

## **DEVICE AND METHOD FOR IDENTIFYING ERRORS IN A SOFTWARE APPLICATION**

### **TECHNICAL FIELD**

[0001] Various aspects of this disclosure relate to devices and methods for identifying errors in a software application.

### **BACKGROUND**

[0002] An essential part of developing software applications is validation and testing. In particular, errors which cause failure of an application should be identified and corrected. There exist a variety of test tools to run test cases for applications being developed. Such a test tool typically generates a log which indicates how the application has behaved under a certain test cases. Usually, it is necessary to run a high number of test cases to ensure that an application works in all kinds of situations. Accordingly, a high number of logs are created and it requires a high effort to go through the logs for finding and correcting causes for failures.

[0003] Accordingly, approaches are desirable which allow a more efficient identification and correction of causes for failures in a software application.

### **SUMMARY**

[0004] Various embodiments concern a method for identifying errors in a software application comprising obtaining a log of an execution of a plurality of test cases for a software application under test, searching for matches between regular expressions with strings in the log, wherein each regular expression is associated with an error type and, if a match of a regular expression with a string the log has been found, outputting information about the test for which the log contains the string and outputting the error type associated with the regular expression.

[0005] According to one embodiment, obtaining the log comprises receiving test result data for the plurality of test cases and parsing the log from test result data.

[0006] According to one embodiment, obtaining the log comprises testing the software application according to the plurality of test cases.

[0007] According to one embodiment, the method comprises determining the association of the error types with the regular expressions.

[0008] According to one embodiment, the information about the test comprises an indication of the test case, a test class, or both.

[0009] According to one embodiment, outputting information about the test and outputting the error type comprises writing the information about the test and the error type into a report log.

[0010] According to one embodiment, outputting information about the test and outputting the error type comprises uploading the report log to a server.

[0011] According to one embodiment, outputting information about the test and outputting the error type comprises displaying the information about the test and the error type.

[0012] According to one embodiment, displaying information about the test comprising displaying the information in association with the test in a list of tests.

[0013] According to one embodiment, displaying information about the test comprises displaying a root cause of failure assigned to the test.

[0014] According to one embodiment, the method comprises assigning the root cause in response to a user selection from a list of predefined root causes.

[0015] According to one embodiment, displaying information about the test comprises obtaining a screenshot of the user interface output of the software application under test at the time the error has occurred and displaying the screenshot.

[0016] According to one embodiment, the method comprises displaying the screenshot in response to a user request for the screenshot.

[0017] According to one embodiment, displaying information about the test comprises displaying an excerpt of the log comprising the string of the log where the match has been found.

[0018] According to one embodiment, the method comprises displaying the excerpt of the log in response to a user request for the excerpt of the log.

[0019] According to one embodiment, for at least one error type, a plurality of regular expressions is associated with the error type.

[0020] According to one embodiment, the regular expressions are regular expressions of a first set of regular expressions and the method further comprises searching for matches

between regular expressions of a second set of regular expressions with strings in the log, wherein each regular expression of the second set of regular expressions is associated with at least one of a test case having passed, a test case having failed and the start of a test case.

[0021] According to one embodiment, searching for matches between regular expressions of the first set of regular expressions with strings in the log comprises first searching for a start of a test case by searching for a match with a regular expressions of the second set of regular expressions associated with the start of a test case and then searching for matches between regular expressions of the first set of regular expressions with strings the log contains for the test case.

[0022] According to one embodiment, searching for matches between regular expressions of the first set of regular expressions with strings in the log comprises first searching for a test case having failed by searching for a match with a regular expressions of the second set of regular expressions associated with a test case having failed and then searching for matches between regular expressions of the first set of regular expressions with strings the log contains for the test case.

[0023] According to one embodiment, searching for matches between the regular expressions with strings in the log comprises going through the log and stopping going through the log when finding a match between a string of the log and a regular expression which is associated with an end of a portion of the log relevant for searching for matches of the regular expressions with strings of the logs.

