Hindawi
Mobile Information Systems
Volume 2017, Article ID 9319362, 25 pages
https://doi.org/10.1155/2017/9319362

Research Article

Automatic Data Logging and Quality Analysis System for Mobile Devices

Yong-Yi Fanjiang and Chih-Pin Wu

Department of Computer Science and Information Engineering, Fu Jen Catholic University, New Taipei City 24205, Taiwan

Correspondence should be addressed to Yong-Yi Fanjiang; yyfanj2@gmail.com

Received 2 November 2016; Revised 10 March 2017; Accepted 23 March 2017; Published 27 June 2017

Academic Editor: Porfirio Tramontana

The testing phase of mobile device products includes two important test projects that must be completed before shipment: the field trial and the beta user trial. During the field trial, the product is certified based on its integration and stability with the local operator's system, and, during the beta user trial, the product is certified by multiple users regarding its daily use, where the goal is to detect and solve early problems. In the traditional approach used to issue returns, testers must log into a web site, fill out a problem form, and then go through a browser or FTP to upload logs; however, this is inconvenient, and problems are reported slowly. Therefore, we propose an "automatic logging analysis system" (ALAS) to construct a convenient test environment and, using a record analysis (log parser) program, automate the parsing of log files and have questions automatically sent to the database by the system. Finally, the mean time between failures (MTBF) is used to establish measurement indicators for the beta user trial.

1. Introduction

Communication technology is advancing. Mobile phones from the first generation (1G) only used voice communications; the second generation (2G) had the wireless application protocol (WAP) function. The first 2.5 generation (2.5G) contained general packet radio services (GPRS), the third generation (3G) contained data communications, and the fourth generation (4G) had long-term evolution (LTE), high-speed Internet access, and Wi-Fi AP (access point) popularity. Because the technology and external environment convenience has allowed the right conditions to improve work processes and methods early, systems are more convenient and automated, which results in greater output and accelerates processes.

During the development phase of handheld devices, developers must often observe a large number of logs to understand the behavior of the device, that is, to determine occurring exceptions and to take steps to track an issue [1, 2]. When the problem is repeated, logs are created to understand the root cause of the problem. There are two important testing jobs, the field trial and the beta user trial, which are performed by a quality control unit during a later product development stage as shown in Figure 1. These stages are called the Engineering/Running (E/R) and Production/Running (P/R) phases, where the former selects the test items, and, during the latter stage, many users are asked questions and encouraged to make comments regarding the daily use of the product. The E/R phase will be shared by professional testers, whereas, for the P/R phase, the users consist of company's colleagues who are interested and undergo training. An automatic logging analysis system (ALAS) would automatically upload logs, have a simple user interface, and create automatic analysis logs [3–5] and summary reports, which would effectively enhance the efficiency of developers and testers.

Regarding the traditional field trial and beta user trial, issuing a return requires connecting to the system and filling out a questionnaire. Then, a computer or notebook must be used in a browser or file transfer protocol (FTP) software to upload the logs, which requires manually downloading and updating the software version [6]. All personnel involved in the testing must complete training that includes how to return an issue, dump logs, upload logs, download new versions, and update mobile versions. The corresponding systems and services must build and open the account service, which

MRS: Marketing Request System
E/R: Engineering/Running
M/P: Mass/Production
S/R: Sample/Running
P/R: Production/Running
EOL: End of Life

Figure 1: Product life cycle.

includes miscellaneous process. Developers typically view the logs of problem devices, where most developers use a text editor and manual interpretation operations for issues related to logs. Keyword tracking and the number of logs that can be addressed are extremely limited, and the time required is considerable; often, potential problems are not found.

Unlike traditional approaches, in response to the maturation of new technology, the goal of this paper is to use ALAS (a combination of log uploads and problem returns using an Android application, wireless firmware update (FOTA) service, issue tracking systems, statistical analysis reports, and automated log analysis system (mashup system)) to improve the workflow of the development and testing personnel, where MTBF concept is used to establish quality indicators. The contribution of this study can be divided into the following three parts.

(i) Improving the Problem Return Mechanism. In the traditional field trial and beta user trial, the following are the steps of a return issue: (1) problems found by a tester; (2) logging in to the problem-reporting system; (3) dumping device logs; and (4) uploading device logs to provide a more convenient way to report issues and upload logs for testers. The ALAS system integrates the original Steps (2)–(4) in an application process after a problem is determined, and only a key combination is required to begin an application that can simultaneously report a problem and dump and upload logs.

(ii) Building an Automated Analysis Logs Mechanism. Unlike in the past, developers use a text editor to search log files in the ALAS system that were uploaded to the server. The automated analysis log mechanism can immediately process large user-uploaded logs and can achieve cross-platform and cross-unit processing bottleneck. Using an automated analysis log mechanism can effectively reduce the number of human errors.

(iii) Importing a Statistical Analysis Report Mechanism. Logs sent to the developer or tester in the past were often stored in his PC or placed on the file server, which is extremely inconvenient in terms of managing and searching logs. The ALAS system provides comprehensive log management, error statistics, and device and quality management with MTBF as indicators.

The rest of this paper is organized as follows. Section 2 discusses the relevant studies. The user scenario, system analysis and the design, algorithms, and data analysis are described in Section 3. Section 4 presents the implementation results, and the statistical data analysis is described in Section 5. The conclusion and future work are presented in Section 6.

2. Background

2.1. Android Log System. Android is a Linux-based, open-source operating system, which is primarily used in mobile devices, and has the highest market share in terms of the mobile device platform according to 2016 statistics [8]. The Open Handset Alliance (OHA, open handhelds Union), which Google established, continues its leadership and development. The latest version that has been released is Android 7.1 [9]. The Android system contains the log function to record messages by application; the system is shown in Figure 2. The logging system consists of

(i) a kernel driver and kernel buffers for storing log messages,

(ii) C, C++ and Java classes for creating log entries and accessing log messages,

(iii) a standalone program to view log messages (logcat),

(iv) ability to view and filter log messages from the host machine (via eclipse or ddms).

There are four different log buffers in the Linux kernel, which provide logs for different parts of the system. Access to the different buffers is provided by device nodes in the file system (/dev/log). The four log buffers are

(i) main: main application log,

(ii) events: for system event information,

(iii) radio: for radio and phone-related information,

(iv) system: a log for low-level system messages and debugging.

The log system described above, which originated from [10], and information on how to use the Android debugging tool, Dalvik Debug Monitor Server (DDMS), and how to use logcat command to read/write logs refer to the Android

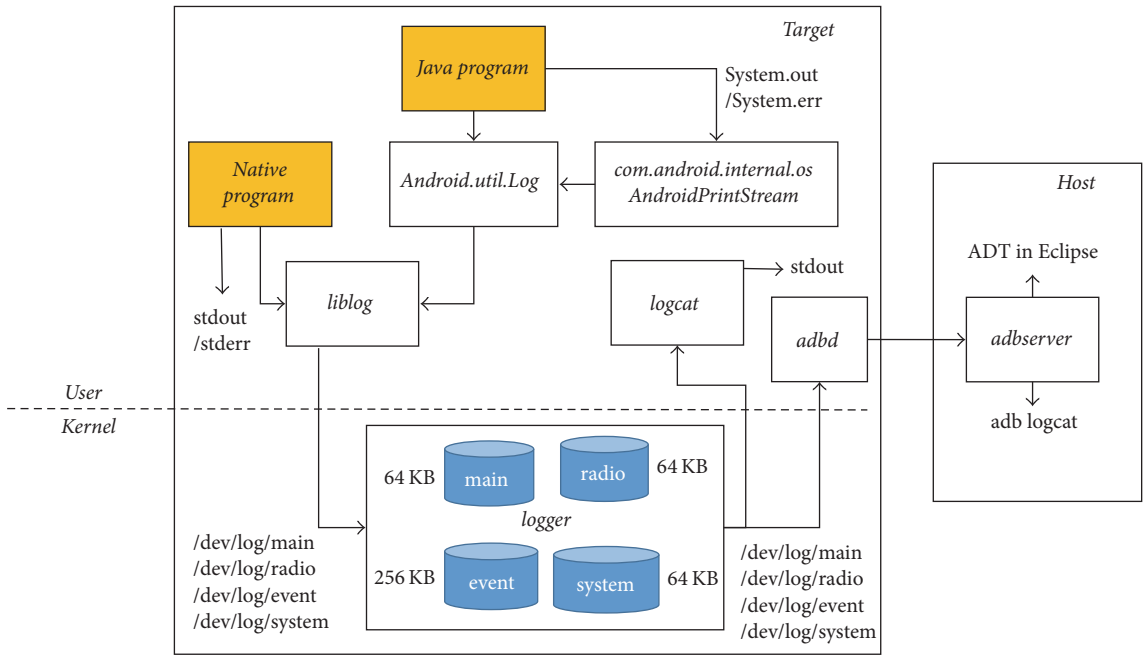


Figure 2: Overview of the Android logging system [7].

Developer website (http://developer.android.com/develop/index.html), where you can obtain rich, detailed information that will not be presented in this paper.

2.2. Mobile Log. The purpose of mobile logs is to record important information at the time of an event and the actions, status, and device settings, which are used in the development stage to reproduce problems and trace abnormalities. This is the major basis to fix bugs. When a problem occurs, in addition to what can be observed from the tester or camera, obtaining the background and the status of the device at the time relies on logs to understand the problem. Therefore, many studies on handheld device logs will overwrite the self-defined mechanism [3, 4] to obtain meaningful information to implement a system. In addition to Android itself and chip vendors logs, we also have additional files with the label (tag).

2.2.1. Traditional Way of Collecting Logs. Early in the collection of mobile logs, where a USB (universal serial bus) connection to the computer was typically used, after the logs files were uploaded to the FTP server or file server, because different machines were used, the installed software would be different; thus, log settings would differ. For example, testers will determine the general terms and CTS (compatibility test suite), which depend on the user version; monkey and other automated test items will be performed, which will also depend on the user-debug version. Occasionally, developers will be working on validation modem-related issues and will also use user-debug version. One of the differences that arise from using different versions is the different device logs; in the user-debug version of recording logs, the types and content will be more detailed. In addition to connecting a computer to dump mobile logs, which is conducted in academic research, the program can also use Bluetooth to collect test machine's logs data [3].

2.2.2. Modern Way of Collecting Logs. Presently, because mobile network transmission rates have improved dramatically, we do not have to dump logs to a computer because now mobile logs can be uploaded directly to where they need to be saved on the computer, which removes the aforementioned lengthy procedure, where, depending on the user version and user-debug version of the logs, content will differ. However, the greatest difference is the number of logs and file size; for recording modem logs, under normal use, during a single day, the log size will exceed 1 GB even if the current speed is 3G or WiFi is used to upload a large number of logs. This is an already difficult transmission problem without wireless networks being relatively unstable and prone to disruptions.

Therefore, this paper uses three methods to ensure the success rate and efficiency of uploading logs. First, we developed a software program to upload logs, which can be installed on mobile devices, where users can upload logs directly through the wireless network. Then, the user can use scripts to set or obtain the log file location. The user must know the location and dump log location to obtain and upload logs. The second method is that we readjusted the log mechanism; only necessary information is recorded to reduce the log size. In the third method, uploaded logs are compressed, and the log file is divided; for initial upload log files, only a simple packaged compression is conducted. However, occasionally, the size of compressed files is larger than 50 MB, which means the upload will sometimes fail. The program will subsequently and repeatedly attempt to upload logs,

Table 1: Error types.

Error type	Description
Freeze	Touch panel or unresponsive buttons
Self-shutdown	Device automatically shuts down or reboots
Unstable behavior	Device instability, for example, device lag
Output failure	Device's responses do not meet the external expectations, such as abnormal picture and abnormal sounds
Input failure	The user presses the button, and operating screen appears to not respond or there is abnormal operation or unexpected response

which uses up the user's network bandwidth. The success rate of a large file upload is lower than that of small file uploads; thus, we created the split-and-upload file mechanism, such that problems regarding uploading logs loss and failure rarely appeared; the users also observed less problems regarding this issue.

2.3. Error Type Analysis. In addition to using log records and collecting data, we must also track the type of error that occurred to classify the problem, facilitate statistical analysis, and improve the reliability and stability of the system [4, 11–14]. As described in [4], we used their defined error types, shown in Table 1.

2.4. MTBF Indicator. MTBF (mean time between failures) is the average number of working hours of a trouble-free product and is a measure of the product's reliability; its unit is hours. For example, if the MTBF value of a product is fifty thousand hours, then the MTBF test assessed that, after fifty thousand hours of normal operation, failure will not occur. Each product exhibits different behavior in different environments. Additionally, different handling and conditions will mean that product failure might not occur in a similar situation. Furthermore, between commercial and military products, product quality requirements will be different, and therefore, the MTBF standards will be different. In fact, the MTBF value is a result from software, which uses parameters to calculate the MTBF. The MTBF value is an important parameter, though it should be considered that, during product design, reliability engineers and designers often use a variety of methods and standards to estimate the MTBF value of a product. Related standards include MIL-HDBK-217F, Telcordia SR332, Siemens Norm, Fides, and UTE C 80-810 (RDF2000). However, with these methods, there is still a considerable gap between the estimated value and the actual MTBF value [15].

