Content Table

What is it? Why is it important?

What is it?

Model-based testing (MBT) is a software testing approach that necessitates the creation of a second, lightweight version of a software build known as a model by developers. In this testing, test cases are automatically generated from models. The models are created and maintained within the code and are part of the software development process.

To get a clear idea of how a model is created, we can take a closer look at an example of an actual model for simple login with valid credentials:

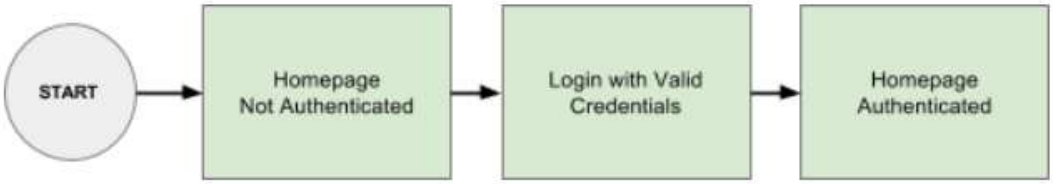

Fig: Model for simple login with valid credentials

Why is it important:

Importance of Model-Based Testing:

- It addresses the challenges that developers and testers face when attempting to create better and faster software.
- In software development, the goal is to enable rigorous, in-sprint testing that allows stakeholders—from testers to business analysts—to stay in alignment and remain flexible, which can be achieved by employing the model-based technique.
- Model-based testing approach enables teammates to work in parallel and achieve agile testing, even as software requirements change.
- Helps to create better software quality by getting the team thinking about the models.

- It allows for flexibility to generate many tests using different algorithms (smoke, regression, integration, end-to-end, and targeted testing for new/modified features).

Different Types of Model-based Testing:

There are two types of Model based testing framework.

1. Offline / a priori: Generation of Test Suites before executing it. A test suite is nothing but a collection of test cases.
2. Online / on-the-fly: Generation of Test Suites during test execution

Here are different ways of creating models:

1. Data Flow Diagram – graphically representing the functions, or processes
2. Dependency Graphs – describing processes and/or relation between two functions
3. Decision Tables – specifying which actions to perform depending on given conditions
4. State transition machines – mathematical models represented in the form of tables or charts
5. Control Flow – individual statements, instructionsor function calls of an imperative program

Relation between Model-Based testing and SDLC stages:

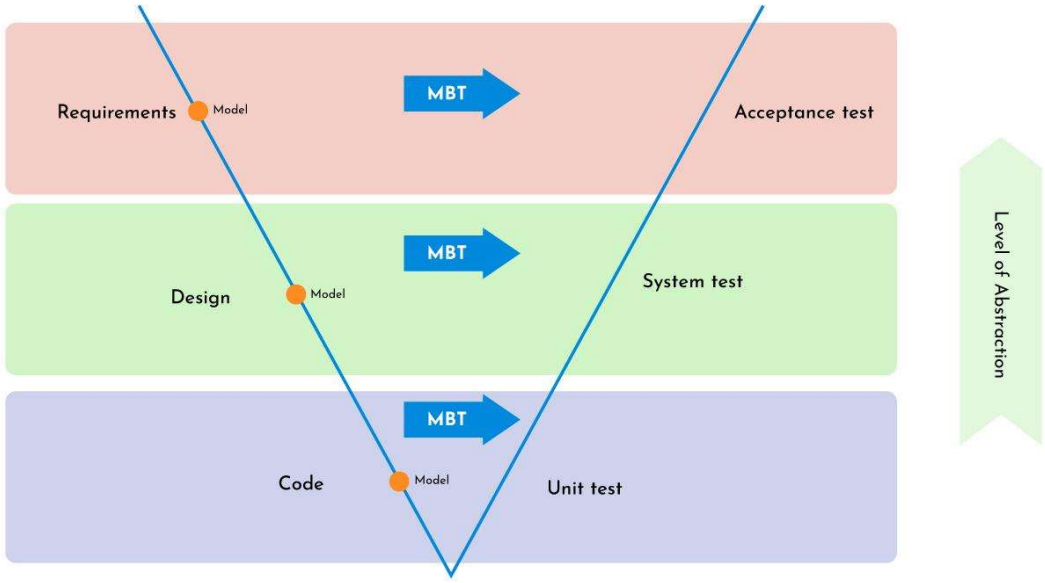

Figure: MBT in action in SDLC V-model

The high-level representation of what a software should do is the model used for test generation. Such models can be created at different levels of abstraction e.g. requirement gathering, designing, coding, etc phases of SDLC. The model based testing tools then automatically generate and execute tests that will pass if and only if the system under test is conforming to the model. For example, a model of a requirement can be made and the acceptance testing can be automated. At the code level, a model of a method can be made and unit testing can be automated. Any level in between is also possible. A great feature of Model Based Testing is the possibility to model only parts of a system e.g. a small set of requirements, and already obtain the benefits of automated testing. Model based testing consists in creating a precise representation of what is expected. The actual checking whether a