[0024] According to one embodiment, the regular expressions comprise at least one regular expression which is associated with a default error type and wherein the method comprises outputting the default error type if a match of the at least one regular expression with a string in a line of the log is found unless a match of a regular expression associated with a specific error type is also found in the line of the log.

[0025] According to one embodiment, a server computer is provided comprising a communication interface, a memory and a processing unit configured to perform the method according to one of the embodiments described above.

[0026] According to one embodiment, a computer program element is provided comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method according to one of the embodiments described above.

[0027] According to one embodiment, a computer-readable medium is provided comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method according to one of the embodiments described above.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

[0028] The invention will be better understood with reference to the detailed description when considered in conjunction with the non-limiting examples and the accompanying drawings, in which:

- FIG. 1 shows a computer for development of software applications.
- FIG. 2 shows a test system according to an embodiment.
- FIG. 3 shows a report parser according to an embodiment.
- FIG. 4 shows an example of a report detail screen displayed according to an embodiment.
- FIG. 5 shows a section of the report detail screen of FIG. 4 with a drop down box being displayed for a test case.
- FIG. 6 shows a section of the report detail screen of FIG. 4 with a screenshot being displayed for a test case.
- FIG. 7 shows a flow diagram illustrating the identification of an error type from a test log according to an embodiment.
- FIG. 8 shows a flow diagram illustrating a method for identifying errors in a software application according to an embodiment.
- FIG. 9 shows a server computer system according to an embodiment.

### **DETAILED DESCRIPTION**

[0029] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure. Other embodiments may be utilized and structural, and logical changes may be made without departing from the scope of the disclosure. The various

embodiments are not necessarily mutually exclusive, as some embodiments can be combined with one or more other embodiments to form new embodiments.

[0030] Embodiments described in the context of one of the devices or methods are analogously valid for the other devices or methods. Similarly, embodiments described in the context of a device are analogously valid for a vehicle or a method, and vice-versa.

[0031] Features that are described in the context of an embodiment may correspondingly be applicable to the same or similar features in the other embodiments. Features that are described in the context of an embodiment may correspondingly be applicable to the other embodiments, even if not explicitly described in these other embodiments. Furthermore, additions and/or combinations and/or alternatives as described for a feature in the context of an embodiment may correspondingly be applicable to the same or similar feature in the other embodiments.

[0032] In the context of various embodiments, the articles “a”, “an” and “the” as used with regard to a feature or element include a reference to one or more of the features or elements.

[0033] As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items.

[0034] In the following, embodiments will be described in detail.

[0035] Fig. 1 shows a computer 100 for development of software applications.

[0036] The computer 100 comprises a CPU (Central Processing Unit) 101 and a system memory (RAM) 102. The system memory 102 is used to load program code, i.e. from a hard disk 103, and the CPU 101 executes the program code.

[0037] In the present example it is assumed that a user intends to develop a software application using the computer 100.

[0038] For this, the user executes a software development environment 104, e.g. an integrated development environment such as Xcode (for iOS), on the CPU 101.

[0039] The software development environment 104 allows the user to develop an application 105 for various devices 106, in particular smartphones. For this, the CPU 101 runs, as part of the software development environment 104, a simulator to simulate the device for which an application is developed, e.g. for an iPhone.

[0040] When the user has successfully developed an application, the user may distribute it to corresponding devices 106 via a communication network 107, e.g. distribute it to smartphones by uploading it to an app store.

[0041] Before that happens, however, the user should test the application 105 to avoid that an application that does not work properly is distributed to devices 106.

[0042] To do this, the user further runs a testing tool 108 on the CPU 101, for example XCTest. By means of the testing tool 108, the user can run multiple test cases. As a result, the testing 108 tool outputs test result data in one or more test result files, e.g. XCResult files which the user may use to find errors in the application 105 and eventually correct them.

