

Automated Software Testing in Educational Environment: A Design of Testing Framework for Extreme Programming

Ghazy Assassa, PhD
Dept. of Computer Science,
College of Computer and
Information Sciences,
King Saud University.
ghazy@ccis.ksu.edu.sa

Hassan Mathkour, PhD
Dept. of Computer Science,
College of Computer and
Information Sciences,
King Saud University.
mathkour@ccis.ksu.edu.sa

Bander Al-Ghafees
Information Systems,
General Organization
for Social Insurance.
balghafees@gosi.org.sa

Abstract

Automation of the testing process is a novel approach in software engineering that was proposed in the incremental Extreme Programming (XP) methodology. Test driven development process such as XP follows a core practice of 'Test first'; where developers are supposed to write test cases, in particular for unit testing, before they actually start coding the application increment. Applying the principle of test driven development has a positive impact on the way students develop their assignments. Students usually rely on trail-and-error approach in performing unit and functional testing. The present paper discusses how automated software testing in educational environment could be implemented and provides a testing framework for extreme programming that students and educators can share. The paper presents a web-based testing framework system to help students systematically run and track their test cases. The system acts as an integration-testing machine where students upload the latest code of both the application increment and the associated test cases. The system runs the test suite comprising all previously uploaded test cases together with the newly uploaded test cases; the objective being to make sure the newly uploaded test cases and application increment do not affect previously successfully integrated application increments. Students following XP will save the time it takes to run each test individually. Moreover, the testing framework will help students to easily create test cases by following the standards provided by the framework.

Key words

Extreme Programming, Unit Testing, Test Automation, Framework, Educational Environment.

1. Introduction

Extreme Programming (XP) practices and principles have helped many projects to finish on the expected time with good quality software products. Unlike traditional methods such as waterfall model, XP embrace changes while the wheel is spinning. However, the values of XP may be faced with some difficulty when it comes to apply in the academic field. For example, XP emphasizes on having a customer onsite, but within the educational environment it may not be possible to have such a customer to respond to students' questions [14]. Moreover, some of the practices that are highlighted by XP are hard to apply such as having a workspace where developers work together. However, one of the core practices that could be applied in academic field is testing. XP is a test driven development

process where developers should write test before they actually start coding [9]. Applying the principle of Test Driven Development has a positive impact in the way students write their programming assignments [4].

Most computer science departments introduce testing to their students. However, few of them give assignments on testing techniques. Many graduates of computer science department just know the basics of testing but they don't apply it because they think "testing is for professionals only" [8]. Therefore, most of them don't have the testing skills and they always rely on trail-and-error to find bugs in the programs they implement. Another issue that prevents students from testing their software is their lack of time. Creating test cases and running them would require some time that students may not be

privileged of having. Therefore, students usually don't spend time in creating test cases because they will not be graded; what will be graded are their programs not the test cases they write. Computer science students may not be encouraged or rewarded by the educators on performing testing on their own. Most of the programming courses in the academic field don't provide students with means to motivate them to test their code. Educators deviate from requiring students to perform software testing practices because of some obstacles. One of the difficulties educators may face is the lack of time. Instructors don't have the time to cover software testing details because they are already full with material they have to cover within their courses. Moreover, introducing software testing may require more work by the instructors since they have to assess the tests provided by the students as well.

It is possible to overcome the problems mentioned above by providing a common framework that students and educator can use. The testing framework should minimize the burden on teachers so they don't need to spend lengthy lectures explaining how to test programs. Moreover, the testing framework can reduce the time in assessing and grading students programs. These benefits relate to educators, yet there are many benefits students can gain as well. Providing a testing framework to students will encourage students to apply software testing to their programs. That is because the framework will make test cases easier to create and maintain. In addition, students will be able to automate running a set of test cases; therefore, time requirement is reduced. The framework will let students to gain professional testing skills, which cannot be gained with the normal methodology of trail-and-error.

The next section discusses some background information, the importance of testing, and test automation and related studies. Section 3 describes the framework and the main functions it should provide. Section 4 provides general overview about the framework design. Section 5 presents conclusion recommendations for future work.

2. Background and Related Studies

Before we dive deep into the proposed framework specifications and design, we will set

the stage by discussing the importance of testing and test automation in the following subsections. Moreover, we will look into some other related studies that have been done in creating such testing framework.

