A unit testing approach to building novice programmers’ skills and confidence

Jacqueline L. Whalley and Anne Philpott
School of Computing and Mathematical Sciences
AUT University
PO Box 92006, Auckland 1142, New Zealand
jwhalley,aphilpott@aut.ac.nz

Abstract
This paper discusses the integration of unit tests into a first semester programming course. The students were supplied with unit tests to support their learning and assessments. A questionnaire was completed by the student cohort about their use and perceptions of these unit tests. As a result of both the students and our experiences we examine the advantages and disadvantages of introducing unit tests early and make some pedagogical recommendations for the introduction and use of unit tests in first year programming.

Keywords: testing, novice programmers, assessment.

1 Introduction
Novice programming students struggle to take a structured approach to gaining a full understanding of what is required of their program. This initial inadequate consideration then leads to haphazard tinkering in an attempt to produce a piece of code that does what it should (Perkins and Martin, 1986). For many students the barrier to identifying and fixing bugs or extending and refactoring code is a negative experience and they simply "abandon all hope of solving the problem on their own" (Perkins et al., 1989, p.265). Several factors motivated the early introduction of unit tests. Student numbers in our introductory programming course had more than doubled over the last two years, increasing the student to teacher ratio considerably. One of the most noticeable consequences of this change was that students were not receiving immediate feedback during their laboratory sessions and often waited long periods of time to receive assistance. Students were also receiving less individualised feedback and support when undertaking their take home programming assignment. At the same time the sole lecturer on the course was swamped with marking. This resulted in delays in students receiving critical early feedback. As a result of the increased marking load student tutors were employed to assist with the laboratory teaching and assignment marking. Along with the introduction of tutors there was then the need to ensure that the marking was consistent across markers.

It was anticipated that the introduction of unit tests would have several advantages. In particular, that marking would be more efficient and consistent and that students would be provided with a mechanism to get immediate feedback on their code writing.

It has been suggested that students should have more success if we can move them away from a trial and error approach to writing code and towards a reflection in action approach (Schön 1983). Subsequently, Edwards (2006) noted that software testing tends to move students towards a reflective approach to programming. We propose that by supplying students with unit tests we provide structure for them that supports them in reflective tasks such as requirements extraction and semantic bug finding. One of the primary goals, of this work, was therefore to provide a tool that guides the students through their initial design process. However, Ginat (2007) found that test instances supplied later and fed to the students one at a time had led to hasty design and futile patching. In the approach reported here foreknowledge of relevant test cases is provided and we believe that this should trigger the observation and consideration behaviour that Ginat found lacking in his experience of using unit tests with novice programmers.

2 Background
Test units have been used in introductory programming courses in the past. However, generally the approach reported in the literature is focused on test driven development (TDD), with students writing the tests. Desai et al. (2008) provide a useful survey of results from these studies. Marrero and Settle (2005) introduced a test first early approach to their assessments for novice programmers. As the first stage of their assessment students were required to write unit tests and submit them ahead of the actual code submission. They suggest that despite the extra load of learning to write unit tests, an emphasis on testing is beneficial. An evaluation of a test-driven learning approach, where again students were required to write their own unit tests, found that there was a strong reluctance on the part of novice programmers to adopt a test-first approach (Janzen and Saiedian 2008).

Edwards (2003) identified several roadblocks to integrating TDD into an early programming paper:
1. Knowledge
   a. Students are not ready for testing until they have basic programming skills
2. Time
   a. Is limited and there isn’t enough time to also teach testing
   b. Teachers already have their hands full grading code correctness, it may not be feasible to also grade unit tests
3. Value
   a. Students must be able to see the value in the non-functional code

Studies in general have found that novice programmers find TDD hard largely because they are struggling to find purpose in the functional code, so testing it is difficult. Despite these difficulties studies to date have reported that courses using TDD increased student confidence (Edwards 2004, Janzen 2008, Kaufmann 2003, Muller 2001) and increased code comprehension (Jones 2004, Muller 2002, Janzen 2006).

Because the students in the course are assumed to have no prior programming experience, and, in light of the roadblocks to integrating TDD, it was decided to supply the test units rather than require the students to write them. We propose that this is a way to introduce good practice early without adding too much additional learning load for the students. The focus is on students using the unit tests as a tool that will help them see the value in testing their code and on the students being able to read and comprehend unit tests.

