5

Software Testing Strategy for Mobile Phone

Guitao Cao, Jie Yang, Qing Zhou and Weiting Chen
Software Engineering Institute, East China Normal University, Shanghai, China

1. Introduction

With the rapid development of embedded systems and extension of embedded application domain, the scale and complexity of requirements for embedded software have continuously improved. And the product quality and marketing time depend upon its software quality and development cycle. As a result, embedded software testing becomes a hot issue of the research.

Embedded system usually adopts hierarchy as their software architecture based on real-time kernel, as shown in Fig.1 [1]. From this figure we can see that the system software is separated into the following layers:

Application Layer
Middleware Layer
Operating System Layer
Driven Layer
Hardware

Fig. 1. Software structure of embedded system

1. Driven Layer (also called Hardware Abstraction Layer, HAL), as the bottom of software, is closest to hardware. This layer directly makes contact with hardware, providing hardware interfaces for operating system and applications, in other words, gives drivers support. The manufacturers generally supply complete software of driven layer during the process of selling hardware so that developers can use them directly.
2. Operating System Layer (known as System Software Layer). It is responsible for cooperating with application layer to improve their scheduling, management and exception handling. Thanks to the support of this layer, especially the real-time kernel, the difficulty of developing embedded software has been eased greatly and the development cycle has been shortened.
3. Middleware Layer refers to Software Realization Supporting Layer. The realization of embedded application software requires the support of programming languages, such as procedure-oriented C, and object-oriented C++ and Java. When using these languages to develop application programs running in embedded systems, relevant compiler and interpreter are required to convert the programs into machine code in order to realize the corresponding functions.

4. Application Layer (called Application Software Layer). Application layer lies in the top of the embedded system hierarchical architecture, mainly comprising multiple relatively independent application tasks to accomplish specific work, such as I/O, computing and communicating. Operating system is in charge of scheduling these tasks.

Embedded software testing is more complex than general business-oriented software because of its characteristics such as real-timing, insufficient memory, shortage of I/O channels, expensive development tools, close relationship with hardware dependency, and various CPUs. Meanwhile on account of the high reliability demands of embedded system, safety invalidation will probably bring out catastrophic results. So it is of great importance to perform strict test, validation and verification on increasingly complicated embedded software[2].

Embedded software testing shares the same objectives and principles with other general software testing to verify and reach the reliability requirements. However, as a special kind of software testing, embedded software testing owns its unique characteristics as follows:

1. Specific hardware dependency. Embedded software can only run in specific hardware environment. Thus, the most important purpose for testing is to ensure the embedded software could run more reliably in this particular environment.
2. Real-time characteristics. Besides high reliability, the testing also needs to guarantee the real-time characteristics.
3. Memory test. In order to meet the demands of high reliability, memory leak is not allowed in embedded system. Therefore, embedded software testing not only consists of performance test, GUI test and coverage analysis test but also memory test.
4. Product testing. The ultimate purpose of embedded software testing is to make embedded products satisfy safety and reliability while fulfilling other functions. Hence, we need to perform product testing after the first embedded product has been produced.

By now, the majority work of embedded software testing emphasizes on debugging phase instead of studying comprehensive techniques. Along with the widespread popularization of embedded software application and increase requirements of quality assurance, testing needs more systematic and engineered technical supports apart from debugging.

This paper takes mobile phone testing as an example to introduce the implementation process of embedded software testing and analyze the methods after presenting testing techniques and tools, attempting to form a kind of engineered solution for testing.

2. Software testing

Software testing is a process of choosing appropriate test cases and executing the program to be tested for the purpose of detecting program defects according to the IEEE 1983 standard. In IEEE Std 829-1998, which is the revision of IEEE (1983), testing is defined as software analysis procedure of (A) testing one or more sets of test cases, or (B) testing one or more sets of test processes, or (C) testing one or more sets of both test cases and test processes, aiming to find the place where the realization of software function does not accord with the requirements and also to evaluate the software[3].

2.1 Software testing techniques

Software testing technology provides specific support method for each stage of software test. It can be divided into static testing and moment testing from the perspective of whether tested software needs to be executed.

2.1.1 Static testing and moment testing

Static testing is the process of searching for defects that probably exist in program and assessing program code, rather than executing the program. Static testing mainly consists of code reviews, code walkthrough, desktop check, technology check and static analysis. Among them, the first four processes are all conducted manually while the static analysis proceeds automatically by software tools[4].

Moment testing, also called dynamic testing, executes test cases via software to obtain the real operation situation. It chiefly includes black box and white box testing. Every engineering product can choose one of the following ways to test:

1. Suppose the function specification of the product has been generated, whether each function complies with the requirements can be proved by testing.
2. Suppose the internal working process of the product has been determined, we can verify that each internal operation is in accordance with the designed requirements and guarantee all internal components have been inspected.

These two kinds of methods from different angels are both commonly used in testing. The former is black box testing and the latter is white box testing.

2.1.2 Black and white box testing

Black box testing is also known as functional and data-driven testing, considering system as a dark box. We only need to check whether the program functions meet the specification requirements while ignoring internal logic of the program. Testers are forbidden to use their knowledge of system internal structure or experience. The typical black box testing methods are equivalence partitioning, boundary value analysis and cause-and-effect diagram.

Black box testing mainly contributes to find a few kinds of mistakes[5]:

1. Incorrect functions or omitted functions;
2. Improper input and output;
3. Data structure error or access failure to external information (such as data files);
4. Poor performance compared with requirements;
5. Initialization or termination error.

White box testing, referred to structure testing and logic-driven test, regards test object as a transparent box, allowing testers designing and choosing test cases in the view of internal logical structure of the program and relevant information, and testing program logical paths. Through inspection of the program status at different states, testers can determine whether the real state is consistent with expected[6].

We can follow these steps to examine the program:

1. Test all independent execution paths in program modules at least one time;
2. For all logical judgments, test them at least one time in true and false conditions, respectively;
3. Execute the loop in the boundary of circulation and body;
4. Test the effectiveness of internal data structures, etc.

White box testing is an exhaustive path testing. In fact, the number of independent path running through the program is probably fabulous in amount, and it is difficult to traverse all paths. Even though we think each path has been tested and the coverage has reached 100%, the program still might be wrong.

White box testing methods comprise logical coverage, circulation coverage and basic path coverage test. Among them, logical coverage can be distributed into statement, decision, condition, decision/condition, condition combination and path coverage.

Besides white box and black box testing, there is another kind of test method-gray box testing. Just like black box testing, gray box testing is executed through the user interface. Moreover, testers must understand the design of the source code or relevant software functions, and even have already read part of the source code. Owning in-depth insight into the product internal design and thorough knowledge about it, testers can perform certain purposive condition/function tests and test its performance from the user interface more effectively.

2.2 Software testing type

The process of software system testing includes conducting unit test of software modules, assembling them for integration testing, combining the subsystems into software, and finally validating and verifying the system functions in the real environment according to the specification[7].

2.2.1 Unit testing

Unit testing inspects the correctness of program modules, aiming to find the errors possibly exist in each module by testing cases designed according to the program internal structure, which is the reason why we choose white box testing technology. For embedded software, it runs usually on host machine[8].

2.2.2 Integration testing

Integration testing is a system testing technology to detect errors associated with software interfaces after each module is integrated into subsystem or system according to the design requirements (e.g. structure diagram). Both white box testing and black box testing are used in this technology, and test cases are mostly built by black box testing. Integration testing usually executes in host environment similarly[9].

2.2.3 System testing

System testing contributes to discover the inconformity and contradictory between software and system definition by comparing with the system specification. The test object is the entire

software system, including not only the software itself, but also the hardware/software environment it relies on. System testing needs to be performed on the target machine.

2.2.4 Acceptance testing

Acceptance testing is also known as validation testing. It is responsible for verifying the validity of the software, which means to verify whether the software function, performance and other properties are consistent with the user requirements. This testing usually adopts black box testing and commonly implements on the target machine[10].

