

Student Transitions

Learning and Teaching Enhancement Project Report, May 2016

Project Title: Transition of automated feedback and marking into student's learning environment

School & department: Computer Science, School of Mathematical and Computer Sciences

Please be aware that this report will be publicly available on the University's webpages

Please complete in **Word** and return to Mirren.McLeod@hw.ac.uk by 30.5.16 (reports not completed in Word will be returned)

1	<p><i>Names and Heriot-Watt University contact details of project team (please identify the project lead/ report author):</i></p> <p>Project lead and main report author: Gudmund Grov (G.Grov@hw.ac.uk). Co-authors: Verena Rieser, Michael Lones and Greame Reid</p>
2	<p><i>Key words:</i></p> <p>Technology Enhanced Learning and Teaching, Student Transitions, 2st year students, self-directed learning, Automation, formative feedback, marking, computer programming</p>
3	<p><i>The problem being addressed, with background and context:</i></p> <p>This is the second phase¹ of a project concerned with the problem of how to provide timely and constructive feedback to first year Computer Science (CS) students by developing an automated feedback framework.</p> <p>Learning how to program computers is one of the major hurdles for our first year students across Heriot-Watt campuses. A multi-national, multi-institutional assessment of programming skills of first-year CS students found that many students do not know how to program at the conclusion of their introductory courses (McCracken et al., 2001). Some of the potential issues for this poor performance discussed in the study are constrained lab time, large classes and insufficient feedback. In previous work, we found that automated feedback can help students to become independent learners when faced with new programming tasks (Rieser, PGCAP report, 2014).</p> <p>The aim of this project was to develop technology and apply it within teaching in order to help first year CS students to become independent learners by using automated technology.</p> <p>The goal is to support students in mastering the art of programming during their first year at the university, and, at the same time, liberate staff from low-level and repetitive question from students, enabling them to focus on deeper learning problems.</p>

¹ In order to make the report self-sufficient we have included outcomes from the first phase (entitled 'Helping 1st Year CS Students to Become Independent Learners Through Automated Feedback') as well in this report. As a result, we will repeat some of the necessary background and summarise previous outcomes in this report.

Student Transitions

4 **Project overview & aims:**

The main objective of this project is to develop and evaluate a platform which automatically generates feedback for programming projects to **provide students with immediate formative feedback** and **help them to become independent learners**. Furthermore, this platform will provide students with a **uniform flexible learning experience across the different campuses**, and support staff by automating some of the marking.

The basis for the feedback platform is a technique known as *unit testing*, which enables us to write a set of tests. These tests can be used by students to automatically evaluate their code. Unit tests allow to automatically identify mistakes. However, these tests currently do not generate any meaningful feedback, which would allow students to learn from their mistakes. Here, **we study whether this automatic analysis can be used to generate formative feedback**, for example, in terms of clues and tips how to improve the code. This technique is called '*scaffolding*' (Vihavainen et al. 2013).

Our focus is on the Java language, which is used in our first year courses. Java supports unit testing via the JUnit framework.

Using JUnit, this projects builds an automated test harness for Java programming to generate formative feedback to the students. This type of automated feedback has been shown to be successful in, for example, teaching students how to code using Java in a different platform (Vihavainen et al. 2013). The tests are a piece of Java code, which are able to automatically find errors in the code and detect missing aspects.

We run two programming courses, Software Development 2 (SD2 led by Rieser and Lones) and Software Development 3 (SD3 led by Grov), in parallel during second semester for first year Computer Science students. Before attending these courses, students will already have had some basic introduction to programming in Java via Software Development 1, which is taught in the first semester. They are expected to deepen their knowledge and skills in these two more advanced courses.

Our original plan was to develop a joint coursework for Software Development 2 and Software Development 3, which depends on running the courses in parallel. There are currently discussions of restructuring the courses so that this will no longer be the case. As a joint coursework will limit generic use for others we have decided to prioritised this down. Our current focus is to make the tool as applicable to as many situations as possible, rather than targeting it at a specific piece of coursework. A key challenge is to find which type of exercises are amendable to automatic feedback, and which type will require help from lab helpers/lecturers. Once identified, this can be further extended to automate some of the marking process. As such, the final objective is to extend the framework with automated marking, and discover which properties can be automatically marked and which require human intervention.

5 **Activities and details of project steps taken to achieve aims:**

To achieve these goals, the following approach was taken (see Section 7a for further details):

