**Test Plan for Ataccama TestCalc application**

**Test Description**

* Performing math operations (addition, subtraction, division, multiplication).
* Verify absence of errors and exceptions in case of valid input values.
* Verify stability of the application.

Record operation result. Compare with the result from the external calculator (e.g. Windows Calculator).

**Test Environment for the UI tests**

According to <https://gs.statcounter.com/>, the most popular browser is Chrome (64.45%). So, the UI tests will be run against Chrome browser (especially manual test). Also, the automated tests can be ran for Firefox, too, if it’s required.

**Test parameters**

**Value1:** integer values from -2147483648 to 2147483647

**Value2:** integer values from -2147483648 to 2147483647

**Operations:** addition, subtraction, multiplication, division

The main challenge is to choose a valid set of test values to cover all risky combinations. To best way to achieve this is to use such Test Design techniques as Equivalence Partitioning and Boundary Value Analysis.

All data can be divided into the following equivalence classes:

1. **Below zero**. Values from -2147483648 to 1.
2. **Zero**. Contains only one value – 0.
3. **Above zero**. Values from 1 to 2147483647.
4. Integers with **leading zero**, e.g. 0000012345689 – zeroes should be removed automatically.

In addition, there can be considered а negative class, which contain values, causing the exception in the calculator.

1. **Negative**.

* -2147483649, 2147483648 – values out of integer range
* empty values
* non-integer values (e.g. 1.0)
* strings (e.g. “aBd%##@(%)@”)
* valid integers with trailing empty strings, e.g. “1111111 “
* zero division

Final classes:

1. **Below zero**:

-2.147.483.648 boundary value

-46.340

-215

-15

-1 boundary value

1. **Above zero**:

2.147.483.647 boundary value

46.340

215

15

1. boundary value
2. **Zero:** 0
3. **Negative**:

-2147483648 Integer.MIN\_VALUE - 1

2147483648 Integer.MAX\_VALUE + 1

99.0 Non-integer

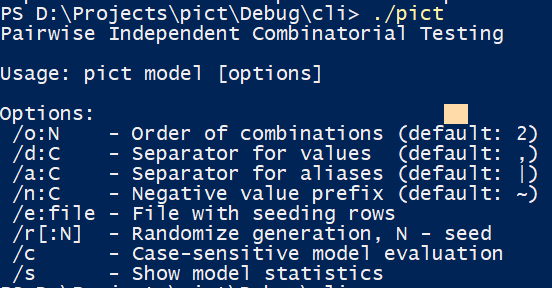
!@#$%^&\*() as8o5 String

123 34 3 Invalid number with spaces

“ “ Empty string

For decreasing the number of test combinations was used **Pairwise Testing** technique.

Pairwise testing (a.k.a. all-pairs) is an effective test case generation technique that is based on the observation that most faults are caused by interactions of at most **two** factors.

Pairwise-generated test suites cover **all combinations** of two therefore are much smaller than exhaustive ones yet still very effective in finding defects.

For generating test parameters combinations was used tool PICT (Pairwise Independent Combinatorial Testing tool) by Microsoft

**PICT model for positive combinations**:

**Operations**: addition, subtraction, multiplication, division

**Value1**: -2147483648, -46340, 215, 15, 1, 0, 2147483647, 46340, 215, 1

**Value2**: -2147483648, -46340, 215, 15, 1, 0, 2147483647, 46340, 215, 1

Combinations: 180

Generated tests:100 (test number was reduced after removing some invalid values, e.g. zero-division).

**Test data sample**

|  |  |  |
| --- | --- | --- |
| **Operations** | **Value1** | **Value2** |
| addition | 215 | 15 |
| addition | 0 | 1 |
| addition | 2147483647 | 215 |
| addition | 1 | 1 |
| addition | 46341 | 2147483647 |
| addition | 15 | 46341 |
| addition | 215 | 215 |
| addition | -46340 | 0 |
| addition | 215 | 0 |
| addition | 2147483647 | -46340 |
| addition | 1 | 15 |
| addition | 215 | -2147483648 |
| addition | -2147483648 | -46340 |
| addition | 215 | 215 |
| addition | 215 | 1 |
| addition | 1 | -46340 |
| addition | -46340 | 1 |
| addition | 0 | 0 |
| addition | 1 | 0 |
| addition | 215 | 2147483647 |
| addition | -46340 | 15 |
| addition | 1 | 1 |
| addition | 1 | 15 |
| addition | 0 | 2147483647 |
| addition | -2147483648 | 215 |
| addition | -2147483648 | 1 |
| addition | 215 | 15 |
| addition | -46340 | 215 |
| division | 0 | 215 |
| division | 2147483647 | 1 |
| division | 15 | 215 |
| division | 46341 | 15 |
| division | 215 | 1 |
| division | 0 | -46340 |
| division | -2147483648 | 15 |
| division | -46340 | -2147483648 |
| division | 2147483647 | 1 |
| division | 46341 | -46340 |
| division | 215 | -2147483648 |
| division | 215 | 215 |
| division | -46340 | 2147483647 |
| division | 46341 | 1 |
| division | 46341 | -2147483648 |
| division | -46340 | 1 |
| division | 15 | 215 |
| multiplication | 1 | 0 |
| multiplication | 46341 | 215 |
| multiplication | 0 | -2147483648 |
| multiplication | 1 | -46340 |
| multiplication | 215 | 1 |

**Manual test cases examples**

**\* \* \***

*Preconditions for each test case*:

* Installed Apache Tomcat web server
* Application .war file stored into Tomcat\webapps folder
* Web server is launched and works correctly
* Installed the Chrome browser

**TC 1 - positive**

*Steps*:

1. Open the browser, go to URL <http://localhost:8080/testCalc/webUI>
2. Select **addition** radio button
3. Value1 🡪 Enter a valid integer number
4. Value2 🡪 Enter a valid integer number
5. Click **Calculate**

*Expected Results*:

1. Application launched successfully
2. Operation radio button selected
3. **Value1** populated successfully
4. **Value2** populated successfully
5. Calculated result is correct, there is no exceptions

Repeat the test for all generated positive test conditions

**TC 2 - negative**

*Steps*:

1. Open the browser, go to URL <http://localhost:8080/testCalc/webUI>
2. Select **multiplication** radio button
3. **Value1** 🡪 Enter invalid value (empty string, non-integer, too big value, etc.)
4. **Value2** 🡪 Enter a valid integer number
5. Click **Calculate**

*Expected Results*:

1. Application launched successfully
2. Operation radio button selected
3. **Value1** populated successfully
4. **Value2** populated successfully
5. There is the exception thrown

Repeat the test for all generated negative test conditions

**Test Automation Approach**

**Test Automation Framework for ESET Log Collector**

**Technologies stack**

For the automation of the app was chosen Java. The main reason of this choice – the application under test also uses Java, and it has a lot of benefits, such as:

1. Same programming language fosters collaboration between developers and QA engineers.
2. Same programming language makes software environment setup easier: the same IDE, the same CI server, the same development environments.

For the UI testing was used Selenium WebDriver.

For the API testing was used REST Assured – a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs.