King Saud University
College of Computers and Information Sciences
Department of Computer Science
2nd Semester, 1427/1428H

Development of an Automated Testing Tool for Students' Programs

Report (Part 1)

Submitted by:

Alia Bahanshal - ID: 425221329
abahanshal@yahoo.com

Submitted To:

Dr. Ghazy Assassa

Submission Date:

May 30, 2007

Abstract

There is a need to increase the quantity and quality of the treatment of testing in the curriculum because 100% of instruction is spent on the process of developing software, and very little to how to test software.

Furthermore, instructors and teaching assistants are overburdened with work while teaching courses and have little time to devote to additional assessment activities. On large courses, providing feedback on several programming assignments requires automatic assistance. As in the Computer Science Department at King Saud University they have 150 students enrolled in (CSC112) an introductory java programming course which requires a lot of works from the instructors and teaching assistants to assess each student's work with the same standards. The obvious benefits of using automatic assessment tool to assess the students programs are objectivity, consistency and speed of assessment.

This project aims to design and Implement an automated black-box (functional) testing tool for student's programs in an introductory Java programming course and investigate the impact of using automated testing tool in both students and educators.

Acknowledgment

To my professor, Dr. Ghazy Assassa, thank you for making this project a very valuable learning experience.

Table of Contents

Abstract 2

Chapter 1: Introduction 5
1.1 Introduction 5
1.2 Problem Statement 6

Chapter 2: Literature Review 7
2.1 Introduction to Software Testing 7
2.1.1 What Is Software Testing? 7
2.1.2 Level of Testing 7
2.1.3 Software Testing Strategies 8

2.2 Introduction to Automated Testing 9
2.2.1 What is Automated Testing 9
2.2.2 Automated Testing Life Cycle Methodology 9
2.2.3 Automated Testing Environment 11
2.2.4 Automated Testing Techniques 12
2.2.5 Manual Testing Vs. Automated Testing 14
2.2.6 Benefits of Automated Testing 14

2.3 Automated Testing Tools 16
2.3.1 A Survey of Coverage Based Testing Tools 16

2.4 Case Study: Automated Testing in Courses 19
2.4.1 Challenges on programming courses 20
2.4.2 Assessing programming assignment 22
2.4.3 Integrate Software Testing Throughout the Curriculum 22
2.4.4 Test Driven Development In Courses 23
2.4.5 Test-driven learning (TDL) 24
2.4.6 Automated Grading and Assessment Systems 24

2.5 Related Works 26
2.5.1 The Web-Based Grading Project (WBGP) 26
2.5.2 Web-CAT: A Web-based Center for Automated Testing 27
2.5.3 Submit and Progtst: 27
2.5.4 List of Previous Projects Features: 28

Chapter 3: System Analysis 31
3.1 The Automated Testing Tool for Students' Program Goals and Objectives 31
3.2 The Automated Testing Tool for Students' Program Features and Specifications 32
3.3 Automated Testing Tool For Students' Program Architecture: 36
3.4 Project Plan: 37
3.5 Java Testing Tools 38

References 40

Chapter 1: Introduction

1.1 Introduction

Testing is becoming more important with the huge improvement in the software developing because the programmers' use of the GUI made software products more complex and led to the development of new testing tools.

In [Srivastava, 2002] the author explained that software testing may be viewed as a sub-field of software quality assurance but typically exists independently where software process specialists and auditors take a broader view on software and its development, examine and change the software engineering process itself to reduce the amount of faults that end up in the code or deliver faster.

[Li, 2004] shows that performing computer software testing can be done at different levels early from unit testing and moving on to integration testing, systems testing and acceptance testing. During the early stages of the testing cycle, the software developer does most of the testing and this activity is boring, tedious and uncreative. Regardless of the methods used or level of formality involved the desired result of testing is a level of confidence in the software so that the developers are confident that the software has an acceptable defect rate.

In [Berner, 2005] the author shows a problem with software testing that the number of defects in a software product can be very large and bugs that occur infrequently are difficult to find in testing. A rule of thumb is that a system that is expected to function without faults for a certain length of time must have already been tested for at least that length of time. This has severe consequences for projects to write long-lived reliable software.

[Mayer, 1976] explains a common practice of software testing that is performed by an independent group of testers after the functionality is developed but before it is shipped to the customer. This practice often results in the testing phase being used as project buffer to compensate for project delays. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes. Another common practice is for test suites to be developed during technical support escalation procedures. Such tests are then maintained in regression testing suites to ensure that future updates to the software don't repeat any of the known mistakes. It is commonly believed that the earlier a defect is found the cheaper it is to fix it.

Manual testing was and still the known way for software testing but it is also a very highly costing one. In [Dustin, 1999] the author shows that software project managers and software developers today build applications while facing the challenge of doing so within an ever-shrinking schedule and with minimal resources. As part of their attempt to do more with less, organizations want to test software adequately, but as quickly and thoroughly as possible. To accomplish this goal, organizations are turning to automated testing.

In [Niemeyer, 2003] the authors showed number of benefits of the automated testing For one, the tests are repeatable so when a test is created, it can be run each time the testing process is launched. Automating testing reduces the fatigue of performing testing manually, which leads to more consistent results. Also, because the tests are automated, they're easy to run, which means that they will be run more often. As new bugs are discovered and fixed, tests can be added to check for those bugs, to ensure that they aren't reintroduced. This increases the overall completeness of testing.

To accomplish the testing phase successfully we need qualified testers that will perform all the testing tasks or monitor the test automation. The first step to produce fine testers starts from first years of education according to [Shepard, 2001] which showed that more testing should be taught to CS students because they are not well equipped to apply widely practiced testing techniques, and they are graduating with a serious gap in the knowledge they need to be effective software developers. Even new software engineering curricula tend to be weak in software testing because highly effective practices such as software inspection and testing are hardly taught at all, and many computer science professors do not know or care what inspection is and why it is valuable.

Furthermore, instructors and teaching assistants are overburdened with work while teaching courses and have little time to devote to additional assessment activities. Using an automated testing tool to test students program would benefit in both reducing the amount of assessment work on instructors and allow them to devote more efforts and time on teaching testing practices and techniques to produce good quality testers in the future.

1.2 Problem Statement

The objective of this project is to design and develop an automated black-box (functional) testing tool for student’s programs in an introductory Java programming course (CSC112) and investigate the impact of using such tool in both students and educators.

Chapter 2: Literature Review

Many researches were made and papers were published in the test automation topic. The results from collecting and studying these works are:

- Selecting this important topic.
- Understanding the scope of the proposed idea.
- Identifying the objectives via analyzing the optimistic conclusions.
- The desire to experiment such experiences in our university.

2.1 Introduction to Software Testing

2.1.1 What Is Software Testing?

[Myers, 1976] defines testing as “the process of executing a program or system with the intent of finding errors.”

Other definitions of testing include: “Finding bugs in programs”, “Showing correct operation of a program”, “Testing is the process of establishing confidence that a program or system does what it is supposed to do”.

Software testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors.

2.1.2 Level of Testing

There are four levels of software testing. Each level builds on the last.