Concurrent with our investigations Cardell-Oliver et al. (2010) were investigating the use of automated testing tools to improve the quality of programs produced by software engineering students. The suite of tools they investigated included JUnit for assessing the correctness and Checkstyle to evaluate the readability of student code. For lab exercises they provided students with unit tests, but for the assignment they required the students to write unit tests. On reflection they suggested that for first year students using instructor test cases is more appropriate than an approach where students are required to write the test cases. Their experience led them to believe that this approach focuses students’ attention on the quality of their programs. Additionally, they found that, with supplied unit tests, most students were able to write fully compliant functionally correct code. This work lends support to our current approach to integration of testing early with instructor supplied unit tests.

4. Teaching practice

First semester programming students attended two one hour lectures and a two hour supervised laboratory session each week. They were taught Java using an objects-first approach and the BlueJ development environment. All the lab exercises that required code writing were supported by JUnit test cases that were supplied to the students. In the third week of the course the students were given a hands-on, instructor led introduction to the running and reading of test units. They were also introduced to common novice errors that might result in compiler errors and test case failure. The BlueJ debugger was used to demonstrate how to discover the source of the error using the unit test. A large part of this session was devoted to learning how to read the unit test cases, elicit the specification for a method and interpret assert statements and test failures. Students were taught that unit tests are a tool.

The lab exercises supported with unit tests were based on a kiwi world project that simulated kiwis and their zoo enclosure. For the kiwi project students were provided with existing code in the first week and in each subsequent week built on that project. For each lab they were supplied with a new set of unit tests that they needed to copy into their project to test the new functionality they would be adding.

Because the unit tests were tightly coupled with the code the students were testing we found it useful to provide the unit tests with the test cases commented out. As the students completed each task they then uncommented the related test cases and compiled and ran the tests.

The students also worked on a three stage take-home assignment. For this assignment the students were provided with a set of unit tests and starting code for each stage. The assignment required the modelling of an organic pig farm with a waste management system that supplied fertiliser to the farm.

In the first stage the unit tests supplied had detailed in-line comments and test case header comments to assist the students to read and comprehend the unit tests. Additionally a detailed assignment specification that specified in words the method signatures and any rules that the method output should meet was provided. At the second stage the assessment prescriptor provided was more general and lacked detailed descriptions of the method signatures so students had to read the unit tests before starting to write their code in order to elicit some simple requirements for their methods. From stage 2 extended inline comments and header comments were removed from the unit tests.

For the final stage of the assignment the prescriptor was a general overview of each problem to be solved. The students needed to read and understand the unit tests in order to get a full specification of the code to write, including the algorithmic rules.

4 Results

In the tenth week of the 12 week semester students were asked to complete a questionnaire (see Appendix A) about their use and perceptions of unit tests. We asked them to focus solely on the use of unit tests in their assignment. Of the 172 students in the class, 133 students provided responses for analysis (Figure 1).

Question 1 asked the students whether or not they used the unit tests. Ninety nine percent of the students responded that they used the unit tests supplied. Of that 99%, 88% said they read the related unit test cases (Q2) when preparing to answer a question and 86% said that this helped them to decide what the code they wrote needed to do (Q3). This high percentage of students indicating they used the tests is not surprising given that from stage two the assignment structure forced the students to elicit some of the requirements from the tests.

When asked if they found the unit tests helpful for debugging code 87% of the students said that they were helpful (Q4).

There was a general consensus, 94% of students, that unit tests gave the students confidence that their code was correct and complete (Q5). The remaining students for whom it did not provide confidence said that they found unit tests hard to understand and read.
To elicit how the unit tests were being used the students were asked how frequently they used unit tests and how they used the unit tests. Question 1B asked, if they used the unit tests, how many of the unit tests they ran. Seventy two percent of students responded that they ran all of the tests, 17% ran most of the tests and the remaining students said that they ran a few of the tests.

Question 6 asked at what point in the development of their code the students ran the unit test cases: (a) as soon as the functionality the method tests was implemented, (b) on completion of the question, (c) on completion of a related set of questions or (d) on the completion of the assignment. The majority of students responded that they ran the relevant unit test cases when they had completed a question while a minority of students ran them at the end of the assignment (Figure 2).

In Question 7, the students were asked how often they ran the unit tests for a marked exercise: (a) 0-5 times, (b) 5-10 times, or (c) 10 or more times. This question appears to have been misinterpreted by some students. As they ran the unit tests at the end of each question they must have run them more than 5 times, given that there were more than 5 questions, but some responded 0-5 times. Despite this complication 56% of the students stated that they had run the tests more than ten times showing that the tests were used frequently during the development process.

