

Automated Unit Testing with **JUnit**

A tool for
unit testing, design, lab grading,
student motivation, reasoning, and stress reduction

by Dave Wittry

The use of JUnit to aid in the design of a class and the creation of error-free code has been well discussed in other works. While this paper discusses design and error-free code to some degree, its main focus is on the most useful and motivational aspects of JUnit for the student and teacher in a high school computer science classroom. The approach is, therefore, practical. The following pages simply detail the understanding and advice of one teacher after a year's use of JUnit in the classroom. Accompanying the discussion of JUnit below is an explanation of a complementary and free lab grading tool, Jamtester.

Table of Contents

JUnit from the Teacher's Perspective

What is JUnit?

How does JUnit work?

How and when should I introduce and use JUnit in the classroom?

How should I use JUnit to help students test their classes?

How can I possibly fit one more topic into my course?

How can JUnit aid in regression testing?

Why are *ad hoc* testing techniques flawed?

How does JUnit help students design cohesive methods?

How can JUnit help me give my students more feedback?

Does all of this mean that I will save time on grading labs?

JUnit from the Student's Perspective

Can students create, edit, and run JUnit tests easily and independently of any IDE?

Can use of JUnit alleviate students' stress levels?

How does the use of JUnit increase student-student cooperation?

JUnit from the Teacher's Perspective

What is JUnit?

JUnit (www.junit.org) is open-source software, an API framework, used to automate unit and regression testing. In short, it is software-exercising software. Martin Fowler says of it, "Never in the field of software development was so much owed by so many to so few lines of code."

How does JUnit work?

Each Java class that you write will correspond to a JUnit Java class that will test the methods of your class.

Here's an example:

```
// your students are trying to write/test this class
public class Student {
    private int exam1, exam2;

    public Student(int ex1, int ex2) {
        exam1 = ex1;
        exam2 = ex2;
    }

    public int bestScore() {
        if (exam1 > exam2) return exam1;
        return exam2;
    }

    public int worstScore() {
        return exam1; // intentional "failure".
                    // student needs to fix logic
    }
}
```

```
// this is the test class
import junit.framework.*;

public class StudentTest extends TestCase {

    // here is one unit test method
    public void testBestScore() {
        Student stud1 = new Student(90, 85);
        int answer = stud1.bestScore();
        assertEquals(answer, 90); // compare to known answer
    }

    // here is one unit test method
    public void testWorstScore() {
        Student stud1 = new Student(90, 85);
        int answer = stud1.worstScore();
        assertEquals(answer, 85); // compare to known answer
    }

    // more unit tests not shown...

    // will generate results of tests (text output)
    public static void main(String[] args) {
        TestSuite suite = new TestSuite(StudentTest.class);
        junit.textui.TestRunner.run(suite);
    }
}
```

this tests this

Classes and methods from the JUnit framework are in blue. The above generates the following text output: "Tests run: 2, Failures: 1, Errors: 0," however, with a bit more detailed information.

As far as the `testBestScore()` method above,

- an object to be tested was instantiated,
- a method on the object was invoked,
- and the returned answer was compared with the known answer by using JUnit's `assertXXX` most common method, `assertEquals`.

That's it!

If the test fails, the student knows to edit her code and rerun the test suite.

How and when should I introduce and use JUnit in the classroom?

At the beginning of the school year, you routinely spell out the methods to write for each Java class; this is the perfect time to give students a JUnit test file for their class, complete with a full suite of test cases.

Students will benefit by this early introduction. They will not need to devise the test cases themselves, nor will they need to worry about writing a main program to test their classes. Instead, they will be able to focus on writing their classes, and they will naturally over time gain a comprehensive understanding of JUnit, including arranging syntax and developing a complete set of test cases. They will also know *when* they have finished the lab, that is, when all tests are successful.

You, as a teacher, will also benefit. Since you do not begin the year by teaching “main” with all of its syntax and keywords, you can focus immediately on teaching objects and inheritance. This way, you can, from the start, model and communicate good design methods through the use of test cases by guiding students through a systematic way of testing their code. Moreover, you will feel confident that your students will be writing correct method headings, and your students will feel confident as they earn perfect scores on their labs.

How should I use JUnit to help students test their classes?

Over time you will give your students JUnit test files which are less and less complete. In other words, the test suites will intentionally be missing some of the test cases. This way, the students will devise and write test cases on their own, and they will, at the same time, develop the analytical skills necessary to recognize that they have produced a complete set of test cases.

Eventually, you will stop giving them JUnit test files altogether. At this point, an amazing thing will happen: the process will become completely student-centered. Without being prompted, students will create their own JUnit test files. (See also “JUnit from the Student's Perspective” below to learn about their motivation.) In this manner, you will have systematically and gradually taught them to analyze, design, test, and function independently of you, the teacher.

