Computing with Codio at Coventry University

Online virtual Linux boxes and automated formative feedback

David Croft
Coventry University
Coventry, U.K.
David.Croft@coventry.ac.uk

Matthew England
Coventry University
Coventry, U.K.
Matthew.England@coventry.ac.uk

ABSTRACT

We describe our experience using Codio at Coventry University in our undergraduate programming curriculum. Codio provides students with online virtual Linux boxes, and allows staff to equip these with guides written in markdown and supplemental tasks that provide automated feedback. The use of Codio has coincided with a steady increase in student performance and satisfaction as well as far greater data on student engagement and performance.

CCS CONCEPTS

• **Social and professional topics → Computing education**; • **Applied computing → Interactive learning environments**; **Computer-assisted instruction**; **Computer-managed instruction**; **Learning management systems**;

KEYWORDS

Programming Education; Virtual Machines; Automated Feedback

ACM Reference Format:
David Croft and Matthew England. 2019. Computing with Codio at Coventry University: Online virtual Linux boxes and automated formative feedback. In *Computing Education Practice (CEP '19), January 9, 2019, Durham, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3294016.3294018

1 INTRODUCTION

The School of Computing, Electronics & Mathematics (CEM) within the Faculty of Engineering, Environment & Computing (EEC) at Coventry University runs 7 BSc degrees which share initial modules on programming led by the authors. This includes BSc Computer Science, but also degrees split with other fields such as BSc Information Technology for Business, and BSc Mathematics & Data Science. We describe recent innovations in these programming classes.

1.1 Situation Prior to Codio

Prior to the initiatives we discuss in this paper, the content in these programming classes was fairly traditional[1]. There were a mix of lectures based around slides and labs based around a print out (or more recently a pdf on screen) with a series of problems to tackle. Students attempt these, getting help from staff when needed, with solutions released at some later date. To gain formative feedback students had to do one of the following:

- Wait until the nominated release date of solutions and check their answers. This is usually a week later by which time the student may have forgotten the details of the exercise and their solution, dampening the learning that can take place.
- Request feedback from staff in lab sessions. Many students seem to find this difficult, particularly in Year 1: perhaps over fear/embarrassment of asking for help, or simply not reaching a stage where feedback can be useful during labs.

Finding a method to more effectively deliver meaningful feedback to student was a key goal for the authors. This motivation was compounded by a rapid increase in student numbers in these classes. These have doubled over just a few years to reach 480 Stage 1 students taking programming in the 2017/18 academic year. The corresponding increase in staff is not of the same scale, putting a greater strain on the delivery of formative feedback.

Even where staff resources do rise we have a related problem of consistency of feedback. For example, in Semester 1 2017/18 the first year programming module took place over 40 hours in 20 different lab session taught by a combination of 12 different staff members. Even with the best of intentions, consistency of quantity, quality and content of formative feedback was not possible. We decided to use automated testing to improve the situation. Our aims were:

(1) more interactive lab material with a degree of gamification;
(2) instant formative feedback for students; and
(3) better monitoring of student progress on modules.

1.2 Codio

The decision was made to implement our solutions via Codio[2], a commercial cloud based infrastructure focused on STEM, and in particular Computer Science, education.

Teachers prepare Codio Units for students. Each unit is built around a Linux based Virtual Machine (VM) environment[3], augmented with a guide (written by the tutors) which can contain a variety of tasks to provide instant formative feedback.

[1] We note the presence of innovative practice elsewhere in the computing curriculum however, in particular our Activity Led Learning (ALL) projects [1]. Briefly: all students start their degrees with an integrative project that requires ideas from all the modules they are studying but is led by the problem statement and can be developed differently depending on the students interests. The ALL initiative began prior to the ones we focus on here and continues to this date. The ALL classes offer students an opportunity to put their modules into context, and practice their programming skills as part of a larger project. This paper focuses on how we have improved the actual programming classes which take place alongside ALL and provide those skills.

[2] https://codio.co.uk/

[3] Each student has their own VM - the actions of one student do not affect the others.

CEP '19, January 9, 2019, Durham, United Kingdom
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6631-1/19/01.
https://doi.org/10.1145/3294016.3294018

In Section 2.1 we describe the benefits we found from using Codio as a development environment and in Section 2.2 we focus on the most important one: the automated programming feedback now provided at Coventry. In Section 3 we describe out initial findings on the value this had added to our students education, in Section 4 we summarise some challenges we encountered, and we finish in Section 5 with some plans for the future.

