E-ASSESSMENT OF COMPUTER PROGRAMMING

Rh. Gwynllyw[*,a] and J. Smith[b]

[a] Dept. of Engineering and Mathematics, University of the West of England, Bristol, UK.
[b] Dept. of Computer Science, University of the West of England, Bristol, UK.

*rhys.gwynllyw@uwe.ac.uk

Abstract

This paper demonstrates how we have used Dewis, an algorithmic open source e-assessment system, to automatically assess programming skills, in particular, in the C programming language. Teaching and assessing programming skills is challenging; prior to the implementation of this automatic assessment system, computing assessments were marked manually and this proved unpopular with students and academics due to the delay in marking and providing feedback. This new approach enables students to submit their computer code online through a link on their Virtual Learning Environment. From a student's perspective the marking process takes a matter of seconds before the student is provided with a mark and feedback. A number of pre-submission and post-submission checks are performed on the student's supplied code. These are essential to ensure that the student's code satisfies certain operational and security requirements before running on the system. Typically, the e-assessment system executes the student's supplied code a number of times with different input sets and the student's code is evaluated based on their code's resulting output. Prior to execution, the student's code may be augmented by Dewis-specific code to facilitate deeper analysis of the student's code. The analysis of the code's output enables the system to respond to the student's submission with 'intelligent feedback'. This feedback explains to the student, where appropriate, reasons for their submission not scoring full marks. There are also in-built detectors for 'common student errors' which, when triggered, further enhances the intelligent feedback. In their first few weeks at UWE Bristol, students are presented with a number of mini-tasks involving assessing C competencies with Dewis. These tasks are purely formative, support is given to students in computer lab sessions and students may submit as many times as they wish. Later on in their first year, students are given more significant programming projects (e.g. the n-Queens problem, path-finding problem), the assessing of which is summative. Results show that this innovative work is making a positive impact on students.

Keywords: *C programming, e-Assessment, intelligent feedback*

Introduction

The use of e-assessments in the Department of Computer Science at the University of the West of England (UWE) is well established both for formative assessments and for examinations. These assessments have been designed to primarily assess the students' knowledge of computer science principles as opposed to computing in practice. Further, with most of these assessments being of the form of multiple-choice questions, there is limited scope for 'intelligent feedback' in response to the students' input. We do make extensive use of Blackboard 'tests', which provide a range of different question styles, and considerable scope for bespoke feedback attached to different correct/incorrect responses. These have proven highly successful for summative and formative assessments, and provided useful cohort-level analysis of different questions. Some of this work was supported by external funding (e.g. UK Higher Education Authority grants) and these resources have been made available for sharing by the community. However, assessing more than simply 'recall' requires considerable expertise in question design to test higher level skills in problem analysis and reflection. The success of these e-assessments led to the consideration of whether e-assessments could be applied to test students' ability in the practice of computer programming. At the start of this project we undertook a review of available systems for providing automated teaching and assessment of coding. While there is a body of materials available for interpreted languages such as Python (such as the courses in Codeacademy.com) there was nothing available for languages which require code to be compiled and run (e.g. C, Java/C++) because of the security implications and difficulties in providing bespoke feedback related to specific learning activities.

In previous years, the summative assessment of computer programming in the module Artificial Intelligence on the Computer Science award at UWE, took the form of students submitting computer program solutions to set assessments via a VLE with the academic marking each submission by running the code and assessing the output. The main disadvantage of this process was, with a large number of student

submissions, the turnaround time it took for the academic to mark the submissions and provide feedback. To some extent this can be ameliorated by arranging for students to provide in-class 'walkthroughs' of their code, which provides great scope for immediate feedback and discussion, but this creates significant timetabling issues on large modules and it can be hard to automatically capture face-to-face feedback for students to reflect on later.

With UWE having their own algorithmic e-assessment system (Dewis) running on a Linux web-server, it was decided to develop that system to e-assess these computer code submissions.

