Assessing with a unit test framework: variations of approach

Dr. Mike Lance
School of Computing
Christchurch Polytechnic Institute of Technology
lancem@cpit.ac.nz

Amitrajit Sarkar
School of Computing
Christchurch Polytechnic Institute of Technology
sarkara@cpit.ac.nz

Ranran Bian
School of Computing
Christchurch Polytechnic Institute of Technology
rab446@student.cpit.ac.nz

This quality assured paper appeared at the 1st annual conference of Computing and Information Technology Research and Education New Zealand (CITRENZ2010) incorporating the *23rd Annual Conference of the National Advisory Committee on Computing Qualifications*, Dunedin, New Zealand, July 6-9. Samuel Mann and Michael Verhaart (Eds).

Abstract
This work describes two different uses of a Unit Testing Framework for automated marking of programming assignments. Usually unit testing focuses on verifying the correctness of individual methods. Here we firstly show how to use unit tests to give novice programmers feedback as they learn how to code simple data-centric Creation, Retrieval, Updating and Deletion (CRUD) tasks. Following this there is an explanation of how advancing novice programmers can be guided to create robust methods in a complex system through the feedback from automated acceptance tests. These are novel variations of the standard use of unit tests for automatic assessment of programming assignments and showcase the possibilities for vocational focused programming courses.

Keywords
automated marking, unit testing framework, testing, education, programming assessment

Introduction

Many people have developed software to automatically mark programming assignments (Doulce et al., 2006). The now well-established advantages of such automatic marking (Higgins et al., 2005) is a reduced workload for teachers and an improved learning experience for

students due to rapid feedback about their coding. Most programming assignment marking schemes are based on language specific code (eg QuizPACK as reported by Brusilovsky & Sosnovsky, 2005) which cannot be readily used with other than the target programming language. Such systems are also typically closed-source and proprietary. If teaching a programming language which differs from the tool's target language the intellectual investment in test authoring cannot be reused.

All published schemes focus on computer science course programming assessments. Very different types of assessments are used in computing courses in vocational and applied education. Such courses are industry focused and aim to ensure that students are work ready when they graduate. Assessments appropriate to vocational course are not well covered by any existing 'testware'. The goal of this paper is to show how automatic assessment of business focused programming assignments can be implemented. Two different teaching situations are described in which unit tests have been used to automatically mark programming assessments.

1. Automatically Assessing CRUD

The usual use of a unit testing framework for assessing programming (Edwards, 2003; Edwards, 2004; Helmick, 2007; Lance, 2005) is to have students program to a set interface and require them to write tests which establish the correctness of their code. In contrast business computing courses often have an early emphasis of getting student to "…write scaled-down versions of standard programs" (PR50n course prescription) which involves an emphasis programs as both structures and algorithms (Content can include: Data structures including arrays, simple file handling, interactive dialogue, commercial report programs, data validation, simple exception handling, simple file updates, all standard syntax options, efficiency considerations). The standard description (Wikipedia) of such programming is C.R.U.D. as it involves Creating, Reading, Updating and Deleting tasks. Assessments of business specific programming contrast strongly with the existing work on automatic marking. Work to date has emphasized "… programming-in-the-small….implement specific modules (viz., classes), for which the instructor provides the appropriate specification (viz., interfaces), not large-scale programs." (Tremblay et all 2006). Existing testware is suited to situations where "almost all the programming assignments that students construct are command-line based…" (Douce et al., 2006). The following demonstrates that automated marking of CRUD assessments is possible.

Writing code which correctly creates, reads, updates and deletes data with simple data structures typically follow the Whole-Part Pattern (Buschmann et al., 1996). Figure 1 below illustrates such a situation: a Company has many SalesPersons and many Products.

Figure 1

Students can be exposed to a range of different variations of the pattern (e.g. chains of data, hubs with surrounding data) and over a series of practical

exercises learn (by repetition and generalization) how to code typical CRUD operations. Figure 2 shows assignment instructions for such a task.

4. Write a Jadescript loadTestData()method that
- Calls the deleteAll() method you wrote for question 3.
- Creates an instance of the Company class
- Creates the following 4 SalesPersons, using the addSalesPerson() method that exists in the Company class

ID	Name	Salary	Commenced
14	Jim Peters	39,000	1998
11	Helen Brash	47,000	1999
12	Don Clarke	37,000	2003
13	Winston Anderton	42,000	2002

(3 marks)

Figure 2

While coding these tasks students can get formative feedback from a suite of unit tests. The students do not need to understand the internals of the unit testing framework but can be expected to respond to assorted tips and warnings and error messages until they have produced correct code. The scope of test coverage includes: checking if data structures exist, making sure nothing which was initially provided has been broken, making sure published interfaces (eg set, add and get methods) being followed, checking that expected data is correctly populating the database, ensuring correct data types are used, and that all references and collections being used correctly. Figure 3 shows a typical unit test which establishes that a data structure has been created with appropriate properties.

