CONFORMIQ
Automated Test Design™

Testing Correctly: Your Guide to Test Design Techniques

Test design concerns making the decisions on (1) what to test and what not to test, (2) how to stimulate the system and with what data values, and (3) how the system should react and respond to the stimuli. It is a separate task from test execution and is done before executing the tests against the system. *Test design techniques*, on the other hand, are used to identify the test scenarios through which the test cases are created. Different testing goals will need to employ different test design algorithms. The more algorithms that are available to the test designer, the more effective and complete the testing can be.

There are numerous effective and efficient test design techniques for identifying those test scenarios which are applied in the industry. The purpose of this short paper is to give an overview of some of the most well-known techniques and how you can apply them using tools like Conformiq Designer and Conformiq Creator. These techniques can be divided into *static* and *dynamic techniques.* In this paper we will focus on dynamic techniques only. The dynamic techniques can be broadly divided into two categories called *black-box* and *white-box testing techniques* where the "color of the box" refers to the visibility that the test has to the internals of the SUT. (Also check out a post about Black-box vs. White-box Coverage at http://www.conformiq.com/2013/11/black-box-vs-white-box-coverage/). The fundamental problem with the techniques mentioned here is that they are typically explained with simple examples, while in the real world these methods need to be applied, for example in the case of equivalence class partitioning, in relation to every equivalence class of the reachable system states of the application that we are testing. For real world systems, carrying this out manually easily becomes prohibitively difficult. The beauty of the Conformiq test generation platform is that it can apply those techniques automatically throughout the application saving a lot of time and increasing the quality of testing at the same time.

Black-box testing techniques

Black-box approach assumes no internal details of the system and is based on judging the quality and correctness of the system using an external reference such as system specification.

Equivalence class partitioning

The idea of *equivalence class partitioning* is to divide all possible inputs to the system into "equivalence classes", i.e. sets of inputs that should produce "analogous" results and "work the same". By applying equivalence class partitioning technique we test only a few (maybe just one) conditions from each partition as it is assumed that all the conditions in a given partition will trigger the same behavior in the SUT. If that condition works properly, then all the other conditions within the partition are assumed to work properly as well, meaning that we can dramatically

narrow down the number of test scenarios that we need to execute. Similarly if that one condition happens to trigger an error in the SUT then it is assumed that all the other conditions will have the same effect. For more information about equivalence class partitioning, check out this post: http://www.conformiq.com/2013/12/equivalence-class-partitioning-and-boundary-value-analysis-as-black-box-test-design-heuristics/

Boundary value analysis

Boundary value analysis or BVA is a refinement of the equivalence class partitioning method, which is an applicable method when the equivalence classes involve numbers and there are *decision boundaries* between them. These decision boundaries are places where the behavior of the system changes. What makes boundary value analysis an interesting method is that it is widely recognized that values on the boundaries cause more errors in system. Therefore we always should be checking the boundaries because, if the system fails, it is likely to fail on these decision boundaries. For more information about BVA, check out this post: http://www.conformiq.com/2013/12/equivalence-class-partitioning-and-boundary-value-analysis-as-black-box-test-design-heuristics

Decision or cause-effect tables

Decision tables are used represent how combinations of inputs result in different actions taken and as such it is a testing technique focused more on business logic and rules. While above mentioned equivalence class partitioning and boundary value methods are extremely useful methods for narrowing down the number of inputs as they apply to a specific condition or input, decision tables are used define the system action given a combination of these inputs. It is a systematic way of describing complex business rules and they help the tester to understand and see the effect of combinations of different inputs. This way they help the tester to identify a subset of the combinations to test, while decision tables themselves do not provide an efficient way of selecting the combinations. Applying decision tables in testing may well result in an inefficient and poor test result.

Use case testing

Use case testing is a method that utilizes specification assets called use cases which are descriptions of particular uses of the system by the end user. Use cases can be used to design and capture test cases that exercise the whole end-to-end system. As use cases are described in terms of the end user and they describe what the user does with system, they do not describe the inputs that should be used to stimulate the system nor the exact expected response. However use cases may be very valuable as a test design process for establishing understanding about the type of test scenarios that should be included in the testing work.

Combinatorial testing (all-pairs, n-wise, etc.)

The basis for *combinatorial testing* is the *interaction principle* which states that most software failures are induced by single “factor” faults or by a combinatorial effect – that is *interaction* – of two factors, with progressively fewer failures induced by interactions between more factors. Therefore if a significant part of the faults in the system under test can be induced by a combination of N or fewer factors, then testing all N-way combinations of values provides a high rate of fault detection. This is also what has been empirically observed over the years. For more information about combinatorial testing, check out this post: http://www.conformiq.com/2014/10/combinatorial-test-data-generation-with-conformiq

Classification tree method

The *classification-tree method* is an approach to partition testing which uses a descriptive tree-like notation. The basic idea is to separate the inputs of the SUT into different classes that directly reflect the relevant test scenarios or classifications, which are followed by test case selection by combining classes of the different classifications. The selection of classes typically follows techniques such as equivalence class partitioning and boundary value analysis. All the classifications form the classification tree. In the next step, the test cases are composed by selecting one class from every classification of the classification tree. The selection of test cases can be done manually but there are tools that automate and optimize this process.

