November 11, 2016

Callisto
Unit Test Plan
Version 1.0.0

Project team
J.N. Adegeest | 0841357
G. Foks | 0847293
S.J. den Hertog | 0853459
K.R. Klaren | 0812587
S. Lovett | 0927993
K. Man | 0929573
K. Nassar | 0926380
B. Raducanu | 0870280
T. van Schijndel | 0857767
J.W. Sleijster | 0856823

Project managers
M. Brunings
L.D. Stooker

Project supervisor
N. Zannone

Customer
M.A. Weffers-Albu
Abstract

This document describes the unit test plan for The Callisto plug-in. Callisto is a plug-in for Jupyter which enables analysis of data from the MIMIC-III database [1]. The MIMIC-III database contains medical information of more than 40,000 critical care patients. With this information, researchers hope to obtain relevant information which can help critical care patients in the future. This document complies with the ESA software standard.
Contents

1 Introduction 6
  1.1 Purpose ................................................................. 6
  1.2 Overview ............................................................ 6
  1.3 Definitions and abbreviations .................................. 7
    1.3.1 Definitions .................................................... 7
    1.3.2 Abbreviations ................................................. 8
  1.4 List of references ................................................. 8

2 Test plan 9
  2.1 Test items ........................................................... 9
  2.2 Features to be tested .............................................. 9
  2.3 Test deliverables .................................................. 9
  2.4 Testing tasks ....................................................... 10
  2.5 Environmental needs ............................................. 10
  2.6 Test case pass/fail criteria ..................................... 10

3 Test case specifications 11
  3.1 Docx conversion .................................................. 11
  3.2 Database engine .................................................. 14
  3.3 Test query builder ................................................. 15

4 Test procedures 18

5 Coverage report 19

6 Test report 21
DOCUMENT STATUS SHEET

GENERAL

Document title: Unit Test Plan v1.0.0
Identification: UTP/1.0.0
Authors: K.W. Man
Document status: Final version

DOCUMENT HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.1</td>
<td>18-10-2016</td>
<td>K.W. Man</td>
<td>Initial document structure</td>
</tr>
<tr>
<td>0.1.0</td>
<td>07-11-2016</td>
<td>G. Foks</td>
<td>First version for feedback</td>
</tr>
<tr>
<td>1.1.0</td>
<td>10-11-2016</td>
<td>G. Foks</td>
<td>Feedback has been implemented</td>
</tr>
</tbody>
</table>
DOCUMENT CHANGE RECORDS

There are no changes yet.
1 INTRODUCTION

1.1 PURPOSE

This is the Unit Test Plan (UTP) for Callisto. Each unit test will be runned. The results of the tests should be as described in this document.

1.2 OVERVIEW

This document contains the unit tests procedures, the actual unit tests and the test reports with the encountered problems. Chapter 2 describes the items that need to be tested and the criteria for each unit test. Chapter 3 specifies the unit tests, this includes the input and the output. Chapter 4 lists the test procedure. Finally, chapter 5 and 6 describes the results.
1.3 DEFINITIONS AND ABBREVIATIONS

1.3.1 DEFINITIONS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Callisto Cell</td>
<td>A Jupyter cell which contains the graphical user interface for the creation of the graph and the graph itself. Furthermore it contains options for exporting to Microsoft Word en MATLAB</td>
</tr>
<tr>
<td>Disease class</td>
<td>All diseases which would be in the same category, is called a disease class.</td>
</tr>
<tr>
<td>Jupyter Notebook</td>
<td>A web application that allows the creation of live code, visualisations and text.</td>
</tr>
<tr>
<td>Jupyter Project</td>
<td>A document in the Jupyter Notebook which can contain a set of cells.</td>
</tr>
<tr>
<td>JupyterHub</td>
<td>An application which enables an instance of Jupyter Notebook for every user which logs in.</td>
</tr>
<tr>
<td>MATLAB</td>
<td>A computing environment for multi-paradigm numerical data.</td>
</tr>
<tr>
<td>MIMIC Graph</td>
<td>A graph created with the data of the MIMIC-III database.</td>
</tr>
<tr>
<td>Philips</td>
<td>A company focused on improving people’s lifestyle with meaningful innovations in healthcare, consumers lifestyles and lightning.</td>
</tr>
<tr>
<td>Python</td>
<td>Python is a widely used high-level, general-purpose, interpreted, dynamic programming language.</td>
</tr>
<tr>
<td>Top 6 disease classes</td>
<td>Cardiovascular disease, chronic lung disease, chronic kidney disease, cancer, diabetes, injuries.</td>
</tr>
</tbody>
</table>
1.3.2 Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>TU/e</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td>URD</td>
<td>User Requirements Document</td>
</tr>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>ADD</td>
<td>Architecture Design Document</td>
</tr>
<tr>
<td>ATP</td>
<td>Acceptance Test Plan</td>
</tr>
<tr>
<td>AT</td>
<td>Acceptance Test</td>
</tr>
<tr>
<td>SVVP</td>
<td>Software Verification and Validation Plan</td>
</tr>
<tr>
<td>ICU</td>
<td>Intensive Care Unit</td>
</tr>
</tbody>
</table>

1.4 List of References


2 TEST PLAN

2.1 TEST ITEMS

The software to be tested is Callisto. Callisto consist of a costum jupyter cell, a querying module and a save as docx module. Information about the detailed design can be found in the DDD [4]

2.2 FEATURES TO BE TESTED

Callisto mainly consist of GUI elements, therefore there are not many unit tests. GUI elements are tested during the acceptence tests. The features that are tested are the docx conversion functions, the query builder functions and the database engine.

2.3 TEST DELIVERABLES

Before testing starts, the following documents must be delivered:

- DDD [4].
- UTP (this document).

After completing the tests the following documents must be delivered:

- Unit test output data.
- Problems reports (if any).

2.4 TESTING TASKS

Before testing the following tasks need to be done:

- Designing the unit tests.
- Define the input data for the unit tests.
- Ensuring that all environmental needs for the unit tests are satisfied.

When the above tasks are done, the unit test can be performed according to the procedures described in chapter 4.
2.5 ENVIRONMENTAL NEEDS

The following resources are nescaserry for testing:

- Testing framework nose. Which can be installed using pip.
  - 'pip install nose'

- For test coverage the module coverage needs to be installed.
  - 'pip install coverage'

- A server with the latest version of jupyter notebook and Callisot installed.

- A computer with the lastest version of the tests.

- Connection to the server

2.6 TEST CASE PASS/FAIL CRITERIA

Every test contains a description when that test is passed. The overall unit tests pass is only passed when all unit tests are passed.
3 TEST CASE SPECIFICATIONS

Below are the descriptions for the unit tests. The tests are described with the following structure:

- Test case identifier
  An unique identifier for the test case.

- Test items.
  A description of the functions that are tested.

- Input specifications
  A specification of the input for the functions that are tested.

- Output specifications
  The expected output of the functions.

- Environmental needs
  A description of any environmental needs that are important for this test case.

3.1 DOCX CONVERSION

It is hard to automate the tests for docx conversion because there needs to be a check if the outputted documents are properly formatted and contain all the information needed. This is thus tested by hand.

UT0

Test items: filter_callisto_cells(cells)
Check if all the cells source is deleted from the json

Input specifications:
An array containing cells. The cells have the same structure as when they are stored in an .ipynb file. The array contains one Callisto cell.

Output specifications:
An array containing the one original cell. Only now the value of cell.source is empty.

Environmental needs:
None.

UT1

Test items: filter_callisto_cells(cells)
Check if filter_callisto_cells keeps only the cells are checked for export. Input specifications:
An array of cells from a .ipynb file. This ipynb file contained three cells in the following order: a callisto cell without the export checkmark checked, a code cell, a callisto cell with the export checkmark checked.

**Output specifications:**
An array containing only one cell: the cell with the export chechmark checked

**Environmental needs:**
None.

---

**UT2**

**Test items:** `can_filter_cell(cell)`
Gives a callisto cell and checks if it returns 'true'.

**Input specifications:**
`cell = {'metadata': {'callisto': 'true'}}`

**Output specifications:**
A boolean with the value 'true'

**Environmental needs:**
None.

---

**UT3**

**Test items:** `can_filter_cell(cell)`
Gives an empty cell and checks if it returns 'true'.

**Input specifications:**
`cell = {'metadata': {}}`

**Output specifications:**
A boolean with the value 'true'

**Environmental needs:**
None.

---

**UT4**

**Test items:** `cell_export_filter(cell)`
Gives a cell that should not be exported and checks if it returns false.

**Input specifications:**
`cell = {'metadata': {'isExported': 'false'}}`

**Output specifications:**
A boolean with the value 'false'.

**Environmental needs:**
None.
None.

**UT5**

**Test items:** cell_export_filter(cell)
Gives a cell that should be exported and checks if it returns 'true'.

**Input specifications:**
cell = {'metadata': {'isExported': 'true'}}

**Output specifications:**
A boolean with the value 'true'.

**Environmental needs:**
None.

**UT6**

**Test items:** cell_export_filter(cell)
Gives an empty cell and checks if it returns 'true'.

**Input specifications:**
cell = {'metadata': {}}

**Output specifications:**
A boolean with the value 'true'.

**Environmental needs:**
None.

**UT7**

**Test items:** attribute_value_to_boolean(value, default)
Checks if it asserts to false when given a string 'False' and a default of 'true'.