system under test actually fulfills the expectations is automated. To conclude, Yes, there is a relationship between SDLC stages and Model-Based testing. On the V-model this automates work from left to right.

How do you perform model-based testing?

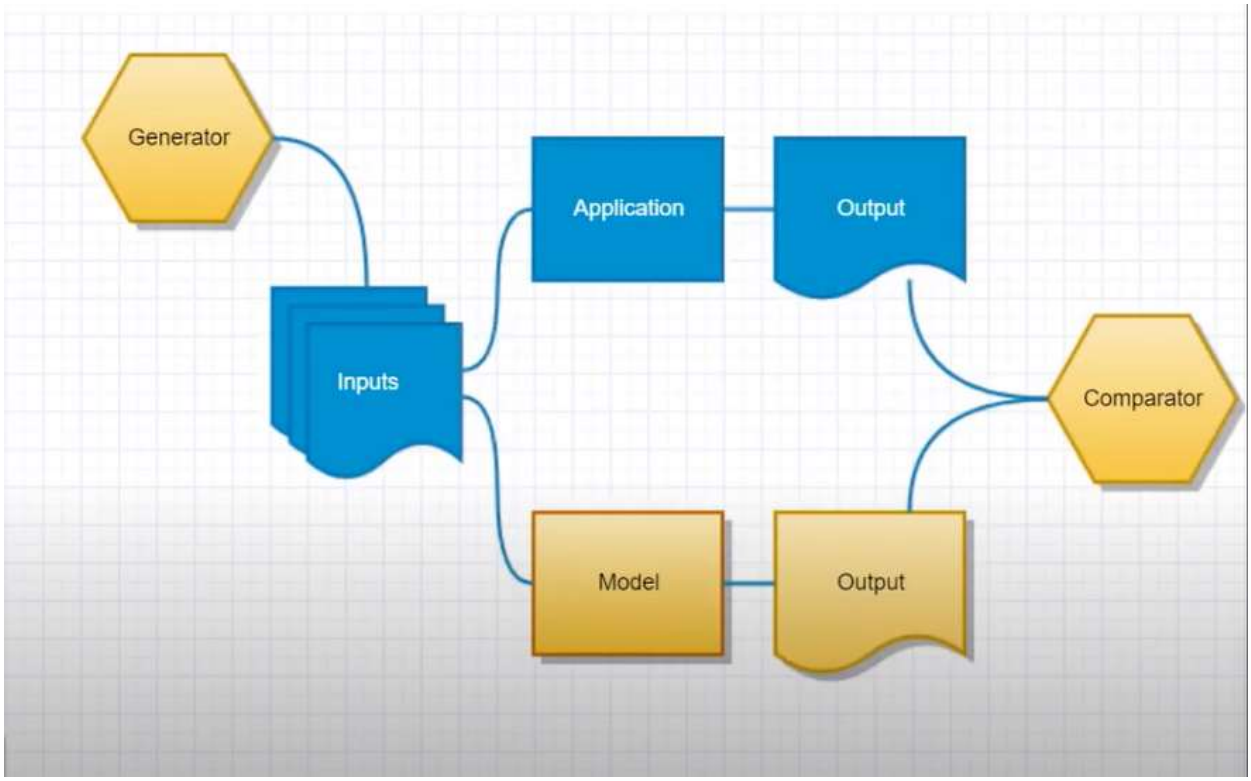

Figure: Performing Model Based Testing

Generator:

It generates different combinations of data for the system that will be tested for analyzing the performance of the system.

Application:

It is basically the original application that will be tested against the test model.

Model:

The model is basically the test model. The model is created to get output from incoming inputs. And its output is compared with the original application to find bugs.

Comparator:

The output came from the original application and the test model is compared and later based on this comparison necessary changes are being made.

Briefly explain if there are any automated tools to perform the given testing:

For performing model based testing, there are several automated tools in both open-source and commercial categories. The following are some of the amazing MBT tools. First, we are going to explain some tools of open-source categories. Then, we will proceed to commercial categories.