3. Embedded software testing technology

Due to the characteristics of embedded system differing from the desktop system, the development and testing of embedded software is pretty different from commercial software. Developers still develop their own testing platform because of lack of available general tools. The following factors will influence the software testing:

a. Platform of application and development separation from operating;
b. Complexity and diversity of development platform;
c. Strict limit of hardware resource and development time;
d. Software and hardware developing concurrency, and modularization and hierarchy combination alternatively;
e. Lack of visual programming mode;
f. Ceaselessly upgrading of software quality requirements and certification standards because of business change.

3.1 Testing environment

Simulation is commonly used to test external devices in embedded software testing, which makes use of specific software or hardware simulation tools so as to simplify the testing environment. Simulation classifies into hardware simulation and software simulation. In embedded software testing, hardware simulation attempts to replace hardware and software by means of external equipments, which are identical to the target device in software interaction functions and fairly close performance. Software simulation is developing corresponding software to substitute for peripherals. The alternative software has the same software interaction function as target hardware, but their performance differs a lot[11].

3.1.1 Emulator

According to the difference between testing environment and real environment, embedded software simulation testing environment (ESSTE) can be distributed into full physical, semi physical and full digital simulation testing environment[12].

3.1.1.1 Full physical environment

The software is tested in the real physical simulation testing environment. The entire system (including hardware platform and embedded software) directly cross links with other physical equipments, forming a closed loop. It focuses on examining the interface between

testing system and other interactive equipments. The requirements of testing environment are relatively low.

3.1.1.2 Semi-physical environment

Semi-physical simulation testing utilizes simulation model to emulate the linking equipments, and the testing system is actual. The linking environment of the system consists of hardware and software formed by input/output devices and their I/O interfaces, fulfilling closed-loop testing automatically, real-timely and non-intrusively. The emulator is required to simulate input and output of the real physical environment, and can drive the tested software to run preventing any other input and acquire the output results.

3.1.1.3 Full digital environment

Full digital simulation testing environment is a set of software system including simulated hardware and peripheral. It is established on the host machine with the combination of CPU control chip, I/O, terminal and clock simulation, providing a precise digital hardware environment model. Full digital simulation testing has the most complex requirements among these three kinds of test environment.

3.1.2 Cross-debug

Embedded software debugging has a great difference with general software. In common desktop operating system, the debugger and programs to be debugged are located at the same computer with the same operating system. For example, when we develop applications using Visual C++ in Windows platform, the debugger is an independent process, running and controlling the tested process via interfaces provided by the operating system. However, in embedded operating system, developing host and target locate at different machines. So we could adopt cross-testing to debug and test the program on target machine, and capture whether it receives test data normally.

Cross debug, also regarded as remote debug, means that the debugger runs on the desktop operating system of the host, and the tested program is executing on the embedded operating system of the target machine, respectively. Cross-debug allows debuggers control the operation mode of the process, examine and modify all kinds of variable values in memory unit, register and process on the target machine.

The communication between host and target machine can be realized through serial port or Ethernet port basis on TCP/IP protocol.

3.2 Testing tools

We can adopt general software testing techniques and tools for embedded software system testing, such as static and dynamic test techniques, and requirement analysis and static analysis tools. According to the research on embedded software testing, technical challenges still exist:

1. The poor commonality or even lack of testing tool that fits some certain area.
2. Inability to visualize the software execution procedure because of instruction pipeline, dynamic reset, cache, etc.

3. Waste of time to correct problems that can be prevented formerly, such as memory allocation error.
4. Failure to confirm the testing validity. The testing accuracy is sensory evaluated, so we do not know which tests are valid and which are not[13].

Nevertheless, the use of test tools is necessary. At present time, software plays an important role in embedded system. The testing stress occurs because of the rapid development of software itself and great pressure in transaction cost, the complexity of software functions and the shortage of development time. Moreover, systematized testing becomes inaptitude for meeting the requirements independently in a short period of time. Therefore, we must use test tools such as CodeTest so as to ensure the progress and quality of testing.

Currently, the testing tools of embedded software can be divided into software testing tools, hardware testing tools, and combination tools of software and hardware[14]. The principle, advantages and disadvantages of these kinds of testing tools are described as follows.

3.2.1 Software testing tools

Software testing mostly adopts software simulation, which is to simulate the target machine in the host, making the majority of the test can be done in the simulated host. Most of the embedded testing tools use this technology, including LogiScope by Telelogic and Coverage Scope from Wind River.

Host/Target software testing tools use instrumentation technology[15], inserting some functions or statements in the test code to generate data and send them to the shared memory of target system. In the meantime, a task is running in target system to preprocess data and convert them to the host platform through the debug port by target processor. In this way, testers are able to learn the current running state of the program.

However, instrumentation functions and preprocessing tasks inevitably exist and will increase the system code and reduce the efficiency of the system by more than 50%. Preprocessing occupies the resource of CPU time, shared memory, and communication channels of the target system to complete data processing and transmission. Besides, large amounts of instrumentations will affect system operation during coverage analysis. Therefore, software testing tools based on Host/Target lack of performance analysis because they are unable to analysis functions in the target system and run time of the tasks accurately. They also fail to observe memory allocation dynamically.

3.2.2 Hardware testing tools

Multimeter, oscilloscope and logic analyzer are used not only in hardware design and testing, but also in software testing. Among these tools, the logic analyzer is the most commonly-used. It can be acquainted with the working status of the system and the current condition of the program by the means of monitoring system instruction cycle on bus, capturing these signals in certain frequency and analyzing these data. Whereas some important signals will be inevitably lost because of the way it works by sampling, and it can merely analyze limited functions. So it is really difficult to come to satisfactory results.

Meanwhile, hardware tool has no ability of analyzing and checking memory allocation[16] because hardware tools capture data from system bus when doing coverage analysis. For example, when Cache opens, the system will adopt the instruction prefetch technology to read a section of code from the auxiliary storage into L1 Cache. Once the logic analyzer has monitored signals that the codes have been read on bus, it will report that the code has been executed. However, the codes sent to the Cache may not be hit actually. To avoid this error, Cache must be closed out which means the system is not in real environment. More seriously, Cache off can lead to the system working disability.

3.2.3 Combination tools of software and hardware

The combination testing tools of software and hardware have inherited the respective advantages of software and hardware testing tools, abandoning their shortcomings at the same time. For instance, CodeTest, designed by Applied Microsystems Corporation (AMC) is a kind of high performance testing tool for embedded developers using instrumentation technology, applying for both native and in-circuit testing.

CodeTest inserts an assignment rather than instrumentation function as in software testing tools. It runs extremely rapidly while avoiding interrupted by other interrupts, so it has little effects on the target system and instrumentation processes. Meanwhile, it captures data on bus only when the program runs into an inserted special point by monitoring the system bus instead of fixed sampling. So it ensures precise data observation.

CodeTest provides the following functions:

1. Performance analysis;
2. Coverage analysis;
3. Dynamic memory allocation analysis;
4. Execution traces analysis.

Although the software instrumentation and bus data-capturing techniques have improved, the drawback that CodeTest depends on hardware is exposed. CodeTest must be customized for different signal acquisition probe and related data acquisition mechanism on different hardware platforms, which makes CodeTest poor in flexibility and portability. At the same time, this mechanism also greatly restricts the product adaptability to the market, increasing the cost and the development cycle for new products[17].

From the above analysis, we can find that one or more following technologies are used in various testing tools:

1. Part of the program has been pushed forward by inserting lots of “printf”.
2. Software testing tool is a kind of simulation that it doesn’t work in real system.
3. Target software testing tool runs on the same hardware platform with tested software at the same time, taking up resources of the testing system, such as CPU time and communication port.
4. Hardware testing tools, such as logic analyzer and simulator, cannot run in the Cache open mode.
5. Hardware-aided software tool, like CodeTest, is a real-time online testing system, available in the cache open mode and not occupying any resources on the target board.

Although some test tools have already led to a certain good results in some applications, these tools still fail to meet the requirements in specific applications because of the diversity of the embedded system. So it is difficult to form test solutions that fit all specific test cases.

4. Mobile phone software testing

This paper takes the testing of mobile phone contact module that locates in software system MMI (Man Machine Interface) layer as an example to research testing technology of embedded software.