White-box testing techniques

White-box approach assumes that the tester has full visibility into the box being tested and how the system operates internally. White-box methods are based on the structure of the implementation. Therefore the applied techniques are also referred to as structure-based techniques.

Statement or line coverage

Statement coverage identifies statements or code lines executed by a test suite. It then calculates the percentage of executed statements versus those that are not executed.

Decision or branch coverage

Decision or *branch coverage* answers the question has each branch of a control flow construct been executed or, to put it in in different terms, has each possible outcome of a decision been executed. For example, with an *if* statement this means do we have a test that exercises the *then branch* and a test that exercises the *else branch*.

Condition or predicate coverage

Condition or *predicate coverage* answers the question has each Boolean sub-expression evaluated both true and false where a *condition* is a Boolean sub-expression that cannot be broken down into a simpler Boolean expression.

More advanced condition / decision coverage

Additionally, there are numerous more advanced and exhaustive ways to test and asses the completeness of testing in terms of more advanced condition and decision coverage heuristics such as *MC/DC (Modified Condition / Decision Coverage)* that requires that each condition should affect the decision outcome separately and *MCC (Multiple Condition Coverage)* that requires that all the combinations of conditions inside decisions are tested.

For more information about various condition and decision coverage options, check out this post: http://www.conformiq.com/2014/11/conditional-coverage-heuristics

Path testing

Path testing aims to cover *every control path* at least once which also includes all the iterations / loops that the code might have for zero, one, and multiple times. If the code includes unbounded iteration, the number of tests needed for testing the code is also unbounded which naturally is not practical. Therefore, in practice when using in path we fix some upper bound for these constructs for limiting the number of test cases to a more practical size.

Putting it all together

The basic idea is that a test design technique helps us to identify a “good” subset of test cases from infinitely many. Each of the techniques listed above focuses on certain specific characteristics of the system and has its own strengths and

weaknesses. Focusing only on one particular test design technique is therefore not really enough, as each technique is good at identifying only certain types of issues with the implementation while being hopeless in finding some others. It is really up to you the tester to select the best set of techniques for identifying the test cases that meet your testing requirements and goals.

When it comes to the classification of test design techniques in this paper, it is also worth noting that black-box and white-box test design are not mutually exclusive and even if I have categorized structural coverage aspects under white-box test design techniques, it does not mean that you could not also apply those techniques when doing black-box test design. What we need to remember is that black-box techniques assume no visibility into the tested system, meaning that we cannot apply those design techniques on white-box aspects. However, what we can do when we, for example, conduct model-based black-box test design, is apply those techniques classified under white-box techniques to the model and not to the actual implementation. By applying structure based test design techniques for black-box test design from models, we can improve the quality of the testing because we increase the coverage of the testing and increase the chances of finding defects in the implementation that other test design techniques might overlook.

In addition, it is important to note that a test suite designed using a black-box approach (even when also applying white-box test design techniques on the model) with a demonstrably good coverage of the functional specification (whether implicit or explicit) does not necessarily mean a high white-box coverage measurement. This is because there can be disproportionate amounts of code in the system that are either unused or dead code or that are related to unspecified functionality. It is known that good coverage of functional requirements does not always lead to high white-box coverage due to the aforementioned issues. On the other hand, good white-box coverage can be of only limited value if the tests are derived from the code itself. The key point of black-box testing is that the system is judged against an independent reference. For more information, refer to this article: http://www.conformiq.com/2013/11/black-box-vs-white-box-coverage

Testing techniques with Conformiq test generation platform

As I mentioned early in this paper, an additional and substantial problem with using the test design techniques mentioned previously is that they are typically explained with simple examples, but for real world systems carrying these out manually quickly becomes prohibitively difficult.

The Conformiq test generation platform implements the selected set of test design techniques *automatically* across the whole system specification (as encoded in a *model*). The tool is able to generate test suites that cover the aspects of the user selected set of techniques (and if not produce a report of what it failed to cover during the test generation) that at the same time are of reasonable size with full coverage and traceability information. As the tester, you are relieved from doing test design by hand, a time-consuming and error-prone process that is also very complicated for larger systems.

The difference between *automatically* and *automated* (usually meaning user intervention is needed to direct or select how the test design is done) is a unique value proposition of Conformiq's test generation platform that separates the platform from ALL other model-based testing approaches and tools available in the market.

The author of this paper, Kimmo Nupponen, has been developing automated test design software for over ten years. He is the Chief Scientist at Conformiq.

www.conformiq.com

4030 Moorpark Ave
San Jose, CA 95117
USA

Tel: +1 408 898 2140
Fax: +1 408 725 8405

Westendintie 1
02160 Espoo
FINLAND

Tel: +358 10 286 6300
Fax: +358 10 286 6309

Stureplan 4C
SE-11435 Stockholm
SWEDEN

Tel: +46 852 500 222
Fax: +358 10 286 6309

Maximilianstrasse 35
80539 Munich
GERMANY

Tel: +49 89 89 659 275
Fax: +358 10 286 6309

29 M.G. Road Ste 504
Bangalore 560 001
INDIA

Tel: +91 80 4155 0994

v0715