[0043] Typically, a high number of tests are run. Accordingly, a high amount of test result data is generated and it may be cumbersome for the user to go through all the result data.

[0044] Therefore, according to various embodiments, a system for processing test result data is provided which allows more efficient analysis of test results.

[0045] FIG. 2 shows a test system 200 according to an embodiment.

[0046] The main components can be seen in a report parser (or result parser) 201 and a reporting server 204 having a report front end (FE) application 202 (also referred to as RCA (root cause analysis) FE application) and a report backend (BE) server 203 (also referred to as RCA backend server). The reporting server 204 may for example be part of a cloud, e.g. coupled to the computer 100 via the communication network 100.

[0047] The report parser 201 receives test result data 205 (e.g. an XCResult file) which is generated by a testing tool 206 testing a software application on a certain device (which may be simulated as mentioned above). The testing tool 206 and the report parser 201 may for example be implemented by the computer 100 but may also be distributed over multiple machines. They may for example be implemented by a Jenkins node. The report parser 201 uploads its results to the reporting server 204.

[0048] The report parser 201 and the reporting server 204 perform reporting of test results, analysis of test results and provide options to record the root cause for failures by combining multiple techniques (rather than simple HTML reporting). In particular, according to various embodiments, the report parser 201 parses information from test result data 205 to help a user 208 to easily understand the root cause of a failure of the software application. The user 208 may retrieve the results of the report parser 201 via the reporting server 204.

[0049] For example, the report parser parses an XCResult file and prepares a report in JSON and XML format. The RCA FE App 202 shows the report in tabular format. It may for example be up and running constantly RCA FE App 202 to allow the user 208 constant access.

[0050] The RCA BE Server 203 uses database for storing reports. It may be provided with sufficient memory to store the media files like screenshots and logs and with sufficient processing power to serve many requests concurrently. It may also be up and running constantly to allow the user 208 constant access.

[0051] In the following, the operation of the report parser 201 is described in more detail with reference to FIG. 3.

[0052] FIG. 3 shows a report parser 300.

[0053] As explained with reference to FIG. 2, the report parser 300 receives test result files 301. The test result files 301 may be the result files for tests of software applications on different devices (which may be simulated) or of different software applications. Each test result file can be seen as a raw version of a test report. For example, after execution of a test case is over, the Xcode (tool for iOS Development and Testing) generates a test report in XCResult format. The test report can be parsed by Xcode UI and command line tools.

[0054] Each test result file 301 may includes one or more logs for test cases. From a test result file 301 (e.g. an XCResult file) the report parser 300 extracts logs for the test cases and parses the log in 302. For example, for an XCResult file, the report parser 300 internally uses Xcode CLI (command line interface) to parse the log and get details like test class, method, status, duration for each test case. It uploads the log to an RCA server 309 (corresponding to report server 204) in 303.

[0055] The report parser 300 further reads the log line by line for each test case and to find out the type of error in the application (in case of a failure of the application for the test case) and the operation where the test case encountered failure (to help identification of the possible causes of failures). The report parser 300 does this in 304 using a regex mechanism, i.e. finding matches of regular expressions in the log. It then prepares a report 311 (containing the result of the parsing using the regex mechanism) and uploads this report 311 to the RCA server 309, for example in JSON format. The report parser 300 may further generate a Junit XML for reporting to a Jenkins server 310.

[0056] The report parser 300 may further find screenshot attachments in the test result data for failing cases in 307, map them to respective test cases and upload the screenshots to the RCA server 309 in 308.

[0057] According to various embodiments, the RCA FE App 202 is a web application (e.g. based on React) to show reports 311 uploaded by the report parser 300 in table format with an interactive user interface (UI). It is for example based on Material UI but any other UI framework may be used.

