Menditect Test plan Template

Incorporating MTA testing in an agile development

Version August 2022

Index

Contents

Introduction

This document helps Menditect customers who have limited time available for test management to structure their test efforts. It is also meant for customers who are just getting started working with MTA to take away any impediments to get to know the product better.

ISTQB

A test plan according to ISTQB is a document describing the

- scope,
- approach,
- resources and
- schedule

of intended test activities.

It identifies amongst others

- test items,
- the features to be tested,
- the testing tasks,
- who will do each task,
- degree of tester independence,
- the test environment,
- the test design techniques and
- entry and exit criteria to be used, and the rationale for their choice, and
- any risks requiring contingency planning.

It is a record of the test planning process.

IEEE 829

According to the IEEE 829 standard, making a test plan consists of these steps :

1. Analyze the product (what are you going to test?)
2. Design the Test Strategy (how are you going to test?)
3. Define the Test Objectives (why are you testing?)
4. Define Test Criteria (what are the expected results?)
5. Resource Planning (who will be testing?)
6. Plan Test Environment (where will you be testing?)
7. Schedule & Estimation (when will you be testing?)
8. Determine Test Deliverables (what is the output?)

Test plan template

Product outline

A brief summary of the Mendix app being tested.

What is the raison d'être? Outline all the app's functions at a high level, starting with the primary functions. What is the impact on the organization if the function fails?

Test plan Objectives

Why do we need a test plan when automating tests with MTA?

Basically you need a test plan to know where to start. Objectives could be any of these;

- tests are prioritized based on risk and impact,
- Mendix code is ready to be tested with MTA,
- the work for test scripting and analysis is assigned,
- gained insight in function coverage,
- functions in scope are covered by at least a single test script,
- etc.

Test objectives

What do we hope to accomplish by testing with MTA? Make it SMART.

Generally speaking you would expect the quality of your product to go up and therefore have to spend less time on fixing bugs in production. Also automated testing should take less time than manual testing over a period of time. Below are a few examples.

After one year of using MTA, in a single sprint, we expect to have:

- found 2 fewer high prio bugs in production in the 1st week after a new release,
- 50% less time spent on regression testing,
- 30% less time spent on manual testing,
- 20% less time spent on fixing bugs,
- 10% less time spent on support,
- etc.

Tasks

What do we have to do to make the testing happen?

For example:

- set up environments (configuration management),
- review requirements and documentation,
- define tests and test priority,
- design tests,
- define test outcome,
- change app code to make testing possible,
- automate tests,
- run tests,
- analyze test results,
- communicate findings,
- etc.

Resources

Who is part of your team and what are their roles? How much time will they have or are they available per sprint to spend on testing? An example:

- IT manager (overall support, vision on quality, budget)
- Productowner (budget, time, priority of test automation)
- Tester (designing tests, building tests, analyzing results)
- Developer (making testable code, writing tests, assisting the tester)

Test scope

What is included in tests and what is not included?

Scope is limited to the domain model and microflows when testing with MTA. This means that

- business logic
- data manipulation

are included;

- page functions,
- input validation,
- spelling/grammar

are excluded.

Also consider excluding any aspect that actually tests if the Mendix platform is working correctly, like numeric ranges, attribute value unicity, etc.

Test strategy

What approach will you use when you start building your Test Suites? Where will you start? For which existing functions will you write tests? For example:

- use existing requirements documentation as a basis for testing,
- focus testing on functions that have proven to be unreliable in the past,
- use tester's or developer's knowledge and experience to determine what to test,
- test all new features developed in a sprint,
- etc.

Note that some techniques do not work well in coordination with test automation, like black box testing (writing tests without knowing or looking at the software code), exploratory testing (creating tests based upon the result of previous tests) and error-guessing (using experience of the tester to anticipate defects and designing tests specifically to expose them).

Secondly, how will you integrate (automated) testing in your development cycle? Which (other) tools do you need for testing and test management? And how will you communicate and keep track of bugs found? It may be useful to create a priority matrix that describes which priorities should be fixed first (and even include this in the SLA with your customer). An example from Menditect's own SLA, where response time to customers and bug resolution time is separately indicated:

	Response time	Resolution time
Critical priority	< 2 office hours	< 8 office hours
High priority	< 8 office hours	Next Business Day
Medium priority	Next Business Day	Reasonable effort
Low priority	Reasonable effort	At own discretion

Lastly, think about the different kinds of testing and describe how they will be done, and who will be responsible for making sure that they are done.

Unit testing

Describe how unit testing will be performed, what error level is acceptable and what code coverage is desired. Usually when you unit test a single microflow, you want to test all possible execution paths, meaning a 100% code coverage, unless there is a valid reason to deviate from that. Make sure to read the best practices on the Menditect documentation site about unit testing:

➔ https://documentation.menditect.com/bestpractice/unittest

During test design, start by making a list of microflows that should be unit tested. Then note what data you need to create the unit test and what the expected output should be.

One example to base your list of microflows on:

- use Mendix' Advanced Find window to search for the largest microflows;
- use the Find usages per microflow to search for microflows that have the most dependencies (although this will not show submicroflow usage)
- use Teamserver commit list to find the microflows that are changed the most, or changed most recently

User Acceptance / Process Testing

How will you approach the testing of user transactions in the software, how will automation mock user input, will customers be part of the user acceptance testing? Make sure to read the best practices on the Menditect documentation site about process testing:

➔ https://documentation.menditect.com/bestpractice/processtest

System and Integration Testing

Who will be conducting System and Integration Testing on your project, and how? Which systems are integrating with your app? How can you test the microflows that perform web service calls or handle published web services?

Regression Testing

Do you have a Ci/Cd pipeline that you can use for regression testing? Consider which parts of the software need to be regression tested to make sure that changes in the software do not have unexpected side effects. Enable Ci/Cd on the Test Configurations that cover this.

Design Microflow testing

How to prepare for testing microflows is a perpendicular strategy. A single microflow can be regarded as a unit, a process or can be used for regression testing. More important to consider when designing tests is the amount of different possible execution paths or outcomes that exist in the microflow. This amount can grow rather quickly.

An example to illustrate:

- Microflow ACT_CreateOrder takes 2 input parameters,
    - a Product object (with mandatory price and name attributes)
    - a boolean HasDiscount
- The output of the microflow is an Order object.

Given these parameters there are already 10 test variations to consider:

	Happy flow?	Product object	/price	/name	Has Discount	Order object
Scenario 1	Yes	Not Empty	Not Empty	Not Empty	true	!= Empty
Scenario 2	Yes	Not Empty	Not Empty	Not Empty	false	!= Empty
Scenario 3	No	Not Empty	Empty	Not Empty	true	= Empty
Scenario 4	No	Not Empty	Empty	Not Empty	false	= Empty
Scenario 5	No	Not Empty	Not Empty	Empty	true	= Empty
Scenario 6	No	Not Empty	Not Empty	Empty	false	= Empty
Scenario 7	No	Not Empty	Empty	Empty	true	= Empty
Scenario 8	No	Not Empty	Empty	Empty	false	= Empty
Scenario 9	No	Empty	n/a	n/a	true	= Empty
Scenario 10	No	Empty	n/a	n/a	false	= Empty

To achieve a 100% test coverage on this microflow at least 10 tests are needed, and even this is only considering data; without defining any expected error messages that should or should not occur.

As part of the test strategy, **define the maximum number of variations** that are acceptable to achieve a 100% test coverage. Furthermore, define a strategy for the situation when exceeding this amount. What should the tester base their decisions on which scenarios need to be covered? Consider the following elements:

- Happy path (at least these should be covered)
- Probability (more probable scenarios first)
- Risk (high risk scenarios first)
- Impact (high impact scenarios first)

Test environment

What environment will you test on? Does it need to match certain specifications? How will you make sure that it is available and testing does not interfere with other activities? In MTA you can choose from an environment in the Mendix Cloud, another cloud provider, but you can also test an environment on a local computer.
This is described on the Menditect documentation site:

➔ https://documentation.menditect.com/howtos/test-and-debug-locally

Also think about what kind of hardware and browser clients should use, what authentication provider, etc.

Features to test

This is a listing of what is to be tested from the user's viewpoint of what the system does. The user can base their list on elements shown on pages. Make sure not to describe the technical parts of the system, no references to code or model elements but only to what the user sees in the app. No need to go too much in detail here, details like input/output per user transaction and expected results are part of test design, not test planning.

Note that "Manual" Test Cases in MTA are also very useful to create this list.

Features not to test (optional)

It can be helpful to note features that are out of scope, even though this is also generally covered in the test scope chapter. For example: user management, API configuration, master data setup, etc.

Entry criteria

List all prerequisites needed to start testing. In general:

- test environment is set up and test data is available,
- all relevant documentation is available,
- test activities are estimated and testing is included in budgeting,
- testers are included in meetings about the product,
- testing is performed in sync with development,
- testers have investigated and understood the functionality and available documentation and have reviewed the test cases.
- etc.

When designing tests, you need to think about entry criteria for creating test cases in MTA. For example:

- MTA is running & MTA users associated to Mendix user with API key and PAT,
- testers in MTA have access to environments,
- no new revisions are to be deployed to the test environment,
- the relevant microflows are known and readable by the same person(s) creating the test cases,
- the revision that is deployed to the environment is downloaded.
- etc.

Exit criteria

When do we stop testing? In general:

- when budget is hit, or:
- all tests are executed and pass at least once, and
- all failed tests are analyzed, and
- there are no outstanding changes in the software, and
- all bugs found are assigned and the app does not ship with high priority bugs,
- etc.

Schedule / planning

Make an estimation with your team for your testing efforts and create a realistic planning of your activities. Estimating test activities should be done with all roles in the team as part of the (sprint) planning and can make use of the same refinement and estimating techniques as used for development activities. Plan for testing the most obvious process of the software first, and split this up into a set of units that can be tested separately or consecutively as a whole process.

Make sure to meet regularly to update the schedule.

Risks and contingencies

What are the overall risks to the project with an emphasis on the testing process?

For example:

- lack of resources,
- lack of communication between developers and testers.
- lack in training of the tools used, including MTA,
- lack of time due to other priorities,
- changes in the requirements or in the software being tested.

And what are the overall risks affecting the success of the product?

- poor quality of the software code, making it harder to understand,
- poor code testability (large microflows, entangled domain models),
- high failure count, making it harder to fix defects,
- hidden defects (that only occur in production or specific circumstances),
- performance issues.

For each risk, make sure to create a contingency plan. For example: test only high risk features, have people work overtime, accept shipping with known issues, etc.

Deliverables

What documents do you plan to create while testing? How will you turn test results into something that you can handover to other members of the team or to management?

Make sure these documents are thorough so that they can still be read after a while or by someone new in the project; cover the explanation of any technical jargon and include architectural images of the state of the project at the time.