Because this system uses logs as a data source to develop and build logs, in research papers, the MTBF [12, 16–21] as a quality index is commonly used because the MTBF reliability value can effectively reflect the product under testing. Additionally, an adjustment is often calculated to create a custom MTBF [4, 12] because of the importance cited in paper [4] regarding freeze errors and the device automatically rebooting (self-shutdowns) and also to establish its corresponding index MTBFr and MTBS.

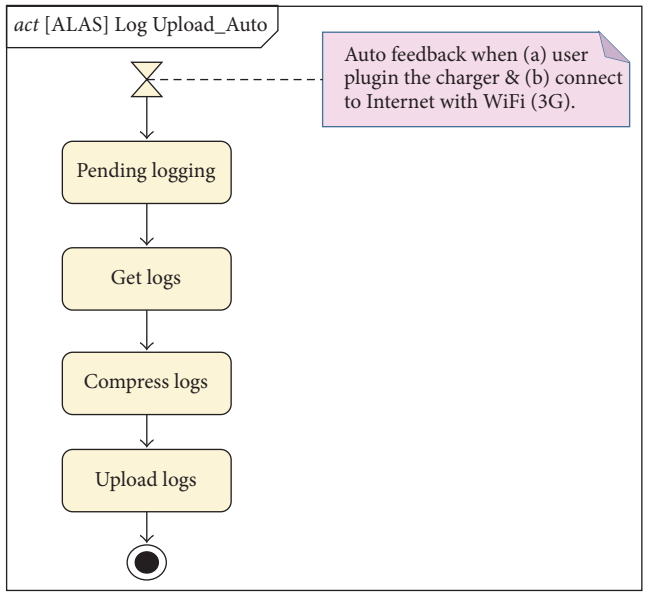


Figure 3: Log upload activity diagram (automatic upload).

3. System Analysis and Design

3.1. Scenarios. After the user installs the log uploader application, in accordance with the operating condition of the system, the program can perform two tasks: the device can automatically upload logs or the user can manually return to the problem, which will be described below.

3.1.1. Automatic Upload. Joyce is a housewife, who does not often use information products and only knows the basic operation of her phone. She obtains a beta user trial phone, and, as long as she charges the phone every night, the phone automatically uploads the system logs of the day for analysis, as shown in Figure 3.

3.1.2. Manual Upload. Tony is a computer engineer, who gets off work and takes his MRT. Tony uses the phone to watch YouTube, and, then, YouTube suddenly crashes when playing. The program is forced to close. Then, the log uploader can be used immediately, and the phone returns to the problem and uploads the logs, as shown in Figure 4.

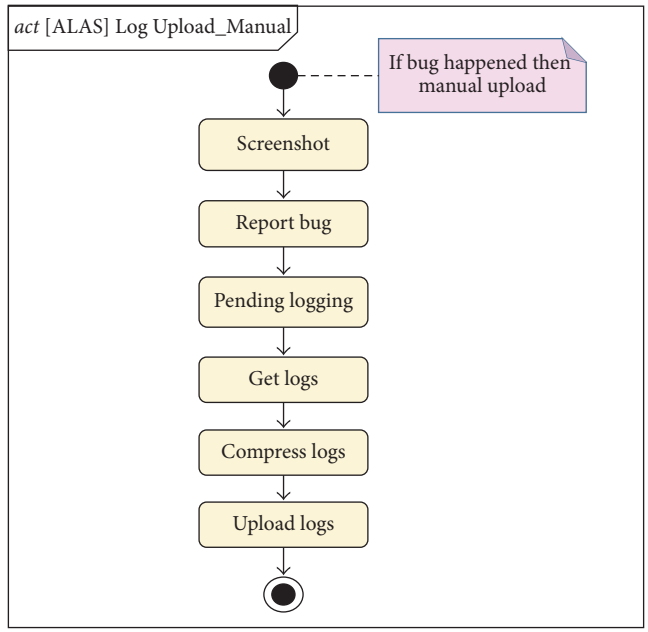


Figure 4: Log upload activity diagram (manual upload).

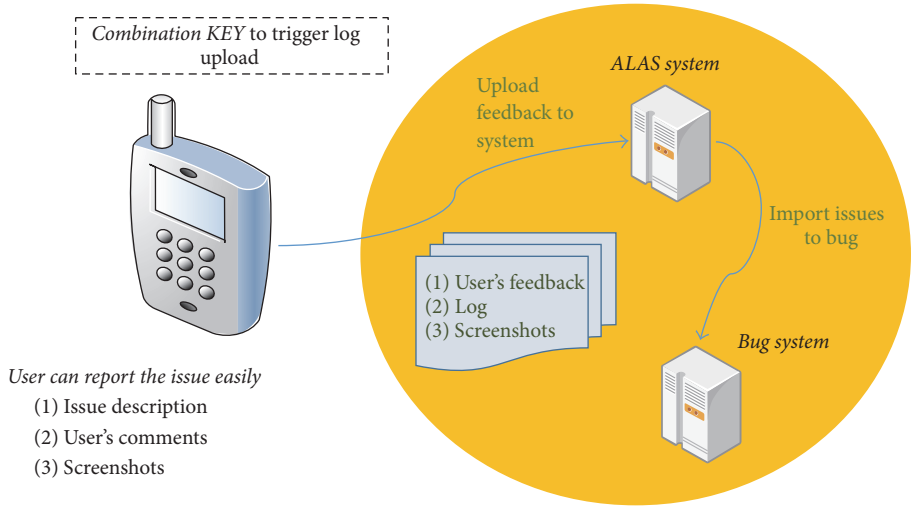
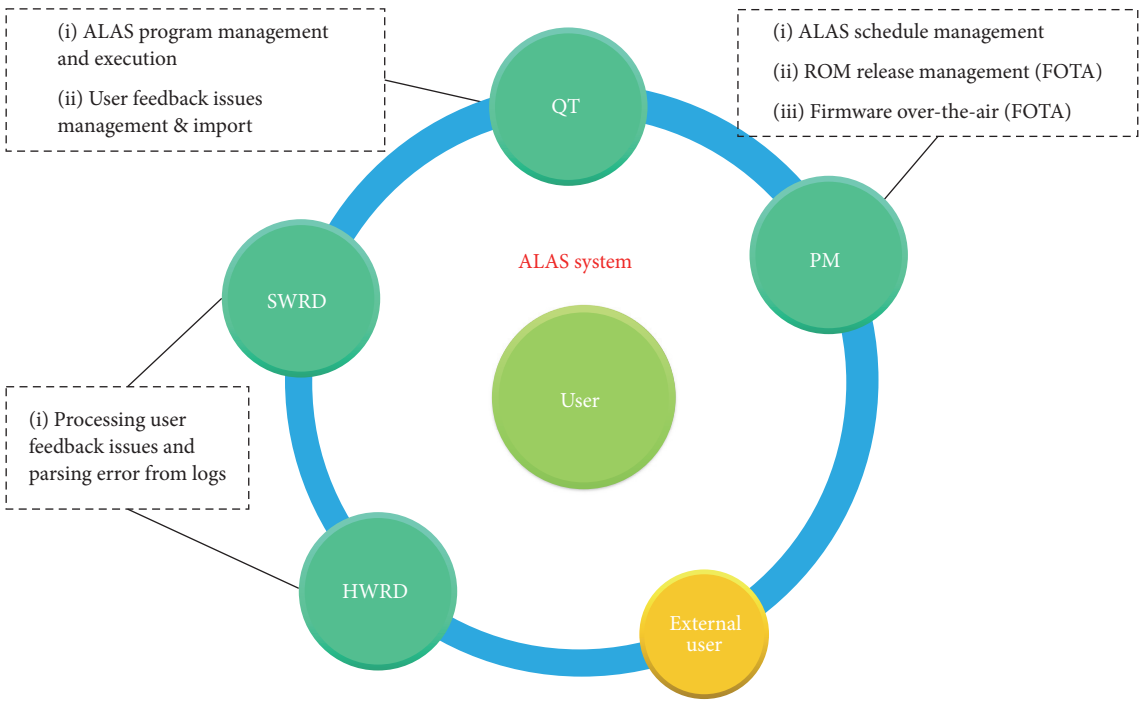


Figure 5: Architecture diagram.

3.2. Architecture Analysis. The ALAS system, in accordance with the overall process architecture point of view, can be seen as a client-server system (Figure 5). The client-side task is installed on the phone through the log uploader application to finish uploading the reported issue logs, and, in the server side, the work contains the log processing, log downloads, sent questions, and statistical reports. In Figure 6, we can clearly understand the user of each unit's role and tasks given. In Figure 6, there are four roles in ALAS system, namely, Quality Tester (QT), Project Manager (PM), Software Research and Development Engineer (SWRD), and Hardware Research and Development Engineer (HWRD). QT is primarily responsible for testing and execution of ALSA programs, as well as managing feedback from external users. PM is the manager of the ALSA project and is responsible for the ROM release management and the firmware over-the-air. SWED and HWRD are actually responsible for solving the problems from user's testing feedback in logs. The interaction between the user and the work modules is organized into the use case diagram (Figure 7).

3.2.1. Client Side. If the handheld device uses an Android system, the log uploader application on the client side can be installed. Logs are uploaded using HTTP protocol using our application, which requires the user to dump, package, and upload logs. These actions and changes can be accomplished by the press of a button, and to ensure the success rate of uploaded logs, the logs are compressed, and the logs will



(i) QT: Quality Tester
(ii) PM: Project Manager
(iii) SWRD: Software Research and Development Engineer
(iv) HWRD: Hardware Research and Development Engineer

Figure 6: Roles and task analysis diagram.

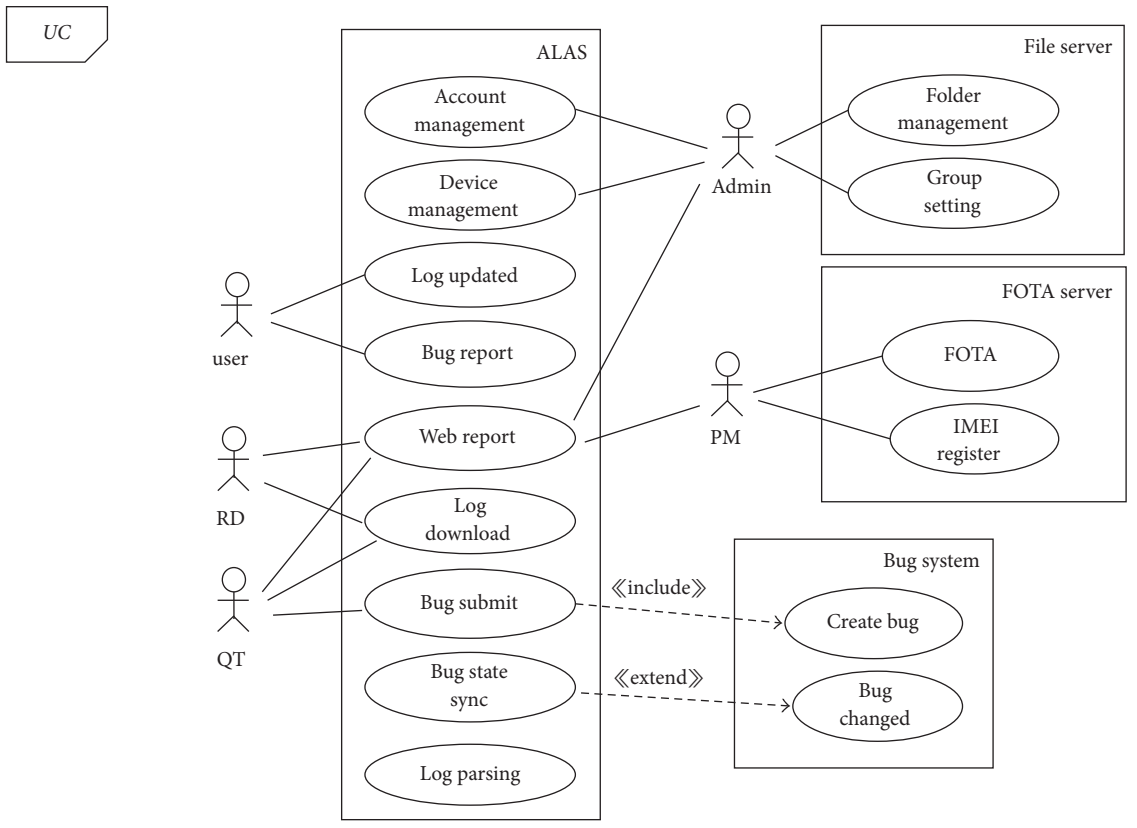


Figure 7: Use case diagram.

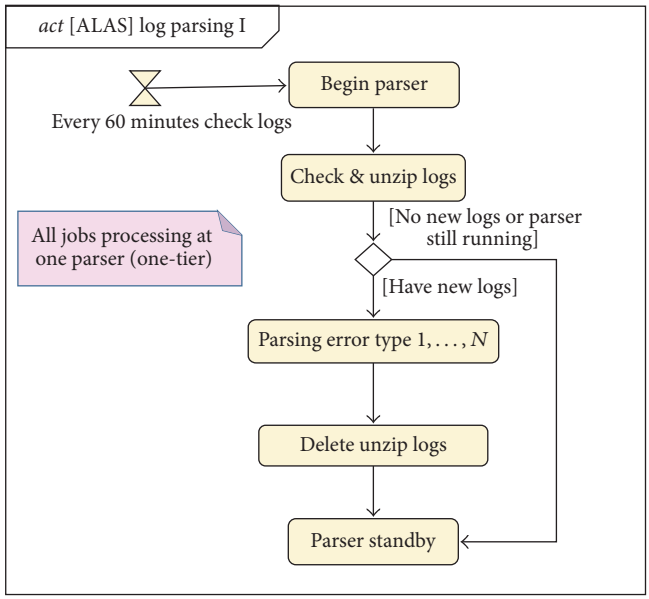


Figure 8: Log parsing (one-tier) activity diagram.

be uploaded at different times to avoid network congestion with a simple user interface, which allows issues to easily be reported. Later, in Section 4, the operation will be described.

