

DEVICE AND METHOD FOR AUTOMATED GENERATION OF PARAMETER TESTING REQUESTS

TECHNICAL FIELD

[0001] Various aspects of this disclosure relate to devices and methods for automated generation of parameter testing requests in a software application under test.

BACKGROUND

[0002] An essential part of developing software applications is validation and testing. There exist a variety of test tools to run test cases for applications being developed. Typically, for each testing parameter request (e.g., during API testing), it is essential to test whether the software (e.g., API) can handle all the various standard variable types. However, as there are many parameters and/or variables to test, users tend to spend a long time or high effort to manually craft the test code to suit a specific structure or develop testing parameter requests to test all various standard variable types, which requires a large maintenance effort.

[0003] Under such circumstances, when performing API testing, automated generation of parameter testing request is critical to achieve good return on investment (ROI) of test automation, and to achieve higher testing efficiency and to reduce overall testing duration.

[0004] Accordingly, approaches are desirable which allow more efficient testing of a software application.

SUMMARY

[0005] Various embodiments concern a method for automated generation of parameter testing requests for a software application under test including: receiving a test input, identifying a plurality of parameter testing fields in the test input, slicing the test input into a plurality of testing slices based on the plurality of parameter testing fields, generating a parameter testing request for each slice of the plurality of testing slices, and generating a combination of parameter testing requests from the parameter testing request for each slice of the plurality of testing slices.

[0006] According to one embodiment, each slice of the plurality of testing slices corresponds to a corresponding parameter testing field of the plurality of parameter testing fields.

- [0007] According to one embodiment, the method includes generating the parameter testing request for each slice of the plurality of testing slices based on the corresponding parameter testing field of the plurality of parameter testing fields.
- [0008] According to one embodiment, the method includes identifying the plurality of parameter testing fields in the test input by permutation of an index field of a parameter testing field in the plurality of parameter testing fields.
- [0009] According to one embodiment, the test input is a non-proto test input.
- [0010] According to one embodiment, the software application is written in Golang.
- [0011] According to one embodiment, the method includes generating a plurality of test cases from the combination of parameter testing requests.
- [0012] According to one embodiment, the method includes using the plurality of test cases for API testing of the software application under test.
- [0013] According to one embodiment, the method includes receiving test result data for the plurality of test cases in a report.
- [0014] According to one embodiment, the method includes receiving test result data by writing information about the plurality of test cases into the report.
- [0015] According to one embodiment, the method includes identifying a status of each test case of the plurality of test cases from the test result data.
- [0016] According to one embodiment, the method includes determining a source of error if the status of a test case of the plurality of test cases indicates a fail status.
- [0017] According to one embodiment, a server computer is provided including a communication interface, a memory and a processing unit configured to perform the method according to one of the embodiments described above.
- [0018] According to one embodiment, a computer program element is provided including 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.
- [0019] According to one embodiment, a computer-readable medium is provided including 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

[0020] 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 according to an embodiment.
- FIG. 2 shows a testing tool according to an embodiment.
- FIG. 3 shows a flow diagram illustrating a method for automated generation of parameter testing requests in a software application under test according to an embodiment.
- FIG. 4 shows a server computer system according to an embodiment.

DETAILED DESCRIPTION

[0021] 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.

[0022] 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.

[0023] 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.

[0024] 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.

[0025] As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items.

[0026] In the following, embodiments will be described in detail.

[0027] Fig. 1 shows a computer 100 for development of software applications.

[0028] The computer 100 includes a CPU (Central Processing Unit) 101 and a system memory (RAM) 102. The system memory (e.g., a random-access memory (RAM)) 102 is used to load program code, i.e. from a hard disk (i.e, HDD) 103, and the CPU 101 executes the program code.

[0029] In the present example it is assumed that a user intends to develop a software application using the computer 100. For this, the user executes a software development environment 104 on the CPU 101. The user may use a computing language such as Golang, which is a statically typed, compiled programming language, to develop the application.

[0030] The software development environment 104 allows the user to develop an application 105 for various devices 106, for example 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 a mobile device.

[0031] 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.

[0032] 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. To do this, the user further runs a testing tool 108 on the CPU 101, for example using a Golang testing framework for example to perform automated testing and/or automated generation of parameter testing requests on Application Programming Interfaces (APIs).

[0033] In a typical automation testing, users tend to spend a long time or high effort to write or develop testing parameter requests to test all various standard variable types. This inevitably reduces testing efficiency. There is no existing tool or library to generate a combination of parameter testing requests of a specified testing scope from an unknown request structure. To achieve higher automation ROI and testing efficiency, an automated generation of parameter testing requests feature is highly desirable.

[0034] Therefore, according to various embodiments, a system for automated generation of parameter testing requests is demonstrated. This allows better testing efficiency.

[0035] FIG. 2 shows a testing tool 108 according to an embodiment.

[0036] In various embodiments, the testing tool 108 may be used for automated generation of parameter testing requests.

[0037] In various embodiments, the testing tool 108 may include an input/output (I/O) module 202. The I/O module 202 may allow the testing tool 108 to communicate with other modules in or outside of computer 100. The I/O module 202 may include a user interface for user control or user input of the testing tool 108. The user interface may include, for example, computing peripheral devices such as display monitors, user input devices, for example, touchscreen devices and computer keyboards. The I/O module may receive a test input, which may be code for example in Golang. The test input may be in a form of a structure such as a Golang structure. The structure or struct may be a user-defined type or array that allows to group/combine items of possibly different types into a single type. The structure may include or may be a code with a plurality of parameters.

[0038] In various embodiments, the testing tool 108 may include an identification processor 204 which may identify a plurality of parameter testing fields in the test input structure.

[0039] In various embodiments, the testing tool 108 may include a slicing processor 206 which may slice the test input into a plurality of testing slices based on the plurality of parameter testing fields. Each testing slice may be a part or a portion of the test input. Each slice is a flexible and extensible data structure to implement and manage collections of data. Slices may be made up of multiple elements, all of the same type. Each slice of the plurality of testing slices may be based on a corresponding parameter testing field of the plurality of parameter testing fields. In other words, each slice may include data from the corresponding parameter testing field.

[0040] In various embodiments, the testing tool 108 may include a request generator 208 which may generate a parameter testing request for each slice of the plurality of testing slices. The parameter testing request for each slice may be based on the corresponding parameter testing field of the plurality of parameter testing fields.

[0041] In various embodiments, each parameter testing request may be a request for a set of test cases for each slice.

[0042] In various embodiments, the request generator 208 may generate a combination of parameter testing requests from the parameter testing request for each slice of the plurality of

testing slices. The combination of parameter testing requests is a combination of the parameter testing request of each slice for the plurality of testing slices.

[0043] In various embodiments, the combination of parameter testing requests may be a combination of all sets of testing cases of all slices, wherein each slice has its own set of testing cases.

[0044] In various embodiments, the identification processor 204 may identify the plurality of parameter testing fields in the test input by permutation of an index field of a parameter testing field in the plurality of parameter testing fields. Permutation may mean iteratively making small changes of the index field, and checking if the changed index field is in the test input structure.

[0045] In various embodiments, the test input is a non-proto test input. The non-proto test input does not have a predefined structure, and may have an unknown request structure which is not predefined in a software library. In various embodiments, the testing tool 108 may be able to recognize different types of structures dynamically.

[0046] In various embodiments, the testing tool 108 may include a memory 210, which may store the parameter testing request for each slice of the plurality of testing slices and/or the combination of parameter testing requests.

[0047] In various embodiments, the testing tool 108 may include a testing processor 212 to generate a plurality of test cases from the combination of parameter testing requests. The plurality of test cases may be stored in the memory 210.

