Test design automation is key to improve productivity

While it is undoubtedly accepted that test automation is the key to increased quality and productivity, the act of test design is often the completely overlooked aspect of test automation. Test design is a separate and distinct task and is a powerful method to boost your test automation's bug detection capabilities.

In this whitepaper, we will discuss two very exciting and powerful automated test design methods in Qentinel Pace. We will talk about their theory and how you can leverage them in your test automation to fool proof your application against errors which were earlier untouched by your test automation.

Test design automation

At large, test design concerns with making the decisions on

1 What to test in the first place?

2 How to stimulate the application and with what test data values?

3 How the application should react and respond to the stimuli provided?

Test automation, even still today, is primarily focused on automating test management and test execution while test design remains largely a manual activity.

Automated test design process significantly reduces the functional testing efforts while at the same time increasing the quality of testing. Not to mention that the quality of your testing is strongly interconnected with the ability to find flaws in the application being tested. And for that specific purpose you need to strengthen the fault detection capability of your test automation. Qentinel Pace leverages efficient and effective test design techniques for designing test scenarios that are strong in their fault detection abilities. Not only do these test design automation capabilities yield measurable improvements in quality, they are extremely straightforward to take into use. Let's look at them closely.

1. Combinatorial Testing

Combinatorial testing is based on the premise that many errors in software can only arise from the interaction of two or more parameters. This is called interaction principle, which states that most software failures are induced by single "factor" faults or by a combinatorial effect of two factors, with progressively fewer failures induced by interactions between more factors. This is a practical hypothesis suggesting that if there is a fault that manifests with a specific setting of input variables, it is most likely caused by only a small subset of those variable values. This implies that software faults can be discovered by relatively simple and small tests.

For example, assume that we have 3 input variables OS, Browser, and JavaScript with possible values:

```
OS = Linux, Windows, iOS, Android
Browser = chrome, Firefox, Edge
JavaScript = enabled, disabled
```

With pairwise testing we aim to generate tests where two variables are in "interaction". All the the interacting pairs are automatically identified for example, (Linux, Chrome), (Linux, Firefox), ..., (Windows, Chrome), ..., (Chrome, enabled) and (Chrome, disabled). After which we create a small subset of test cases covering all the interacting pairs. It has been shown empirically that by choosing the input data values this way, we significantly increase the likelihood of finding software faults while keeping the number of test cases relatively small. Of course, there is no reason why coupling two variables would be always the best strategy and therefore Qentinel Pace does not only support pairwise testing, but more general N-wise testing approach, where the user can freely choose the value for N.

2. Test Data Generation

We are often faced with problems where we need to test the application with valid and invalid email addresses, IBAN bank account numbers, social security numbers, string encoded IP addresses, and so on. While combinatorial testing is an excellent tool for finding flaws in tested applications, combinatorial testing as such does not address the above-mentioned problem, that is the problem of test data design. Experimental evidence and practical experience reveal that it is extremely difficult to create sufficient and proper test data for the design of test cases that comprehensively covers the software logic for any non-trivial software system. This becomes a major part of test design that takes significant effort, experience and skill to excel manually. To alleviate this fundamental problem in test design, Qentinel Pace includes a unique test data generation ability that allows extremely easy-to-use while highly efficient way of automatically generating test data for testing common patterns such as email addresses and IBAN numbers, mentioned above.

The test data generation algorithm of Qentinel Pace deploys a systematic analysis approach for automatically identifying and generating test cases for covering “corner cases” of various types of data. The algorithm does not generate test data in random but instead it can be considered to be a “boundary value analyser” generalized to arbitrary data patterns as it is capable of identifying corner cases that are crucial to analyse and verify thoroughly during the quality assurance process.

Empower your test automation with test design automation

As you might have guessed correctly, Qentinel Pace is running very powerful algorithms and solving multiparameter optimization problems in background. Let’s look at how you can leverage them in your test cases to unleash a whole new test coverage of otherwise untouched or untested use cases of your application.

1. Combinatorial Testing
Test are generated from one or more test data combination. For example, let’s say you have a CRM system say Salesforce for which you have built a new integration and you would like to test that this integration works end-to-end faultlessly. The first step is typically to fill a form and enter first name. In your automated test case using PaceWords this step will be

```
TypeText     First name     John
```

Now, if you want to generate test cases using combinatorial testing, you would need to provide a data set for the test case. The alternative data values are expressed in the Paceword test cases using “list notation” where alternative values are placed between square brackets and separated by commas as follows:

```
TypeText     First name     [John, Jane,
                            Harry, Mary]
```

The above defines that there are 4 alternative data values for First name, namely John, Jane, Harry and Mary. Depending on the “combinatorial test generation mode”, the approach that the test generator employs in the test generation, will vary. Irrespective of the chosen test generation mode, the generated tests will always have one of the data values selected. That is, the generated test cases may for example have

```
TypeText     First name     John
```

There are two combinatorial test generation modes; linear and nwise.

