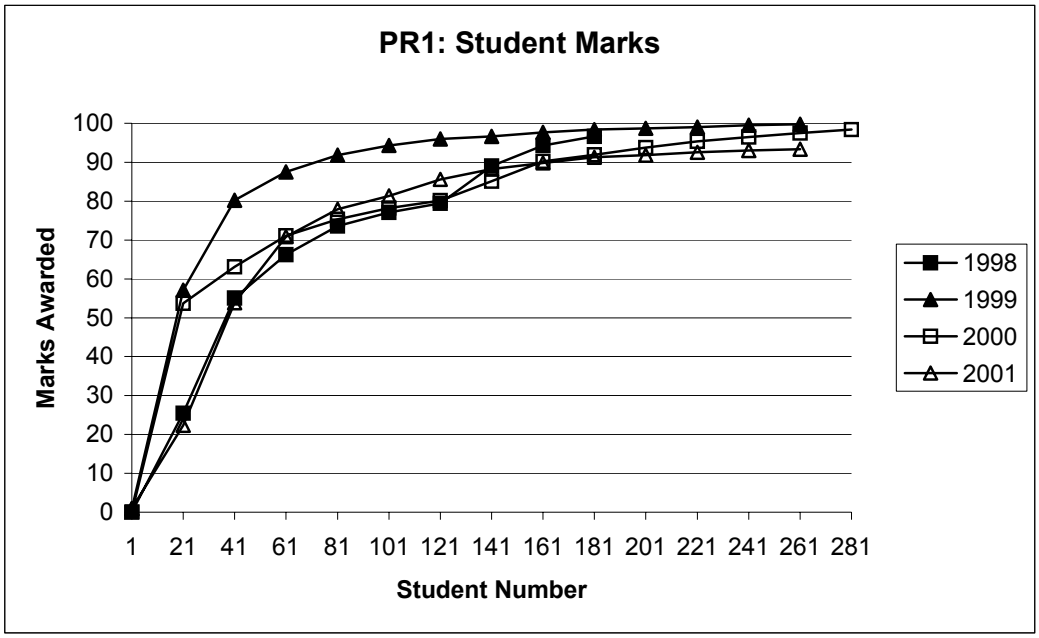


Figure 7: Submissions Used in PR1 and PR2



