Performance Testing based on Test-Driven Development for Mobile Applications

Heejin Kim
Department of Computer Science and Engineering, Ewha Womans University, Seoul 120-750, Korea
+82-2-3277-3508
heejinkim@ewhain.net

Byoungju Choi
Department of Computer Science and Engineering, Ewha Womans University, Seoul 120-750, Korea
+82-2-3277-2593
bjchoi@ewha.ac.kr

Seokjin Yoon
Electronics and Telecommunications Research Institute,
Daejeon 305-350, Korea
+82-42-860-5985
sjyoon@etri.re.k

ABSTRACT
Due to the tight schedule of product development for mobile applications and lack of performance testing methods, the product-oriented performance testing that is mostly done in the end of the development shows problems such as identifying a cause of detected faults, tracking down and modifying the faults when faults occur. The importance of testing is emphasized in TDD and the automated test framework is supported for efficient software development with unit tests. In this paper, we propose the methods of performance testing based on test-driven development with regard to non-functional factors as well as functionality of software during the software development process by advancing performance testing to the development stage and introduce a testing tool that assists performance testing on software development phase. It provides automation of test case generation and test execution at unit test level. It will eventually improve the development productivity as well as the reliability and quality of mobile applications by reducing the time and cost to execute tests in the process of the entire mobile applications development and helping to detect faults.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging – *Error handling and recover, Testing tools.*.

General Terms
Verification, Reliability, Performance.

Keywords
Performance Testing, Test-Driven Development(TDD), Mobile Applications

1. INTRODUCTION
Test-Driven Development(TDD) that has been noted in recent years is a evolutionary software development technique to lead development with automated tests[1]. It gives importance to efficient unit testing to make code cleaned up and work correctly and offers automated testing frameworks such as JUnit[2] for software to be developed efficiently. In TDD, tests are prepared by dividing the software functions to develop into small units and software is developed based on unit tests. The unit test is a code developed by the developer, which performs a very small, specific area of the code function to be tested. TDD dramatically reduces the time wasted on debugging and helps to speed up the development process and to develop reliable software[3].

In general, if the developers can be confident that the code meets all the tested requirements after unit tests pass, performance testing is fulfilled by comparing the integrated code with non-functional system requirements[4]. That is, performance testing is seldom conducted until a product is finished. There are several differences between performance testing of mobile applications and that of the other common software because of the limitations in processing power, memory, UI, and data input of mobile phone itself. Although it becomes highlighted as an important issue to test performance factors for improvement of products' quality related to running speed and stability, it is still insufficient so far. Owing to a tight schedule of mobile applications development, most of tests that are only done on the basis of GUI, functions and key events. When faults occur, it's hard to find the reason of defect and where the defect occurs and figure it out[5].

Mobile applications are usually developed in the emulator-based environment. The emulator-based performance testing on software development phase is rarely carried out. Because developing applications with emulator, results of testing on it differ from those on real target. Hence, most production-focused performance testing is performed on real target in the latter term of the development. In this case, finding faults is very difficult and it takes much time to fix them.

It is required to have a tool that support to generate test case and execute unit tests based on J2ME/WIPI mobile platform [6],[7]to carry out performance testing on software development phase. This tool measures the features of application program related to performance such as response time, execution time, resource consumption, reliability of operation, a limit of processing, efficiency and perform testing. Preferentially supports the time management of mobile applications, that is, performance characteristic for the process reaction speed. Since a mobile application is software that works according to events, it is important to have the event time measured and check the time limit condition because it measures the time between events by unit test and measures the time spent between the methods to improve the program performance by removing unnecessary codes and object creation and to detect a bottle-necked section by

ICUIMC-09, January 15-16, 2009, Suwon, S. Korea

picking up the method that is intensively used in a specific program. Although a mobile application is functionally perfect, if it does not guarantee real-time performance, it is useless. The issues related to timing, synchronization, and speed are important factors in mobile applications, therefore this will focus on time consumption such as Timing, and process reaction time for performance measurement factors to be considered in performance testing. For that, we will present the methods of performance testing in TDD that considers non-functional measurement factors by advancing performance testing that is mostly carried out at the end of development at the unit test level and introduce a testing tool that assists performance testing on software development phase.

This paper is organized as follows. In Section 2, TDD and Test-First Performance(TFP), which are the basis of methods of performance testing are introduced. MOPAD project in progress is also introduced. In Section 3, the methods of performance unit testing for mobile applications are explained. In Section 4, a performance unit testing tool is introduced. Finally, Section 5 concludes this paper with mentioning its conclusion and future work.

2. RELATED WORK

2.1 Test Driven Development(TDD)

Test-driven development(TDD) is a software development technique that uses short development iterations based on pre-written test cases that define desired improvements or new functions. It was introduced by Kent Beck and is also known as Test-First Development. TDD requires developers to generate automated unit tests that define code requirements before writing the code itself. The tests contain assertions that are either true or false. Running the tests rapidly confirms correct behavior as developers evolve and refactor the code. Such development process induces progressive growth of design and completion of progressive codes and results in optimized unit tests carried out[8].

It is a must to automate tests in TDD. Developers would not perform tests frequently enough and not make the development schedule if tests were not automated. Therefore, it is important to have the environment in which automated tests are frequently carried out. Test frames such as JUnit, cppUnit, NUnit, and PyUnit, which support the specified program language provide an efficient mechanism in which developers can apply effective unit tests that are test-based and automated to the general development process. JUnitPerf [9] and JMeter [10] are the typical test frameworks related to performance testing.

2.2 Test-First Performance(TFP)

Test-First Performance(TFP), a technique that combines TDD and performance testing, uses unit tests and JUnit test framework for performance testing. [11] Unlike TDD that uses JUnit assert methods to test the test target, TFP uses performance scenarios to generate performance log files. Also, developers and performance advisors use the log files to recognize performance problem and area.

TFP is carried out in 3-level processes: Test Design Level (Identify performance area, devise performance goal and design test cases), Critical Test Suite Execution Level (Generate log files and find faults) and Master Test Suite Execution (Find performance faults by performing all test cases).

2.3 MOPAD PROJECT

This Section describes the MOPAD (Mobile Platform based Application Developer) Project. The MOPAD project is a progressing project for the goal of developing the mobile application SW development environment technique. We are participating in the research for mobile performance testing method as a part of the project. In general, mobile applications are developed based on J2ME and WIPI mobile platforms. That is, it is downloaded and used in a mobile phone in VM (Virtual Machine) environment.

It is differentiated from general software in the following aspects: First, the life cycle is short. Mobile application SW is developed in a short development period to meet the competitive market's demand. They put more priority in meeting the market's demand than safety that requires a long-term design and tests and as a result, SW with low quality and reliability has been coming out in the market. Secondly, accurate tests are required for mobile application SW before installation because unlike PC, it is impossible to have continuous updates in mobile applications due to technical limitation and there are few additional updates once SW is installed.

We are participating in the project focusing on the development of the architecture-based mobile application SW development environment, which provides shorter design and implementation time than existing development methods and dynamically reconstructs the functions according to the change in performance environment while performing mobile applications when developing mobile applications with such characteristics. We are focusing on mobile application SW unit tests and in the process of researching the unit test support tools in the RAD-oriented mobile applications development frame work. Our methodology of performance unit testing is described in the next section.

3. PERFORMANANCE UNIT TESTING FOR MOBILE APPLICATIONS

In this section, we describe our methodology of the performance unit test method for mobile applications. The figure 1 shows the overview for method of performance unit test that defines and suggests formulated test levels and test environment based on the software development process.

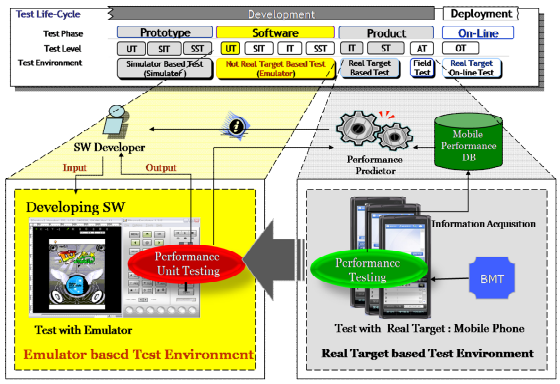

Figure 1. Approach to performance unit testing.

Test Phase is divided into test basics level, prototype test phase, software test phase, and production test phase. Test environment according to the test steps includes document based test environment, simulator based test environment, emulator based test environment, and real target based test environment.

It is emulator based test environment that this paper focuses on. Although developers generally develop mobile applications under the assumption that the test results in the emulator and real target are the same, actually most of the results performed in the emulator are different from the results performed on the real mobile phone. For this reason, there are many cases that applications tested in the emulator work with no problem when developing applications but they do not work properly or show different performance results after the applications are loaded on the real target, mobile phone.

Therefore, test results of performance unit testing are provided by collecting information with BMT (Bench mark Test) in real target based test environment and building Mobile Performance DB in order to perform reliable performance unit tests by providing information that can be expected in the emulator based test environment without loading and testing on a number of mobile phones. Detailed test steps and performance revision methods are described in the next section.

3.1 Test steps

Figure 2 shows the performance steps to carry out performance unit test.

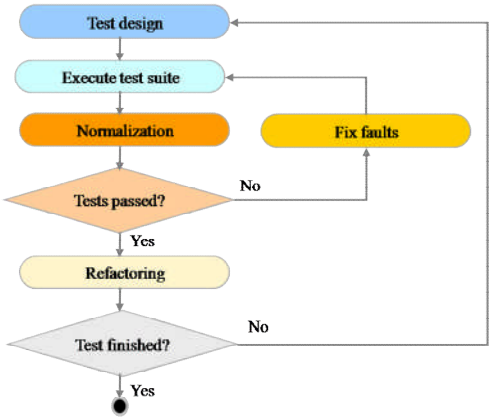

Figure2. Steps for performance unit testing.

As shown in Figure 2, design test by identifying methods and classes which are required for performance check based on the source codes. The unit of test is a method. Test cases and test codes are generated and performed by taking time limit as input from class information. That is, test is failed if there is no response in a given period of time after test loading.

For example, if the time limit condition specified in the requirement is "Max response time must be 50 ms or less when the button is clicked." Developers assign time limit for each method related to click-button functions and check if each method meets the requirement when creating methods. And then, perform normalization on performance time for reliability. Then check the test results: if it passed, end the test by optimizing the code by refactoring and if it failed, modify the code after figuring out which method, and which part of method causes the delay. And keep performing the test till the test result is a pass.

3.2 Normalization of performance

Perform the performance time revision work of emulator and mobile phone to obtain reliable performance results. The important work in Normalization of performance is to establish normalized mobile performance DB and to develop the revision algorithm that gives accurate prediction using DB.

3.2.1 Mobile Performance DB

Mobile performance DB is a core factor to perform the performance unit test. Figure 3 shows the method to establish mobile performance DB with a bench mark test program. DB is built by loading and performing a BMT program by API and organizing the result. The actual information of mobile phone other than performance results including the information such as CPU, Heap Memory Size, and Color Depth are to be used in the performance unit testing in the emulator.

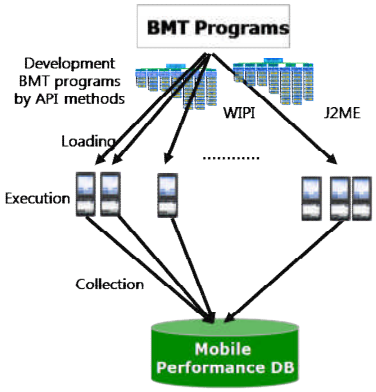

Figure 3. Building mobile performance DB

In the process of building mobile performance information DB, API and Method that are the test target are assigned first. We targeted J2ME and all API of WIPI, and performed the test by the following: File Test, Filesystem Test, Image Test, Display Test, Graphics Test, and Bytecode Test. We first ran the WIPI API test and the BMT program was generated by using WIPI API only. Now we have completed to build mobile performance DB for WIPI and are in the process of J2ME API Test.

3.2.2 Normalization algorithm

It takes the performance information input for various mobile devices when mobile performance DB is building and shows performance testing result for each mobile device in performance unit test in the emulator to allow us to predict the performance when the implementation target is loaded. When testing the unit performance time for a method A that must be carried out within 50 ms for example, the test is a pass in the emulator with performance time measured at 30 ms. But there is no way to find out the performance on each mobile phone unless we install the method A and test each of them. However, it is possible to predict the result with the revision method between mobile and emulator. If Phone1, Phone 2, and Phone 3 show 45 ms, 27 ms. and 60 ms respectively, we can predict that Phone 1 and Phone 2 will 'Maybe Passed' while Phone 3 will have 'Maybe Failed'.

4. PERFORMANCE UNIT TESTING TOOL FOR MOBILE APPLICATIONS

Tools which support writing performance unit tests and performing based on J2ME/WIPI mobile platform are needed to carry out performance test on software development stage. These frameworks measure features of application program related to performance such as response time, execution time, resource consumption, reliability of operation, a limit of processing, efficiency and perform the testing. Features related to time, this is, the process reaction speed per unit time are supported preferentially. Mobile program is the event-driven software, so measuring time according to events matters, and to confirm time limitations is a must. To find out the time cost of methods by measuring time according to events of each unit test helps to improve performance of program by eliminating unnecessary codes or objects generation to find methods called intensively, and to find the bottleneck. Therefore, we have developed the tool that supports the automation of unit test creation and performance with a research on the method to test the performance of mobile applications in the unit test environment.

4.1 Development environment

The mobile performance unit test tool is developed in the Eclipse IDE integrated development environment as shown in Table 1.

Table 1. Development Environment

Classification	Description	
OS	Windows XP	
Language	mobile java	
IDE (Integrated development environment)	JDK	JDK 1.5
	Eclipse SDK	eclipse-rcp-europa
	EclipseME	eclipseme1.7.7
	SUN WTK	Sun Java Wireless Toolkit 2.5.2

Although Eclipse itself does not have extra features to support mobile programming, once EclipseME plug-in is installed, Eclipse becomes one of the most suitable integrated development environment for mobile programs and allows to have the most optimized mobile development environment along with WTK that is provided by Sun. The mobile performance unit testing executioner is developed in the plug-in from of Eclipse and uses Wireless ToolKit 2.5.2 provided by Sun. The Sun Wireless Toolkit, a UEI emulator that supports the standard debugging interface (JVM Profiler Interface, JVMPI) supports the Sun J2ME Wireless Toolkits 1.0.3 and later versions.

The change to the UEI interface means that it is based on the selected emulator interface and used by different libraries to compile a project. When selecting J2ME JDK to support UEI, a target device such as Default Color Phone is determined in the JDK dialogue box selection tab. To select a target device, the library API is automatically installed in the JDK that is selected in the project. Sub part devices that share other same library API can be used within the application. In the project properties, a specific emulator device to test the project can be selected from the list of devices supported by the general library API related to the target device. In the run time properties dialogue box, the emulator device can be approached by selecting the run time settings for the project to test in the run page of the project properties dialogue box.

4.2 Architecture

Figure 4 shows the performance unit testing tool architecture. The tool consists of 4 modules.

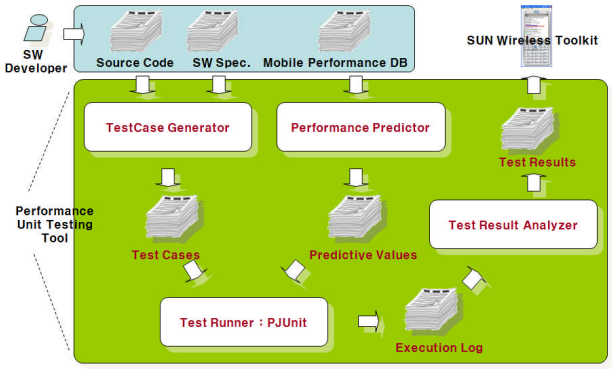

Figure 4. Performance unit testing Tool Architecture

4.2.1 TestCase Generator

The TestCase Generator module is an Eclipse plug-in and considered as a part of Java Project. It generates test case. It is implemented as the TestCase Wizard form. The generated test cases are as shown in Figure 5.

```
public class Demo1TestCase extends TestCase {

    private static final String TEST_CASE_NAME = "Demo1TestCase";
    private static final int TEST_METHOD_COUNT = 1;
    private static final int TEST_DEVICE_COUNT = 4;
    public Demo1TestCase(){
        super(TEST_CASE_NAME, TEST_METHOD_COUNT, TEST_DEVICE_COUNT);
    }
    public void test(int testNumber) throws AssertFailedException{
        switch(testNumber){
        case 0:
            testGetActiveJlet();
            break;
        }
    }
    public void setExpectedTime(){
        setExpectedTime(0, 100);
    }
    public void testGetActiveJlet() throws AssertFailedException{
        //TODO getActiveJlet
    }
    public void setDeviceList() {
        setDeviceList(0, "SPH-W240");
        setDeviceList(1, "IM-S220L");
    }
```

Figure 5. Example of Test Case

4.2.2 Test Runner : PJUnit

The test performance module allows to execute the test by running the generated test cases. Unit test engine which can be operated is needed to perform mobile application software unit test. JMUnit(Java Micro Unit) [12] is JUnit for mobile java, which operates on Java ME or J2ME. Two version of CLDC1.0, CLDC1.1 are released, JMUnit supports assert methods such as assertTrue(), assertFalse(), assertSame(), assertNotSame(), assertEquals(), assertNotEquals(), assertNul(), assertNotNull().Reflection is used to access constructors, fields, methods by loading other class dynamically within already loaded class at runtime. Mobile java cannot perform reflection and stack tracing, so mobile performance unit testing cannot print tests,

methods and classes to see whether the test fits time limitation and trace where the exception occurs. Hence, JMUnit is not a good choice to perform the test we want. Or, according to JMUnit properties, it cannot perform reflection and stack tracing and does not have UI to print out the information in detail, execution time measuring and performance unit testing of each unit method is impossible. For this, Unit Test Engine run in mobile environment like JMUnit is required, so this study developed PJUnit, unit test engine for java mobile performance.

The hierarchy structure of test executed through the test performance module is as shown in Figure 6.

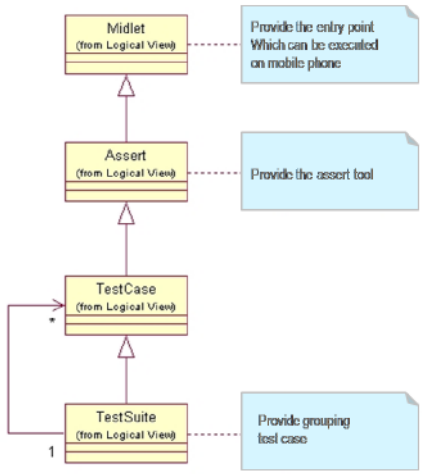

Figure 6. Test Case Hierarchy Structure

When performing the test, the TestContext module is implemented to preserve the test phrase information. The TestContext module contains testContext.open ("Test") at the beginning of the test function that adds a test under the current test phrase and declares the phrase that is currently testing and goes back to the parent phrase after ending the current phrase at the end. We made an alternative module to do stack tracking because we cannot use stack tracking.

4.2.3 Performance Predictor

Performance Predictor as a module that predicts and analyzes the phone test result with mobile performance DB input is implemented as a prototype in the mobile performance unit testing tool and is currently under development.

4.2.4 Test Result Analyzer

The test result analyzer module is developed as an Eclipse plug-in and is implemented to view various results by selecting a pass or fail module from the list of test result in the emulator. The contents of visualization are the following.

- *Phone* : a list of cell phones to be tested
- *Method :* a method to be tested
- *Class :* a class which contains test method
- *Expected Time:* the time limit conditions
- *Actual Time:* the execution time on emulators
- *Estimated Time :* the predicted execution time on phones
- *Determination of Pass/Fail :* to determine whether testing results pass, maybe passed, fail, maybe failed or faults

Test results of performance unit testing are as shown in Figure 7. The difference between the predicted time and actual time is shown in graph and the test prediction results by phone list are visualized.

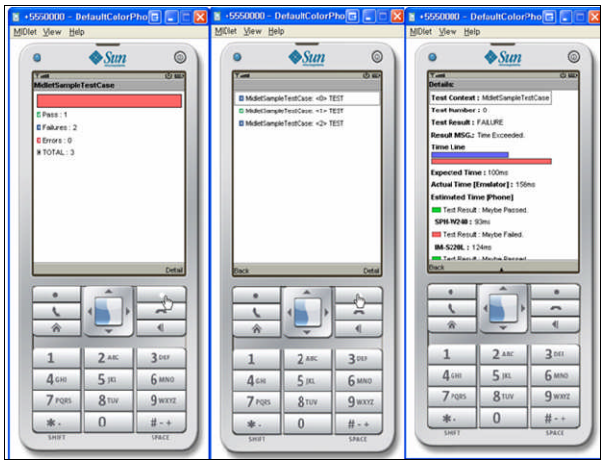

Figure 7. Test Results Screenshot

5. CONCLUSIONS AND FUTURE WORK

Due to the intense development schedule and lack of mobile performance unit testing, product-oriented performance testing that is mostly done in the end of the development involves problems such as identifying the cause of faults and tracking down and modifying the faults when faults occur. The mobile performance unit testing method that considers non-functional performance characteristic of software helps to improve the quality and reliability of software and development productivity by advancing performance testing in the development level in the process of software development. Therefore, it is important to presents mobile performance testing method and develop the mobile performance unit testing tool that supports performance unit testing generation and automation of tests.

The goal of this study is to develop a mobile performance unit testing tool that not only supports the functional testing in the development process of unit testing environment but also supports performance unit testing generation and performance automation in order to improve the quality and reliability of mobile applications.

In consideration of the above points, we present the performance testing method for mobile applications and developed the mobile performance unit testing tool. The mobile performance unit testing tool that we develop is implemented as an Eclipse plug-in and consists of test case generation module, test performance module, phone testing result prediction module, test result module, analysis module. It means a lot that performance unit testing generation and performance are automated. This research can be used in building appropriate development environment by customizing mobile performance unit test generator which is developed for the development environment of the specific organization itself, which develops mobile applications. With the development of mobile performance unit testing generator, the time and cost required for test performance have been reduce and it has become possible to have the base technology of mobile

applications quality and performance testing by helping to detect software faults. In the future research, it is required to expand the research on the performance unit testing method, which reflects the characteristics of mobile applications. Also, more concrete research is required in relation to the test prediction module which was developed as a prototype. We can promote a fast development and quality improvement having reliable and useful performance information provided in the development level.

6. ACKNOWLEDGMENTS

This work was supported by the IT R&D program of MKE/IITA. [2007-S032-02, Development of mobile application software development environment technology supporting multi-platform]. Thanks to XCE Co.,Ltd. for building mobile performance DB.

7. REFERENCES

[1] Kent Beck, "Test-Driven Development By Example", 2003

[2] http://www.junit.org

[3] Hamlet, R.,'Unit testing for software assurance', Computer Assurance, 1989.

[4] M.Glinz ., "On Non-Functional Requirements."Pro. 15th IEEE Int'1 Requirements Eng. Conf. (RE'07), IEEE CS Press, pp.21-26.

[5] E.J. Weyuker and F.I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study," IEEE Trans. Software Eng., vol. 26, no. 12, Dec. 2000, pp. 1147–115

[6] J2ME : http://java.sun.com/j2me/

[7] WIPI : http://www.wipi.or.kr/

[8] L.E. Williams, M. Maximilien, and M.A. Vouk, "Test-Driven Development as a Defect Reduction Practice," Proc. 14th Int'l Symp. Software Reliability Eng., IEEE CS Press, 2003, pp. 34–35.

[9] JUnitPerf: http://www.clarkware.com/software/JUnitPerf.htm

[10] JMeter: http://www.jmeter.org

[11] Johnson, Michael J.; Ho, Chih-Wei; Maximilien, E. Michael; Williams, Laurie, "Incorporating Performance Testing in Test-Driven Development," Software, IEEE Volume 24, Issue 3, May-June 2007, pp. 67-73.

[12] JMJUnit: http://sourceforge.net/projects/jmunit/