2.1. Importance of Testing

There always have been concerns about the software development process, whether it will lead to successful product at the end or not. Furthermore, producing full functional software is impossible without testing. Not only testing is conducted in the software industry, but testing is also performed in other industries such as airplanes, cars, furniture manufacturing. However, testing the software is totally different and is far more difficult. In the software development world, there are many types of testing, e.g., white-box testing, black-box testing, stress-testing, path testing, integration testing and many others. All of these types have different roles, and running all of these types of tests is not enough to ensure having the correct software. Besides, the software that is being developed may have infinite number of paths and combinations, which would require very long time to test all possible paths. Testing is considered one of the cornerstones of the software development process, and it could be the hardest part.

Software is usually made of small units or components, just like cars that are made of small parts. Each component must be tested extensively to make sure it functions correctly. Imagine software with thousands of components, and each component is also made of number of functions or methods. Thus, these functions or methods must also be tested. Is this the end? The answer to this is no. That is because each method or function also contains lines and different paths and the testing must cover all of these to be confident that the software will operate as expected when it is run in a production environment. Let us assume that testing is done, and an enhancement is carried out on one component; therefore, the previous tests must be run plus a new test for the newly introduced enhancement. The loop will continue and will never stop until the system is correctly working. Testing in this case requires repeatable task of running the test cases over and over, especially when the software development process embrace changes and the change is allowed at any point

during the development process. Agile software development, such as XP, requires intensive testing since changes are introduced while the software is being developed [6]. Therefore, there is a pressing need for a framework that allows developers and students to develop and run test cases easily and in an automated way.

2.2. Test Automation

To appreciate the idea of test automation, it is very important to understand why to automate. Tests automation provides many benefits over manual testing, though their initial cost may be high at the beginning they will payback later on the development process. Tests automation buys developers and testers extra time and mind space to deal with other aspects and to more effectively find defects in their software [7]. Moreover, since the tests can be run at any time and many times, developers will be able to easily reproduce the error and track the code to find the defects. On the other hand, it is very hard to reproduce the error in manual testing because sometimes while performing manual testing, a tester may not remember all the actions he/she performed [13].

XP is different from traditional development processes since the code changes continuously. Therefore, it is very tedious to rerun the tests again and again manually because that would kill the time. One may argue there is only a need to run tests for the newly added features; however, this argument is not valid since the new features may introduce problems to other dependant features [11]. Therefore, automating the tests would make it easy for the development team to run a set of old tests plus the new tests.

One of the important practices of XP is to release working software during a short iteration [2]. To be able to achieve this goal or practice, it is critical to run the tests before the product is released to the customer. Being able to automate the tests, the task of releasing the product would be much easier and faster. The automated test will enable the development team to “explore the whole product every day” [13].

Last but not least, human are not precise of comparing the actual results of the tests with the expected results. That is because human may forget the correct results, and they are not as

precise as computers when it comes to compare decimals.

Automating tests is very important while adopting XP. They have many benefits over the manual tests. Moreover, they let the team to drive the development process faster and give it more time with other issues.

2.3. Related Studies

Many testing frameworks have been developed over the past years; reacting to the great effort spent in testing software products before releasing into a production environment. Each framework has its advantages over the others. No one framework is suitable for all types of applications or environments. What is best for critical systems may not be affordable or appropriate for systems with less criticality. Furthermore, most of these frameworks were created to help developers, and were not developed with education in mind.

JUnit, Cactus, NUnit and Abbot are some of the well-known testing frameworks [1, 3, 5, 10]. Each of these frameworks has its advantages and disadvantages. For example, JUnit is operating-system independent testing framework that can operate under any type of operating system; whereas, NUnit can only be run under windows environment. Moreover, each of these frameworks supports a specific set of programming languages. JUnit, Cactus and Abbot support testing for programs written in Java; however, NUnit provides a testing framework for programs written in .NET languages such as C#. Of these frameworks only Abbot allows developers to test their GUI. In addition, Abbot supports writing acceptance tests which cannot be done using JUnit, Cactus or NUnit that only support unit testing. The test-data of test cases in these frameworks have to be embedded within the test case code which forces the users of these frameworks to compile the test case code each time they do change in the test-data. Table 1 shows a comparison between the above-mentioned frameworks.

