Early experiences of computer-aided assessment and administration when teaching computer programming

Steve Benford, Edmund Burke, Eric Foxley, Neil Gutteridge and Abdullah Mohd Zin

Department of Computer Science, University of Nottingham, University Park, Nottingham NG7 2RD

Abstract

This paper describes early experiences with the Ceilidh system currently being piloted at over 30 institutions of higher education. Ceilidh is a course-management system for teaching computer programming whose core is an auto-assessment facility. This facility automatically marks students programs from a range of perspectives, and may be used in an iterative manner, enabling students to work towards a target level of attainment. Ceilidh also includes extensive course-administration and progress-monitoring facilities, as well as support for other forms of assessment including short-answer marking and the collation of essays for later hand-marking. The paper discusses the motivation for developing Ceilidh, outlines its major facilities, then summarizes experiences of developing and actually using it at the coal-face over three years of teaching.

Introduction

Assessment and feedback are key aspects of the educational process. This is particularly true when learning practical skills where hands-on experience is of critical importance. Computer programming is one such skill. Indeed, like riding a bicycle or playing the piano, it is not possible to learn to program a computer without extensive practical experience. However, computer programs are also notoriously difficult to assess by traditional methods such as hand-marking printed output. Firstly, it is virtually impossible to tell by visual inspection of a program source whether it will run at all, let alone whether it will run as required. Detailed assessment using print-outs is simply out of the question. Secondly, program development is an iterative process, often involving many cycles of development and testing. Hand-marking is too slow to help in this process. These problems are exacerbated by increasing student

numbers, particularly on first-year programming courses where class sizes in excess of a hundred are common. A dedicated lecturer teaching such a course would be hard pressed to manage one iteration of one exercise a week, even if he or she had little else to do. One possible solution to this problem may be for a group of people (for example, postgraduate assistants) to mark the solutions. However, this often leads to inconsistencies in the level of marks awarded, and drastically increases the administrative burden.

Of course, such problems are not confined to the realm of computer programming, and there are a number of more general problems with hand-marking paper scripts. Scripts can easily get lost, detached or damaged. Marking on paper is inappropriate for most computer-based learning (not just for programming). Administration of assessment is a time-consuming process, made more difficult by having to deal with large quantities of paper. Finally, there is increasing pressure to save paper. Our solution to these problems has been to explore the role of the computer in directly assessing students' programs. To this end, we have constructed a system called *Ceilidh* based on an automatic assessment mechanism which tests programs from several perspectives, including dynamic correctness (i.e. do they work?) and programming style and complexity (i.e. are they elegant solutions?). Ceilidh also supports course administration by providing facilities for setting exercises, handing in all types of work online, handing out solutions, dealing with questions, and tracking and summarizing progress.

The idea of automatically assessing computer programs has direct parallels in industry where automated test-harnesses and software quality-control environments are becoming increasingly common. Thus, Ceilidh provides students with valuable experience of working in a quality controlled environment. Ceilidh also aims to support the administration and assessment of a range of courses beyond programming. To this end, we have begun experimenting with other complementary assessment techniques, including multiple-choice and short-answer marking.

The first version of Ceilidh was developed in 1988 and used to support a C programming course. The second phase of development was funded through Nottingham University's Enterprise in Higher Education (EHE) initiative, and resulted in Ceilidh being used to support a C++ course for a class of 160 in the 1991/92 session. At present, Ceilidh is at the centre of a UFC Teaching and Learning Technology Programme (TLTP) project – *Courseware for the Automatic Assessment of Programming* – which aims to pilot the system at over 30 departments of Computer Science throughout the UK in the next three years. Ceilidh has also been installed at sites in North America, Australia, Belgium, India, Malaysia, New Zealand, Portugal, Russia and Spain. Beyond this, Ceilidh should be of interest within the many other disciplines involved in teaching computer programming (particularly in engineering and science) and also computer centres which are often heavily involved in service teaching. The administrative side of Ceilidh might also interest a much broader range of people.

Overview of Ceilidh

Let us begin with an overview of the functionality of the Ceilidh system, starting with a high-level summary of the functions available, and moving on to a more detailed examination of the steps involved in setting up, completing and marking a programming exercise. The logical structure of the Ceilidh system is summarized in Figure 1.

Ceilidh can support a number of on-going courses. Each course is divided into units representing different chapters or topics. Each unit may include a number of exercises which may be programming exercises, short-answer questions or essay questions. Of these,