2 OUR EXPERIENCES WITH CODIO

2.1 Development environment

One of the more irritating factors in teaching programming are the limitations imposed by developing on a centrally managed IT system. Programming classes at Coventry are taught in PC labs, and whilst some students may bring personal devices, most use those machines in class. These machines are restricted in a number of ways for obvious security and management issues: for example, students are not able to install additional software on them. This extends to code libraries and Integrated Development Environments (IDEs). Tutors must decide over the summer what software is required for all classes in the year ahead. Even when this level of advance planning is achieved it shuts down the ability for students to research and choose their own libraries for projects and extra curricular work.

At Coventry the initial approach to resolving this issue was the use of cloud VM providers on an ad-hoc basis in individual teaching modules. The passing of the General Data Protection Regulation (GDPR) in 2016 put an end to this practice as it left the university open to significant fines[4]. The decision was made to use a single GDPR compliant cloud solution. Although Codio does not have as many development features as other providers, its focus on education and testing functionality as detailed below, made it the best choice for our teaching purposes.

Figure 1 shows a screenshot from a typical Codio session: on the left we have the file tree of the main workspace; middle bottom a text editor[5]; middle top a Linux bash terminal where students run and test their code themselves; and on the right the guide. The guide is written by the teacher in `Markdown` with simple environments for text, mathematics (using LaTeX) and code (with syntax highlighting for common languages). Most importantly, the guides contain embedded assessments. The assessments can be multiple choice, fill in the blanks etc. but of most interest are those that run a test script prepared by the teachers and hidden to the students, as discussed in Section 2.2.

As an environment Codio offers two further notable advantages. First, it essentially doubles as a cloud storage system for a student's programming work: they can access Codio from anywhere they have internet and pickup their work with the code and files where they left of, removing the need to backup onto USB drives or email files at the end of each class[6]. Secondly, if students are having issues then staff are able to remotely open their projects and look into the issue directly. This saves considerable time compared to lengthy email exchanges that contain code snippets without syntax highlighting (and often with email software removing indentation), which may not actually even contain the bug.

2.2 Automated feedback on code

At Coventry we still have a significant proportion of students who have not programmed much before university[7] and so there is a need for regular and detailed support on often fairly basic points. Even amongst those who have programmed before, effective software testing is a separate skill that few students enter with.

We thus implemented our own tests that would judge a student's code. This offers a number of advantages:

- Our tests are usually more thorough than those a student would implement.
- A student gets validation that their code works, even if it differs from the *model solution* provided by the teacher.
- This feedback is available at any time to suit the students, in or outside class. They are able to enter a cycle of development and testing without waiting to see the teacher.
- While not contributing to their grade, some student derive satisfaction from a *high score* on a unit (gamification).

The tests and feedback can take any form (that may be delivered by terminal) as they are coded by the instructor. For us, they are usually a detailed set of unit and/or system tests, although in some cases we can pose further restrictions on students if appropriate for the topic of study (e.g. no use of loops on the tasks for studying recursion). We usually let students see which test cases fail, and sometimes augment this with additional text feedback that expands on the error message from the language or points out common mistakes (e.g. if their code produces a Python `IndexError` we might remind them that Python lists index from 0).

Example. An early lab task has students writing a function to identify if a given positive integer is a leap year or not[8]. The complete leap year logic is shown in Python in Figure 2. The task is partially intended to demonstrate a student's ability to correctly declare and call a function. The main aim is to provoke a discussion regarding the importance of correct requirements gathering and to demonstrate the dangers of making unsupported assumptions when writing their code.

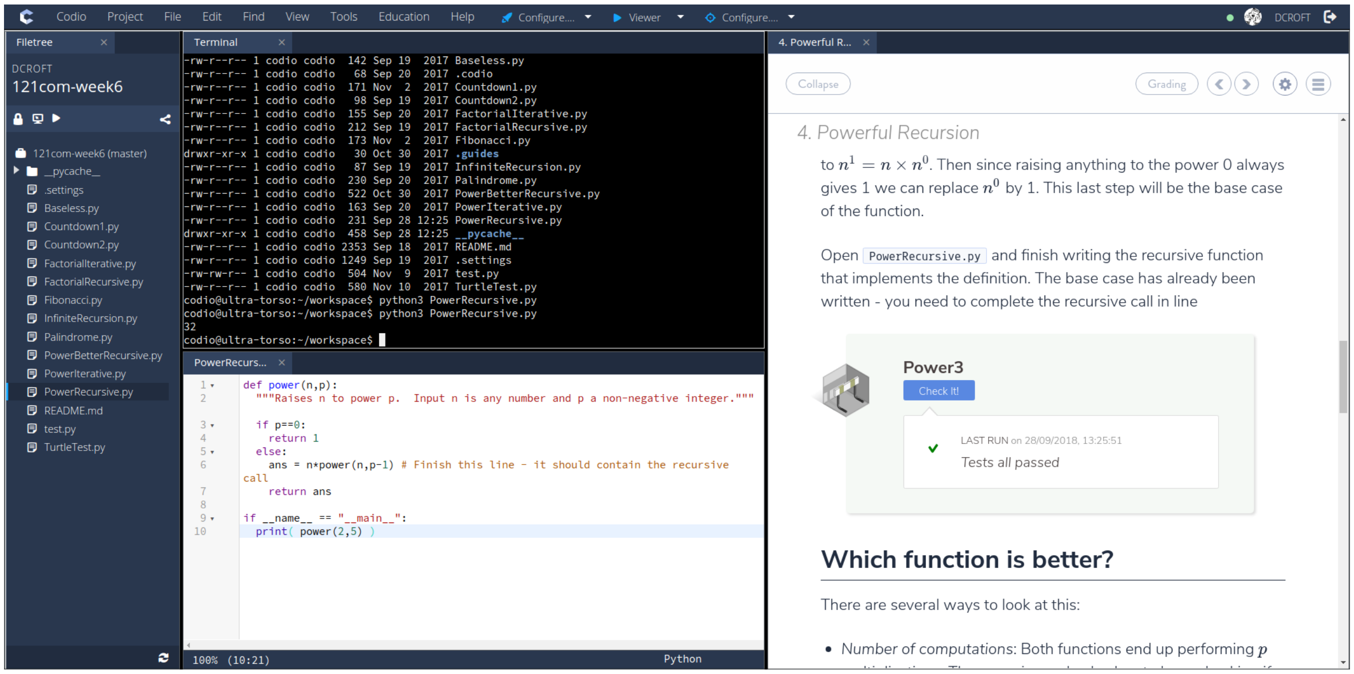
The majority of students will initially write functions that simply test if a year is divisible by 4 and will test only a handful of values based on that condition (usually the current and adjacent years) while our automated tests would examine all 4 branches of the logic. The end result is a demonstration of good development practice as well as ensuring that all students have completed a task that tests their ability to write functions that test multiple logical conditions.

3 VALUE ADDED

Codio was initially trialled on a single module basis in 2016/17 with a roll out to all first year programming in 2017/18. Going into the 2018/19 academic year Codio use has expanded into further modules and courses.

[4]For details on the GDPR and practical examples see for example [6].
[5]The editor has basic IDE tools such as syntax highlighting and auto-complete of existing function and variable names.
[6]It also removes one of the key excuses for not doing homework!

[7]While the UK government may now mandate programming education in schools this has not fully filtered through to UG entrance yet, and even then does not take into account our substantial international student base.
[8]Without using the language's `datetime` libraries - something tests can easily detect.



```
-rw-r--r-- 1 codio codio  142 Sep 19  2017 Baseless.py
-rw-r--r-- 1 codio codio   68 Sep 20  2017 .codio
-rw-r--r-- 1 codio codio  171 Nov  2  2017 Countdown1.py
-rw-r--r-- 1 codio codio   98 Sep 19  2017 Countdown2.py
-rw-r--r-- 1 codio codio  155 Sep 20  2017 FactorialIterative.py
-rw-r--r-- 1 codio codio  212 Sep 19  2017 FactorialRecursive.py
-rw-r--r-- 1 codio codio  173 Nov  2  2017 Fibonacci.py
drwxr-xr-x 1 codio codio   30 Oct 30  2017 .guides
-rw-r--r-- 1 codio codio   87 Sep 19  2017 InfiniteRecursion.py
-rw-r--r-- 1 codio codio  230 Sep 20  2017 Palindrome.py
-rw-r--r-- 1 codio codio  522 Oct 30  2017 PowerBetterRecursive.py
-rw-r--r-- 1 codio codio  163 Sep 20  2017 PowerIterative.py
-rw-r--r-- 1 codio codio  231 Sep 28 12:25 PowerRecursive.py
drwxr-xr-x 1 codio codio  458 Sep 28 12:25 __pycache__
-rw-r--r-- 1 codio codio 2353 Sep 18  2017 README.md
-rw-r--r-- 1 codio codio 1249 Sep 19  2017 .settings
-rw-rw-r-- 1 codio codio  504 Nov  9  2017 test.py
-rw-r--r-- 1 codio codio  580 Nov 10  2017 TurtleTest.py
codio@ultra-torso:~/workspace$ python3 PowerRecursive.py
codio@ultra-torso:~/workspace$ python3 PowerRecursive.py
32
codio@ultra-torso:~/workspace$
```

```
def power(n,p):
  """Raises n to power p.  Input n is any number and p a non-negative integer."""

  if p==0:
    return 1
  else:
    ans = n*power(n,p-1) # Finish this line - it should contain the recursive call
    return ans

if __name__ == "__main__":
  print( power(2,5) )
```

100% (10:21) Python

4. Powerful Recursion

to $n^1 = n \times n^0$. Then since raising anything to the power 0 always gives 1 we can replace n^0 by 1. This last step will be the base case of the function.

