**REPORT**

We perform mutation testing on the roman-convertor file and test suites

Developed. Initially we divided the test suites into following:

1. **Interface testing for integers**
2. **Interface testing for strings**
3. **Functionality testing for Arabic numbers**
4. **Functionality testing for roman numerals**

We have written test suites based on above categories. Each test suite was individually

Considered for mutation testing and the results were noted.

We had 4 test suites separated based on the above characteristics:

Table

Description automatically generated

1. **Interface testing for integers**

In this test suite we usually cover the interface testing characteristics of the

Integers in general.

The following result was obtained:

Timeline

Description automatically generated with medium confidence

1. **Functionality testing for Arabic numbers**

In this test suite we covered the functionality characteristics of the integers. It specifies

How integers should behave with respective to the roman numeral convertor program.

The following result was obtained:

Timeline

Description automatically generated

1. **Interface testing for strings**

In this test suite we usually cover the interface testing characteristics of the

Strings in general.

The following result was obtained:

Timeline

Description automatically generated

1. **Functionality testing for Roman numerals**

In this test suite we covered the functionality characteristics of the strings. It specifies how strings should behave with respective to the roman numeral convertor program.

The following result was obtained:

Timeline

Description automatically generated

**Mutation testing for roman convertor with 100%:**

**Graphical user interface

Description automatically generated with medium confidence**

**Analysis**

Interface Testing for Integers test suite performed better compared to others.

We are familiar with the integer properties, and we designed interface based

Characteristic blocks and determined what could be possible unit tests for it.

Hence it has better mutation coverage than others.

Common points from all the reports are:

All the reports have the line coverage and if there are any value mutations

Then those value mutations can be easily killed.

Unlike conditional operators’ mutation where sometimes it can lead to

Survival of the mutation.

There are three mutations which occurred above:

1. Value mutation
2. Decision mutation
3. Statement mutation

There are value mutations which are easy to kill, and the statement and decision are hard to kill

Comparatively. Conditional boundary mutation operators in the above case proved that it is harder to kill, if not handled properly by the test suites since it survived.

Yes, we were able to achieve 100%. The program here has smaller functionality, and it was

Easy to determine the test suites and tried to cover as many cases as possible in order to

Achieve good confidence.