



MORPHEMIC

Test cases and testing

Modelling and Orchestrating heterogeneous Resources and Polymorphic applications for Holistic Execution and adaptation of Models In the Cloud

H2020-ICT-2018-2020
Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number
871643

Duration
1 January 2020 –
31 December 2022

www.morpheMIC.cloud

Deliverable reference
D4.5

Date
1 June 2020

Responsible partner
7bulls.com

Editor(s)
Katarzyna Materka

Reviewers
Alexandros Raikos
Amir H. Taherkordi

Distribution
Confidential

Availability
www.morpheMIC.cloud

Executive summary

MORPHEMIC is a complex multi-cloud solution, containing a variety of modules, created by the cooperation of many people and organizations. Therefore, it is crucial to put extra attention to testing, preparing and executing test cases, managing the lifetime of test cases, and reporting issues. This approach allows for uninterrupted supervision of meeting functionality, quality, and reliability requirements by immediate and constant detection and resolution of any problems or missed objectives.

This deliverable is intended to be a set of guidelines on a lower development and architecture level, completing the Validation Framework. It contains definitions and a comprehensive description of test cases, as well as a description of the process of creating a test case, its life cycle, and guidelines for test cases' types and priorities. A specification of the testing environment is given, then all executed test cases in both releases, Release 1.0 and Release 1.5, are listed, grouped, and summarized.

Author(s)
Anna Wyszomirska, Marcin Byra



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Revisions

Date	Version	Partner	Description
01.02.2021	0.1 (draft)	7bulls	First Draft
10.02.2021	0.5	7bulls	Two sections added
16.05.2021	1.0	7bulls	Preliminary version
20.05.2021	1.1	University of Piraeus	Official review by Alexandros Raikos
21.05.2021	1.2	7bulls	Revised version
25.05.2021	1.3	University of Piraeus	Second review by Alexandros Raikos
26.05.2021	1.4	7bulls	Final version of the contents
26.05.2021	1.5	7bulls	Final style formatting
08.06.2021	1.6	University of Oslo	Second official review by Amirhosein Taherkodi
10.06.2021	1.7	7bulls	Revised version
22.06.2021	1.8	IS-Wireless	Review of the document with comments inline.
29.06.2021	1.9	7bulls	Final revision

Table of Content

1	Introduction.....	5
1.1	The scope	5
1.2	Audience	5
1.3	Structure of the document.....	5
2	Test Case Management.....	6
2.1	Test Case definition.....	6
2.2	Test case creation and update process.....	6
2.3	Test case life cycle.....	9
2.4	Test case elements.....	11
2.4.1	Required.....	11
2.4.2	Optional	11
2.4.3	Generated – elements added automatically by JIRA	11
2.5	Test case types.....	11
2.5.1	Functional Test Cases	12
2.5.2	Regression Test Cases	12
2.5.3	Non-Functional Test Cases.....	12
2.6	Test case priorities	12
2.7	Positive and negative test cases	13
3	Testing environments.....	13
4	Test cases	13
4.1	Summary of test cases	13
4.2	Naming patterns	14
4.3	Executed test cases.....	14
5	Conclusions.....	18

Index of tables

Table 1: Engineering testbed specification: Dell Inc. PowerEdge M630.....	13
Table 2: Categories of test cases in Release 1.0 and Release 1.5	13
Table 3 Naming patterns.....	14
Table 4: Abbreviations.....	14
Table 5: Test cases executed in MORPHEMIC release 1.0	14
Table 6: Test cases executed in MORPHEMIC release 1.5	17

Index of figures

Figure 1: JIRA login view	6
Figure 2: JIRA Create button.....	7
Figure 3: JIRA new issue creator.....	7
Figure 4: JIRA search box	8
Figure 5: Searching for issues in JIRA	8
Figure 6: Search filters in JIRA	8
Figure 7: Search results in JIRA	9
Figure 8: Test Case lifecycle graph	10

1 Introduction

1.1 The scope

This document presents information related to the testing of MORPHEMIC project. Testing is an essential part of every huge project created by the cooperation of many groups and people. To ensure quality and reliability, it is crucial to follow all procedures of testing, prepare and execute test cases, report any issues and take care of the whole lifetime of the test case or issue. An exhaustive description of a validation framework can be found in Deliverable *D6.2 Validation framework design*. This document presents a realization of the assumptions introduced in Deliverable *D6.2 Validation framework design*.

All new functionalities introduced during Release 1.0 and Release 1.5 need to be formally tested and a comprehensive list of these tests is presented here. Release 1.0 was focused on the integration of the MELODIC¹ platform with the ProActive system. The first implementation of polymorphic adaptation using component variant activation has been introduced. ProActive is already existing and constantly maintained Open Source solution with extensive cloud resource management functionality. It fits perfectly as an Executionware of the MELODIC platform by preparing the ground for polymorphic adaptation needed in the MORPHEMIC project since it supports edge computing in fog networks. All required work was to provide a reliable interface between MELODIC Upperware and the ProActive system. Each test case of Release 1.0 is related to this new integration. Release 1.0 was initially planned for 31st of March 2021, but as some essential information flows between two projects proved to be more difficult to implement or needed to be developed from scratch, the effective release date was 15th of June 2021.

Release 1.5 is a part of the MORPHEMIC project, providing a further extension of the existing system by introducing the Proactive Adaptation as well as adding the Forecasting Module. The test cases related to this release are either a regression (as the test cases from the Release 1.0 should apply to Release 1.5) or new test cases focused on the Proactive Adaptation and Forecasting module. Release 1.5 is planned to be delivered on 31st of August 2021.