Dewis is an e-assessment system created in 2006 at the UWE's Mathematics department (Gwynllyw and Henderson, 2009; Dewis Development Team, 2012). The primary purpose of the system was to provide algorithmic and intelligent e-assessments in numerate based subjects. In addition, the system was designed to be data-lossless so that all data for all assessment attempts is kept on the system's server. The question-editing mechanism on Dewis allows significant flexibility in the design of the question and includes coding the assessment on the server side (primarily using Perl) and on the client side (using html, css and javascript). The Dewis system is used extensively at UWE Bristol, and its satellite colleges, across several subject areas including Mathematics and Statistics, Engineering, Computing, Accounting, Science, Nursing and Business Studies. It is also used at other UK HEIs. The system is designed to support both formative and summative assessments and also has an 'examination setting' designed specifically for e-assessments in controlled conditions. The success of the system, its flexibility in the question authoring and its data-lossless storage made Dewis a natural consideration for hosting the e-assessment of computer programming at UWE. In addition Dewis had already been used to communicate with other programming environments. For example, Dewis has used system calls to Python/Sympy to implement a computer algebra system and it has also been used to interact with R to produce statistics e-assessments (Gwynllyw, Weir, and Henderson, 2015; Weir, Gwynllyw and Henderson, 2015).

The Dewis system was modified to allow for students to submit computer code, concentrating initially on submissions of the C-programming language. Two projects were started in 2015 which we will refer to as project-F (formative) and project-S (summative) for the purposes of this paper.

Project F (Formative e-Assessment)

Being able to program in C is not an entry requirement of the Computer Science award at UWE but it is highly desirable that students learn the basic C programming syntax at an early stage. Depending on the award taken, students at UWE may learn C (BSc Robotics), Java (BSc Computer Science) or C++ (BSc Games Technology), but in the early stages C is a good choice as it avoids the issues of object orientation and the syntax is common to these (and other) languages.

Retention on Computer Science and related awards is a national problem and, although UWE does well in this regard compared to other UK HEIs, it is still highly desirable to increase the retention rate. One contributory factor to low retention is that some students have particular problems with programming and it was recognised that such issues need to be identified and addressed as early as possible. Further it was recognised that a large number of computing students are activist learners and thus it is desirable to support their learning by supplying formative computing tasks to aid their learning. Providing such computing tasks with manual marking and fast feedback is infeasible due to the large number of students and the lack of staff time on the award. In a previous year, first year students were invited to email their tutor with their solutions to a small programming task. Take-up on this invitation was high which lead to excessive delays in feedback and a negative student experience.

Following a 'works in principle' period with the e-assessment of C computer code, it was recognised that automated assessment could support a 3-week introductory course in programming starting in induction week. This course was aimed to remove some of students' fears about programming by enabling a sense of recognised achievement. It was also aimed at facilitating a rapid understanding of basic C programming syntax, especially in the use of conditional statements and loops. The course provided computer laboratory sessions but students were told they also had to commit a significant amount of self-study hours (the pace of delivery was aimed to require approximately 4 hours per week) and to become used to the habit of using other resources such as the Faculty's drop-in '*espressoProgramming*' sessions. The use of e-assessments accommodated for the different work patterns that students employed and provided instant feedback on the students' submitted work, thus enhancing the student's experience of the course.

An important part of this project was the monitoring of student engagement. Students' engagement with these e-assessments was monitored through the e-assessment system's performance reporter; this monitoring was in addition to the traditional engagement measure of attendance recording. Non- or low-engagement students were emailed directly by the module team in addition to passing engagement information on to a dedicated retention monitoring team.

With regards the e-assessment's specific contribution to this short course, eleven formative tasks were identified by academics to teach basic computer programming constructs. These C-programming tasks were as follows:

Task 1: Print out a specified string and then wait between 5 and 10 seconds, and then terminate. This assesses the ability to combine and modify simple code snippets.

Task 2: Print out a specified string and then wait for an input character from the user. Echo that character back

and terminate. This assesses simple i/o and the use of 'print and pause' constructs useful in debugging.

Task 3: Perform the addition of two floating point numbers that are read interactively during run time from the keyboard (via scanf). Output the resulting addition if both the two inputs are numbers, otherwise output 'invalid input'. This assesses simple interactive conditional flow.

Task 4: Same as Task 3 except that the inputs for the program are taken at the start of the run process - provided via the command line (via arg).