[0048] In various embodiments, the testing processor 212 may use the plurality of test cases for API testing of the software application under test. In various embodiments, the plurality of test cases may be in a test suite (i.e., validation suite). The test suite may be a collection of test cases that may be intended to be used to test a software application to show that it has some specified set of behaviours.

[0049] In various embodiments, the testing processor 212 may generate test results of the plurality of test cases under test. The test results may be stored in the memory 210. The testing processor 212 may parse the report from test result data. In an embodiment, parsing the log from the test result data may include error detection or source of error analysis from the test result data. The report may be parsed by command line tools.

[0050] In various embodiments, the user may receive the test result data for the plurality of test cases in a report through the I/O module 202. The testing tool 108 may write information

about the plurality of test cases into the report. For example, information about the test may be written into the report which may be in JSON or XML format. The log may be shown in a tabular format. In various embodiments, testing tool 108 may display information about the test in a list or table of tests.

[0051] In various embodiments, the testing processor 212 may identify a status of each test case of the plurality of test cases from the test result data. The status may be a pass status or a fail status.

[0052] In various embodiments, the testing tool 108 may include an error detection processor 214 which may determine a source of error if the status of a test case of the plurality of test cases indicates a fail status. The source of error may be interim failures or product code failure.

[0053] In various embodiments, information about the test and/or the source of error may be uploaded to a server.

[0054] In various embodiments, a server computer is provided including a communication interface, a memory and a processing unit configured to perform the method according to one of the embodiments described above.

[0055] In various embodiments, a computer program element is provided including 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.

[0056] In various embodiments, a computer-readable medium is provided including 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.

[0057] In various embodiments, the testing tool 108 may use the following exemplary source code to generate parameter testing requests:

[0058] GenerateRequests generates a slice of parameter testing requests from the testing input structure (input strut) by the permutation of all fields:

```
func (rpt *ReflectParameterTester) GenerateRequests(inputStructInterface interface{})
(outputSliceInterface interface{})
```

Calls buildRequests.

[0059] In various embodiments, the input may be a singular request with or without default value(s), e.g. pb_order_action.GetOrderExtraDataResponse{ }.

[0060] In various embodiments, the output may return an interface of a slice of requests. An interface may be named collections of method signatures or may be the parameter request for each slice.

[0061] In various embodiments, `GenerateRequests` may be a wrapper for `GenerateRequestsByField()`, which may mean that `GenerateRequests` will use or will contain `GenerateRequestsByField()`.

[0062] In various embodiments, software clients may assert the output back to perform an intended purpose e.g. `outputSliceInterface.([]pb_order_action.GetOrderExtraDataResponse)`.

[0063] In various embodiments, `GenerateRequestsByField` may generate a slice of parameter testing requests from the test input structure by the permutation of a specified field index.

[0064] In various embodiments, the testing tool 108 may use the following exemplary source code to generate parameter testing requests of a specific field:

[0065] `func (rpt *ReflectParameterTester) GenerateRequestsByField(inputStruct interface{}, fieldIndex int) (outputSliceInterface interface{})`.

[0066] In various embodiments, the input may be a singular request with or without default value(s), e.g. `pb_order_action.GetOrderExtraDataResponse{}`, and an index field of the specific field.

[0067] In various embodiments, the output may return an interface of a slice of requests of the specific field.

[0068] In various embodiments, the source code may include internal logic such as `BuildRequestsByField` based on the field index to build each parameter request of each field index.

[0069] In various embodiments, software clients may assert the output back to perform an intended purpose, e.g. `outputSliceInterface.([]pb_order_action.GetOrderExtraDataResponse)`.

[0070] In various embodiments, the slices of the test input provided by the users are generated, and then populate with values using the generator with user configurations. The user may select a scope of parameter testing and the slices will be populated with values in the scope of parameter testing (i.e., permutation of the index field of a parameter testing field).