Student Transitions

	<ol style="list-style-type: none"> 1. We developed a specification of the requirements for the unit tests based on the existing lab exercises. 2. We developed generic unit test cases, answering the question of “<i>what type of feedback can we provide automatically?</i>” 3. We applied the framework in teaching and evaluated the results. 4. Improved platform based on feedback from evaluations. <p>In future work we will:</p> <ol style="list-style-type: none"> 5. Continue to improve usability and flexibility of the tool to make its usefulness more wide-reaching. 6. Extend platform with automated marking functionality for coursework. 7. Test platform, debug, and review units with respect to their pedagogical quality regarding the feedback they produce (in collaboration with academic staff). <p>We are currently:</p> <ol style="list-style-type: none"> 8. Disseminating activity through appropriate channels.
6	<p>Key points including challenges your team may have encountered:</p> <p>The experiments have shown that unit testing is a suitable mechanism to provide feedback concerning shallow surface learning but prove difficult when applied to deep student learning. Deep learning involves the critical analysis of new ideas, linking them to already known concepts and principles, and leads to understanding and long-term retention of concepts so that they can be used for problem solving in unfamiliar contexts. In contrast, surface learning is the tacit acceptance of information and memorization as isolated and unlinked facts.</p> <p>In terms of exercises for computer programming, this means that unit tests are suitable to check whether students were able to solve a prescriptive task (shallow learning), but they are not suited to test whether the students is able to creatively apply the underlying principles to solve an unseen problem.</p> <p>The use of the tool in a lab with students, has shown that students found the feedback useful and would like to see similar tools in other courses. However student feedback has highlighted the following shortcomings: Automatic feedback should be more specifically geared towards the problem at hand. Students found the feedback too generic, which was perceived as not as helpful.</p> <p>In the future, we plan to investigate how to use interactive systems which can generate tailored language-based feedback on the fly, such as tutorial dialogue (Litman, 2013).</p>
7 a	<p>Describe specific project outputs so far:</p>

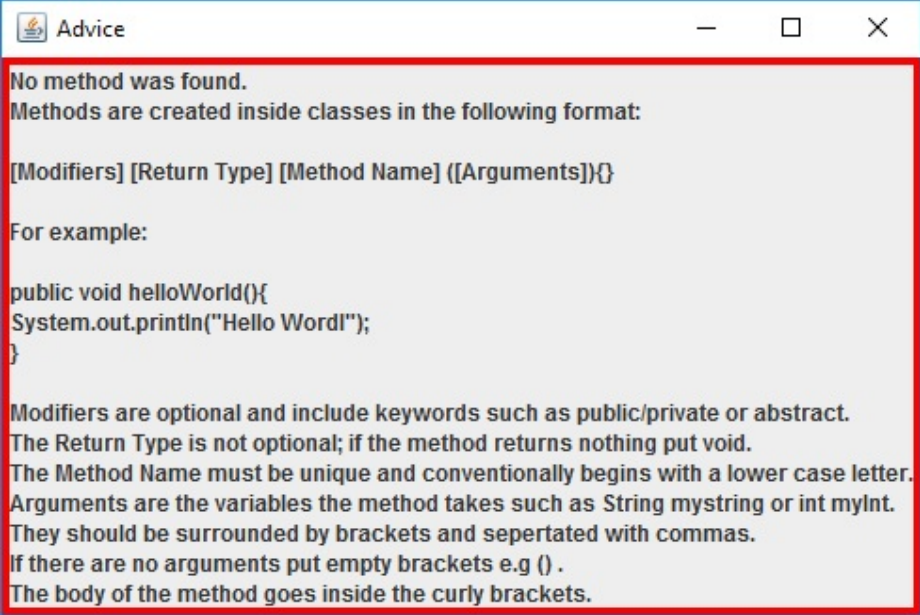
Student Transitions

We have created a series of tests using three libraries: Junit (described above), Reflect (enables us to examine runtime features of programs) and ASM (enables us to look at the methods in a class and the instructions comprising those methods).

Using the reflect and ASM libraries we have managed to create software tools that are capable of analysing whether student written classes exist; the fields, variables and properties those classes possess (whether they are abstract or inherit from another class for example); and the methods those classes contain.

Methods can also be tested for certain behaviours, such as whether they return expected output based on given input or whether a method calls itself, indicating recursion. If a student's code fails the test, the tool also displays messages to the student offering feedback and advice.

Using this range of abilities it is possible to test and provide feedback on a number of different lab tasks and enable us to check for the existence of specific Java features such as classes, variables and methods. Crucially, it allows us to check whether students have successfully accomplished simple program creation tasks. When one of these tests fails, messages are displayed to the student offering advice that is relevant to the failure point. For example, if the test fails to detect a given class, then a message is displayed offering information on the correct creation of classes. Common pitfalls are shown to the student, such as not closing brackets and the fact that class names are case sensitive.



The screenshot shows a window titled 'Advice' with a red border. The text inside reads: 'No method was found. Methods are created inside classes in the following format: [Modifiers] [Return Type] [Method Name] ([Arguments]);'. Below this is an example: 'public void helloWorld(){ System.out.println("Hello Word"); }'. Further down, it explains: 'Modifiers are optional and include keywords such as public/private or abstract. The Return Type is not optional; if the method returns nothing put void. The Method Name must be unique and conventionally begins with a lower case letter. Arguments are the variables the method takes such as String mystring or int myInt. They should be surrounded by brackets and separated with commas. If there are no arguments put empty brackets e.g (). The body of the method goes inside the curly brackets.'

The tests are nested which allows us to display advice that is relevant to only the currently encountered problem. This allows us to not overwhelm the student with information. This provides

immediate advice when it is needed to progress in a task; a feature of the educational *Scaffolding theory*.

One of the early drawbacks we had to overcome for these tests is that they look for specific class, method or variable names. This meant that students performing the coursework needed to be told specifically what to create, potentially limiting their opportunities for creativity and problem solving. This is not such a problem in the early stages of the course where the goal of the coursework is to simply teach the mechanical how-to of program writing but it will become more of an issue as the course progresses and the tasks, concepts and learning outcomes become more complex (see above discussion on shallow vs deep learning).