How can I *possibly* fit one more topic into my course?

I wondered this, too, but the answer is simple: the students learn JUnit on their own. Since it is easy to teach and can be introduced slowly, it consumes very little instructional time. The payoff is extraordinarily high in terms of student understanding and performance.

How can JUnit aid in regression testing?

JUnit *is* regression testing! But it is regression testing that is efficient and precise. It can perform and repeat tests at any time.

Think of it this way: how many times does a student, or professional for that matter, test a method until it *seems* to work, and then destabilize the method already tested by making changes to code?

Often. *Too* often, in fact.

JUnit is a tool that enables *all* tests to be run at *any* time. Thus, students will know immediately and at will that a tested method has begun to behave badly.

Why are *ad hoc* testing techniques flawed?

When using *ad hoc* testing techniques such as embedded print statements and customized main programs, students are not naturally inclined to rerun or, more importantly, recreate tests on their own. Who would? Such techniques are inefficient, tedious, time-consuming, and error-prone.

As you know, students wait until they have *completely* finished the creation process before they test their projects even *once*. This approach illustrates flawed reasoning and compounds the problems that they will encounter as they attempt to remove errors in their code. As you also know, coding and testing should not be treated as disparate and independent functions. They are integral functions, complementary to one another. JUnit reconnects coding and testing in a manner that causes the students to embrace logical thinking and good coding practices from the outset.

How does JUnit help students design cohesive methods?

As they master JUnit testing techniques, students will also be preparing to design cohesive methods, that is, methods that perform a single well-defined task. Importantly, cohesive methods are hallmarks of a well-designed class.

Using JUnit early in the year, you will lecture about the proper design of cohesive methods, among other topics, and you will reinforce your lectures by model. As the year progresses, you will wean your students from the complete test suites, causing them to test and design more and more on their own and undoubtedly applying the good design skills that they have learned through your modeling. This way, students will be more likely to recognize a method that needs refactoring, that is, broken down into several, more cohesive methods. Through these techniques, you will communicate your goal: that students will become good testers and good designers.

How can JUnit help me give my students more feedback?

When you give your students a complete test suite early in the year, you will essentially be giving them the ability to receive your feedback continuously but vicariously throughout the year. The feedback you give, that is, that JUnit gives, is immediate, specific, and continual. Students will run their tests over and over, that is, each time they change code. Your own feedback could not be any more specific, useful, immediate, convenient, or time-saving.

Does all of this mean that I will save time on grading labs?

In one word, YES.

In another word, Jamtester.

Jamtester, **JUnit Auto-Matic Tester**, is a free software tool which will enable you to grade *all* of your students' labs *simultaneously*.

You will no longer need to compile and run each student lab individually.

You will no longer need to stop and write down results.

You will no longer need to interact with a program by typing in user input to grade it.

Jamtester works very simply: you give it a JUnit test file, student directories containing the .java file(s) to be graded, and the .jar/.class files needed to make the project compile. Hit "go" and sit back as it grades the labs.

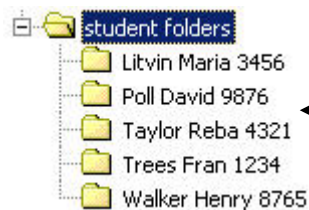
The tool allows you to save results to a .csv file, a format readily imported by Excel and most grading programs. You can also edit the student's source directly in the tool, then rerun the tests and have the new results right there in a table.

sample output
(see www.jamtester.com for an animated demo)

Field 1	Field 2	Field 3	Success	Percentage	testBestScore	testWorstScore
Litvin	Maria	3456	1 of 2	50.0%	F	
Poll	David	9876	2 of 2	100.0%		
Taylor	Reba	4321	0 of 0	0.0%	C	C
Trees	Fran	1234	1 of 2	50.0%		E
Walker	Henry	8765	1 of 2	50.0%		F

```
junit.framework.AssertionFailedError: expected:<90> but was:<85>
    at junit.framework.Assert.fail(Assert.java:47)
    at junit.framework.Assert.failNotEquals(Assert.java:282)
    at junit.framework.Assert.assertEquals(Assert.java:64)
    at junit.framework.Assert.assertEquals(Assert.java:201)
    at junit.framework.Assert.assertEquals(Assert.java:207)
```

this is the
code
example
from above



A folder of student folders (each folder in this example contains a Student.java file)

JUnit from the Student's Perspective

Can students create, edit, and run JUnit tests easily and independently of any IDE?