The students were also asked three open questions about their reaction to the use of unit tests in their assignments (Questions 8-10, Appendix A). A content analysis approach was adopted to evaluate the student responses (Stemler, 2001). The categories for content analysis for each question were established using the emergent coding guidelines (Haney et al., 1998). This categorisation system was used by two independent raters to classify student responses. There was very strong inter-rater agreement on the classification of the responses to all three questions1.

It should be noted that some students made more than one statement in their response to a question. Each statement was classified to a category. Therefore the total number of ratings for a question can exceed the number of students.

The following subsections report on the results of the application of the content analysis to the student responses. The implications for teaching practice are discussed in a later section.

### 4.1 Question 8

Twenty one percent of students reported that they had found the unit tests helpful for debugging their code.

"Helps as a starting point for debugging"

Twenty-six percent of students reported that unit testing helped them overcome the difficulty of knowing what was required. They mentioned finding it useful for identification of the correct method signature and understanding parameters, return types and values.

"If I was in doubt about a task I read the unit test first."

Thirty one percent of respondents reported that using unit tests helped them confirm the correctness of their code. Respondents in this category also mentioned that being able to confirm the correctness of their code gave them confidence and motivated them to complete the assignment.

"It gives me confidence <that> I got the question right ... If it is wrong at least it will show me which part is incorrect."

"The unit tests help me feel confident my code was correct and complete...<they> let me know that I am on the right track.

"...knowing the previous answer is shown to be correct it give a lot of heart to answer the more tougher ones."

Some students mentioned finding unit tests particularly useful to identify code that runs but does not fully meet the specifications.

"When I make a semantic error it was easier to identify using the unit tests"

Seventy one percent of students provided an answer to question 8.

### 4.2 Question 9

The great majority of the students did not mention anything that they did not like or find helpful about unit testing.

---

1 Q8 $\kappa = 0.915$; 95% confidence interval from 0.862 to 0.969.

Q9 $\kappa = 0.934$, 95% confidence interval from 0.861 to 1.008.

Q10 $\kappa = 0.939$, 95% confidence interval from 0.880 to 0.998.
Sixteen percent reported that they found the unit tests or the results of the tests complicated and the error messages difficult to understand. They said that they had difficulty learning from the unit tests results. Some had problems knowing what was being tested and comprehending the unit test code. Almost four percent of students mention frustration with unit tests repeatedly failing their code.

"A little complicated to get your head around it at first"

"When errors and a fail occurred I could not always understand the tests"

Two percent of students did not like the mismatch between the specification provided by the unit tests and the assignment prescriptor. One student was not happy about losing marks for code quality when all the unit tests passed.

4.3 Question 10

Most students did not respond to the opportunity to make any further comment about the use of unit tests. Twenty five percent of students gave an additional positive judgement regarding the use of unit tests while another five percent gave a further general negative comment about the use of unit tests in their assignments.

Five percent of students made recommendations regarding the teaching and presentation of unit tests for their assignments. The recommendations included:

• Providing more comments in the unit test code.
• Providing more explanation or teaching of unit tests
• Wanting the opportunity to write their own unit tests.

5 Implications for teaching

We identified several advantages and a significant overhead in this approach. The advantages identified included:

• Students are more confident of their submission as they now know their code meets correctness requirements.
• Students gain more immediate feedback as they work. This immediate feedback is important as it means that students are not left floundering in uncertainty and are able to recognise when they need help, and to seek that help.

"They are a great way of getting feedback and direction while writing code particularly when you are out of school."

• Increased student confidence was observed and there was an increased engagement in online peer support.

"Thank you for including unit tests. This helped me understand java programming better."

• The course leader was confident about the consistency and correctness of marking by student tutors.

It should be noted that unit tests are not a substitute for student-instructor interaction and we found that our lab sessions and on-line discussion forums were busier than usual just as observed by Cardell-Oliver et al. (2010). We believe that this is because the unit tests provided a means not only for students to identify problems in their code but also a means to express the error in a way that allowed others to answer their questions.

The significant overhead was the increased preparation necessary to provide a clear and complete set of unit tests for all lab and assessment work. However this had the advantage of highlighting issues regarding the assessment requirements that may have otherwise been missed.

5.1 Presentation of material

Our results indicate that the adopted approach of explicitly teaching the reading and running of tests early in the course was effective for most of the students. However a minority of students (16%) still indicated they found the tests or the generated error messages difficult to understand. It may be beneficial to explicitly assess test reading and error message interpretation early to identify and provide additional support to this group.