[0071] In various embodiments, the parameter testing request may be generated for each slice. The system (e.g., API) may concatenate all the different permutations of each slice and

may return an output to the user as an array of requests which may be a set of tests for each parameter).

[0072] In summary, according to various embodiments, a method is provided as illustrated in FIG. 3.

[0073] FIG. 3 shows a flow diagram 300 illustrating a method for automated generation of parameter testing requests for a software application.

[0074] In 302, a test input may be received, e.g., by the I/O module 202.

[0075] In 304, a plurality of parameter testing fields in the test input may be identified.

[0076] In 306, the test input may be sliced into a plurality of testing slices based on the plurality of parameter testing fields.

[0077] In 308, a parameter testing request may be generated for each slice of the plurality of testing slices.

[0078] In 310, a combination of parameter testing requests may be generated from the parameter testing request for each slice of the plurality of testing slices.

[0079] Steps 302 to 310 are shown in a specific order, however other arrangements are possible. Steps may also be combined in some cases. Any suitable order of steps 602 to 612 may be used.

[0080] The approach of FIG. 3 can be used for different kinds of software automation testing like API automation for different platforms. According to various embodiments, a tool is provided to improve the productivity and efficiency of Automation Testing lifecycle by providing a tool to perform rerun of test cases with the inherent ability to serve as a unified test automation report platform. It may be used by any software company that utilizes Automation testing across multiple platforms (Web, Mobile, API), especially for software applications in Golang format.

[0081] The method of FIG. 3 is for example carried out by a server computer as illustrated in FIG. 4.

[0082] FIG. 4 shows a server computer system 400 according to an embodiment.

[0083] The server computer system 400 includes a communication interface 401 (e.g. configured to receive a test input structure or test result data or a report and configured to output information about the test for example status of test cases or source of error). The server computer 400 further includes a processing unit 402 and a memory 403. The memory 403 may be used by the processing unit 402 to store, for example, data to be processed, such as the report

or parameter requests or test cases. The server computer is configured to perform the method of FIG. 3. It should be noted that the server computer system 400 may be a distributed system including a plurality of computers.

[0084] 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 402) 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.

[0085] 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 automated generation of parameter testing requests for a software application under test comprising:
 - receiving a test input;
 - identifying a plurality of parameter testing fields in the test input;
 - slicing the test input into a plurality of testing slices based on the plurality of parameter testing fields;
 - generating a parameter testing request for each slice of the plurality of testing slices;
 - and
 - generating a combination of parameter testing requests from the parameter testing request for each slice of the plurality of testing slices.
2. The method of claim 1, wherein each slice of the plurality of testing slices corresponds to a corresponding parameter testing field of the plurality of parameter testing fields.
3. The method of claim 2, further comprising:
 - generating the parameter testing request for each slice of the plurality of testing slices based on the corresponding parameter testing field of the plurality of parameter testing fields.
4. The method of any of claims 1-3, further comprising identifying the plurality of parameter testing fields in the test input by permutation of an index field of a parameter testing field in the plurality of parameter testing fields.
5. The method of any of claims 1-3, wherein the test input is a non-proto test input.
6. The method of any one of claims 1-5, wherein the software application is written in Golang.
7. The method of any of claims 1-6, further comprising generating a plurality of test cases from the combination of parameter testing requests.

8. The method of claim 7, further comprising:
using the plurality of test cases for API testing of the software application under test.
9. The method of claim 8, further comprising:
receiving test result data for the plurality of test cases in a report.
10. The method of claim 9, further comprising:
receiving test result data by writing information about the plurality of test cases into the report.
11. The method of claim 11, further comprising:
identifying a status of each test case of the plurality of test cases from the test result data.
12. The method of claim 12, further comprising:
determining a source of error if the status of a test case of the plurality of test cases indicates a fail status.
13. 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 12.
14. 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 12.
15. 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 12.