[0058] The RCA FE App 202 has for example a summary screen which it uses to display a Reports Summary & RCA Summary. The summary screen shows a list of reports 311 with summary (date and time, test category, job name, count of passed test cases, failed and total cases), supports pagination, shows a list of reports with RCA summary (how many failed because of Server issue, UI Issue , Script issue etc.) and has button to delete each report.

[0059] The RCA FE App 202 further has a report detail screen for showing detailed information of each test case in a report 311.

[0060] FIG. 4 shows an example of a report detail screen 400 displayed by the RCA FE App 202.

[0061] The report detail screen shows key information for each test case such as class name 401, method name 402, status 403 and duration 404.

[0062] Further, it highlights a failure tag 405 like UI, functional and framework etc. for failing cases. The user may hover with the cursor over the status to cause the RCA FE App 202 to show a failure reason.

[0063] The report detail screen 400 further provides a button 406 for each test case to cause the display of a drop down box to select root cause reasons.

[0064] FIG. 5 shows a section of the report detail screen of FIG. 4 with a drop down box 500 being displayed for a test case.

[0065] The report detail screen 400 further has a button 407 for each test case to show a screenshot for the test case and has a button 408 to show the log for the test case.

[0066] FIG. 6 shows a section of the report detail screen of FIG. 4 with a screenshot 600 being displayed for a test case.

[0067] The screenshot 600 shows the user interface of the application tested at the time of the failure.

[0068] Similarly, a log may be displayed for a test case.

[0069] Further, the report detail screen 400 has a button 409 to allow the user to enter comments for the test case.

[0070] According to various embodiments, the RCA BE Server 203 is a web server. It exposes APIs (Application Programming Interfaces) for REST calls to upload and get reports. It uses for example Django as framework Django and a Django rest API framework, Gunicorn + Nginx as deployment Configuration and MySQL as database.

[0071] It for example provides the following APIs:

api/reports => Provides list of reports with details info including case details

api/report => Provides case info for that specific report

api/summary => Provides only test summary and RCA summary of each build

api/update\_rca => PUT call to update RCA for each test case

api/upload\_logs => POST call to upload test log txt file to server

api/upload\_screen\_shot => POST call to upload screenshot to server

media => call to access media (image/txt) uploaded previously

[0072] For example, a Jenkins job which is executing test cases on node machines looks for the test result files (e.g. XCResult files) from different testing devices. Once it finds the raw reports (i.e. the test result files), it starts parsing it, merging reports from all devices and uploading it to the RCA server using the above APIs. Finally users can access the report by loading the web application in their browser. The report browser can support various platforms and may provide more information than the one described above.

[0073] As explained above, an important feature of various embodiments is the identification of type of error (leading to failures of the tested application for a test case) using a regex mechanism in 304.

[0074] FIG. 7 shows a flow diagram 700 illustrating the identification of an error type from a test log.

[0075] Upon reception of a test result file 701 (e.g. an XCResult file), the report parser 300 parses the test result file 701 in 702 to find the logs of test cases in the test result file 701. In case of an XCResult file 701, the report parser 300 may use XCParser to do this.

[0076] When the report parser 300 has found a log, it processes the log line-by-line, i.e. starts an (outer) loop over the lines in 703. In each iteration of the (outer) loop, it uses regular expressions in the following manner to process a current line.

[0077] In the present example, it uses the following regular expressions, referred to as Regex 1 to Regex 7 in the following.

Regular Expressions:

1. test\_start\_regex = r'Test [Cc]ase '\\-\\[(.+)\\s(.+)\\]' started'
2. passed\_test\_regex = r'Test [Cc]ase '\\-\\[(.+)\\s(.+)\\]' passed\\s\\((.+)\\)'
3. failed\_test\_regex = r'Test [Cc]ase '\\-\\[(.+)\\s(.+)\\]' failed\\s\\((.+)\\)'
4. assert\_fail\_regex = r'(.+)\\serror:\\s(.+) [Ff]ailed(.\*)'
5. cs\_cmd\_fail\_regex = r'(.+)\\serror code=\\s(.+)'
6. tear\_down\_regex = r'(.+)Tear Down(.+)'
7. log\_end\_failed\_regex = r'Test Suite '\\RetryTestSuite\\' failed at (.+)'
8. log\_end\_passed\_regex = r'Test Suite '\\RetryTestSuite\\' passed at (.+)'

[0078] In 704, the report parser 300 checks whether there is a match of Regex 1, whether there is a match of Regex 7 in 705 and whether there is a match of Regex 8 in 706 in the current line.