The development of mobile phone software system uses sequential life cycle model, named V model[18], as shown in Fig.2.Correspondingly, testing must accompany with the development process of the project, as shown in Fig. 3. Therefore, test plan ought to be set down when development plan is being formulated; test case should also be prepared when the software requirements have been determined; moreover, when the project development is completed, it must be followed by the reports on test execution, progress and summary.

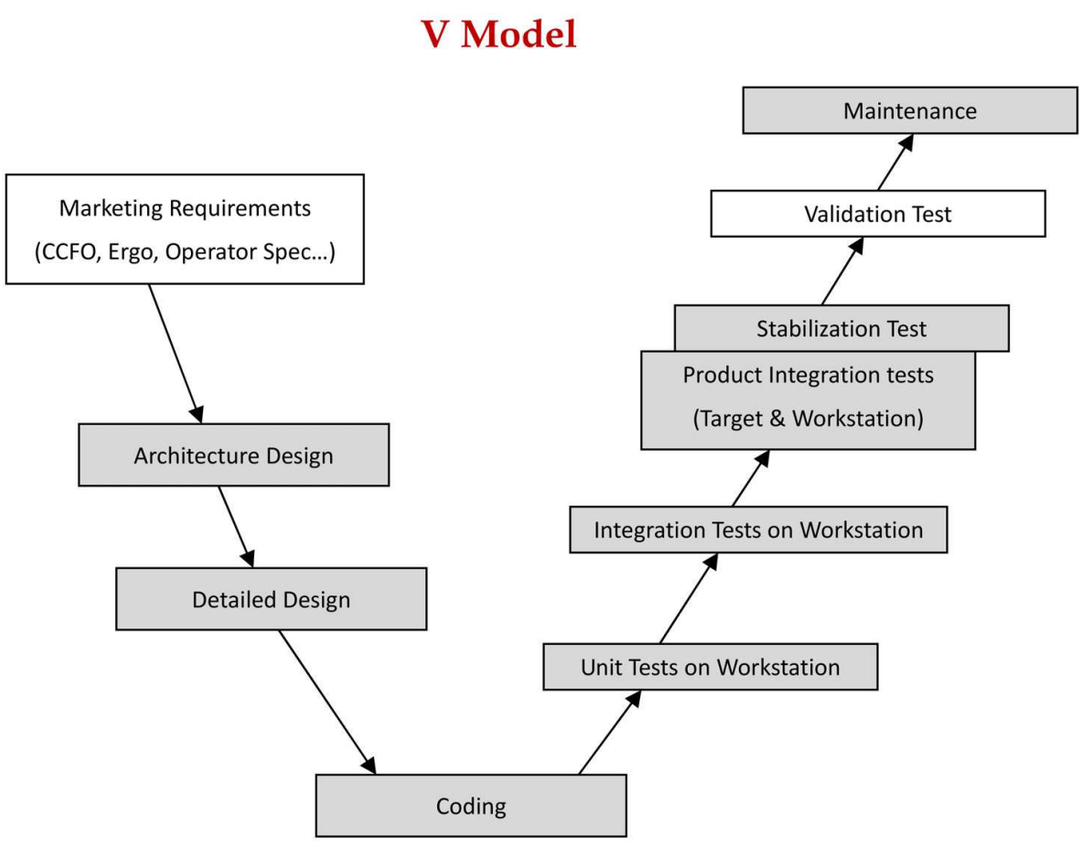


Fig. 2. V model

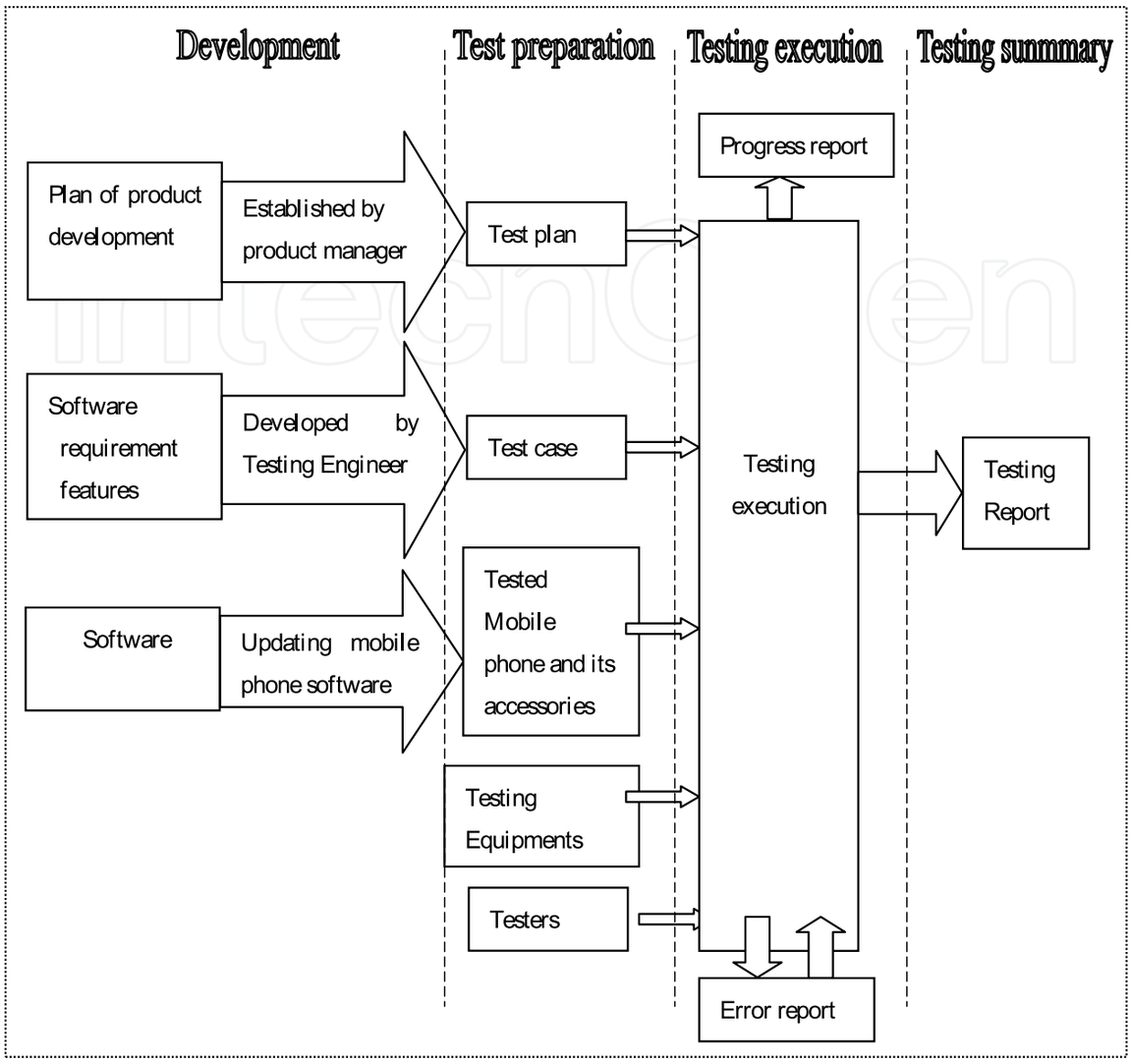


Fig. 3. Testing flow chart

Testing process must base on the overall project requirements to define, restrict and manage all the processes, activities and changes in the whole testing life cycle. Its critical function includes testing requirements extraction, test strategy and planning, test design, test execution and test result analysis.

This research uses TestLink[19] and Bugzilla[20] as management tools in the testing process.

4.1 Introduction to mobile phone operating system

Different from the general operating system and its variety functions, the purpose of embedded operating system is so definite and direct that it only works to resolve one or several functions. Mobile phones are required to satisfy the specific requirement of fast response time as communication tools, which is a typical real-time operating system. For instance, phone call should be answered in 90 seconds; otherwise it will be hung up.

Mobile phone operating system can be divided into two parts: one is the kernel, named Real-Time Executive (RTX); another is I/O. Since there is less demand for I/O in embedded system, the mobile phone operating system is essentially a real-time executive program to manage increasingly complex system resources and map hardware effectively, especially the growing embedded processors. Whatever embedded Micro Controller Unit (MCU), Embedded Micro Processor Unit (EMPU), Embedded Digital Signal Processor (EDSP) or highly-integrated System on Chip (SOC), the operating system can provide identical API interfaces, making developers get rid of busy with driver transplantation and maintenance. Moreover, it can also supply library functions, drivers, tool sets and applications, etc.

