Teaching Software Quality Assurance by Encouraging Student Contributions to an Open Source Web-based System for the Assessment of Programming Assignments

Olly Gotel
Pace University
Seidenberg School of Computer Science and Information Systems
New York, NY, USA
ogotel@pace.edu

Christelle Scharff
Pace University
Seidenberg School of Computer Science and Information Systems
New York, NY, USA
cscharff@pace.edu

Andrew Wildenberg
Cornell College
Computer Science Department
Mount Vernon
IA, USA
awildenberg@cornellcollege.edu

ABSTRACT

This paper presents a novel and innovative pedagogical approach for teaching software quality assurance in the undergraduate computer science curriculum. The approach is based on students contributing programming problems to an open source web-based system that is used for student practice and instructor assessment of assignments. WeBWorK, and some of the latest web-based systems, use a mechanism based on unit testing to account for variation in the way in which the same problem can be answered in an accurate manner, making such systems highly appealing for education. Tackling open-ended programming problems within WeBWorK therefore requires students to write a code fragment that is then checked for semantic correctness. Given that WeBWorK is open source, the teaching approach that we have evolved revolves around students creating their own problems for other students to practice with. This requires students to construct comprehensive unit tests that can assure both the usability and accuracy of their work prior to deployment. The paper describes this approach, gives examples of student work, presents findings from the experience of using the approach in the classroom, and discusses broader lessons and reasons for integrating software quality assurance practices into the computer science curriculum.

Categories and Subject Descriptors

K.3.2 **[Computer and Information Science Education]**: Computer Science Education, Curriculum.

General Terms

Verification.

Keywords

Automated Assessment Systems, Java, JUnit, Open Source, Peer Review, Requirements, Programming, Software Quality Assurance, Unit Testing, WeBWorK.

1. INTRODUCTION

Programming tends to be the primary skill that is emphasized throughout the undergraduate computer science curriculum. In general, there are no dedicated courses on requirements, testing and software quality assurance (SQA) in its wider sense in the undergraduate curriculum, and teaching these closely inter-related software engineering topics is often relegated to an already overwhelming senior capstone software engineering course. While such capstone courses require students to participate in time-consuming team projects, integrating and demonstrating the skills they have acquired to date, it is difficult to simultaneously do justice to the important discipline of SQA.

As a consequence, the undergraduate computer science curriculum does not align well with the reality and needs of professional software development where, in a typical large project, only 16-30% of the software development effort directly relates to implementation (programming) [7]. Moreover, a recent update to the CHAOS report finds that 15% of software development projects fail outright and 51% are seriously challenged (i.e., are late, over budget and/or missing functionality), only 34% being considered successful [8]. The failure of such projects is mainly due to poor requirements-related activities, particularly inadequate user involvement, alongside insufficient validation and verification (i.e., testing-related activities). A discipline for assuring quality, introduced early in a future software engineer's education, is one way to address this.

Test-driven development is an established industrial practice that emphasizes the importance of formulating requirements in a testable manner as a baseline for quality-oriented development. This approach to development can be undertaken from the earliest days in which students specify problems to be solved by program code. Where software quality is viewed as conformance to requirements, focusing on test-driven development across the curriculum is potentially an effective way for instructors to address the lack of attention to SQA topics.

Numerous web-based systems for the assessment of programming assignments (mostly commercial ones) have emerged in the last few years and have been employed to support the teaching and learning of programming fundamentals. In addition to their capacity to present and grade multiple-choice quizzes and short answer questions, they offer the possibility to check the correctness of program fragments. The pedagogical advantage of open source systems of this kind is that they permit the contribution of problems and the creation of problem libraries that can be accessed, re-used and augmented by instructors and students at different institutions worldwide.

This paper describes educational research conducted to exploit the underlying mechanism of an open source web-based assessment

ITiCSE'08, June 30–July 2, 2008, Madrid, Spain.

system to harness students' interest in creating programming problems for other students to use. Through the process of developing, deploying and gaining feedback on the use of problems, SQA takes place at several levels. In creating quality assured programming problems, students no longer focus purely on writing code, but learn about and practice the activity of test-driven development. Since they are contributing to a library of problems that they will observe being used by others, real application and end-use of their work makes the activity engaging.