Yes. Jamtester is not only a teacher tool, it is also a student tool. Regardless of the IDE you choose for your classroom, students will be able to test their source files using JUnit files. Students can seamlessly go between their IDE and the student tool, testing as often as they like. If they wish, they can even edit their source files directly in the student tool without having to switch over to their IDE.

(see www.jamtester.com for an animated demo)

The screenshot displays the JAM* Tester Student Tool v1.10.0521041 interface. The window title is "JAM* Tester Student Tool v1.10.0521041". The interface is divided into several sections:

- Table:** A table with columns: Field 1, Success, Percentage, testBestScore, and testWorstScore. The first row shows "DaveVttry", "1 of 2", "50.0%", a green box, and a red box with "F".
- Callout:** A yellow callout box with the text "Clicking on the 'F' for testWorstScore produces the error message here" pointing to the "F" in the table.
- Stack Trace:** A text area showing an error message: "t.framework.AssertionFailedError: expected:<90> but was:<85>". Below it are stack frames: "at junit.framework.Assert.fail(Assert.java:47)", "at junit.framework.Assert.failNotEquals(Assert.java:282)", "at junit.framework.Assert.assertEquals(Assert.java:64)", "at junit.framework.Assert.assertEquals(Assert.java:201)", and "at junit.framework.Assert.assertEquals(Assert.java:207)".
- Buttons:** A "Run Test Class" button is visible below the stack trace.
- JUnit Test File Manipulation:** A section with three buttons: "Auto-Generate JUnit Test Class for your Source", "Create new JUnit Test Class", and "Load JUnit Test Class".
- Code Editor:** A text area containing JUnit test code. The code includes imports, a class definition for "StudentTest" extending "TestCase", and two test methods: "testBestScore()" and "testWorstScore()". The "testBestScore()" method creates a "Student" object with scores 90 and 85 and asserts that the best score is 90. The "testWorstScore()" method asserts that the worst score is 85. A "main" method is also present for running the tests.

Can use of JUnit alleviate students' stress levels?

If you have read my booklet to this point, you will not have a hard time understanding that JUnit will indeed reduce or likely *eliminate* student stress. If you are starting your reading at this point, this may sound far-fetched.

Stress in students arises from uncertainty. Using some of the traditional procedures, students cannot be certain that their coding is actually and accurately finished. Although they *can* be capable of *convincing* themselves that their code is perfect, they turn in their solutions, still anxious that they may have forgotten something—probably something fundamental, something that will cost them valuable points.

However, by using JUnit, especially from the beginning of the year, the student is virtually guaranteed that she will turn in a perfectly working Java class. *You* are giving her that chance. She knows that if she invests the time, she will reap the benefits. Most importantly, because she knows exactly where she stands academically, she can operate essentially stress-free. *She* is in control of her grades.

The famous physicist Niels Bohr once said, “An expert is a man who has made all the mistakes. . . .” However, we also know that the student is a person who becomes stressed at the prospect of *making* mistakes. JUnit gives your students the academic and emotional freedom to make mistakes and become unstressed experts.

How does the use of JUnit increase student-student cooperation?

While I do not allow my students to pass their source code around the lab for others to see, I do encourage them to share their JUnit files. This way, they share the work associated with devising test cases by brainstorming together. A natural and positive outcome of a student discussion of test cases is that the students themselves analyze their own code. In so doing, they may very well discover a deficiency in their own algorithm or design before they even write or run the test.

If your students work in pairs, they may jointly create test suites. In a recent study, Kessler and Williams determined that this technique reduces cheating. Pair-programming, along with unit testing, is part of a discipline of software development called Extreme Programming. See bibliography.

Selected Bibliography

- Vaaraniemi, Sami. "The benefits of automated unit testing." *The Code Project*. 9 November 2003. 29 April 2004 <<http://www.codeproject.com/gen/design/onunittesting.asp>>.
- Jeffries, Ron. "Essential XP: Emergent Design." *Xprogramming.com: An Extreme Programming Resource*. 21 October 2001. 29 April 2004 <<http://www.xprogramming.com/xpmag/expEmergentDesign.htm>>.
- Jeffries, Ron. "What is Extreme Programming?." *Xprogramming.com: An Extreme Programming Resource*. 08 November 2001. 29 April 2004 <<http://www.xprogramming.com/xpmag/whatisxp.htm>>.
- Kessler, Robert and Laurie Williams. "Experimenting with Industry's 'Pair-Programming' Model in the Computer Science Classroom." *Pair Programming*. 29 April 2004 <<http://www.pairprogramming.com>>.
- Weirich, Jim. "Design by Contract and Unit Testing." Online posting. 6 July 2003. Ruby Buzz Forum. 29 April 2004 <<http://www.artima.com/forums/flat.jsp?forum=123&thread=6794>>.