**Input specifications:**
A string 'False', and a boolean 'true'.

**Output specifications:**
A boolean with the value 'false'

**Environmental needs:**
None.

**UT8**

**Test items:** attribute_value_to_boolean(value, default)
Checks if it asserts to false when given a string 'True' and a default of 'true'.


Input specifications:
A string 'True', and a boolean 'true'

Output specifications:
A boolean with the value 'true'

Environmental needs:
None.

UT9

Test items: attribute_value_to_boolean(value, default)
Checks if it asserts to default when given a 'None' value and a default of 'true'.

Input specifications:
A 'None' value, and a boolean 'true'.

Output specifications:
A boolean with the value 'true'

Environmental needs:
None.

3.2 DATABASE ENGINE

UT10

Test items: engine.execute_raw(query)
Tests if the mimic data engine can correctly execute a select statement on an existent table in the MIMIC-III database.

Input specifications:
The following query: 'SELECT * FROM mimiciii.ADMISSIONS'

Output specifications:
A response that is not empty, or an error.

Environmental needs:
The user should have access to the database specified in the config.py file.

UT11

Test items: engine.execute_raw(query)
Tests if the mimic data engine throws 'NoSuchTableError' for a select statement on a non-existent table in the MIMIC-III database.

Input specifications:
The following query: 'SELECT * FROM TABLETHATDOESNOTEXIST'.

**Output specifications:**
An NoSuchTable error.

**Environmental needs:**
The user should have access to the database specified in the config.py file.

### 3.3 TEST QUERY BUILDER

#### UT12

**Test items:** QueryBuilder._build_filters_conditions(filters, filter_objects)
Checks that it returns a correctly formed string that consists of all the query strings from all the filter objects.

**Input specifications:**
4 filter objects containing the key "value" for which the value is 'value.'
4 dummy filter objects containing the key "make_query". This key is used to build the query strings from the filter objects.

**Output specifications:**
The following string: 'value AND value AND value AND value'.

**Environmental needs:**
None.

#### UT13

**Test items:** QueryBuilder._get_unique_tables(filter_objects)
Tests QueryBuilder._get_unique_tables and checks that it returns a set of unique tables given different capitalizations of the same tables.

**Input specifications:**
Filter objects containing the following tables in the options: "ADMISSIONS", "PaTiEnTs", "CHARTevents", "chartEVENTS", "admisslONS", "patients".

**Output specifications:**
A set containing only the following strings: "admissions", "patients", "chartevents".

**Environmental needs:**
None.

#### UT14

**Test items:** QueryBuilder.build(presets, filters)
Tests if the function returns an error on an invalid preset

**Input specifications:**
An empty preset. Four dummy filters.

**Output specifications:**
an error stating: "QueryBuilder.build preset is not in available presets"

**Environmental needs:**
None.

---

**UT15**

**Test items:**QueryBuilder.build(presets, filters)

Tests if the function returns an error when given empty filters.

**Input specifications:**
preset = 'corr_anal_age_v_los'. filters = None

**Output specifications:**
an error stating: "QueryBuilder.build given None filters. Pass an empty filter dictionary for an unfiltered query."

**Environmental needs:**
None.

---

**UT16**

**Test items:**QueryBuilder.build(presets, filters)

Tests if the query builder returns an correct query.

**Input specifications:**
preset = 'corr_anal_age_v_los'. filters = { }

**Output specifications:**
The following string:

```sql
SELECT CASE WHEN CAST(EXTRACT(YEAR FROM AGE(ADMISSIONS.ADMITTIME, PATIENTS.DOB)) AS INT) > 89 then '300'
when CAST(EXTRACT(YEAR FROM AGE(ADMISSIONS.ADMITTIME, PATIENTS.DOB)) AS INT) <= 89 then CAST(EXTRACT(YEAR FROM AGE(ADMISSIONS.ADMITTIME, PATIENTS.DOB)) AS INT) END AS X,
CAST(EXTRACT(DAYS FROM AVG(ADMISSIONS.DISCHTIME - ADMISSIONS.ADMITTIME)) AS INT) AS Y
FROM ADMISSIONS
INNER JOIN PATIENTS ON ADMISSIONS.SUBJECT_ID = PATIENTS.SUBJECT_ID
```