The paper is organized as follows. In Section 2, we provide a brief summary of SQA activities and motivate the need for more attention to this discipline in computer science education. Web-based programming assessment systems are outlined in Section 3, to provide context for the model of teaching and learning we describe in Section 4 and exemplify in Section 5. Our findings from this experience are given in Section 6 and the paper ends with broader lessons for development and uptake in Section 7.

2. SOFTWARE QUALITY ASSURANCE

In the context of software development, quality is typically measured in terms of the specified requirements that are satisfied by the software system. Software quality assurance (SQA) is, at a fundamental level, all those process-related activities that are undertaken in the pursuit of achieving software quality. It involves careful attention to the specification of the requirements to be satisfied, to ensure they accurately capture what is wanted or intended (validation), and the formulation of the test cases that can be used to demonstrate their eventual satisfaction in code (verification). Requirements, design and code reviews are hence critical interactive activities undertaken by participants (both internal and external) to identify any problems early on. Requirements and defect tracking are also essential activities to undertake if changes and raised issues are to be handled smoothly.

Given the CHAOS report figures quoted earlier, it is apparent that the majority of software projects suffer from quality problems and that improving the quality of software development is a major concern for the US economy. With a focus on programming in computer science education, and a perception of centrality that can be difficult to dispel after having spent formative freshman years solely writing code, limited attention is paid to those disciplines that are critical to establishing quality. Indeed, the first real mention of software quality often does not arise until the capstone software engineering course, but it is a topic that is generally more thoroughly covered in specialist graduate courses.

We suggest that a competence for quality be emphasized in the initial stages of an undergraduate computer science degree and sustained throughout. Simple SQA techniques like peer review can be introduced early on and evolved into the professional concept of more formal inspection over time. Rather than always providing students with problems to write programs for, getting students to formulate and express their own problems gives them early exposure to writing requirements, since requirements are merely the expression of a problem to be solved by some system (albeit usually at a higher level of complexity). Furthermore, recent tooling environments like Eclipse (*http://www.eclipse.org*) make it straightforward to associate automated unit testing with single classes, giving little reason for students not to write at least one test case to demonstrate their code does what is expected of it.

3. WEB-BASED PROGRAMMING ASSESSMENT SYTEMS

There are a growing number of web-based systems that are designed to support the practice and assessment of programming tasks [3]. Being the result of commercial or academic enterprises, most of these systems tend to have a per-seat cost for student use and the problems available are created, administered and managed by the organization behind the venture. A small number of these systems relinquish some of this control to the instructor and encourage the contribution of an instructor's own problems. A number of these systems are characterized in [6].

WeBWorK (*http://webwork.math.rochester.edu*) [4] is a free and open source web-based formative assessment system used to generate, deliver, and (automatically) grade homework problems and distribute their solutions. The University of Rochester has led WeBWorK's development since 1999 and over sixty institutions worldwide currently use it to teach mathematics (calculus, pre-calculus, algebra, applied, finite and discrete mathematics). WeBWorK offers traditional multiple-choice and short answer questions, but its real power and flexibility comes from its underlying engine that can understand mathematical formulæ and support randomization and parameterization.

Since 2004, the authors of this paper have been collaborating to adapt and extend WeBWorK for use in the core courses of the computer science curriculum, in particular to deliver programming assignments in Java and Python (*http://csis.pace.edu/~scharff/webwork/webwork.htm*) [1, 2, 5, 6]. A major part of this initiative has been to develop an extension called WeBWorK-JAG (where JAG stands for Java Auto Grader) to automatically collect and grade free-form program fragments written in Java in real-time, thereby extending the types of problem supported by WeBWorK to make it a better fit for the demands of programming practice [6]. Based upon JUnit (*http://www.junit.org*), WeBWorK-JAG permits students to exercise their code against a set of test cases to determine correctness and provide feedback hints.

4. TEACHING MODEL

This section describes the teaching model that was designed to emphasize SQA based upon students contributing programming problems to WeBWorK libraries. It outlines the process undertaken by the student, with details about the steps of problem formulation, the design of unit tests to check the semantic correctness of the solution and the underpinning peer review process used to ensure the quality of both steps.