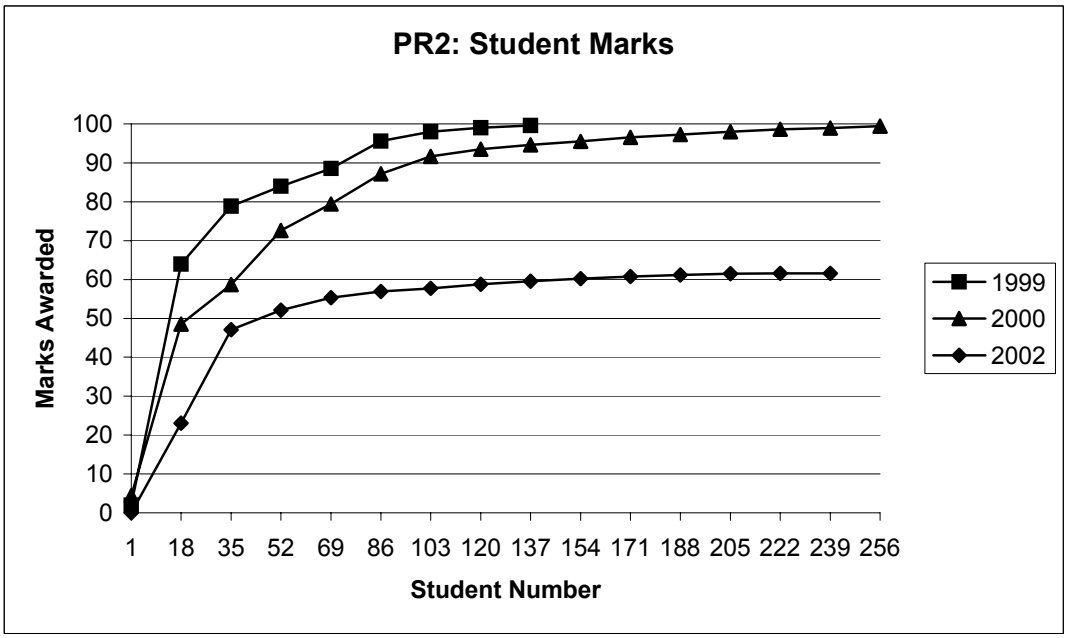
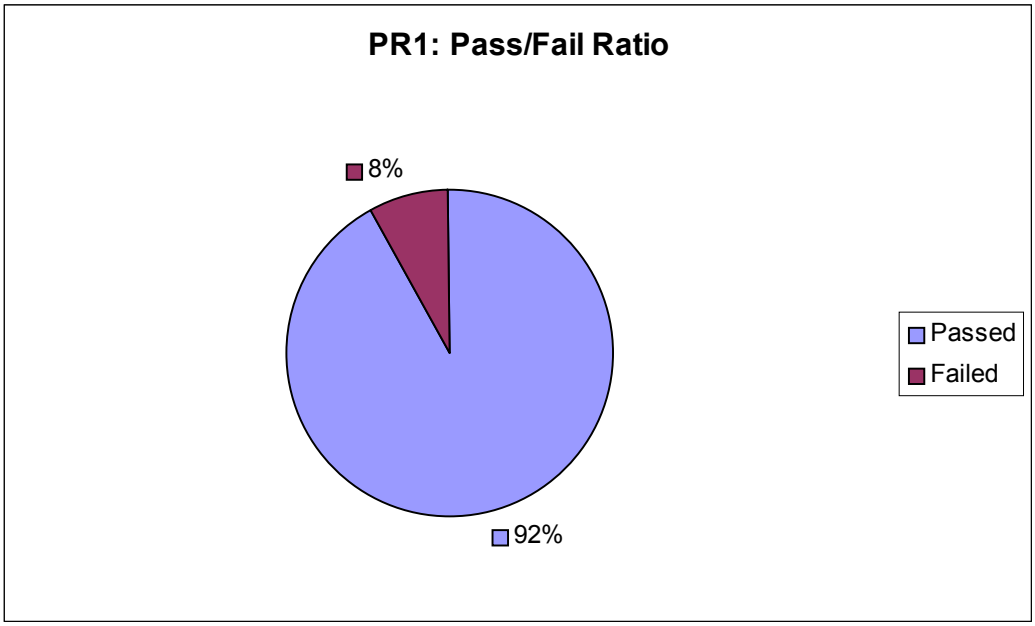


Figure 8: Students Marks achieved in PR1 and PR2



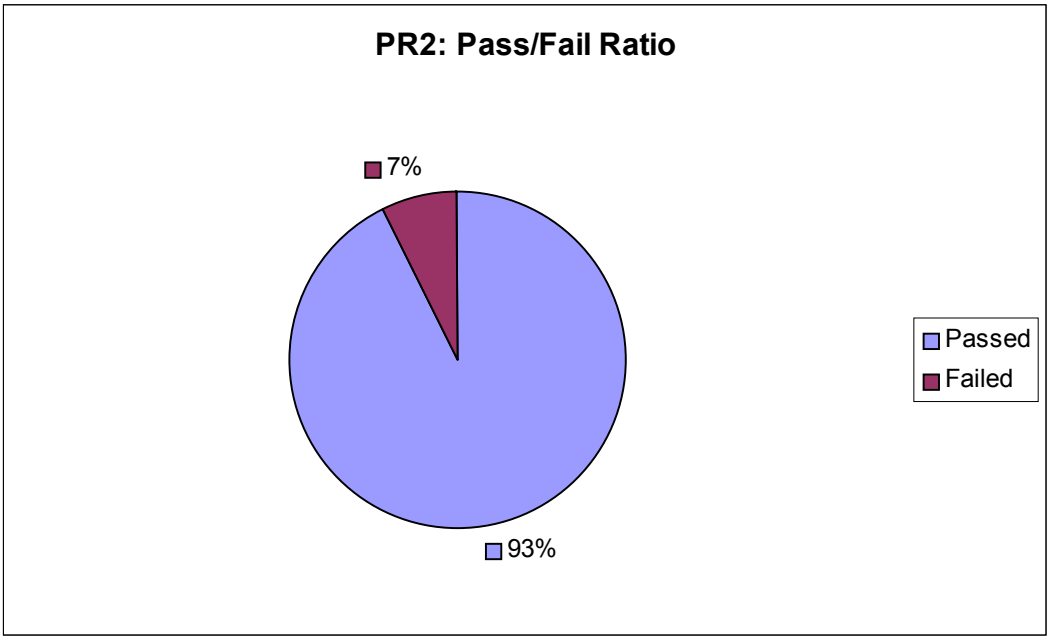


Figure 9: Pass/Fail Ratio for PR1 and PR2

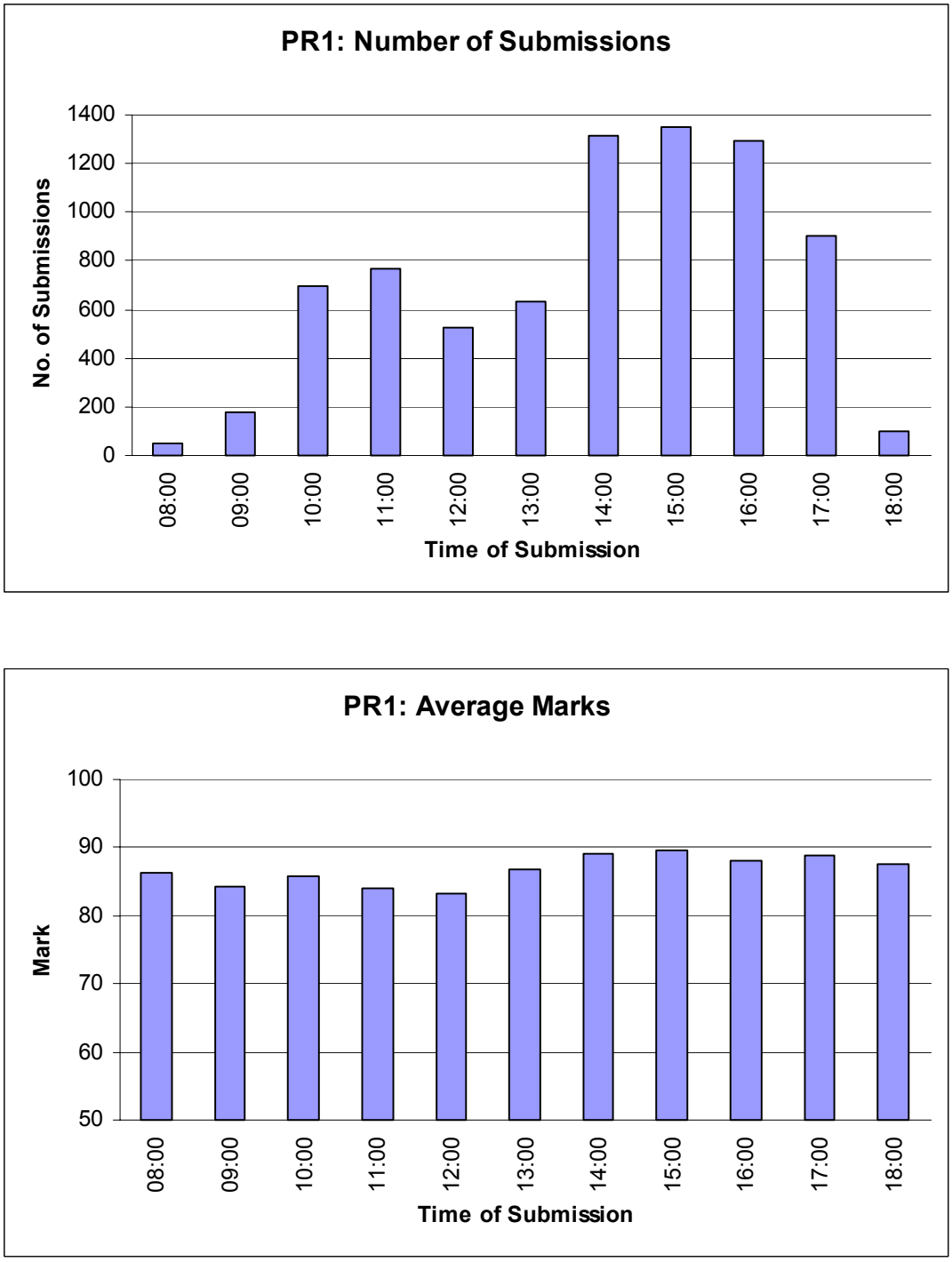
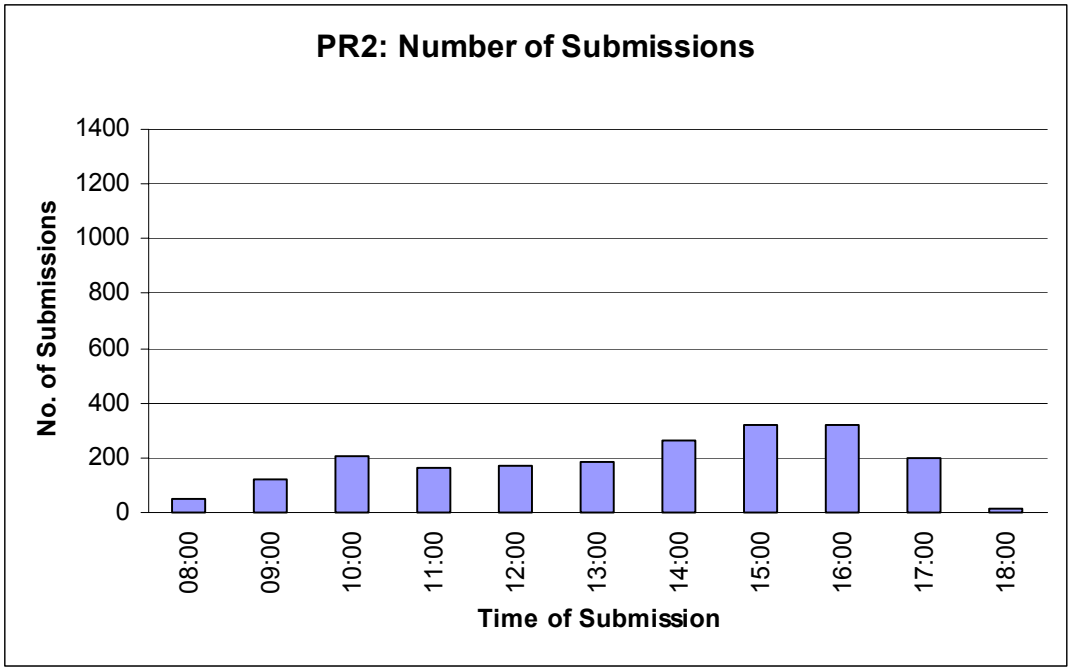


Figure 10: Students submissions and their respective marks for PR1



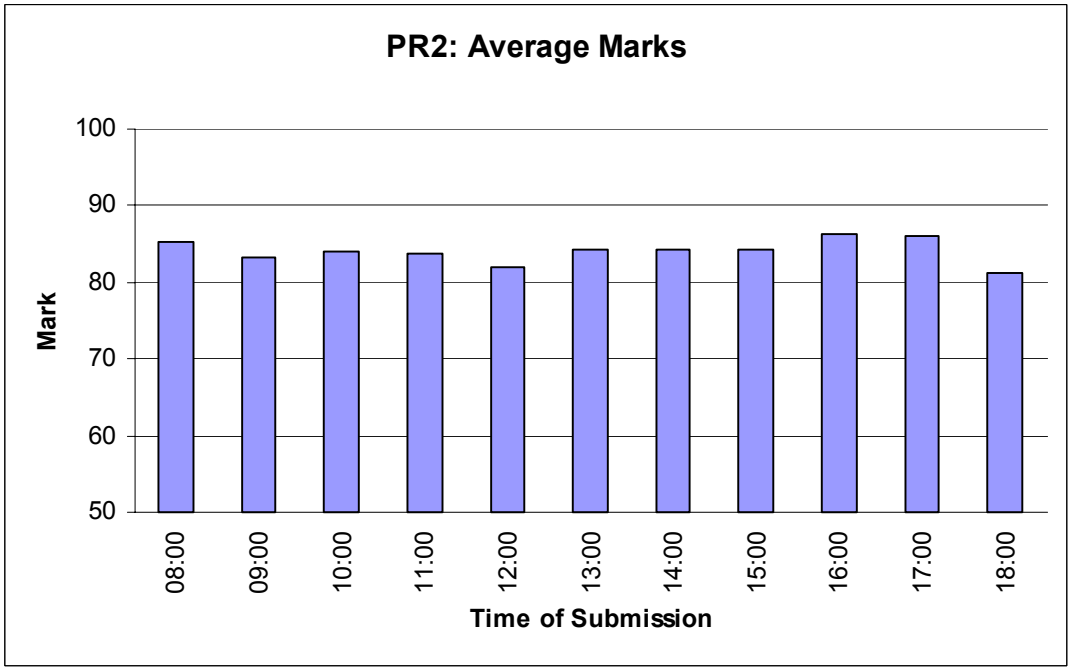


Figure 11: Students submissions and their respective marks for PR1

7.2. CourseMarker's Architecture

The decision to re-design the system using an object-oriented architecture has greatly enhanced CourseMarker's maintainability. CourseMarker has supported many changes and extensions over the three years of its use. Exercises have been customised to a much greater level than in Ceilidh. Any platform that supports Java can be used to run CourseMarker. Thorough testing under real conditions has been performed on Microsoft Windows 95, 98, Me, NT 4.0, 2000, Solaris and Linux.

Usability has also been improved. The choice of separating the system logic between the clients and servers has given great flexibility and led to the development of several clients. Over the years of running Ceilidh and CourseMarker, questionnaires were given to the students. The results indicate that students find both systems helpful in enhancing their learning experience.

Question	Result Value
Overall experience with CourseMarker	2.5
CourseMarker's interface	1.8
Easiness of learning CourseMarker	1.5
Exercises appropriate and helpful	1.65
Complexity of PR1 exercises	3.1
Complexity of PR2 exercises	2.2

Table 5: Results from the CourseMarker questionnaire

CourseMarker-specific questionnaire results suggest that all students prefer CourseMarker, mostly for its usability, user-friendliness and improved feedback. Table 5 lists an excerpt of questions of the questionnaire along with the average result values. The questionnaires' responses range from 1 to 5, where 1 is very positive and 5 is very negative (i.e. 3 is the median). The results show that students find CourseMarker easier to learn than Ceilidh. This is due to both CourseMaster's interface and CourseMaster's exercises. When asked whether automatic assessment affects the learner's level of anxiety, 72.4% of students responded that, on the contrary, they felt CourseMarker improved the equitability and fairness of assessment. In summary, student responses indicate that CourseMarker is user friendly and that the exercises were well designed, balanced and suitable for the needs of the classroom.

7.3. Automatic Assessment in CourseMarker

The flexibility of the CourseMarker marking system allows the exercise developer more freedom in expressing the specifics of the marking process. CourseMarker's marking subsystem supports extensive customisation via the use of marking programs written in Java that add properties and an improved feedback mechanism. This facility has been found invaluable in improving Ceilidh's existing exercises and fine-tuning students' learning experience.

It would be easy to assume that the exercise authoring process is harder under CourseMarker than under Ceilidh. However, this is not the case. Certainly, the process is more time consuming, but in essence no more complex. The additional effort spent during

exercise authoring benefits students. A simple exercise takes, on average, a few hours to create whereas a complex one takes up to a day. The time spent includes writing the exercise's question, its model solution and related marking files and then testing the exercise. Converting an existing Ceilidh exercise to CourseMarker takes considerably less time, however, this depends on the nature of the modifications.

Discussions have been held with students at the end of each course. The general impression given is that students find CourseMarker to be remarkably supportive. Questionnaire results regarding reliability gave average values of 2.5 for PR1 and 2.4 for PR2 respectively. These values suggest that CourseMarker has met its design goals of being a reliable and dependable automated assessment system. The students felt the system was responsive and that submission results appeared instantly on the screen (result: 1.7), in contrast with the Ceilidh system where students waited for much longer. However, some students expressed their concern for not having enough available submissions to try raising their total mark from mid nineties to high nineties. The number of available submissions for an exercise is a property set by the exercise author. Conversely, other students reported that they use CourseMarker to get a good mark and then proceed to their next assignment. When asked where they spend most of their time while solving an exercise, a scale from 1 (no time) to 5 (all of their time) was given. The students reported that their time was mostly spent on the exercise itself (3.4) rather than fine-tuning the solution to obtain a better grade (2.3). In general students use CourseMarker in a helpful and appropriate manner.

7.4. Administering CourseMarker

Over the years, questionnaires have been given to administrative and support staff. The staff firmly believes that CourseMarker is much easier to set-up and run than Ceilidh. They also report that the administrative workload has decreased since the times of Ceilidh, even if the number of students has more than quadrupled.

The monitoring of student activities has become easier with CourseMarker's web facilities. Extensive use of links makes the administrative tasks quicker to perform.

Security has been included in the design phase. Encryption between clients and servers hides the information from unauthorised users. Session key identification improves security by ensuring that users are who they claim they are.

A remote server console tool can be used to dynamically shutdown, unlink, and reload selective CourseMarker subsystems at runtime, should a reason to do so occur. This can be particularly useful if the CourseMarker developers add a feature or fix a bug in a specific subsystem. The CourseMarker servers don't have to be stopped in order for the new Java class definitions to be loaded.

Finally, the choice of archiving all the information concerning student submissions has been found to be imperative in cases of disagreements and disputes with the students over their grade and for gathering statistics on the performance of the system.

7.5. Limitations

Driving the marking process by a Java control program means that no subsequent alterations performed on the program will be loaded at runtime. The consequence of this is that the CourseMarker system needs to be restarted if the control aspect of the marking process of an exercise is modified. Nevertheless this does not affect alteration and customisation of the metrics, as these are external and are only driven by the marking program. The above is a limitation of the Java language, in which any Java class that gets loaded is cached by the Java Virtual Machine (JVM) for performance and security reasons. However, there are ways of instructing the JVM to re-load a specific class on run-time. We will be addressing this issue in the near future.

There is no specific support for multiple-choice questionnaires (MCQs) under CourseMarker. At the moment, if automatic marking is required, MCQs have to be written as small programs that the students run and answer via their CourseMarker clients. If only gathering of those questionnaires is required, then they can be written in a simple text file that will be distributed to the students via CourseMarker. The students can then complete the questionnaire using a text editor.

8. Future Work

There are a number of issues that either are currently being addressed or that will be addressed in the future. These include:

- Dynamic re-loading of the marking program.

- Support for multiple-choice and other types of questionnaires.
- Conversion of all the Ceilidh courses to CourseMarker.
- Design and implementation of a Web CourseMarker client for the students which will allow students to use the system from any web enabled computer (currently in testing phase)
- A complete on-line learning environment to provide maximum support for staff and students.
- A multimedia framework that links with CourseMarker in order to provide multimedia content to the students.