Task 5: Similar to Task 4 except the requirement is for the calculation of the quotient of two floating point numbers. If the second number is a zero then output 'invalid input – division by zero not allowed'. This assesses simple conditional flow.

Task 6: Read an operation from the command line of the form $x \circ y$ where x and y are expected to be numbers and $\circ$ is expected to be one of '+', '−','×' or '÷'. If the input is not as expected, then output 'invalid input'. If the calculated output is not a number then output 'invalid output'. Otherwise, output the numerical value. This assesses more complex conditional flow (e.g. embedded *if* or *switch* constructs).

Task 7: Modify Task 6 to include the option of a 'running total' whereby the code accepts, as input, a file containing a sequence of operations of the form $x \circ y$. In addition, if the character p appears in the place of an expected number, then the p assumes the number in the immediately preceding calculation. This assesses the use of mechanisms for storing state/history.

Task 8: Similar to Task 7 but the reading of the input and its validity testing is done using a call to a function called 'read_and_validate_input'. Further, the implementation of the numerical operations should be done using calls to functions 'addition()', 'subtraction()', etc., implemented using signatures supplied to the student. This assesses the use of modularisation and code re-use.

Task 9&10: Print out an 8×8 checkerboard containing o's and x's generated using two nested 'for' (**Task 9)** or 'while (**Task 10**) loops with each execution of the inner loop producing exactly one character. These two tasks assess the use of iteration.

Task 11: Read a string via the command line and output eight strings on different lines, with each output containing the input string but with the final character replaced by the loop count (1..8). This assesses the use of more complex data types such as arrays.

It was an essential requirement of the construction of these 11 e-assessments that the student experience was a positive one. This included consideration to the ease of the student submitting their solutions and the efficacy of the feedback in the case of a student's submission being rejected (e.g. their code containing illegal content or not compiling) or their submission being incorrect in that their code does not satisfy the assessment criteria.

The process of submission of C-code involves the student accessing the relevant module's web page on the university's VLE and, from there, accessing the assessment task's specific Dewis page via an LTI link (such a link allows for Dewis to pass back the student's attainment mark back to the VLE's Grade Centre). The Dewis system then prompts the student to submit their C-code. Such a prompt, for Task 1, is shown in Figure 1.

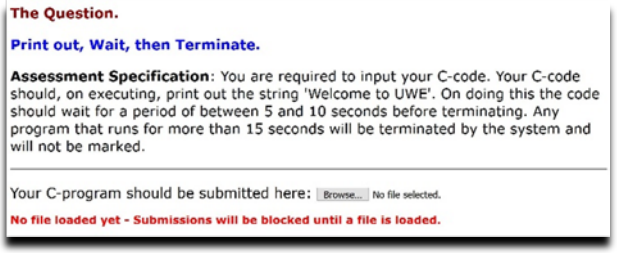

Figure 1: Dewis prompt for the student to submit their code for Task 1.

The submission process involves Dewis eventually running the student's C-code on the Dewis server. This action has significant security implications and hence there are a number of checks the Dewis system makes to ensure that the student's code cannot be malicious. A result of this is that there are a number of commands that Dewis prohibits from being submitted in the student code. These commands include, for example, potentially malicious system calls and the inclusion of undesirable header files. As part of ensuring a positive student experience of the process, these security checks are made on the student code when they link their C-code to their web browser. That is, these checks are made prior to the student submitting their code to the Dewis system. On detecting prohibited content, the student is informed immediately that their submission cannot proceed together with the reason for the barrier. Text on the VLE page for each task provides more help in these cases.

Having submitted the code, the next stage is for Dewis to compile the code. If the student's code fails to compile, the system will echo back to the student the compilation error. In the case of a common compilation error being triggered, the system will aim to supply suggestions to the students to address the error. A compilation error can be construed as detrimental to the student experience and, in the case of a student preparing their code on a different operating system to that of the Dewis system (Linux), a compilation error may not have been anticipatable by the novice programmer. Because of this, most Dewis assessments of this form have an associated 'compiler check' assessment whereby students can check their programs for compilation on the Dewis server without foregoing an assessment attempt.

For some of the assessment tasks, prior to executing the student's submitted code, the Dewis system will augment this code with additional code to facilitate the marking process. For example, for Task 9, augmented code is used in order to ensure that the correct number of nested for-loops are used in the construction of the required output.

An example of the Dewis system detecting a code submission on this assessment task not satisfying the criteria is given in Figure 2. The feedback informs the student as to the reason why their submission was deemed incorrect. In this example, the student used only one for-loop (1..8) with each execution producing complete lines in the grid ('oxoxoxox' or 'xoxoxoxo').

Following the feedback in Figure 2, this student subsequently submitted an entry which was unaltered except that it contained a 'null' inner for-loop (1..8) which produced no output. Again, the Dewis system informed the student that their submission did not satisfy the criteria. This later student submission was presumably an attempt to trick the Dewis system. It was encouraging to note that the student did eventually submit code that satisfied the task's requirement.

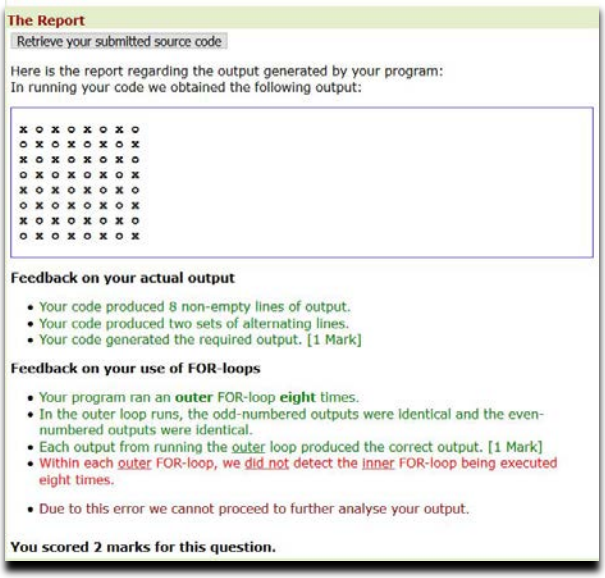

Figure 2: Example feedback for Task 9 where the student's submission did not use nested for-loops to generate the required 8×8 grid.

In Figure 2, we also show that, in supplying the student feedback, the system also provides a link to the student's original submitted code. This feedback is available to students at any future time via the VLE/Dewis link to the assessment so that students may revisit their past submissions.

On executing the student's code, certain system limiters are applied to ensure that the student's code does not consume an excessive amount of the system's resources. For example, the student's code is terminated if it consumes excessive CPU or memory, or simply does not terminate within a reasonable period of time. In such cases the student is informed of the reason for their submission not being suitable for marking together, where possible, with suggestions as to how to address this issue.

Impact

Student uptake on these tasks has been encouraging. For example, Table 1 shows the uptake results for the 2017/2018 academic year.

Task #	# students	# attempts	# correct
1	189	535	171
2	187	613	163
3	168	836	152
4	151	796	127
5	140	577	111
6	122	783	84
7	64	290	37
8	35	148	17
9	71	180	59
10	88	454	70
11	32	135	17

Table 1: Engagement statistics for the eleven summative tasks. For each task, the 2nd column lists the number of students that submitted their C code, the 3rd column lists the total number of submitted attempts, the 4th column lists the number of students who fully achieved the tasks' criteria.

Since these tasks were formative, there was no limit as to the number of attempts for a student to attempt these tasks. In most cases the students had several attempts at each task, seeking targeted help in response to the system's feedback and worked through their code to eventually produce code that met the assessments' criteria. For example, from Table 1, we see that 171 out of the 189 students that attempted Task 1, succeeded in submitting C-code that satisfied the task's requirement.

This level of engagement is mirrored by the volume of emails, and hence staff-student interactions generated. Notably, as we have refined the system over 2-3 years, the volume of emails, particularly regarding later tasks, has been reduced without the patterns of engagement changing. The process of identifying recurring issues in emails and error logs, and then amending the system to recognise and respond to those cases, has been successful in moving from manual to automated feedback. The system is transparent enough for students to 'learn how to learn' – taking more time to use the automated feedback.