4.1 Process

The process of contributing a programming problem involves the following steps:

1. Formulation of the problem (Requirements);
2. Peer review of the problem formulation (Requirements Inspection and Validation -- SQA);
3. Design of unit tests (Requirements Refinement and Testing);
4. Peer review of unit tests (Test Case Inspection and Verification -- SQA);
5. Integration of the problem with its test cases into the web-based system (Deployment); and
6. Testing of the problem and feedback by users (User Acceptance Testing -- SQA).

4.2 Formulation of Programming Problems

Programming problems are commonly aimed at asking students to write a method. To homogenize the way in which problems are presented to students, we developed a template for the formulation of problems. The template is sufficiently general that it can be used to write problems in different programming languages. The template for a Java problem, its instantiation for the *factorial* method and a solution to the problem are provided in Tables 1, 2 and 3. This example will be used for illustration throughout Section 4. The use of the template privileges thinking about the requirements in terms of normal execution and the exceptions generated by the method, and facilitates unit test design. It also enforces good coding habits to think about all possible inputs.

Table 1. Template for problem formulation.

Description. A short description of the method to be written.
Method name. The name of the method.
Method signature description. A description of the method signature in terms of:

- Modifier, i.e. either static, public, protected, private or package;
- Type of the method, i.e. either static or instance;
- Number and type of parameters; and
- Return type.

Exceptions. A description of the exceptions to be thrown along with the cases in which thrown.
Code. Code provided to support writing the method.
Notes. Particular restrictions concerning the method to be written.

Table 2. Instantiation of the template for the *factorial* method.

Write a method that computes the factorial of a given number.
The method will be called *factorial* and must:

- Be *public* and *static*;
- Take an *int* as a parameter;
- Return the factorial of that *int* as an *int*; and
- Throws an *IllegalArgumentException* for a negative input or an input that would not return a Java *int.*

Table 3. Example of the expected *factorial* method.

```
public class Factorial {
public static int factorial(int n) {
if (n <= 12 && n > 0) {
   return n * factorial(n - 1);
} else if (n == 0) {
   return 1;
}
throw new IllegalArgumentException("Argument " + n + " not in
range");}}
```

4.3 Design of Unit Tests

Designing the unit tests for a method involves writing tests for:

1. The signature of the method;
2. The semantic correctness of the method for its normal flow, describing the execution of the method under standard and expected conditions; and
3. The anticipated error conditions and exceptions that could occur and be generated during the execution of the method.

Moreover, the unit tests must be written to take into account any unexpected failure that could happen during the execution of the method and generate appropriate error messages. These messages are particularly important for the learning experience of the students who will attempt the programming problems within WeBWorK; they provide students with accurate feedback concerning the semantic correctness of the method they submit. The level of granularity of the feedback can vary. Feedback can cover the number of pass/fail tests or include the exact inputs from specific equivalence classes that generated the failure.

To illustrate the process, Table 4 shows how JUnit and the Reflection API are used to check the signature of the *factorial* method. Tables 5 and 6 demonstrate JUnit tests for *factorial* in the cases of normal and exception-generating executions of the method respectively. In this example, the level of granularity of the feedback is such that the input that generates a failure is provided in the error messages. Each tested input is part of different JUnit test methods to provide for a high granularity of feedback to students concerning the number of tests that pass/fail – here each test corresponds to one input of factorial.

Table 4. Unit tests for the *factorial* method signature.

```
public class FactorialTest extends TestCase {

boolean exists = false; // factorial exists
boolean returnType = false; // factorial return type
boolean paramnbType = false; // factorial parameters and types
boolean isStatic = false; // factorial static
Method factorial;

public void setUp() {
   Method[] methods = Factorial.class.getMethods();
   for (int i = 0; i < methods.length; i++) {
      if (methods[i].getName().equals("factorial")) {
         exists = true;
         factorial = methods[i];
         isStatic = Modifier.isStatic(factorial.getModifiers());
         returnType = factorial.getReturnType().
         getName().equals("int"));
         Class[] params = factorial.getParameterTypes();
         paramnbType = params.length == 1 &&
         params[0].getName().equals("int"));
}}}

// Method that checks the signature of factorial
public void testMethodSignature() {
   Assert.assertTrue("The method should be called factorial!",
   exists);
   Assert.assertTrue("The method has to be static!", isStatic);
   Assert.assertTrue("The method should return an 'int'!",
   returnType);
   Assert.assertTrue("The method should take an 'int' as
   parameter!", paramType);
}...}
```