Feature	JUnit	Cactus	Abbot	NUnit
OS dependent	No	No	No	Yes
Language support	Java	Java	Java	.NET
Unit testing	Yes	Yes	Yes	Yes
Acceptance testing	No	No	Yes	No
Test script language	Java	Java	XML	.NET
GUI support	No	No	Yes	No
Web support	No	Yes	No	No
Input data	Embedded	Embedded	Embedded	Embedded
User interface	GUI/CL	GUI/CL	GUI	GUI/CL
Capture/Playback	No	No	Yes	No
Reporting	On screen	On screen	O screen	On screen
Support distribution	No	No	No	No

Table 1: Comparison between four famous testing frameworks.

The frameworks described above were created for advanced developers and they do not address specific academic environment. It is true that students can get familiar with these frameworks, yet there can be more functionalities to be added for the education environment. In the following section, we will describe the main functions to be included in the proposed framework for adaptation to the academic environment.

3. Testing Framework Description

Students should be provided with a utility that could help them to practice testing techniques before they graduate and work in real industry environment. Therefore, the main goal of this framework is to provide functionalities that are most helpful to students in their educational environment. In the following subsections we will describe the educational environment requirements as well as the system features.

3.1. Educational Environment Requirements

Unlike the testing frameworks discussed in section 2.3, the proposed framework is targeting the educational environment. Just as there are special requirements for the industrial environment, there are special specifications for educational environment. The instructors and the students are the major two parties we have to study in the educational environment. Here we will describe the educational environment requirements from the two perspectives of instructor and student. Instructors at universities have specific goals for each course they teach.

For instance, freshmen students are taught introductory programming courses, which don't involve any testing technique. The instructor main aim of such courses is to explain the cornerstones of programming and not testing. This will leave students to test their programs with a trail-and-error technique. Testing is not included in most introductory courses because instructors have so much material they have to cover during the course semester. Moreover, testing is usually taught in a theoretical way and students don't get the chance to apply testing theories because there is no common environment that serves such purpose. Students in the educational environment have their requirements as well. First of all, most students are not experienced so they can pick any testing framework and use it to write test cases for their programs. This will require more time from students to find a framework and then learn how such a framework could be used. Most students are loaded with so many courses and they always consider time as a critical issue. Second, students believe that trail-and-error is faster in testing their programs than writing test cases. This claim is true when the testing framework is hard to use and does not support most of their requirements. Third, students are not motivated to write test cases because they are not graded on their tests, but their programs are graded based on their output.

The academic environment is quite different from the industrial environment. Students in academic environment may start developing their programs in one language and in the next course they may write their programs in another programming language. If the testing framework

only supports one programming language, students will put so much effort to learn a different testing framework for each programming language. On the contrary, in the industrial environment most developers stick to one programming language for more than two years, which give them the chance to learn another testing framework if they have to switch to another language.

3.2. System Features

The main objective of the system is to provide a testing framework that makes students' tasks of creating, running and maintaining test cases simple and easy. The system should help students to apply many testing techniques such as unit testing, regression testing and integration testing in simple way. Moreover, the system should provide standards for students in creating their tests cases that will make the instructors' tasks of teaching testing techniques easier. Besides, the system should be extendable to include academic data such as grade students programs.

The proposed framework will be an extension to the famous testing framework JUnit. The proposed system will provide some functions that are not present in JUnit. These functions shall make JUnit more applicable for academic use in term of simplicity. The following is a brief description of the proposed framework features.

- **Operating System Independent**
The system should be operating system independent because in the academic field there can be many operating systems which students may use to write their programs. Moreover, making the framework independent of the operating system will make it more portable.
- **Supports Testing for Java Programs**
Java programming language is the mostly taught language in the academic field. That is because this language is operating system independent and it is object oriented. Also, many new applications now are written in this language.
- **Supports Unit Testing**
One of the corner stones of XP is unit testing. Moreover, since the proposed framework is an extension of JUnit, it

should support unit testing by inheritance.

