Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 16, no. 8, August 2020, pp. 1279-1288
DOI: 10.23940/ijpe.20.08.p15.12791288

Incremental Data Mining-based Software Failure Detection

Pan Liu[a,b] and Wulan Huang[a,*]

[a]*Faculty of Business Information, Shanghai Business School, Shanghai, 201400, China*
[b]*Engineering Research Center for Software Testing and Evaluation of Fujian Province, Xiamen, 361000, China*

Abstract

It has been proved by practice that mining weblogs to detect software errors is an effective software testing method. This paper presents a software failure detection method based on the incremental mining weblog strategy and gives data mining steps for the implementation of this method. A case is studied for the proposed test method. In this case, we use Splunk, a data analysis tool, to analyze some weblogs that record some linked information of mobile applications for android. The result of the data analysis shows that the proposed method can effectively detect the software failure problem in the download process of mobile applications. Therefore, the proposed method can be used for software reliability assessment.

Keywords: incremental data mining; software failure detection; weblog; Splunk; reliability assessment

(Submitted on May 30, 2020; Revised on June 28, 2020; Accepted on July 26, 2020)

1. Introduction

Computer software has been integrated into our lives, but their reliability is not satisfactory, which leads to the trust crisis of software [1-2]. In the past, software testing has always been an effective technique to ensure software quality. As the complexity of computer software has grown, software testing methods have also been continuously improved. According to the design and execution of test cases, we can arbitrarily classify the existing software testing methods into three categories: 1) the testers manually design test cases and then execute them to test the software; 2) the testers design test scripts and then use test tools, such as Junit [3] and Selenium [4], to automatically execute test scripts; 3) Generate test cases from the model describing software behaviors, such as FSM [5-6], EFSM [7], and RFSM [8], and then the testers observe whether the software running is consistent with its behavior model by executing test cases. These software testing methods have a very good role in ensuring software quality.

A new test method based on data mining was proposed [9-10]. This proposed test method is to detect software defects by mining log files on the server. Because the log is generated during the user's practical use of the software, it records both user operation information and software operation information. By mining the logs collected from the web server, software failure information recorded in the logs can be detected. The advantage of this method is that it does not require the manual design of test cases and execution of test cases. Instead, the log mining method is used to detect software failure. The corresponding data mining techniques have become the key to this test method.

However, a large number of software running logs recorded in the server have caused the difficulty of data storage and processing [11]. On the one hand, log files cannot be directly stored in a relational database because their data format is unstructured. Therefore, we cannot use SQL to process massive log files. On the other hand, there are many incomplete or malformed "dirty data [12]" in real-world data. Therefore, researchers used some data cleaning methods [13-15] to filter these "dirty data". However, in the era of big data, any data has its value. A good data mining method cannot only reduce the cost of data analysis, but also mine the value from both "non-dirty" and "dirty data".

To solve the problem of massive log analysis, this paper proposes an incremental data mining method to detect

* Corresponding author.
E-mail address: huangwl@sbs.edu.cn

software failures. Before the log analysis, we cannot determine what valuable things can be mined from the massive logs. Therefore, it is a risky and costly thing to directly analyze and mine the entire amount of data. The proposed incremental data mining technique is to extract a small amount of data for mining first and then constructing a data mining model to detect software defects. If software defects cannot be found in a small amount of data, we can add some extra data from the overall data or modify the data mining model. If software defects still cannot be found when the amount of data analyzed has reached a certain threshold, then we will stop mining those logs. If software failures can be found, all data needs to be further analyzed.

To demonstrate our proposed method, we downloaded a batch of weblogs that record download information of mobile APKs on the web server. Then, we used a big data analysis software, Splunk [16], to perform incremental analysis on this batch of data. We found: i) Some failures occur during the download process of some users' mobile phones; ii) We obtained the specific failure ranking of the APKs' download links; iii) We detected the version number of the android operating system that was related to the APKs' download links with software failures.

The organizational structure of the paper is as follows: Section 2 investigates the related work of the paper. Section 3 introduces the incremental data mining method and discusses the difference between the proposed method and some existing test methods. In Section 4, we use the big data analysis software Splunk to analyze the weblogs and find some software details. Then, we also give a discussion for comparing our method with two kinds of existing software testing methods. Section 5 summarizes the full paper and gives two future research directions.

2. Related work

Data mining-based software testing has been studied for 20 years. Some log mining methods are proposed to detect software defects.

In 2001, Kallepalli and Tian [9] mined software usage and failure information by analyzing weblogs of software. Their method is to use statistical strategies for web testing and use failure information to measure the reliability of web applications. They constructed Unified Markov Models (UMMs) to simulate the behavior and use of the software. Then, the reliability analysis of Web applications was carried out through frequency-based testing on UMMs. Their contribution is to use statistical methods to achieve data mining and software testing.

In 2006, Song et al. [10] evaluated the safety and reliability of the software by mining the relationships between software attributes in logs. Their method is based on the assumption that if there is a relationship $a \rightarrow b$ in the attributes of the software, then b can be found wrong when it is found in a. Therefore, if a can be found in logs and b is not found in the real software test results, it can be judged that the software is unreliable. In their experiments, their method had 23% more defect correction ability then PART, C4.5, and Naïve Bayes.

In 2008, Lim et al. [17] analyzed the failure of the enterprise telephone system by mining logs. They presented three problems in log processing mining: 1) the loss of part information in logs, 2) the difficulty for storage and processing of a large number of logs, and 3) the barrier of professional domain knowledge for data miners. The reduction of storage costs and the improvement of data calculation capabilities can solve the second problem. The third problem can be figured out by studying and training the professional domain knowledge for data miners. However, the first problem will bring out a challenge for data analysis.

In 2010, Jiang et al. [18] proposed a method to estimate the reliability of software by mining the software execution logs and software deployment logs. Different from the previous way of mining logs, their method is carried out by mining user acceptance test logs for a limited time, thus saving the time cost of user acceptance testing. However, when no software defects have been discovered in a limited time, their method will fail.

Because the interactions between users and the system can be recorded in logs, in 2014, Rubin et al. [19] suggested mining those logs to improve software performance. They used the tool Disco to build the event model for software, where nodes represent the actions, and arcs denote the executive orders of the nodes. Then, they used the tool Disco to process two industrial projects and analyzed the behavior of software users. However, the modeling ability of Disco determines its data analysis ability. Some incomplete data in logs will not be modeled by Disco.

In 2018, Ioannou et al. [20] improved the plug-in *Rabbit Eclipse* of Eclipse and used it to generate event logs of the use of the IDE. Then, they analyzed the event logs of six developers by this method and obtained the development preferences

of software developers. By mining the logs generated by the plug-in *Rabbit Eclipse*, they analyzed the possible software defects based on the developer's behavior.

Unlike existing log mining methods, we propose an incremental data mining method to detect software defects. Our method considers both the cost of data mining and the problem of incomplete data formats. We used the software Splunk to realize the massive data mining.

3. Our Method

We propose an incremental data mining method, shown in Figure 1, to detect software defects. Our method contains seven main steps as follows:

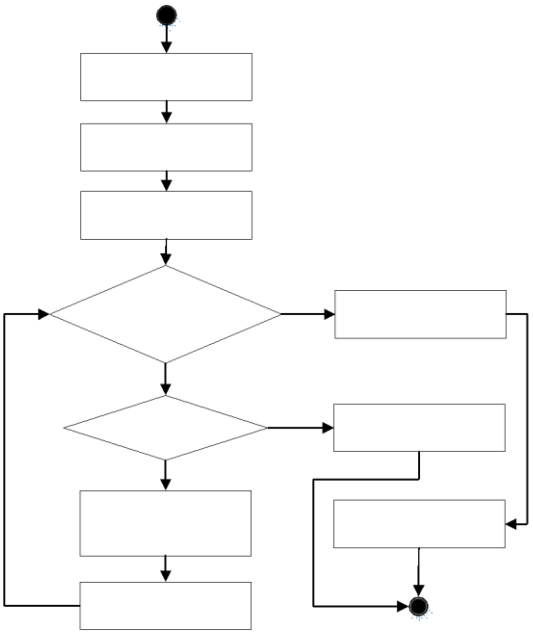


Figure 1. An incremental data mining-based software failure detection method

Step 1 Sampling part of the data from the overall data.

Step 2 Construct a model to analyze those samples.

Step 3 If software failures can be found by the data analysis result, we use the model to analyze the overall data.

Step 4 Otherwise, expand the proportion of data being analyzed or improve the model.