Table 5. Unit tests for *factorial(n)* where n >= 0 and n <= 12.

```
public final void testFactorial3() {
try {
   assertEquals(6, Factorial.factorial(3));
} catch (Exception e) {
   fail("Test failed for factorial(3)!");}}
```

Table 6. Unit tests for *factorial(n)* where n < 0.

```
public final void testFactorialMinus4() {
try {
   Factorial.factorial(-4);
   fail();
} catch (Exception e) {
   if (e instanceof IllegalArgumentException)
      assertTrue(true);
   else
      fail("Test failed for factorial(-4)!");}}
```

4.4 Peer Review and Quality Assurance

The quality assurance of the problem contribution process is achieved by assessing the quality of the produced artifacts (e.g., problem formulation and unit tests) using peer review and user acceptance testing. The students formulating and writing the unit tests for a problem are not the same as the ones involved in peer review. Moreover, user acceptance testing is also undertaken by a separate group of students to facilitate objectivity, as well as to reflect industry practices. Students only write the code of the programming problem they have proposed to integrate in the assessment system that tests their unit tests; they concentrate on the writing of requirements and unit tests, leveraging their experience of test-driven development and its potential for SQA.

5. PRACTICAL APPLICATION

The teaching model described in the previous section was implemented in a CS2 programming course, Fundamental Computer Science with Java II, taken by nineteen students. This course covers object-oriented data structures with Java (e.g., linked lists, stacks, queues, trees and graphs). The topics also include memory

management, Java 5 and 6 features, and the study of APIs such as the Reflection API and JUnit.

At the beginning of the semester, students reviewed the topics taught in the CS1 programming course (e.g., loops, conditionals, methods, arrays, recursion, and searching and sorting in arrays of primitives) by completing two WeBWorK assignments – one composed of multiple-choice questions and the other composed of programming problems. This familiarized them with WeBWorK and they gained an understanding of the kind of programming problems they would be expected to contribute.

The students contributed nine programming problems to WeBWorK focusing on topics of CS1 and CS2. They formulated the programming problems, designed the JUnit tests (within Eclipse) and peer reviewed them. Due to lack of time, the problems were integrated into WeBWorK by the instructors. The problems they proposed included the sum of even numbers, checking whether a number is prime, specific cases of the binary search and sorting problems, testing if two arrays of integers are mirrors, two problems on linked lists (adding an element at the end or front) and one problem on counting the number of methods defined in a particular Java class (requiring the use of the Reflection API). Figure 1 presents a problem contributed by the students, a method *sortIntArray* to *s*ort an array of *ints*, illustrating the feedback provided for an incorrect solution submission.

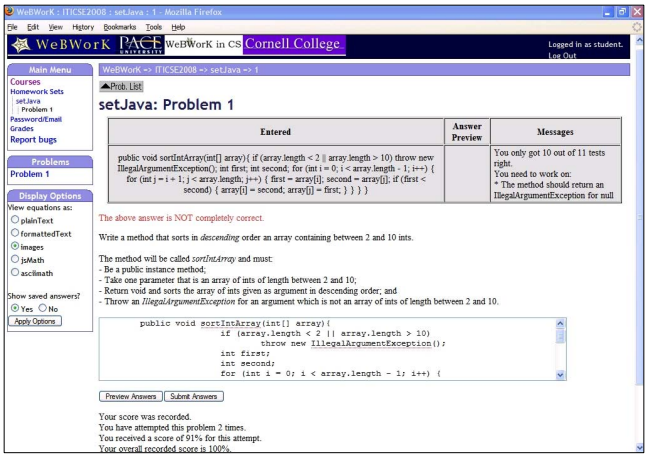

Figure 1. Example for *sortIntArray* – Wrong Answer.

6. RESULTS

Problem users. The difficulty of the proposed programming problems was appropriate to the level of the students. They focused on questions they considered feasible but nevertheless challenging. Interestingly, students mentioned that they always had the user (i.e., CS2 students) in mind during the process of specifying the programming problems.

Problem topics. Students found it difficult to come up with topics for their problems. Most of the programming problems (six out of nine) referred to topics introduced in their CS1 course. The most innovative problem was one about counting the number of methods defined in a particular Java class, whose solution requires the use of the Reflection API. There were two problems on linked lists, topics newly introduced in the CS2 course. Moreover, the problems focused more on the procedural constructs than on the object-oriented aspect of Java, probably due to the fact that they were not taught initially using an object-first approach.

Problem formulation. Students found it difficult to formulate the requirements of their programming problems in a precise way. They mainly concentrated on the normal course of execution of the method to be written, and had difficulties expressing the error and exception cases. They also formulated unfounded assumptions and failed to provide specific setup information required to answer the problem, leading to difficulties in evaluating the scope of the problem by others. For example, for programming problems related to linked lists, it was important to provide the skeleton of the linked list class as it conveys information about the chosen implementation. Students needed to think simultaneously about the problem formulation and the unit tests they would write to check semantic correctness. Some of the constraints expressed in the problems by the students were not appropriate and went beyond what can be tested with unit tests. For example, some students specified that the solution to their programming problem mandated the use of a particular algorithm (e.g., a recursive algorithm to implement sorting).

Writing JUnit tests. Writing JUnit tests was supported by use of Eclipse. The quality of the feedback to be provided to the students when dealing with a programming problem is related to the quality of the designed tests and the granularity of the messages for test failure (if any). Poor results ensued because students overlooked the user at that phase of the problem contribution process. However, when asked for improvement suggestions for WeBWorK in general, students were consistently concerned about getting more specific and detailed feedback about test failure, including the number of tests that failed and the exact data on which they failed. The main reason for students not providing such messages was due to technical difficulties; students were not yet very familiar with exception handling.

Testing knowledge. Students were also not familiar with many testing techniques. JUnit tests for checking the signature of the method to be defined were well written, but JUnit tests that concentrated on correctness had several flaws. Even though the topics of blackbox/whitebox testing, test coverage and equivalence classes were covered in class, students did not get enough time to assimilate these techniques and implement them appropriately. Students tended to use blackbox testing techniques only, missing tests related to the structure of the method. For example, in the binary search problem, no tests were proposed that related to finding an element exactly in the middle of an array of *ints*. Students sometimes wrote a large number of tests, but they were only variations of the same underlying test case; students were not able to identify equivalence classes. Students also often forgot the boundary cases in their tests; it was very rare to see the case of the *null* object considered for a method taking an object as a parameter. The problem about counting the number of methods in a class was not tested for a class that inherits from another class, and students were not able to correctly write the JUnit tests for *void* methods employing objects as parameters due to difficulties in understanding Java parameter passing.

Peer review and SQA. Peer reviewing the problem formulation and unit tests allowed for an improvement in a problem before integrating it into WeBWorK. Broken tests (e.g., tests with errors) were caught during the process but students really needed more peer review sessions to perfect their work. Due to time constraints, the instructor integrated the questions in WeBWorK after extensive reworking and made them accessible to all the students. Some of the questions were reformulated and the JUnit tests were almost

completely rewritten due to the problems above. The user acceptance testing occurred by having students submit bugs to the instructors, but not to the originator of the question.

Overall experience. At the end of the semester, students were asked to reflect on the learning experience of this pilot program. Students stated that this approach permitted them to gain experience with testing from the outset and to learn all the steps they need to go through to assure the quality of their contribution. Students thought about the tests and the code in parallel, but the code was not the main focus, the objective we wanted to reinforce with this assignment. Students liked the experience as it provided them with a hands-on way to contribute to open-source software. Specifically, they wanted to be certain that future students would be able to access their problems which they had taken pride in.

7. LESSONS

Teaching model adaptation. Our teaching model for the contribution of problems to the open source web-based assessment system described in Section 3 is very general and does not focus on WeBWorK exclusively. Instructors could adapt the model and ask their students to contribute problems and JUnit tests independently from any system. The problems could then be used as programming assignments. The JUnit tests could help instructors automatically grade the assignments and provide feedback. The interest of using WeBWorK is that it automatically prints the number of pass/fail tests and provides feedback via the messages that are provided in the JUnit tests that fail.