This document was divided into two parts. First, all information related to test case management is presented, containing all necessary definitions, explanations, and guides. Then, testing environments and detailed lists of executed (that is planned, created, and performed) test cases are given with analysis of the work done during both releases. A detailed structure of the document is presented in Section 1.3 below.

1.2 Audience

This deliverable is intended for those involved in the software quality assurance process and their outcome:

- mandatory for test teams and architects:
 - the test team needs to know what the test case creation process is, what the life cycle of the test case is, and which elements the test case includes
 - architects must check whether the test cases are consistent with the (system) specifications
- recommended for developers – they should know what the life cycle of the test case is and how the system will be tested
- optional for the rest of the project members.

1.3 Structure of the document

This deliverable describes the two integration releases and their initial test environments. It contains the following information, which is structured in the subsequent chapters:

- Chapter 2: Test Case Management - a full introduction into testing, including test case creation process, test case lifecycle, covering all elements of a test case, as well as general classification of test cases.
- Chapter 3: Testing environments - a specification of the environments used for testing
- Chapter 4: Test cases - A detailed list of all executed test cases during Releases 1.0 and 1.5

¹ <https://melodic.cloud/>

- Chapter 5: Conclusions - A brief summary of this deliverable

2 Test Case Management

The test process starts at the very beginning of a project life cycle with the Analysis and Design phases. The test team prepares a *test plan* which contains test cases (as described in Deliverable 4.4 Test Strategy). The test plan also explicates the dependencies between the test cases; in particular, it clarifies in what order the test cases should be executed. This approach has important benefits:

- It allows the test team to understand the system to be developed
- It serves as a review of the system specifications and requirements
- It eases solving issues, as both the test teams and the development teams have the same base data (the test data; input parameters for test cases, necessary to execute test cases and to reproduce bugs, if occurring during test case execution)

In the following sections, we will present a Test Case definition (Section 2.1), detailed Test Case creation and update process guide (Section 2.2), lifecycle, states and roles guide (Section 2.3), test case elements as they are presented in the JIRA software (Section 2.4), types of test cases (Section 2.5), test case priorities (Section 2.6) and positive and negative testing definitions (Section 2.7).

2.1 Test Case definition

A test case is a set of test data, pre-conditions, expected results and post-conditions for the tested implementations, developed for a specific purpose or for the condition mapping to the test, such as the execution of a program path, or to verify compliance with a specific requirement. A test case describes how to perform a specific test. Test cases will be created by the test team, either through its members or through the test leader.

2.2 Test case creation and update process

Test cases will be prepared in the JIRA² system for testing, which was chosen as a test management system for the MORPHEMIC project. The discussion of the management system can be found in Deliverable *D4.4 Test strategy*. MORPHEMIC uses the same test and bug tracking system as MELODIC because bugs observed can be related to both platforms and should be solved across the two platforms.

The process below details how to create a test case in the JIRA system:

1. In the browser, open the page: <https://jira.7bulls.eu>
2. Enter your credentials in the fields *Username* and *Password* and press the *Log In* button:

Login

Username

Password

☐ Remember my login on this computer

Not a member? To request an account, please contact your JIRA administrators.

[Can't access your account?](#)

Figure 1: JIRA login view

² <https://www.atlassian.com/software/jira>

3. To create a new test case, press *Create* button:

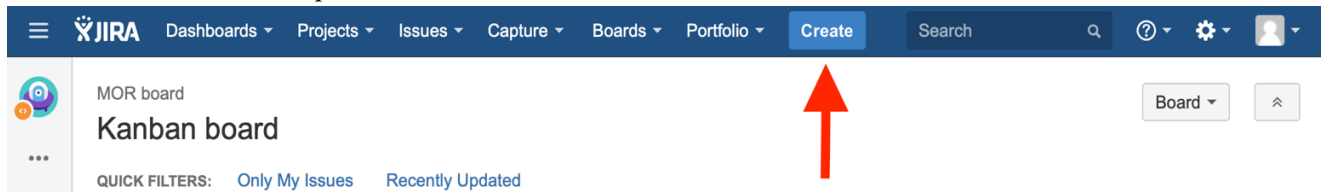


Figure 2: JIRA Create button

4. Fill in all required fields and press the *Create* button:

A screenshot of the 'Create Issue' form in JIRA. The form is titled 'Create Issue' in a light gray header bar. Below the header, there are several fields: 'Project' with a dropdown menu showing 'Morphemic Testing (MORPH...)', 'Issue Type' with a dropdown menu showing 'Test Case', 'Summary' with a text input field, 'Reporter' with a dropdown menu showing 'Marcin Byra', 'Priority' with a dropdown menu showing 'Medium', and 'Assignee' with a dropdown menu showing 'Automatic'. Below these fields are three large text input areas labeled 'Input Conditions', 'Steps To Complete', and 'Expected results'. At the bottom, there is a 'Labels' dropdown menu. The form is styled with a light gray background and white text. A red arrow points to the 'Create' button in the top navigation bar of the previous screenshot.

Figure 3: JIRA new issue creator

5. After following these steps, a new test case will be created.

Chapter 2.5 covers all the (required and optional) information that can be provided about a certain test case.

A test case can be identified and edited via the following procedure:

1. Enter the issue number in the search box and press the *Enter* button:



Figure 4: JIRA search box

2. Search the issue via:

- a. pressing the arrow next to the *Issues* button
- b. choosing the option *Search for Issues*

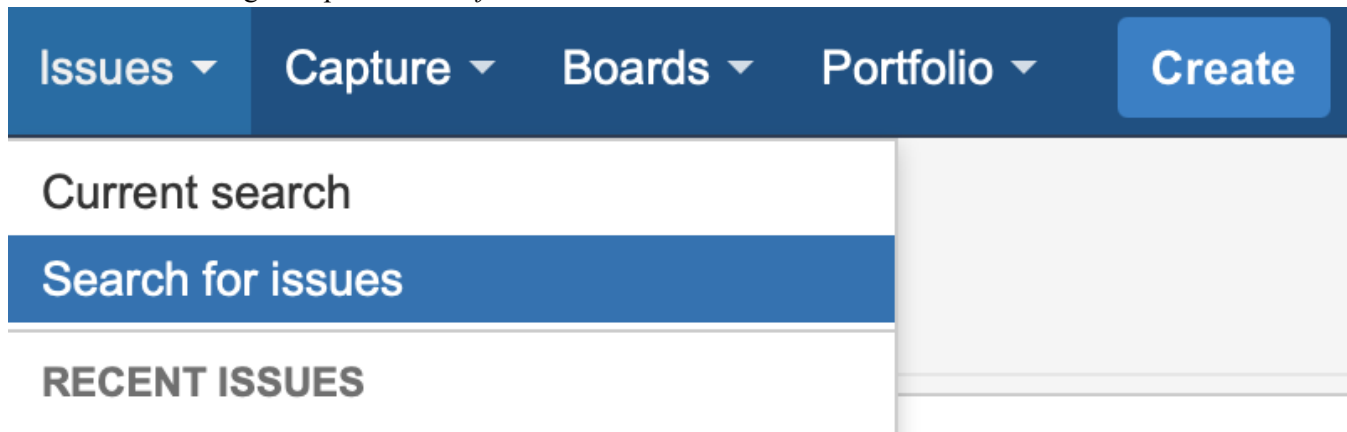


Figure 5: Searching for issues in JIRA

- c. entering the appropriate search criteria and waiting for automatic results update (a query can be written manually by clicking on *Advanced* and pressing Enter):

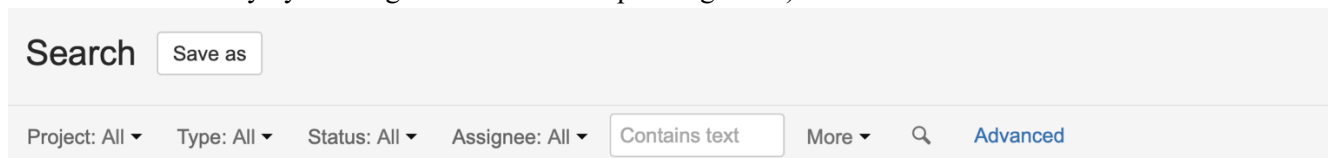
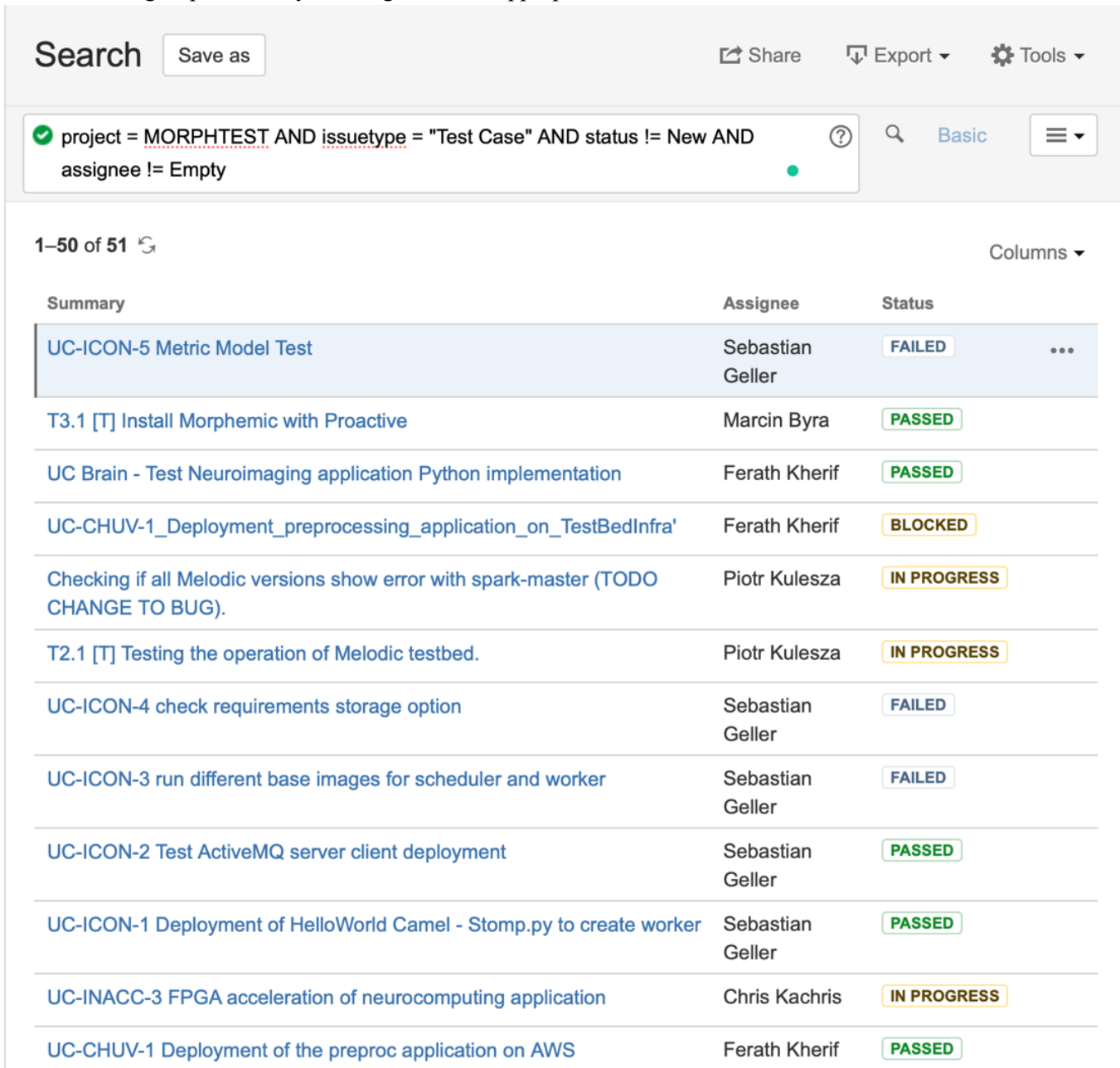