The drop-off in the number of submissions reflects to some extent the different rates of progress that students were making on their 'standard' programming modules. Anecdotal evidence via students emails and anonymous end-of-module feedback also suggests that some students engaged less if they felt that the system was 'overly restrictive or strict' in terms of the constructs allowed and the way that exact output formats are required (e.g. use of capitalisation, spaces etc.). Capturing this feedback has been invaluable for staff, and we now pay considerably more attention in-

class to pointing out that the days of the '*lone developer*' are largely over, and that all code should be designed for a specific purpose and tested to meet specific requirements and interfaces. Being able to pull specific (anonymised) examples of code that '*looks right but does not meet the specifications'* is invaluable in providing concrete examples of more abstract ideas.

These formative tasks are now being used on several different modules across various awards and being used in level two to help students refresh their C-skills as a form of pre-requisite test.

Anecdotal evidence of the success of this was an improved performance in the summative assessments discussed in the next section. Since the introduction of this system, tutors in-class have noted a significant increase in the proportion of students discussing algorithmic issues in more advanced problems as opposed to programming syntax issues. E.g. students were more comfortable with writing out algorithms in pseudo-code to discuss functionality as opposed to syntax.

A highly positive outcome has been that by the fourth week of term we are able to get students collaborating in-class to work on pseudocode designs and mechanisms for simple algorithms – valuable learning activities that had been not previously been possible. Familiarity with basic programming constructs means that, from far earlier in the module run, lectures can include pseudo code and code snippets to demonstrate search and machine learning algorithms. For many of our students this helps demystify the abstract definitions.

Other staff who used some of the first activities during induction week have also commented that these provided a useful 'icebreaker' mechanism to get students collaborating and problem-solving.

Project S (Summative e-Assessment)

Dewis was also used for the e-assessment of C-programs in a summative environment in the module 'Introduction to Artificial Intelligence', part of the Computer Science award. There are two such assessments on this module and initially they were quite low stakes (each accounting for 12.5% of the total marks for the module). Although the two assessments seem quite different, they are closely related in that they both require the student to produce C-code that solve problems involving search algorithms. The two problems are:

- the 8-Queens problem, solved using Depth-First Search;
- the shortest path problem (SPP) using Dijkstra's algorithm on a map modelled by a square grid.

These two problems are both 'search' problems, and use a common code framework (written by the academic) provided to the students and used for earlier tutorial work. One of the intended learning outcomes is to see how a framework can be used to (i) implement different algorithms and (ii) tackle very different problems, with only very minor changes (typically a few lines).

In previous years, students were given a tool written in Netlogo that involved PacMan searching a maze (Smith, 2009). Switches allowed students to choose policies to apply whenever a junction was made, and these effectively implemented different algorithms. The students were then asked to write down their results and to submit their solutions via the VLE for manual marking. However, while providing a nice visual interpretation of the effects on a toy problem, the module leader wanted the students to implement the algorithms in actual code with the aim of ensuring a better understanding of the functionality of the algorithm.

For the assessment of these search algorithms, the Dewis marking process will be two-fold, namely:

- the solution is checked for correctness;
- the number of candidate solutions considered is consistent with the method of search.

The latter criterion was included to ensure that the student's code implemented the correct method of search. Full marks are awarded to the student's submission if, and only if, both these criteria are met.

For both types of problems, the student's code was executed a number of times for different inputs. The inputs for the two problems are as follows:

- 8-Queens: the position of the Queen in the first row of the chessboard;
- SPP: the start and end point of the path on a square grid, together with the location of the obstacles on the grid.

In the case of the 'number of candidate solutions' considered being incorrect, the marking algorithm would attempt to identify a pattern in the number reported by the student's code. For example, some students' code consistently produced a numerical value one higher than the correct solution. In such a case, the student would be informed of this, together with stating common reasons that code consider one more candidate solution than is required.

The two current metrics effectively perform black box testing of the supplied code, therefore test problems have been designed to ensure that different algorithms give different results. For the next academic year, the code has been further refined to perform 'white-box testing', via checking of values that student's code passes as parameters to supplied functions.

For these summative e-assessments, the student was only required to provide the 'main' function call for solving the problem. As part of the student's development of their code, their code would be built with pre-supplied code (written by the academic), and they are told what a subset of the intended results should be (i.e. for one starting queen position or map).