[0079] If there is a match of Regex 7, the report parser 300 stops the loop since the end of the log (indicating a fail) has been reached.

[0080] For example, the test log contains the line

Test Suite 'RetryTestSuite' failed at 2020-11-04 16:49:06.060

and thus there is a match of Regex 7

[0081] If there is a match of Regex 8, the report parser 300 stops the loop since the end of the log (indicating a pass) has been reached.

[0082] For example, the test log contains the line

Test Suite 'RetryTestSuite' passed at 2020-11-04 16:49:06.060

and thus there is a match of Regex 8

[0083] If there is a match of Regex 1, the report parser 300 parses the test class and the test case from the current line in 707. The report parser writes the result to a dictionary 722 comprising the content for the current test case for the report to be uploaded to the RCA server 309.

[0084] For example, the test log contains the line

Test Case '-[TestMicrositeTextComponent testTextContentAsSet]' started

and thus there is a match of Regex 1

[0085] When the report parser 300 has found the start of a test case in the log (by finding a match of Regex 1), it starts an inner loop in 708 to process the following lines. In each iteration of the inner loop it checks, for a current line, whether there is a match of Regex 4 in 709, whether there is a match of Regex 5 in 710, whether there is a match of Regex 2 in 711, whether there is a match of Regex 3 in 712 and whether there is a match of Regex 6 in 713.

[0086] If it finds a match of Regex 4, it checks in 714 whether one of the following UI error regular expressions matches the current line:

- query\_fail\_regex = r'Get number of matches for: Elements matching predicate \'(.+?)\'"
- query\_fail\_regex2 = r'Get all elements bound by index for: Elements matching predicate \'(.+?)\'"
- exits\_fail = r'Checking existence of (.+?)'
- find\_fails = r'Find: Elements matching predicate \'(.+?)\'"
- find\_fails2 = r'Find the\s(.+?)'
- snap\_shot\_fails = r'Using snapshot previously cached by application'

[0087] If one of these regular expressions matches, the report parser 300 considers this as a UI error having occurred and writes "ui\_err" to the dictionary 722 in 715.

[0088] If none of these regular expressions matches, the report parser 300 considers this as an error of another type and writes "other" to the dictionary 722 in 716.

[0089] For example, the test log contains the line

```
t = 282.14s Assertion Failure: TestMicrositeTextComponent.m:54: Failed to get
matching snapshot: No matches found for Elements matching predicate 'identifier ==[c]
"viewText"' from input {(
```

and thus there is a match of Regex 4

[0090] If the report parser 300 finds a match of Regex 5, it considers this as a CS Core Server) error and writes "cs\_err" to the dictionary in 717.

[0091] For example, the test log contains the line

```
2021-06-04 09:37:55.615466+0800 ShopeeSG-Runner[6735:1581092] Updating shop info
failed , error code= 19
```

and thus there is a match of Regex 5.

[0092] If the report parser 300 finds a match of Regex 2, it considers the test case as passed. It parses the duration from the log and writes the status and duration to the dictionary 722 in 718.

[0093] For example, the test log contains the line  
Test Case '-[TestMicrositeTextComponent testTextContentAsSet]' passed (44.243 seconds)  
and thus there is a match of Regex 2.