Step 5 If the proportion of data being analyzed reaches a given threshold and no software defect is found, stop mining data, and go to Step 7.

Step 6 Otherwise, return to Step 4.

Step 7 Output the software failure report.

Note that based on our experience, we recommend that 30% be the threshold of massive sampled data. Besides, the analytical model of the sampled data is the key to our method. We use the cut-and-try method in our method to build and rebuild the model.

Compared with some existing software defect detection methods based on data mining [9-10, 17-20], our method can reduce the cost of data analysis. Before data mining, we are not sure whether there are defects in the software and they can be found by mining logs. Therefore, if we are blindly performing data mining when some of the data is missing or the data mining method is improper, it may lead to the failure of software defect detection based on data mining, thereby wasting the

cost of massive data analysis. Our incremental data mining method can reduce the data analysis costs because only part of the data is analyzed in our method. One challenge of our method is that the data sampling strategy determines whether our method can be successful.

4. Case Study

We have downloaded some weblogs about the download links of mobile application packages (APKs) from the web server. Readers can download a weblog to follow this case study from https://pan.baidu.com/s/12V9IyeJ6Jjeq_wRjEOTPnw with the password y5sz. Figure 2 shows parts of logs, which include the client IP addresses, download time, download link, HTTP request way, HTTP status, browser, the mobile phone operating system, android version information, mobile phone brand, and User-Agent. We have analyzed_link failure information of APKs from logs to formulate corresponding strategies to solve the failure problem. In this case study, we used the software Splunk[1] to analyze those logs. Splunk treats each record in logs as an event and adds a timestamp field time to differentiate each record. We used the incremental data analysis method to analyze these logs. After we upload a weblog to Splunk, several Splunk search statements shown are automatically generated as follows.

Source = "2019-04-13-0000-2330_3.log" host = "pan-PC" sourcetype = "access_combined"

```
69.30.227.146 - - [13/Apr/2016:00:15:27 +0800] "GET http://www.download.cn/down/dl/dl.apk HTTP/1.0" 200 8327108 "-"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.3) Gecko/2008092416 Firefox/3.0.3"
38.108.108.178 - - [13/Apr/2016:00:14:55 +0800] "GET http://www.download.cn/down/dl/dl.apk HTTP/1.1" 200 22485545 "-"
"Mozilla/5.0 (X11; U; Linux i686) Gecko/20071127 Firefox/2.0.0.11"
69.197.181.84 - - [13/Apr/2016:00:17:45 +0800] "GET http://www.download.cn/down/dl/dl.apk HTTP/1.0" 200 11841472 "-"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.3) Gecko/2008092416 Firefox/3.0.3"
58.32.234.155 - - [13/Apr/2016:00:01:26 +0800] "GET http://www.download.cn/down/8.3/dl_v832_0226.apk HTTP/1.1" 304 242 "-"
"Mozilla/4.0 (compatible;)"
58.161.77.105 - - [13/Apr/2016:00:08:19 +0800] "HEAD http://www.download.cn/down/8.3/dl_v832_0226.apk HTTP/1.1" 200 303
"-" "Mozilla/5.0 (Linux; U; Android 4.4.4; zh-cn; m1 note Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0
Mobile Safari/537.36"
58.161.77.105 - - [13/Apr/2016:00:08:49 +0800] "GET http://www.download.cn/down/8.3/dl_v832_0226.apk HTTP/1.1" 200
1253303 "-" "Mozilla/5.0 (Linux; U; Android 4.4.4; zh-cn; m1 note Build/KTU84P) AppleWebKit/537.36 (KHTML, like Gecko)
Version/4.0 Mobile Safari/537.36"
```

Figure 2. Sample of part of logs

As shown in Figure 3(a), we find that the log contains 434,914 events.

2.1. Incremental Data Analysis

To realize the incremental data analysis strategy, we need to 1) construct a data filtering model consisting of Splunk search codes, and 2) master the relevant knowledge of the fields involved in the data. Before the data analysis, we do not know whether there are errors in the software system, so we use incremental detection methods to analyze a small percentage of data in anticipation of finding software failure at a low cost. In practice, we recommend that the size of the data increment is 1%, 5%, 10%, and 30%.

Our incremental data analysis strategy is implemented as follows.

1) First of all, we selected 10,000 events from all events and then analyzed them.