Compared with the common desktop operating systems, mobile phone operating system has distinctive features in storage space, tailorability and reliability[21]. Popular mobile phone platforms are Symbian, Windows Mobile, Linux, Palm, Apple, Google Android, and so on. In this research, we choose Google Android as our testing platform.

Android is an open source operating system based on Linux developed by Google, including all required software for mobile phone, such as operating system, user interface and applications, and without any proprietary obstacles that will hinder mobile phone innovation. The system uses WebKit browser engine, with the functions of touch screen, senior graphic display and Internet. Users can check emails, web search and watch video programs on the mobile phones.

Android system architecture as shown in Fig. 4 is separated into four layers, Linux kernel, system runtime libraries, application framework, and applications[22].

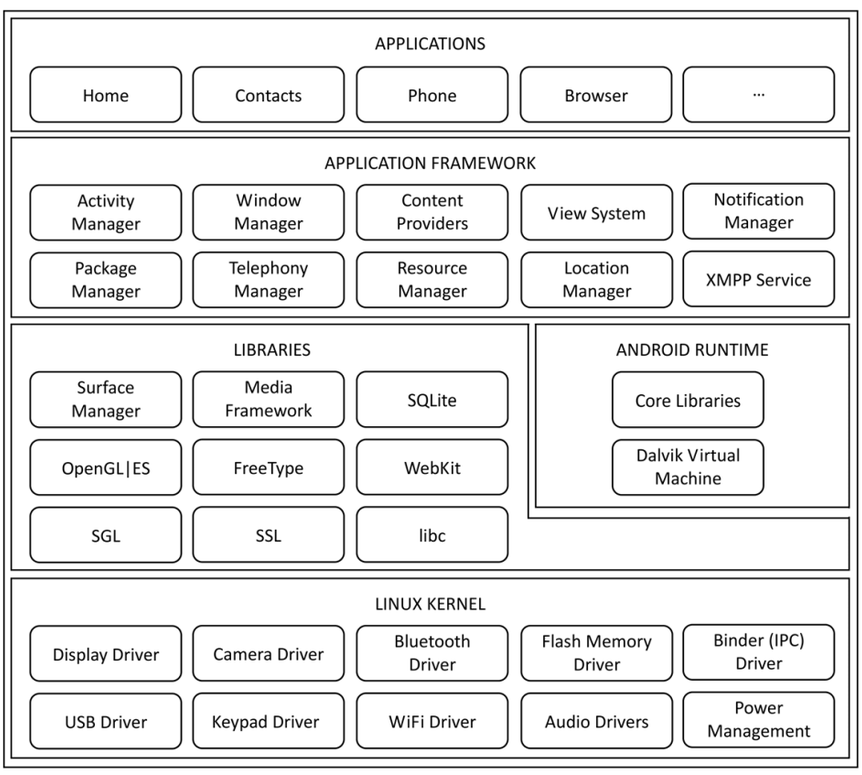


Fig. 4. Android System Architecture

4.1.1 Linux kernel

Android depends on Linux kernel 2.6 to provide core services, such as security, memory and process management, network protocol stack, and driver model. At the same time, the kernel can be served as the abstract layer between hardware and software stack.

4.1.2 System runtime libraries

Android contains a number of C/C++ libraries, which can be used in different components in Android system. They serve for developers via Android application framework. Besides, it includes a core library, called runtime library, providing most functions in programming language JAVA.

Each Android application program runs in its own process with an independent Dalvik virtual machine instance. Dalvik is designed to be a device that can run multiple virtual systems efficiently at the same time, relying on some Linux kernel functions, such as threading and memory management mechanism.

4.1.3 Application framework

Developers can access the API in the core application framework. This architecture simplifies the reuse of components, with each application can publish its functional blocks and others can use them under the security restriction. Similarly, the application reuse mechanism also allows users to replace the program components easily.

A series of services and systems are concealed in each application, they are:

1. Rich and expandable views, building applications including lists, grids, text boxes, buttons, and even embeddable web browser.
2. Content providers, allowing the application to access or share data with another application (such as contact database).
3. Resource manager, providing access to non-code resources, such as local strings, graphics, and layout files.
4. Notification manager, displaying custom message in status bar.
5. Activity manager, managing the life cycle of the application and navigation regression function.

4.1.4 Applications

Android can release a series of core application packages at one time, including Email clients, SMS short message program, calendar, maps, browser, and contact management programs. They are all written in language JAVA.

4.2 Test process management tools

4.2.1 TestLink

TestLink[19] works for managing testing process by managing test requirement, test design, and test execution. In the meantime, this tool also provides many kinds of statistics and analysis of test results, making them clearly.

TestLink, as one of the open source projects of Sourceforge, is a Web based management system. Its main functions include:

- Test requirement management;
- Test case management;
- Coverage management of test cases for test requirement;
- Test plan formulation;
- Test case execution;
- Measurements and statistics of large amounts of test data.

TestLink has following features:

- Supporting Web access, including Mozilla, Firefox, and IE browser;
- Product test in the test plan according to testing process;
- User can customize characters, such as test leader, tester etc.;
- Keywords are used to support profound test organization;
- Assigning testing to testers according to the priority and defining milestones;
- Providing test report;
- Exporting documents to HTML, WORD or Excel format;
- Sending out test report message directly;
- Localization and internationalization;
- Combining with general bug tracking system, such as Bugzilla, mantis and Jira etc.;
- Requirements management based on testing.

Use this tool to implement complete management of testing process, combining defect management tool—Bugzilla, can guarantee testing accomplish successfully.

4.2.2 Bugzilla

Bugzilla establishes a perfect defects recording and tracking architecture, including Bug report and solution, record query, and report forms generation automatically. The characteristics of the Bugzilla are shown as follows:

1. It is a tool based on Web with simple installation, convenient operation and safety management.
2. Bugzilla is propitious to explain bugs clearly. The system provides comprehensive and detailed report entry and generates standard bug report with the help of database management. There are also a large number of options and powerful query matching ability for bug statistic according to various condition combinations. When the error changes in its life cycle, developers, testers, and managers will receive dynamic information timely, allowing them to obtain and refer to its history during checking error state.
3. The Bugzilla system is flexible and occupies powerful configurable capacity. The tool can set different modules for software product and designate developers and testers, so it can send reports to assigned responsible person automatically. Besides, different groups and authorities can be set and divided to make users with different authorities can deal with bugs effectively. Distinguishing severity and priority can manage errors

from the initial report to the final solution, to ensure that the error will not be ignored while focusing on the errors with high priority and severity.

4. The tool can also notify relevant personnel by sending emails automatically. The latest dynamic information will be sent to the different person according to their responsibilities, promoting communications between testers and developers effectively.

So, the test process can start with the help of these two test tools.

4.3 Mobile phone software testing process

Testing process must systematically define, restrict, and manage all the processes, activities and changes in the testing life cycle, following a series of principles, methods, theories and demand of the project. The critical activities include test requirement extraction, test strategy and planning, test design, test execution, test result analysis, and so on. The test process mainly establish the main test plan, directive testing strategies and methods, partition priority, restrict milestone, effective organize test resources (test team, hardware and software environment, for instance), etc. based on project goals. And then, make certain test requirements, test object, test objectives and performance index. According to the test plan and test design, testers can work effectively, such as developing and designing test cases, selecting test tools, designing automatic test framework, writing automated test code, executing test, and finding and reporting defects, etc.

4.3.1 Test requirement

To evaluate software project success or not, we need to judge whether it has solved users' problem which is embodied as user requirements in software engineering. The carelessness in requirement phase may lead to large amounts of rework in software implementation. Statistics show that more than 50% of the system errors are due to wrong requirement or lack of requirement, more than 80% expense is spent in the tracking requirement error because of the intertwined and duplicated work in the tracking error process.

