Wireless Application Testing

A Whitepaper

Copyright © 2006. AppLabs Technologies Pvt. Ltd. All Rights Reserved. AppLabs is a registered trademark of AppLabs, Inc. All other trademarks are owned by the respective owners.

Table of Contents

1. Introduction 3

2. Types of Testing 3

2.1. END-USER CONSOLE/DEVICE APPLICATION TESTING 3
2.1.1. Functional Testing 3
2.1.2. Compatibility Testing 4
2.2. BACK-END TESTING 5
2.2.1. Functional Testing 5
2.2.2. Performance Testing 6
2.2.3. White Box Testing 7
2.2.4. Automated Testing 8

3. Testing Process 8

3.1. TEST PLAN CREATION 8
3.2. TEST EXECUTION 10

4. Brew Certification 11

4.1. PROCESS 11
4.2. EXPERIENCE 11
4.3. SAMPLE CERTIFICATION TEST CASES 11

5. Business Case Study – Similar Experiences 16

1. Introduction

Wireless applications are booming in the modern PDA and cell phone market. Standalone applications and value-added extensions of existing applications are both excellent ways to attract customers and increase the value of the hardware. Many companies use these applications as a way to retain customers and improve customer experience and satisfaction. Testing these applications has continued to be a road block for many developers. Difficulty in securing all platforms, and testing the many disparate configurations is cumbersome and time-consuming. AppLabs now offers a solution to your wireless testing needs.

AppLabs's CMMI Level 5 testing process brings reliability, quality, and value to the wireless marketplace. AppLabs Technologies provides offshore and on-site quality assurance services. We have a dedicated staff of QA professionals to create and execute test plans, perform stress, load, & capacity testing, and report defects.

2. Types of Testing

2.1. End-User console/device application testing

2.1.1. Functional Testing

A set of functional tests will be developed based on client documentation. Functional tests verify that the application performs the tasks correctly and accurately as explained in the client documentation.

Dedicated QA Engineers will write test cases for each component of functionality and execute those tests. Tests will be clearly defined prior to execution, with an emphasis on actions and expected corresponding behaviors.

2.1.2. Compatibility Testing

Various mobile phones with varying screen sizes, dimensions, and hardware/software technologies

The wireless market has a myriad of devices available. The differences in each device can be extensive. Each device can have a different operating system, support different technologies, and have different hardware. Presenting a similar end-user experience on each device is challenging.

AppLabs can perform compatibility tests across a multitude of different platforms. Platform requirements are set by the client. AppLabs will develop compatibility test cases for each device, using the functional testing requirements as the baseline for acceptance criteria.

Generally, each platform requires a separate version of the application, and in some cases, the same platform may have different versions for each different device. This often happens when developing applications for cell phones. Different screen sizes/resolutions and differences in other physical hardware often make it necessary to have different versions for each platform.

Common issues encountered in compatibility testing on wireless devices:

Screen Size

The size of the screen on mobile devices is typically small. Presenting a usable interface on it can be challenging, especially when adapting an existing product for use with mobile technology. Additionally, screen sizes vary on each device, even on the same technology platforms. AppLabs can test for usability, as defined by the client, on each device. To overcome issues with screen size, applications are often tailored to each device

Other Hardware limitations

At a minimum, different devices will have differences in internal hardware. These limitations can include memory size, processor speed, and button/input differences. AppLabs is experienced in testing mobile applications on a variety of hardware. End-user performance can be affected by the speed at which the application runs. Applications that require animation or other continuous quick-response, such as games, can be tested on each platform to ensure that the application meets the minimum functional and usable requirements set by the client.

Software limitations

Testing an application on platforms with different platforms of embedded software often changes how a program is designed. For example, some mobile phones are enabled with HTTP web browsers, while others only have WAP. If the application under test communicates with a web server (HTTP), there would need to be a layer that could determine if the end-user were on a WAP or HTTP enabled phone. Additionally, another layer would be needed to translate the HTTP data into usable WAP data (and back the other way) for WAP-only phones.

Even in stand-alone applications, the underlying software technology may require different software design for the client's application. For example, some phones support BREW (Binary Runtime Environment for Wireless) while others support J2ME. Testing in each of these environments may be different depending on the client's application.