The student would submit, to the Dewis system, only the 'main' part of the code. That is, the pre-supplied code that is part of the build, already resides on the Dewis system. The version that resides on the Dewis system keeps track of the number of candidate solutions considered in the search process as well as the final solution obtained in the search process. As such, the Dewis system does not depend on the student's code telling it the solution nor the number of candidate solutions. This ensures that the correct solutions are not obtained artificially (e.g. hard coded in the student code).

Results

Quantitatively, any difference in the coursework pass rates is smaller than the annual fluctuations seen on any course. Qualitatively, feedback from students has been that they appreciated the opportunity to submit in their own time – giving them the chance to manage different demands on their time safe in the knowledge of the marks they would get.

Following the success of the phase one project, the assessment regime of the module has been changed to incorporate a further two exercises (again each worth 12.5%). In the third task, students submit a text file containing the knowledge base for a chatbot in AIML. The Dewis system marks and provides feedback by running a java programme that exploits the file handling and output-interpreting mechanism developed. Students are told the 'questions' in advance, and marks are awarded according to how well their knowledge base exploits different language features.

The final task requires students to submit a C code implementation of the machine learning algorithm of their choice, which is assessed via its predictive accuracy on a number of datasets designed to test aspects such as handling duplicates, class imbalance etc.

The impact of these two more 'open-ended' pieces of coursework, where competition has been encouraged, has been incredibly positive. In both the last two years an 'arms-race' has developed with students contributing specific ideas to discussions of how the tests could be made harder/ more discriminating – via different chatbot questions, or datasets with different characteristics. In the machine learning task some students implement simple algorithms such as K-nearest neighbours, but we have seen example of Bayesian networks, Rule Induction algorithms, and Multi-layer Perceptrons being submitted.

As we have said previously, many of our students self-identify as being predominantly activist or pragmatist learners and many are more likely to submit credit-bearing work. Dewis' '*instant marking and feedback*' means that the depth of insights displayed during in-class discussions about the merits of different approaches has been raised to new levels by the provision of learning activities more suited to our students' styles of learning.

Discussion

The success of the approach described in this paper, which allow students to "self-learn" programming skills, has led us to develop the system further. Competence in programming is desirable in many academic disciplines, not just for Computing students. Indeed Bond (2018) recommends that computer programming becomes a core part of mathematics degrees. In the forthcoming 2018/19 academic year, the e-assessment of computer programming will be extended to Level 3 Mathematics students using Python on the Numerical Analysis module at UWE Bristol. Students will be required to write numerical methods in the Python programming language and this will be assessed automatically using Dewis.

Previously, a manual marking process was employed for the Numerical Analysis module but the workload involved in processing these student submissions resulted in difficulties in producing appropriate and timely feedback. The cases whereby the feedback was delayed resulted in negative student feedback about the process. The previous deployment of Dewis to e-assess computer code in C means that the development time required for Dewis to e-assess Python was significantly reduced.

The success of this extension to the project will be evaluated using student feedback via the module evaluation process and a comparison of student performance in their programming competencies.

References

Bond, P. (2018). *The Era of Mathematics*. Retrieved from https://epsrc.ukri.org/newsevents/pubs/era-of-maths/

Dewis Development Team (2012) Dewis welcome page. Retrieved from http://dewis.uwe.ac.uk

Gwynllyw, R. and Henderson, K. (2009). DEWIS: a computer aided assessment system for mathematics and statistics. *CETL-MSOR 2008 Conference Proceedings*, pp. 38-44.

Gwynllyw, R., Weir, I. and Henderson, K. (2015). Using DEWIS and R for multi-staged statistics e-assessments. *Teaching Mathematics and its Applications*, 35(1), pp. 14-26.

Smith, J.E. (2009). Learning Through Programming Games: Teaching AI with Pacman and Netlogo. In *Proc 5th UK Conference on AI in Education, Higher Education Academy Information and Computer Sciences Subject Centre (HEA-ICS), 2009*.

Weir, I., Gwynllyw, R. and Henderson, K. (2015). Using technology to inspire and enhance the learning of statistics in a large cohort of diverse ability. In*: IATED, ed. (2015) Edulearn15 Proceedings.*