One solution we have found is to identify whether classes or methods exist by their output rather than their name because it is the output, and not the name, that is the relevant part. For example, we have created a sample lab that deals with recursively calculating various outputs, such as the sum of numbers or multiplication. It has been possible to create tests that take a class and analyse each of the methods in it, looking for the desired behaviour. If no method is found that fulfils the criteria of the lab task then the tool provides targeted feedback based on where the test failed. However, this too becomes problematic as tasks become more complex as every method would need every test applied to it.

There is also the problem, once again, of restricting the way students solve a problem if the tests are looking for methods that act in a specific way. It is also difficult to identify when a student solution is nearly correct, i.e. differentiating between different levels of problem solving, using this method. A missing semi-colon will carry the same weight as a fundamental error in the way a student is approaching a problem, which makes targeting advice difficult.

We have then applied this tool to (part of) a lab of SD3. In this lab student had to implement functions for the following 3 problems and using the tool for automatic feedback generation:

Recursive sum of numbers

Write a recursive method **sum**, which given a number n , returns the sum of all positive numbers up to n - that is, it computes:

$$\text{sum}(n) = 1 + 2 + \dots + n$$

Recursive multiplication using only addition

Write a recursive method **multiply** which given two (positive) integers m and n as arguments, computes $m*n$ using only addition (and recursion).

Computes a Fibonacci number

The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, You should write a method **Fibonacci** which given n as an argument, returns the n th Fibonacci number using recursion, that is

Student Transitions

Fibonacci(0) = 0, Fibonacci(1) = 1, Fibonacci(2) = 1, Fibonacci(3) = 2, Fibonacci(4) = 4, Fibonacci(5) = 5, Fibonacci(6) = 8, Fibonacci(7) = 13, Fibonacci(8) = 21, Fibonacci(9) = 34 and so on.

Afterwards we asked the students had to fill in an online (anonymous) questionnaire, which 41 of the students did. The full details of this can be found in section 11. Here we summarise the key points. A majority of the students **found the feedback useful** (either slightly or very) and that the tool was easy to setup and **worked as expected**. A majority of the student **would like to see more tools like this in teaching**. There were mixed views on how the use of the tool improved upon previous labs and whether or not the tool made the lab easier or harder to complete. A main reason for the negative comments seems to be technical. It should also be noted that this was the first time we tried to the tool and students used such a tool. One would therefore expect there to be issues that can be resolved next time. We therefore would expect students to be more positive if we try to run it again. The students had several comments on specific elements that didn't improve or features that they missed, and a common view was that the **feedback was too generic** and not specific enough for the task in hand. Here are some of the comments:

"It would be VERY helpful if the test program displayed my program's output"

"It worked nicely to not give too much detail to how you went wrong but enough to help you work out the problems"

"meant i could only use the tools advice and I couldn't self check my code or test it to see what would happen if i change the code myself"

"even when my code was wrong it would still say it was almost correct"

"All information was too generic to be helpful"

"did not show which operation failed, always showing the first method as the fail"

"Did not provide any useful feedback aside from whether you were right or wrong"

"dont give such stock feedback even though i had nothing written in the method and it said i was close"

"The tool provided generic information which did not particularly benefit me"

"Specific information as to what went wrong as opposed to the generic information currently being provided"

"spews out too much text"

"The tool ran just fine but I struggled to understand the feedback"

Given these results, we investigated using and extending the tool. Improving the usability of the tool was a priority for us as its purpose is to make the learning process easier for students, not more confusing. Previous versions of the tool used a mix of the console and Eclipse's JUnit result display to provide answers and feedback, examples of which can be seen below.

Student Transitions

The image displays two screenshots of the Eclipse IDE, illustrating student transitions in a programming course.

Top Screenshot: Shows the Eclipse IDE with a Java project named 'SD3TEST'. The main editor displays the code for 'Lab3Part1.java', which includes a recursive method 'sum(int n)' and a 'multiply' method. The console output shows the results of running the tests, indicating success for 'Fibonacci' and 'multiply' methods.