2.2. Back-End testing

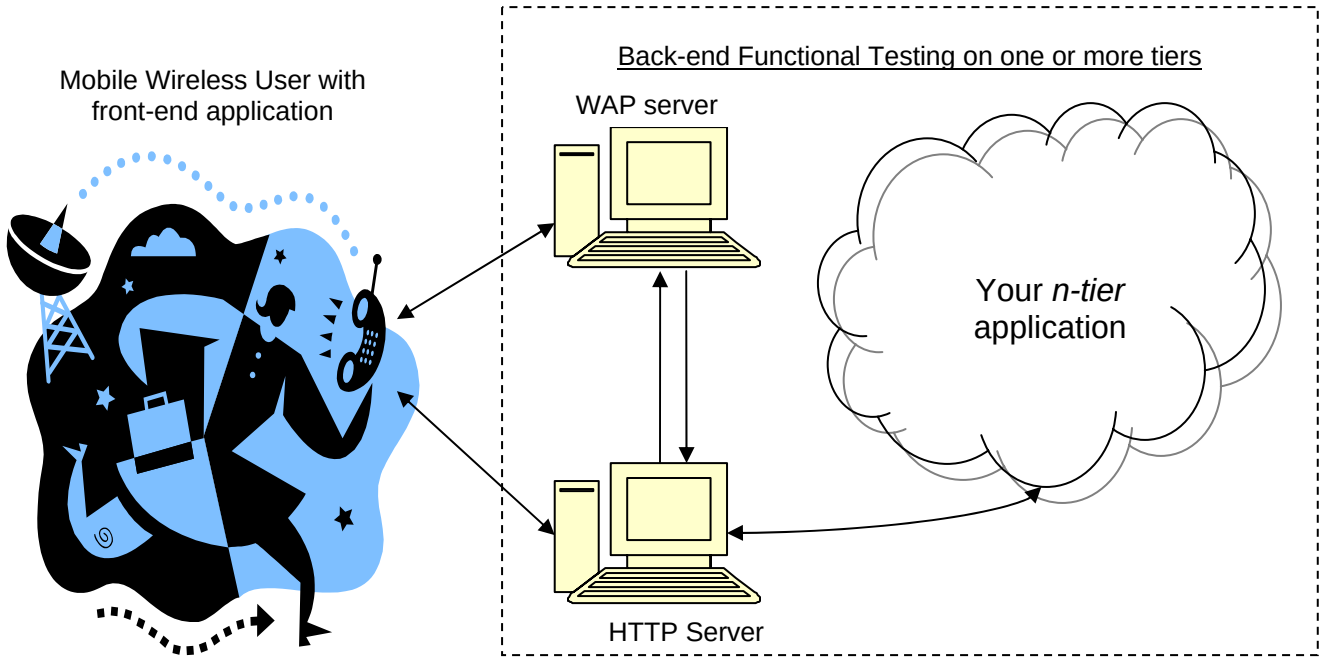
While some wireless applications are standalone products, many include connectivity to a larger system. Anything that doesn't reside on the wireless device is considered to be part of the back-end. For these applications, the back-end is often the most critical component(s) of the system. AppLabs offers a full spectrum of testing services on these systems.

2.2.1. Functional Testing

On the back-end functional testing can be performed on any user or administrative tools. Applications that serve content to mobile users typically have a data-entry system that is a good candidate for functional testing.

As with all of our testing processes, AppLabs will design and implement a test plan in accordance with the requirements set forth by the customer.

Example of web-based wireless application using different mobile technologies (WAP and HTTP enable devices)