2) We found that there are many events with status 404, shown in Figure 3(b). By looking for the HTTP status codes [21], we know that "status = 404" means "Not Found", that is, downloaded APKs were not found.

3) So, we want to get how many events have a status code of 404 out of all events.

From the data analysis chart, shown in Figure 3(b), the status codes included in the log are 0, 200, 206, 302, 304, 404, and 416. Among them, there are 4081 events with the status code 206, and 3436 events with the status code 404. By viewing the list of the HTTP status codes[2], we know that

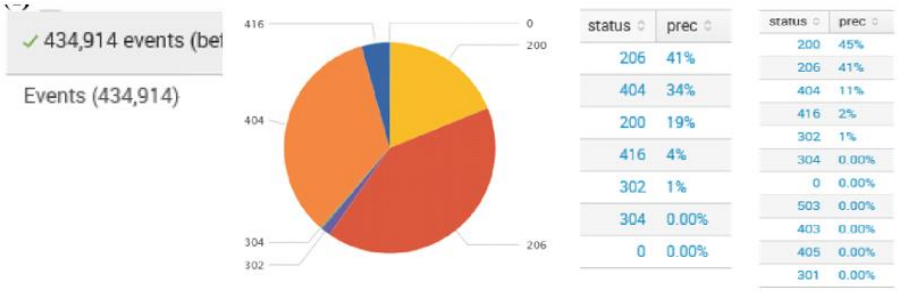
[1] https://www.splunk.com/
[2] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

- 206 indicates ***Partial Content***, and the server has successfully processed part of the GET request, which is the meaning of the resuming of the breakpoint.
- 302 indicates ***Move Temporarily***, and the requested resource temporarily responds to the request from a different URI.
- 304 means ***Not Modified***. If the client sends a conditional GET request and the request has been allowed, but the content of the document has not changed, the server should return this status code.
- 416 indicates ***Requested Range Not Satisfiable***.

In addition, we do not find the status code 0 in the list of the HTTP status codes. Through the analysis of the status code, we know that 404 and 416 indicate that the download is unsuccessful. To realize the above analysis, we construct the following Splunk search statements.

source = "2019-04-13-0000-2330_3.log" host = "pan-PC" sourcetype = "access_combined" | head 10000| stats count by status | sort - count | eventstats sum(count) as total | eval prec = round(count/ total,2)*100 ."%"

By executing the above code in Splunk, we obtained the percentage of each status code in 10000 events, as shown in Figure 3(c). From Figure 3(c), events with the status code 404 accounted for 34.4% of total events. Then, we can remove the statement "| head 10000" in the above codes and then obtain the analysis result of all data, shown in Figure 3(d). Compared with Figure 3(c), we found some new status that did not appear in the original 10000 events, and those events with 404 status accounted for 11% of the total events in Figure 3(d).



status	prec
206	41%
404	34%
200	19%
416	4%
302	1%
304	0.00%
0	0.00%

status	prec
200	45%
206	41%
404	11%
416	2%
302	1%
304	0.00%
0	0.00%
503	0.00%
403	0.00%
405	0.00%
301	0.00%

(a) (b) (c) (d)

Figure 3. Some information in the weblog

To detect which APK links have the download failure issue, we need to do further analysis of the data. We process the following steps to analyze APK links.

1) To remain those events with the state code 404 and two fields APK's link and status, we construct the following Splunk search statements.

source = "2019-04-13-0000-2330_3.log" host = "pan-PC" sourcetype = "access_combined" status = 404

2) To count the number of events with the status = 404 for each link, we construct the following codes. The execution result of the above codes is shown in Figure 4(a). From Figure 4(a), we found that the link with the most failure is dl_v834_0325.apk, followed by the links dl_v823_1224.apk and dl_v835_0408.apk.