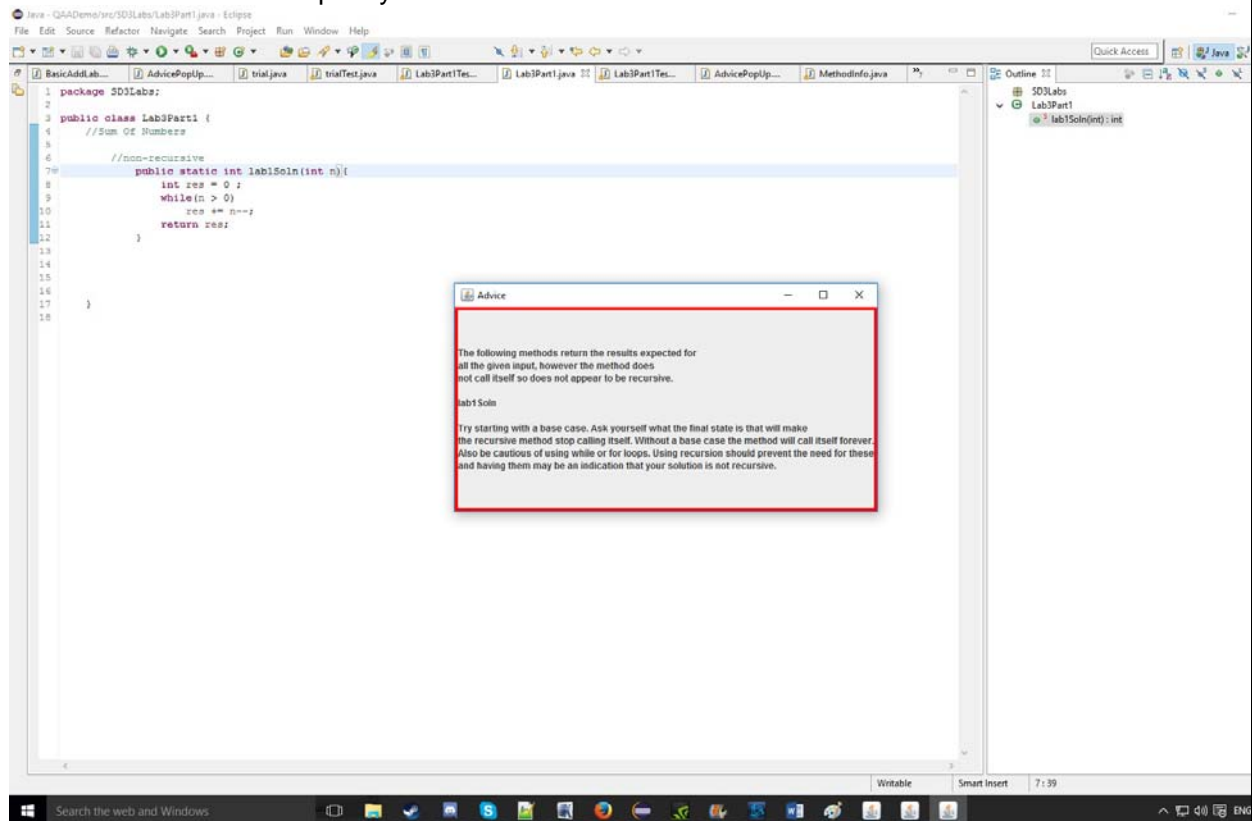
```
1 package SD3Labs;
2 public class Lab3Part1 {
3     //Sum Of Numbers
4
5
6     //non-recursive
7     public static int sum2(int n){
8         int res = 0;
9         while(n > 0){
10            res += n--;
11        }
12        return res;
13    }
14    //recursive
15    public static int sum(int n){
16        if (n==0)
17            return 0;
18        else
19            return n + sum(n-1);
20    }
21
22    // Multiplication using addition
23    //recursive
24    public static int multiply(int m, int n){
25        if(n==0)
26            return 0;
27        else
28            return m + multiply(m, n-1);
29    }
30 }
```

Bottom Screenshot: Shows the Eclipse IDE with a Java project named 'lab4'. The main editor displays the code for 'Animal.java', which includes an abstract class 'Animal' and a 'test' method. The console output shows the results of running the tests, indicating success for 'AnimalTest'.

```
1 package Lab4.foxesAndRabbits.GRAPH.agents;
2
3 import java.util.List;
4
5 /**
6  * A class representing shared characteristics of animals.
7  *
8  * @author David J. Barnes and Michael Kölling
9  * @version 2011.07.31
10  */
11 public /*abstract*/ class Animal
12 {
13     // whether the animal is alive or not.
14     private boolean alive;
15     // The animal's field.
16     private Field field;
17     // The animal's position in the field.
18     private Location location;
19
20     /**
21      * Create a new animal at location in field.
22      *
23      * @param field The field currently occupied.
24      * @param location The location within the field.
25      */
26     public Animal(Field field, Location location)
27     {
28         // ...
29     }
30 }
```