[0094] If the report parser 300 finds a match of Regex 3, it considers the test case as failed. It parses the duration from the log and writes the status and duration to the dictionary 722 in 719. Further, it parses a screen shot corresponding to the current test case (showing what the tested application has displayed at the time of the fail) in 720. It then leaves the inner loop and continues with the outer loop 703 (with the line following the current line, i.e. following the lines processed in the inner loop). When the entire suite of test cases has been parsed, the content of the dictionary 722 (e.g. converted to JSON and/or XML in 721), as well as the screen shots, are uploaded to the RCA server 309 in 723.

[0095] For example, the test log contains the line  
Test Case '-[TestMicrositeTextComponent testTextContentAsSet]' failed (283.255 seconds)  
and thus there is a match of Regex 3.

[0096] The report parser 300 determines that Regex 3 fits and can parse the duration from this line.

[0097] If the report parser 300 finds a match of Regex 6, it leaves the inner loop and continues with the outer loop 703 (with the line following the current line, i.e. following the lines processed in the inner loop). For example, the test log contains the line

t = 282.85s Tear Down

and thus there is a match of Regex 6

[0098] Further features may be added to the embodiments described above such as

- Executing tests (i.e. actively performing tests rather than merely receiving test results)
- Integration with JIRA
- Emailing Report with filled RCA
- Graphical representation test suite health
- User Authentication

[0099] In summary, according to various embodiments, a method is provided as illustrated in FIG. 8.

[00100] FIG. 8 shows a flow diagram 800 illustrating a method for identifying errors in a software application.

[00101] In 801 a log of an execution of a plurality of test cases for a software application under test is obtained.

[00102] In 802, a search for matches between regular expressions with strings in the log is performed, wherein each regular expression is associated with an error type.

[00103] If a match of a regular expression with a string the log has been found, then, in 803, information about the test for which the log contains the string and the error type associated with the regular expression are output.

[00104] According to various embodiments, in other words, regular expressions are used to find strings in a test log which allow deriving an error type that lead to a failure or a software application under test. When a match for a regular expression is found, the corresponding error type is output together (i.e. in correspondence with) information about the test case.

[00105] The approach of FIG. 8 may, as explained in context of the embodiments described above, for example be used in a tool for converting raw test result files (e.g. XCResult files) to web based test reports with ability to aggregate the test reports from other test frameworks e.g. Espresso + Junit and Selenium + Junit into a single web app, wherein the tool provides the ability to analyze and aggregate the reasons for failures and to publish the test report. Thus, the tool may implement an algorithm to convert XCResult files to Web based reports and to incorporate test reports from multiple platforms into a single source.

[00106] The tool may thus have the ability to unify reports from automation platforms like iOS, Android and web automation and to perform analysis and find out possible causes of failures. Further, it may provide the option for a user to select predefined root cause reasons and to provide custom comments for each failure. It may for example output results (for various test cases) in a table representation of Report and Root cause analysis and support maintaining older reports.

[00107] Supporting root cause analysis in the testing of a software application may reduce the requirement of manual work and thus helps automating the software development cycle. The tool provides a detailed test report which helps the developer (user) to understand the issues in test execution very quickly and fix them. It reduces the time of report analysis as it automatically identifies some of the failures by reading the log itself to find indications of errors in the log, e.g. using regex (regular expression) validations. Regular expressions which are defined in accordance with the log pattern may be used to parse the log to quickly find required information.

[00108] The approach of FIG. 8 can be used for different kinds of software automation testing like UI automation for different platforms and API automation. According to various embodiments, a tool is provided to improve the productivity and efficiency of Automation Testing lifecycle by providing a platform to visualize and document test results with the inherent ability to serve as a unified test automation report platform. It can possibly be used by any software company that utilizes Automation testing across multiple platforms (Web, Mobile, API).

[00109] The method of FIG. 8 is for example carried out by a server computer as illustrated in FIG. 9.

[00110] FIG. 9 shows a server computer system 900 according to an embodiment.

[00111] The server computer system 600 includes a communication interface 601 (e.g. configured to receive test result data or a test log and configured to output the information about the test for which the log contains the string and the error type). The server computer 600 further includes a processing unit 602 and a memory 603. The memory 603 may be used by the processing unit 602 to store, for example, data to be processed, such as the log. The server computer is configured to perform the method of FIG. 8. It should be noted that the server computer system 900 may be a distributed system comprising a plurality of computers.

[00112] The methods described herein may be performed and the various processing or computation units and the devices and computing entities described herein (e.g. the processing unit 602) may be implemented by one or more circuits. In an embodiment, a "circuit" may be understood as any kind of a logic implementing entity, which may be hardware, software, firmware, or any combination thereof. Thus, in an embodiment, a "circuit" may be a hard-wired logic circuit or a programmable logic circuit such as a programmable processor, e.g. a microprocessor. A "circuit" may also be software being implemented or executed by a processor, e.g. any kind of computer program, e.g. a computer program using a virtual machine code. Any other kind of implementation of the respective functions which are described herein may also be understood as a "circuit" in accordance with an alternative embodiment.

[00113] While the disclosure has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the appended claims. The scope of the invention is thus indicated by

the appended claims and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced.

## CLAIMS

1. A method for identifying errors in a software application comprising:  
obtaining a log of an execution of a plurality of test cases for a software application under test;  
searching for matches between regular expressions with strings in the log, wherein each regular expression is associated with an error type;  
if a match of a regular expression with a string the log has been found, outputting information about the test for which the log contains the string and outputting the error type associated with the regular expression;  
wherein obtaining the log comprises receiving test result data for the plurality of test cases and parsing the log from test result data;  
wherein the regular expressions are regular expressions of a first set of regular expressions and the method further comprises searching for matches between regular expressions of a second set of regular expressions with strings in the log; and  
associating each regular expression of the second set of regular expressions with a test case having passed, a test case having failed and the start of a test case, wherein if the regular expression is associated with a start of the test case, the method further searches for the error type.
2. The method of claim 1, wherein obtaining the log comprises testing the software application according to the plurality of test cases.
3. The method of claim 1 or 2, comprising determining the association of the error types with the regular expressions.
4. The method of any one of claims 1 to 3, wherein the information about the test comprises an indication of the test case, a test class, or both.

5. The method of any one of claims 1 to 4, wherein outputting information about the test and outputting the error type comprises writing the information about the test and the error type into a report log.
6. The method of claim 5, wherein outputting information about the test and outputting the error type comprises uploading the report log to a server.

7. The method of any one of claims 1 to 6, wherein outputting information about the test and outputting the error type comprises displaying the information about the test and the error type.
8. The method of claim 7, wherein displaying information about the test comprising displaying the information in association with the test in a list of tests.
9. The method of claim 7 or 8, wherein displaying information about the test comprises displaying a root cause of failure assigned to the test.
10. The method of claim 9, comprising assigning the root cause in response to a user selection from a list of predefined root causes.
11. The method of any one of claims 7 to 10, wherein displaying information about the test comprises obtaining a screenshot of the user interface output of the software application under test at the time the error has occurred and displaying the screenshot.
12. The method of claim 11, comprising displaying the screenshot in response to a user request for the screenshot.
13. The method of any one of claims 7 to 12, wherein displaying information about the test comprises displaying an excerpt of the log comprising the string of the log where the match has been found.
14. The method of claim 13, comprising displaying the excerpt of the log in response to a user request for the excerpt of the log.
15. The method of any one of claims 1 to 14, wherein, for at least one error type, a plurality of regular expressions is associated with the error type.

16. A server computer comprising a communication interface, a memory and a processing unit configured to perform the method of any one of claims 1 to 15.
17. A computer program element comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method of any one of claims 1 to 15.
18. A computer-readable medium comprising program instructions, which, when executed by one or more processors, cause the one or more processors to perform the method of any one of claims 1 to 15.