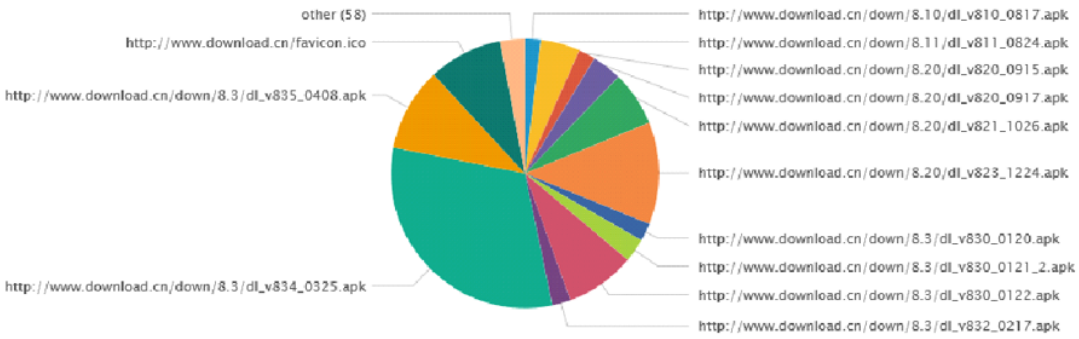
3) To obtain the failure percentage of each APK, we construct the following Splunk search statements.

source = "2019-04-13-0000-2330_3.log" host = "pan-PC" sourcetype = "access_combined" status = 404 | head = 10000 | stats count by uri | eventstats sum(count) as total | eval prec = round(count/total,2)*100 ."%" | fields - total

In the above codes, we use "status = 404" to get all events with the state code 404. Figure 4(a) shows the execution result of the above codes. From Figure 4(b), the total number of three APKs with the most link failure has exceeded about 53% of the total number of failures in 10,000 events. Therefore, we need to focus on the links of the three APKs and analyze the reasons for the download failure. Note that we will remove the statement "head = 10000" in the above codes to obtain the real percent of the link failure events in all data.

2.2. Version Number Analysis

To get the Android version information related to link failure, we built a model to filter out the data with a link failure. We used a heuristic method to construct a data filtering model based on known information. To ensure the effectiveness of the model, we checked those data filtered by the model. If the data that needed to be retained are deleted by the model, we need to improve the model to retain them.



(a)

uri	prec
http://www.download.cn/down/8.3/dl_v834_0325.apk	31.37%
http://www.download.cn/down/8.20/dl_v823_1224.apk	12.27%
http://www.download.cn/down/8.3/dl_v835_0408.apk	10.00%
http://www.download.cn/favicon.ico	8.91%
http://www.download.cn/down/8.3/dl_v830_0122.apk	8.42%
http://www.download.cn/down/8.20/dl_v821_1026.apk	6.57%
http://www.download.cn/down/8.11/dl_v811_0824.apk	4.75%
http://www.download.cn/down/8.20/dl_v820_0917.apk	3.61%
http://www.download.cn/down/8.3/dl_v830_0121_2.apk	2.83%
http://www.download.cn/down/8.3/dl_v832_0217.apk	2.22%
http://www.download.cn/down/8.3/dl_v830_0120.apk	2.14%
http://www.download.cn/down/8.20/dl_v820_0915.apk	2.03%

(b)

Figure 4. The result of the link failure APKs' analysis

From Figure 5(a), the format of the version is similar to Android 4.4.4. So, we first constructed a regular expression

"Android (?<AndroidVer>\d.\d.\d)"

to retain those data that satisfy the regular expression.

In the above regular expression, AndroidVer is an array to store Android version numbers. The software Splunk will automatically generate a new field AndroidVer if it is not empty. From Figure 5(b), about 66.149% of the data in the field AndroidVer meets the regular expression, which means that either a large number of records in the data do not contain Android version information, or there are other types of version information that do not meet the regular expression. Therefore, we checked those data that do not meet the regular expression "Android (?<AndroidVer>\d.\d.\d)". From Figure 5(c), we found that some Android versions are similar to "Android 4.3", "Android\4.3" and "Android\5.1.1". Thus, we modified the regular expression "Android (?<AndroidVer>\d.\d.\d)" to a new regular expression

"Android[/](?<AndroidVerc>\d.\d.\d|\d.\d)"

Then, the following Splunk search statements are constructed to get Android versions with APK link failure in the weblog.

source = "2019-04-13-0000-2330_3.log" host = "pan-PC" sourcetype = "access_combined" "Android" status = "404" | rex "Android[/](?<AndroidVerc>\d.\d.\d|\d.\d)" | stats count by AndroidVerc | eventstats sum(count) as total | sort - count |

eval prec = round(count/total,2)*100 ."%" | fields - total | search NOT prec = "0.00%"

Finally, the result of the percentage of android versions with APK link failure is shown in Figure 5(d). From Figure 5(d), the failure rate of Android 4.1.1 is 84%, followed by Android 4.2.2. Therefore, we need to investigate the failure of these two links to look forward to discovering the cause of the failure.

