Software testing techniques

Denys Dushyn
Feb 26 · 3 min read

Dear colleagues, engineers, developers, and coders.

This article is the first one in the series of articles about software quality. And especially about software testing techniques that should lead us to better test case design and better software quality.

Below is a compilation of software techniques that are of high importance during test case design for a new component/system/service or class. It does not matter what framework you are going to use for unit tests. The more important here is how to select a meaningful subset from a set of all test inputs(test vectors).

I stick to the idea that the most efficient test cases have to be designed, not just randomly selected from a set of all test cases.

I prefer to split all test vectors into four categories/groups/types. The purpose of grouping is to make software testing systematic and quickly identify all possible test cases in a particular group. So far, the classes are the following:

1. *Unit Testing*
   Scopes are functions/classes
2. *Integration Testing*
   Scope is the interaction between classes and components
3. *System Testing*
   Our running software
4. *Acceptance Testing*
   All tests, which when completed successfully, will result in the customer accepting the software.

After a deep dive into the testing theory and long conversations with manual/automation QAs guys, we have concluded that the same techniques should be used during software development. And these techniques are:

Main techniques

Equivalence partitioning

It is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. These data rages are called equivalence classes. An equivalence class consists of a set of data that is treated the same by the module or that should produce the same result.

https://en.wikipedia.org/wiki/Equivalence_partitioning

Boundary Value Analysis

It is a software testing technique in which tests are designed to include representatives of boundary values in a range. BVA focuses on the boundaries because that is where so many bugs hide. The most interesting here is that defects can be in the requirements. These techniques and Equivalence partitioning form the solid base for Domain Analysis Testings and for Pairwise Testing.

https://en.wikipedia.org/wiki/Boundary-value_analysis

Domain Analysis Testing

It is a generalization of Equivalence partitioning and Boundary Value Analysis. We usually use EP and BVA techniques when we have only one variable. The domain analysis allows the testing of multiple variables simultaneously because useful programs, in most cases, produce output from more than one input variable.

https://www.slideshare.net/LakshminarayananNeel1/domain-analysis-in-software-testing

Pairwise Testing

The culmination of the previous ones.
It is a combinatorial method of software testing that, for each pair of input parameters to a system, tests all possible discrete combinations of those parameters. Using carefully chosen test inputs, this can be done much faster than an exhaustive search of all combinations of all parameters, by “parallelizing” the tests of parameter pairs.

For more information:
Definition of pairwise testing of wiki
A good article with a description of the current state
A collection of tools for pairwise testing

Additional techniques

Some techniques show a system from a different perspective and allow the development of test cases that can be a good base for acceptance test suit or regression test suit.

Decision Table Testing

A decision table is a good way to deal with different combination inputs with their associated outputs. It is a black box test design technique to determine the test scenarios for complex business logic. It is applicable when you have clearly stated business rules. If so, then it is a good idea to use decision tables for test case preparation.

Decision tables on the wiki
One more example

State-Transition Testing

If the software has a notion of a state and some parts of system behavior can be represented as states, then it is an excellent option to create state diagrams and derive test cases from it. There are several options for how to choose good test cases.
A definition and explanation

Notes

It is possible to achieve a high quality of code by combining the mentioned techniques. They are battle-tested and straightforward. It is better to start thinking about quality before the implementation. Just do it and relax.

Happy high-quality code :)

Software Testing Software Development Testing Techniques

About Help Legal