Software Testing Notes

Found at: benjaminshaw.uk

1 Terms

Black-Box

Examine the functionality of an application without looking at its inner workings.

White-Box

Testing the internal structures of an application

ITF: Independently Testable Features

Features that should be tested separately.

2 Functional Testing

Specification-based testing, black-box testing

A type of testing that focuses on *what* the system does. Test cases are derived on the specification of the component under scrutiny.

Functional testing does not look at methods of a module, rather it tests a *slice of the whole program's functionality*; verifying the program by checking it against the specification.

Functional testing is a *systematic* style of testing; trying to select inputs that are *valuable*.

Functional testing is *not random* because its not an effective method of finding issues with the software.

2.1 Partition Testing

If we have knowledge that some inputs are more likely to fail than others, we can utilise this to produce more effective testing.

The entirety of possible inputs may be sparse in errors overall, but some regions may be more dense than others.

These partitions are created using the specification. These partitions will be tested, alongside their *boundaries*, which tends to be an area rife with errors.

2.2 Creating Test Cases

1. Break the specification into *Independently Testable Features*
2. Identify *representative* values to be used as input, *e.g. correct values and malformed values*
3. Create *test specifications* that are a combination of these inputs

3 Combinatorial Testing

Where we identify some *attributes* of an environment, and produce *combinations* of these attributes to be tested.

3.1 Category-Partition Testing

Similar to functional testing, makes use of manual identification of values.

Parameters (inputs) and *environmental elements* are defined for each ITF - these values are referred to as *categories*.

We generate a number of variables for each of these categories.

Finally we define constraints to remove combinations of variables that would not make sense (*property constraints*), or a combination that we only need to test once, *i.e. one variable renders the meaning of the other variables useless*, known as *single constraints*.

The constraints allow us to reduce the number of test cases that we have, as an unwieldy test suite does not provide much benefit.

3.2 Pairwise Combinatorial Testing

Category partitioning is quite an exhaustive process, which pairwise combinations try to avoid.

Most bugs in software are a result of an erroneous single input. It is far less common to find a bug that involves an interaction between three or more parameters.

Pairwise covers all combinations of size 2. This produces an inherently smaller test suite than taking the entire Cartesian product.

3.3 Catalog Based Testing

This method of testing is based on experience made by an organisation over time in deriving valuable test suites.