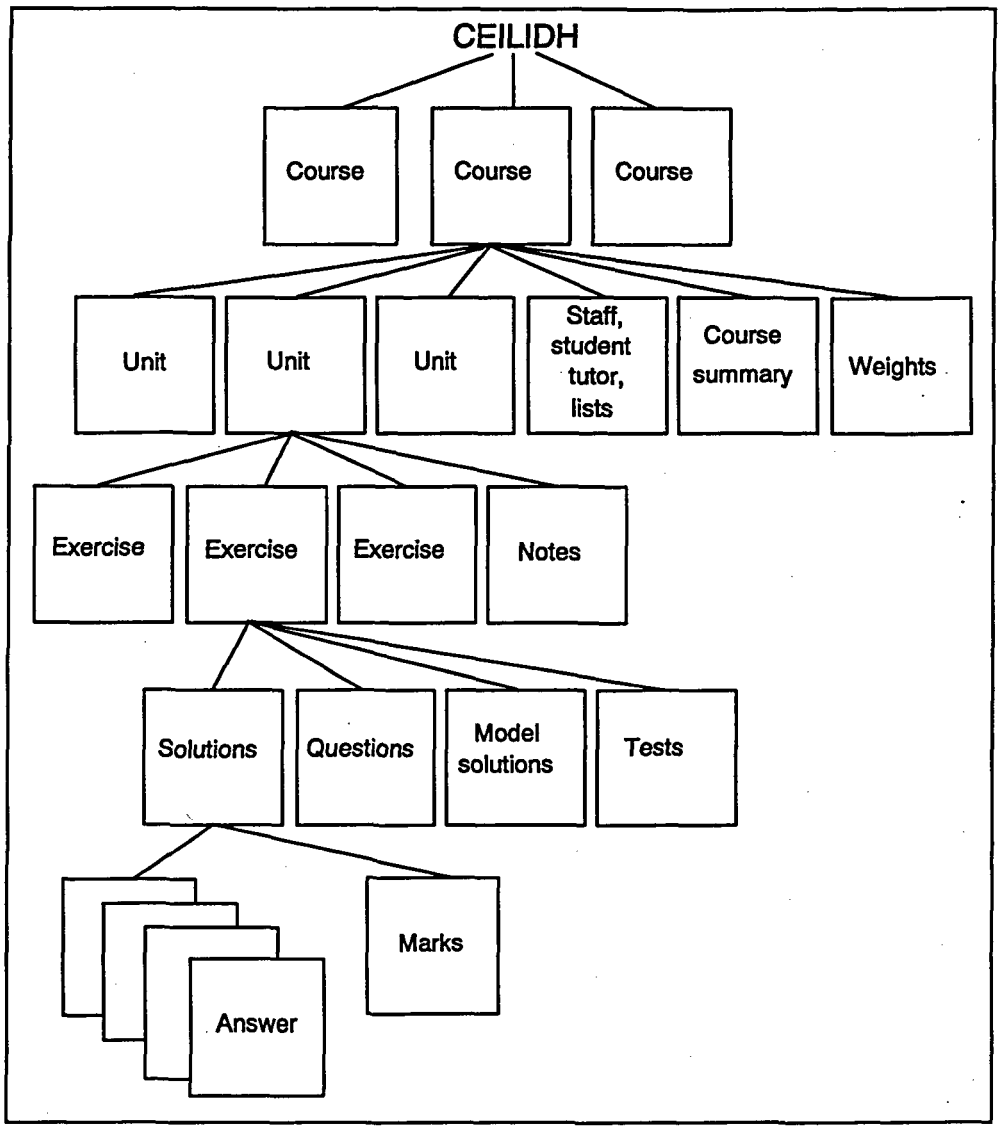

Figure 1: The Ceilidh information structure

programming and short-answer exercises are supported by automated assessment facilities. Essay-style exercises include facilities for online collection of work for later paper hand-marking.

The overall functionality of Ceilidh is categorized according to four different classes of user: students, tutors, teachers and course developers. Students use the system to obtain, complete and mark work, and to access course resources such as notes, work and lecture schedules and even the teachers. Tutors represent teaching assistants, and are provided with additional

facilities to inspect work and to summarize the progress of individuals or groups of students. Teachers are provided with facilities to administer entire courses. Course developers can amend the course information including notes, exercises and test data. We now consider each class of user in turn.

Student facilities

Ceilidh provides the following key facilities to students:

(i) It allows students to view general temporal course information such as hand-in times for coursework and more permanent information such as lecture notes. This information can be viewed on-line or to be printed.

(ii) It provides access to the questions set in exercises.

(iii) It offers outline program source (skeleton programs to be completed by students), and associated modules and header files for exercises where appropriate, to assist in the solution of programming exercises. For an essay exercise, an outline of the main sections/heading expected in the essay may be given.

(iv) It allows students to edit, compile and test-run their computer programs. The extent to which compilation details are hidden from the student is determined by the teacher.

(v) The students can submit their work to the system. If the exercise is programming, the system will mark their program. A summary of the marks is made available to the students to help them assess their program quality (and a copy of the program and of the marks awarded is retained centrally for later reference). Marking can take place many times providing an iterative process of development and assessment. If the exercise is an essay, the system simply stores a copy of the solution.

(vi) It allows them to view a model solution, to run this solution, to view test data and to run both their own solution and the model solution against the test data. The model solution can be seen only after the deadline for submission has passed.

(vii) It allows them to comment on a specific exercise or on the system as a whole. Comments are stored for later browsing by teachers.

(viii) It offers help facilities, including an overview of the marking metrics employed by the system and a good programming style guide.

Tutor facilities

Ceilidh provides tutors with the following additional facilities:

(i) List details of the work submitted by all of the tutor's students, or by any named student. For each item of coursework, the listing gives a summary of the marks awarded and the time at which it was submitted, including whether it was early or late. Details of the work, such as the program source code and a more detailed breakdown of the marks, can be inspected if requested.

(ii) List the names of students who have not submitted work, or who have submitted late.

(iii) List the marks awarded to students for a particular exercise.

(iv) Summarize the average marks across all the exercises on a given course.

Teacher facilities

Ceilidh provides teachers with the following additional facilities:

(i) Declare certain exercises to be open. These are the exercises on which the student will be expected to work.

(ii) Declare certain previously opened exercises to be late. Students submitting after this date will be warned that their work is late. For each exercise made late, the teacher will be invited to request plagiarism tests, and overall class software metrics. The latter is particularly important to enable the teacher to keep in close touch with the class progress, and its strengths and weaknesses.

(iii) Declare certain exercises to be closed. Students will no longer be able to submit work for these exercises.

(iv) Teachers can browse and respond to the students' comments.

(v) Set weighting and scaling factors so as to calculate final assessments from marked exercises in any desired way.

Course developer facilities

Ceilidh provides course developers with the following additional facilities:

(i) Create new coursework, or amend existing coursework. The system prompts the user to ensure that all the necessary data items have been input (see below).

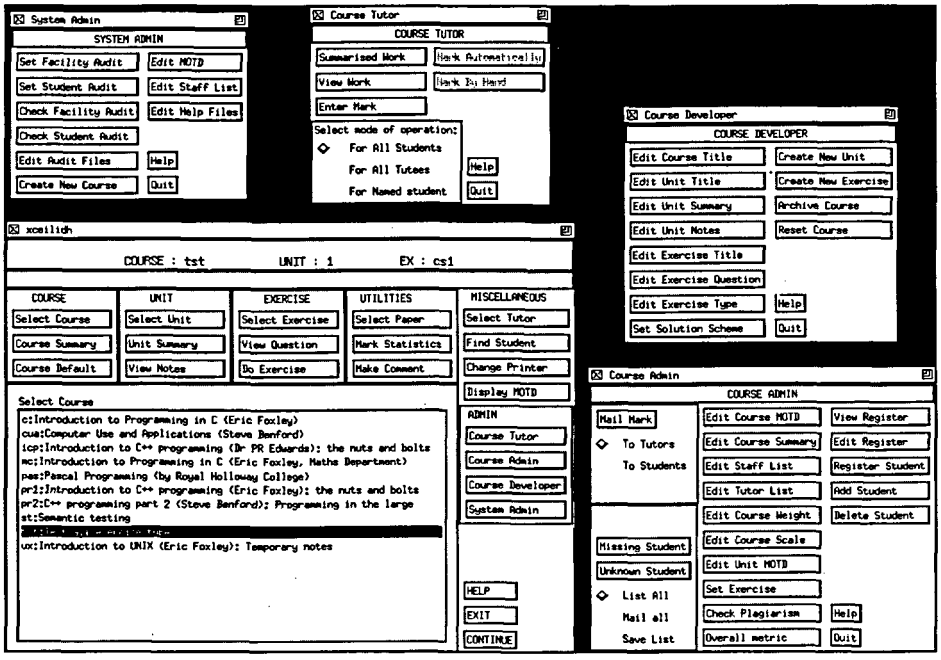

Figure 2: Ceilidh X-Windows interface

(ii) Set up new courses and manage registers of tutors and students. The various different roles within the Ceilidh system are illustrated in the screen dump of the X-Windows interface in Figure 2.

Completing exercises

Let us consider in more detail the steps involved when a student completes a specific exercise. We will consider programming exercises as these currently form the core of Ceilidh's work.

The student selects the exercise to be tackled (a default is usually set by the teacher) and reads the specification (the programming term for the question). Specifications form a reasonably complete description of the problem to be tackled, and may vary greatly in complexity and difficulty. Figure 3 shows a student viewing an early C++ exercise, and the skeleton file provided.

Later in the course, specifications may run to several pages, including descriptions of C++ class definitions to be implemented or used, and details of exception states. The goal at this stage is to develop the students' ability to work to specification. More advanced tasks might typically involve implementing a C++ class to work with a given interface program, writing a program to use a given class, or writing a program which combines several given classes. For example, the following specification might be given later on in our C++ course:

> A telephone book consists of an arbitrarily long collection of entries where each entry contains a name and a number. Users can add new entries, delete entries, look up a number belonging to a specific name, and produce an alphabetical listing of the whole telephone book. A telephone book can be represented by a 'Phonebook' Abstract Data Type which could be implemented in C++ as a 'Phonebook' class.
>
> The following is an outline class definition for a simplified 'Phonebook' class.
>
> ---
>
> Outline C++ class definition here
>
> ---
>
> The following details should be carefully noted:
>
> * A number of status codes are defined to be returned by Phonebook methods.
>
> * Each person is limited to having a maximum of one entry (meaning that a person's name can only appear in one entry).
>
> * A name cannot be the empty string.
>
> * A phone number must consist of digits in the range 0-9 inclusive, and must be exactly NUMBER_LENGTH digits long.
>
> * The overloaded < operator produces a list of all entries currently in the Phonebook. This list should be alphabetically ordered by name.
>
> The status codes have the following meanings:
>
> OK – operation completed successfully.
>
> BAD_NAME – the supplied name has the wrong format.
>
> BAD_NUMBER – the supplied number has the wrong format.
>
> ALREADY_EXISTS – an entry for this person already exists.
>
> DOES_NOT_EXIST – there is no entry for this person.

From this example, one can deduce that the current emphasis of Ceilidh is on encouraging students to produce correct programs from existing specifications. Ceilidh may offer students varying levels of support for developing their programs including skeleton implementation files which provide common definitions and outline code, invocation of editors, automatic compilation and linking and even automatic running of their program against test data. The level of support provided can be tailored by the teacher throughout the course. Thus, at one extreme, Ceilidh can provide a fully integrated implementation environment which completely shields the student from the details of the operating system, and at the other the student can be exposed to all the details of compiling and linking (in our case at the Unix operating-system level). A half-way house is also possible where Ceilidh invokes the necessary operating-system commands but also echoes them back to the student. At any stage throughout development, students can request the system to mark their program. A variety of standard software metrics are used, combining dynamic and static analysis techniques – these are more fully described in Zin (1991) and Benford (1993).

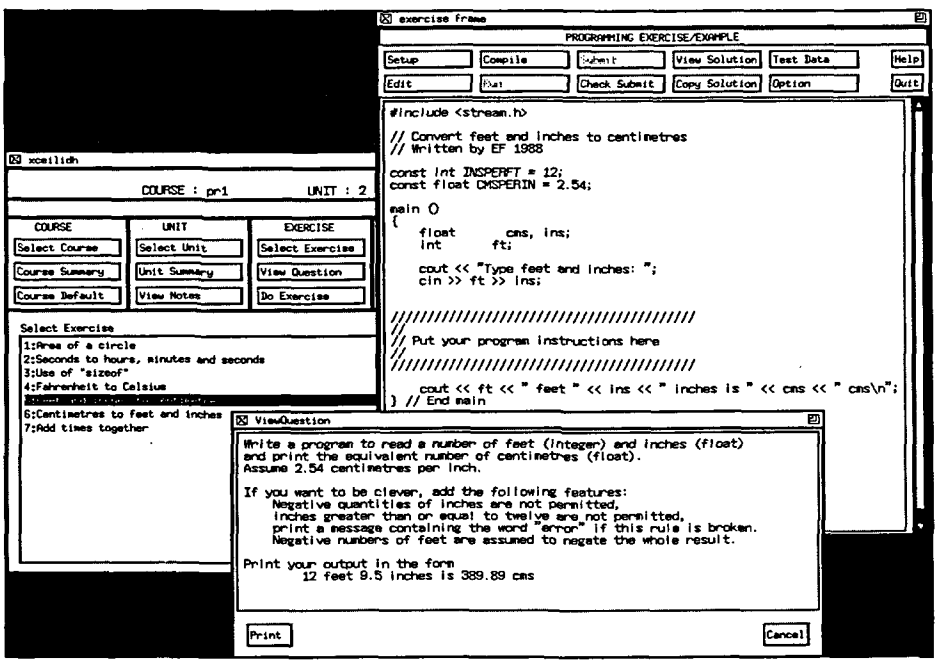

Figure 3: Student viewing a question and skeleton program

A summary of the marks is made available to the students to help them assess their program's quality, and a common question concerns how the system helps students to identify the problems with their program given the relatively terse summary of marks. The answer is that Ceilidh does not try to identify what a problem is in detail – it only reports that there is one and roughly what it is. The student is then expected to go back to his or her program and figure out what is going on, usually by more extensive testing on his or her own. This seems quite satisfactory, since the goal of the system is to aid the learning process, not to provide an instant solution. However, we recognize that determining an appropriate level of feedback is a

difficult issue, particularly in relation to the tightness of the specification and the generality of the test data. Ceilidh leaves all these factors configurable to some degree.

A key feature of Ceilidh is that students can re-mark their work as many times as they like, although a teacher may optionally impose a minimum time-delay between markings. The system maintains a record of the full mark history for each piece of work, including the date and time of submission. A copy of the current program is also held for later analysis. On submission, the system notes whether the coursework is late and, if it is, asks the student if he or she still wishes to go ahead with submission. It also asks if he or she wishes to comment on the mark. Short-answer exercises are different in that the student is presented with an on-line test in the form of a set of short questions. Ceilidh then reads simple answers in the form of words, phrases or maybe whole sentences, and matches them against keywords supplied by the teacher. Marks are recorded by the system.

Essay exercises involve the student reading the question and perhaps being given a skeleton essay to complete which can be submitted back to the system when ready. The teacher may then hand-mark the essays and enter the marks back into Ceilidh for subsequent processing.

Setting up exercises

In order to set up an exercise, the course developer needs to provide a number of files:

(i) A question/specification.

(ii) A working model solution or model answer.

(iii) A file of test data for each of the dynamic tests to be carried out in programming exercises. Test data may either take the form of raw data to be input to the program, or may take the form of a Unix shell script which can be used to drive the program in a more flexible way.

(iv) A file of 'keywords' for each test. These are matched against the program output for programming exercises and against the students' responses for short-answer exercises.

The keywords are actually Unix regular expressions to be matched against the program's output. The use of regular expressions offers a high degree of flexibility in making this comparison. For example, the expressions:

ft|feet

ins|inches

[Ee]rror

might appear in the keywords for a distance conversion problem. Once again we can see that the teacher needs to strike a careful balance between tightness of specification and flexibility of testing.