---

TU/e
AND ADMISSIONS.hadm_id IN
(SELECT hadm_id FROM admissions GROUP BY hadm_id)
GROUP BY X
ORDER BY X ASC

Environmental needs:
None.
4 TEST PROCEDURES

Running the unit tests is quite easy. It consist of the following steps:

1. Open a terminal.

2. Go to the root folder of Callisto.

3. Run the following command: "./script/test". This will run all the tests that are in the project.

4. The results are directly shown in the terminal. The test coverage is saved in an html file in the folder cover and detailed test results are stored in a xml file. To see the test coverage open cover/index.html. To see a detailed test results in xml format open nosetests.xml
# Coverage Report

For the coverage report a screenshot of the index.html is provided.

**Coverage report: 50%**

<table>
<thead>
<tr>
<th>Module Path</th>
<th>statements</th>
<th>missing</th>
<th>excluded</th>
<th>coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>src.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto.py</td>
<td>24</td>
<td>11</td>
<td>0</td>
<td>54%</td>
</tr>
<tr>
<td>src/callisto/bundlerextensions.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/bundlerextensions/docx.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/bundlerextensions/docx/filter_functions.py</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/config.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/config/filters.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/config/presets.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/data.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/data/exceptions.py</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/data/query.py</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/filtering.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/filtering/query_builder.py</td>
<td>56</td>
<td>5</td>
<td>0</td>
<td>91%</td>
</tr>
<tr>
<td>src/callisto/graphs.py</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/graphs/bars.py</td>
<td>28</td>
<td>21</td>
<td>0</td>
<td>25%</td>
</tr>
<tr>
<td>src/callisto/graphs/graphs.py</td>
<td>71</td>
<td>44</td>
<td>0</td>
<td>38%</td>
</tr>
<tr>
<td>src/callisto/graphs/pie.py</td>
<td>19</td>
<td>14</td>
<td>0</td>
<td>26%</td>
</tr>
<tr>
<td>src/callisto/graphs/scatter.py</td>
<td>99</td>
<td>86</td>
<td>0</td>
<td>13%</td>
</tr>
<tr>
<td>src/callisto/graphs/threebar.py</td>
<td>37</td>
<td>32</td>
<td>0</td>
<td>14%</td>
</tr>
<tr>
<td>src/callisto/graphs/types.py</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/bar2d.py</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>50%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/corranal.py</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>31%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/line2d.py</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/line3d.py</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/pie2d.py</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/scatter2d.py</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>src/callisto/graphs/types/scatter3d.py</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>src/callisto/nbextensions.py</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>src/callisto/settings.py</td>
<td>42</td>
<td>5</td>
<td>0</td>
<td>88%</td>
</tr>
</tbody>
</table>

**Total** 492 245 0 50%

The coverage report contains some files that are not tested which still has a percentage of coverage. For example src/callisto/bars.py gets 25% coverage, this is because when a python file gets imported it runs through the class and function definition once, and hence get covered. Also lines containing only strings are not considered code. Files that are imported and containing a high string/function ratio therefore get a relative high percentage of coverage. This per-
percentage does not indicate that the actual functions in those files are tested during the unittests.
6 TEST REPORT

For the test result a screenshot with the test results is provided. The test results are in xml format.

```xml
<testsuite name="novetests" tests="18" errors="0" failures="0" skip="0">
    <testcase classname="tests.test_database.DatabaseTestCase" name="test_incorrect_table" time="0.859"/>
    <testcase classname="tests.test_database.DatabaseTestCase" name="test_read_table" time="27.103"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_attribute_value_to_boolean_default" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_attribute_value_to_boolean_false" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_attribute_value_to_boolean_true" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_can_filter_cell" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_cannot_filter_cell" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_cell_export_filter_empty" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_cell_export_filter_false" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_cell_export_filter_true" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_filter_calisto_cells" time="0.000"/>
    <testcase classname="tests.test_docx.TestFilterFunctions" name="test_filter_calisto_cells_with_non_exported_cells" time="0.000"/>
    <testcase classname="tests.test_query_builder.QueryBuilderTestCase" name="test_build_conditions" time="0.000"/>
    <testcase classname="tests.test_query_builder.QueryBuilderTestCase" name="test_build_inner_join" time="0.000"/>
    <testcase classname="tests.test_query_builder.QueryBuilderTestCase" name="test_invalid_preset" time="0.000"/>
    <testcase classname="tests.test_query_builder.QueryBuilderTestCase" name="test_no_filters" time="0.000"/>
</testsuite>
```