3.2.2. Server Side. The primary work of the server side can be divided into the following: receive logs, log analysis, log storage, log downloads, database access, statistical reports, and transmission to the client-side log archive. We use an MS Windows Server 2008 R2 for storage and manage through the group permissions and shared directory to help users download logs. We used IIS 7.0 to build our web site, where the database uses MS SQL Server 2008 R2. The functional modules, which analyze the statistical reports, will be introduced in the next two sections.

3.3. Log Parsing Module Design. Because of the different units and products from chip manufacturers, their log format and content are not the same. Furthermore, for cross-platform use, that is, not limited to only the Android system, the iOS system and Windows system must also be able to integrate with the same system. Besides the fact that log uploaders should be rewritten according to different operating systems, the greatest difficulty is in the analysis module. The analysis module must handle the different types of logs for analysis, use the same error definition, and write to the database to facilitate the log parser development. To not be limited by the programming language during development, our server-side work has adopted the call executable manner to facilitate the different programming languages used by developers to join the parser development.

Logs analysis of the entire process definition is as follows: (1) Merge the uploaded compressed split file; (2) unzip the logs to a specified directory; (3) archive the compressed log file; (4) execute each unit's parser main; (5) execute each unit's error analysis (error type) module; (6) allot 30 minutes to reexecute Steps (1)–(5) to achieve these demands using the following four modules, where the overall log analysis process is shown in Figure 10.

(i) Parser Daemon Module. It is responsible for handling Steps (1), (2), (3), and (6). When performing Step (1), it will first be confirmed whether the previous round of logs was processed. If a unit is still running the parser, this unit will be skipped. The function of the daemon is to start and stop the parsing service and address several common processes. Information that can change and affect the configuration of each unit can be used in this module, and this module can initialize the operation and control the parser main module.

(ii) Parser Main Module. It is responsible for Step (4) and primarily controls the error type module for each unit because logs are different in each unit. Certain programs that require separate processing can be performed here.

(iii) Error Type Module. It is responsible for processing Step (5). The parser dismantles four modules. One advantage is that you can rapidly develop a new error type to be online running as long as it is new or modified. We can separate the development of an error type such that it will not be linked to other programs to achieve rapid development.

(iv) Common Function Module. The program uses various functions to integrate into the DLL (dynamic link library) and help when the process, architecture, or database is modified, which can be unified together, including database insert/update/delete/query, invoking a web service, recording error messages, and file compression and decompression.

(v) Evolution of the Log Parsing Module. The functional architecture of the log parsing module had already been described. The initial design of the system architecture is shown in Figure 8, and Figure 9 shows the system architecture after

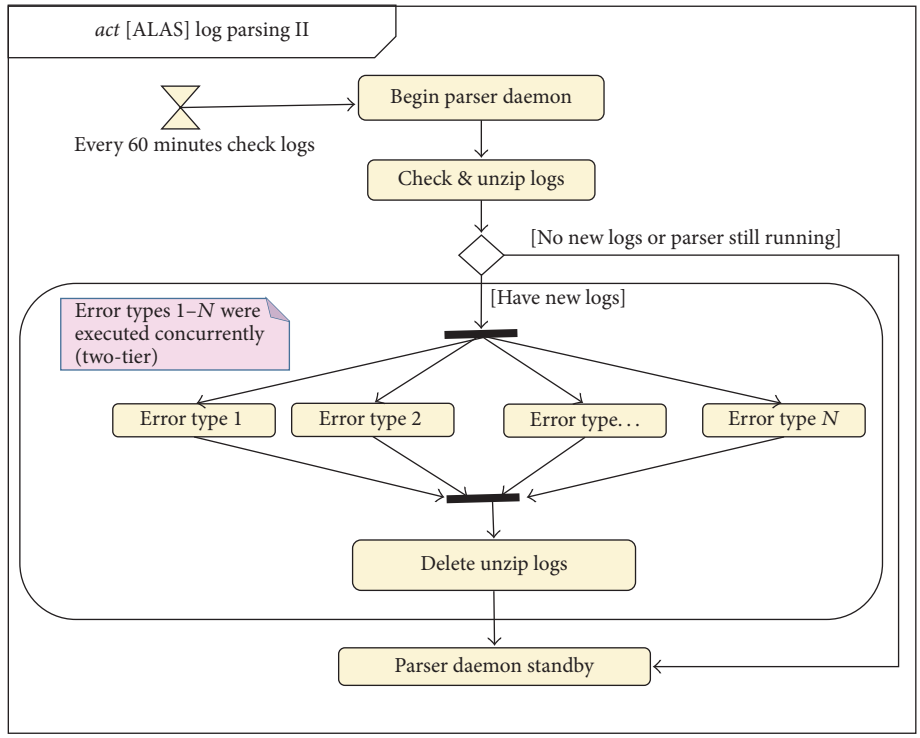


Figure 9: Log parsing (two-tier) activity diagram.

evolving. Finally, Figure 10 shows the final architecture chosen.

As shown in Figure 8, the server-side program is first used by the parser, which is responsible for processing all the work in the same program. The primary reason is that, in the early development of the system, in order to facilitate rapid development and reduce system development difficult, we put all the work of the server-side unified into a single program to deal with it. Every 60 minutes, the parser program checks for new uploaded logs to the server. If the system obtains a new log, the program will begin to unzip the file and perform log parsing. After completing all the work, it will return to standby and wait for 60 minutes for the next round.

However, for this architecture, there are two drawbacks. The first drawback is that the performance time will be relatively long because all the work is processed by the same program, which means the order must be followed in a step-by-step execution. The second point is that the maintenance program is inconvenient. Each error type is parsed in the same program, which means the management and maintenance will be inconvenient; whenever there is a small modification, the parser version will change. To ensure that the other error type code is not modified, we changed the parser from a one-tiered architecture to a two-tiered architecture, as shown in Figure 9.

In the two-tiered architecture, we will divide common jobs in the parser daemon, such as unzipping log files, moving files to their specified director, and deleting files. By having the parser daemon control the start of each error type program, the end of each error type program will notify the parser daemon. The parser daemon will wait until after the end of all the error type programs and will then perform the last action (delete decompressed archives). The parser daemon will then return to standby and wait 60 minutes for the next round.

There are three drawbacks with the two-tiered architecture. The first drawback is only applicable in the use of units. Each unit's log name, log location, and error type rule are different. If there are two units in the parsing rule development in a two-tiered architecture, the code for the two units would have to be the same. Care must be taken to ensure that the other code is not modified, which means code maintenance is a difficult task. The second drawback is that its confidentiality is poor because each unit parsing pule is confidential, and so the other units cannot know and, therefore, cannot be in the same program. The third disadvantage is that it is difficult to import a new unit, which increases the programming difficulty in the two-tiered architecture. The code of each unit will be mixed together, and therefore, when a new unit is imported, more time is required to explain the program logic and process workflow.

To solve the aforementioned drawbacks, a three-tiered architecture is used, as shown in Figure 10. The parser daemon is responsible for all the common jobs of the units. The parser daemon does not control the error type program. The control of the units' parser main was changed such that each unit's parser main controls their error type program. Thus, the three-tiered architecture promotes the imports of new units as long as these units are responsible for maintaining their own programs (parser main and error type). These units can modify their code and version, where their own common jobs are processed by their parser main. The error type program

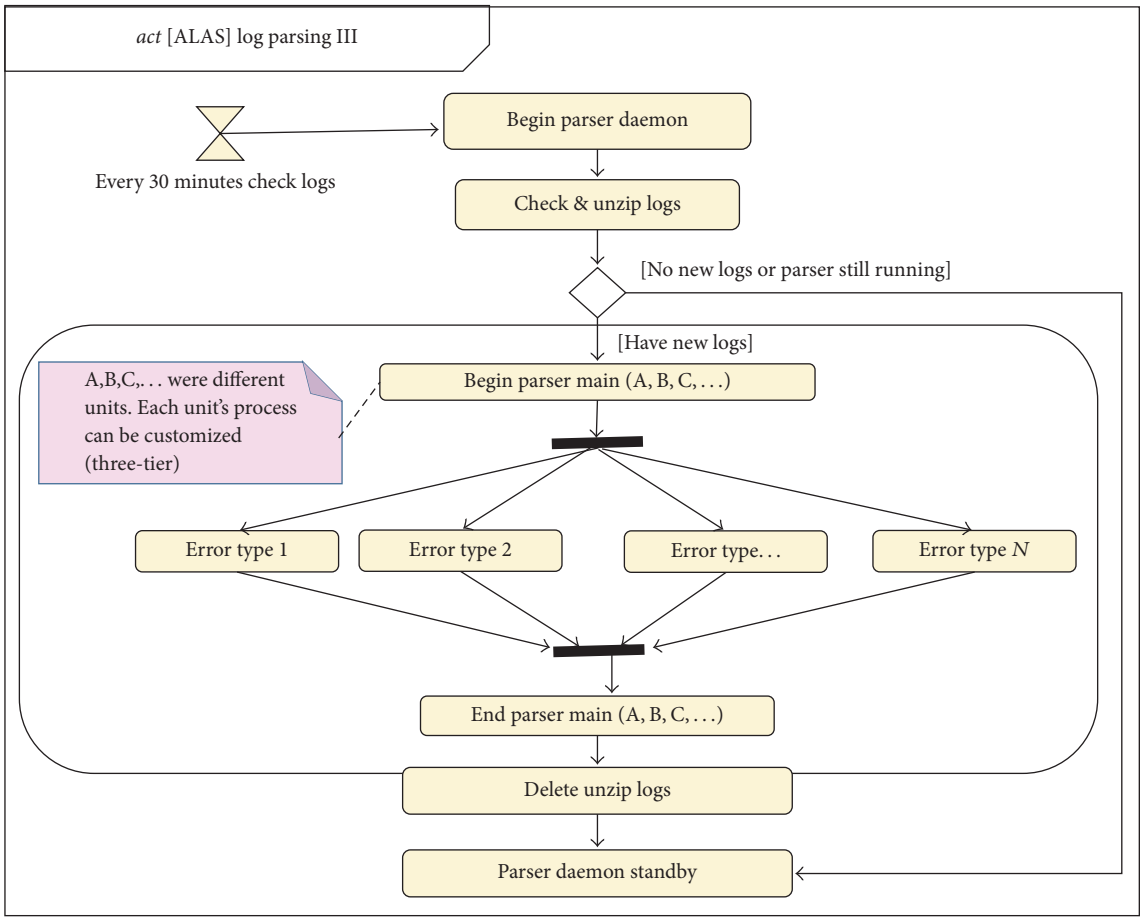


Figure 10: Log parsing (three-tiered architecture) activity diagram.

only parses the errors, and thus, the program is simplified and easy to program and maintain. Because more units will inevitably join, the number of projects will also increase. To immediately process logs, we adjusted the waiting time to 30 minutes.

3.4. Analysis of the Activity Diagram. In this system, there are two primary flows and a secondary flow. The main flow is divided into uploading mobile logs (Figures 3 and 4) and parsing logs (Figures 8–10). The secondary flow sends questions to the bug-tracking system (Figure 11). The management and reporting system manages and provides information, which has nothing to do with the flows; this is not explained by the activity diagram.

A tester will check for new issues every day. The system automatically filters duplicates that have occurred, which are then submitted from the tester to the bug-tracking system. The system will perform daily timings to synchronize the status from the bug-tracking system, as shown in Figure 11.

3.5. Web Report Module. We use the web as the interface for data management, and in accordance with the system design, the following are the modules used.

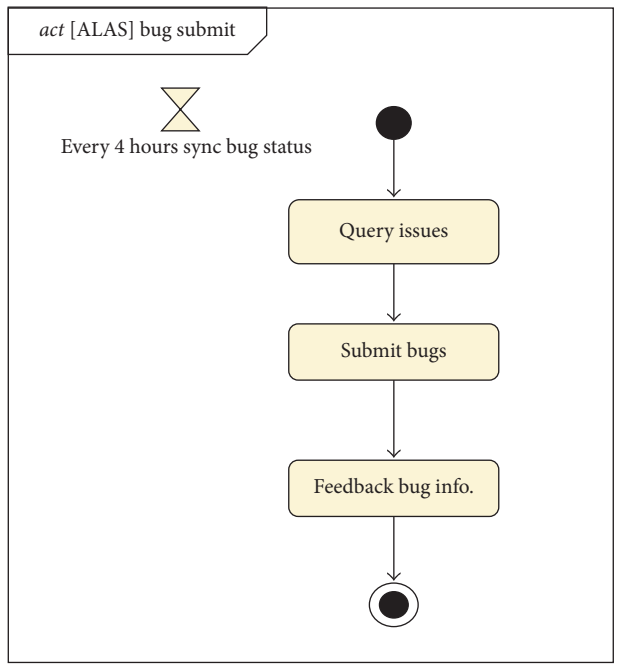


Figure 11: Bug-submission activity diagram.