Open `PowerRecursive.py` and finish writing the recursive function that implements the definition. The base case has already been written - you need to complete the recursive call in line

Power3

LAST RUN on 28/09/2018, 13:25:51

Tests all passed

Which function is better?

There are several ways to look at this:

- Number of computations: Both functions end up performing p

Figure 1: The Codio environment

```
def is_leap( year ):
    if year % 400 == 0: return True
    elif year % 100 == 0: return False
    return year % 4 == 0
```

Figure 2: Example of solution expected from students.

3.1 Monitoring and Engagement

Codio provides monitoring features at both class and individual level. We have developed additional functionality to record historical engagement and compare with performance on summative assessment (which takes place outside of Codio). We aim to use this data to improve engagement, and hence performance and retention.

Our main goal is for students to gain a functional level of programming competence, and a pre-requisite for this is simply putting in the time to practice. The often quoted figures are "*10 years or 10,000 hours to become an expert*" [2, 3]. Of course, for that time to be effective students should be challenged to explore more complex programs rather than repeating the same tasks.

This link from engagement to achievement is well established [5] but not always obvious to students. Now when we make the claim we can back it up with aggregated evidence produced by our own students. For example, Figure 3 shows the relationship between minutes in Codio and marks on relevant assessment for one cohort – the linear relationship between the two apparent. Further, we can show students their own data and a predicted final mark based on current engagement. Figure 4 shows an example of the type of data that could be extracted for an individual student. Figures 3 and 4 make clear the strong correlation between Codio engagement and

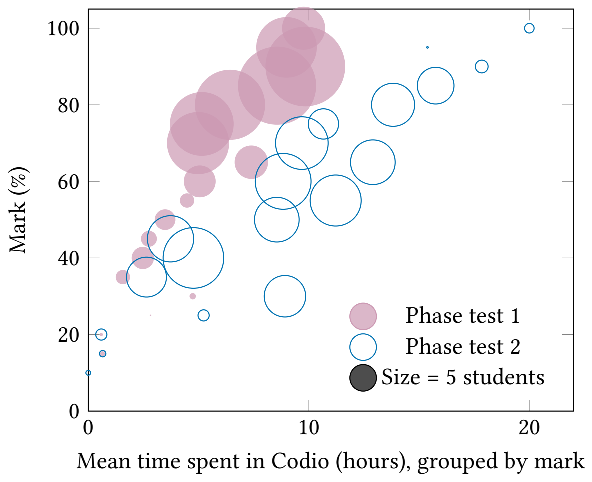


Figure 3: Relationship of marks to time engaged in Codio.

grades. To be clear, the inclusion of these figures is not to make a claim of causation derived from Codio use, but to demonstrate the type of data that can be extracted for the purposes of convincing students of the importance of engagement!

During the 2017/18 year the Codio data was combined with information from the university's register system and used to send emails to all students. These emails were automated, but also personalised (templates with blanks for personal data). They gave specific actionable advice based on individual situation, e.g. if doing well told to keep up the good work; or if evidence of missing knowledge told how to catch up. The emails made mention of the specific topics of study: possible via the Codio tracking.

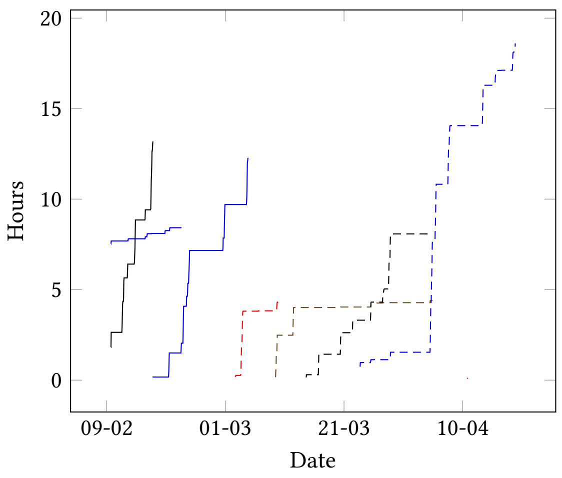


Figure 4: Example of engagement tracking for an individual student. Each line is time spent on a week's Codio activities.