2.2.2. Performance Testing

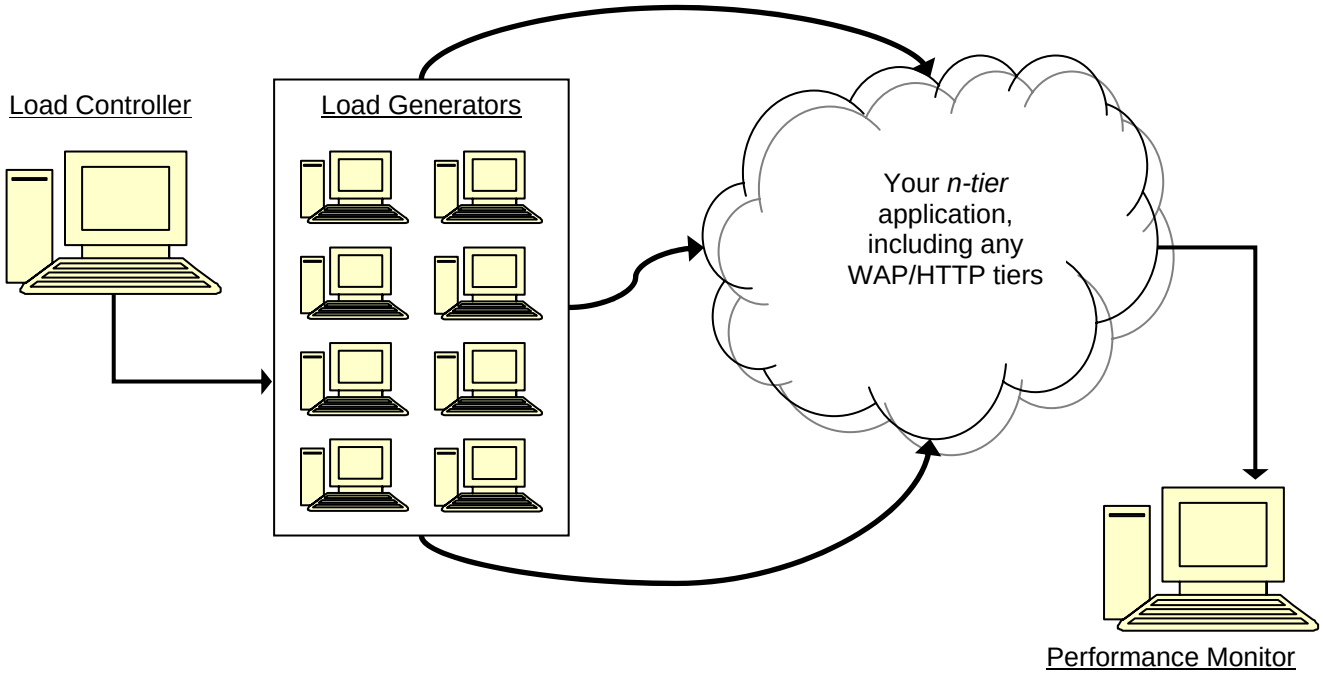
Performance testing is critical to the deployment and maintenance of the back-end of wireless applications. Load testing is done by creating a set of virtual users that simulate a load on a real application. It increases confidence and knowledge of response times, performance, reliability and scalability. Without conducting proper load tests, it is difficult or impossible to know the number of users an application can effectively support. Load testing increases up-time, customer satisfaction, and decreases both real and potential costs of hardware, software, and development resources. Load testing can identify bottlenecks and defects that can't be found with conventional methods of testing.

The back-end of a wireless application may include web servers, SQL servers, translation modules, and any number of tiers in the client's n-tier architecture. AppLabs can test for and identify critical points in the system, and recommend solutions for existing bottlenecks, and potential ones found during testing. Three types of performance tests are typically executed – load, stress, and capacity tests.

AppLabs uses industry standard load generation tools, and when needed, AppLabs can create custom applications to generate load on specific systems. Mercury Interactive's LoadRunner® or Radview's Webload® is recommended in many scenarios; however, some applications require the use of custom developed tools. AppLabs has developed AppMeter, a Load Testing tool based on the open source Java tool, JMeter. AppLabs can

also develop a custom tool solution known as a “Test Harness” if it is required by the application under test.

Performance Testing example of back-end wireless application using simulated users via load generators



For more details on performance testing, please refer to the “Performance Testing Whitepaper” available at http://www.applabs.com

2.2.3. White Box Testing

White box testing can discover back-end problems more accurately and efficiently than can be done directly using the supported wireless devices. Some clients may require all or some of the types of white box testing found below. Decisions on what types of back-end testing to undergo often depend on the complexity of the application under test, deadline considerations, and any other client needs. It is minimally recommended that all wireless applications that have a supporting back-end undergo system testing. A detailed description of general white box testing services and procedures is available at http://www.applabs.com

2.2.3.1. Unit Testing

This phase of testing will uncover problems found in specific components of the back-end application. Each back-end tier will be comprised of modules, and each module will be comprised of one or more components. Each of these components has a specific task to perform. Testing each component is possible by writing small test applications that can be used to access the component. Test data is generated to adequately cover various conditional branches within the component’s source code.

2.2.3.2. Integration Testing

The components that make up each module, or tier, of the application can then be integrated and tested to verify that each component is communicating correctly with proper requests and responses, as designed in the client's development architecture. This phase will uncover problems found in the way that components work together.

2.2.3.3. System & Sub-System Testing

Each tier is then combined into the overall system. AppLabs can perform system tests to ensure that each tier is properly integrated into the larger system. These system tests can encompass the entire system, or sub-systems as needed. This phase will uncover problems in the way each tier works together, and how that system functions holistically, without the integration of the wireless devices. This is important so as to prevent against falsely identifying issues that appear on some devices as being device-dependent problems.

For wireless applications, tests can be run to simulate the requests that would be expected from the wide range of devices on which the application under test is designed to work. In some cases it will be necessary to use the actual devices when simulation is not possible. This final step of white box testing will uncover any specific problems related to each device designed to work with the system.

2.2.4. Automated Testing

Automated testing is currently not possible on wireless devices. However, on stable back-end support systems, automation is sometimes a viable option for regression testing. AppLabs has the capacity to devote resources dedicated to developing automated test scripts. The areas of the application where automated testing is applied will be identified by AppLabs and the client during the scoping phase. Typically, areas which remain relatively constant and that are considered reasonably stable are considered strong candidates for automated testing. Well-documented areas are also a major additional "plus" when considering automation.

3. Testing Process

AppLabs conforms to the CMMI Level 5 process and documentation standards for all its testing and development work. The following is a brief description of the process used in all testing projects. More information can be found on our website and in the QA Process white paper available at www.applabs.com.

3.1. Test Plan Creation

AppLabs takes considerable care in developing and managing test plans and test cases. Developing accurate, clear and concise test cases is critical to the success of black box, functional testing.

Client Documentation

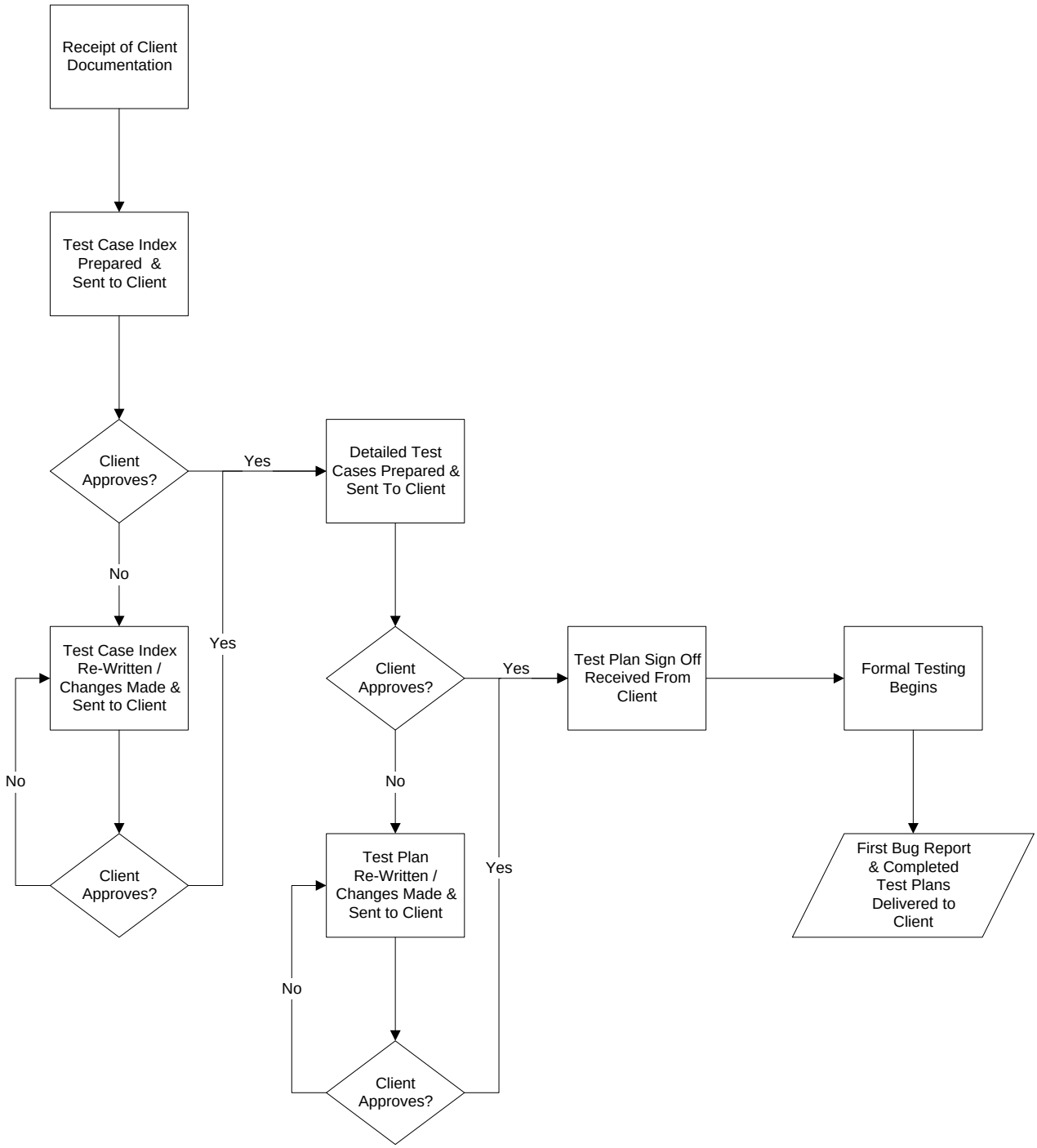
The first step in developing test cases is receiving and reviewing the Functional Requirements Documents provided by the client. The FRD review process includes the following:

- Unit Level Functionality dissected
- Understanding of each page's/component's rendering logic
- Creation of Application flow
- Integration Scenarios

Test Plan Sign-off Process

AppLabs follows the below diagram in order to gain client approval of the test cases developed by AppLabs. The basic tenets of the sign-off process include:

- Developing Test Plan Indexes
- Developing Detailed Test Scenarios (test cases)
- Review and Adjustment cycle with client
- Test Plans accepted



3.2. Test Execution

Tests are executed on a daily basis. Issues are handled through use of defect management software. As problems are found, each problem is explicitly documented with a full description, steps to reproduce, and reference to the appropriate test case that was used. Each day AppLabs will provide a report of bugs found and issues or questions that need to be addressed.

AppLabs places a strong emphasis on reporting identified issues correctly and thoroughly. In order to provide our clients with the best possible information concerning an issue, we take the following steps while reporting quality assurance issues.

- Identify: Following Test cases & logical thinking
  - Test cases provide a *basis* for testing
  - Other scenarios should be run as application grows; these should be added to the test cases
- Investigate: After the initial bug is found
  - Investigate opportunities to eliminate variables
  - Identify associated pages or areas where the immediate bug shows impact
  - Perform 'gray' box investigating by accessing the database or other back-end system data, if applicable
  - Rework the bug until its impact is fully understood.
- Reporting: AppLabs QA engineers follow a strict reporting process. This includes all relevant information including type and severity of the bug, the platform(s) it occurs on, and a detailed description including steps needed to reproduce the problem.

4. Brew Certification

4.1. Process

TRUE BREW certification is a rigorous process performed by NSTL. An application must be approved on each device it is compatible with. BREW is the only major mobile device API that requires approval by a centralized approving body. The comprehensive TRUE BREW certification includes the following testing categories:

- Application packaging/presentation
- BREW compatibility
- Application core functionality
- Application primary and secondary functions
- Core mobile device functionality
- Live network with core mobile device functionality
- Stability
- QUALCOMM user interface requirements
- Interoperability with required accessories

4.2. Experience

AppLabs provides complete end-to-end testing for TRUE BREW certification. We have tested dozens of applications that have passed this process and made their way onto the wireless market through BREW's distribution channels. AppLabs can ensure that your application meets all TRUE BREW requirements.

Full TRUE BREW testing requirements can be obtained from Qualcomm and found at http://brew.qualcomm.com/brew_bnry/pdf/documentation/80-D4187-1_E.pdf

4.3. Sample Certification Test Cases

Below are sample test cases that are found in TRUE BREW certification process. AppLabs can conduct all of these tests and more to ensure a smooth certification process when the application is submitted to NSTL.

3.3.5 Automatic data storage

Requirement:	An application that automatically saves data is able to retrieve the saved data.			
Assumptions:	The AUT is already loaded on the mobile device.			
Maps to application specification:	N/A			
Step	**Action**	**Expected results/detailed requirements**	**Status**	**Comment**
1.	Start the AUT and run it to generate data that is automatically saved. Ensure that the data is saved.	N/A		
2.	If the data can be retrieved at this time, review the saved data.	• All data is intact. • No adverse affect on AUT's functionality.		
3.	Exit the AUT, restart it, and review the saved data.	• All data is intact. • No adverse affect on AUT's functionality.		

3.3.8 Continual keypad entry

Requirement:	An application that accepts keypad entry is able to process a rapid succession of keypad input (tight event loop). A data call must be able to handle continual keypad entry without causing an error.			
Assumptions:	The AUT is already loaded on the mobile device. Access to the external server is provided, and the server is the production system (not a test system).			
Maps to application specification:	6.2.5, 6.2.15			
Step	**Action**	**Expected results/detailed requirements**	**Status**	**Comment**
1.	Go to every screen in the AUT that accepts keypad entry. Supply continual user input for at least 90 seconds.	• AUT handles all keypad input by either correctly processing it or ignoring it. • No adverse affect on AUT's functionality.		
2.	Go to random screens that allow a data call and initiate a data call. After the data call is successful, supply continual user input for at least 30 seconds. NOTE: This step does not apply to Tier 1 applications.	• Data connection remains intact. • No adverse affect on AUT's functionality.		

3.5.1 Wireless service availability

Requirement:	An application is able to start and perform some basic functionality regardless of the type of wireless service available to the mobile device.			
Assumptions:	The AUT is already loaded on the mobile device. The AUT is running during each service scenario.			
Maps to application specification:	N/A			
Step	**Action**	**Expected results/detailed requirements**	**Status**	**Comment**
1.	Exit the AUT. Force the mobile device into having no wireless service. Start the AUT and perform some basic functionality.	• AUT starts without error. • AUT can perform functions that are not dependent on having a signal.		
2.	Exit the AUT. Force the mobile device into digital mode wireless service. Start the AUT and perform some basic functionality.	• AUT starts without error. • AUT can perform functions that are not dependent on having a digital signal.		

3.5.2 Abnormal powerdowns

Requirement:	An application starts and performs some basic functionality after the mobile device has performed an abnormal powerdown.			
Assumptions:	The AUT is already loaded on the mobile device.			
Maps to application specification:	N/A			
Step	**Action**	**Expected results/detailed requirements**	**Status**	**Comment**
1.	While the AUT is running, normally exit the AUT using all available methods.	N/A		
2.	Restart the AUT and perform some basic functionality.	• AUT starts without error. • No adverse affect on AUT's functionality.		
3.	Repeat steps 1 and 2 for each method available to abnormally exit the AUT.	• As stated above.		

3.5.3 Alerts

Requirement:	An application handles all of the alerts that the mobile device can generate.
Assumptions:	The AUT is already loaded on the mobile device. The device data sheet documents all alerts available on the mobile device and the type of notification provided to the user. The device data sheet documents normal and low battery levels.
Maps to application specification:	6.2.12

Step	Action	Expected results/detailed requirements	Status	Comment
1.	Start the AUT. From multiple unique screens and events, receive each type of alert, including but not limited to: • Low battery. • One minute connection. • Low service. • Loss of service. • Power save. • Privacy. • Key guard. • Roaming. • SMS message, which needs to be a combination of the following: – Short (less than 50 characters). – Long (greater than 100 characters). Types of screens or events should include the following: • In a voice or data call. • Progress monitor displayed. • Scroll bars displayed • Help screen • Data entry screen • Sounds emitted by the AUT (MIDI or other)	• No adverse affect on AUT's functionality. NOTE: When a TRUE BREW test lab performs the low battery test, a test fixture is used to put the mobile device into a low battery condition.		
2.	For each of the alerts performed in step 1: after causing the alert, receive an incoming voice call.	• No adverse affect on AUT's functionality.		
3.	For each of the alerts performed in step 1: after causing the alert, receive an incoming SMS message.	• No adverse affect on AUT's functionality.		
4.	Repeat step 1 followed by step 2 for each alert.	• As stated above.		
5.	Repeat step 1 followed by step 3 for each alert.	• As stated above.		

3.7.1 Duration

Requirement:	An application is able to run over a period of time. The amount of free RAM before and after running the application remains is identical.
Assumptions:	The AUT is already loaded on the mobile device. The mobile device is cabled to work with The Grinder. The application developer knows how to configure The Grinder for the following settings: changing the time and generating events. The mobile device is running BREW version 1.0.x.x.
Maps to application specification:	N/A

Step	Action	Expected results/detailed requirements	Status	Comment
1.	Set up The Grinder to run over a period of time (2-3 hours, approximately 4,000 events) and generate a high number of events. This test should be executed on the mobile device.	N/A		
2.	Start the BREW Emulator, position the cursor on the AUT (do not start it yet), and note the amount of free memory displayed at the bottom of the screen.	N/A		
3.	Run The Grinder. Review the logs and state of the AUT.	• AUT does crash. • No exceptions logged. • Only expected changes to file system. • No abnormal file size changes. • Only expected outgoing data calls occur (BREW version 1.1.x.x). • Only expected outgoing voice calls occur (BREW version 1.1.x.x).		
4.	While the BREW Emulator is still running, position the cursor on the AUT and note the amount of free memory displayed at the bottom of the screen.	• This number should be exactly the same as the value noted in step 2.		
5.	Set up The Grinder to run over a period of time (10-12 hours) and generate a high number of events. This test should be executed on a PC.	N/A		
6.	Start the BREW Emulator, position the cursor on the AUT (do not start it yet), and note the amount of free memory displayed at the bottom of the screen.	N/A		
7.	Run The Grinder. Review the logs and state of the AUT.	• AUT does crash. • No exceptions logged. • Only expected changes to file system. • No abnormal file size changes. • Only expected outgoing data calls occur (BREW version 1.1.x.x). • Only expected outgoing voice calls occur (BREW version 1.1.x.x).		
8.	While the BREW Emulator is still running, position the cursor on the AUT and note the amount of free memory displayed at the bottom of the screen.	• This number should be exactly the same as the value noted in step 6.		

5. Business Case Study – Similar Experiences

Lotus Challenge – Mobile Racing Game

Overview

Lotus Challenge is a single-player race-car game developed using the BREW API. Users are able to download the BREW application to their mobile phones and play a demo and/or purchase a full version. The game was a basic two-dimensional race with hidden/unlockable cars and tracks/courses.

Project Description

AppLabs QA team was required to perform functionality and compatibility testing across over a dozen different mobile devices. In parallel, TRUE BREW certification pre-testing was done. Testing the application was critical to a timely release and successful certification. Each device had a separate build and was tested independently.

Methodology/Approach

In early testing, AppLabs utilized Win32 compatible device emulators to test for functionality and initial compatibility problems. Testing tools (e.g., BREW Logger, BREW Grinder) provided as part of the BREW SDK were used to test for memory and performance issues As testing and development progressed, the builds were installed on each phone using the BREW AppLoader and tested independently. In addition to functional and compatibility testing on the actual devices, formal pre-certification testing was conducted according to the comprehensive test cases set forth by BREW.

Results

All bugs were found and fixed that were necessary for TRUE BREW certification. Over a dozen different device-builds were submitted and passed TRUE BREW certification as a result of the testing conducted by AppLabs.

URL: http://www.kujuwireless.com/games_lotus.htm