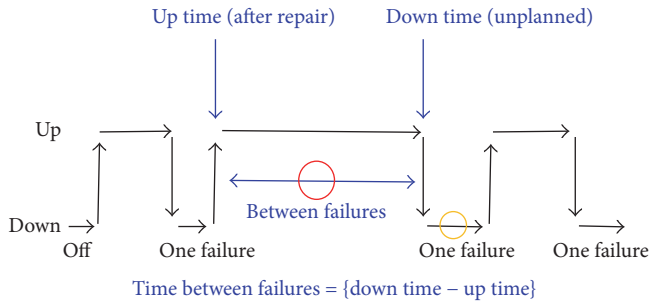


Figure 12: MTBF mathematical definition and time schematic diagram.

(i) Log Receive Module. HTTP is used to upload the log archive in accordance with each product to be placed in the corresponding path.

(ii) Authentication Module. Lightweight directory access protocol (LDAP) is used to integrate the account, the user can log into the system using a domain account. Users do not have to remember a set account and password to use the system. Additionally, login is simple; administrators can save information regarding the account creation and management.

(iii) Device Management Module. It manages the release of the test device and user. To record this information for control convenience and to reproduce a bug, a user can also find the bug report.

(iv) Error Summary Module. It contains the statistical analysis charts of each error type.

(v) MTBF Report Module. It contains error information regarding the MTBF and establishes statistical information that can be queried by project/device/error type.

(vi) Log Search Module. It allows the user to view the status of uploaded logs and download paths.

(vii) Error Query Module. It provides the user with the query error type page.

(viii) Application Using the Record Module. It contains statistics regarding the user device application, that is, number, time, and other information.

(ix) Power Consumption Report Module. It provides users with long-term battery use status and uses the information to determine if there are abnormal consumption phenomena.

(x) Privilege Management Module. It manages user account permissions settings, edits basic information, and manages roles.

3.6. Algorithms and Data Analysis. This section describes the use of the MTBF in the system's application and describes the error type definitions.

3.6.1. MTBF. MTBF is widely used in reliability analysis indicators in manufacturing. The MTBF indicates the average product use during continuous operation or the test time to failure. Note that the MTBF is not a measured value; during the product design phase, engineers estimate the MTBF based on the theory of reference values [15, 22]; refer to Wikipedia for additional details (Figure 12).

A classic MTBF value does not mean that the system will appear physically damaged; this will depend on how failure is defined for a system. For example, in complex systems that require high reliability, failure may mean that an unexpected situation occurs and that the system must be stopped and repaired. Good maintenance helps to avoid failure, where failure can also be a direct result of equipment being decommissioned; however, the need to stop the device for planned maintenance work is not considered within the scope of this definition of [15, 22].

Using the definition in the previous paragraph, when we calculate the MTBF, the actual demand and the system must define failure. In addition to using software and parameters to calculate the MTBF, in industry, the MTBF stability test for handheld devices shifts to the long-term implementation of the selected test items in the laboratory to calculate the MTBF. For example, at the Chinese telecommunications company, China Mobile, and the American telecommunications company, AT&T, the MTBF testing that will be performed by testing machines from the company's laboratory will continue for a long time. Test items include the telephone, SMS, e-mail, file upload/download, use of the browser, and network connection. The overall use of an automated testing process used by the company's network ensures a consistent user environment. According to the tested machine, the time divided by the number of machine errors is the MTBF test result. MTBF testing at each company is different, and the test item content, test methods, calculation methods, and passing criteria are confidential at each company.

$$\text{MTBF} = \frac{\sum(\text{start of downtime} - \text{start of uptime})}{\sum \text{number of failures}}. \quad (1)$$

In Section 2, we have referred to logs as data sources in research papers. The MTBF is calculated based on an actual test time in the numerator and the number of errors in the denominator. The implementation of MTBF testing is the

Table 2: Error types.

Error type	Description
Unknown	Uncategorized question or one that cannot be discriminated
Freeze	Touch panel or buttons unresponsive
Self-shutdown	Device automatically shuts down or reboots
Unstable behavior	Device instability, for example, device lag
Modem crash	Modem-related problems
ANR and App force close	Application is not responding or forced to close
Battery loss	Abnormal battery power consumption
Sensor	Device sensing element anomalies or failures
Device overheats	Device overheating problem
Output failure	Device's responses do not meet the external expectations, such as abnormal picture and abnormal sound
Input failure	User presses a button, and operating screen appears to not respond or there is abnormal operation or unexpected response

same in a laboratory as in industry. In this paper, the actual test execution time is also used to calculate the MTBF. A good or bad MTBF depends on the results and is the sum of the length of time and number of failures. Generally, the system designer self-adjusts the definition. In this system, because the phone downtime is typically extremely short, there is essentially no change in the device usage time (use + standby), shown in (2) below. When we collect logs, the unit of the recorded machine time is an hour, which is the same as the general industry standard; however, at the time that this report was created, the unit was changed to a day. In a report presented in such a way, this is more intuitive, which allows the user to gain a better understanding.

$$\text{MTBF} = \frac{\sum \text{Device Usage Time}}{\sum \text{Device Report Errors}}. \quad (2)$$

3.6.2. Definition of Error Type. In this section, we clearly define several types of handheld device errors that often occur, which are described in Table 2.

Due to the parsing rule of the error type being confidential information, we will not list its instructions. The following are descriptions of two types of errors that may not be included in the MTBF.

(i) Unknown. An unknown error occurs when the error only appears when the user manually returns to the problem, where the issue type is selected as "Other." The user cannot confirm how the issue should be classified to which error type because mobile phone users likely do not understand the behavior or certain users use the experience feedback. Suggestions should therefore not be included in the MTBF calculations.

(ii) Force Close of Application and ANR. Applications are forcibly closed or there is no response. This error typically has the highest proportion in terms of reported bugs. However, there are many applications that the user installs that are not in the original shipment when the preapplication set is installed. Errors that occur due to these nondefault application installations should not be included in the MTBF calculations because we are not the nonoriginal application developers; therefore, we cannot fix these problems. Thus, a system that includes only the factory-default applications and GMS (Google Mobile Service, such as Gmail and YouTube) error correction should be included in the MTBF calculations.

4. Implementation

According to Section 3 on the development and needs, the functions we created are introduced individually. The system architecture can be divided into the mobile program development, parser program development, web development, database development, and data connection interface development.

(i) Mobile Program Development. The Android system uses JAVA as the programming language in the script to obtain the required logs. The compressed files are then cut and packaged into compressed files, which are then used for data upload by HTTP.

(ii) Parser Program Development. Because the log storage and upload server uses MS Windows Server 2008 R2, thus, for the parser program, we use Visual C# and use XML file as an external profile.

(iii) Web Development. Web development uses ASP.NET C#, and the website service uses management, and reporting interface and the receiving logs from the device uploads use MS IIS7.0.

(iv) Database Development. To record logs, errors, devices, reports, and other related information, the database uses MS SQL Server 2008 R2 and uses the trigger and view to assist in the development, parser, and website; thus, you can use

the traditional SQL-connection or web service to access the database.

(v) Data Connection Interface Development. There are two methods to connect with external systems. One method calls the LADP query to achieve a single integrated account, and the other method uses the web service to access the ALAS and the data connection and synchronization with the issue tracking system.

In accordance with the above requirements and architecture planning, we have completed and implemented the following system functions and modules.

4.1. Mobile Operating Instructions. User scenarios described in Section 3 mentioned the automatic upload and manual upload processes; now, further explanations will be given.

4.1.1. Automatic Upload

Step 1. Meet the following two conditions so the system will automatically upload daily logs. The first condition is that the mobile must be charged. The second condition is that the mobile must be connected to the Internet.

Step 2. In addition to these two conditions, the system will automatically package daily logs every day.

Step 3. The mobile's default setting is to use the WiFi connection to upload logs, where the 3G upload is set separately to limit the data use of users; otherwise, an additional accounting burden would be generated.

Step 4. If the device has not uploaded logs, meet the two conditions of Step 1, which will attempt to reupload.

Step 5. For each time packing, compressed files that are larger than 5 MB will be cut to ensure the success rate of the uploads.

4.1.2. Manual Upload

Step 1. When problems occur and the user wants to manually report the problem, a key combination can be used to begin the application.

Step 2. There are three fields to fill, "Issue Type," "Comment," and "Reporter." The "Issue Type" field is drop-down menu to select the mapping error type, the "Comment" column contains the operation at the time of the error and an issue description is given, and the "Reporter" column asks for the reporter name.

Step 3. Finally, pressing the submit button completes the return process, and the upload history can be viewed at the HISTORY page.

4.2. Implementation of the System Page. The login screen is shown in Figure 13. Through the LADP, a single integrated account is created, which makes it easier to perform the initial and account maintenance of the system. Figure 14 shows a schematic diagram of a user's computer, the ALAS system, LDAP server, and database server.

Figure 13: Web report login screen.

We will now introduce how to implement the functions of the ALAS system (Figures 18–36), including the devices, logs, errors, MTBF statistics report, system settings, and permissions management page.

(i) Device List. This page (see Figure 15) allows managers to add/modify/delete/query device information (query fields: IMEI, USER_CATEGORY, PROJECT).

(ii) Device Usage Detail List. This page (see Figure 16) allows managers to query the device in various versions, device total hours, usage hours, and other relevant information to understand the state of the device (query field: PROJECT, SKU, USER_CATEGORY, FW_VER, IMEI, and user name).

(iii) User Manager. This page (see Figure 17) allows managers to add/modify/delete/query user relevant information and set the corresponding roles to the user (query fields: DIVISION, ROLE, and KEYWORD (USERNAME)).

(iv) Role Manager. This page (see Figure 18) allows managers to add/modify/delete/query role relevant information and set the corresponding role permissions (query fields: DESCRIPTION).

(v) Permission Manager. This page (see Figure 19) allows managers to add/modify/delete/query permission relevant information and set the corresponding web page name to permission (query fields: CATEGORY, KEYWORD).

(vi) LogMain List. This page (see Figure 20) allows users to query uploaded logs with relevant records, including the device suspended and usage time of the log (query fields: PROJECT, KEYWORD (FILENAME or IMEI)).

(vii) Error Detail List. This page (see Figure 21) allows users to query error detail relevant information, including the log download path, error description, bug id, and bug link (query fields: PROJECT, SKU, FW_VER, ERROR TYPE, SUB ERROR TYPE, USER CATEGORY, IS MTBF ERROR, ERROR DATE BEGIN ~ END, LOG DETAIL ID, and KEYWORD (IMEI, ERROR DETAIL, BUG ID)).

(viii) Power List. This page (see Figure 22) allows users to query long-standing power consumption records, including

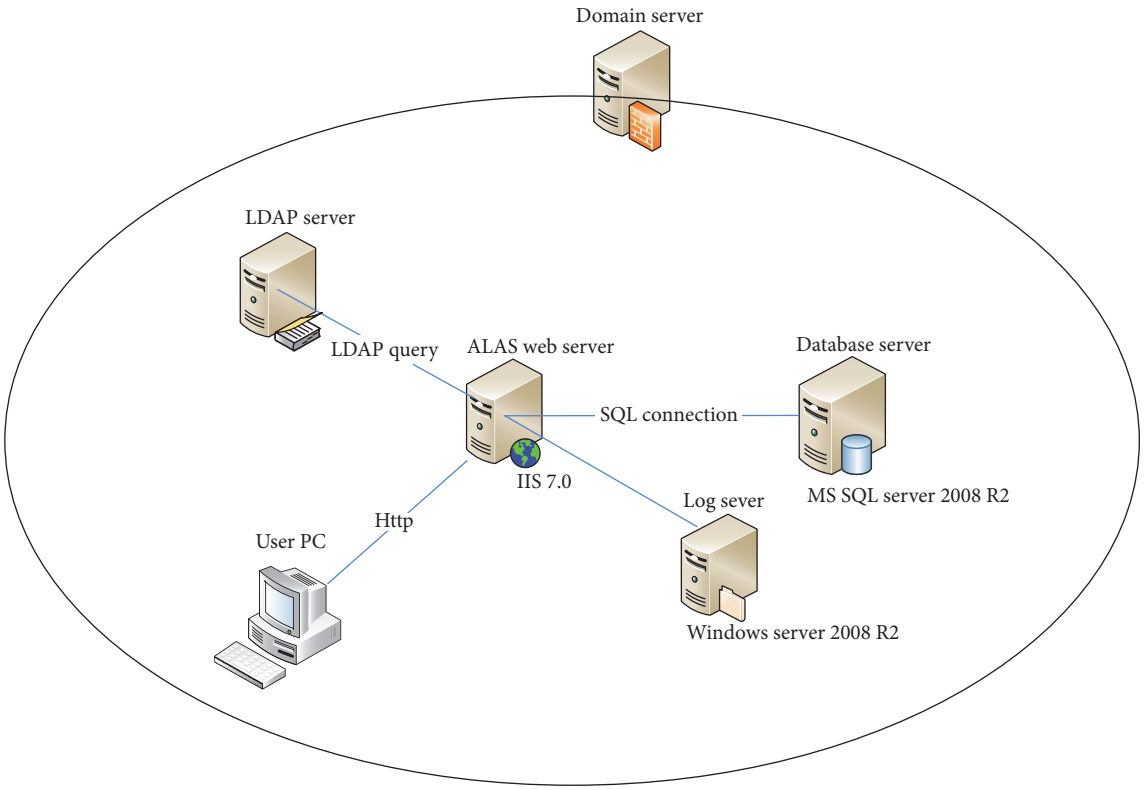


Figure 14: LDAP schematic diagram.

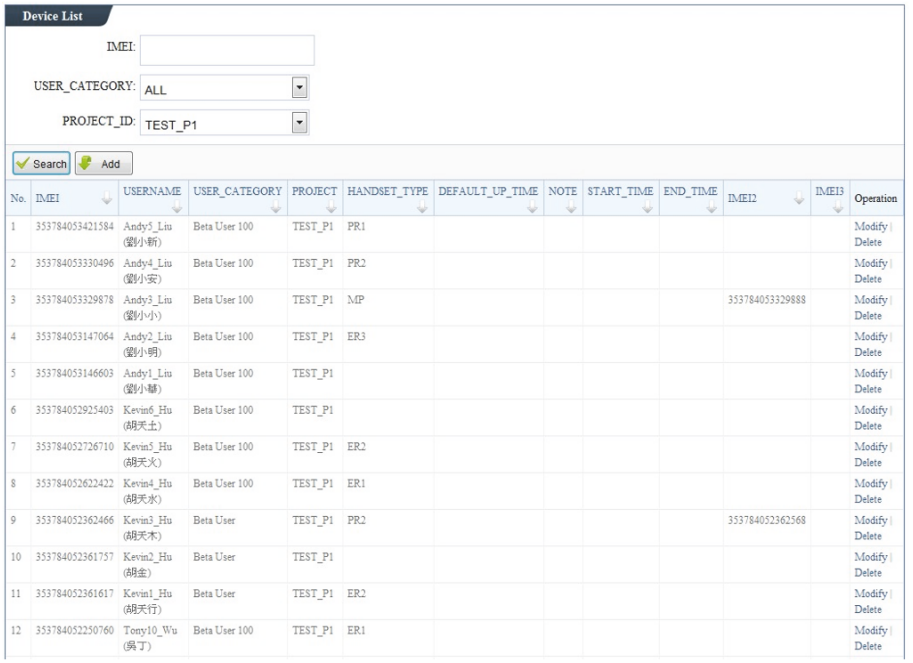


Figure 15: Device management page.

the standing start and end time, and power consumption and proportion (query fields: PROJECT, SKU, VERSION, CONCLUSION, START TIME, and KEYWORD (IMEI)).

(ix) MTBF by Project. This page (see Figures 23 and 24) allows users to query MTBF statistical data and charts (by version), including version information, device total hours, error counts, and MTBF results (query fields: PROJECT,

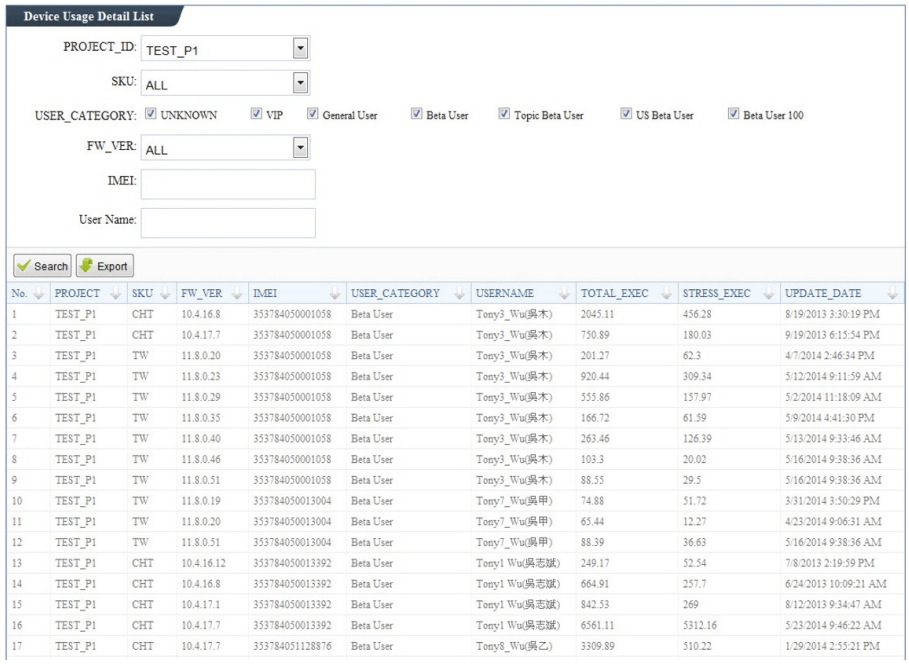


Figure 16: Device usage query page.

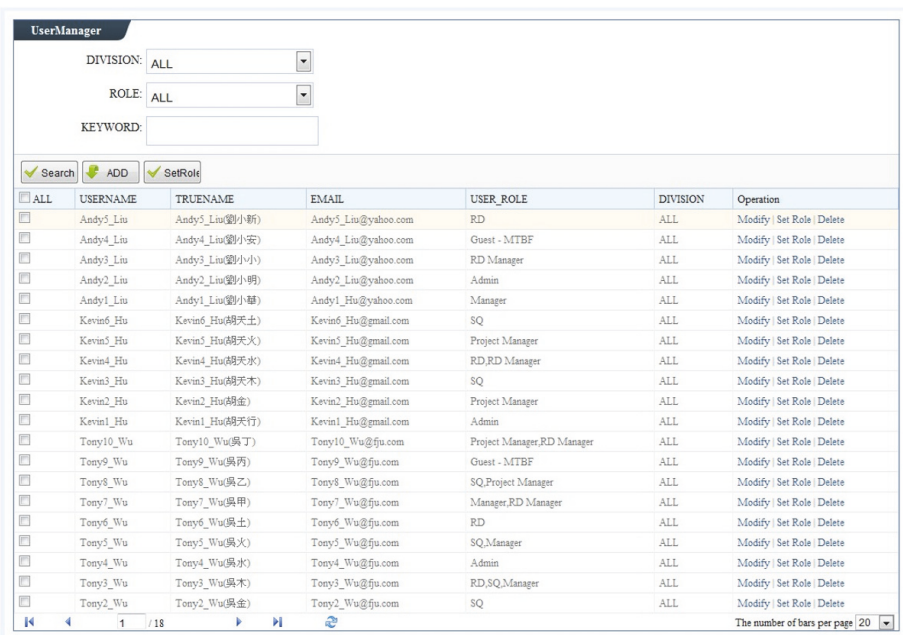


Figure 17: User management page.

SKU, VERSION, USER CATEGORY, TOTAL EXEC, and Last Update Date).

(x) Package Usage Summary. This page (see Figure 25) allows the user to query the application usage data and charts, including the Package Name, Usage Count, and Usage Time (query field: Using Count, Using Time, and Display Top). The application usage charts by count and hours are shown in Figures 26 and 27, respectively.

(xi) Package Usage by Project. This page (see Figure 28) allows users to query the project application usage data and charts, including the Package Name, Usage Count, and Usage Time (query fields: PROJECT, SKU, USER CATEGORY,

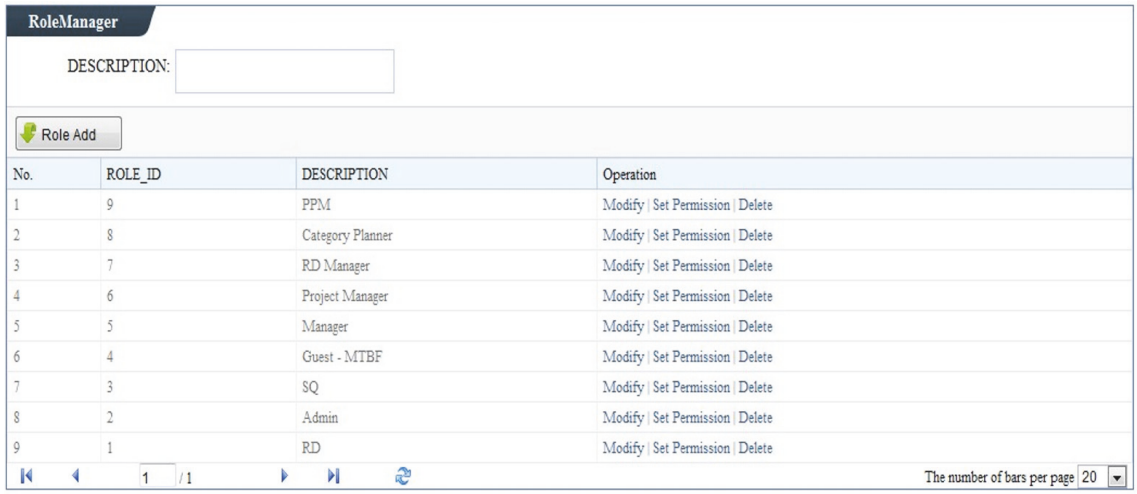


Figure 18: Role management page.

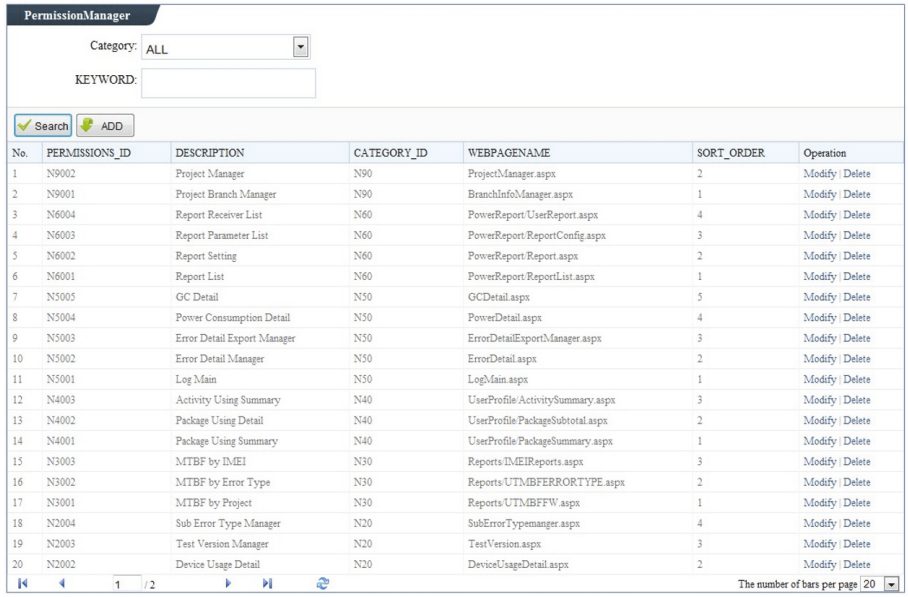


Figure 19: Permission management page.

VERSION, and Display Top). The project application usage charts by count and hours are shown in Figures 29 and 30, respectively.

(xii) Activity Usage Summary. This page (see Figure 31) allows users to query activity usage data and charts of the package, including the Package Name, Activity Name, Usage Count, and Usage Time (query field: Package Name, Usage Count, Usage Time, Display Top, and Keyword). The activity usage charts by count and hours are shown in Figures 32 and 33, respectively.

4.3. Implementation of the Parser Program. The log parsing module was described in detail in Section 3. Here we introduce the practical implementation of parser programs. The parser daemon is the primary control program and is responsible for calling and controlling various units of the log parsing program. After the program is complete, it will wait for 30 minutes; then, for the next round of log parsing, the execution screen shown in Figure 34 displays the current wait time in the implementation process. The screen also displays the current status of the program address logs, such as the errors and exceptions that have occurred. Because the current status is immediately known on the screen, a user can fix a problem immediately and also record the execution to log files to help the manager track the status of the parser program and help in debugging.

Figure 35 shows the execution screen of the parser main after obtaining certain common information. The program will wait for the error type execution of its department to be

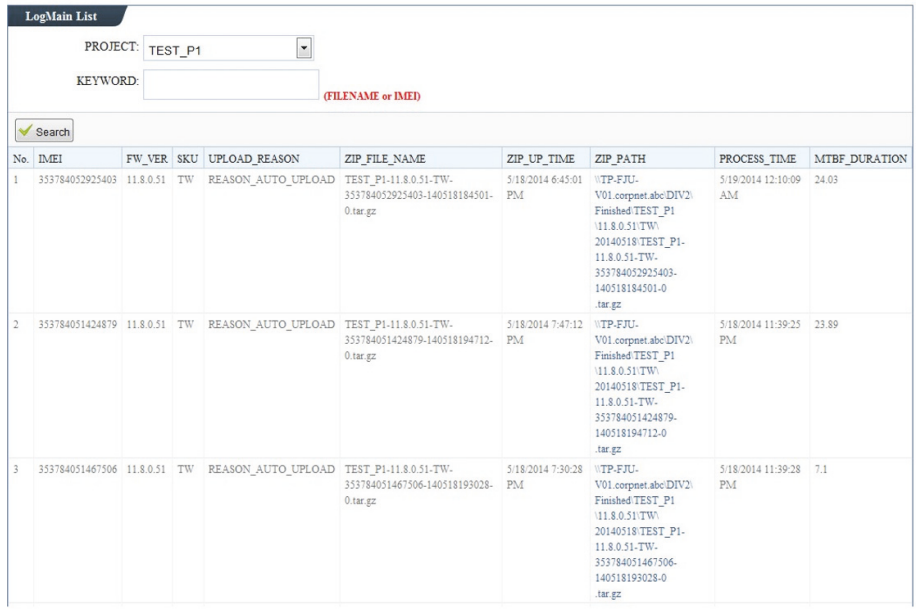


Figure 20: Log upload record query page.

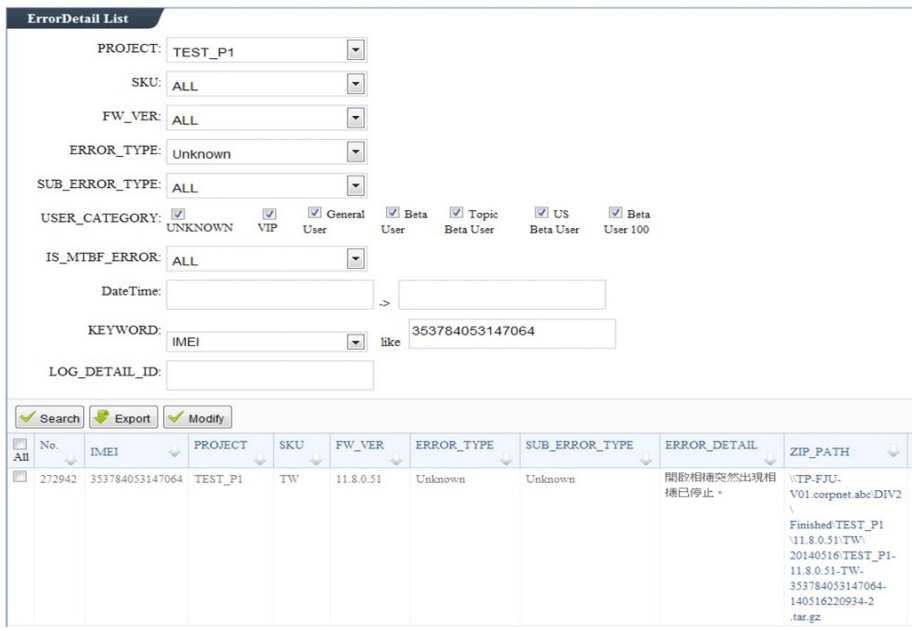


Figure 21: Error query page.

complete by checking every three minutes until it is finished. The parser main will execute the end job and notify the master program.

Figure 36 shows the execution screen for error type 1. Figure 37 shows the error type 1 log file. Because the error type program's implementation modalities and log file formats are the same, we used error type 1 as an explanation example. From the image, we can see that, when parsing logs, the executing subprogram and the log path will be marked. If there are errors or exceptions, they will be printed on the screen; the same content will be immediately written to a log file for tracking or querying records.

5. Statistical Analysis

In this chapter, we will list the project statistical data and charts after a test version run to illustrate the effectiveness of the system. Officially, the system has been online now for more than one year and six months, has performed more than twenty projects, and has tested more than three

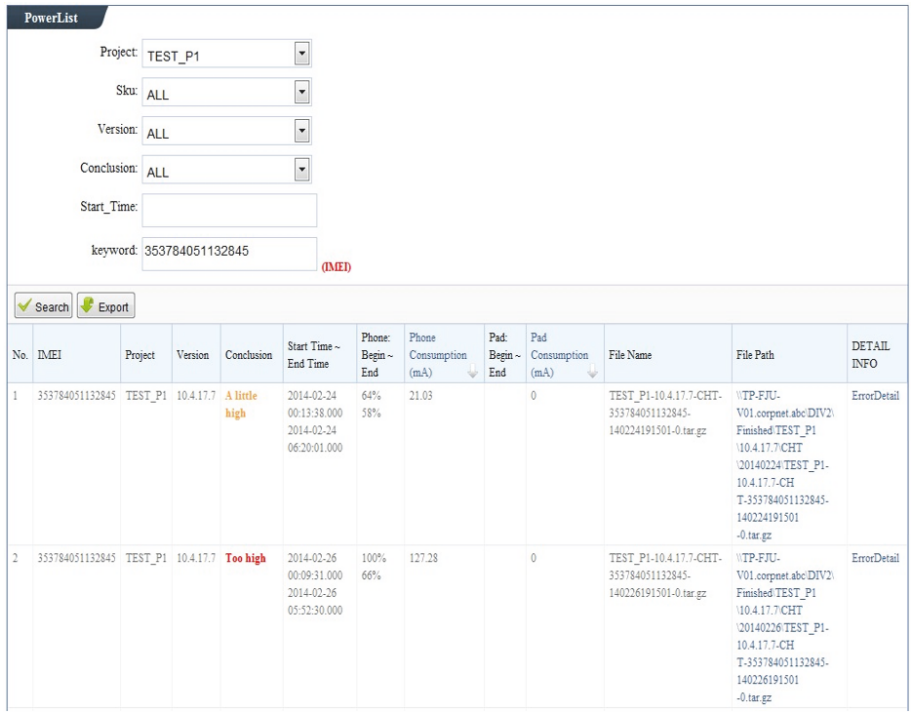
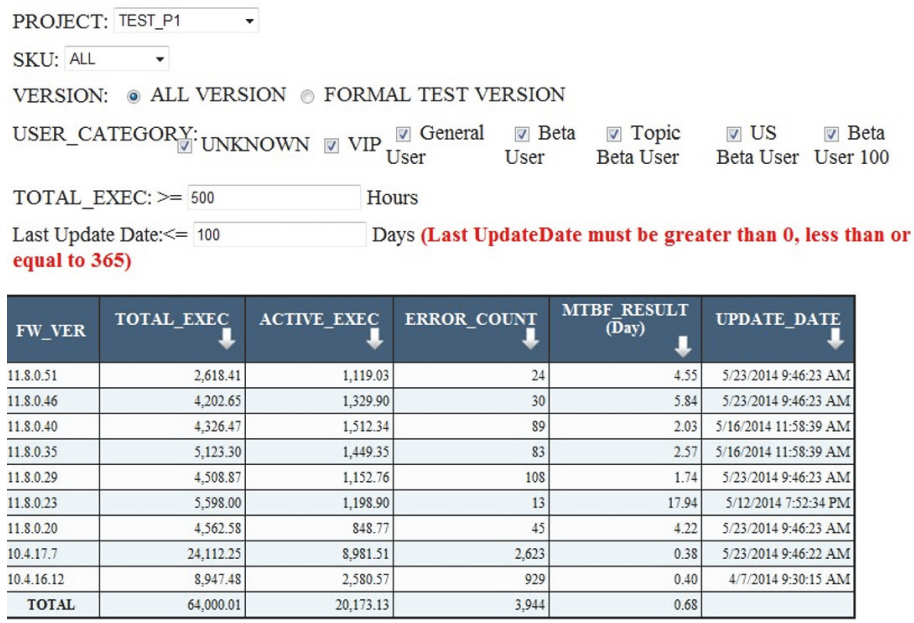


Figure 22: Device power consumption record query page.



FW_VER	TOTAL_EXEC	ACTIVE_EXEC	ERROR_COUNT	MTBF_RESULT (Day)	UPDATE_DATE
11.8.0.51	2,618.41	1,119.03	24	4.55	5/23/2014 9:46:23 AM
11.8.0.46	4,202.65	1,329.90	30	5.84	5/23/2014 9:46:23 AM
11.8.0.40	4,326.47	1,512.34	89	2.03	5/16/2014 11:58:39 AM
11.8.0.35	5,123.30	1,449.35	83	2.57	5/16/2014 11:58:39 AM
11.8.0.29	4,508.87	1,152.76	108	1.74	5/23/2014 9:46:23 AM
11.8.0.23	5,598.00	1,198.90	13	17.94	5/12/2014 7:52:34 PM
11.8.0.20	4,562.58	848.77	45	4.22	5/23/2014 9:46:23 AM
10.4.17.7	24,112.25	8,981.51	2,623	0.38	5/23/2014 9:46:22 AM
10.4.16.12	8,947.48	2,580.57	929	0.40	4/7/2014 9:30:15 AM
TOTAL	64,000.01	20,173.13	3,944	0.68	

Figure 23: MTBF report query page (by version).

thousand handheld devices, and there have been more than five hundred users participating.

5.1. Statistical Data. According to statistical data, the system has established effective MTBF indicators. The project generally gave poor results early in the test; afterwards, the projected slowly stabilized in terms of growth, such that developers could use the data to learn about the before-and-after versions to explain the achievement gap, to track abnormal version factors, and also to compare the differences

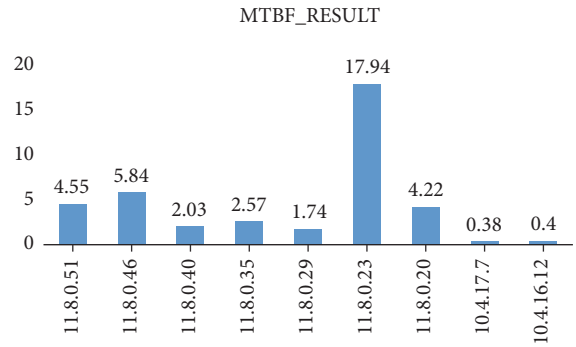


Figure 24: MTBF chart (by version).

Using Count: > 50000 Times
Using Time:> 10000 Hours
Display Top: 10

Total Using Count : 23,629,476 Times Total Using Time : 1,825,657.58 Hours Total Device : 2,443,838.76 Hours

RANKING	PACKAGE_NAME	USING_COUNT	USING_COUNT_PERCENT	USING_TIME	USING_TIME_PERCENT
1	com.fju.launcher	4,349,854	18.41 %	1,071,508.12	58.69 %
2	com.android.phone	803,167	3.40 %	88,135.16	4.83 %
3	com.google.android.gsf.login	192,167	0.81 %	71,055.12	3.89 %
4	com.android.settings	931,419	3.94 %	53,711.10	2.94 %
5	com.fju.contacts	639,210	2.71 %	24,953.41	1.37 %
6	jp.naver.line.android	3,879,834	16.42 %	24,393.39	1.34 %
7	com.fju.camera	276,988	1.17 %	20,880.09	1.14 %
8	com.android.vending	332,256	1.41 %	14,337.43	0.79 %
9	com.facebook.katana	1,042,530	4.41 %	14,075.31	0.77 %
10	com.fju.deskclock	220,925	0.93 %	11,132.12	0.61 %
	TOTAL	12,668,350	53.61 %	1,394,181.25	76.37 %

Figure 25: Query application usage page.

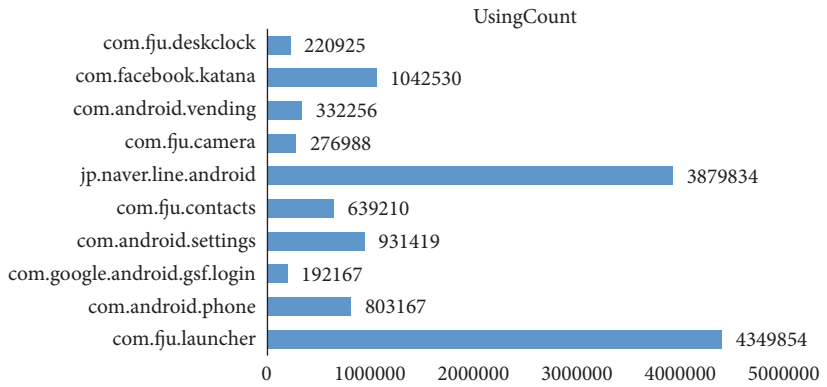


Figure 26: Application usage chart (by count).

in the various versions of the data. Before the product is shipped by the project manager, you can use the system to choose a good version.

For the test results, our goal is as follows: every beta user trial that is performed includes at least 30 devices in the test and contains each version and the duration of the tests is at least seven days to reach 30 (units) ∗ 7 days ∗ 24 = 5040 hours. The execution time of each version has more than 5040 hours of high credibility according to shipping standards. Our current goal is to set MTBF > 10 days to ensure that this version has a higher degree of stability.

Figure 38 contains information of each version to present the recent achievements. The figure clearly shows that the performance of version 11.4.5.32 significantly dropped. A

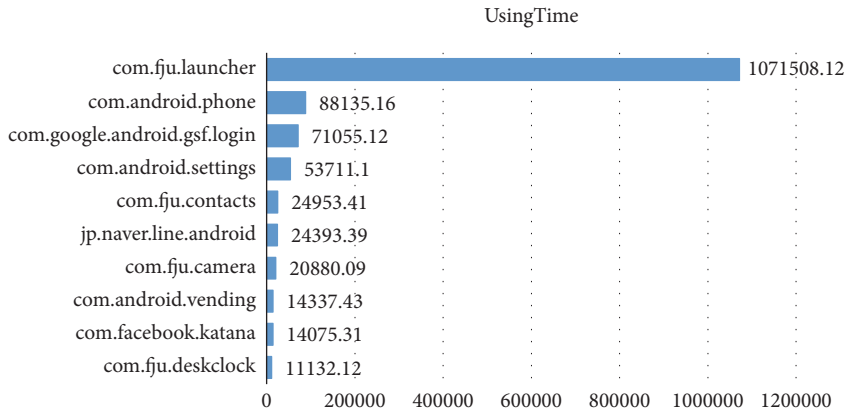
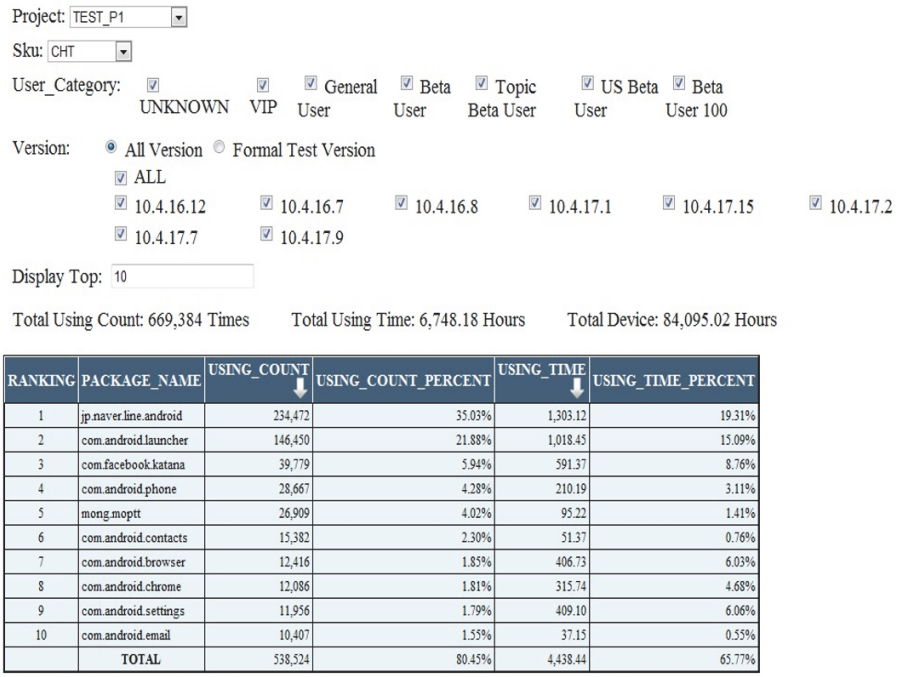


Figure 27: Application usage chart (by hours).



RANKING	PACKAGE_NAME	USING_COUNT	USING_COUNT_PERCENT	USING_TIME	USING_TIME_PERCENT
1	jp.naver.line.android	234,472	35.03%	1,303.12	19.31%
2	com.android.launcher	146,450	21.88%	1,018.45	15.09%
3	com.facebook.katana	39,779	5.94%	591.37	8.76%
4	com.android.phone	28,667	4.28%	210.19	3.11%
5	mong.moptt	26,909	4.02%	95.22	1.41%
6	com.android.contacts	15,382	2.30%	51.37	0.76%
7	com.android.browser	12,416	1.85%	406.73	6.03%
8	com.android.chrome	12,086	1.81%	315.74	4.68%
9	com.android.settings	11,956	1.79%	409.10	6.06%
10	com.android.email	10,407	1.55%	37.15	0.55%
	TOTAL	538,524	80.45%	4,438.44	65.77%

Figure 28: Query application usage by project.

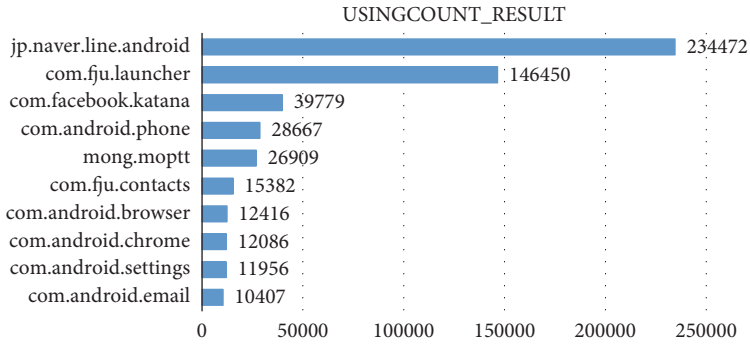


Figure 29: Project application usage chart (by count).

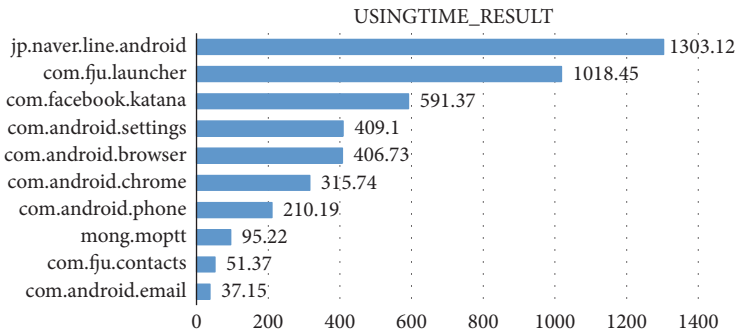


Figure 30: Project application usage chart (by hours).

Package Name: jp.naver.line.android

Using Count : > 0 Times

Using Time:> 0 Hours

Display Top: 10

Keyword:

Total Using Count : 23,629,476 Times Total Using Time : 1,825,657.58 Hours Total Device : 2,443,838.76 Hours

RANKING	PACKAGE_NAME	ACTIVITY_NAME	USING_COUNT	USING_COUNT_PERCENT	USING_TIME
1	jp.naver.line.android	activity.chathistory.chathistoryactivity	1,433,246	6.07 %	11,966.33
2	jp.naver.line.android	activity.main.mainactivity	1,313,906	5.56 %	2,471.37
3	jp.naver.line.android	activity.pushdialog.pushdialogsleepactivity	505,187	2.14 %	7,670.27
4	jp.naver.line.android	activity.splashactivity	164,815	0.70 %	24.61
5	jp.naver.line.android	jp.naver.gallery.android.activity.chatphotodetailactivity	92,113	0.39 %	345.57
6	jp.naver.line.android	common.passlock.inputpassactivity	44,857	0.19 %	40.32
7	jp.naver.line.android	activity.pushdialog.pushdialogactivity	31,734	0.13 %	432.43
8	jp.naver.line.android	jp.naver.gallery.android.activity.photodetailactivity	21,193	0.09 %	25.17
9	jp.naver.line.android	activity.addfriend.addfriendactivity	16,592	0.07 %	56.45
10	jp.naver.line.android	activity.stickershop.stickershopdetailactivity	15,495	0.07 %	25.48
	TOTAL		3,639,138	15.40 %	23,058.00

Figure 31: Query activity usage page.

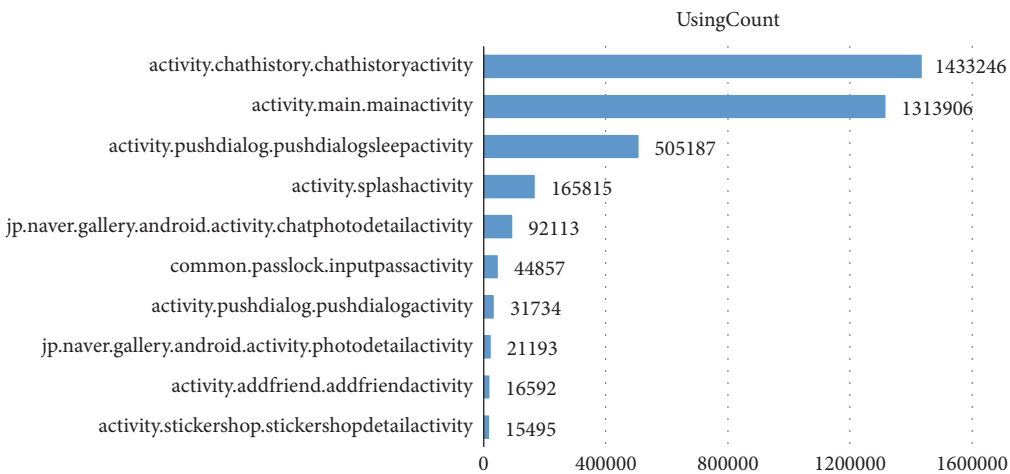


Figure 32: Activity usage chart (by count).

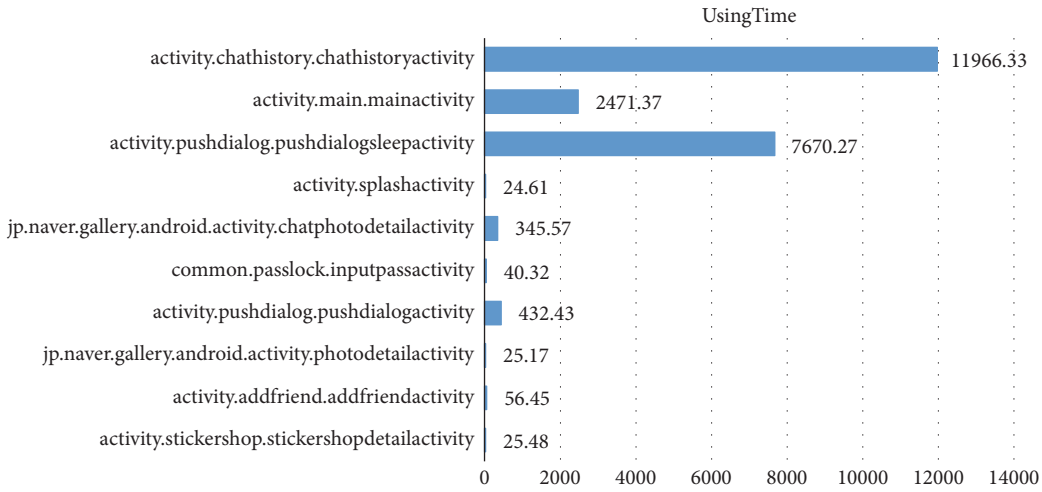


Figure 33: Activity usage chart (by hours).

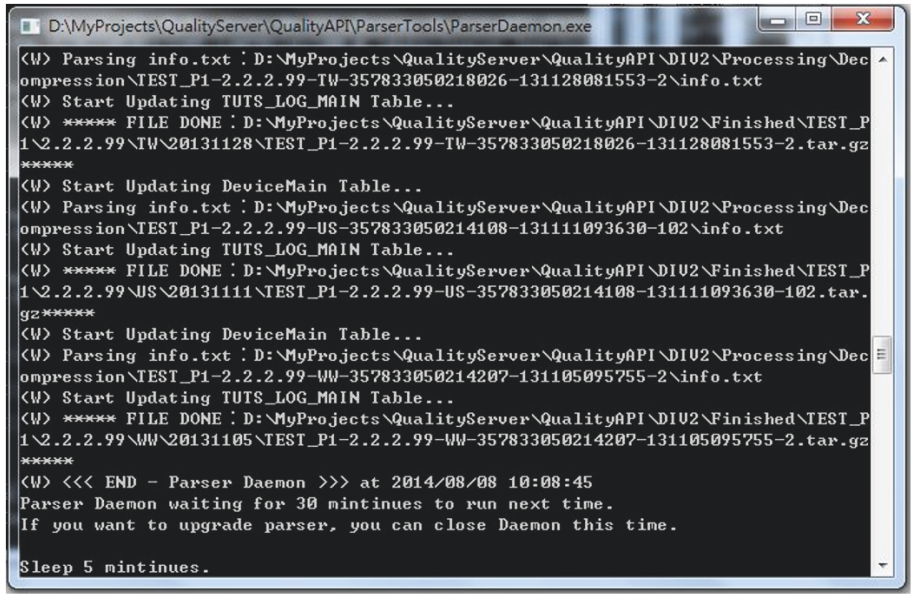


Figure 34: Parser daemon execution screen.

Table 3: Error summary list.

Error type	Error count
Self-shutdown	10
ANR and App force close	15
Modem crash	2

survey found that the number of modem crashes increased significantly; thus, the modem of this version is likely a problem.

Table 3 shows the results of the statistical data of a total of 27 errors, which shows that the ANR and App force close error occurred most often. Figure 39 lists the detailed statistics of the application errors.

Figure 40 shows the battery usage statistics information of a recent version. The reported statistics are when the device had to be standing for a long time. If there is an abnormal situation regarding battery power, because the user usually does have a long record of observing battery power situation, you want to obtain only relevant information on the logs analysis. Developers can monitor the device's power status every day and track the root cause.

Table 4 contains the test version usage of the devices with the following fields: user number is the number of test devices, daily use calculation = (total number of uploaded logs of this version/(version using days $*$ user number)) $*$ 100%, use time > average use time (%) = number of usage hours > average number of usage hours, as a percentage of total logs of this number, average use time (hour) = daily use (active) average number of hours, total duration (hour)

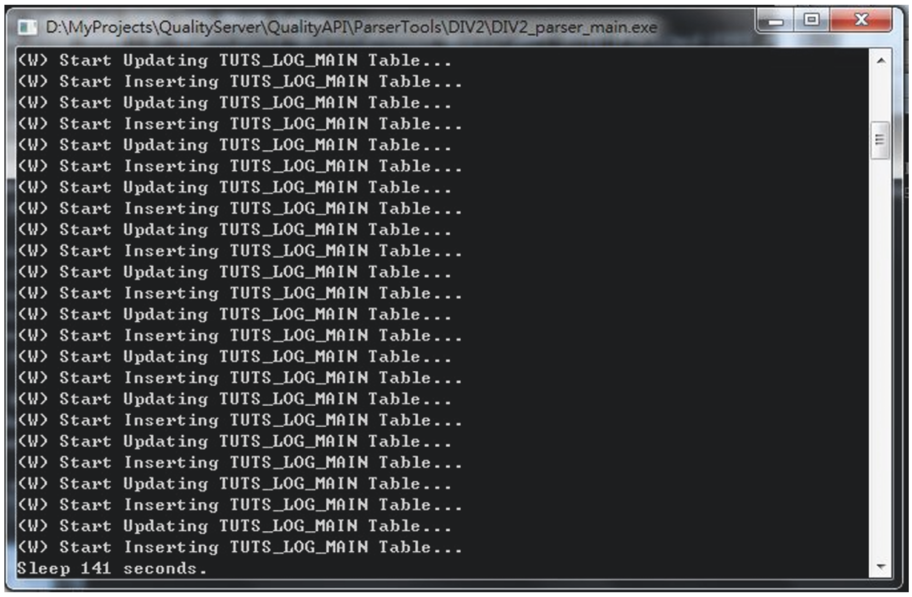


Figure 35: Parser main execution screen.

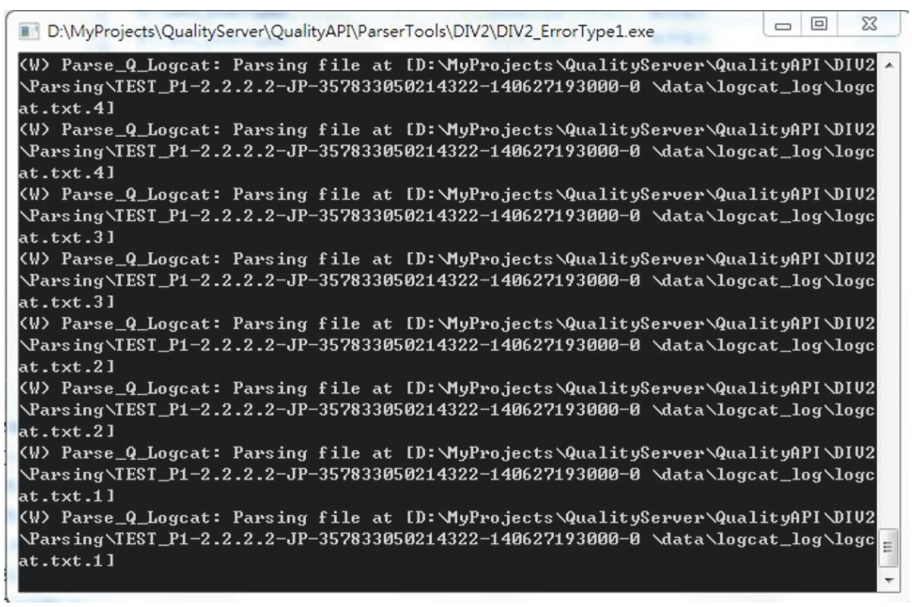


Figure 36: Error type 1 execution screen.

= all active hours of this version, and update rate (%) = proportion that has been updated to the latest version → 48/52 = 88.888%.

6. Conclusions

The ALAS system simplifies testing processes such that the issues of return and log uploads are more immediate and more efficient, which is combined with a FOTA server that can update the version for a specific group. Thus, users do not need to download and manually update the version. The entire testing process runs more smoothly.

The ALAS system establishes a mechanism for automated analysis of logs on the server side, which significantly reduces the workload of manually analyzing logs by a developer. The system can also rapidly collect and analyze log data, which helps if the user cannot find the background service error or if the problems are found by prolonged observation, which are difficult to find by a manual test.

In a traditional beta user trial, through only daily use and waiting for users to report a problem, a reference basis

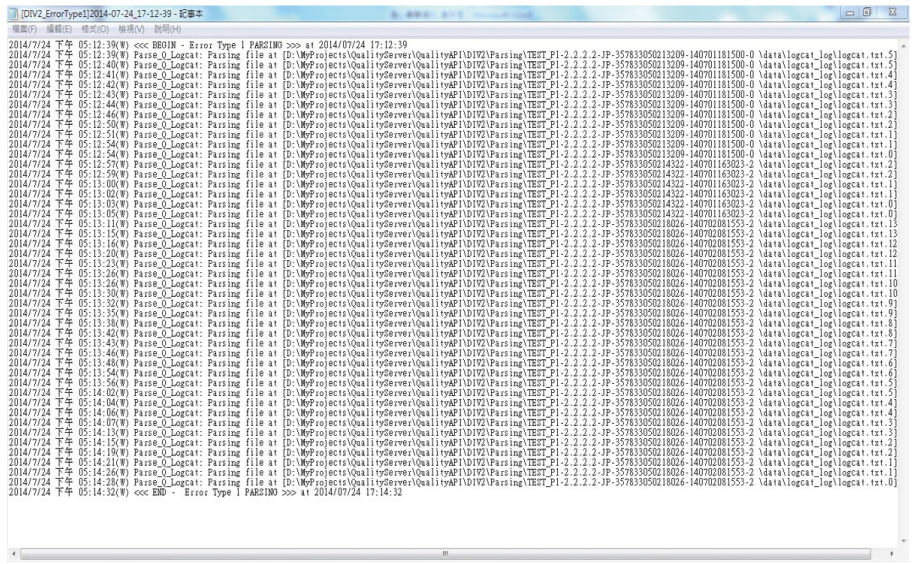


Figure 37: Error type 1 log file content.

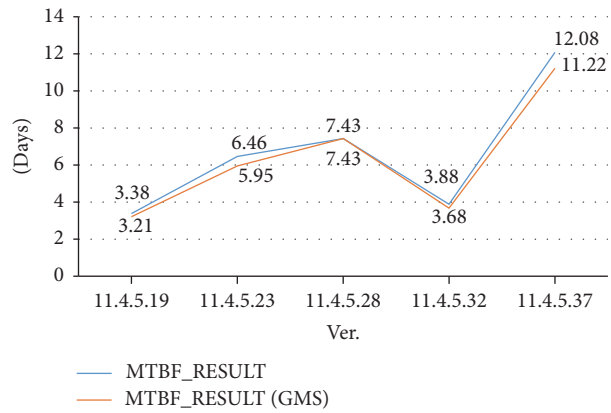


Figure 38: MTBF summary chart.

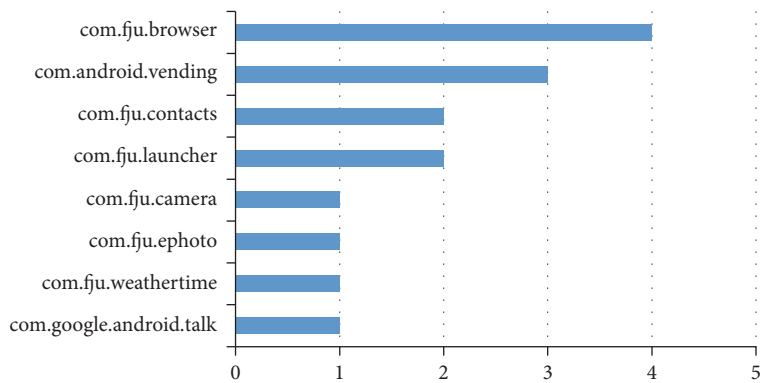


Figure 39: Application error chart.

Table 4: Test device usages list.

Project	Version	User number	Daily use (%)	Use time > average use time (%)	Average use time (hr)	Total duration (hr)	Update rate (%)
TEST_P1	11.4.5.37	54	82%	39%	4.85	2,357.41	89%

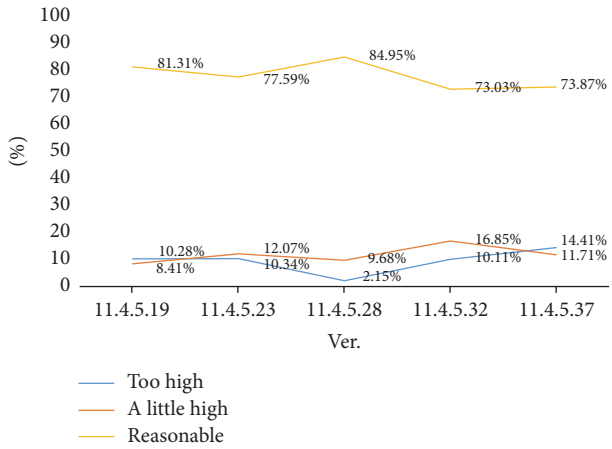


Figure 40: Battery usage chart.

cannot be quantified. In addition to providing statistical error information, the ALAS system also provides an MTBF report, which serves as a reliability index. Clearly labeled versions of the test results can be used as a reference standard for shipping.

This is the time of big data, from using big data to deriving useful information. Using a method with a higher reliability and using an extensive collection of a large number of logs, data analysis will be more complete; however, there are still a few deficiencies, such as the modem log, because the modem-related message is too complex and limited in terms of Internet restrictions regarding uploads. For example, the parser program cannot be imported into the system, and, perhaps with the future updates in communication transmission technology, this requirement can be satisfied. Using MTBF indicators, we incorporate the concept of weights that depend on the severity of the error, and the classification index increases the sensitivity to also import the MTTF (mean time to failure) and MTTR (mean time to recover) [23, 24], which enriches the quality of the indicators in our system. More log analysis modules are used, which also enhances the direction of the degree of completion in the system. In addition to doing the wrong returns, in the future, we hope to add new features to allow users to use applications and have direct feedback, which would include recommendations regarding the design, mobile performance software functionality, application interface, and behavior of the mobile device. These recommendations would include a new application plan, an extensive collection of user feedback combined with a forum for discussion. Thus, users and developers can interact more closely.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. Grover, "Android forensics: Automated data collection and reporting from a mobile device," *Digital Investigation*, vol. 10, pp. S12–S20, 2013.

[2] H. Martín, A. M. Bernardos, J. Iglesias, and J. R. Casar, "Activity logging using lightweight classification techniques in mobile devices," *Personal and Ubiquitous Computing*, vol. 17, no. 4, pp. 675–695, 2013.

[3] P. Ascione, M. Cinque, and D. Cotroneo, "Automated logging of mobile phones failures data," in *Proceedings of the 9th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 8–15, 2006.

[4] M. Cinque, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "How do mobile phones fail? a failure data analysis of symbian OS smart phones," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 585–594, 2007.

[5] P. Antonio and M. Cinque, "Log-based failure analysis of complex systems: methodology and relevant applications," in *Innovative Technologies for Dependable OTS-Based Critical Systems*, pp. 203–215, Springer, Milan, Italy, 2013.

[6] D. T. Wagner, A. Rice, and A. R. Beresford, "Device analyzer: large-scale mobile data collection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 53–56, 2014.

[7] T. Kobabayshi, Image by Tetsuyuki Kobabayshi, of Kyoto Microcomputer Co., 2011.

[8] IDC, Smartphone OS Market Share, 2016 Q3. Availible in: http://www.idc.com/promo/smartphone-market-share/.

[9] Wikipedia, Android (operating system). Availible in: https://en.wikipedia.org/wiki/Android_(operating_system).

[10] Android, 2012, Availible in http://elinux.org/Android_Logging_System.

[11] M. Cinque, "Enabling on-line dependability assessment of Android smart phones," in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W '11)*, pp. 286–291, 2011.

[12] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "Improving log-based field failure data analysis of multi-node computing systems," in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks (DSN '11)*, pp. 97–108, 2011.

[13] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2013.

[14] A. Pecchia and M. Cinque, "Log-based failure analysis of complex systems: methodology and relevant applications," in *Innovative Technologies for Dependable OTS-Based Critical Systems*, pp. 203–215, Springer, Milan, Italy, 2013.

[15] J. V. Jones, *Integrated Logistics Support Handbook*, McGrawHill, New York, NY, USA, Special Reprint edition, 1998.

[16] C. D. Martino, M. Cinque, and D. Cotroneo, "Assessing time coalescence techniques for the analysis of supercomputer logs," in *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '12)*, pp. 1–12, 2012.

[17] U. Varshney and A. D. Malloy, "An integrated approach for improving the dependability of the emerging wireless networks," in *Proceedings of the IEEE Global Telecommunications Conference*, pp. 3693–3697, 2001.

[18] J. Chen, J. Chen, and X. Zhang, "MTBF-oriented prediction model for airborne equipment reliability based on SOM," in *Proceedings of the 7th World Congress on Intelligent Control and Automation*, pp. 3660–3663, 2008.

[19] J. Tian, S. Rudraraju, and Z. Li, "Evaluating web software reliability based on workload and failure data extracted from server logs," *IEEE Transactions on Software Engineering*, vol. 30, no. 11, pp. 754–769, 2004.

[20] M. Braglia, G. Carmignani, M. Frosolini, and F. Zammori, "Data classification and MTBF prediction with a multivariate analysis approach," *Reliability Engineering and System Safety*, vol. 97, no. 1, pp. 27–35, 2012.

[21] T. T. Lin and D. P. Siewiorek, "Error log analysis: statistical modeling and heuristic trend analysis," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp. 419–432, 1990.

[22] A. G. Colombo and A. S. Bustamante, "Systems reliability assessment," in *Proceedings of the Ispra Course Held at the Escuela Tecnica Superior de Ingenieros Navales. In Collaboration with Universidad Politecnica de Madrid*, Springer, 1990.

[23] M. Cinque, D. Cotroneo, and S. Russo, "Collecting and analyzing failure data of bluetooth personal area networks," in *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 313–322, 2006.

[24] S. M. Matz, L. G. Votta, and M. Malkawi, "Analysis of failure and recovery rates in a wireless telecommunications system," in *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 687–693, 2002.