MODEL-DRIVEN TEST DESIGN

The methodology is also known as model-driven software development (MDSD), model-driven engineering (MDE) and model-driven architecture (MDA). The MDD approach focuses on the construction of a software model. The model is a diagram that specifies how the software system should work before the code is generated.

What is Model-Driven Test Design?

MDTD is built on the idea that designers will become more effective and efficient if they can raise the level of abstraction. This approach breaks down the testing into a series of small tasks that simplify test generation. Then test designers isolate their tasks and work at a higher level of abstraction by using mathematical engineering structures to design test values independently of the details of the software or design artifacts, test automation, and Test Execution.

Different phases in MDTD

MDTD can be done in 4 different phases. Each type of activity requires different skills, background knowledge, education, and training.vIt is better to use different sets of people depend on the situation.

1. **Test Design** — This can be done in either Criteria-Based where Design test values satisfy coverage criteria or other engineering goals or in Human-Based where Design test values based on domain knowledge of the program and human knowledge of testing which is comparatively harder. This the most technical part of the MDTD process better to use experienced developers in this phase.

2. **Test Automation** — This involves embedding test values to scripts. Test cases are defined based on the test requirements. Test values are chosen such that we can cover a larger part of the application with fewer test cases. We don’t need that much domain knowledge in this phase, however, we need to use technically skilled people.

3. **Test Execution** — The test engineer will run tests and records the results in this activity. Unlike the previous activities, test execution not required a high skill set such as technical knowledge, logical thinking, or domain knowledge. Since we consider this phase comparatively low risk, we can assign junior intern engineers to execute the process. But we should focus on monitoring, log collecting activities based on automation tools.

4. **Test Evaluation** — The process of evaluating the results and reporting to developers. This phase is comparatively harder and we expected to have knowledge in the domain, testing, and User interfaces, and psychology

2.5.2 TEST AUTOMATION

The final result of test design is input values for the software. Test automation is the process of embedding test values into executable scripts. Note that automated tool support for test design is not considered to be test automation. This is necessary for efficient and frequent execution of tests. The programming difficulty varies greatly by the software under test (SUT). Some tests can be automated with basic programming skills, whereas if the software has low controllability or observability (for example, with embedded, real-time, or web software), test automation will require more knowledge and problem-solving skills. The test automator will need to add additional software to access the hardware, simulate conditions, or otherwise control the environment. However, many domain experts using human-based testing do not have programming skills. And many criteria-based test design experts find test automation boring. If a test manager asks a domain expert to automate tests, the expert is likely to resist and do poorly; if a test manager asks a criteria-based test designer to automate tests, the designer is quite likely to go looking for a development job.

2.5.3 TEST EXECUTION

Test execution is the process of running tests on the software and recording the results. This requires basic computer skills and can often be assigned to interns or employees with little technical background. If all tests are automated, this is trivial. However, few organizations have managed to achieve 100% test automation. If tests must be run by hand, this becomes the most time-consuming testing task. Hand-executed tests require the tester to be meticulous with bookkeeping. Asking a good test designer to hand execute tests not only wastes a valuable (and

possibly highly paid) resource, the test designer will view it as a very tedious job and will soon look for other work.

2.5.4 TEST EVALUATION

Test evaluation is the process of evaluating the results of testing and reporting to developers. This is much harder than it may seem, especially reporting the results to developers. Evaluating the results of tests requires knowledge of the domain, testing, user interfaces, and psychology. The knowledge required is very much the same as for human-based test designers. If tests are well-automated, then most test evaluation can (and should) be embedded in the test scripts. However, when automation is incomplete or when correct output cannot neatly be encoded in assertions, this task gets more complicated. Typical CS or software engineering majors will not enjoy this job, but to the right person, this is intellectually stimulating, rewarding, and challenging.

2.5.5 TEST PERSONNEL AND ABSTRACTION

These four tasks focus on designing, implementing and running the tests. Of course, they do not cover all aspects of testing. This categorization omits important tasks like test management, maintenance, and documentation, among others. We focus on these because they are essential to developing test values. A challenge to using criteria-based test design is the amount and type of knowledge needed. Many organizations have a shortage of highly technical test engineers. Few universities teach test criteria to undergraduates and many graduate classes focus on theory, supporting research rather than practical application. However, the good news is that with a well-planned division of labor, a single criteria-based test designer can support a fairly large number of test automators, executors and evaluators. The model-driven test design process explicitly supports this division of labor. This process is illustrated in Figure 2.4, which shows test design activities above the line and other test activities below.

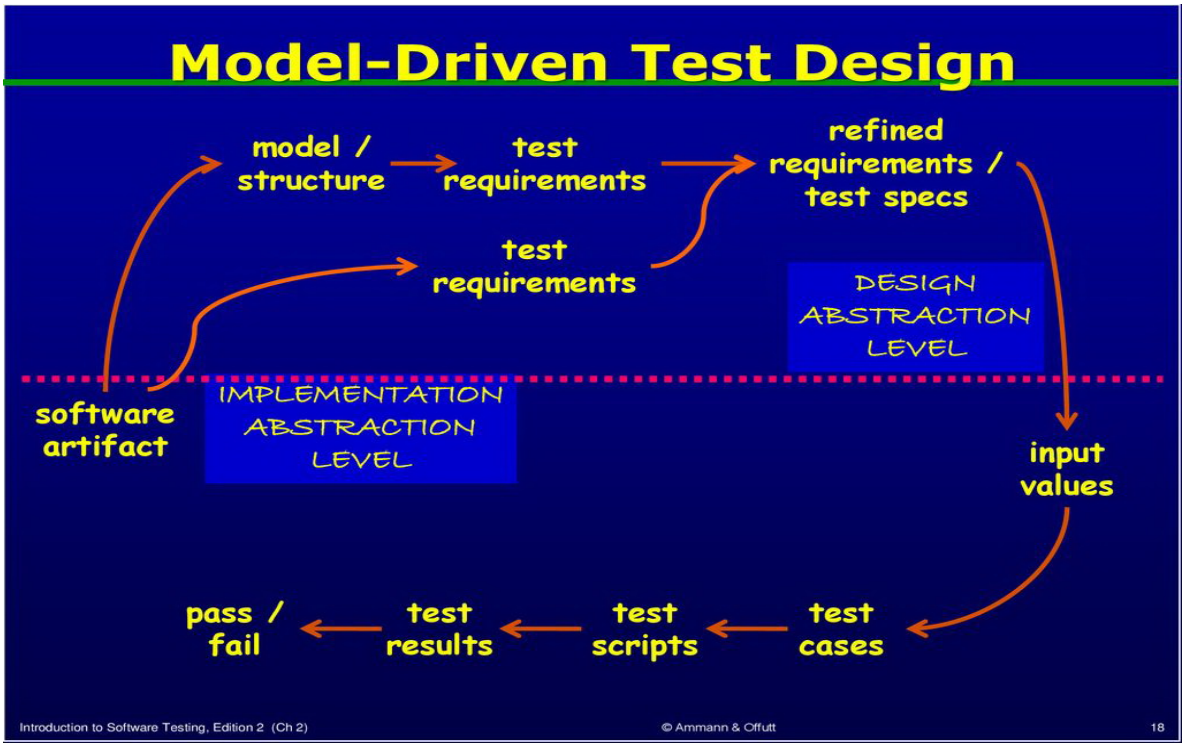


Figure 2.4. Model-driven test design.

The MDTD lets test designers "raise their level of abstraction " so that a small subset of testers can do the mathematical aspects of designing and developing tests. This is analogous to construction design, where one engineer creates a design that is followed by many carpenters, plumbers, and electricians. The traditional testers and programmers can then do their parts: finding values, automating the tests, running tests, and evaluating them. This supports the truism that "testers ain't mathematicians." The starting point in Figure 2.4 is a software artifact. This could be program source, a UML diagram, natural language requirements, or even a user manual. A criteria-based test designer uses that artifact to create an abstract model of the software in the form of an input domain, a graph, logic expressions, or a syntax description. Then a coverage criterion is applied to create test requirements. A human-based test designer uses the artifact to consider likely problems in the software, then creates requirements to test for those problems. These requirements are sometimes refined into a more specific form, called the test specification. For example, if edge coverage is being used, a test requirement specifies which edge in a graph

must be covered. A refined test specification would be a complete path through the graph. Once the test requirements are refined, input values that satisfy the requirements must be defined. This brings the process down from the design abstraction level to the implementation abstraction level. These are analogous to the abstract and concrete tests in the model-based testing literature. The input values are augmented with other values needed to run the tests (including values to reach the point in the software being tested, to display output, and to terminate the program). The test cases are then automated into test scripts (when feasible and practical), run on the software to produce results, and results are evaluated. It is important that results from automation and execution be used to feed back into test design, resulting in additional or modified tests. This process has two major benefits. First, it provides a clean separation of tasks between test design, automation, execution and evaluation. Second, raising our abstraction level makes test design much easier. Instead of designing tests for a messy implementation or complicated design model, we design at an elegant mathematical level of abstraction. This is exactly how algebra and calculus has been used in traditional engineering for decades.

TEST CASE ORGANIZATION AND TRACKING

A **Test Case** is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

Test Scenario Vs Test Case

Test scenarios are rather vague and cover a wide range of possibilities. Testing is all about being very specific.

For a Test Scenario: Check Login Functionality there many possible test cases are:

- Test Case 1: Check results on entering valid User Id & Password
- Test Case 2: Check results on entering Invalid User ID & Password
- Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

This is nothing but a Test Case.

The format of Standard Test Cases

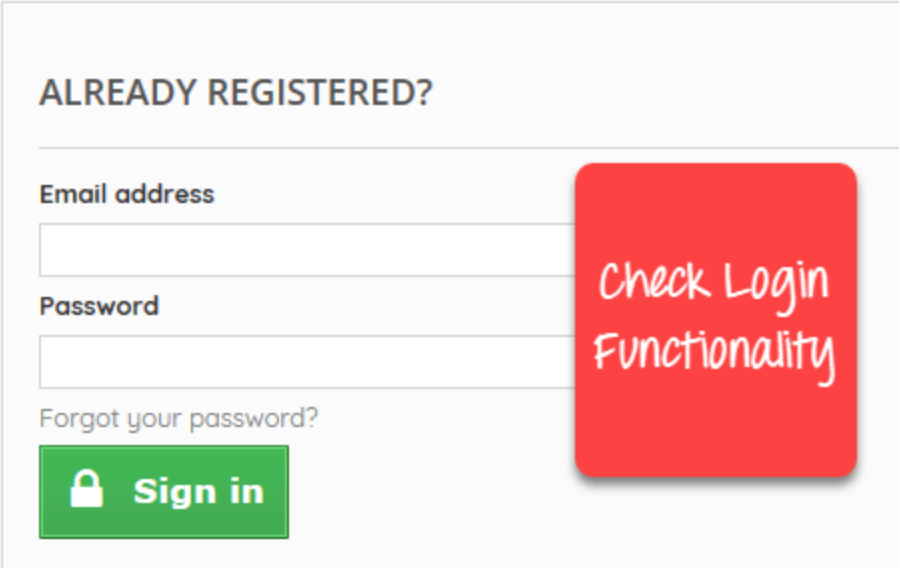
Below is a format of a standard login Test cases example.

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Customer Login with valid Data	1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit	Userid = guru99 Password = pass99	User should Login into an application	As Expected	Pass
TU02	Check Customer Login with invalid Data	1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit	Userid = guru99 Password = glass99	User should not Login into an application	As Expected	Pass

This entire table may be created in Word, Excel or any other Test management tool. That's all to Test Case Design

How to Write Test Cases in Manual Testing

Let's create a Test Case for the scenario: Check Login Functionality



Step 1) A simple test case to explain the scenario would be

Test Case #	Test Case Description
1	Check response when valid email and password is entered

Step 2) Test the Data.
In order to execute the test case, you would need Test Data. Adding it below

Test Case #	Test Case Description	Test Data
1	Check response when valid email and password is entered	Email: mary99@email.com Password: lNf9^Oti7^2h

Identifying test data can be time-consuming and may sometimes require creating test data afresh.
The reason it needs to be documented.

Step 3) Perform actions.
In order to execute a test case, a tester needs to perform a specific set of actions on the AUT.
This is documented as below:

Test Case #	Test Case Description	Test Steps	Test Data
1	Check response when valid email and password is entered	1) Enter Email Address 2) Enter Password 3) Click Sign in	Email: guru99@email.com Password: lNf9^Oti7^2h

Many times the Test Steps are not simple as above, hence they need documentation. Also, the author of the test case may leave the organization or go on a vacation or is sick and off duty or is very busy with other critical tasks. A recently hire may be asked to execute the test case. Documented steps will help him and also facilitate reviews by other stakeholders.

Step 4) Check behavior of the AUT.
The goal of test cases in software testing is to check behavior of the AUT for an expected result.

This needs to be documented as below

Test Case #	Test Case Description	Test Data	Expected Result
1	Check response when valid email and password is entered	Email: guru99@email.com Password: lNf9^Oti7^2h	Login should be successful

During test execution time, the tester will check expected results against actual results and assign a pass or fail status

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail
1	Check response when valid email and password is entered	Email: guru@email.com Password: lNf9^Oti7^2h	Login should be successful	Login was successful	Pass

Step 5) That apart your test case -may have a field like,

Pre – Condition which specifies things that must be in place before the test can run. For our test case, a pre-condition would be to have a browser installed to have access to the site under test. A test case may also include Post – Conditions which specifies anything that applies after the test case completes. For our test case, a post-condition would be time & date of login is stored in the database

BEST PRACTICE FOR WRITING GOOD TEST CASE.

Test Case Best Practice

1. Test Cases need to be simple and transparent:

Create test cases that are as simple as possible. They must be clear and concise as the author of the test case may not execute them.

Use assertive language like go to the home page, enter data, click on this and so on. This makes the understanding the test steps easy and tests execution faster.

2. Create Test Case with End User in Mind

The ultimate goal of any software project is to create test cases that meet customer requirements and is easy to use and operate. A tester must create test cases keeping in mind the end user perspective

3. Avoid test case repetition.

Do not repeat test cases. If a test case is needed for executing some other test case, call the test case by its test case id in the pre-condition column

4. Do not Assume

Do not assume functionality and features of your software application while preparing test case. Stick to the Specification Documents.

5. Ensure 100% Coverage

Make sure you write test cases to check all software requirements mentioned in the specification document. Use Traceability Matrix to ensure no functions/conditions is left untested.

6. Test Cases must be identifiable.

Name the test case id such that they are identified easily while tracking defects or identifying a software requirement at a later stage.

7. Implement Testing Techniques

It's not possible to check every possible condition in your software application. Software Testing techniques help you select a few test cases with the maximum possibility of finding a defect.

- **Boundary Value Analysis (BVA):** As the name suggests it's the technique that defines the testing of boundaries for a specified range of values.
- **Equivalence Partition (EP):** This technique partitions the range into equal parts/groups that tend to have the same behavior.
- **State Transition Technique**: This method is used when software behavior changes from one state to another following particular action.
- **Error Guessing Technique:** This is guessing/anticipating the error that may arise while doing manual testing. This is not a formal method and takes advantages of a tester's experience with the application

8. Self-cleaning

The test case you create must return the Test Environment to the pre-test state and should not render the test environment unusable. This is especially true for configuration testing.

9. Repeatable and self-standing

The test case should generate the same results every time no matter who tests it

10. Peer Review.

After creating test cases, get them reviewed by your colleagues. Your peers can uncover defects in your test case design, which you may easily miss.

While drafting a test case to include the following information

- The description of what requirement is being tested
- The explanation of how the system will be tested
- The test setup like a version of an application under test, software, data files, operating system, hardware, security access, physical or logical date, time of day, prerequisites such as other tests and any other setup information pertinent to the requirements being tested
- Inputs and outputs or actions and expected results
- Any proofs or attachments
- Use active case language
- Test Case should not be more than 15 steps
- An automated test script is commented with inputs, purpose and expected results

- The setup offers an alternative to pre-requisite tests
- With other tests, it should be an incorrect business scenario order

TEST CASE MANAGEMENT TOOLS

Test management tools are the automation tools that help to manage and maintain the Test Cases. Main Features of a test case management tool are

1. **For documenting Test Cases:** With tools, you can expedite Test Case creation with use of templates
2. **Execute the Test Case and Record the results:** Test Case can be executed through the tools and results obtained can be easily recorded.
3. **Automate the Defect Tracking:** Failed tests are automatically linked to the bug tracker, which in turn can be assigned to the developers and can be tracked by email notifications.

- **Traceability:** Requirements, Test cases, Execution of Test cases are all interlinked through the tools, and each case can be traced to each other to check test coverage.
  Protecting Test Cases: Test cases should be reusable and should be protected from being lost or corrupted due to poor version control. Test Case Management Tools offer features like
- Naming and numbering conventions
- Versioning
- Read-only storage
- Controlled access
- Off-site backup