Finally, the teacher needs to provide a mark-weighting file which assigns a name and relative mark-weighting to each test involved. Such a file for a simple distance problem might be:

15 Simple test

15 Simple test

20 Check "feet" "ins"

25 Inches > 12

25 Negative inches

The teacher may also configure the relative weighting of style and complexity tests if required, and may even alter the relative weightings of the various software metrics used.

Monitoring progress

Both teachers and tutors have the ability to track the progress of individuals or groups of students. This is supported by a statistical summary package within Ceilidh. Three kinds of summary can be produced.

(i) A student summary shows the progress of an individual student over a given course. This includes displaying his or her mark for each exercise along with the class average. It also includes the number of submission attempts per exercise, thus providing some indication of how hard the student found the exercise.

(ii) An exercise summary displays the distribution of student marks for a given exercise, including the number of submission attempts per student.

(iii) A course summary displays the average marks across the whole course, as well as the average number of submissions across the whole course.

These facilities are available to tutors and teachers. Students are encouraged to obtain a stripped-down personal summary of their own progress across a course. Summaries may be presented as tables or as graphs which may appear on-screen or be printed. Facilities are also

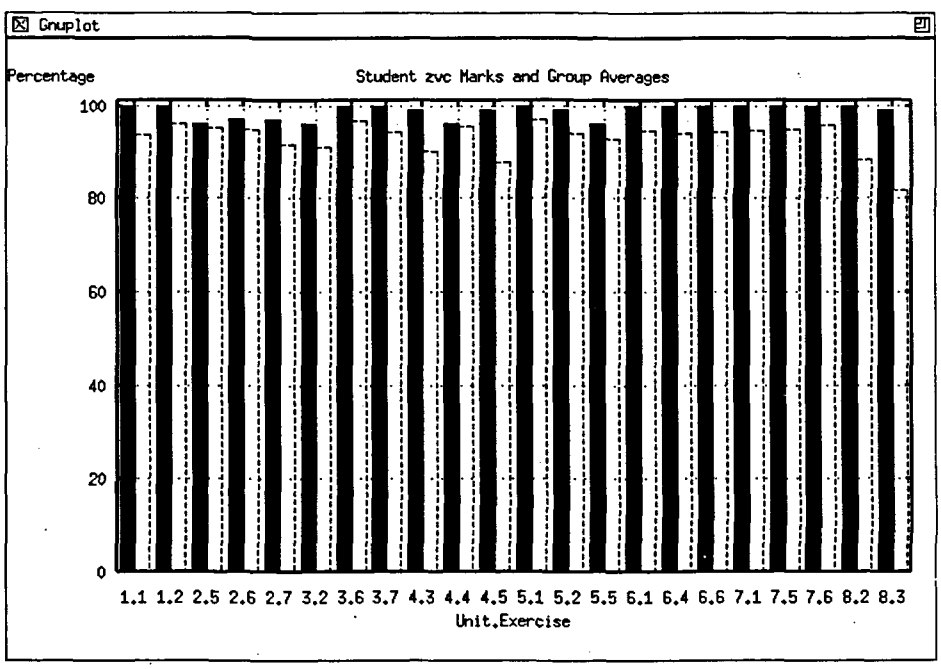

Figure 4: Student summary – mark (solid) against class average (dashed) for all exercises

provided to batch-print summaries for each student on the course. For example, at Nottingham, we have established the practice of mailing summaries to all tutors on a fortnightly basis. Figure 4 shows a typical student summary.

The plagiarism detection tool provides a quite different kind of summary. This facility takes all the solutions for a given exercise and compares them, looking for similarities. It reports both near-misses and what it believes to be exact matches. The facility is quite sophisticated in that it first removes all personalized aspects of each program (for example, comments and identifier names) in order to arrive at a core structure before making the comparison.

Final assessment

The marking described so far is used to provide instantaneous feedback to students and staff, but several additional issues have to be considered if the system is to be used for formal assessment. First comes that of weighting. When computing a final mark for a course, it may not be sensible to give all exercises the same value (for example, early exercises are often worth less than later ones). Secondly, the use of Ceilidh tends to result in high marks, often above 80%. It is therefore necessary to scale marks to arrive at a final assessment in order to comply with institutional marking conventions (for example, a 60% average in Ceilidh may represent a basic pass on the course). Finally, there is the issue of deadlines for exercises and the problems of dealing with late work. Ceilidh deals with these issues by letting the course teacher set up two files, the weights and scale files. The weights file lists each exercise along with an indication of its relative weighting and also its current status. The status may be open so that work can be submitted; late so that work can be submitted but is noted as late; or closed so that nothing can be submitted. The weights are used to automatically calculate a weighted average course mark for each student. Teachers are provided with menu options to set the weights and alter the status of exercises. A separate scale file allows a teacher to define a scale mapping between raw marks obtained and marks on some final scale of assessment.

Implementation

In this section we very briefly touch on some implementation issues. The current version of Ceilidh runs under Unix and several of its derivatives. It provides three styles of user interface:

(i) A line interface for dumb terminals.

(ii) An X-Windows interface.