Test requirement analysis is the most important work in requirement analysis in V model, coping with users' original demands and software functional demands, and then decomposing the results reasonably. It targets at the entire product, concerning the whole product system level.

Product testing requirement analysis activities are divided into original requirement extraction and product testing analysis. First is to extract the original demand and analyze each of them using certain engineering method, then to generate the product test specification. There may be repetition and redundancy in the specification via different engineering methods, which could be regarded as initial test specification. Afterwards, it can be integrated according to the testing type and functional interaction.

There are two main requirements for testing phone book module in the phone software, one is SIM card and the other is mobile phone. And the SIM card phone book can be separated into AND (Abbreviated Dialing Numbers) and FAN (Fixed Dialing Numbers). All these phone records must comply with the international standards.

Now, we need to decompose and organize the product testing requirement on TestLink because a product can contain multiple test requirement specifications and a specification can contain multiple test requirements.

1. Create test specification. To simply describe the test requirement, including name and scope.
2. Create requirement. Test requirement includes requirement ID, name, scope, state, and the cases for requirement coverage. TestLink provides two kinds of state to manage requirements: valid and not testable.
3. Import test requirements from file. TestLink makes the possibility of importing requirements from files whose types are CSV and CSV (door).

So, the test requirements can be formed and created.

4.3.2 Test plan analysis and design

Professional testing must be on the basis of a good test plan. Although each step in the test is independent, a test plan is of necessity which is the initiative step and the architecture to link the whole testing process.

Preparing test plan can add opportunities of discussion and communication in the project team. After collecting sufficient information of project, function, state, and performance by group chat, we can confirm the goal of the test plan in the review meeting, including function, performance and schedule goals and so on. So, we can carry the work forward toward a common goal. Otherwise, the team often puts a premium on details in the later phase of testing which will even deviate from the original target finally.

Test plan consists of many contents, from test methods to all individual test cases (referred to test set or test package) which will be used in real testing. In this project, we can distribute test plan into two parts: one is test plan mainly about the plan content; the other is a specific test case document, namely test package.

A detailed test plan needs to elucidate the testing scope, software and hardware resources, staffing, entry and exit criteria, test method, scheduling, major milestone, bug tracking, and version management, etc. In TestLink, it comprises:

1. Name of the testing stage, such as integration test stage and system test stage.
2. Milestone identifies the beginning and end time of each test stage, and implements the ratio of three priorities which are A, B, and C.
3. Build version defines the version to be tested in this test plan, generally named as “product name + time”.
4. Test participants arrange test participants through selecting personnel from users list, shown as Fig.5.
5. Test case set.

Establish rules of priority. The system will determine the priority grade in the light of the combination of custom importance and risk levels which refer to Low, Medium, High and 1, 2, 3, respectively.

Fig. 5. Selecting test participants in TestLink

- Select test case sets of this test plan from test cases.
- Define the importance level and risk level of each test case Category.
- Assign the ownership of each test case Category. The owner is selected from the list of responsible tester and the tester will complete the test cases execution.

So test plan is required to write at length for later reference. Along with the refinement of the test requirements, we could clarify the test plan and pay more attention on the following contents, as shown in Fig.6.

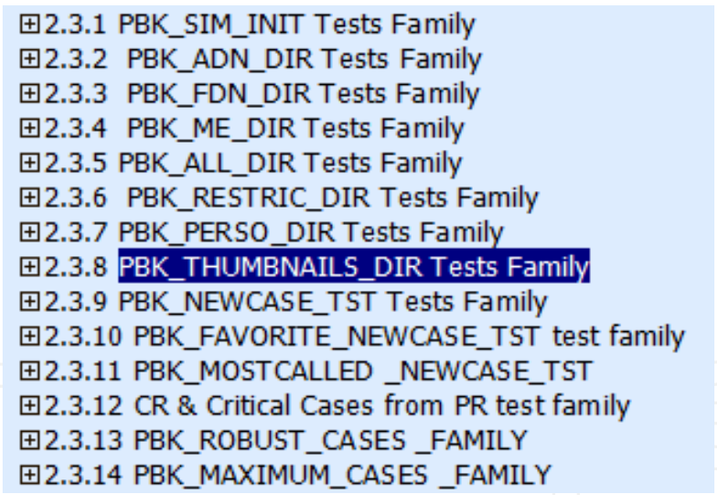


Fig. 6. Test plan in TestLink

4.3.3 Test case design and implementation

We can design detailed test case after test plan has accomplished and passed the review. If there is no test case or only a simple test function description, testing will become difficult to control and test results will be poor in reliability. In addition, simple test cases with poor reliability and reusability may produce ambiguity and errors in the test execution stage, which is extremely adverse to the test result. Hence only detailed test cases can guarantee high reliability, the convenience to estimate the execution time, and the ease of control.

Designing test cases should account for a basis, a purpose and a basic idea. The fundamental base is the requirement specification. For system test cases, testers should design the cases strictly according to the user requirement specification to ensure that they are in line with the requirements. For integration test cases, both software or user requirement specification can be applied as bases to design. The main purpose of designing test case is to make the case simple to be understood. That is to say test case should be in detail enough for any person, including the fresh man, to start testing quickly. The basic idea—one point more cases, means tester could build multiple test cases corresponding to one test or functional point. All test cases design run through these three factors.

Test case management supported by TestLink consists of three layers: Component, Category, and Test case. Component corresponds to the project function modules and each module is linked with Category where Test cases are written. We could use search function to find the required test cases from different projects and thousands of test cases, and even copy test cases directly from other projects. These would help us solve the problem of test case management and reusability.

In test management, testers are deeply concerned about the test case coverage to the test requirements. However, the correspondence between them keeps still unsolved. TestLink provides function of managing their corresponding relationship after test requirements are extracted from specification.

1. Create component: component consists of name, introduction, scope, relevant content, and constraint.
2. Create category: category contains name, test scope and goal, configuration information, test data, and test tools.
3. Create test case: case elements include test case name, brief description, steps, expected results, and keywords.

Fig.7 presents a test case tree in TestLink. For one test case, select the corresponding requirement to assign, their coverage relationship establishes. The ultimate test case spanning tree is shown in Fig. 8.

4.3.4 Testing process

After testing preparation has been well done and the first version of the software has been published, test group could begin testing according to the test plan and schedule. In addition, due to time pressure, we usually appropriately optimize and select the testing content during the test execution process in order to reduce the workload, achieving test purpose as well. Then we execute test cases and record the execution situation of each build edition using one of the following four kinds of test results:

1. Not Run: unexecuted;
2. Pass: execution successes;
3. Failed: execution fails;
4. Blocked: the test case cannot perform due to other failed cases.

TestLink provides abundant measuring statistical functions according to the recorded data in testing process, allowing testers acquire the data to be analyzed and summarized directly during the test management process:

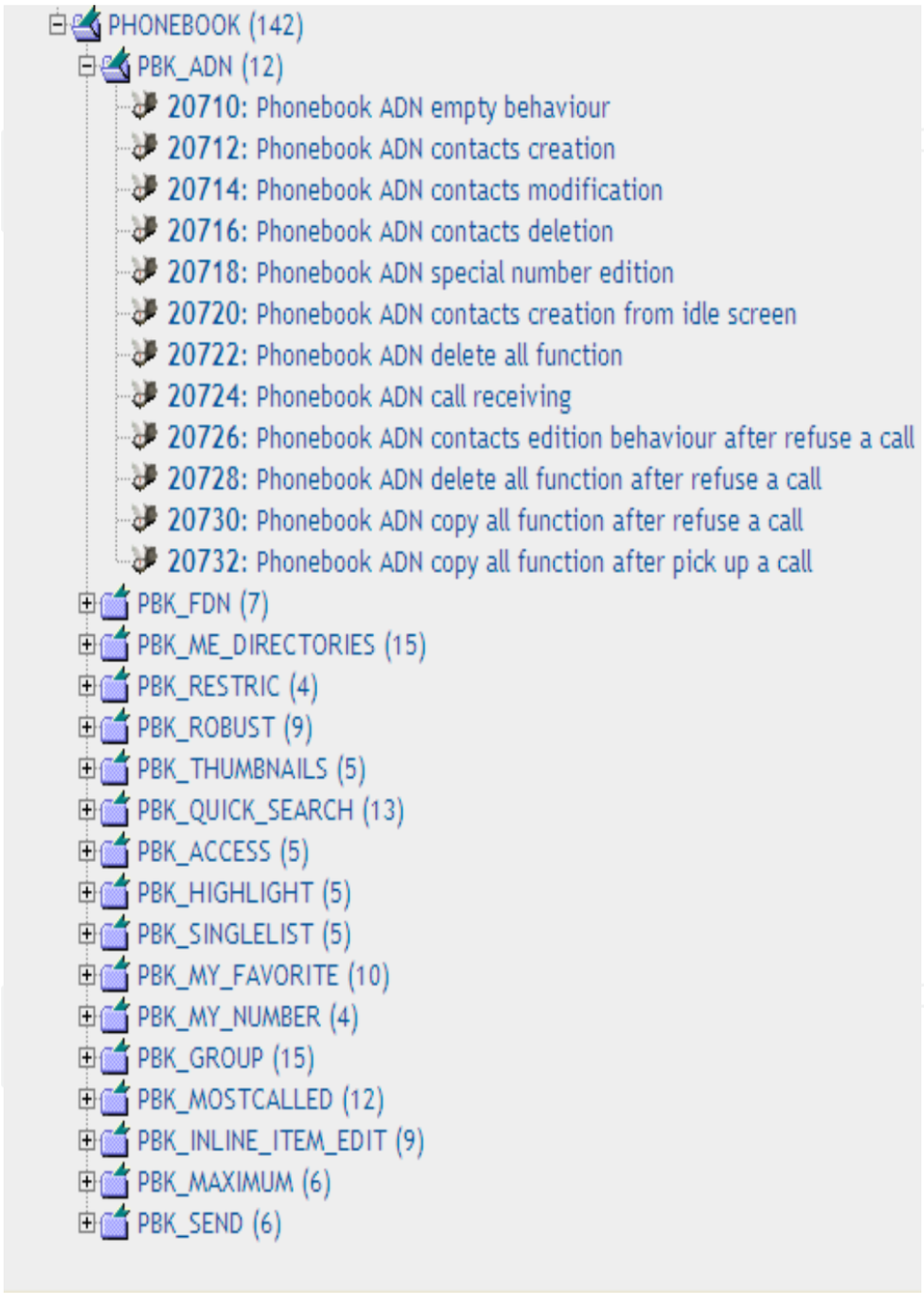


Fig. 7. Test case tree

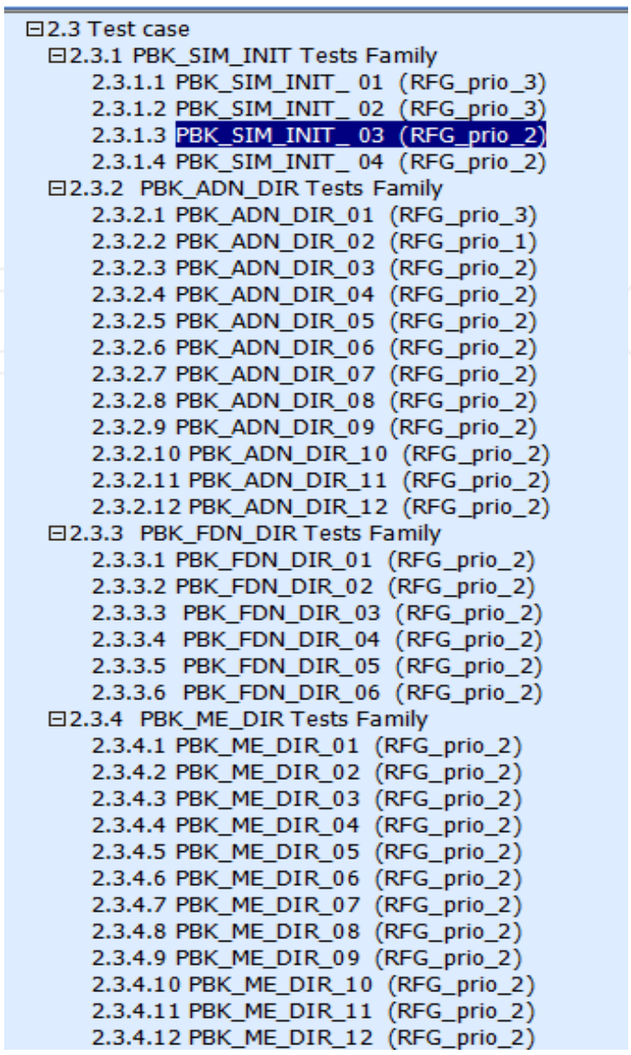


Fig. 8. Ultimate test case spanning tree

1. The requirement coverage: including which requirements have passed the test and which have not, which are in blocked state, and which testing has not yet begun.
2. The test case execution results of each version:
   - The ratio of test case execution with various priorities;
   - The ratio of test case execution for each module;
   - The ratio of test case execution for each tester.
3. The execution result of each version.
4. The execution situation of all test cases in different versions.
5. The list of blocked test cases.
6. The list of failed test cases.
7. The bug report in testing. If combined with bug tracking system, the tool can also statistic the number of bugs in each test case.

When there are blocked and failed tests, we need software patches and new software version to achieve the final effect on mobile phone.

4.3.5 Software defect management

During test process, testers will find a lot of bugs (also called problem, defect, or error), fault and other content that hard to record on paper. It is unrealistic and extremely unfavorable to attempt to use human brains or a document to remember or record all the mistakes, because this would hinder effective contact between the test team, programmers, other developers and the project management team, which is very negative for improving product quality and working efficiency. Therefore, there is a need for an effective method to track a series of state of each error from beginning to end. The seamless combination between TestLink and Bugzilla (shown in Fig. 9) will make bug easier to manage.

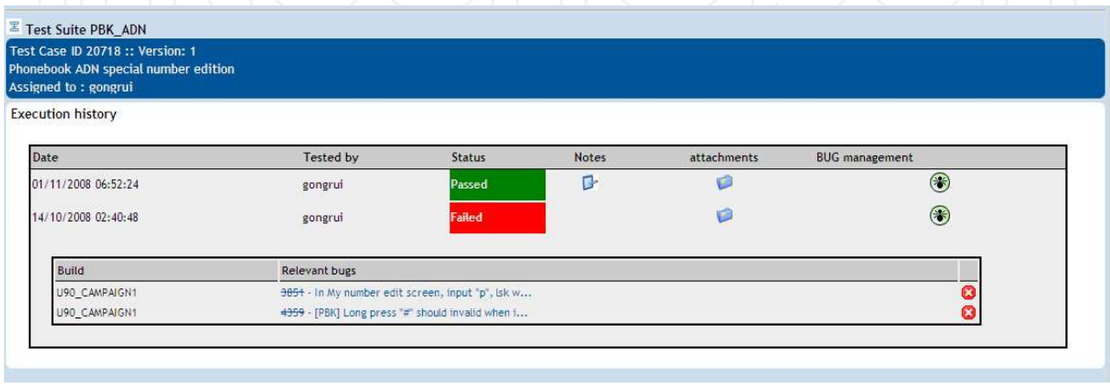


Fig. 9. The combination of TestLink and Bugzilla

a. Once testers or developers find bug, they will judge which module does the error belong to, then they will fill in the bug report, notifying the project leader or developers by emails.
b. The project leader reassigns the task to the developer who is responsible for the bug.
c. After the developer receives Email information and determine whether the bug should be revised by him. If not, he will reassign it to the project leader or the one ought to fulfill it; if yes, then he is required to resolve it and get the solution by creating patch attachment and supplementary specification.
d. After bugs have been modified by developers, testers will perform test again, creating test case attachment. If the tester verifies the test case to be correct, the state will be set to "VERIFIED". When the entire product has been released, the state should be updated to "CLOSED". If there are still problems existing, the tester needs to change the state back to "REOPENED" and email for notification.
e. If the bug keeps unprocessed in one week, Bugzilla will use email to harass its owner until he take actions[23].