Programming skills. Students not only improved their problem formulation and testing skills in this pilot, but they also improved their coding. Students were exposed to good programming practices and had to think about all the possible arguments that could be input to a method. For example, the factorial method regularly appears erroneously in textbooks, not accounting for negative arguments or arguments that would generate a factorial that is not an *int*. We recommend our teaching model in that it develops programming, testing and wider SQA skills in students.

Practical introduction. There is no class on software testing and SQA per se in the undergraduate computer science curriculum. We believe that these software engineering activities should have a more prominent place in the curriculum, maybe not by having a dedicated course, but by having modules on these topics in core courses. Our approach of students contributing problems to an open source web-based assessment system is one practical way to do this. JUnit can be introduced along with methods in CS1. Test coverage, equivalence classes and regression testing could also be covered at that time in an informal manner. In CS2, students can build on their JUnit knowledge to consider cases where exceptions are generated. More advanced testing and SQA techniques could then be introduced in later courses.

Building problem libraries. Additional motivation behind the introduction of the teaching model described in Section 4 was to have students help in the development of a library of WeBWorK problems that could be used in our teaching and also by instructors at other institutions. The reality is that the problems contributed by the students could not be integrated directly into WeBWorK without major rework. There is a need to dedicate more time for peer review with the students during class time to guide them in catching problems. Students also need to see more examples of problem specifications and well-written JUnit code (i.e., simple JUnit tests using exceptions). Formal peer review guidelines will be established in the future with lists of known weaknesses in JUnit tests (e.g., coverage and feedback messages). User acceptance testing could also be formalized to have students submit bugs through an online system, exposing them to bug tracking tools. In the next implementation of this model, more time will be dedicated and the exercise may be repeated to have students develop first problems on the topics they learned in CS1 and then in CS2. We would also like students to see the complete process of contributing a problem, which includes the integration of their problems into WeBWorK with user acceptance testing.

8. CONCLUSIONS

This paper has described an educational study carried out to explore how the functionality of an extensible web-based assessment system can be exploited to enable computer science students to learn about and practice a subset of critical SQA activities early on in the curriculum. It illustrates a model of teaching and learning that revolves around students creating their own problems and associated test cases for judging solution correctness, then undergoing an extensive round of peer review prior to acceptance and use. This model was found constructive in getting a pilot set of CS2 students to understand the role of precise problem formulation (i.e., requirements specification) and test-first development as important counterparts to achieving better quality code and software systems. The model will be developed to address shortcomings and re-implemented in the near future.

9. ACKNOWLEDGMENTS

This project is supported by NSF CCLI AI grants #0511385 and #0511391. We are grateful to Jacqueline Baldwin, Nathan Baur and Sophal Chiv for their help with WeBWorK.

10. REFERENCES

[1] J. Baldwin, E. Crupi, and T. Estrellado. WeBWorK for programming fundamentals. In *Proc. 11th Conference on Innovation and Technology in Computer Science Education*, Page 361, New York, NY, USA, 2006. ACM Press. (Poster).

[2] J. Baldwin, E. Crupi, T. Estrellado, O. Gotel, R. Kline, C. Scharff, and A. Wildenberg. Examples of WeBWorK programming assignments. In *Proc. 37th SIGCSE Technical Symposium on Computer Science Education*, Houston, Texas, USA, 2006. (Poster).

[3] P. Brusilovsky and C. Higgins. Preface to the special issue on automated assessment of programming assignments. *Journal of Educational Resources in Computing*, 5(3), 2005.

[4] M. Gage, A. Pizer, and V. Roth. WeBWorK: An internet-based system for generating and delivering homework problems. In *Joint Meeting of the American Mathematical Society and the Mathematical Association of America*, 2001.

[5] O. Gotel and C. Scharff. Adapting an open-source web-based assessment system for the automated assessment of programming problems. In *IASTED Web-based Education Conference*, Chamonix, France, March 2007.

[6] O. Gotel, C. Scharff and A. Wildenberg. Extending and contributing to an open-source web-based assessment system for the automated assessment of programming problems. In Proc. *ACM Conference on Principles and Practices of Programming In Java*, Lisboa, Portugal, September 2007.

[7] C. Jones. Software Cost Estimating Methods for Large Projects. *CrossTalk: The Journal of Defense Software Engineering*, April 2005.

[8] The Standish Group. *CHAOS Chronicles III*. 2003. (http://www.standishgroup.com/chaos/toc.php).