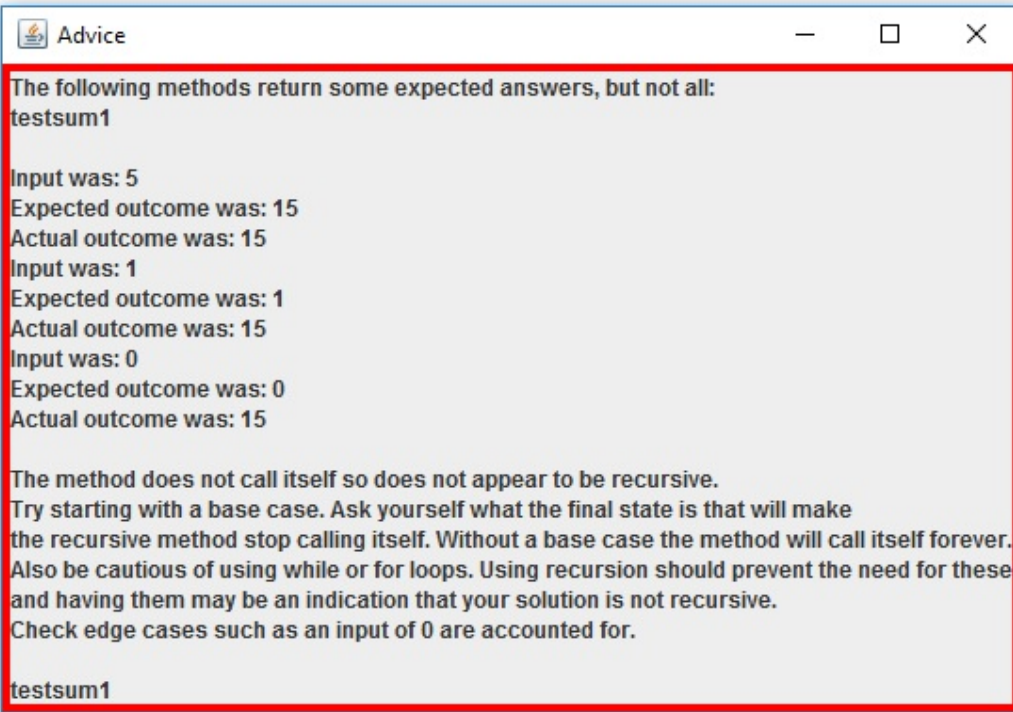
Student Transitions

Having the feedback appear in multiple places was not ideal, particularly as the console display can be shrunk, moved or closed by users of Eclipse, meaning the size and location of the feedback could vary. Student feedback talked of struggling to understand the feedback and there being too much text. As such we implemented a new method of providing advice to the student. Pop up windows deliver the targeted advice regardless of whether the console is open or closed. This window can be repositioned and kept open for reference as the student continues to work, unlike the console which would be quickly over-written.



Due to a common piece of feedback being that the responses were too generic we focused on making the advice more tailored to specific circumstances. For example, if a method returned some right answers and some wrong answers then the previous version of the tool only reported the method returned some, but not all, expected answers. No further details were given. A feature requested in the feedback was to show the answers that the test was expecting and the answers that it had received from the output of the students program. This was implemented and an example can be seen below.

Student Transitions



The following methods return some expected answers, but not all:
testsum1

Input was: 5
Expected outcome was: 15
Actual outcome was: 15
Input was: 1
Expected outcome was: 1
Actual outcome was: 15
Input was: 0
Expected outcome was: 0
Actual outcome was: 15

The method does not call itself so does not appear to be recursive.
Try starting with a base case. Ask yourself what the final state is that will make the recursive method stop calling itself. Without a base case the method will call itself forever.
Also be cautious of using while or for loops. Using recursion should prevent the need for these and having them may be an indication that your solution is not recursive.
Check edge cases such as an input of 0 are accounted for.

testsum1

A demo of the tool running can be found from the following link:

<https://youtu.be/otilrunmCBc>

One of the learning outcomes for SD2 is that students are aware of and can apply object oriented programming principles, such as *coupling and cohesion*. This has proven difficult to test for. One early idea was to capture the stack trace of the program to try and see how many steps were being taken but this has not produced anything useful so far. We have also investigated an alternative method, which is to count how many methods each class has and how many calls are made to other methods, which is possible using the ASM library. These numbers could be compared against an “ideal solution” to see if they fall within an acceptable threshold. However the concept of an “ideal solution” may be at odds with student creativity.

Furthermore, we investigated how to make the testing framework generic so that it may be used on multiple pieces of coursework. The *QuickCheck* library allows randomised test data to be created but the type of randomised data will still have to be manually set up as it will be specific to each coursework task. So far randomised test data seems to be more useful in catching edge cases in student code rather than creating a blanket set of tests for all pieces of coursework.

This project supports the school's L&T strategy across many objectives. Firstly, one of the main goals within the L&T strategy is the integration of modern technology into our teaching methodology to improve the student experience. This project directly tackles this by developing a more flexible environment and new methods to engage with the material. Secondly, as the school moves to teaching more courses at different campuses and through partnerships with other FE and HE institutions this project will help provide students with a uniform learning experience through the use of modern technology. Thirdly, the main development have been conducted by an undergraduate student, and such incorporation of students in research projects is a good example of the University's strategy of incorporating research in teaching.

Innovation

This project looks to use modern technology in an innovative way to foster engagement of the students with the material and provide a uniform teaching experience across various Heriot-Watt University campuses. The use of technology in the proposed project is an innovative teaching method.

- 9 ***Describe how you are sharing good practice within Heriot-Watt and beyond. In particular, give details of any liaison with other universities in Scotland (e.g. plans for papers, attendance at conferences):***

Impact on teaching:

This project has multiple immediate impacts on improving student experience within the department. The initial test platform will be utilised in the first year of teaching programming within the department. Verena Rieser, Michael Lones and Gudmund Grov are currently in charge of this teaching within the department and will assist the integration of the development platform within the teaching syllabus. This will have impact for students across several of Heriot-Watt's campuses.

The vast majority of courses in later years of the Computer Science programme (including Programming Languages, Language Processors, Data Structures and Algorithms, Web Programming, Computer Network Security, Artificial Intelligence, etc.) assume that students have acquired solid programming skills during their first year. Students that haven't done so, will struggle in their later years. Thus, providing an improved method of delivery of programming teaching will have a positive impact on these courses as well.