(iii) A command interface, where each Ceilidh facility is available as a Unix command. This allows people to build their own interfaces on top of Ceilidh, and to integrate it with other applications.

Work is also currently underway to port Ceilidh to a networked PC environment and to provide a PC Microsoft Windows interface.

Experiences with Ceilidh

Ceilidh was initially conceived and constructed not as a research project but out of the necessity of having to teach programming to large first-year classes. Consequently, Ceilidh has always been used to support real programming courses, and has evolved to meet the needs

of these courses. The fact that Ceilidh was constructed while in actual use gave rise to many difficulties, particularly in the area of managing rapid updates to a live system. However, the great benefit of this approach is that instead of being an experimental prototype briefly demonstrated in the research laboratory, Ceilidh is a fully working system that meets a real need and that has evolved to meet the needs of teachers and students.

Sources of feedback

Here we summarize our experiences of building and using Ceilidh. This summary does not represent the results of a formal statistical/experimental evaluation of the effects of the system on the educational process. Instead, it encapsulates several years of feedback, observation and opinion from a wide range of sources. These include:

(i) Our own subjective view as teachers and implementors.

(ii) Analysis of user-questionnaires. Ceilidh contains questionnaires about the system and the courses supported, and these are made available online at the end of each course.

(iii) Archives of the many student comments and questions submitted through the online comment facility.

(iv) The marks obtained across three years of use.

(v) Experiences of discussing Ceilidh in examiners' meetings.

(vi) Feedback from other members of staff passing on student comments from tutorials.

(vii) Feedback from other organizations involved in piloting the system worldwide.

(viii) Student essays discussing their opinions of the system. More specifically over 100 students recently answered the following essay question:

Does the Ceilidh system for teaching C++ produce good programmers who can write high-quality, thoroughly tested and stylistically correct programs, or does it merely produce students who have learned how to 'play the system' and who effectively rely on it to solve their problems for them?

Impact on students

We begin this section with observations on the ways in which Ceilidh has affected different kinds of student:

(i) The system has acted as a confidence builder for novice programmers who benefit greatly from the kind of positive early feedback that arises from the early simple exercises. One particular aspect of this has been building the confidence of female students who may often initially feel intimidated by the 'macho' image associated with programming.

(ii) The system has enabled us to spot the really weak students early on, enabling us to focus effort on helping them.

(iii) Nearly all the critical feedback has come from experienced students, particularly over the marks assigned. The notion of good programming style was a particularly contentious issue, and we had several discussions on the issues of style and

standardization of layout. This turned out to be beneficial as it is the self-taught 'experts' who often need the most help in this area (even if they think they do not).

This last observation is of key importance. One of the most surprising and pleasing aspects of Ceilidh was its role in consciousness-raising. The provision of immediate feedback by a machine produced much more discussion of programming correctness and style than did previous hand-marking. Ceilidh's online comment facility played a key role in this discussion, and one of the early extensions to the system was to offer the students the opportunity to comment on each mark when given. Consequently, our experience of automatic assessment is that far from reducing contact with students, the quantity and quality of discussion is increased. Looking back, we suspect that this effect stems from a number of factors. Firstly, immediate feedback means that marks are received while the problem is uppermost in a student's mind. Secondly, students may be happier to argue with a machine than with a teacher. Thirdly, the marking process is generally more open to inspection than with hand-marking (for example, style rules are published and are applied consistently). As a further comment, hand-marking of any form of coursework can lead to a student being treated less fairly than others. For instance, coursework marked by more than one person will lead to inconsistencies in marks awarded due to differing ideas of what the correct answer should be. This coupled with other problems such as racism, sexism and favouritism can lead to certain students achieving poorer or better marks than they deserve. We believe that such explicit discrimination is reduced, if not eliminated, by the use of the Ceilidh system, since it marks each solution consistently. However, we recognize that implicit discrimination through inappropriately chosen questions must still be guarded against.

Of course, the introduction of Ceilidh was not without its problems and we did observe a number of more negative reactions to the system. One problem-group came to be known as the Perfectionists. These students seemed unable to stop working on an exercise even when a satisfactory mark had been obtained. A second problem-group was that of Gamblers: students who iterated around the marking cycle many times, tweaking their programs in an attempt to pick up extra marks without necessarily thinking through the problem at hand. Perfectionists and Gamblers resulted in our making two further extensions to the system. The first was to introduce a minimum time-delay between markings during which time a student could not re-mark his or her program. This delay is tailorable by the teacher, and may range from a few seconds to several days. A more general advantage of this extension is that it allows Ceilidh marking to be used in a once-only fashion if the teacher wishes. The second extension was to stretch the progress-monitoring facilities to spot the problem-students. This involved three new facilities:

(i) Producing graphs of number of marking attempts per student across an exercise or course.

(ii) Producing graphs of the 'development profile' for each student completing an exercise. The development profile charts the mark obtained against each marking attempt and so visually shows progress across an exercise.

(iii) Calculating and producing graphs of a 'development ratio' for each student and exercise, a numerical measure that indicates whether the student has been making progress.