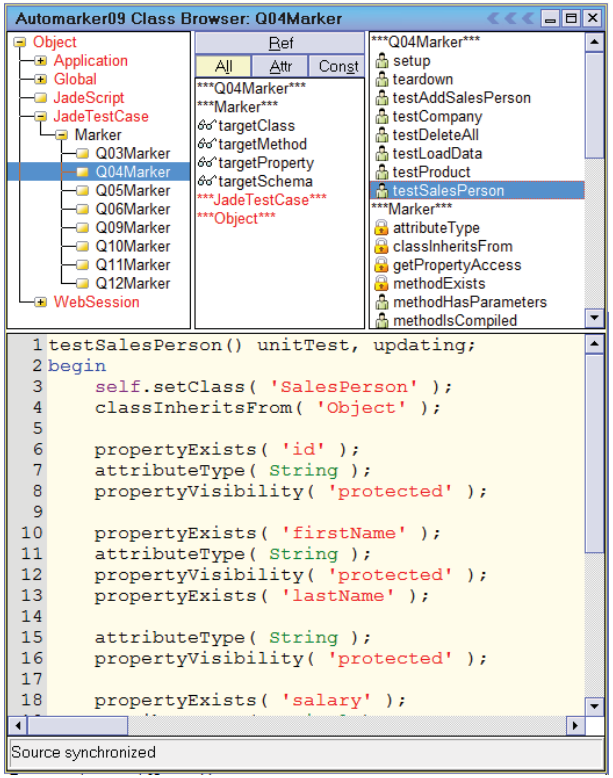

Figure 3

If a student has succeeded in the required task the tests confirm this (See figure 4 below). Students come to like the green progress bar and the 'passed' message.

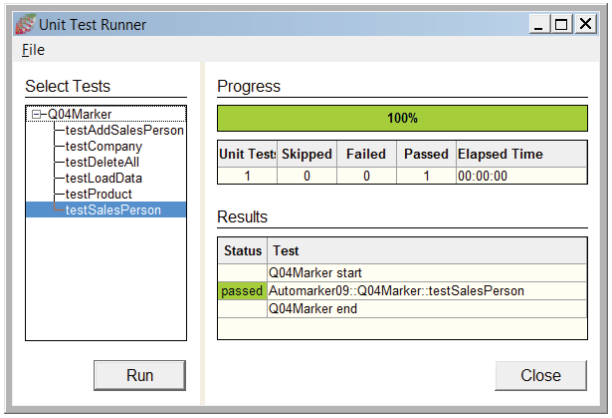

Figure 4

The advantage of using unit tests in this basic data construction task is that students get a lot of feedback as to what aspects of the program are to important. It is very time consuming for a teacher to check the 75 items covered by assertions in the unit test for question 4. The feedback provided can be either a very generic warning (i.e. 'Encapsulation has been broken') or a very specific instruction of what needs to be done next (i.e. 'make sure the set method of the Product class assigns the newDescription parameter to the classes description property). Figure 5 shows a typical problem report. The attribute of the SalesPerson class have been changed to public visibility to avoid having to write a set method

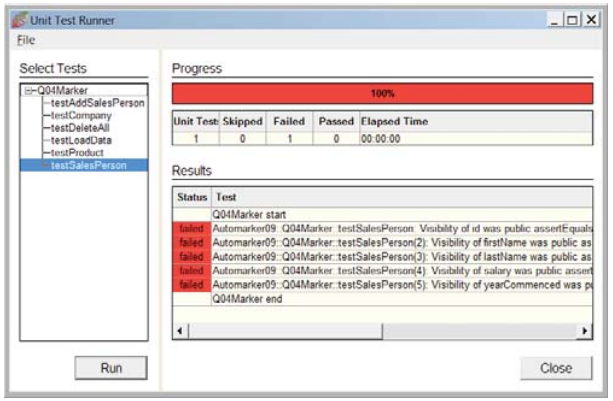

Figure 5

In this introduction to testing, students are not required to be aware of the internals of the unit tests, but do have to use unit tests to get immediate feedback of the quality and functionality of their code. The immediacy of the feedback starts their 'test addiction'.

2. Automatically Assessing Complex Algorithm Creation

Once students are aware that automated tests can mark their work and provide a substitute for inspection of code by a teacher a next step is to expose students to the internals of how tests are written and get students writing their own tests.

One way to do this is to require students to develop a complex algorithm with many different possible 'correct' implementations. Provide a complex system and direct students to one place where a crucial extension of code is required. For example in a vending machine (see Figure 6) the task is to add a single

method to dispense the change for a transaction. This is a relatively complex system with many collaborating objects needing to be taken into account. Careful testing is needed to avoid unexpected consequences.

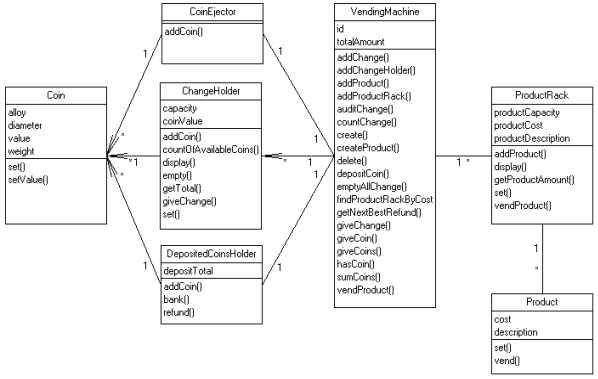

Figure 6.

There are many possible correct ways of coding the required change giving algorithm. A good solution requires the calcChange method to check for possible error conditions such as the unavailability of required coins. For example what if 50 cents of change is required and there are no 50 cent coins left in the machine. Two 20 cent coins are available but no 10 cent coins. Robust code will detect that it is impossible to give the correct change, and void the transaction returning the initial deposit and activating the out-of-change sign.

The unit tests are assessing the range of situations which the student's algorithm can deal with eg what happens when the system runs out of a particular denomination of coin? What happens if there is no correct change which can be given?

The unit tests for such a system are non-trivial. The vending machine system has 591 lines of code and the associated unit tests have 2824 lines of code. The unit testing framework provides a before and after dump of the state of every object in the system and a suite of 30 different acceptance test scenarios of varying complexity.

An extension task requires the student to inspect and extend the unit tests to cope with altered requirements. Is their change giving algorithm sufficiently robust that it can cope with US rather than NZ currency? To do this students need to modify existing tests creating new test setup methods which load a different currency into the system and change assertions to reflect different expected change. Testing is being learnt by modifying existing examples.

Conclusions

This work has documented how a unit testing framework can be used to automatically mark two different types of business computing programming assignments. It establishes that it is possible.

Observing students using these tests it is evident that novice programmers are unaware of testing strategies and heavily reliant on rote learning of rules. Using and reading the unit tests showcases different testing techniques and helps novices to analyze what testing is relevant in the context in which they are working. The next step in extending this research is to investigate what feedback is most effective in promoting this learning.

References

Beck, K. (2002) *Test Driven Development: By Example*, Addison-Wesley, Boston, MA.

Brusilovsky, P and Higgins, C. (2006) *Preface to the Special Issue on Automated Assessment of Programming Assignments.* ACM Journal of Educational Resources in Computing, Vol. 5, No. 3, September 2005. Article 1.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and Stal. M. (1996) *Pattern-Oriented Software Architecture: A System Of Patterns*. West Sussex, England: John Wiley & Sons Ltd.

Douce, C., Livingstone, D., & Rrwell, J. (2006) *Automatic test-Based Assessment of Programming: A Review*. ACM Journal of Educational Resources in Computing, Vol. 5, No. 3, September 2005. Article 4.

Create, read, update and delete in *Wikipedia*. Retrieved from 15 may 2010 from http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

Edwards, S.H. (2003) *Rethinking Computer Science Education from a test-first perspective*. In the 2003 Proceedings of OOPSLA '03, October 26-30, 2003, Anaheim, California, USA

Edwards, S.H. (2004) *Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action*. In the 2004 Proceedings of SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA

Helmick, M.T. (2007) *Interface based Programming Assignments and Automatic Grading of Java Programs*. In the 2007 Proceedings of ITiCSE'07, June 25–27, 2007, Dundee, Scotland, United Kingdom

Higgins, C., Tsintsifas, A., and Symeonidis, P. (2002). *CourseMaster marking programs and diagrams*. In Proceedings of the Dealing with Plagiarism in ICS Education Conference (Warwick, April 11–12)

Lance, M. (2005) *Teaching with a Unit Testing Framework*. In the 2005 Proceedings of NACCQ2005, July 3–7, 2005.

Tremblay,G and Labonté, E (2003). *Semi-automatic marking of Java programs using JUnit*. In International Conference on Education and Information Systems: Technologies andApplications (EISTA '03), pages 42–47, Orlando, FL, July 2003. International Institute of Informatics and Systemics.

Tremblay, G., Guérin, F. and Pons, A. (2005) *A Generic and Extensible Tool for Marking Programmin*g Assignments. In IASTED Intl Conf. on Educ. and Tech., pp 55-60.