Furthermore, we hope to integrate this platform in a joint coursework between Software Development 2 and 3. This will allow students to integrate their knowledge in a joint project. Having an automated method to assess and provide feedback to students, also means a reduction of additional workload for the lecturers, while students profit from having more practical experience.

Dissemination:

9. The outcomes from the first phase was disseminated at the HWU Learning & Teaching symposium on 7 October 2015, and we plan to present the final outcomes of the two phases at L & T in October 2016. We are also planning to submit a paper to suitable conference or journal

Student Transitions

describing our tool and the result of our evaluation. For example, we are in the process of preparing a paper submission for the Creative Education Special Issue on "Teaching, Learning and Assessment" (Submission Deadline: June 28th, 2016).

Internal & external liaison:

Internally, we plan to present this framework to our colleagues at the CS department, but also from Math and EPS and Dubai, and seek opportunities to collaborate. In particular, within MACS we have regular meetings discussion teaching and technology where we can present this. We will also seek collaboration opportunities throughout the university.

To date we have not cooperated directly with another University, as the tool is very specific for the two modules we are teaching. However, we have close contacts to other Scottish Universities through the Scottish Informatics and Computer Science Alliance (SICSA), the Scottish Crucible (Rieser, Lones) and the Royal Society of Edinburgh Young Academy (Rieser). For example, Verena Rieser is part of the RSE Young Academy working group "Computing in Schools", where this framework will be a useful tool to explore. This is future work.

1 **Next steps:** 0

As it stands, we have been able to create satisfactory tests for labs comprising of discrete tasks and provide feedback for them. We have tested them in a lab, and analysed the feedback. This has indicated a need for more specific feedback on the tasks students are solving, and a next step will be to investigate how this can be done.

We are now working on combining tests such as these so that it is possible to automatically provide feedback and marks for a more complex piece of work such as a class project. To this end we will continue to re-factor the code to make the addition of more types of test easier, and to make combining them into larger tests simpler with the eventual goal being a robust way of creating test for large end of year scale projects. The ongoing goal is to make the tool as flexible and easy to use as possible so it is accessible for people beyond the Heriot Watt Computer Science department. As discussed earlier, the tool must be suitably flexible and easy to use for the person setting up the test as well as those using it for feedback. A challenge would be to study how much of the tests created can be automatically marked. This is ongoing work in this project. We do not believe it is feasible to fully automate this process, but it would be interesting to see just how far automated grading could go. The rest of the coursework will be graded manually. One way of testing the automated marking, will be to grade the automatic part manually and see how the results compare (at least for a selection of solutions). We would then be able to use this data to detect flaws and limitations in the automated marking procedure. The automatic marking could also be further developed. Measures should be taken to prevent the tests being tampered with to provide false positives (i.e. with "checksums"). Further functionality could be added to compare students' answers to check for plagiarism. Once security is robust enough it should be possible to have the tool automatically update the students marks, further freeing up the lab-helpers to assist the students who were really struggling and also reducing the chance of a students marks not being updated. We could add further functionality e.g. coupling and coherence tests or loop monitoring. It should also be possible

Student Transitions

to use the reflect library to get all the classes in a given package. This could be used to leave the students free to name their own classes. Plus any other desired functionality that became apparent during the testing process. However, we do not believe that this will be feasible during the lifetime of the project.

Another piece of functionality we discussed including was the concept of timed feedback. If the tool notices no input for a while it would be good if it could ask the student if they needed assistance. This would need to be tested to prevent it being intrusive or annoying but it would possibly prevent stuck students staring at a screen unsure how to progress.

This initial exploration of using an automatic testing harness to provide feedback on programming performance has also opened interesting research questions which we will not be able to complete during the lifetime of the project but hope to explore in future work. In particular, we want to investigate how to support deep learning using tailored automated feedback by, for example, using interactive tutorial dialogue technology (Litman, 2013).

1 **Additional information:**

1

Bibliography:

Michael McCracken, Vicki Almstrum, et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In Working group reports from ITiCSE on Innovation and technology in computer science education. ACM, New York, NY, USA, 125-180.

Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding students' learning using test my code. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education (ITiCSE '13). ACM, New York, NY, USA, 117-122.

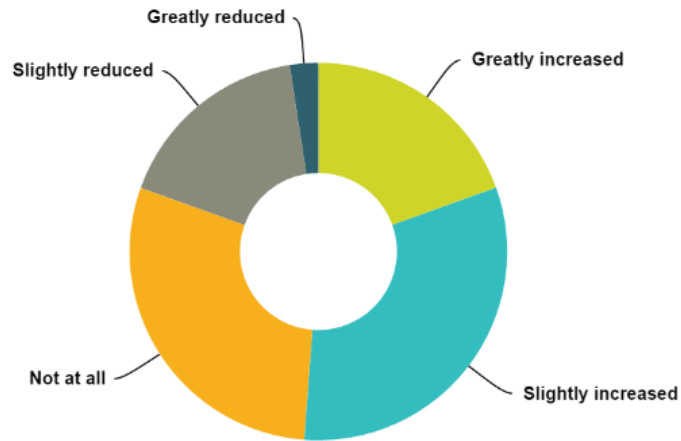
Diane Litman. Enhancing the Effectiveness of Spoken Dialogue for STEM Education. Proceedings Workshop on Speech and Language Technology in Education (SLaTE), pp. 13-14, Grenoble, France.