3.2 Performance and Satisfaction

The second author ran the same module in 2016/17 without Codio and then in 2017/18 with Codio. The other changes made between these two years were minimal. The material that formed the Codio guides was taken from existing pdfs and the tasks from existing lab questions. The entry criteria onto the degrees did not change significantly and the teaching style was otherwise unchanged.

The summative assessment points (two multiple choice quizzes and a written exam) were similar in style, structure, and difficulty between the two years[9]. Overall pass rates remained broadly the same. However, average marks increased significantly. The average grade on the first quiz increased from 57% to 63%, while the grade on the written exam from 51% from 59%. It seems Codio had improved performance for most, but not the very weakest students.

Module satisfaction is measured using an online questionnaire that mimics the UK National Student Survey. For this module overall satisfaction increased significantly from 82% to 89%. 154 students left free text comments that mentioned Codio, with 84% of those positive. Student feedback is almost unanimously in favour of automated feedback, however, views on Codio itself are more diverse. While a large majority like the system, some are vocally against.

Those against are often the students with the most prior programming experience who resent being denied the use of their IDE of choice[10]. We note that in addition to the tasks set in Codio we also set students larger extended projects and problems which can be tackled on any platform the student chooses (such as the ALL projects mentioned in footnote 1). We tend to use Codio for the basic (and thus most important) tasks that introduce the material.

[9]While of course having different questions to maintain integrity.

[10]It is certainly true that environments such as CodeBlocks and Visual Studio have more features (and a wealth of online tutorial resources) but students who use these systems can lack an understanding of what tasks are being performed for them. For example, we force students in Codio to manually compile their C++ programs at the start deliberately, to ensure they have an understanding of the process before moving on to using automated build tools.

4 CHALLENGES

Some challenges the authors encountered include the following.

- The initial creation of the automated tests requires a large up front investment of time.
- Dependency on an online system means that in the unlikely event of a campus internet failure classes come to a halt.
- Codio itself has been found to contain occasional bugs. The Codio support team acted rapidly to fix these.

The greatest challenges though, have been in the legal and administrative side of things. GDPR requirements were an original motivation for using Codio, but the university's changing interpretation of them has created further hurdles, e.g. on the location of hosted data and how student consent is taken. Although Codio does have built-in integration for popular Virtual Learning Environments (VLEs) such as Moodle, Coventry regulations have blocked our use of these, leading to a painful manual signup procedure.

5 FUTURE WORK

We see a number of future developments in this initiative, especially in further use and analysis of the data Codio produces. Given the success in formative assessment an obvious next step is the automated assessment of summative work. Whilst this does pose some challenge in assessment design, again, the main barriers to date are university regulations: as a 3rd party platform, Coventry currently does not allow the use of Codio for summative assessment.

We are now extending our existing automated summative assessment (multiple choice quizzes in the Moodle VLE) to include question types that ask students to write code which is evaluated against unit tests. For this we are using `Coderunner` [4], a free open source Moodle plugin that can run entirely on university hardware.

6 SUMMARY

We now extensively use automated feedback and cloud VMs in the programming curriculum at Coventry University, with positive results and further developments underway. A message for colleagues with similar plans is that difficulties may include not only upfront development costs but also legal and institutional regulations.

ACKNOWLEDGEMENTS

The authors thank their colleagues who teach in these classes, and acknowledge the support of The Institute of Coding.

REFERENCES

[1] E. F. Anderson, C. E. Peters, J. Halloran, P. Every, J. Shuttleworth, F. Liarokapis, R. Lane, and M. Richards. 2012. In at the Deep End: An Activity-Led Introduction to First Year Creative Computing. *Computer Graphics Forum* 31, 6 (2012), 1852–1866. https://doi.org/10.1111/j.1467-8659.2012.03066.x

[2] B. Bloom. 1985. *Developing Talent in Young People.* Ballantine Books. https://www.xarg.org/ref/a/034531509X/

[3] J.R. Hayes. 1989. *The Complete Problem Solver, Second Edition.* Lawrence Erlbaum Associates, Publishers. https://www.xarg.org/ref/a/0805803092/

[4] R. Lobb and J. Harlow. 2016. Coderunner. *ACM Inroads* 7, 1 (Feb 2016), 47–51. https://doi.org/10.1145/2810041

[5] V. Trowler. 2010. Student engagement literature review. *The Higher Education Academy* 11, 1 (2010), 1–15. https://www.heacademy.ac.uk/sites/default/files/StudentEngagementLiteratureReview_1.pdf

[6] P. Voigt and A. von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR).* Springer International Publishing. https://doi.org/10.1007/978-3-319-57959-7