In addition, we found that some events have the status code 0 in Figure 3(d). To explore which version number produced the largest number of events with the status code 0, we constructed the following code.

source="2019-04-13-0000-2330_3.log" host="pan-PC" sourcetype="access_combined" "Android" status=0 | rex "Android[/](?<Data>\d.\d.\d|\d.\d)"

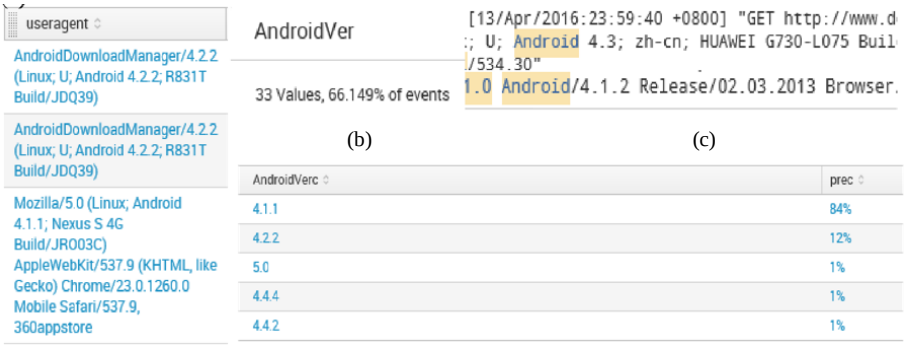


Figure 5. The analysis of the Android version

The execution result of the above code is shown in Figure 6. From Figure 6, the version number 4.1.1 contributes the greatest number of events with the status code 0. Therefore, we need to redesign some new test cases to test the link with Android 4.1.1 so that the software failure with the status code 0 in the web server can be fixed.

Top 10 Values	Count	%
4.1.1	80	45.198%
4.4.4	24	13.559%
4.4.2	19	10.734%
5.0.2	12	6.78%
4.3	10	5.65%
4.2.2	7	3.955%
5.1.1	5	2.825%
4.4	3	1.695%
5.0	3	1.695%
6.0.1	3	1.695%

Figure 6. The version number of events with the status code 0

2.3. Discussion

In this section, we compare the features among the traditional test methods, the data mining-based software test method, and our method. In addition, we also discuss four kinds of data analysis software, including Splunk, SPASS, Excel, and Python.

As shown in Table 1, traditional software testing methods need to manually construct test cases or generate test cases from models. Then, the testers execute test cases and determine whether the software has errors by observing its execution results. Both the data mining-based software testing method and our method use data analysis techniques to discover software execution abnormalities recorded in software logs instead of test case design and test execution. A big difference is that traditional software testing methods occur in the software development phase, while both the data mining-based software testing method and our method occur after the software is delivered to users. To fix software errors, both the data mining-based software testing method and our method need to design a few test scenes to reproduce software errors. In fact,

these two testing methods are the complement and perfection of the traditional testing methods. Both the data mining-based software testing method and our method can easily find a small number of small-probability software failure events, which are hardly found by traditional software testing methods. According to the report [22], traditional software testing methods had accounted for 50% of the total software development cost. The cost of data mining-based testing method lies on the cost of data analysis. Because our method does not need to analyse all data, the cost of our method is less than that of the data mining-based software testing method.

From Table 2, it can be seen that SPASS, Excel, Python, and Splunk have data analysis and data visualization capability. Both SPASS and Excel can only analyse structured data, while both Python and Splunk can analyse structured and unstructured data. In addition, since both Python and Splunk are free, the cost of data analysis for software testing is reduced. Excel, Python, and Splunk can collect data from websites or web servers. The level of difficulty of data analysis from low to high is Excel, SPSS and Splunk, and Python. Therefore, in summary, Splunk is a better choice for analysing unstructured data (such as weblogs).