Nice error tracking system requires misattributions and stakeholders record history log of the error report in order to capture those important events in the wrong life cycle process, which means testers, developers, project managers and other relevant person need add their logs to the error report, especially the changes. For instance, when developer has modified a bug and he or she needs to set the state to 'review the test case', and add log to explain the revision content. It can facilitate technical experts' review and supply technical clues for acceptance testing. When developers and project manager consider a certain error report as invalid and need to change the case state into invalid, there is a must to add log

demonstrating the reason as much detail as possible, making testers understand and agree the modification easily. Otherwise, testers will still resubmit the error, leading to unnecessary circulation. When the project manager or developers think that a certain bug report needs delay processing, the reason for the delay should be explained in logs so that others can re-examine this error report clearly later, etc. Obviously, the log information is of great importance not only to the project manager, but also to the developer and tester. However, in many projects, part of the team members are not accustomed to writing logs, instead, they just click the button and change the case state. They do not recognize that a few more operations on log can benefit a lot.

In our project, we suggest choose good tool which is complete in functions to work as our bug tracking system. The more comprehensive functions and higher automation degree the tool has, the better it will support the work flow. To take ClearQuest as an example, when changes of the error state occur, the tool can be set to automatically send an email to the stakeholders, notifying the relevant personnel to timely cope with the mistakes or abstracting their attention.

Of course, process automation is only half of the process. Each member of the project team, project manager and test manager must conscientiously carry out their duties, ensuring that each error report go through the life cycle and reached the final state quickly. This is the key of guaranteeing the project progress and quality.

4.3.6 Test case maintenance

No matter how well the test case design and review performs, there are always loopholes in test system. Budget and time pressure, along with human error, will result in incomplete test. Some test cases miss the conditions they ought to cover, some test packages do not contain important test case, or some procedure faults of test case can lead to the wrong purpose of testing. Therefore, test case maintenance, or so-called incremental improvement, is an indispensable step in testing process. If the requirement changes, such as increasing or modifying certain demand, there is also need for upgrading test cases.

The maintenance and upgrading of test case is a necessary long-term work. It can not only deal with errors in the test case, reducing the occurrence of incomplete tests, but also learn from test omission and summarize, providing an experience for next test case design, which will definitely turns to less incomplete test or case error. Its critical part lies in when to be modified and which to be increased. Case upgrading is simple that as long as there is a notification of changing requirement, the test manager could arrange relevant personnel to modify corresponding test cases. The maintenance is relatively elastic for two reasons. The first is that it depends on the tester's consciousness which is associated with their self-quality. Once tester discovers that the test result differs from expected and confirms the inconformity does not arise from bug, or he finds incomplete, error, or uncovered case, he needs to timely inform the case author and test manager. Secondly, the maintenance also relies on test manager's supervision. The manager has the responsibility to check whether the problems occurred in test execution are all covered by test cases, if not, he needs to arrange new test cases.

We recommend a method usually used in project development, named derived case method, for test mangers. Test manager can inspect error tracking database weekly or daily and check the existing test package to ensure that each problem has a test case to cover. If it is found that some problems have no corresponding test cases, the test manager needs to

arrange related personnel to add cases and certain derivation, which means to consider other related cases according to the new case.

Some questions found by developers instead of testers in bug tracking database will result in lack of test cases with the testing steps designed by testers. On the other hand, sometimes problems are found when testers are performing random testing that the testing itself is very casual and without strict steps. In this case, we could draw out these problems as a new test case to design, write and verify.

Using above method to improve and maintain test cases absolutely leads to more and more abundant contents of test package. The coverage goes wider and the quantity of test cases will correspondingly become larger, which makes setting priorities for these cases extremely important. Derivation cases also need to specify the priority and pass the review. Otherwise, it will probably bring about the waste of labor force or testing omission due to too high or too low priority.

5. Testing results analysis and suggestions

The research mainly targets on mobile phone software testing on account of its characteristics. In terms of the software testing theory and methods, we divide test plan and execution into six phases: test requirement analysis, test plan analysis and design, test case design and implementation, test execution and results, test software defect management, and test case maintenance analysis. In the meantime, multiple testing strategies are applied to guarantee the correctness and feasibility of these test processes. The experience in each stage of the test is summarized as follows.

5.1 Test requirement analysis

Requirement analysts must keep in mind that users cannot give complete, clear, and standardized demands at a draught. Instead, they will continue coordinating and regulating the requirements. The analysts need to constantly dig and regulate the demands to acquire user desired requirements. So, we recommend the following ways to collect and coordinate requirements.

5.1.1 Data

Data is static information related to the requirements and described from the user perspective. In the system design process, they are managed and standardized using object-oriented method, forming Class Diagram.

5.1.2 Activity

Activity is the business logic and rules the project has to meet. It comprises at least two levels:

- Basic goal: briefly describing the business logic and rules.
- Function description: representing the execution process, as well as the relevant resources, association, dependencies, and constraints. On account of the consistency of the requirements analysis and system design, activities in system design phase can be expressed by use cases and refined by sequence diagram, activity diagram and state diagram.

5.1.3 Personnel organization

Personnel organization describes the structure of personnel, such as enterprise leaders, departments, department personnel, related customer information, and so on.

In addition, there are some popular requirement analysis software, such as IBM's RequisitePro, Telelogic's DOORS, and Borland's CaliberRM. They occupy different characteristics which can meet the demands of functional requirement analysis.

To summarize, requirement analysis is a team work, and relevant commercial products can be adopted under the guidance of requirement analysis theory.

5.2 Analysis and design of test plan

Good software test plan takes various factors into account. The following is what we need to pay attention to in this process.

Firstly, clarify the test purpose. The test goal must be clear, measurable and quantifiable rather than an ambiguous macroscopical description. In addition, the goal should be relatively concentrated and a series of targets should be avoided. Because it will probably make us emphasize on trivial goals while ignoring those of great importance. Besides, making efforts to fulfill every target will cost a waste. According to the analysis of user requirement document and design specification, we could determine the software quality requirements and the goal need to achieve. Moreover, there are also other demands in test plan, including high coverage of functional requirements, effective test methods, testing tools that are easy to use, intuitionistic and accurate test results, etc.

Secondly, the real-timing of test plan. Test plan comprises many contents and may be restricted by testers' experience and degree of understanding software requirements. Software development is an incremental process, so it is possible that the initial test plan is not perfect and still needs to be improved. When a new requirement generates or omission has been found in test execution process, testers should timely update the test plan.

Finally, explicate the test process and content. Test plan, as the architecture and guide of the test, should give a detailed description about the test content and process, highlighting the critical points in tested content by listing the key and risk content, attributes, scene, and testing technology. Besides, practical methods should be applied in partition of testing process stages, document, bug, and schedule management.

Only meet the three above-mentioned characteristics of the test plan can we satisfy the basic needs of the project. In establishing mobile phone testing plans, we are also required specialized document for detailed test plan due to the limitation of TestLink.

5.3 Test case design

There are three layers of test case management supported by TestLink in the project: Component, Category, and Test case. The Component corresponds to the project function module, with its functions corresponding to Category where Test cases are written. In addition, TestLink can also provide the assignment between requirements and test cases. This guarantees the integrity and reusability of test case design in essence.

With the excellent test case management of TestLink, we need to pay attention to the following aspects:

5.3.1 Cross test case

Cross test is of great significance in mobile phone as a multi-task system with different priority levels because of their various functional orientations. For example, making phone calls and sending short messages occupy higher priority than any other applications. As a result, the priority order and division of different tasks appears to be particularly critical in mobile phone design. Meanwhile, the phone, as a communication device, can receive messages from the outside world at any moment, such as calls, short messages, third-party Bluetooth connection requests, and so on. These events undoubtedly will affect the running tasks, i.e. games and media playback. When a new event approaches, the system needs to make correct response. Therefore, cross test between different applications plays a primly important role in mobile phone software system testing.

5.3.2 Robust test

Compared with general application software, embedded software requires higher on reliability, which makes robust test essential. For instance, keyboard keys, memory fill, and read and write test of phone book are all in need of repeatability extreme test to ensure their stability.

5.4 Analysis of test results and version management

TestLink, providing abundant measuring statistical functions according to the recorded data in the testing process, can directly acquire the data to be analyzed and summarized in the test management process as long as each build version is guaranteed, and cases could be completely executed and verified constantly in the new version.

