Mobile Apps Testing

Version 0.1, Draft

INFO

COPYRIGHT

About This Document

This document is intended for MIDP developers to perform effective testing. This document describes testing basics, activities, classifications, checklists etc.

Scope

This document is about mobile applications testing. It assumes good knowledge of Java programming language, Unit testing concepts and therefore explaining the Java technology and Java ME is out of the scope in this documentation.

To know more about Java ME basics and Java programming language, refer to the Knowledge Base under Samsung Mobile Innovator (SMI).

http://innovator.samsungmobile.com/platform.main.do?platformId=3

Table of Contents

Introduction

Mobile applications like other types of software must be tested to ensure functionality and usability under all working conditions. Testing is more important for mobile apps because these are developed in high end environment & actually runs under strict limitations.

Testing Overview

Software testing is an activity to find errors in software system and ensures quality of the deliverables after error corrections.

Testing is an iterative process and usually starts at the beginning of the project. This involves multiple phases such as Test planning, test schemes, writing test cases, execution & reports. Different methodologies are adopted based on size, scope & nature of the project.

Testing Activities

Whole testing activity is broadly split into two classes: **Black box testing** & **White box testing**.

Black Box Testing: This is the functional testing. System as a whole is treated as a black box, and testers verify that it supports all the features identified (often as use cases) in the requirements documents. This includes:

- **Unit Testing:** In this testing activity, components are tested separately. Because some objects may depend on other objects that are not yet available, you may need to develop test drivers and test stubs. A test driver simulates the part of the system that calls the component under test. A test stub simulates a component called by the tested component.
- **Integration Testing:** In this activity, objects are integrated in increasingly large and complex subsystems. This is an incremental testing process.

- **System Testing:** In this activity, the system is tested as a whole. Testers employ various techniques at this stage, including functional testing (testing actual behavior against documented requirements), performance testing (testing nonfunctional requirements), and acceptance and installation testing (testing against the project agreement).

White Box Testing: This is the structural testing. Focus is to test the code which produces behavior. This includes:

- **Control flow Testing:** This requires checking the control flow of the program, basically steps are checked.
- **Data flow Testing:** This requires checking the data or the state at different steps.
- **Branch Testing:** This requires validating all the code braches & ensuring no branching leads to abnormal behavior.
- **Path Testing:** This requires checking of all possible paths in the program to ensure the code coverage.

Challenges in Testing Mobile Apps

There are wide varieties of mobiles available in market running on different implementation of CLDC & MIDP. Varying display sizes add to the complexity of testing process. In addition, some vendors provide proprietary API extensions. As an example, some Java ME vendors may support only the HTTP protocol, which the MIDP 1.0 specification requires, while others support TCP sockets and UDP datagrams, which are optional.

Testing Mobile Apps

The testing activities described above are applicable to testing mobile Java applications. In other words, you perform unit or class testing, then you integrate components and test them together, and eventually you test the whole system. Below are the certain guidelines to test mobile applications.

Validating the implementation

Ensuring that the application does what it's supposed to is an iterative process that you must go through during the implementation phase of the project. Part of the validation process can be done in an emulation environment such as the SDK toolkits, which provides several phone skins and standard input mechanisms. The toolkit's emulation environment does not support all devices and platform extensions, but it allows you to make sure that the application looks appealing and offers a user-friendly interface on a wide range of devices. Once the application has been tested on an emulator, you can move on to the next step and test it on a real device, and in a live network.

Usability Testing

Usability testing (or GUI navigation) focuses is on the external interface, relationships among the screens of the application & the user friendliness.

You need to test the GUI navigation of the entire system, making notes about usability along the way. If, for example, the user must traverse several screens to perform a function that's likely to be very popular, you may wish to consider moving that particular function up the screen layers.

Points to consider during usability testing:

- Is the navigation depth (the number of screens the user must go through) appropriate for each particular function?
- Can screens of all supported devices display the content without truncating it?
- Check for Internationalization.

Network Performance Testing

The goal of the next type of testing is to verify that the application performs well in the hardest of conditions (for example, when the battery is low or the phone is passing through a tunnel). Testing performance in an emulated wireless network is very important. The problem with testing in a live wireless network is that so many factors affect the performance of the network itself that you can't repeat the exact test scenarios. In an emulated network environment, it is easy to record the result of a test

and repeat it later, after you have modified the application, to verify that the performance of the application has improved.

Server Side Testing

It is very likely that your applications will communicate with server-side applications. If your application communicates with servers you control, you have a free hand to test both ends of the application. If it communicates with servers beyond your control, you just need to find the prerequisites of use and make the best of them. You can test server-side applications that communicate over HTTP connections using tools available in market.

Automating Unit Testing

There are many tools available in market to do unit testing in Java. Most popular is the JUnit framework but this does not hold good for Java ME applications. Extensions of JUnit are available for Java ME – JMUnit & J2MEUnit. User can generate the test cases & then execute the test suite. These come with a very good GUI which tells the statistics of execution.

Debugging Information

Adding debugging information in your code is very important. You can display trace points, values of variables, and other information during testing and debugging. One way to minimize the tedium of writing **System.out.println()** calls is to write a utility method such as the following:

```
public void sop(String s) {
    System.out.println("DEBUG: "+s);
}
```

You can easily use the sop() method to display debugging information, then later remove the calls from production code.

Testing Checklists

This section provides checklists you will find useful when testing your application, in both emulation and live environments.

Navigation Checklist

- **Application name:** Make sure your application displays a name in the title bar.
- **Keep the user informed:** If your application doesn't start up within a few seconds, it should alert the user. For large applications, it is a good idea to have a progress bar.
- **Readable text:** Ensure that all kinds of content are readable on both grayscale and color devices. Also make sure the text doesn't contain any misspelled words.
- **Repainting screens:** Verify that screens are properly painted and that the application doesn't cause unnecessary screen repaints.
- **Soft buttons:** Verify that the functionality of soft buttons is consistent throughout the application. Verify that the whole layout of screens and buttons is consistent.
- **Screen Navigation:** Verify that the most commonly used screens are easily accessible.
- **Portability:** Verify that the application will have the same friendly user interface on all devices it is likely to be deployed on.

Network Checklist

- **Sending/Receiving data:** For network-aware applications, verify that the application sends and receives data properly.
- **Name resolution:** Ensure that the application resolves IP addresses correctly, and sends and receives data properly.
- **Sensitive Data:** When transmitting sensitive data over the network, verify that the data is being masked or encrypted. Use the SSL protocol.
- **Error handling:** Make sure that error messages concerning network error conditions (such as no network coverage) are displayed properly, and that when an error message box is dismissed the application regains control.
- **Interruptions:** Verify that, when the device receives system alerts, SMS messages, and so on while the application is running, messages are properly

displayed. Also make sure that when the message box is dismissed the application continues to function properly.

Other Important Issues

- **Successful startup and exit:** Verify that your application starts up properly and its entry point is consistent. Also make sure that the application exits properly.
- **Classes outside the MIDP and CLDC specifications:** Unless you are willing to sacrifice portability and, in some environments, certification, ensure that the application does not use classes not included in the MIDP and CLDC specifications.
- **User manual:** Verify that all product documentation is accurate, and consistent with the software's actual behavior.