Figure 6: Search filters in JIRA

After following steps 2.a-2.c, you will get a list of appropriate issues:



Search

✓ project = MORPHEMIC AND issuetype = "Test Case" AND status != New AND assignee != Empty

1–50 of 51 Columns

Summary	Assignee	Status
UC-ICON-5 Metric Model Test	Sebastian Geller	FAILED
T3.1 [T] Install Morphemic with Proactive	Marcin Byra	PASSED
UC Brain - Test Neuroimaging application Python implementation	Ferath Kherif	PASSED
UC-CHUV-1_Deployment_preprocessing_application_on_TestBedInfra'	Ferath Kherif	BLOCKED
Checking if all Melodic versions show error with spark-master (TODO CHANGE TO BUG).	Piotr Kulesza	IN PROGRESS
T2.1 [T] Testing the operation of Melodic testbed.	Piotr Kulesza	IN PROGRESS
UC-ICON-4 check requirements storage option	Sebastian Geller	FAILED
UC-ICON-3 run different base images for scheduler and worker	Sebastian Geller	FAILED
UC-ICON-2 Test ActiveMQ server client deployment	Sebastian Geller	PASSED
UC-ICON-1 Deployment of HelloWorld Camel - Stomp.py to create worker	Sebastian Geller	PASSED
UC-INACC-3 FPGA acceleration of neurocomputing application	Chris Kachris	IN PROGRESS
UC-CHUV-1 Deployment of the preproc application on AWS	Ferath Kherif	PASSED

Figure 7: Search results in JIRA

After following steps 1 or 2, you will be able to see and edit the details of a chosen test case. Additional information about the usage of JIRA for testing can be found in the JIRA Users Guide³.

2.3 Test case life cycle

This chapter contains information about the test case lifecycle in the MORPHEMIC project. The following states apply for a certain test case:

- NEW
- TO DO
- IN PROGRESS
- TO TEST
- DONE
- REOPEN

³ <https://confluence.atlassian.com/jira064/jira-user-s-guide-720416011.html>

- *CLOSED*

The Figure 8 below depicts the workflow for test case handling in MORPHEMIC.