Linear option (on by default) instructs the test generator apply a linear data selection mechanism where Qentinel Pace generates a test collection where each test data value is tested at least once. For example

```
My test case
   [Tags]       testgen       linear
   TypeText     First name    [John, Jane]
   TypeText     Last name     [Johnson,
                              Janeson]
```

Would produce two test cases, one with data combination (John, Johnson) and second with (Jane, Janeson). As a note, testgen tag informs Qentinel Pace your intent to use test generation.

N-wise option, on the other hand, enables the combinatorial test data generator and it is provided with a numeric argument which defines the number of variables interacting. Pairwise testing, for example, is enabled by setting nwise=2

```
My test case
    [Tags]        testgen        nwise=2
    TypeText      First name     [John, Jane]
    TypeText      Last name      [Johnson,
                                 Janeson]
```

The example above would generate 4 test cases with data combinations (John, Johnson), (John, Janeson), (Jane, Johnson) and (Jane, Janeson). Yet another example is shown below where there are more than two fields with alternative data values.

```
My test case
    [Tags]        testgen        nwise=2
    TypeText      First name     [John, Jane]
    TypeText      Last name      [Johnson,
                                 Janeson]
    TypeText      Company        [Qentinel,
                                 ACME, Inc
                                 Incorporated]
```

From which we will get 6 test cases with data values (John, Johnson, Qentinel), (Jane, Janeson, Qentinel), (Jane, Johnson, ACME), (John, Janeson, ACME), (John, Janeson, Inc Incorporated) and (Jane, Johnson, Inc Incorporated).

Drawing from our vast datasets, we recommend pairwise testing (nwise=2 for conducting combinatorial testing. However should there be a need to have more than two interacting variables, nwise can be configured accordingly, for example nwise=3 or nwise=4.

2. Test Data Generation

Test data generation capability are used by supplying Pacewords with predefined test data modifiers, which cause the test generator to trigger a proprietary test data generation algorithm. As alluded earlier, this algorithm systematically analyses the provided test data pattern and creates test cases for covering each “corner case” of the pattern. Therefore, the algorithm can be considered to be a “boundary value analyser” generalized to arbitrary data patterns.

Drawing from our Salesforce integration test example, let’s say you have to test that your form excepts all valid email addresses and show a message ‘Valid email address provided’. Your test case will look like as shown below

```
My test case
    [Tags]        testgen
    TypeText      Email          VALID_EMAIL_
                                 ADDRESS
    VerifyText    Valid email address provided
```

Here, the predefined test data modifier VALID_EMAIL_ADDRESS instructs the test data generation algorithm to create a collection of test cases where (1) each generated test case contains a valid email address in such a fashion that (2) each distinct test case verifies a unique feature or corner case of email address pattern. Qentinel Pace will systematically create tests for verifying that all possible corner cases of email validation functionality are carefully covered. The end user only needs to supply Pacewords with the specific modifier, in this case VALID_EMAIL_ADDRESS and the algorithm fully automates the test creation.

Furthermore, the test data generation expands to negative and invalid test data. That is, the algorithm can be used to generate test data that might look valid on surface level, but in fact is somehow wrongly formatted. This ensures that your application behave properly not only for valid data but also handles invalid data values properly by rejecting them gracefully. Invalid test data is generated simply by prefixing the test data modifier with INVALID instead of VALID as shown below

```
My test case
    [Tags]        testgen
    TypeText      Email          INVALID_
                                 EMAIL_ADDRESS
    VerifyText    Invalid email address provided
```

There is an extensive list of such modifiers, which are used most often.

3. Combinatorial testing and test generation

You do not need to stop here, the test data modifiers can be used in conjunction with combinatorial test generation as well. Let's take a test case wherein, you first want to enter a first name and then a last name from combination of two data values each and then a valid email address.

```
My test case
    [Tags]        testgen        nwise=2
    TypeText      First name     [John, Jane]
    TypeText      Last name      [Johnson,
                                 Janeson]
    TypeText      Email          VALID_EMAIL_
                                 ADDRESS
```

And later append that test case to test you application's behaviour for invalid IBAN.

```
My test case
    [Tags]        testgen        nwise=2
    TypeText      First name     [John, Jane]
    TypeText      Email          VALID_EMAIL_
                                 ADDRESS
    TypeText      IBAN           INVALID_IBAN
    VerifyText    Valid email address but invalid
                  IBAN provided
```

Test Generation in Qentinel Pace

Test generation happens right before test execution. Qentinel Pace looks at your test cases and determine whether or not you have called for a test generation. Should you have warranted any auxiliary test data generation or combinatorial testing, Qentinel Pace will generate the necessary test cases and schedules the generated test cases for test execution. You could define a maximum number of tests you want Qentinel Pace to generate for you, by default it is set to 100.

Once test generation ends, you are notified about the generated test cases, which are scheduled for execution. You can analyse the generate test cases and gain further insights into the generated tests. Should you realise that you have generate far too many cases by mistake and you would not necessarily want to execute them all, you may abort the execution process. Moreover, should you realize that a set of generated test case is of special significance, you could include it in your regression test set for good.

Qentinel | Robotic Software Testing

www.qentinel.com
info@qentinel.com