Open-Source tools for MBT

fMBT:

fMBT (free Model-Based Testing) is a set of tools for completely automatic test generation and execution as well as a collection of utilities and libraries that provide a high degree of test automation. It generates test cases from LTS models. The LTS models are specified in the AAL pre/postcondition language, employing various heuristics such as random, weighted random, lookahead etc. . It also uses python libraries for multi-platform GUI testing. It provides a tool for editing, debugging, executing, and recording GUI test scripts, as well

as a tool for visually examining test models and produced tests. It is quite fast to uncover test pathways that human test designers will never test. When compared to typical test automation, this elevates test coverage while decreasing test maintenance effort.

Modbat:

A renowned model-based testing tool based on annotated (extended) finite-state machines. It focuses on testing software's application programming interface (API). Modbat's model is compatible with any Java-based application or program. The user creates and compiles a model, which Modbat explores and executes against the system under test. The structure of the model is that of a non-deterministic finite-state machine. Out of all potential transitions, a transition is chosen to be tested. The annotations on the transition describe which code to run.

Tcases:

Tcases is a combinatorial testing tool where the inputs of the system could be provided in an XML file (containing conditions, failure values, don't cares, and so on). Tcases can produce either n-wise or randomized test suites.

GraphWalker:

A model based testing tool where tests are generated from Finite State Machines. Different search algorithms such as, A* or random, with a limit for various coverage criteria (state, edge, requirement) has been used here.
GraphWalker provides an editor called the "studio" where models can be created and edited. In the tool, the models can be verified by running test path generations so the user can verify the correctness of the models. Formerly it is known as mbt.

Commercial tools for MBT:

Tosca:

Tosca is a comprehensive test automation framework that produces test data based on many criteria (risks, pairwise etc.). If the test data is provided, the structure of the test data objects may be extracted automatically from the implementation.

BPM-X:

A model based testing tool BPM-X, which generates test cases from business process models based on different criteria (statement, branch, path, condition). It can import models from many modeling programs and output test cases to Excel, HP Quality Center, and other formats.

MBTsuite:

A model based testing tool MBTsuite, which can generate test cases from UML models based on various coverage criteria (path, edge...) or randomly. It can import models from several model editors and it is compatible with a variety of test management and execution tools.

TestOptimal:

TestOptimal supports FSM and E-FSM modeling with several test case generation algorithms. It includes plugins for online testing web applications, windows programs, databases, and web services, among other things. TestOptimal also supports data-driven testing and pair-wise algorithms directly within the model, as well as load testing utilizing the same models.

Are there any similarities of the given testing with others? Briefly explain.

Model Based Testing tools are capable of generating test cases that will only pass if the system follows the rules of the model. Models for different levels of abstraction can be created which can automate different types of testing.For example -

- If a model of requirements is made, it can automate acceptance testing.
- If a model of method at code level is made, it can automate unit testing.
- Regression testing checks if the system behaves the same after a change, similarly in MBT automated testing can be done after bringing change to a model
- MBT provides the flexibility to generate many kinds of tests through different algorithms(smoke, regression, integration, end-to-end, and targeted testing for new/modified features).

Strength of Model-Based Testing:

- Automatic test case generation:

  Model-based testing automates the creation of the traceability matrix and the precise design of test cases. The test designer builds an abstract representation of the system under test rather than creating hundreds of test cases. The model is then used to produce a set of test cases using the model based testing tool.

  Thus, there is no hassle of manually writing the test cases. Rather a bunch of test cases are created automatically based on the model.

- Early and increased fault detection:

  Primary aim of model-based testing is to expose the system's fault. Fault detection depends on the model and criteria of test selection. Because model-based testing automatically generates the test cases, there is a higher chance that it encompasses those rare cases that would have been missed if written manually.

  So, the fault detection capability of model-based testing is more than that of any manual approach.

- Reduced testing time and cost:

  In model-based testing failures are reported in a consistent way. Some model-based testing tools are capable of finding shortest possible test cases that cause failure.

  This causes, model-based testing approach to take much less time and effort than that of manually designing test suites.

- Improved test quality:

  Manual testing puts a question mark on the test quality. Because the quality of test cases depends on the test engineer. Moreover, the test design process is usually not reproducible.

  The quality of model based testing is dependent on the model.

- Model Coverage:

  Coverage evaluates the generated test cases and indicates the test progress. In model-based testing model coverage is used. Model coverage expresses the thoroughness and effectiveness of testing effort.