Data collected from questionnaire after lab:

Student Transitions

How did the tool affect the time it took to complete the lab compared to previous labs?

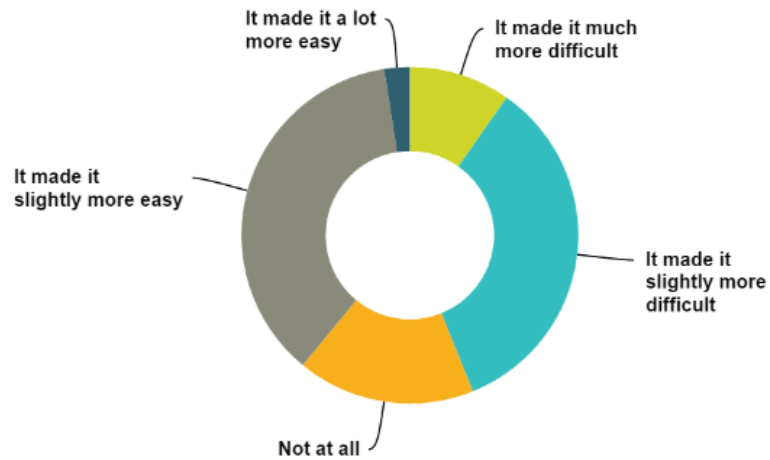
Answered: 41 Skipped: 0



Student Transitions

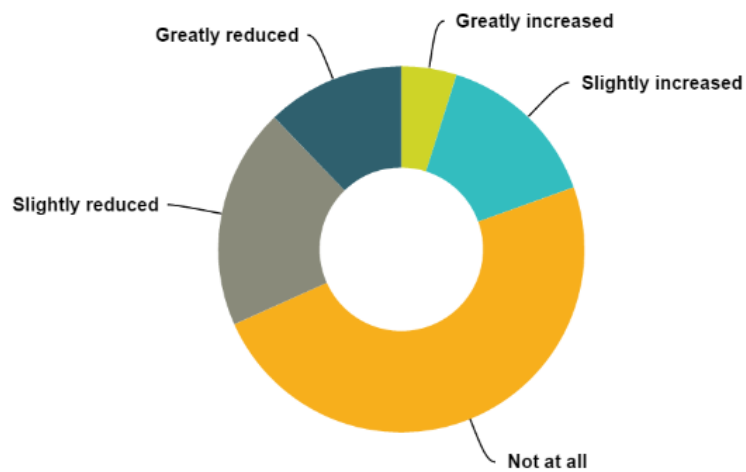
How did the tool affect the difficulty of the lab compared to previous labs?

Answered: 41 Skipped: 0



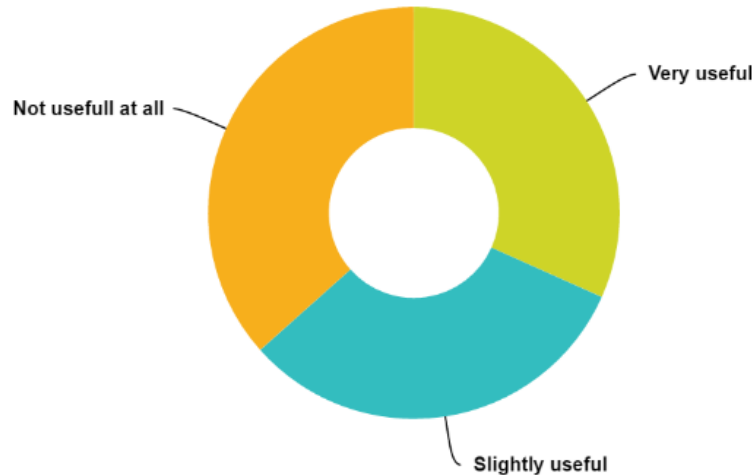
Compared to previous labs, how did the tool affect the length of time you spent waiting to see a lab helper?

Answered: 41 Skipped: 0



How useful did you find the text feedback the tool provided?

Answered: 41 Skipped: 0



Please provide any areas of text feedback you feel the tool did not provide:

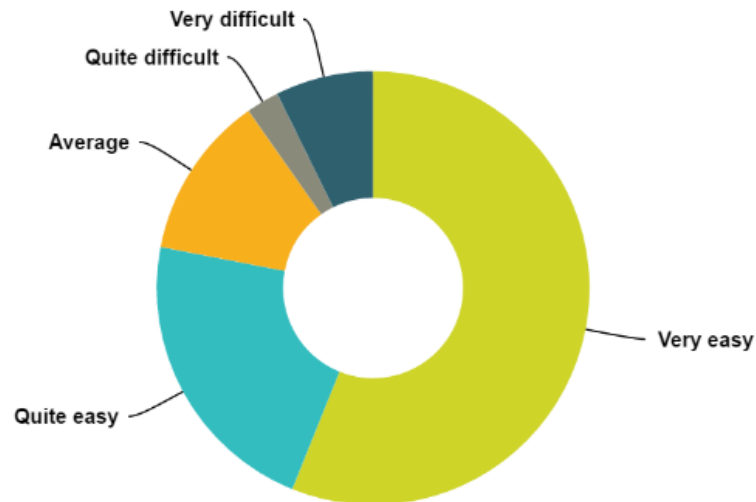
- Didn't have an explanation of tool before hand had to find out from a friend.
- It couldn't detect my methods and therefore was, sadly, useless
- More detail regarding "the following methods return some expected answers, but not all" - it would be useful to see which details are correct
- Anything specific to any errors. The information was just generic.
- I was not happy with the use of the feedback tool in this lab. As I was unable to see the output of my program (Without writing additional methods), I often had no idea why my program was not passing a test. The test also resulted in me being unable see the effect changes I made had on the output, which made coding a case of trial and error. At times, even though I felt my program correctly fulfilled all requirements of a task, the test still failed. I personally prefer how previous JUnit tests were done, as it allowed me to see how specific input data was handled. It would be VERY helpful if the test program displayed my program's output, and the expected output. I feel this would resolve the problems mentioned above.
- didn't tell me test needed to become static
- It worked nicely to not give too much detail to how you went wrong but enough to help you work out the problems
- It could provide us with the correct answer, that would be great (joking)
- It meant i could only use the tools advice and I couldn't self check my code or test it to see what would happen if i change the code myself. Not a big deal but slightly annoying.
- It was slightly unclear what the feedback was actually telling me was wrong.
- Sometimes even when my code was wrong it would still say it was almost correct.

Student Transitions

- IT SAID IT WAS RIGHT WHEN IT WAS WRONG!! IT ALSO DIDN'T WORK IF THE METHOD WASN'T STATIC OR PRIVATE.
- it was very unclear how the test was checking for correct results and as such you couldnt tell if the method was incorrect or the test was just expecting something different
- the tool was difficult to understand, and it didn't provide specifications for correcting the code
- Anything relevant to the actual error in the code. All information was too generic to be helpful.
- The code did not show which operation failed, always showing the first method as the fail. This was due to a small copy paste error
- Did not provide any useful feedback aside from whether you were right or wrong.
- dont give such stock feedback even though i had nothing written in the method and it said i was close
- The tool provided generic information which did not particularly benefit me (i.e. describing how to write a new method despite a method already existing but it being able to find it) the inbuilt failure trace method in JUnit was more helpful as it provided line information.

How easy was the tool to setup and run?

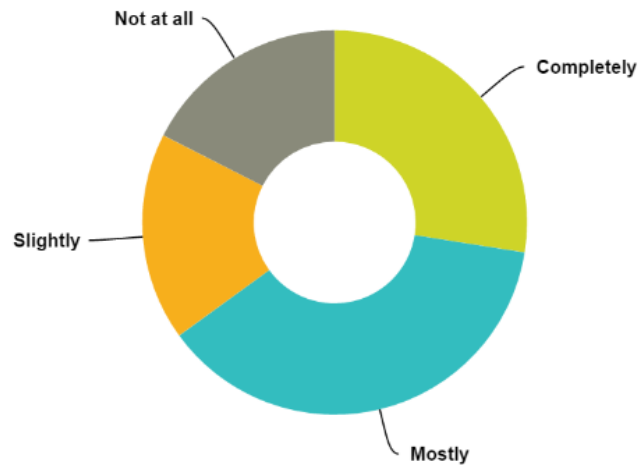
Answered: 41 Skipped: 0



Student Transitions

Do you feel the tool worked as intended?

Answered: 40 Skipped: 1



Please describe any technical issues you had using the tool.

- It was saying I was not considering the inputs zeros when I had. The problem was that it was was running the wrong method in the third test.
- no issues
- As said before. It couldn't detect my methods.
- I had working procedures doing the tasks in the format it asks yet it didn't pick up on them at all.
- i think is not working
- none
- At one stage, one of my methods continued to fail, even though I was sure it was correct. After approx 1 hour of making changes, I realised I had forgotten to make the method static (I admit this was my own stupidity, but still very annoying).
- static methods are required but not told that is the case
- None
- none
- The tool ran just fine but I struggled to understand the feedback.
- It required us to change the way bits of code work without explaining why. You had to change every method to Public Static, without ever being told to.
- n/a
- it marked a correct method as incorrect
- Unware that methods had to be static
- Doesn't always work
- Delivered the wrong feedback as was mentioned above.
- Incorrect importing
- the stock answers
- did not recognise my methods to me the ones it is looking for
- wouldn't see the method as recursive without the static modifier

Student Transitions

Is there anything you feel is missing or unnecessary in this tool?

- No
- n/a
- a lab helper
- it spews out too much text
- Useful advice.
- No
- I feel the tool automates the process of writing Junit tests. Clever, in concept perhaps for SD1. But since we are learning how to write junit tests, perhaps we should write our own.
- no
- the stock answers
- commentary.
- Specific information as to what went wrong as opposed to the generic information currently being provided (if failure trace in JUnit was also left on the tools usefulness would be increased by several orders of magnitude)

Would you like to see tools like this for other labs?

Answered: 41 Skipped: 0



Please be aware that this report will be publicly available on the University's webpages
Please **complete in Word** and return to Mirren.McLeod@hw.ac.uk by 30.5.16 (reports not completed in Word will be returned)