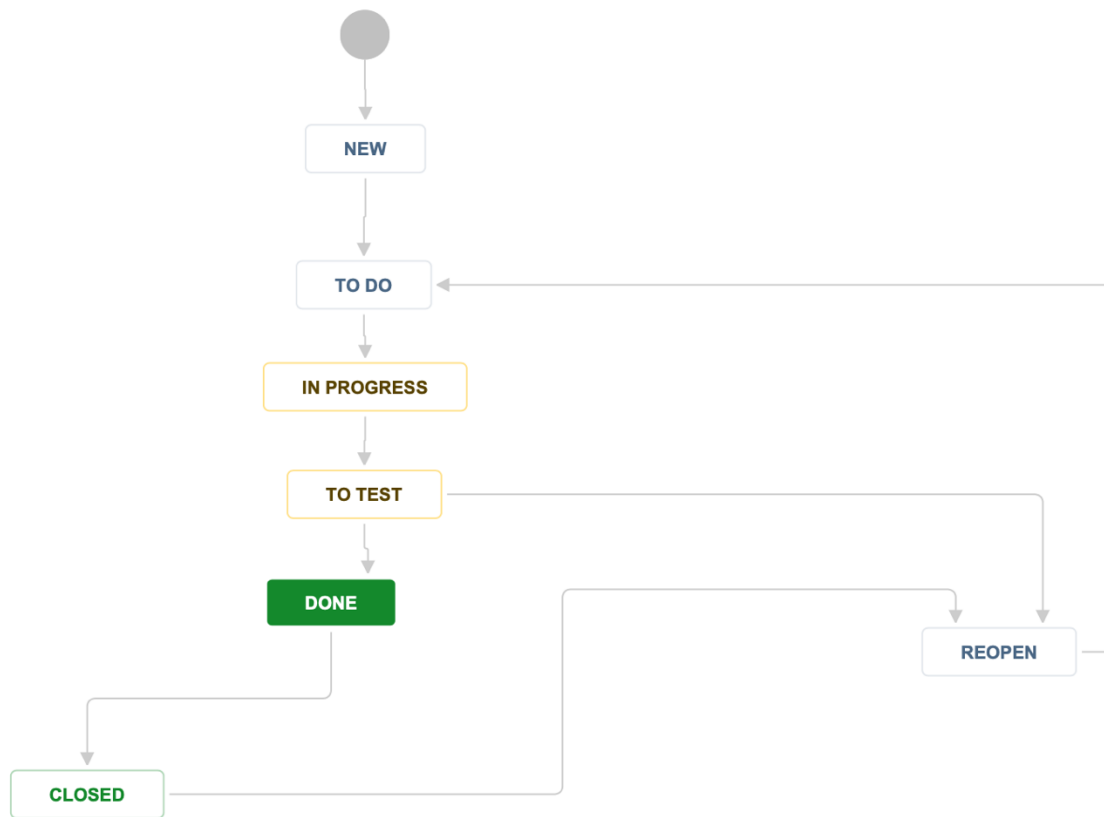


Figure 8: Test Case lifecycle graph

After a test case is created by one of the members of a Test Team, it has the *NEW* state and it is assigned to the Test Leader. Then the Test Leader accepts the test and assigns it to one of the Test Team members, changing the state to *TO DO*. When a Test Team member starts working on the given test case, the state is changed to *IN PROGRESS*. If the test case is executed without problems, the Test Team member changes the state to *TO TEST* after completing all steps specified in the test case description.

After the test case is executed, the Test Team member changes its state to *TO TEST*. It can be assigned to the same or another Test Team member. From there, the state may be changed to:

- *DONE*, if the result is as expected and the test case is considered as completed. It is assigned to the Test Leader.
- *REOPEN*, if there are any problems, required modifications or another reason to rerun the test. It can also be assigned to the Test Leader, who decides who should be assigned to this test case after reopening.

When the Test Leader accepts the executed test case, it changes its state to *CLOSED*, what basically finishes its lifecycle. However, if there are any reasons for reopening the test case (for example, modifying related functionalities require test case to be verified again), its state can be always changed to *REOPEN*.

In each state, a test case can be marked as blocked. For example, if there is some missing software functionality that is necessary to perform a test case, a Test Team member has to create another issue in JIRA and mark the test case as blocked but the new issue. It is a popular scenario, most often happening while a test case is in *TO DO*, *IN PROGRESS* or *REOPEN* states. A Test Team member assigned to the issue has to wait until the problem is resolved, then the test case can be executed again.

2.4 Test case elements

A typical test case consists of some basic elements that are always present, regardless of project or organization. However, for clarity, let us skip the theoretical introduction and focus on specific information filled in the respective JIRA form to construct a new test case. Any test case contains elements, which can be divided into three groups: Required, Optional and Automatically generated. Each test case is also in a specific state during all its life cycle, but the state is not considered as test case element, and it is not required to specify it while creation as all new test cases are automatically set to state *NEW* by JIRA.

2.4.1 Required

These elements require user input on each test case. They are:

- **Description** – a brief description or title of a test case, usually not longer than one sentence,
- **Creator** – JIRA software usually suggests some user, but the test creator must accept (or change) this person,
- **Priority** – one must specify how important or how urgent is the test case. Default is *Medium* and the priority system is explained below in Section 2.6
- **Assign to** – one must specify who should handle the use case after it is created.

Note that the three elements below are not formally required by JIRA form, but it is essential to fill them for every test team member to fully understand the test case. Thus, they are listed in this Required group.

- **Input Conditions** - a comprehensive description of prerequisites necessary to go through the test case steps,
- **Steps To Complete** - all steps that a Test Team member must perform to complete the assigned test case,
- **Expected Results** - description of the results, expected conditions or state of a program a Test Team member should get after the successful execution of all test case steps.

2.4.2 Optional

- **Labels** - short names related to a subject of a given test case, or related to a set of test cases, for example *Release_1*, *GUI*,
- **Related issues** – links to other JIRA tasks (e.g., test cases) together with a relation type (e.g., blocks, is blocked by, duplicates, etc.),
- **Attachments** - files uploaded that are necessary or useful for a given test case (for example, *.jpg* or text files containing data),
- **Planned finish date** - approximate time of finishing the assigned test case, if it is possible to estimate,
- **Fixed in** – a release version used to track different software developments and updates,
- **Description** - usually *Input Conditions*, *Steps to Complete* and *Expected Results* are self-explanatory, but any supplementary information should be placed here.

2.4.3 Generated – elements added automatically by JIRA

- **ID** - an identifier which is unique amongst all JIRA test cases, tasks, issues, etc.,
- **Creation date** - the specific time when a test case was created,
- **Update date** – the time of the last modification of the given test case (status change, assigning to another Team Member, changing description, etc.),
- **Activity** - a log containing every action performed by any user related to given use case.

2.5 Test case types

There exist several types of software tests. A list of the test case categories utilized in the MORPHEMIC project with basic definitions is presented below. A more exhaustive explanation of testing types along with their objectives and tools can be found in Deliverable *D4.4 Test strategy*, Chapter 4.

2.5.1 Functional Test Cases

Each Functional Test Case is designed to validate the features and operational behaviour of software to ensure that they correspond to the specification. The Functional Test Case usually corresponds to some requirement, be it functionality, a user interface element or flow, integration with another module, etc. They are an essential part of a testing architecture, allowing the development team to correct any issues or implement missing features blocking the Functional Test Case from execution.