9. Conclusions

Ceilidh and its successor have proven to be invaluable to the University of Nottingham and to other academic institutions. CourseMarker demonstrates considerable improvements over Ceilidh. Its architecture and implementation satisfy the objectives of maintaining or enhancing the old functionality while increasing performance, scalability, maintainability, extensibility and usability. The changes made to the assessment and administration processes have also been successful, as they improve the expressiveness of the marking process and ease the management of courses, making automatic assessment more accessible to academic institutions.

Bibliography

Foxley E. and Mohd Zin A., (1991), *Automatic Program Quality Assessment System*, Proceedings of the IFIP Conference on Software Quality, S P University, Vidyanagar, India.

Benford S., Burke E., Foxley E., (1993a), *Courseware to support the teaching of programming*, LTR Report, Computer Science Dept, The University of Nottingham, UK.

Benford S. D., Burke E.K., Foxley E., Gutteridge N. H., Mohd Zin A., (1993b), *Experiences with the Ceilidh system,* Proceedings of the 1st International Conference on Computer Based Learning in Science, Vienna, Austria.

Benford S. D., Burke E.K., Foxley E., Gutteridge N. H., Mohd Zin A., (1993c), *Ceilidh: A course administration and marking system,* Proceedings of the 1st International Conference on Computer Based Learning in Science, Vienna, Austria.

Benford S., Burke E., Gutteridge N., Foxley E., Zin A. M., (1993d), *Experience using the Ceilidh System,* Proceedings of the All Ireland Conference on Delivering the Computer Curriculum, Dublin, September 1993.

Benford S., Burke E., Foxley E., Gutteridge N. and Zin A.M., (1994), *The Design Document for Ceilidh version 2,* LTR Report, Computer Science Dept, The University of Nottingham, UK.

Kay D., Scott T., Isaacson P., Reek K., (1994), *Automated grading assistance for student programs,* Selected papers of the 25th annual SIGCSE Symposium on Computer Science Education, Phoenix, AR USA, March 10–12, 1994, Pages 381–382.

Benford S., Burke E., Foxley E., Higgins C., (1995), *The Ceilidh System for the Automatic Grading of Students on Programming Courses,* ACM Press, Proceedings of the 33rd Annual ACM Southeast Conference, Clemson, South Carolina, March, 1995.

Foxley E., Higgins C. and Gibbon C., (1996), *An overview of the Ceilidh system 1996,* LTR Report, Computer Science Department, The University of Nottingham, UK.

Foxley E., Nobar M. P., Tsintsifas A., (1997), *Ceilidh and the World Wide Web,* Proceedings of the 3rd International Conference on Computer Based Learning in Science, Leicester, UK.

Jackson D., Usher M., (1997), *Grading student programs using ASSYST,* Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education, San Jose, CA USA, February 27- March 1, 1997, Pages 335–339.

Charman D. and Elmes A., (1998), *Computer Based Assessment : A guide to good practice,* Volume I, University of Plymouth.

E. Foxley, C. Higgins and A. Tsintsifas, (1998a), *The CEILIDH System: a general overview*, Proceedings of the 2nd Annual CAA Conference, Loughborough 2-4 July.

Foxley E., Higgins C., Symeonidis P. and Tsintsifas A., (1998b), *Security Issues under Ceilidh's WWW Interface,* LTR Report, Computer Science Department, The University of Nottingham, UK.

Joy M., Luck M., (1998), *Effective electronic marking for on-line assessment,* Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on changing the delivery of computer science education, Dublin, Ireland, August 18 - 21, 1998, Pages 134–138.

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

SUN Microsystems, (1999), *The JAVA HotSpot Performance Engine Architecture*, White Papers. (http://java.sun.com/roducts/hotspot/whitepaper.html).

Jackson D., (2000), *A semi-automated approach to online assessment,* 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, 11 – 13 July, 2000, Pages 164-167.

Korhonen A., Malmi L., (2000), *Algorithm simulation with automatic assessment,* 5th annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science and Education, Helsinki Finland, July 11-13, 2000, Pages 160-163.