Model-based testing in practice

Summary

A technique for generating suite of test cases from the requirements is known as Model-based testing. A broad coverage on input domain with less number of test cases can be achieved by this testing.

Model bases testing approach enables a better option to test the product with in the short time. Automated execution of tests does not address costly problems and uncertain coverage of input domain.

In Model based test, Model is an essential specification of inputs to the software and can be developed early in the life cycle and a regression test suit can be generated. The model based testing will help testers in reducing time on developing test cases and is also cost effective. This technique does not require tester to have development or system engineering knowledge.

The notion used for model, test-generation algorithm and tools for supporting infrastructures for test are technologies that Model based testing depends on.

model based testing can be applied on different levels.

Model based testing can be applied to very small units, collections of units, or entire systems during development and maintenance life cycle. At lowest level, model based testing is exercised on a single module and a small set of tests can be developed rapidly and used during unit test activities.

In the intermediate level, this technique is applied on a single operation and allows computation of expected outputs. In the complex systems the step-oriented tests can be chained to generate test suits.

Model Notations

The model notation must tester readable and must also convey a large problem and must also be understood by test generation tool. A data model specifies all possible values (along with constraints like accepting null values) for a parameter and a test-generation data specifies valid or invalid for the parameter in the test along with seeds (combination of data).In order to provide a meaningful notation with above mentioned characteristics, the author has used simple specification notation called AETGSpec. AETGSpec is used to capture functional model of data and can create effective test cases.

The model based testing was implemented on four different projects such as
Arithmetic and Table Operators, Message parsing and building, A rule-based system and A user interface

The central idea behind AETG is the application of experimental designs to test generation. Each separate element of a test input parameter is treated as a factor, with the different values for each parameter treated as a level. For example, a set of inputs that has 10 parameters with 2 possible values each would use a design appropriate for 10 factors at 2 levels each. The design will ensure that every value of every parameter is tested at least once with every other level of every other factor, which is called pairwise coverage of the input domain. The Pairwise coverage provides a huge reduction in the number of test cases when compared with testing all combinations. The AETG approach uses avoids in order to remove the combinations that are not valid. It also allows pairwise combinations which results in greater coverage for the input domains.

The major strength of the AETG approach is that the test suit generation is on par with the requirement specification. But the tester need to have minimum development knowledge. This is applicable to a system whose behavior can be captured by a data model.

Concepts Learned
The generation of test suite from the requirements in the model bases testing would be helpful for testing more cases with in the short time.
My own experience

During my work, we used some automation code to test the application on IOS device. Testers used to have hand written test cases. To reduce the complexity, the model based testing captures test suite from the requirement with in the short time.

Comments
The concept is old when compared to contemporary techniques.

Critiques:
Questions
Could please explain more recent techniques that generates test suite from the requirements, Any tools support would be helpful.

Failure Triggering and Fault Interactions

Summary:

FTFI (Failure Triggering and Fault Interactions) means the interactions between any two variables which result in fault or failure in the system. How can FTFI would be set for any software system was discussed.

the history of the failure data, the parameters which caused and apply testing to those parameters in the present software will be done by software testers.

why should we consider FTFI is the time given for a tester is very limited and there will not be enough resources to cover the whole test cases.we consider the most fault prone areas and parameters.

In the research on empirical data, the author has collected 329 errors from a large integration testing from NASA Goddard Space Flight Center, which produces a large amount of data which are store in various subsystems. they were all corrected at the beginning of and then sent to the data bases,If any faults were found. All the faults are categorized in database by submission date, severity, priority of their fix location, status the activity they were performing and also with an additional field explaining how the fault was fixed. By looking at the data provided the author was saying that the result finding must be easier in the development projects than that of the deployed projects. But that is not the case what is happening in the NASA. All the fault interactions were difficult to find in the development projects rather than the fielded or deployed. The fault tolerance for the fielded projects like medical devices or the famous Ariane Projects were two(2). But they have caused any faults. In case of the Ariane 5 the failure was due to error in numerical conversion.
So the author suggests that it is not safe to assume that the failures always due to the rare combinations of conditions.

In testing the implications, the fault interactions the number of test cases to be written can be substantially reduced with the empirical data. For example the author has mentioned an example which consists of 20 inputs. So, there are 20 parameters(v). It consists of 4 tuples and 6 tuples in each test cases. So there are total 10 tuples(n). So the number of test cases required are v^n. But this solution is for the discrete values. But in many real time values where the parameters are in random continues values, the parameters are catogerized into classes, where system operation one class might be correct or incorrect for another class. So in order to test the number of continuous data we need to know how classes we need to test. This method is called pseudo exhaustive testingSince most of the software development, testing will be the final phase and there will not be much to take care of, managers generally opt for the pseudo exhaustive testing by dividing all the variables into classes and then the test cases are generated and the testing is performed.

Reflecting upon my experience:

Deriving test cases from dividing all the possible values of the variables into classes generates more test cases. We used maintain log of errors reporting with previous version of software. The possible bugs reported earlier considered test cases for the current system under test.

Comments

Reduction of time in writing test cases with fault interaction mechanism helpful for improving the performance of testing. Testers get times cover more test cases. But there might be some faults which might not be covered.

Critiques on the results presented in the data:

Questions