It is necessary to analyze the ultimate execution results of the test cases. If failed or blocked cases exist, we need to use software patches or new software version to execute the cases over and over again. Mobile phone software usually needs a dozen of versions to ensure its stable. During this process, bugs are required to timely submit to Bugzilla for management.

5.5 Defect management

Software defect is a by-product in the process of software development, even leading to software products failure to fulfill the needs of customers to some extent. Whether software testers can express the severity and priority of the bug accurately not only has an effect on the quality of software defect management, but also possibly influences the time of treating bugs. Especially in the later testing period, it will affect if the software can release on time. The standard of classification should refer to the customers' demand, which means to classify the severity in terms of whether the users can accept this defect or not.

Bugzilla is utilized in managing defects, identifying the defect severity and priority by P0, P1, and P2. Among them, the partition of the three priorities relies on three indicators the probability of defect reproduction × the extent to influence customers × customer satisfaction. Only strictly divided priority in accordance with such criteria can we guarantee the rationality of defect severity classification.

The generation of bugs will definitely be accompanied by software patch release. However, it is often seen that old patch causes new defects in projects. To avoid this situation, validating the software patch carefully and considering its influence synthetically seems to be particularly significant. And developers could also mutually supervise each other, establishing good mechanism to prevent this kind of bug.

Automation tools and history logs of the error report are very important for a good error tracking system.

5.6 Maintenance and analysis of test cases

During test execution phase, the test manager is often on the wing executing tests and bug tracking so that he possibly ignores the case maintenance. In this project, we recommend derived case method for test manager examine bug tracking database every week or every day. If some problems are found not corresponded to test case, the manager should ask concerned personnel to add cases and do certain derivation, that is, to think over other related cases on account of the new case. For the problems submitted by developers or random testing which result in lack of test cases, the manager could draw out them as a new function test case to deal with.

6. Conclusion

Along with the acceleration of information society, personal mobile communication has increasingly become so popular that the number of mobile phone users grows rapidly, with diversified requirement and high quality. However, the research on mobile phone software testing and testing tools are quite rare in view of the particularity of its testing platform.

This chapter has made a systematic and detailed study on the mobile phone software testing process. In addition, using software engineering theories combined with actual projects, we have assessed experience and lessons, and put forward some improved practical methods. The contribution includes making requirement analysis methods, test plans and test scheme, designing and executing test cases, analyzing test results, selecting and using test tools, and so on.

Our later work will primarily focus on studying test tools as well as white box testing methods. We hope that, with the increasingly maturity of science and technology, we could become capable of doing further in-depth research on testing tools, especially in automation testing tools, accomplishing more effective automation rather than simple button simulation. Only with an efficient testing tool can testers guarantee the software quality. Moreover, studying on white box testing methods and tools are recommended in order to increase test coverage and expand test range.

7. Acknowledgment

This work was supported by Natural Science Foundation of China (61021004, 81101119), and the National High Technology Research and Development Program of China (2011AA010101863).

8. References

[1] Kang Y.; Zhang Y.; Li Z.; Hu .; Hu W. (2008). *Testing Embedded Software*, China Machine Press , Beijing

[2] Hawkins J.; Howard R. B.; Nguyen. H. V (2002). *Automated real-time testing for embedded control system*, IEEE 2002: 647-652.

[3] IEEE, *IEEE Standard for Software Test Documentation*. IEEE Std 829-1998.

[4] Park C.Y.(1997). Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths. Real Time Systems. 1997, 13(1): 67-91.

[5] Gu L.; Shi J.L. (2004). *An introduction to software testing technology*, Qinghua University Press, Beijing

[6] Myers C.J., Revised and Updated by Tom Badgett and Todd M.Thomas with Corey Sandler(2004). *The Art of Software Testing(Second Edition)*, John Wiley&Sons Inc, Hoboken, NewJersey: 14-20.

[7] Lin N.; Meng Q.(2005). *Practical guide to software testi*ngQinghua University Press, Beijing

[8] Hatton L.(2004). *Embedded System Paranoia:a tool for testing embedded system arithmetic*. Information and Software Technology, 47(2005)555–563, 2004.10.

[9] School of Computer and Information Science, University of South Australia. *Software Testing*, Available from: http://www.sweforum.net/test/softwaretesting.pdf. 2002.2.

[10] Pressman R. S.(2007). *Software Engineering: A Practitioner's Approach*. China Machine Press, Beijing

[11] Chen Y.; Li M.; Yang Y.(2004). *Open source embedded system software analysis and practice - based on SkyEye and ARM development platform*, Beihang University Press, Beijing

[12] Zhang H.; Ruan L. (2002). *A study on the Simulation Modeling for Embedded Software Testing*, Measurement and control technology

[13] Zhang L. (2005). Structural Testing Technology for Embedded Computer System Software. *Ship Electronic Engineering*, 2005, (3): 63-64.

[14] Wan K.; Lu Q. L.; Peng Y. L.(2001). Study and Realization of the Chain Method for Testing Examples Automatic Building. *Journal of Armored Force Engineering Institute*, 2001,Vol.15, No.3: 55-58.

[15] Sun C.; Jin M. (2001). Program Instrumentation Approach to Implementing Software Dynamic Testing. *Journal of Chinese Computer Systems*, 2001, 22(12): 1475-1479.

[16] Sun C.; Le L.; Liu C.; Jin M. (2000). Test technology of real-time and embedded software. *Journal of Chinese Computer Systems*, 2000.9, 21(9), 920-924.

[17] Zhang X. (2006). Chinese embedded software industry development report. *Software World*. 2006.5.

[18] Brian Marick. Analysis of V model problem. Available from http://tech.ccidnet.com/art/1086/20030225/38813_1.html. 2003.2.

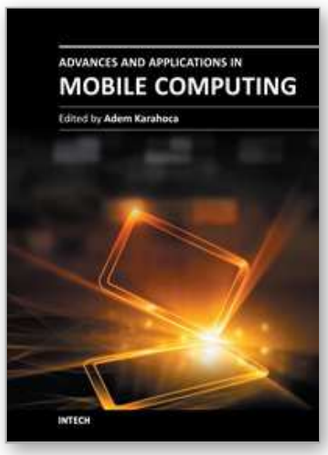
[19] TestLink User Manual.version1.8. Available from http://testlink.sourceforge.net/docs/testLink.php.2009.4.

[20] The Bugzilla Team. The Bugzilla Guide - 3.3.4 Development Release. Available from http://www.bugzilla.org/docs/3.4/en/pdf/Bugzilla-Guide.pdf. 2009.

[21] Lin X. (2006). Research on mobile phone software testing. Master degree thesis. Tonji University. 2006.

[22] Developers.What is Android. Available from http://developer.android.com/guide/basics/what-is-android.html.

[23] Meng Y.; Liu Z. F. (2005). The experience and practice of bug management (2rd) how to set up the bug management system. *Programmer*. 2005(2)



Advances and Applications in Mobile Computing

Edited by Associate Prof. Adem Karahoca

ISBN 978-953-51-0432-2
Hard cover, 224 pages
Publisher InTech
Published online 30, March, 2012
Published in print edition March, 2012

Advances and Applications in Mobile Computing offers guidelines on how mobile software services can be used in order to simplify the mobile users' life. The main contribution of this book is enhancing mobile software application development stages as analysis, design, development and test. Also, recent mobile network technologies such as algorithms, decreasing energy consumption in mobile network, and fault tolerance in distributed mobile computing are the main concern of the first section. In the mobile software life cycle section, the chapter on human computer interaction discusses mobile device handset design strategies, following the chapters on mobile application testing strategies. The last section, mobile applications as service, covers different mobile solutions and different application sectors.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Guitao Cao, Jie Yang, Qing Zhou and Weiting Chen (2012). Software Testing Strategy for Mobile Phone, Advances and Applications in Mobile Computing, Associate Prof. Adem Karahoca (Ed.), ISBN: 978-953-51-0432-2, InTech, Available from: http://www.intechopen.com/books/advances-and-applications-in-mobile-computing/software-testing-strategy-for-mobile-phone

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen