CUSTOMIZED POWER EFFICIENT ANDROID ROM

Testing and implementation document

Submitted by,

Ajith K. M	-	ETAKECS004
Anjana Sasikumar	-	ETAKECS007
K. Haridas	-	ETAKECS026

Guided by,

Prof. Bisna N.D

7th April 2013

Purpose

This document lists the various software testing methods and the types of testing applied on our Power Efficient Android ROM.

Advantages and limitations

- Advantages

1. Open-source software
2. Uses linux kernel
3. Android version 2.3 Ginger bread
4. Low cost of implementation

- Disadvantages

1. Warranty goes void
2. Device specific roms are required
3. System performance varies with the configuration of devices

Future extensions if possible

- Adding support to additional languages,specifically Indian Languages.
- Increased power efficiency.
- Adding interactive user interface for the installation of applications at the time of ROM installation.
- Addition of more CPU governors.

Testing methods

There are a number of software testing methods:-

1. Unit testing

Unit testing focuses verification effort on the smallest unit of the software design, the module. This is known as module testing. Since the proposed system has modules, the testing is individually performed on each module.

2. Integration testing

Data can be tested across an interface. One module can have adverse effect on another sub-function. When combined it may not produce the desired function. Integration testing is a systematic technique for constructing the program structure while at the same time conducting test to uncover errors associated within the interface.

3. Validation testing

Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirements.

4. Output testing

After performing the validation testing, next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specific format. The output generated or displayed by the system under consideration is tested by asking the users about the format required by them.

5. Acceptance testing

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required.

Types of testing done:

- White-box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, eg. In-Circuit Testing (ICT).

While white-box testing can be applied at the unit. Integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include:

- API testing (Application Programming Interface) - Testing of the application using public and private APIs
- Code coverage - Creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods - intentionally introducing faults to gauge the efficacy of testing strategies
- Mutation testing methods
- Static testing methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested. Code coverage as a software metric can be reported as a percentage for:

- *Function coverage*, which reports on functions executed
- *Statement coverage*, which reports on the number of lines executed to complete the test 100% statement coverage ensures that all code paths, or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

- Black-box testing

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight." Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case, or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

Table 1:Black box testing

Sl No.	**Modules**	**Expected output**	**Whether obtained the expected output or not**
1.	The ROM is Switched on	Boots into the lockscreen menu	Yes
2.	Brightness Control Starts the app and presses the plus or minus button	Brightness increases or decreases according to the button pressed.	Yes
3.	User checks the cpu stats to see the effects of the rom.	User sees that CPU is running with a new frequency(power saving).	Yes
4.	Uses the phone for more than one and half days without charging.	Obtains better battery life than previously obtained.	Yes

Table 2:White box testing

Sl No.	**Modules**	**Function**	**Expected output**	**Whether obtained the expected output or not**

1.	Kernel Optimization	ro.ril.disable.power.collapse=1 pm.sleep_mode =1
		
		wifi.supplicant_scan_interval=220
		
		windowsmgr.max_events_per_sec=150 mot.proximity.delay = 150
		
		dalvik.vm.heapsize=32
			Allows the phone to sleep better Saves power when phone is in sleep mode Allows your wifi to scan less frequently, saving more battery Helps Scrolling Response
			
			Lower to fix black screen after calls issue.
			
			Change the Dalvik VM heap size
				Yes

2.	Brightness Control	OnClick() Detects whether plus or minus is pressed	If plus is pressed brightness increases by 10, If minus is pressed brightness decreases by	Yes

		setBrightness( ) Gets the window properties and then sets the brightness according to input given.	10. Brightness increases or decreases according to the input given.	Yes
3.	Voltage Control	#CONFIG_CPU_GOVERNOR #CONFIG_CPU_FREQ_VDD_LEVELS	When used the required governor is set and other adjustments are made When used the clock speed of the processor is set	Yes
4.	Bloatware Removal	Unnecessary apks were removed from /system/app	On booting they did not show up.	Yes

Reference

1. forums.xda-developers.com
2. androidforums.com