2.5.2 Regression Test Cases

Regression Test Cases are heavily utilized in the MORPHEMIC project. They are responsible for retesting a software system after it was modified in order to ensure that all bugs were resolved and fixed, that no previously working function fails as a result of the modifications, and that newly added features don't create problems to the existing parts of the software. In other words, they are quality control test cases enabling the extension and development of the software system without corrupting the existing parts. In Release 1.5, a lot of the Release 1.0 test cases are reused as Regression Test Cases.

2.5.3 Non-Functional Test Cases

This group of test cases includes various types of test cases, such as performance tests, stress tests, failover tests, and security tests. The amount of each type of test strongly depends on the project's needs. They check if the system achieves required response times and verifies its behaviour under varying workloads, check the system behaviour when there are insufficient resources or network problems. They can also ensure that the system works properly after restart or failure and other extraordinary situations; they also verify the security of the system by trying to find potential threats and vulnerabilities.

2.6 Test case priorities

Prioritization of test cases is a complex and not strictly defined problem. One general purpose of priorities is to minimize cost, time, and effort during the testing phase of a specific software. Tests can be divided into categories from high to lower priority based on the end-user perspective (based on ISTQB methodology):

- **Blockers** - these test cases validate essential functionalities of the software. When they fail, the software is considered as useless,
- **Critical** - the focus is on verifying where the most important cases work and where they do not, causing the whole application to fail in some occasions. If they fail, the software is considered as not having its necessary functions working.
- **Major** – these test cases test basic functionality of the application. From the user perspective, the process of ensuring they can be executed properly can be sometimes postponed, but are noticeably degrading the experience,
- **Minor** – this category includes all small bugs.

However, the above grouping can be considered as one method of categorization but also other methods can be incorporated with each project individually. The groups above describe the significance of a bug for the whole project. Sometimes it is convenient to use simple priorities describing how urgent it is to solve a particular bug at a given time. That is why we utilize the JIRA priorities:

- **Lowest**
- **Low**
- **Medium**
- **High**
- **Highest**

This is a very simple yet easy-to-use way of assigning a priority to the test case. Additionally, JIRA forces the test case creator to choose the appropriate priority from this scale, what is a desirable behaviour. At the same time, the categories from the first grouping can be added as labels if necessary.

2.7 Positive and negative test cases

Almost all test cases can be assigned as Positive or Negative. Below are the definitions of each type.

Positive test case (or **Positive Testing**) is performed by providing valid data (or performing a valid action, following a valid instruction, etc.) to check if the application behaves as expected. It checks if the output of given test is exactly as specified in a test case description.

Negative test case (or **Negative Testing**) is performed by providing specified invalid data (or performing an invalid action, following an invalid instruction etc.). It checks if the application handles the invalid scenario as expected (as specified in test case output) and remains stable despite the invalid input.

3 Testing environments

A testing environment is a setup of software and hardware for the testing teams to execute test cases. In other words, the environment supports test execution with all suitable hardware, software, and network configured. Such environments may vary significantly in size: the development environment is typically an individual developer's workstation, while the production environment may be a network of many geographically distributed machines in data centres, or virtual machines in Cloud computing. The code, data, and configuration may be deployed in parallel, and need not to be connected to the corresponding tier; For example, pre-production code may connect to a production database.

For both releases, we used the following testing environment, which was determined after real case analysis of Use Partners applications so that it does not behave as a 'bottleneck'. Even if the testing environment itself does not include the use case application, it needs to conform to the needs of the testing cases, especially the performance ones.

Table 1: Engineering testbed specification: Dell Inc. PowerEdge M630

Resource	Details
CPU	2x Intel Xeon E5-2670 v3 30 MB cache @2,30 GHz, 12 cores with 24 threads
RAM	384 GB ECC
Storage	2x 300GB Disks in RAID 1
Network	8x network cards at 10 Gbps

4 Test cases

4.1 Summary of test cases

In Table 2, we summarize all executed test cases, divided into groups. The number of test cases of both releases are presented in separate columns. We give a short explanation of each group below. The test cases executed during Release 1.0 test the functionalities developed during this phase, while the test cases of Release 1.5 also include some Release 1.0 tests as Regression Test Cases.

Table 2: Categories of test cases in Release 1.0 and Release 1.5

Test cases groups	Test cases created during release 1.0 (summary)	Test cases created during release 1.5 (summary)
Initial deployment	32	10
Metric management	3	5
Application creation	19	14
Reconfiguration	1	-
Forecasting module	-	2

- **Initial deployment** – This group contains all scenarios related to the initial deployment of an application in the Melodic platform.
- **Metric management** – Metric management means the collection, processing (aggregation), storage and delivery of raw and composite metrics, as well as CAMEL events based on these metrics
- **Application creation** – All test cases related to designing, creating and exporting a CAMEL model are grouped here
- **Reconfiguration** – Reconfiguration of the application based on the new solution found by Reasoning part of the system.
- **Forecasting module** – Tests cases related to the forecasting module

4.2 Naming patterns

In order to formalize the name of test cases, following naming patterns are used. Please consider that this naming convention is optional and final name of the test case is up to decision of the test case creator.

Table 3 Naming patterns

Type	Pattern
use cases	UC-partner name-ordinal number [T/N] Name of the test case
use case test case example	UC-ICON-1 Deployment of the application on AWS
features	Ffeature number.subfeature number-ordinal number [T/N] Name of the test case
feature test case example	F6.1-10- Create metric type Model inside CAMEL model
other test cases	Ttest number.feature number [T/N] Name of the test case
other test case example	T1.5 [T] Create VM on AWS resources

Where optional [T] denotes a positive test and optional [N] denotes a negative test.

Table 4: Abbreviations

Abbreviation	Full name
F	Feature
UC	Use Case
ISW	ISWireless
CHUV	Centre Hospitalier Universitaire Vadois
INACC	InAccell
T	Test
P	positive (test case result)
N	negative (test case result)

4.3 Executed test cases

This subsection presents a list of executed Test Cases for MORPHEMIC release 1.0. In particular, Table 5 shows the identifier of the executed test cases for Release 1.0, including a summary of each case, with optional [P] or [N] denoting positive or negative test case, in terms of its task corresponding identifier and name is explained in Section 4.1.

Note that some tests cases were executed for both releases.

Table 5: Test cases executed in MORPHEMIC release 1.0

KEY	Name	Priority	Group
MORPHTEST-1	T1.1[P] Installation and deployment of FCR application on AWS	Medium	Initial deployment
MORPHTEST-3	T1.2[P] Installation and deployment of two-component application on AWS	Medium	Initial deployment

MORPHTEST-4	T1.3[P] Installation and deployment of one-component application on OpenStack	Medium	Initial deployment
MORPHTEST-5	T1.4[P] Installation and deployment of two-component application on OpenStack	Medium	Initial deployment
MORPHTEST-6	T1.5[P] Create VM on AWS resources	Medium	Initial deployment
MORPHTEST-7	T1.6 [P] Create VM on OpenStack resources	Medium	Initial deployment
MORPHTEST-8	T1.7 [P] Successfully retrieve Node Candidates for AWS	Medium	Initial deployment
MORPHTEST-9	T1.8 [N] Assure that wrong cloud configuration forbids retrieval of Node Candidates	Medium	Initial deployment
MORPHTEST-10	T1.9 [P] Successfully retrieve Node Candidates for OpenStack	Medium	Initial deployment
MORPHTEST-11	F1.2-MARKOS' Web Crawler: Apache configured as source code repository	Low	Initial deployment
MORPHTEST-12	F1.3-MARKOS' Web Crawler Test: GitHub Plugin configured as source code repository	Low	Initial deployment
MORPHTEST-13	F1.1-MARKOS' Web Crawler Test: JQuery Plugin configured as source code repository	Low	Initial deployment
MORPHTEST-14	UC-ISW-T-01: CPU/RAM utilization metric acquisition testing	Medium	Metric management
MORPHTEST-15	UC-ISW-T-02: Constraint testing	Medium	Initial deployment
MORPHTEST-16	UC-ISW-T-03: SLO violation testing	Medium	Metric management
MORPHTEST-17	UC-ISW-T-04: BYON testing	Medium	Initial deployment
MORPHTEST-18	UC-ISW-T-05: vRAN application deployment	Medium	Initial deployment
MORPHTEST-19	UC-ISW-T-06: vRAN configuration and interaction testing	Medium	Initial deployment
MORPHTEST-20	UC-ISW-T-07: application metric acquisition	Medium	Metric management
MORPHTEST-21	UC-INACC-1 FPGA integration on Pro active	Medium	Initial deployment
MORPHTEST-22	F6.1-1- Camel Designer Project Creation in Modelio	Medium	Application creation
MORPHTEST-23	F6.1-2- Camel Designer Create a new Camel Model	Medium	Application creation
MORPHTEST-24	F6.1-3- Camel Designer Import CAMEL Model from CAMEL file	Medium	Application creation
MORPHTEST-25	F6.1-4- Camel Designer Export CAMEL Model into XMI file	High	Application creation
MORPHTEST-26	F6.1-5- Camel Designer Export CAMEL Model into CAMEL file	Low	Application creation
MORPHTEST-27	F6.1-6- Create Deployment Model inside a CAMEL Model	Medium	Application creation

MORPHTEST-28	F6.1-7- Create Requirement Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-29	F6.1-8- Create data type Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-30	F6.1-9- Create organisation Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-31	F6.1-10- Create metric type Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-32	F6.1-10- Create constraint Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-33	F6.1-11- Create unit model Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-34	F6.1-12- Create a software component inside a Deployment Model	Medium	Application creation
MORPHTEST-35	F6.1-13- Create a host port inside a Software Component	Medium	Application creation
MORPHTEST-36	F6.1-14- Create a communication port inside a Software Component	Medium	Application creation
MORPHTEST-37	F6.1-15- Create a script configuration inside a Software Component	Medium	Application creation
MORPHTEST-38	F6.1-16- Create a requirement inside a Requirement model	Medium	Application creation
MORPHTEST-39	F6.1-17- Create an optimization requirement inside a Requirement model	Medium	Application creation
MORPHTEST-40	F6.1-18- Create a metric variable inside a metric type model	Medium	Application creation
MORPHTEST-41	UC-ISW-T-08: MORPHEMIC deployment from the Gitlab repository	Medium	Initial deployment
MORPHTEST-42	UC-CHUV-1 Deployment of the preproc application on AWS	High	Initial deployment
MORPHTEST-43	UC-INACC-2 FPGA acceleration of edge application	Medium	Initial deployment
MORPHTEST-44	UC-INACC-3 FPGA acceleration of neurocomputing application	Medium	Initial deployment
MORPHTEST-45	UC-ICON-1 Deployment of HelloWorld Camel - Stomp.py to create worker	Medium	Reconfiguration
MORPHTEST-46	UC-ICON-2 Test ActiveMQ server client deployment	Medium	Initial deployment
MORPHTEST-47	UC-ICON-3 run different base images for scheduler and worker	Medium	Initial deployment
MORPHTEST-48	UC-ICON-4 check requirements storage option	Medium	Initial deployment
MORPHTEST-49	T2.1 [P] Testing the operation of Melodic testbed.	Medium	Initial deployment
MORPHTEST-51	UC Brain - Test Neuroimaging application deployment in test bed	Medium	Initial deployment