- **Unit testing** tests the minimal software component that can be tested. [Barriocanal, 2002]
- **Integration testing** exposes defects in the interfaces and interaction between integrated components.
- **System testing** tests an integrated system to verify that it meets its requirements.
- **Acceptance testing** allows the end-user or customer to decide whether or not to accept the product.

2.1.3 Software Testing Strategies

[Myers, 1976] explained two different testing strategies:

- **Black-Box Testing**

Black box testing takes an external perspective of the test object to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid input and determines the correct output. There is no knowledge of the test object's internal structure.

This method of test design is applicable to all levels of software testing: unit, integration, system and acceptance. The higher the level, and hence the bigger and more complex the box, the more we're forced to use black box testing to simplify.

- **White-Box Testing**

White box testing or logic-driven testing (clear box testing, glass box testing or structural testing) uses an internal perspective of the system to design test cases based on internal structure. It requires programming skills to identify all paths through the software. The tester chooses test case inputs to exercise all paths and determines the appropriate outputs.

While white box testing is applicable at the unit, integration and system levels of the software testing process, it's typically applied to the unit.

2.2 Introduction to Automated Testing

2.2.1 What is Automated Testing

[Dustin, 1999] explains test automation; it is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

Commonly, test automation involves automating a manual process already in place that uses a formalized testing process, such process includes:

- Detailed test cases, including predictable "expected results", which have been developed from Business Functional Specifications and Design documentation.
- A standalone Test Environment, including a Test Database that is restorable to a known constant, such that the test cases are able to be repeated each time there are modifications made to the application.

2.2.2 Automated Testing Life Cycle Methodology

In [Dustin, 1999] the author discussed the automated test lifecycle methodology which comprises six primary processes or components:

- **Phase 1: Decision to Automate Testing**

  During this phase, it's important for the test team to manage automated testing expectations and to outline the potential benefits of automated testing when implemented correctly. A test tool proposal needs to be outlined, which will be helpful in acquiring management support. Some of the issues that organizations face when adopting automated test systems include those outlined below:

  - Finding and hiring test tool experts.
  - Using the correct tool for the task at hand.
  - Developing and implementing an automated testing process, which includes developing automated test design and development standards.
  - Analyzing various applications to determine those that are best suited for automation.
  - Analyzing the test requirements to determine the ones suitable for automation.
  - Training the test team on the automated testing process, automated test design, development, and execution.
  - Initial increase in schedule and cost.

- **Phase 2: Test Tool Acquisition**

  This phase guides the test engineer through the entire test tool evaluation and selection process, starting with confirmation of management support. Since a tool should support most of the organizations' testing requirements, whenever feasible the test engineer will need to review the system's engineering environment and other organizational needs and come up with a list of tool evaluation criteria.

- **Phase 3: Automated Testing Introduction Process**

  This phase outlines the steps necessary to successfully introduce automated testing to a new project team. ***Test process analysis*** ensures that an overall test process and strategy are in place and are modified, during the test process analysis, techniques are defined. Best practices are laid out, such as conducting performance testing during the unit-testing phase. ***The test tool consideration*** process includes steps that investigate whether incorporation of automated test tools that have been brought into the company without a specific project in mind now would be beneficial to a specific project.

- **Phase 4: Test Planning, Design, and Development**

  The test planning stage represents the need to review long–lead-time test planning activities. During this phase, the test team identifies test procedure creation standards and guidelines. The test design component addresses the need to define the number of tests to be performed, the ways that testing will be approached (paths, functions), and the test conditions that need to be exercised. Test design standards need to be defined and followed. For automated tests to be reusable, repeatable, and maintainable, test development standards need to be defined and followed.

- **Phase 5: Execution and Management of Tests**

  At this stage, the test team has addressed test design and test development. Test procedures are now ready to be executed in support of exercising the application under test.

- **Phase 6: Test Program Review and Assessment**

  Test program review and assessment activities need to be conducted throughout the testing lifecycle, to allow for continuous improvement activities. Throughout the testing lifecycle and following test execution activities, metrics need to be evaluated and final review and assessment activities need to be conducted to allow for process improvement.

2.2.3 Automated Testing Environment

2.2.3.1 Planning for Test Automation

The planning or the decision for automated testing is the first component to establish the automated testing environment as mentioned in the previous (ATLM).

2.2.3.2 Automatic Test Generation

[Edvardsson, 1999] explains how to reduce the high cost of manual software testing and at the same time increase the reliability of the testing processes by automating it and one of the most important components in a testing environment is an automatic test data generator, he proposed three different methods for generating test data: random, path-oriented, and goal-oriented test data generation.

- **Random Test Data Generation**

Generate input values for any type of program where a data type such as integer, string, or heap is just a stream of bits. For example, for a function taking a string as an argument we can just randomly generate a bit stream and let it represent the string. This method is the simplest one of generation techniques.

- **Goal-Oriented Test Data Generation**

In this method the generator finds input for any path, which reduces the risk of encountering relatively infeasible paths and provides a way to direct the search for input values as well. Instead of letting the generator generates input that traverses from the entry to the exit of a program. The goal-oriented approach is much stronger than random generation.

- **Path-Oriented Test Data Generation**

This method does not provide the generator with a possibility of selecting among a set of paths, but just specific one. Successively this leads to a better prediction of coverage. On the other hand it is harder to find test data. Path-oriented generation is strongest among the three approaches.

2.2.3.3 Logging Automated Test Results

Dealing with the test results is a very important concern, it is much more important than the test automation itself. A proper archiving and logging is required to prevent the failure of the whole testing process.

2.2.4 Automated Testing Techniques

2.2.4.1 Test-driven development

[Stephen, 2003] explains Test-driven development; TDD is a new technique that involves repeatedly first writing a test case and then implementing only the code necessary to pass the test. Test-driven development gives rapid feedback. Also in [Edwards, 2004] authors show some benefits of the test-driven development techniques: it can help build software better and faster, It offers more than just simple validation of correctness, but can also drive the design of a program. By focusing on the test cases first, one must imagine how the functionality will be used by clients. Therefore, the programmer is only concerned with the interface and not the implementation. This benefit is complementary to Design by Contract as it approaches code through test cases rather than through mathematical assertions or preconceptions. The power test-driven development offers is the ability to take small steps when required. It allows a programmer to focus on the task at hand as the first goal is to make the test pass. Exceptional cases and error handling are not considered initially. Tests to create these extraneous circumstances are implemented separately. Another advantage is that test-driven development, when used properly, ensures that all written code is covered by a test. This can give the programmer, and subsequent users, a greater level of trust in the code.

2.2.4.2 Automated Regression Testing

[Korel, 1998] explains regression testing which involves testing the modified program in order to establish the confidence that the program will perform according to the modified specification. In the development phase, regression testing may be used after the detection and correction of errors in a tested program. Also, the software maintenance cost may be significantly reduced if an automated regression testing was adopted rather than the time consuming and expensive traditional regression testing.

[Xie, 2005] shows that in comparing the actual outputs of two program versions regression testing is concerned in exposing the internal behavioral differences during the program execution, which can be used to track the quality of program output and not only testing the correctness of it.