The development ratio is calculated as the ratio of the number of times a student's mark increases minus the number of times it decreases divided by the total number of marking attempts. This results in a scaled numerical measure in the range +1 to -1 which shows general progress across an exercise. At the extremes, +1 indicates the student's mark always increased on each successive re-marking, and -1 that it always decreased. A development ratio of 0 would indicate that no progress was made, and one close to 0 that little progress was made. Ceilidh automatically calculates these ratios for all students and exercises, and can display them graphically. As an example, Figure 5 shows the development profile (i.e. mark awarded plotted against the submission attempt) for an individual doing a single exercise. In this case, we see a steady improvement in the mark awarded with each attempt until nearly 100% is reached.

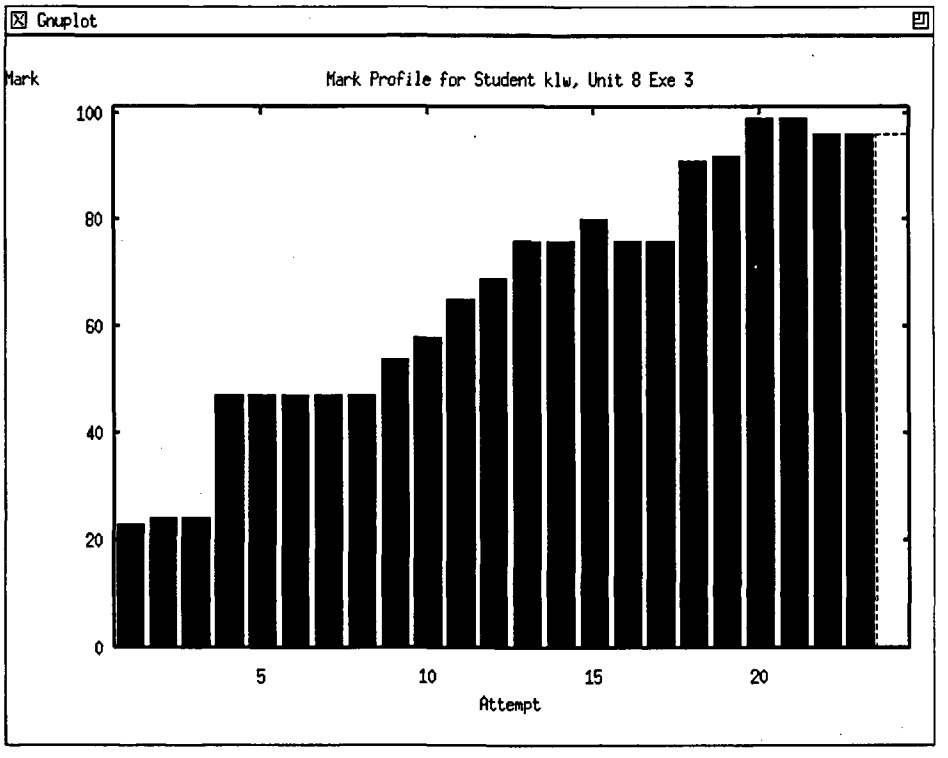

Figure 5: Example graph of progress on an exercise

We emphasize that the development profile and ratio provide only a rough general guide to progress, not a certain measure. Furthermore, graphs have been used by staff to spot potential problems and have not counted towards final marks. Graphs provide teachers with a rough guide to potential problem-students (for example, Perfectionists or Gamblers) who might need additional teaching support. Teachers might then encourage these students to change their working method, either by learning better to manage their work according to sensible quality targets, or to think more about the problem at hand before changing their program. In more general terms, the cases of Perfectionists and Gamblers both highlight Ceilidh's role in

encouraging students to manage their own work in an effective way. We believe that this is possible because Ceilidh devolves greater responsibility to students for determining final marks, and therefore for managing their work load.

Before passing on, we should briefly mention one hidden problem. Ceilidh requires that students hand in their work in electronic format. Indeed, at Nottingham we currently insist that nearly 75% of all core first-year work is handed in online (not just programs). This makes it difficult for students to work with pen and paper in their own residences. Of course, many students own their own computers and are able to transfer their work across to our system. However, we fear that those who do not may be disadvantaged. We believe that the solution to this problem quite simply (in technical if not financial terms) lies in the provision of a better computing infrastructure. In particular, we would like to see all our students armed with personal or notebook computers within the next few years.

Impact on teachers and tutors

So far, we have considered Ceilidh's effect on the students. We also need to consider its effect on the staff. The clear and simple benefit to teachers has been a massive reduction in time spent marking students' programs. Ceilidh has also helped with more efficient administration of courses. Collection and collation of work has been trivial, deadlines are published in advance and are stuck to, and marks are returned to examiners on time. Ceilidh also helps with the trivial, but often annoying, aspects of coursework such as making sure that work is clearly labelled and legible. This was particularly noticeable where the collection and collation of essays was concerned. Clearly some of these issues extend to courses other than programming.

The general progress monitoring facilities within Ceilidh have been of benefit to both tutors and teachers and have allowed us to keep track of our students. As an example, some tutors actually hand out progress charts to students during tutorials to confirm progress on the course. Interestingly, far from viewing this as a Draconian regime, as some people feared, many students seem to appreciate regular confirmation of satisfactory progress.

Use in formal assessment

Ceilidh is used both to give feedback to students and for formal course assessment. The current first semester programming course at Nottingham is entirely coursework-assessed and the second is 50% coursework-assessed with the other 50% being a formal written examination. This dual use of Ceilidh has given rise to several interesting issues. Not only does the iterative use of auto-assessment tend to result in very high raw marks (with marks around 80% and 90% being common) but also the marks tend to be far more tightly grouped (often the case in continuous assessment). Even with the scaling facilities mentioned previously, it may be difficult to categorize the students according to an 'expected' normal distribution. This raises the issue of the relation between feedback and assessment. Indeed, it is obvious that Ceilidh, by providing interactive feedback, is increasing the likelihood of obtaining high marks. One solution is to re-mark the students' work at the end of the course using different criteria (for example, using different and harder tests and more rigorous style checking). However, we then encounter the problem that students feel cheated because the system has been reporting excellent progress which may not match their final mark. In general, this issue relates to whether we are seeking normative or criterion assessment of students. So far we have used Ceilidh on first-year foundation courses where criterion

assessment is a sensible approach (i.e. if you can demonstrate the necessary skills, you can pass the course). The role of Ceilidh, and indeed of any iterative auto-assessment technique, may be limited where normative assessment is required (i.e. on courses in later years).

The issue of students feeling cheated is also an interesting one. There is a danger that Ceilidh might encourage students in the belief that the ability to pass a few tests makes their programs certifiably correct. Of course, this is not true, and it is recognized that testing, no matter how thorough, does not prove program correctness. It is therefore important that teachers carefully introduce the notion of program correctness and its relation to testing, particularly with respect to other approaches such as the use of formal methods in program verification.

Use at other sites

So far, Ceilidh has been installed at 25 sites outside Nottingham and has been used to teach courses in at least five of these. Styles of use have been quite diverse. More than one site has informed us that they installed the system and ran our course directly as given, even using our slides for lecturing. Other sites found it more difficult to accept the given course structure but were happy to use the exercises to form their own course. At the other extreme, one site used the Ceilidh system as an envelope to write an entirely new course for a different programming language (Pascal). This also necessitated slotting their own auto-assessment module into the Ceilidh framework. This course has now been re-installed at Nottingham and will soon be distributed to other pilot sites.

One key point to emerge from early piloting has been the importance of providing an open and extensible framework for Ceilidh. Not only should staff be able to build their own exercises, units and courses, but should also be able to bolt in their own marking modules for other programming languages or even for other kinds of assessment (we know of at least one other site where automated essay marking is being researched). In particular, there should be a separation between Ceilidh's general course administration framework and different marking techniques, with the latter ranging from completely automated marking though semi-automation to hand-marking. Although our current experience would broadly agree with Laurillard *et al* (1993) in that 'the Not Invented Here syndrome is not a major influence on academics considering the use of CBL', we would also stress that courseware should aim to provide as modular a framework as possible so as to allow teachers to mix and match resources, blending them into their own courses with individually tailored structures.

Multi-site use also requires support for the ability to transfer resources across networks for installation in local systems. In turn, this requires the development of adequate standards for course and resource structure (for example, is our current course-unit-exercise hierarchy general enough?). It also requires integration with more general information dissemination technologies including email, file transfer, directory services and network information retrieval systems (for example, Gopher, WAIS, Archie and World Wide Web).

We are hoping that our current pilot project will help develop the necessary standards and undertake the necessary integration to achieve this.

Future work

Future developments of Ceilidh fall into two categories. Those that are scheduled to take place during the TLTP pilot project, and more general developments.

During the TLTP pilot project, the following developments are scheduled:

(i) Porting Ceilidh to a networked PC environment.

(ii) Supporting additional programming languages. The working set should at least include C++, C, Pascal, Modula-2, ADA and ML.

(iii) Providing an extended statistics-gathering and progress-monitoring facility.

(iv) Integrating the current crude facility for reading course notes with common authoring and hypertext systems (key candidates include Authorware and Guide)

(v) Developing full X-Windows and Microsoft Windows interfaces.

(vi) Extending the current 'comment' facility towards a fully-fledged Help Desk with support for rostering queries between a team of advisers working across a computer network.

(vii) Extending facilities for exchanging course resources (e.g. notes and exercises) between sites, perhaps via integration with emerging network information retrieval systems.

A further goal for the near future is to conduct a more formal evaluation of the Ceilidh system. Some limited funding has recently been obtained towards this end, and we hope to compare student performance under Ceilidh with other indicators such as examination results and A-level entry grades. We also aim to carry out some marking experiments comparing automatic and human marking of the same scripts.

We plan a number of longer-term Ceilidh developments. Firstly, we will try to integrate the system with other local administration facilities, including Nottingham University's examinations and timetabling databases. We also aim to provide greater support for co-operative working. We plan experimental integration of conference facilities (both audio and text) to support discussion between students and staff, and also inclusion of bulletin board facilities. Our target here is a kind of networked programming laboratory distributed across several sites.

A final word: we would like to stress that Ceilidh is a fully working system and available from the authors.

References

Benford, S., Burke, E. and Foxley, E. (1993), 'Learning to construct quality software with the Ceilidh system', *The Software Quality Journal.*

Laurillard, D., Swift, B. and Darby, J. (1993), 'Academics' use of courseware materials: a survey', *Association for Learning Technology Journal*, **1**, 1.

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