5.2 Motivating reading and use

Use of the tests must be made as straightforward as possible for the students. Encouraging test reading by enforcing discovery of requirements from the test seems to have been an effective strategy and no doubt contributed to 99% reporting using the tests.

Sequencing of tasks and blocks of tests can prove a challenge in assessment design and this contributed to the preparation overhead identified. When students complete a part, or even a subpart of a question, the tests, or lines within tests that need to be enabled should be clear. We believe that commented out lines or tests supported by clear indicators of the question part they applied to was a better approach than the confusion that might have arisen from multiple version of tests being provided. Any mistakes in sequencing or lack of clarity in the comments however did confuse students.

"I found unit tests/questions out of order frustrating when they were dependant on each other"

5.3 Test coverage

It may be necessary to write some tests cases that would not normally be required to support this approach. This can arise from a need to support task order or allow for students who don’t complete all questions or parts of a question. For example, an accessor method might normally be adequately tested by a constructor test and the use of the accessor when testing other public methods. Students answering a question that requires them to write an accessor method however will require explicit tests that allow them to confirm the correctness of this method and discourage hard coding of return values, an approach taken by some students in this cohort. The unit tests written therefore need to not only cover standard test cases (e.g. data boundaries etc.) but need to take account of task sequence and common novice errors.

5.4 Ensuring quality not just correctness

Care must be taken to avoid an observed tendency to approach assignments in a tick list fashion – tests-pass
good – move on to next task. Students who take this approach fail to take that important reflection in action step and ask themselves "can I write my code better?". A well designed marking schema and the use of supporting tools such as Checkstyle can mitigate this. Clear expectations about code quality and readability should also be communicated to the students at the start.

5.5 Hack and Tinker

While unit tests help the majority of students in the class and give them focus, the weak students still tinker with their code rather than using the guidance of the unit tests. We believe that this is because their knowledge of programming is so fragile they cannot use or understand even the simplest constructs and therefore cannot use or understand the unit tests. We have observed that these students often use the unit test code as a template for their code and copy and paste the unit test code into the class and tinker hoping to get something working.

6 Conclusion

Taking a unit testing approach to learning, teaching and assessment of novice programming was effort well spent as we believe it benefited both the learners and the designers and assessors of the course. In particular we consider that using unit tests in this manner provides support for the often neglected middle learners. By far the majority of experiences reported by these students were positive. We acknowledge that extra effort is required in the preparation of course material but the gain in the quality and clarity of the assessments designed was a bonus we had not predicted.

The unit testing approach however did not decrease the time required for marking as much as we had hoped. There is still a substantive part of the marking time being used to give feedback on the quality of the student code. Supplementary tools, as previously mentioned, may be necessary before there is a significant reduction in the marking time.

The ability to read and interpret unit tests was assessed in the final examination but, given the 16% of students reporting that tests were difficult to read or understand, it may be more helpful to assess this skill earlier in the course and follow up those who are struggling.

We have reported on the students perceptions of the tests and their use. It would also be useful to investigate the way in which students actually use the tests. The impact of this approach on students’ subsequent behaviour would also be interesting. Do students who used the provided unit tests demonstrate a better appreciation of the value of such tests in the future? Does the early test reading behaviour enforced affect their subsequent ability to write good unit tests?

7 References


Appendix A

The questionnaire
(Note: The spaces provided for student responses has been removed)

1. Did you run any of the unit tests when completing the Marked Exercises? YES | NO
   A. If not, why not?
   B. If you did how many unit tests did you run? ALL|MOST|A FEW

2. Did you read the related unit test(s) when preparing to answer a marked exercise question? YES | NO

3. Did the related unit tests help you to decide what the code you wrote needed to do? YES | NO

4. Were unit tests helpful when you needed to debug code? YES | NO

5. Did using the unit tests help you to feel confident that your code was correct and complete? YES | NO

6. When did you run the unit test methods? (circle the most appropriate option)
   A. As soon as the functionality the method tests is implemented
   B. On completion of the question
   C. On completion of a set of related questions
   D. On completion of the marked exercise (assignment)

7. How often did you run the unit tests for a marked exercise? (circle the most appropriate option)
   A. 0-5 times
   B. 5-10 times
   C. More than 10 times

8. Identify a situation when you found a unit test useful. Write a brief account of that situation in the box below.

9. Is there anything you did not like or find helpful about using the unit tests?

10. Is there anything else that you would like to mention about the use of unit tests in your Programming 1 assignments?