- There are two types of regression testing:

1. **Progressive regression testing:** performed when the modified version of the software involves a change in the specification.
2. **Corrective regression testing:** performed when the modification does not involve a change in the software specification.

2.2.4.3 Automated GUI testing

[Li, 2004] explained GUI software testing as the process of testing a product that uses a graphical user interface, to ensure it meets its written specifications. This is normally done through the use of a variety of test cases with most software now driven by graphical user interfaces of such complexity that manual testing is now a time-consuming and costly task; there is an overwhelming case for automation.

GUI Automating test execution is normally justified based on the need to conduct functional regression tests. In organisations where development follows a Rapid Application Development (RAD) approach or where development is messy, regression testing is difficult to implement at all - software products may never be stable enough for a regression test to mature and be of value. A systematic approach to testing GUIs and using tools selectively for specific types of tests can be adopted and tools can be used to find errors during the early test stages.

- **Elements of Automated GUI Testing**
- A process
- A GUI Test Plan
- A set of supporting tools

2.2.4.4 High Volume Test Automation (HVTA)

[McGee, 2004] has presented HVTA techniques as the automated execution and evaluation of large numbers of tests, for the purpose of exposing functional errors that are otherwise hard to find.

By using the HVTA techniques, the reliability of software that works for long time with out stopping is increased because the ability for this technique to find specific types of errors much better than most traditional test techniques. High volume automated testing has been used to qualify safety-critical software, such as air traffic control systems, medical heart monitors, and telephone systems.

2.2.5 Manual Testing Vs. Automated Testing

In [Berner, 2005] the authors found that most new defects is detected by the manual tasks not the automated ones because 60% of the bugs are found during an automated testing effort and 80% are found during the development of the tests. In [Korel, 1998] authors showed that automated functional tests can be used for regression testing, If an organization is running the same manual regression tests repeatedly, then the automated tests can replace some of that effort, but they also add the effort to maintain the tests, which is sometimes more than the work required to just running the tests manually. Some of the effort means that test failures from an automated test run still must be analyzed manually. Also, any part of the process of provisioning and setting up the machine to run the tests, kicking off the test run, and babysitting it along the way that isn't automated will still require manual attention.

In [Srivastava, 2002] the author showed that the testing process will be most benefited if one has an optimum mixture of automated and manual tests. The automated tests should usually be those, which cover the most important features of the product and are likely to be executed in all the regressions. Automated and manual tests must coexist to improve the overall testing productivity. It is not possible to automate all test suites. Some tests cannot be automated because the tool or testing framework does not support automation. For example, with a console-testing tool, the automation of GUI tests will not be possible. There might be other tests, whose automation is not possible because the product under test requires some manual hardware intervention to execute those tests, or the whole automation is not cost-effective which might require enormous amounts of the developers' time for automation/maintenance and might test a small, not very important feature of the product under test.

2.2.6 Benefits of Automated Testing

Software testing accounts for 50% of the total cost of software development. In order to reduce the high cost of manual software testing researchers and practitioners have tried to automate it. In [Niemeyer, 2003] the authors listed many benefits of automated testing, for one, the tests are repeatable, so when a test is created it can be run each time the testing process is launched. Automating testing reduces the fatigue of performing testing manually, which leads to more consistent results. Also, because the tests are automated, they're easy to run, which means that they will be run more often. As new bugs are discovered and fixed, tests can be added to check for

those bugs, to ensure that they aren't reintroduced. This increases the overall completeness of testing.

Testing is a repeatable process and automated testing achieves an important part of the testing process by making it possible to conduct regression testing, to retest the same scenario again.

Automating the process maintains consistency from one run of the test to the next, regardless of how much time passes between the two runs of the tests or who is executing the tests because consistency issues are easiest to observe in teams with multiple testers and developers, but even a single tester would rarely conduct the same tests the same way each time.

Automated testing use self-documenting system, which is the best kind of documentation which does not have to be written and yet is guaranteed to be correct.

By automating the testing process, the computer will usually execute the testing process in less time than it takes a tester to perform manually. Again, manual testing has its place; the advantage of automated testing is that it can easily catch many of the problems before manual testing even begins.

The benefits of automated testing are:

- Is a repeatable process
- Uses a consistent approach
- Follows a documented process
- Frees up developer-hours for more profitable tasks
- Is expandable and flexible, with changes in code propagated to the testing procedure faster and more efficiently
- Negates the fatigue factor as development deadlines approach because automated tests will eliminate the stress and workload of manual testing on developers
- Produce a reliable system
- Improve the quality of the test effort.
- Reduce test effort and minimize schedule.

Some drawbacks are that some features don't easily lend themselves to automated testing. For example, sometimes automation-testing software can be used to test complex GUI applications, but often these applications must be tested by hand.

2.3 Automated Testing Tools

2.3.1 A Survey of Coverage Based Testing Tools

In [Yang, 2006] the authors performed a survey that studies and compares 17 coverage-based testing tools.

The study includes comparison of three features:

- Code coverage measurement
- Coverage measurement criteria
- Automation and reporting

- **Code coverage measurement**

All tools included in this survey have coverage measurement capability, but may apply only to a limited set of programming languages, some to C/C++ only, some to Java only, some to both, and some to other languages such as FORTRAN, COBOL, or JavaScript. Tools covered in the survey are listed below:

	C/ C++	Java	Other
Agitar		X	
Bullseye	X		
Clover		X	.net
Cobertura		X	
CodeTest	X		
Dynamic	X		
EMMA		X	
eXVantage	X	X	
gcov	X		
Insure++	X		
Intel	X		
JTest		X	.net
JCover		X	
Koalog		X	
PurifyPlus	X	X	Basic, .net
Semantic Designs	X	X	C#, PHP, COBOL, PARLANSE
TCAT	X	X	

Table1: Coverage Tools and the Languages to Which They Apply

- **Coverage measurement criteria**

Picking the right measurement requires balancing usability with thoroughness. Some tools provide various levels of code coverage information. There is a large variety of coverage measurement criteria: statement coverage (line coverage), decision coverage (branch coverage), path coverage, function/method coverage, class coverage and so on. Table 2 gives a list of tools with their coverage measurement criteria.

	Line	Decision	Method	Class
Agitar [3]	X	X	X	X
Bullseye [5]		X	X	
Clover [6]	X	X	X	
Cobertura [7]	X	X		
CodeTest [8]	X	X		
Dynamic [9]	X	X	X	
EMMA [21]	X		X	X
eXVantage [4]	X	X	X	
gcov [10]	X			
Insure++ [11]	X			
JCover [13]	X	X	X	X
JTest [15]	X	X		
PurifyPlus [16]	X		X	
SD [17]	X	X	X	X
TCAT [18]	X	X	X	X

Table2: Levels of Coverage Measurement Provided By Tools

- **Automatic Test Generation and Reporting**

Another important feature for comparison is automation; automation of testing process includes many steps, such as test case generation, test execution, and test oracles.
Another important automation area is test generation, which is more tightly linked with code coverage. None of the tools in our list can generate test cases for C/C++ code, but Parasoft, Agitar, and eXVantage claim the capability of generating Java test cases automatically. Parasoft has its patented test case generation technology. Agitator provides a certain level of automation by combining test suite generation and execution.
Besides automation, a friendly graph interface is also an important feature for comparison. The user interface can be a decisive element for a tool's usability. The first impression of a software tool is very important to users in their tool selection. There are two aspects of the user interface in this case: deployment and report generation.
Some tools have both a GUI version and a batch mode to suit the requirements of different users. One part of the GUI display or the output of the batch mode is the coverage report. Most commercial products include sophisticated report generation components, some of which are graph-based and some file-based.
See table 3 for a list of report formats.

	GUI	File-based	Notes
Agitar [3]	X		
Clover [6]	X	X	PDF, XML
Cobertura [7]	X	X	XML
Dynamic [9]		X	
JCover [13]	X	X	XML, CSV
Koalog[14]	X	X	CSV, LaTex, XML
JTest [15]	X	X	Group reporting system
PurifyPlus [16]	X		
SD [17]	X	X	Test coverage vector file, XML
TCAT [18]		X	
eXVantage [4]	X	X	Customizable

Table 3: Tool Reporting Formats

2.4 Case Study: Automated Testing in Courses

With the rapid evolution of computers and information technology, computer science has gained a significant role in the technology education. The basics of computer science are needed in several curricula.

In [Douce, 2005] the authors showed that programming problems and assignments are considered essential elements of software engineering and computer science education and it is usually incorporated to the introductory studies. Programming assignments can help students become familiar with the attributes of modern programming languages, become acquainted with essential tools, and to understand how the principles of software development and design can be applied. The assessment of these assignments places significant demands on the instructor's time and other resources. An automated tools and utilities can be adopted to simplify the tasks that both instructor and student had to carry out so that the assignment could be assessed automatically.

In [Snyder, 2004] the author explained one way to improve the confidence that a program does what it is supposed to do, both from the student point of view and from the teacher point of view, is to use test cases. But, for beginning students to use a testing methodology, the methodology must be fairly simple and consistent from one program to the next. Although the programming model used has input coming from the keyboard or an input text file, there is another source of input, and that is input that is embedded in the program itself.

In [Shepard, 2001] the authors showed that less than 50% of undergraduate curriculum is devoted to testing issues, and this percentage, and resultant depth of understanding, should increase by providing the students with an understanding of:

- The broad issues of testing.
- The proper places for testing activities in software processes.
- How to plan and design good test strategies.
- How to minimize testing.

In [Edwards, 2004] the author showed that many computer science educators have been looking for an effective way to improve the coverage of software testing skills that undergraduates receive. So, the time devoted to testing activities in industry will be reduced as a result of better design and testing practices they will have.

2.4.1 Challenges on programming courses

In [Ala-Mutka, 2004] the authors introduced some typical problems faced by students and teachers and review existing assessment practices for programming.

1. Students have often difficulties in building mental model of computer programs, since it differs from the structure of natural language. Even when the students have learned the programming concepts and languages, they may still lack the skills for using this knowledge to create computer programs. Thus, if the students are expected to learn to generate computer programs, it requires “hands-on” experience with practical programming tasks.
2. Students don’t work on voluntary assignments, a possible reason for that is sometimes they see programming assignments as separate tasks with unnecessarily complex assessment requirements. A proposed solution is getting the students involved with the practical components of the course required frequent (online) evaluations, and they should either frequently submit laboratory assignments or be required to answer weekly quizzes about the contents of the assignments.
3. Real-world applications and software projects are so large that they cannot be covered on one or even several courses, but still the students should learn skills for working in such situations. Therefore, the complexities and practices of professional work must be introduced partly in theory and partly by assignments that are simplified from real-world systems. For teachers, this means that they need to plan the assignments very carefully. For students, this means that they are required to learn and follow several basic rules, although the effects of their neglections cannot always be shown in practice.
4. Novice programmers are usually not very good at evaluating their work, as even incorrect programs can seem to work as desired, because complex requirements of good and correct programming practices make the assignments hard to assess. Also if he students have not yet learned the issues of “good programming”, they cannot assess them effectively either. For this reason, assessment and feedback by an expert is always needed.
5. The work needed for giving good feedback places heavy workload for teachers. also the issues of assessment objectivity, consistency and speed are hard to take care of. These problems become emphasized on large courses, where several tutors are needed for the assessment work.

6. Another widely recognized problem is cheating. Since computer programs are in electronic form, they are easy to copy. In [Sheared, 2002] the authors had concerned results in their study of IT students' attitudes to cheating at two universities. 34% of the respondents admitted that they had copied a majority of an assignment from a friend. 53% had collaborated on an assignment that was meant to be completed individually. This fact needs to be taken into account for ensuring students' learning.

- **Challenges to adopting software testing practices in assignments:**

[Edwards, 2004] stated five perceived roadblocks to adopting software testing practices in assignments:

1. Software testing requires experience at programming, and may be something introductory students are not ready for until they have mastered other basic skills.
2. Instructors just do not have the time (in terms of lecture hours) to teach a new topic like software testing in an already overcrowded course.
3. The course staff already has its hands full assessing program correctness—it may not be feasible to assess test cases too.
4. To learn from this activity, students need frequent, concrete feedback on how to improve their performance at many points throughout their development of a solution, rather than just once at the end of an assignment. The resources for rapid, thorough feedback at multiple points during program writing just are not available in most courses.
5. Students must value any practices we require alongside programming activities. A student must see any extra work as helpful in completing working programs, rather than a hindrance imposed at the instructor's desire, if we wish for students to continue using a technique faithfully.

By combining a suitable testing technique with the right assessment strategy, and supporting them with the right tools, including an automated assessment engine, it is possible to overcome all of these difficulties.

2.4.2 Assessing programming assignment

One approach is to require students to test their own code in programming assignments, and then assess them on this task as well as on the correctness of their code solution. Two critical issues immediately arise one, what testing approach should students use? The approach must provide practical benefits that students can see, and yet be simple enough to apply across the curriculum. Second, how will students be assessed on testing tasks? In particular, if students must test their own code, and then be graded on both their code and their testing, how can we avoid doubling the grading workload of faculty and teaching assistants while also providing feedback frequently enough and specifically enough for students to improve their performance?

On large courses, providing feedback on several programming assignments requires automatic assistance. The obvious benefits of using automatic assessment tool to assess the students programs are the objectivity, consistency and speed of assessment.

Also the assignment descriptions and measurement criteria are carefully designed by necessity, since they have to be programmed to the automaton, which will enhance the quality of assignment and make them more objective and the student will be able to understand carefully the desired output and when students are provided with clearly stated objectives and assessment criteria, they are able to control their learning process and become more self-directed learners.

In the following sections some approaches and automatic assessment tools and grading are presented.

2.4.3 Integrate Software Testing Throughout the Curriculum

In [Goldwasser, 2002] the authors presented an approach to teach student testing skills, students of a programming course were asked to submit both an implementation and test set. Student's grade was then dedicated on both the validity of a student's program on others' test sets and on how that student's test set performed in uncovering flaws in others' programs.

The advantages of this approach are:

1. Competitive scoring provides a bit of bright motivation to course work.
2. Students feel fully included in developing their own test sets.
3. Offers a wonderfully diverse environment for software testing.
4. The scoring system provides a quantitative evaluation of both program validity and test set quality that can be included as part of the overall grade.

2.4.4 Test Driven Development In Courses

[Stephen, 2003] explained how TDD can be practiced in courses, In TDD, student writes one or more test cases before adding new code. The test cases capture what behavior the student is attempting to produce. Then, the student writes new code, these tests tell when the student has achieved his latest goal.

TDD is attractive testing approach for use in an educational setting for many reasons:

- It is easier for students to understand and relate to than more traditional testing approaches.
- It promotes incremental development, promotes the concept of always having a "running version" of the program at hand, and promotes early detection of errors introduced by coding changes.
- It directly combats the "big bang" integration problems that many students see when they begin to write larger programs, where testing is saved until all the code writing is complete.
- It dramatically increases a student's confidence in the portion of the code they have finished, and allows them to make changes and additions with greater confidence because of continuous regression testing.
- It increases the student's understanding of the assignment requirements, by forcing them to explore the gray areas in order to completely test their own solution.
- It also provides a lively sense of progress, because the student is always clearly aware of the growing size of their test suite and how much of the required behavior has already been completed. Most importantly, students begin to see these benefits for themselves after using TDD on just a few assignments.

The tool support that is available for TDD is also important. TDD frameworks are readily available, including JUnit for Java, and related XUnit frameworks for other languages. Although these frameworks are aimed at professional developers, similar educational tool support is also becoming available:

DrJava: which is designed specifically as a pedagogical tool for teaching introductory programming, provides built-in support to help students write JUnit-style test cases for the classes they write,

BlueJ: an introductory Java environment designed specifically for teaching CS1 also provides support for JUnit-style tests. BlueJ's JUnit support allows students to "record" simple object creation and interaction sequences as JUnit-style test cases. Such tools make it easy for students to write tests from the beginning, and also mesh nicely with an objects-first pedagogy.

2.4.5 Test-driven learning (TDL)

In [Janzen, 2006] the authors presented Test-driven learning (TDL) which is an approach to teaching computer programming that involves introducing and exploring new concepts through automated unit tests. TDL offers the potential of teaching testing for free, of improving programmer comprehension and ability, and of improving software quality both in terms of design quality and reduced defect density.

TDL can be employed starting in the earliest programming courses and continuing through advanced courses, even those for professional developers. Further, TDL can be applied in educational resources from textbooks to software documentation.

Test-driven learning has the following objectives:

- Teach testing for free
- Teach automated testing frameworks simply
- Encourage the use of test-driven development
- Improve student comprehension and programming abilities
- Improve software quality both in terms of design and defect density

2.4.6 Automated Grading and Assessment Systems

Unfortunately, instructors and teaching assistants are already overburdened with work while teaching computer science courses and have little time to devote to additional assessment activities. As a result, an automated tool for grading student programs is desirable. Many educators have used automated systems to assess and provide rapid feedback on large volumes of student programming assignments.

- **Generations of Assessment Systems**

[Douce, 2005] presented three generation of assessment systems:

1. First Generation – Early Assessment Systems

The earliest example of automated testing of programming assignments were found at 1960, where students submitted programs written in assembly language on punched cards rather than using compilers and text editors. A grader program was run against a student program and two different results were returned, either “wrong answer” or “program complete”. a key advantages was also the efficient use of computing resources, which allowed a greater number of students to learn programming.

2. Second Generation – Tool-Oriented Systems

The second generation assessment systems were developed using pre-existing tool sets and utilities supplied with the operating system or programming environment. The testing engines and systems are often used and activated in the form of command-line or GUI programming tools. An example of a second-generation assessment tool can be seen in the work of [Isaacson, 1989].

The second generation assessment systems programming assignments assessment involve two activities: checking the program to see that it operates correctly and checking the program to see that the programming style has been applied sensibly.

In [Reek, 1989] the TRY system was introduced, which introduced automated testing to the student and allows students to test their programs using a tester program. When the tester program is executed, the student is presented with a set of results and the test attempt is recorded. Like other systems of that period, testing is performed by a simple character-by-character comparison of results generated against expected ones.

In [Jackson, 1997] The ASSYST system was introduced to introduce a scheme that analyzes submissions across a number of criteria. ASSYST analyzes whether submissions are correct, whether submissions are efficient in their use of CPU time, and whether they have sensible metric scores that correspond to complexity and style.

3. Third Generation – Web-Oriented Systems

Third-generation assessment systems make use of developments in web technology and adopt increasingly sophisticated testing approaches.

Some of these works will be presented in related works sections.

2.5 Related Works

A literature review of previous efforts and works serves several purposes; it is possible to gain an appreciation of the approaches adopted by others in the past by examining the projects that have been undertaken, this is useful from both technical and didactic perspectives. Previous projects may be able to inform current development by illuminating the kinds of problems that were encountered and how they were overcome, whether a particular application was successful and whether other system developers had any insights into how contemporary systems should be constructed.

2.5.1 The Web-Based Grading Project (WBGP)

In (David, 2005) the author described web-based grading software for grading computer science projects which was developed at Ohio University, and it is an open-source effort to provide a set of tools to help computer science educators build web versions of graded student assignments and provides facilities to build, test, and annotate student source code with comments concerning programming style and documentation.

Figure 1 - WBGP Interface

2.5.2 Web-CAT: A Web-based Center for Automated Testing

In (Edwards, 2004) the author presented Web-CAT, an advanced automated grading system designed to process computer programming assignments and was developed at Virginia Polytechnic Institute and State University.

Web-CAT can grade students on how well they test their own code and supports virtually any model of program grading, assessment, and feedback generation.

Web-CAT runs on a server and provides all of its capabilities via a web interface. All submission activity, feedback, viewing of results, and grading activities take place via the web browser.

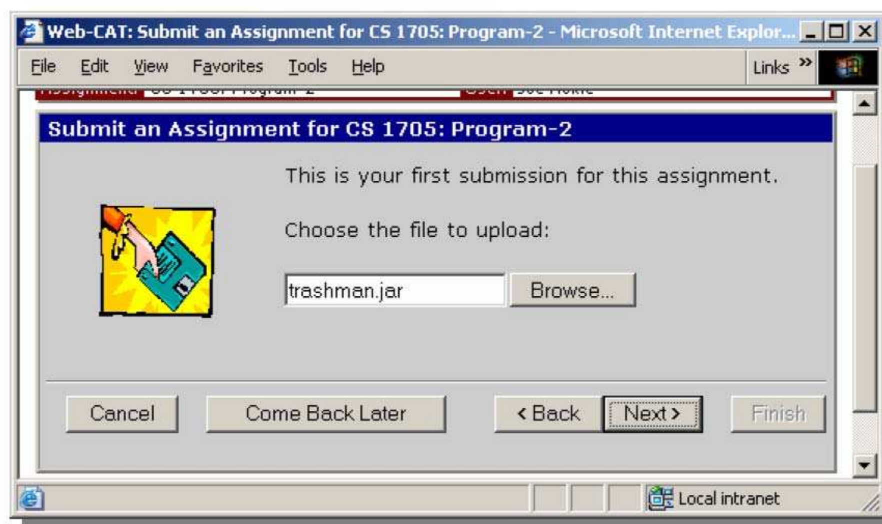

Figure 2 - Web-CAT: Students Submit Assignments

2.5.3 Submit and Progtst:

In [Harris, 2004] the author discussed tools used in introductory programming course that assist in the program evaluation process and make program grading easier. The two tools to automate the programming assignment submission and evaluation process are ***Submit*** and ***Progtst***.

- The ***Submit*** utility provides a mechanism for sending to the instructor all required materials in electronic form with an accurate time-stamp.
- The ***Progtst*** utility works in conjunction with submit to test programs when they are submitted.
- ***Submit*** and ***Progtst*** were presented by the Department of Computer Science at James Madison University, Harrisonburg.

2.5.4 List of Previous Projects Features:

Automated Testing Project	Feature	Description
1. WBGP	1. **Environment**	Linux/Unix based environments (Mac-OS, etc.).
	2. **Choose working directory**	The WBGP uses four main directories for each graded project: Working Directory, Submission Directory, Test Case Directory, Example Solutions Directory.
	3. **Configuring project**	A detailed non-easy configuration needed, for example: Execution time limit: the amount of time that the shell scripts will give each compilation/test case before it is Killed.
	4. **Setup test cases**	Setting up the test, the testing directory should be included.
	5. **Un-Tar students submissions**	Students submit their zipped files and in this step they will be unzipped under student directory.
	6. **Compile students projects**	Compile students' submitted files.
	7. **Evaluate compilation results**	Evaluation of results is made.
	8. **Execute test cases**	Run tests.
	9. **Evaluate results of testing**	Evaluate testing.
	10. **Evaluate comments**	Evaluate comments.
	11. **Evaluate Design/ Code**	Evaluate Design/Code.
	12. **Grade**	Grade and build WebPages.
2. Web-CAT	1. **Environment**	Plug-in-based web application
	2. **For instructors:**	Web-CAT serves as a course management system for instructors in order to conduct computer science courses at universities.
	• **Creating an assignment**	Web-CAT provides a wizard-based interface to instructors for setting up programming assignments.

	• **Uploading a student roster.**	Web-CAT provides instructors with the ability to upload a list of students that are enrolled-in a particular course.
	• **Viewing grades**	Web-CAT provides the instructor with the ability to view grades of students, either individually or as a class.
	• **Add Comments**	Instructors can add their own comments and point deductions to any source file line, enter overall comments on the entire assignment, and view or modify total deductions
	3. For students:	Web-CAT serves as an online submission system for students and allows them to view reports for already submitted assignments.
	• **Submitting an assignment**	Web-CAT allows students to submit an assignment for automatic grading and feedback.
	• **Viewing reports**	Web-CAT allows students to view reports for already submitted assignments using a wizard-based interaction.
	• **E-mail notification**	Students get automatic e-mail notification when their assignment grading has been completed
3. Submit Prog-tst	The ***Submit*** program is used to submit assignment files to the instructor.	
	1. For Student:	
	• **Displaying menu**	Menu of choices indicating which faculty member and for which course the files will be submitted.
	• **Menu of assignments**	The student is provided with a menu of assignments available for submission, specific to course-id.
	• **Submit assignment**	After the assignment is specified, the student is prompted for the names of the files to be submitted. After specifying the file name(s), the student then must respond to an honour pledge declaration and indicate they have received no unauthorized assistance in completing the assignment. If they fail to do so, the submit aborts.

	• **PDF report**	The student is provided with a PDF report. The report provides the submission information (student name, account, date, assignment, late penalty, honour pledge, etc) and a listing of the submitted source files.
	2. For Instructor:	
3. Submit Prog-tst	• **Specify a due date**	The instructor can specify a due date and a schedule of penalties for late submissions, a "late penalty" (if any) is calculated based on the time of submission.
	• **Create file directory**	- The submitted files are copied to a directory that is created to store the submitted material, the created directory and its contents all belong to the instructor and are not accessible to the student. - Since each submission results in a new directory being created, subsequent submissions do not overwrite earlier ones.
	• **Compile files**	- The ***Submit*** program compiles all submitted source code files (the instructor's copy). - If the program fails to compile, an error message is output and the submission aborts.
	• **Test files**	- If the program compiles, the executable that is produced is supplied to the ***Prog-tst*** program. For a number of test cases, the actual output generated by the user's executable is compared to the correct output. - If the student's executable fails a test, an error message is output and the submission aborts. - If the executable passes all the tests, a success message is output and the instructor's directory containing the submission is marked as correct.
	• **Generate report**	- In addition to the onscreen messages, ***Submit*** generates a submission report. - A copy of the report in text format is created both in the instructor's submission directory and in the working directory of the student. - If the program compiled correctly, a summary of the submit output is included. If a test failed, detailed information on that test is included.

Chapter 3: System Analysis

3.1 The Automated Testing Tool for Students' Program Goals and Objectives

1. **Increase the students testing skills by allowing students to understand and submit their own test cases and data along with their programs.**
   Students are not rewarded for performing testing of their own implementations. As a result, students perform less testing on their own. Instead, they rely on instructor provided sample data and ignore the possibility of varying scenarios.

   Using the automated tool will encourage more testing thinking and planning from the student and as a result better testers in the future.

2. **Reduce the amount of educators' works and time spent on assessment process.**
   Instructors are overloaded with work while teaching computer courses. It would be difficult for them to provide extensive feedback on every student program especially if the class size is a large number. The lack of appropriate feedback and assessment of student programs could serve to be a major difficulty to include software testing in the classroom.
   Using the automated testing tools will help the educators in the assessment process and reduce the amount of work needed to teach testing.

3. **Improve the introductory Java course quality by enhancing the assignments form to meet the automation requirements.**
   Students will practice using an enhanced version of assignments. The assignment descriptions and assessment principles are specified in a detailed form to be prepared for automatic assessment. This will involve students in more testing; as a result they will be aware of how to test their codes while writing their programs.

4. **Observe the collected data and investigate the results of using such system**
   The results of using the tool and both the students grades and the collected students' background can be used to analyze the impact of automaton on both students and instructors.

3.2 The Automated Testing Tool for Students' Program Features and Specifications

Actor	Use Case	Description	Priority	Notes
➢ **Instructor:**				
	1. Create Assignment	Instructor creates assignment and fills information: - Assignment name - Assignment description - URL - Upload assignment files - Grade - Max Test files	High	Course is created by another system.
	2. Edit Assignment	- Edit assignment information.	Normal	
	3. Specify a due date	- Assignment due date should be specified and if any penalty to be calculated.		
	4. Close Assignment	- According to the assignment due date, assignment will be closed automatically or manually by instructor.	Low	Instructor has two options either manually or automatically
	5. Delete Assignment	- Instructor can delete an assignment.	Low	
	6. Upload Model Answer	- Instructor upload model answer that student can view after assignment is closed.	Normal	
➢ **Instructor:**	**7. Test assignment files**	- Instructor can use the tool to test the assignment. (instructor copy)	High	
	8. Compile files	- The tool compiles all submitted source code files, if the program fails to compile, an error message is output and the submission aborts.	Normal	
	9. Register Student	- Instructor confirms student registration. - After instructor confirmation, the student is able to use the tool.	Low	

	10. View Student Results	- View student's testing results.	High	Many views options could be made.
	11. Grade Student	- Instructor grades the students according his recorded results, non submitted assignment will be graded zero automatically.	High	Failed testing records could be graded by viewing the code and test files.
	12. Add Comment	- Instructor can add comments about the student code.	High	
	13. View Students Grades	- Instructor views all students' grades of all assignments.	High	Import to excel option can be added.
	14. Delete All Students	- At the end of course instructors delete all registered students.	Low	To prevent students to use the tool after course ends.
	15. Delete Student	- Delete student and disallow him to use the tool.	Normal	To protect the collected results.
➢ **Student:**				
➢ **Student:**	**1. Sign Up**	- Student apply to use the tool by entering his information: - Name - User Name is the Student ID - Password - GPA - Grades in previous programming courses	Low	User Name is the student ID, GPA and previous grades for statistical analysis purposes.
	2. Sign in	- Login to be able to use the tool only confirmed and registered students can use the tool.	Low	
	3. Create directory	- Each student has his own directory - Each submission has its subdirectory under student directory	High	Previous submission will not be over write on them.

	4. Display menu	- Student selects assignment number where she will submit her program.	High	
	5. Submit assignment files	- Student is allowed to upload test files according to the assignment test files assigned limit. - Student can upload files for only opened assignments.	High	
	6. Delete assignment files	- Student deletes uploaded files to upload another set of files. - If files were deleted all recorded tests results for the assignment will be deleted. - Student can delete files for only opened assignments.	Normal	
	7. Compile files	- The tool compiles all submitted source code files, if the program fails to compile, an error message is output and the submission aborts.	Normal	
	8. Run Test	- Student run tests after uploading files	High	
	9. Save Results	- If test passes the student can save the results and a new record is added to be viewed by instructor. - Student can save failed tests, If due date was reached for example. - Student can save results for only open assignments. - Limited number of savings assigned by instructor.	High	Instructor could grade student for failed tests results by viewing the student program code and test files.
	10. Delete Results	- If test fail the student can delete the test results and try again. - Student can delete the passed test results to change his code for example. - Student can delete results for only open assignments.	High	
	11. View Results	- Student can view his results.	Normal	

	12. View Grade	- Student can view his grade given by instructor.	Normal	
	13. View Instructor Comments	- View Instructor comment on student code.	Low	
	14. E-mail notification	- Students get automatic e-mail notification when their assignment grading has been completed.	Low	

3.3 Automated Testing Tool For Students' Program Architecture:

- Automated Testing Tool for Students' Program is a web-based system with three tires architecture:

  1. **Client Browser Tire:**

     Interface of the client where the clients interact with system and it is capable to send and receive requests to and from the server.

  2. **Web Server Tire:**

     All java classes and testing are placed in this tire and it is capable to serve HTML pages to be viewed by the client, and handle their requests.

  3. **Database and Files Server Tire:**

     This tire holds submitted data, where files server holds the submitted files; the database holds the related user information such as grades, results...Etc.

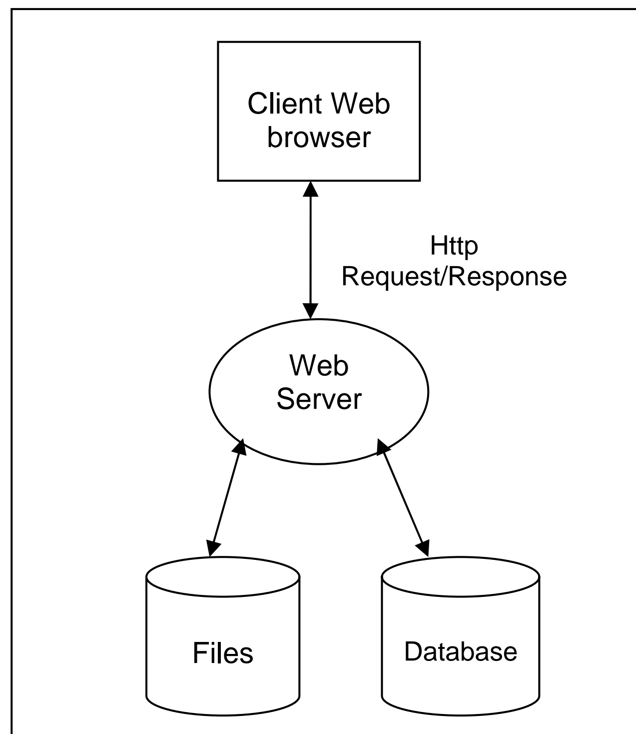

Figure 3 - Automated Testing Tool for Students' Program Architecture

3.4 Project Plan:

Tasks	Months 2007	March	April	May	June	July	August	September	October	November	December
Literature Review											
System Analysis											
System Design											
Implementation											
Testing											
Impact Evaluation											
Documentation											

3.5 Java Testing Tools

- **List of Available Tools**

Java Testing Tool	Description	URL
1. JUnit	JUnit is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java. JUnit is Open Source Software.	JUnit
2. Cactus	Cactus is a simple test framework for unit testing server-side java code e.g. Servlets. The intent of Cactus is to lower the cost of writing tests for server-side code. It uses JUnit and extends it.	Cactus
3. Abbot	The Abbot framework is a Java library for GUI unit testing and functional testing. It provides methods to reproduce user actions and examine the state of GUI components. The framework may be invoked directly from Java code or accessed without programming through the use of scripts. Abbot is a friendly JUnit extension for GUI testing	Abbot
4. Jameleon	Jameleon is an acceptance-level automated testing tool that separates applications into features and allows those features to be tied together independently, creating test-cases. These test-cases can then be data-driven and executed against different environments. Even though it would be possible to write unit tests using Jameleon, Jameleon was designed with integration and acceptance-level testing in mind.	Jameleon
5. TestNG	TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionality that make it more powerful and easier to use, such as: Flexible test configuration. - Default JDK functions for runtime and logging (no dependencies). - Powerful execution model- Supports dependent methods.	TestNG

6. TESTARE	TESTARE is a testing framework that aims to simplify the test development process for distributed enterprise JAVA applications. It tries to achieve this goal by focusing on two main directions: * provide straightforward and easy to use "in container" testing capabilities * provide native support for test environment management.	**TESTARE**
7. Jemmy	Jemmy is a JavaTM library that is used to create automated tests for Java GUI applications. It contains methods to reproduce all user actions which can be performed on Swing/AWT components (i.e. button pushing, text typing ...). JemmyTest is a program written in Java which uses the Jemmy API to test applications.	**Jemmy**
8. Jacareto	Jacareto is a capture & replay tool for programs written in Java. You can capture actions on applications and replay them later on (like macros). Jacareto can be used for many purposes: * GUI tests * Creation of animated demonstrations * Creation of macros	**Jacareto**
9. JTR Java Test Runner	JTR (Java Test Runner) is a framework meant for fastening the building of both complex and simple test environments. It is based on concepts such as Inversion of Control, and is ready for EJB and JMS testing. The JTR 2.0 framework will give you the chance to code only the testing logic. All the boring middleware-related tasks (connecting to Connection Factories, opening connections, sharing connections, opening sessions, handling exceptions, retrieving administered objects) are carried out by the JTR 2.0 runtime on your behalf according to what you have stated in the jtr.xml configuration file.	**JTR Java Test Runner**
10. Jetif	The Jetif is a regression test framework in pure Java. It provides a simple and flexible architecture for Java unit testing and functional testing, and used for testing in both individual and enterprise software development. It's easy to use, but powerful, and with some important features for enterprise software testing. This project was inspired by JUnit, JTestCase and TestNG. There are several ideas come from JUnit, for example the assertion mechanism and TestListener concept, so it's easy to move to Jetif from JUnit.	**Jetif**

➢ **Selected Testing Tool – JUnit**

References:

[1] Ala-Mutka, K. and Jarvinen, H. “Assessment Process for Programming Assignments”, In Proceedings of the IEEE international Conference on Advanced Learning Technologies, IEEE Computer Society, 2004, 181-185. *IEEE Xplore*

[2] Bach, J. “Test Automation Snake Oil”. In *Proceedings of the 14th International Conference and Exposition on Testing Computer Software*. Windows Tech Journal. 1999. *PDF*

[3] Barriocanal, E. G., Urbán, M. S., Cuevas, I. A., and Pérez, P. D. “An experience in integrating automated unit testing practices in an introductory programming course”, *SIGCSE Bull,* 34, ACM Press, 2002, 125-128. *ACM*

[4] Beck, K. "Aim, Fire," *IEEE Software*, vol. 18, no. 5, 2001, pp.87-89. *IEEE Xplore*

[5] Berner, S., Weber, R., and Keller, R. K. “Observations and lessons learned from automated testing”. In *Proceedings of the 27th international Conference on Software Engineering*, 2005, pp.571-579. *ACM*

[6] David, J.”Web-based Grading: Further Experiences and Student Attitudes”, In *Proceedings of ASEE/IEEE Frontiers in Education Conference,* IEEE, 2005, p 18-23. *IEEE Xplore*.

[7] Douce, C., Livingstone, D., and Orwell, J. “Automatic test-based assessment of programming: A review”, Journal on Educational Resources in Computing*,* 2005. *ACM*

[8] Dustin, E., Rashka, J. and Paul, J. Automated Software Testing: Introduction, Management, and Performance. Addison Wesley Professional, 1999. *HTML*

[9] Edvardsson, J. “A survey on automatic test data generation”. *In Proceedings of the Second Conference on Computer Science and Engineering in Linkoping,* 1999, pp.21-28. *PDF*

[10] Edwards, S. H. ”Using software testing to move students from trial-and-error to reflection-in-action”, In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 2004, pp.26-30, *ACM*

[11] Edwards, S. H. “Improving student performance by evaluating how well students test their own programs” J. *Educ. Resour. Comput.* 3, 2003. *ACM*

[12] Harris, J. A., Adams, E. S., and Harris, N. L. “Making program grading easier: but not totally automatic” J. *Comput*. Small Coll. 20, 2004, 248-261. *ACM*

[13] Isaacson, P. C. and Scott, T. A. "Automating the execution of student programs" SIGCSE Bull. 21, 2, 1989, 15-22. ACM

[14] Jackson, D. and Usher, M. "Grading student programs using ASSYST", In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 1997, 335-339. ACM

[15] Janzen, D. S. and Saiedian, H. "Test-driven learning: intrinsic integration of testing into the CS/SE curriculum", In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 2006, pp.254-258. ACM

[16] Korel, B. and Al-Yami, A. M. "Automated regression test generation", In *Proceedings of the 1998 ACM SIGSOFT international Symposium on Software Testing and Analysis*, ACM Press, 1998, pp.143-152. ACM

[17] Li, k. and Wu, M. Effective GUI Testing Automation: Developing an Automated GUI Testing Tool. John Wiley & Sons, 2004. PDF

[18] Link, J. Unit Testing in Java: How Tests Drive the Code, Morgan Kaufmann, 2003.

[19] Massol, V. and Husted, T. JUnit in Action. Manning Publications, 2003. PDF

[20] McGee, P. and Kaner, C. "Experiments with high volume test automation", *SIGSOFT Soft. Eng. Notes* 29, 5, 2004, pp.1-3. ACM

[21] Myers, G.J. The Art of Software Testing, John Wiley & Sons, 1976 PDF

[22] Niemeyer, G. and Poteet, J. Extreme Programming with Ant: Building and Deploying Java Applications with JSP, EJB, XSLT, XDoclet, and JUnit. Sams, 2003. HTML

[23] Reek, K. A. "The TRY system -or- how to avoid testing student programs", In *Proceedings of the Twentieth SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 1989, 112-116. ACM

[24] Shepard, T., Lamb, M., and Kelly, D. "More testing should be taught", *Commun. ACM* 44, 2001, pp.103-108. ACM

[25] Sheard, J., Dick, M., Markham, S., Macdonald, I., and Walsh, M. "Cheating and plagiarism: perceptions and practices of first year IT students", SIGCSE Bull, 2002, 183-187. ACM

[26] Srivastava, A. "Test Automation and Software Development", Technology Review#2002-07. 2002. Tata Consultancy Services. PDF

[27] Snyder, R. M. "Teacher specification and student implementation of a unit testing methodology in an introductory programming course", *J. Comput. Small Coll.* 2004, pp.22-32. ACM

[28] Stephen H. "Teaching software testing: automatic grading meets test-first coding", In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM Press, 2003, pp.318-319. ACM

[29] Xie, T. "Improving Effectiveness of Automated Software Testing in the Absence of Specifications", Doctoral Thesis. University of Washington, 2005. *IEEE Xplore*

[30] Yang, Q., Li, J. J., and Weiss, D. "A survey of coverage based testing tools", In *Proceedings of the 2006 international Workshop on Automation of Software Test*, ACM Press, 2006, pp.99-103

[31] WBGP: http://ace.cs.ohiou.edu/~juedes/wbgp/wbgp.html

[32] Web-Cat: http://web-cat.cs.vt.edu/

[33] JUnit: http://www.junit.org/

[34] Abbot: http://abbot.sourceforge.net/

[35] Jameleon: http://jameleon.sourceforge.net/index.html

[36] TestNG: http://www.beust.com/testng/

[37] TESTARE: https://testare.dev.java.net/

[38] Jemmy: http://jemmy.netbeans.org/

[39] JTR Java Test Runner http://jtrunner.sourceforge.net/

[40] Jetif: http://jetif.sourceforge.net/index.php