- Requirements defect detection:

  The first step in model-based testing is to create an abstract model of the system. This leads to detecting requirement issues at the very beginning.

- Traceability:

  Traceability helps in explaining the test cases and providing justification for why a test case is needed. It is a relation between the model and the test suite. In model-based testing we can optimize the test execution by executing the subset of the test suite that are affected by model modification.

- Updating test suite:

  In manual testing updating the test suite is a huge burden. Whereas, in model-based testing all we have to do is update the model and the updated test suite is generated automatically. It requires much less time and effort than that of manual testing.

- Testing quality attributes:

  Model-based testing is an effective approach to test different quality attributes of a system like security, performance, reliability, and correctness of a given system.

- Executing different tests on different machine:

  Because model-based testing is dependent on models, we can easily transfer a model to a different device and can execute testing on that machine. This way, we can concurrently test numerous models at a time.

Limitations of model based testing

- Detecting difference between model and implementation:

  The model-based testing approach cannot guarantee to find the differences between the model and implementation. This is a common issue for all other testing approaches as well.

- Outdated requirements:

  Because of the continuously evolving nature of software projects, requirements often become outdated. Implementing those requirements will lead to erroneous models and ultimately erroneous test suites.

- Useless metrics:

  Number of test cases generated is no longer the metric to measure the test progress. Because a huge number of test cases can be generated within a moment. Instead the model coverage is used to measure the metrics.

- Inappropriate use of model-based testing:

  Some parts of a system are allowed to be tested manually because of its complexity to convert that segment into a model. However, it requires a level of expertise to appropriately decide which part to be tested manually and which are not.

- Tester skills:

  The skill set required for model-based testing is different from that of other testing approaches. The testers have to appropriately design the model. Because the quality of the test suite is dependent on the model.

- State space explosion:

  Models of any non-trivial software system can grow beyond the manageable level which ultimately results in state space explosion.

- Time to analyze failed test:

  Failure might be caused by the system, adapter code, erroneous model. After any failure, the test engineers have to decide the cause of failure.

Because the automatic test cases are often complex, it might often be difficult to trace the root cause of failure.

Our opinion regarding future works to make model based testing more efficient:

- Finding appropriate selection criteria:
  Model based testing is not widely used in real life applications and the most appropriate reason behind this is due to state explosion problems. Often models of these systems grow beyond the manageable level. With a view to address this issue, we can study an approach to select a weaker criteria that will keep the state space limited to our manageable level.

- Incorporating non-functional testing:
  Model based testing does not provide support for non-functional testings like performance testing, stress testing, load testing. Future studies are required to perform non-functional testing using model based testing.

- Automate model generation from implementation:
  Often developing models from implementation is hard and an extra level of work along with actual implementation. Moreover, it requires a level of expertise and skills. Any change to the system will impose a change to the model as well. Because software development is an agile process, changes are obvious. Repeated change to the model will also take a huge time and effort. In this circumstance, if any automation approach is developed to generate models from the given implementation, it would eradicate the existing issues with model based testing.

- Efficient Utilization of MBT:

  Generating test cases can be automated by using Model-Based Testing for different testing types i.e. smoke testing, unit testing, etc. However, to generate test cases it is necessary to be efficient and not overdo the process which might lead to unnecessary delays in the testing process. So, the generation of efficient test cases and also making the process efficient is a field of study that can improve the testing process in this domain greatly.

References

1. https://www.youtube.com/watch?v=mGrFV8bHPPw
2. https://www.techtarget.com/searchsoftwarequality/definition/model-based-testing
3. https://saucelabs.com/blog/the-challenges-and-benefits-of-model-based-testing
4. https://www.broadcom.com/info/continuous-testing/model-based-testing
5. model-based-testing-and-model-driven-engineering
6. 9 Great Tools to work with Model-based Testing (MBT) (automated-360.com)
7. Model-based testing overview, tools and projects (bme.hu)
8. Model Based Testing Tools |Professionalqa.com
9. https://www.ict.eu/en/projects/what-difference-between-model-based-testing-and-model-driven-engineering
10. http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html
11. https://www.guru99.com/model-based-testing-tutorial.html
12. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.211.8934&rep=rep1&type=pdf
13. http://www.diva-portal.org/smash/get/diva2:831658/FULLTEXT01.pdf
14. http://www.diva-portal.org/smash/get/diva2:831658/FULLTEXT01.pdf.