Table 1. Comparison of three kinds of software testing methods

Item	Traditional software testing methods	The data mining-based software testing method	Our method
Test cases	Y	N	N
Test execution	Y	N	N
Test technology	Test and observation	Data analysis	Data analysis
Data analysis	N	Y	Y
Software error	Y	Y	Y
Test phase	Software development	Software release	Software release
Test role	Leading role	Supporting role	Supporting role
Small probability event	Hard	Easy	Easy
Test cost	About 50% of the total cost	Data analysis cost	Part of data analysis cost

Table 2. Discussion of four data analysis software

Item	SPSS	Excel	Python	Splunk
Data analysis ability	Y	Y	Y	Y
Data visualization ability	Y	Y	Y	Y
Structured data	Y	Y	Y	Y
Unstructured data	N	N	Y	Y
Free software	N	N	Y	Y
Data collection ability	N	Y	Y	Y
Difficulty level	Medium	Easy	Hard	Medium

3. Conclusion

Software failures are frequent occurrences. To record the execution information of the software that includes software failures, there are a large number of logs generated on the Web server. Mining these log data can discover the cause of software failure, and then help testers to conduct targeted tests to repair software defects. This paper presents a software failure detection method by mining weblogs on the server and uses an example to demonstrate our proposed method. From our case study, we get that the data mining-based software test method needs to have two key elements: 1) provide professional tools for data analysis, and 2) skills in data analysis. Although commonly used data analysis tools such as excel can also analyze data, log files are often not in standard format files. For this reason, Excel cannot directly process these files and the data requires specific data analysis tools. In addition, weblogs on the server are huge. When we analyze all the data without purpose, we may not find software failure with a high cost of time. Therefore, selecting the appropriate data mining skills to analyze the log is the key to the test method based on log mining.

To solve this problem, we propose an incremental data mining method. Our method is to first select a part of the data for analysis and then construct a model to detect whether software failure is found. If software failure can be found in a part of the data, we can use the model to analyze the whole data. If we do not find software defects in the part of the data, we can add more data for analysis by the model or modify the model. Finally, we demonstrate our method by the data analysis software Splunk. Our method includes four steps:

1) Data sampling. Extract a small amount of data from all data for analysis, and find possible data features.

2) Data modeling. Establish models based on data characteristics.

3) Filter the data and analyze the filtered data to verify whether the model is valid and complete. If the model is not complete, you need to adjust the model and filter the data again.

4) Use the filtering model to obtain a batch of data, and then analyze them to obtain relevant information about software failure.

Our method can supplement traditional software testing methods, but it cannot replace traditional software testing methods. In practical applications, there is also the problem that software failure cannot be found due to the inaccurate model written. For this reason, how to judge the effectiveness of the data filtering model more effectively is one of our future research directions. Besides, we also need to study the direct relationship between the sampling ratio and software defect detection in our method.

Acknowledgements

This work was supported in part by the National Social Science Fund General Project of China under Grant 18BTQ058, the National Natural Science Foundation of China under Grant 61502299, and the Business Scholar project in the Shanghai Business School.

References

1. A. Johanson and W. Hasselbring, "Software Engineering for Computational Science: Past, Present, Future," *Computing in Science and Engineering*, Vol. 20, No. 2, pp. 90-109, 2018
2. M. P. Cristescu, J. A. Vasilev, M. V. Stoyanova, and A. -M. R. Stancu, "Capability and Maturity. Characteristics Used in Software Reliability Engineering Modeling," *Land Forces Academy Review*, Vol. 24, pp. 332-341, 2019
3. M. L. Gromov, S. A. Prokopenko, N. V. Shabaldina, and A. V. Laputenko, "Model based JUnit Testing," in *Proceedings of the 20th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices* (*EDM*), pp. 139-142, Erlagol, Russia, 2019
4. S. M. Shariff, H. Li, C. -P. Bezemer, A. E. Hassan, T. H. Nguyen, and P. Flora, "Improving the Testing Efficiency of Selenium-based Load Tests," in *Proceedings of the IEEE/ACM 14th International Workshop on Automation of Software Test* (*AST*), pp. 14-20, Montreal, QC, Canada, 2019
5. P. Liu, Y. Li, and Z. Li, "Some Thoughts on Model-based Test Optimization," in *Proceedings of the IEEE 19th International Conference on Software Quality, Reliability and Security Companion* (*QRS-C*), pp. 268-274, Sofia, Bulgaria, 2019
6. P. Liu, H. -K. Miao, H. -W. Zeng, and Y. Liu, "FSM-based Testing: Theory, Method and Evaluation," *Jisuanji Xuebao* (*Chinese Journal of Computers*), Vol. 34, pp. 965-984, 2011
7. Y. Chen, A. Wang, J. Wang, L. Liu, Y. Song, and Q. Ha, "Automatic Test Transition Paths Generation Approach from EFSM using State Tree," in *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security Companion* (*QRS-C*), pp. 87-93, Lisbon, Portugal, 2018
8. P. Liu and Z. Xu, "MTTool: A Tool for Software Modeling and Test Generation," *IEEE Access*, Vol. 6, pp. 56222-56237, 2018
9. C. Kallepalli and J. Tian, "Measuring and Modeling Usage and Reliability for Statistical Web Testing," *IEEE Transactions on Software Engineering*, Vol. 27, pp. 1023-1036, 2001
10. Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software Defect Association Mining and Defect Correction Effort Prediction," *IEEE Transactions on Software Engineering*, Vol. 32, pp. 69-82, 2006
11. A. Miranskyy, A. Hamou-Lhadj, E. Cialini, and A. Larsson, "Operational-Log Analysis for Big Data Systems: Challenges and Solutions," *IEEE Software*, Vol. 33, pp. 52-59, 2016
12. M. A. Hernández and S. J. Stolfo, "Real-World Data is Dirty: Data Cleansing and the Eerge/Purge Problem," *Data Mining and Knowledge Discovery*, Vol. 2, pp. 9-37, 1998
13. H. Liu, S. Shah, and W. Jiang, "On-Line Outlier Detection and Data Cleaning," *Computers and Chemical Engineering*, Vol. 28, pp. 1635-1647, 2004
14. J. Tang, H. Li, Y. Cao, and Z. Tang, "Email Data Cleaning," in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 489-498, Chicago, USA, 2005
15. X. Xu, Y. Lei, and Z. Li, "An Incorrect Data Detection Method for Big Data Cleaning of Machinery Condition Monitoring," *IEEE Transactions on Industrial Electronics*, Vol. 67, pp. 2326-2336, 2019
16. Y. -J. Chen and H. -Y. Chien, "IoT-based Green House System with Splunk Data Analysis," in *Proceedings of the IEEE 8th International Conference on Awareness Science and Technology* (*iCAST*), pp. 260-263, Taichung, Taiwan, 2017
17. C. Lim, N. Singh, and S. Yajnik, "A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems," in *Proceedings of the IEEE International Conference on Dependable Systems and Networks with FTCS and DCC* (*DSN*), pp. 398-403, Anchorage, AK, USA, 2008
18. Z. M. Jiang, A. Avritzer, E. Shihab, A. E. Hassan, and P. Flora, "An Industrial Case Study on Speeding up User Acceptance Testing by Mining Execution Logs," in *Proceedings of the Fourth International Conference on Secure Software Integration and Reliability Improvement*, pp. 131-140, Singapore, 2010
19. V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. van der Aalst, "Process Mining can be Applied to Software too!," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1-8,

Torino, Italy, 2014
20. C. Ioannou, A. Burattin, and B. Weber, "Mining Developers' Workflows from Ide Usage," in *Proceedings of the International Conference on Advanced Information Systems Engineering*, pp. 167-179, Tallinn, Estonia, 2018
21. M. Nottingham and R. Fielding, "Additional HTTP Status Codes," *Internet Engineering Task Force* (*IETF*), 2012
22. W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Transactions on Software Engineering*, Vol. 42, pp. 707-740, 2016

Pan Liu received the MSc degree in computer software and theory from Nanchang University in 2006 and the PhD degree in computer application from Shanghai University in 2011. Now he is a professor at Faculty of Business Information, Shanghai Business School, Shanghai, China. He is also a researcher in Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai, China. His papers have been published in some well-known international journals, and ACM and IEEE conferences. His main interests include software testing, model-based testing, formal method, and algorithm design.

Wulan Huang received the MSc degree in Circuit and system from Hunan Normal University in 2005 and the PhD degree in Management Science and Engineering from Shanghai University of Finance and Economics in 2017. Now she is a lecturer at Faculty of Business Information, Shanghai Business School, Shanghai, China. She is a principal investigator of a project supported by National Social Science Fund of China. Her main interests include algorithm design, data analysis and knowledge management.