- **Supports Story Based Testing**
This deals with clustering test cases related to a specific story. In XP, each story may have more than one test case, that is a 'test suite'. The framework should provide a mechanism to relate each test case to a story. This will make the management of the test cases and test suites easier. Moreover, it will make the task of running the test cases for a specific story simpler.
- **Provides Code Coverage of the Tests**
At unit testing level, a test case may not test all possible paths of the code being tested. The framework should provide a function to show the percentage of code coverage of the application code being tested. This will ensure that the test case is complete and will enforce the developer to test all paths of the code.
- **Multi-Users Environment**
The framework should be a multi-user environment where more than one user can logon into the project and run/upload or view the status of the test cases. Most of the existing testing frameworks are standalone where only one user can run a test. This will allow students who are working on the same project to work simultaneously.
- **Provides Historical Data of Test Results**
The framework should store the results of each test, this will make reviewing the status of a test case easier. Much historical information will be available including number of successful runs, last run date and other information that helps students maintaining their test cases.
- **Allows Running Subsets of Test-Suites**
This feature deals with selecting only a subset of a test suite to be run. For XP refactoring, students may need to run a subset of a test suite that is affected by code refactoring. This will reduce the time to run the test for a given story.
- **Runs Tests-Suites Based on Code Changes**
The framework should provide the students with a functionality to run some or all tests associated with a specific story when changes, not

necessarily via refactoring, has been done in the code. This type of automation will help students to upload the application code to the system, and the system will show all associated tests from which the developers can pick tests to run.

- **Web-Based**
The framework should be a web-based system. This will make the system more usable and more reachable. Students can logon to the system through the internet and upload applications and test cases and run the test suites at any time from any place.
- **Reads Input from a file**
The framework should support reading the input data from an XML file. This will allow students to test the code with

different input data without the need to modify the test case code. Besides, this will allow instructors to make changes to the input data-file and run the tests to validate the application programs with their own input-data.

- **Provides Statistical Data**
The system should provide some statistics based on the project or on a specific story. Such statistics can reflect the progress of the project.

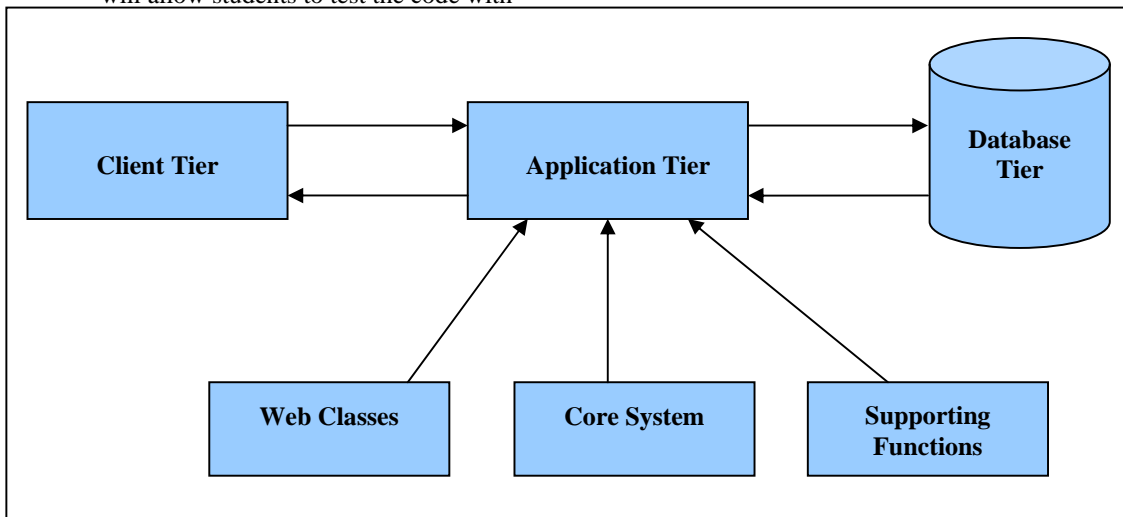


Figure 1: Proposed testing framework architecture.

4. Testing Framework Design

Since the system is an online one, it is decomposed into three tiers: Client Tier, Application Tier and Database Tier. Each of these tiers has its responsibilities as we discuss shortly. Figure 1 illustrates the architecture of the system.

4.1. Client Tier

The client tier is the user browser that is responsible of making HTTP requests as well as receiving responses from the application server where the application resides.

4.2. Application Tier

The core of the system is located in this tier. It

contains all functionalities the system provides to its users. The application tier is further decomposed into three major components: web classes, core system and supporting functions. Each of these three components is also decomposed into smaller subsystems as described below.

4.2.1. Web Classes

The web classes component deals with the user interfaces. There is no logic here except for some validation rules, such as leaving blank fields in a form. These classes will forward the requests to a master controller that will direct the requests to the appropriate subsystem.

4.2.2. Core System

The core system contains the main functionalities supported by the system. It is decomposed into four subsystems each of which has a special function as described below.

4.2.2.1. JUnit Subsystem

This is the JUnit framework which is used to create test cases. The subsystem is responsible to run the test cases and record the test results into the database server.

4.2.2.2. Test Data Loader Subsystem

This subsystem will enable users to put their test-data in a XML file instead of having the test-data embedded into the test case. The subsystem will read the file with the specified XML format and load the data so that user can test their code with multiple input test-data in a simple way.

4.2.2.3. Test Case Coverage Subsystem

The responsibility of this subsystem is to provide a feedback on the coverage percent of the test case. It will have an interface with JUnit subsystem in order to handle the code coverage feature of the test cases.

4.2.2.4. Grading Subsystem

The grading subsystem has special functions to allow an instructor to grade his students' assignments. This subsystem will handle processing students programs and record their assignments grades based on rules specified by the instructor's grading test case.

4.2.3. Supporting Functions

These functions are the additional features that are not related to the core system. Some of these functions are common such as logon and some are specific to special user type. The features are decomposed into three groups:

2.3.1. Students Functions

This will handle the students related functions such as uploading test case files, source-code files, etc. It should include all functions that can only be performed by the students.

2.3.2. Instructor Functions

The instructors special functions or features are grouped in this subcomponent. These functions include creating an assignment, viewing students' grades, etc.

4.2.3.3. Administrator Functions

The administration functions are grouped into this subcomponent. This includes functions such as adding student, assigning course to student, etc.

4.3. Database Tier

The database tier includes the database server and the file server. All files uploaded into the system are treated in this tier. The database will keep records of all data the system needs such as students' data, historical data of tests results, etc.

5. Conclusion

Testing is one of the important skills students should gain before graduation. Providing them with a proper framework where they can apply the principles of testing is an important factor in the academic area. We have proposed a testing framework for students that makes it easy for them to create, run, maintain and store test-cases. The framework is expected to reduce the learning time and testing will be standardized across the courses where testing can be applied. Moreover, instructors will have better way to assess their students programs by evaluating the students' test cases results and converge. The paper presents a general design of a testing framework that could be used with XP for automated software testing in educational environment. The proposed system features, design, and system architecture are discussed. Currently, the system is in the implementation phase. We plan to add some other features to the proposed framework to enhance its functionalities. Two important features that are planned for inclusion are supporting multiple programming languages and multi-threading to run test-suites.

References

- [1] Abbot: <http://abbot.sourceforge.net/>
- [2] Abrahamsson, Pekka. Salo, Outi. Ronkainen, Jussi. Warsta, Juhani. "Agile Software Development Methods: Review and Analysis", VTT-478, 2002.
- [3] Beck, Kent. "JUnit Test Infected: Programmers Love Writing Tests", Java Report, 1998, pp. 37-50.
- [4] Brown, Gary. "A Managers Guide To Test Driven Development With Unit Testing Frameworks", Salt Valley Software Services, 2003.
- [5] Cactus: <http://jakarta.apache.org/cactus/index.html>.
- [6] Cockburn, Alistair. Williams, Laurie. "Agile Software Development: It's about Feedback and Change", IEEE, Jun 2003, pp. 39-43.

- [7] Crispin, Lisa. House, Tip. "Testing in the Fast Lane: Automating Acceptance Testing in an Extreme Programming Environment", XP Universe, 2001.
- [8] Edwards, Stephen. "Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action", AMC Press, 2004, pp.26-30.
- [9] Highsmith, Jim. "Extreme Programming", Cutter Consortium newsletter, Feb 2000.
- [10] NUnit: <http://nunit.sourceforge.net>.
- [11] Jeffries, Ronald. "Extreme Testing", STQE, Mar 1999, pp. 23-26.
- [12] JUnit: <http://www.junit.org>.
- [13] Marick, Brian. "When Should a Test Be Automated?", 1998.
- [14] Mugridge, Rick. MacDonald, Bruce. Roop, Partha. Tempero, Ewan. "Five Challenges in Teaching XP", 2002.