MORPHTEST-52	UC-CHUV-1_Deployment_preprocessing_application_on_TestBedInfra'	Medium	Initial deployment
MORPHTEST-53	UC Brain - Test Neuroimaging application Python implementation	Medium	Initial deployment
MORPHTEST-54	UC Brain - Test Neuroimaging application_docker implementation	Medium	Initial deployment
MORPHTEST-55	UC Brain - Test Neuroimaging application deployment and running with proactive	Medium	Initial deployment
MORPHTEST-56	UC Brain - Test Neuroimaging application deployment and running without proactive	Medium	Initial deployment
MORPHTEST-58	T3.1 [P] Install MORPHEMIC with Proactive	Medium	Initial deployment

Table 6: Test cases executed in MORPHEMIC release 1.5

KEY	Name	Priority	Group
MORPHTEST-100	1.5 - T1.4[T] Installation and deployment of two component application on Openstack	Medium	Initial deployment
MORPHTEST-99	1.5 - T1.5[T] Create VM on AWS resources	Medium	Initial deployment
MORPHTEST-98	1.5 - T1.6 [T] Create VM on OpenStack resources	Medium	Initial deployment
MORPHTEST-97	1.5 - T1.7 [T] Successfully retrieve Node Candidates for AWS	Medium	Initial deployment
MORPHTEST-96	1.5 - T1.8 [F] Assure that wrong cloud configuration forbids retrieval of Node Candidates	Medium	Initial deployment
MORPHTEST-95	1.5 - T1.9 [T] Successfully retrieve Node Candidates for OpenStack	Medium	Initial deployment
MORPHTEST-94	1.5 - F6.1-2- Camel Designer Create a new Camel Model	Medium	Application creation
MORPHTEST-92	1.5 - F6.1-3- Camel Designer Import CAMEL Model from CAMEL file	Medium	Application creation
MORPHTEST-90	1.5 - F6.1-6- Create Deployment Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-89	1.5 - F6.1-7- Create Requirement Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-87	1.5 - F6.1-8- Create data type Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-86	1.5 - F6.1-9- Create organisation Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-85	1.5 - F6.1-10- Create metric type Model inside a CAMEL Model	Medium	Application creation
MORPHTEST-84	1.5 - F6.1-10- Create constraint Model inside a CAMEL Model	Medium	Application creation

MORPHTEST-83	1.5 - F6.1-11- Create unit model inside a CAMEL Model	Medium	Application creation
MORPHTEST-82	1.5 - F6.1-12- Create a software component inside a Deployment Model	Medium	Application creation
MORPHTEST-81	1.5 - F6.1-13- Create a host port inside a Software Component	Medium	Application creation
MORPHTEST-80	1.5 - F6.1-14- Create a communication port inside a Software Component	Medium	Application creation
MORPHTEST-79	1.5 - F6.1-15- Create a script configuration inside a Software Component	Medium	Application creation
MORPHTEST-77	1.5 - F6.1-16- Create a requirement inside a Requirement model	Medium	Application creation
MORPHTEST-76	1.5. - Proactive notifies EMS for a new VM and COMMUNICATION with EMS ActiveMQ	Medium	Metric management
MORPHTEST-75	1.5 - MetaSolver notifies EMS for a CP model update	Medium	Metric management
MORPHTEST-74	1.5 - TEST COMMUNICATION with EMS REST endpoints	Medium	Metric management
MORPHTEST-73	1.5 - Performance model	Medium	Metric management
MORPHTEST-72	1.5 - Persistent storage	Medium	Metric management
MORPHTEST-71	1.5 - WEB-CRAWLER	Medium	Initial deployment
MORPHTEST-68	1.5 - T3.1 [T] Install Morphemic with Proactive	Medium	Initial deployment
MORPHTEST-67	1.5 - F2.1-1 [T] Forecasting Module - TFT test	Medium	Forecasting module
MORPHTEST-66	1.5 - F2.1-2 [T] Forecasting Module - NBEATS test	Medium	Forecasting module
MORPHTEST-65	1.5 Morphemic - Deployment of a FCR application on one Cloud Provider on machine with artifacts from previous deploy	Medium	Initial deployment
MORPHTEST-64	1.5 Morphemic - Deployment of a two-component application on one Cloud Provider	Medium	Initial deployment

5 Conclusions

This deliverable presented performed test cases in 1.0 and 1.5 integration releases and all information related to testing and test management. All team members utilize the guides presented in this document, as well as categories, names, and patterns. The test cases were created and executed using the test case procedures described in this deliverable. In particular, Table 4 contains the list of 55 Test Cases executed during Release 1.0, while Table 5 contains the list of 30 Test Cases executed in Release 1.5. All presented test cases were successfully executed and helped to verify the



correctness of various functionalities, ensure the system fulfils the project requirements, and detect a number of bugs and errors, as expected.