Horus
IMSETY
Unit Test Plan
Version 0.3  21st June 2007

Project team:  Jeroen Keiren  0569081
               Frank Koenders  0575629
               Thijs Nugteren  0574426
               Joeri de Ruiter  0578312
               Stijn Stiefelhagen  0579816
               Carst Tankink  0569954
               Pim Vullers  0575766
               Freck van Walderveen  0566348

Project manager:  Egbert Teeselink

Senior management:  L. Somers  TU/e (HG 7.83)
                    M. v.d. Brand  TU/e (HG 7.44)
Adviser:  R.J. Bril  TU/e (HG 5.09)
Customer:  E. v. Breukelen  ISIS

Computer Science, Eindhoven University of Technology, Eindhoven
Abstract

This document provides the main guidance for the Unit Tests (UT) during the Detailed Design (DD) phase of the IMSETY project. It describes the environment needed to perform the UT. When this environment is set up, all test cases must be executed according to their corresponding test procedures. After a test has been performed a report needs to be written.
Contents

1 Introduction ......................................................... 1
  1.1 Purpose ....................................................... 1
  1.2 Overview ..................................................... 1
  1.3 List of definitions .......................................... 1
  1.4 List of references ........................................... 1

2 Test plan .......................................................... 2
  2.1 Test items ..................................................... 2
  2.2 Features to be tested ....................................... 2
  2.3 Test deliverables ............................................ 2
  2.4 Testing tasks ................................................ 2
  2.5 Environmental needs ....................................... 3
  2.6 Test case pass/fail criteria ................................. 3

3 Test case specifications ........................................... 4

4 Test procedures ................................................... 6
  4.1 Client .......................................................... 6
  4.2 Server ........................................................ 6

5 Test reports ......................................................... 8
  5.1 Booking test .................................................. 8
  5.2 Account test ................................................. 8
  5.3 Command test ................................................. 8
  5.4 Data test ...................................................... 9
  5.5 Log test ....................................................... 9
  5.6 Payload test .................................................. 9
  5.7 Satellite test .................................................. 9
  5.8 Experiment test ............................................... 9
  5.9 Database abstraction test ................................... 10
  5.10 Controller test ............................................... 14
  5.11 Queue test ................................................... 14
  5.12 Scheduler test ............................................... 15
# Document status sheet

<table>
<thead>
<tr>
<th>Document title</th>
<th>Unit Test Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document identifier</td>
<td>IMSETY/doc/UTP</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Jeroen Keiren, Pim Vullers</td>
</tr>
<tr>
<td>Version</td>
<td>0.3</td>
</tr>
<tr>
<td>Document status</td>
<td>Internally approved</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 (Revision 1294)</td>
<td>21-05-2007</td>
<td>Jeroen Keiren, Pim Vullers</td>
<td>Up for review</td>
</tr>
<tr>
<td>0.2 (Revision 1329)</td>
<td>22-05-2007</td>
<td>Jeroen Keiren, Pim Vullers</td>
<td>Fixed review results</td>
</tr>
<tr>
<td>0.3 (Revision 1998)</td>
<td>21-06-2007</td>
<td>Jeroen Keiren, Pim Vullers</td>
<td>Added test report</td>
</tr>
</tbody>
</table>
Document change report

<table>
<thead>
<tr>
<th>Document title</th>
<th>Unit Test Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document identifier</td>
<td>IMSETY/doc/UTP</td>
</tr>
<tr>
<td>Date of changes</td>
<td>22-05-2007</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section number</th>
<th>Reason for change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Almost all</td>
<td>Minor changes according to review remarks.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Purpose

This document describes the plan for testing the developed software units against the detailed design as defined in the DDD [2]. The purpose of these unit tests is to make sure that IMSETY’s components comply with the design as described in the DDD [2]. These tests should be executed during the Detailed Design (DD) phase of the IMSETY project as described in the ESA software engineering standard [1].

1.2 Overview

Chapter 2 mentions the items to be tested together with the general criteria for the UT. A specification for each test case is given in chapter 3. The procedures for these test cases are explained in chapter 4. In chapter 5 the reports for all test cases are presented.

1.3 List of definitions

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD</td>
<td>Detailed Design</td>
</tr>
<tr>
<td>DDD</td>
<td>Detailed Design Document</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>SVVP</td>
<td>Software Verification and Validation Plan</td>
</tr>
<tr>
<td>UT</td>
<td>Unit Test</td>
</tr>
<tr>
<td>UTP</td>
<td>Unit Test Plan</td>
</tr>
</tbody>
</table>

1.4 List of references

Chapter 2

Test plan

2.1 Test items

The software to be tested are IMSETY’s units. Information about the detailed design of this system can be found in the DDD [2].

2.2 Features to be tested

IMSETY must meet the design as stated in the DDD [2]. Each component should adhere to the interfaces given in the DDD [2].

2.3 Test deliverables

Before the testing starts the following items must be delivered:

- SVVP [3].
- DDD [2].
- UTP [4], chapters 1, 2, 3 and 4.
- UT input data.
- IMSETY’s units.

After completing the testing the following items must be delivered:

- UT report (UTP chapter 5).
- UT output data.
- Problem reports (if necessary).

2.4 Testing tasks

Before any testing in the UT phase can take place, the following tasks need to be done:

- Designing the unit tests.
- Tracing all test cases to components.
- All components mentioned in the DDD [2] need to be covered by test cases.
 CHAPTER 2. TEST PLAN

- Creation of UT input data.
- Ensuring that all environmental needs are satisfied for the UT.

When these tasks have been done a UT can be performed according to the procedures described in chapter 4.

2.5 Environmental needs

To be able to perform the UT the following resources are needed:

For the server:
- A computer running Linux with CppUnit, XML-RPC and MySQL++ installed.
- A MCC (-stub) available local or remote to the above described machine.
- A MCS (-stub) available local or remote to the above described machine.

For the client:
- A computer running Windows 2000/XP/Vista, Linux or OpenSolaris.
- Qt4 available on the local machine.

See also the constraints described in the DDD [2].

2.6 Test case pass/fail criteria

Every test should describe the criteria that should be met to pass a specific test. An overall UT pass can only be achieved when all tests described in chapter 3 have been performed and passed.
Chapter 3

Test case specifications

The test case specifications for the server, client and common packages can be found in the source code repository in respectively src/server/test, src/client/test and src/common/test. These test cases are documented there according to the format as described further on in this section, and will not be discussed any further.

Listing 3.1: example_test.h

```c++
/*!
 * \file example_test.h
 * \author Pim Vullers, Horus
 * Unit test for class Example.
 */

#ifndef EXAMPLE_TEST_H
#define EXAMPLE_TEST_H

#include <cppunitTestFixture.h>
#include <cppunit/extensions/HelperMacros.h>
#include "example.h"

// (1) Test description:
/*!
 * \brief Unit test for class Example. */
 class example_test : public CppUnit::TestFixture {

 // (2) Test specification:
 // (3) Unique test identifier:
 CPPUNIT_TEST_SUITE(example_test);
 // (4) Cases to be tested:
 CPPUNIT_TEST(constructorTest);
 CPPUNIT_TEST(methodTest);
 CPPUNIT_TEST_SUITE_END();

 // (5) Test environment construction and destruction:
 public:
 /*!
 * \brief Sets up test objects. (Environment construction) */
 void setUp();

 /*!
 * \brief Destroys test objects. (Environment destruction) */
 void tearDown();
```
CHAPTER 3. TEST CASE SPECIFICATIONS

// (6) Test functions with their descriptions:
protected:
    /*! \brief Tests whether the satellite constructors confirm to their specifications */
    void constructorTest();
    /*! \brief Tests whether the method function confirms to the specification */
    void methodTest();

// (7) Test environment variables:
private:
    /*! \brief Samples to perform the tests on */
    Example ex1, ex2;
    /*! \brief Other variables needed to test Example class */
    OtherClass oc;

#endif // EXAMPLE_TEST.H

The format for the test case documentation will be as shown in listing 3.1. This means that each test case should have a description (1) and specification (2). The specification should contain a unique identifier (3) (this will be enforced by the compiler), the name of the test class. This class should start with the CppUnit macros [5] defining the cases to be tested (4) in this testsuite. The functions mentioned in these macros should be documented (6), giving which functionality of the unit will be tested in this function. The environment for the tests is given using the private fields of the header file (7) and the contents of the setUp and tearDown functions (5).
Chapter 4

Test procedures

In order to be able to automate the unit testing of IMSETY the author of a component is also responsible for writing the specific unit test belonging to that component. The unit tests need to be placed in the directories described further on in this chapter (either in the Client or the Server section). Note that testing components from the src/common directory will be carried out conform the test framework of the server.

By having the author of a component also write the test belonging to the component, it should never occur that a component goes into the source code repository untested. This means that the author is also responsible for carrying out the tests before committing code.

Carrying out tests is different for the client and the server because at the client side we also need to be able to test the graphical user interface (GUI). Therefore we will also describe the test procedures for the client and the server separately.

4.1 Client

On the client side, for each test that needs to be run there is a .pro file in the src/client/test/ directory, along with a .cpp file which implements a specific test. These tests cannot be easily aggregated in a single test suite because of shortcomings of the Qt test framework (if we would aggregate the tests in one test suite, a crash of one test would lead to the rest of the suite not being executed). In order to overcome this we have added a shell script which runs all individual tests, as the tests won’t run from within the build system.

Note that all test names (denoted with testname) should be name_test, as this is needed for running the batch of tests.

Individual client tests can be run by going to the src/client/test/ directory and there executing:

```
# scons testname
# ./testname
```

A batch with all tests can be run by executing src/client/test/unittest.sh.

4.2 Server

Carrying out unit tests for the server is a lot simpler than that of the client. For carrying out server unit tests we use the CppUnit [5] test framework. Whenever we mention the server, this also applies to the common files, substituting server by common.

For each test that needs to be run against the server, there is a name_test.{h, cpp} file in the src/server/test directory.

Individual server tests can be run by going to the src/server or the src/server/test directory and running:
# scons unittest

When only a single test needs to be run this can be done by going to the `src/server/test` directory and running:

# scons name_test
Chapter 5

Test reports

When a test report for a unit test is desired, for all packages a test report can be generated from the test run. In case of the server and common files a test report is always automatically generated respectively in the src/server/test/report or the src/common/test/report folder. For the client a test report is always written to the screen. This can be overcome by redirecting standard output to a file by using the --report option, like src/client/test/test.sh --report.

Due to lack of time during the coding phase of the project not all automated unit tests have been updated and run as much as should be done. Therefore we present here the latest available test report, which still contains some errors and does not test the full functionality.

5.1 Booking test

booking_test::constructorTest : OK
booking_test::directionTest : OK
booking_test::typeTest : OK
booking_test::statusTest : OK
booking_test::startTimeTest : OK
booking_test::endTimeTest : OK
booking_test::experimentIDTest : OK
booking_test::getFirstBookingTest : OK
booking_test::gssIDTest : OK

5.2 Account test

account_test::constructorTest : OK
account_test::passwordTest : OK
account_test::typeTest : OK
account_test::registerTest : OK
account_test::fullNameTest : OK
account_test::emailTest : OK

5.3 Command test

command_test::constructorTest : OK
command_test::payloadIDTest : OK
command_test::parametersTest : OK
command_test::nameTest : OK
CHAPTER 5. TEST REPORTS

5.4 Data test

data_test::constructorTest : OK
data_test::nameTest : OK
data_test::experimentTest : OK
data_test::gatheredAtTest : OK
data_test::dataItemTest : OK
data_test::thumbnailTest : OK

5.5 Log test

log_test::constructorTest : OK

5.6 Payload test

payload_test::constructorTest : OK
payload_test::satelliteIDTest : OK
payload_test::nameTest : OK
payload_test::registerTest : OK
payload_test::getExperimentsTest : OK
payload_test::getListOfCommandsTest : OK

5.7 Satellite test

satellite_test::constructorTest : OK
satellite_test::mccUrlTest : OK
satellite_test::mcsUrlTest : OK
satellite_test::nameTest : OK
satellite_test::registerTest : OK

5.8 Experiment test

experiment_test::constructorTest : assertion
experiment_test::nameTest : OK
experiment_test::payloadTest : OK
experiment_test::scheduleTest : assertion
experiment_test::statusTest : OK
experiment_test::bookingTest : OK

experiment_test.cpp:23:Assertion
Test name: experiment_test::constructorTest
assertion failed
- Expression: exp_empty.getSchedule() == std::make_pair(-1,-1)

experiment_test.cpp:72:Assertion
Test name: experiment_test::scheduleTest
assertion failed
- Expression: std::make_pair(newScheduledFrom, newScheduledTo) == exp_empty.getSchedule()
5.9 Database abstraction test

databaseabstraction_test::getAllSatelliteIDsTest : assertion
databaseabstraction_test::retrieveSatelliteTest : error
databaseabstraction_test::updateSatelliteTest : error
databaseabstraction_test::getExperimentAtTimeTest : assertion
databaseabstraction_test::getAllPayloadIDsTest : assertion
databaseabstraction_test::retrievePayloadTest : assertion
databaseabstraction_test::updatePayloadTest : assertion
databaseabstraction_test::getPayloadExperimentsTest : assertion
databaseabstraction_test::getPayloadListOfCommandsTest : assertion
databaseabstraction_test::getInternalBookingTest : assertion
databaseabstraction_test::getTimeBookingIDsTest : assertion
databaseabstraction_test::getAvailableSatellitesTest : assertion
databaseabstraction_test::getExternalBookingsOfExperimentTest : assertion
databaseabstraction_test::retrieveBookingTest : assertion
databaseabstraction_test::updateBookingTest : assertion
databaseabstraction_test::getFirstBookingTest : assertion
databaseabstraction_test::accountHasRightsToSatelliteTest : assertion
databaseabstraction_test::accountHasRightsToPayloadTest : assertion
databaseabstraction_test::accountHasRightsToExperimentTest : OK
databaseabstraction_test::accountAddPayloadTest : error
databaseabstraction_test::accountRemovePayloadTest : error
databaseabstraction_test::accountAddExperimentTest : error
databaseabstraction_test::accountRemoveExperimentTest : error
databaseabstraction_test::accountExistsTest : OK
databaseabstraction_test::retrieveAccountTest : assertion
databaseabstraction_test::updateAccountTest : OK
databaseabstraction_test::getAllCommandIDsTest : assertion
databaseabstraction_test::retrieveCommandTest : assertion
databaseabstraction_test::updateCommandTest : assertion
databaseabstraction_test::getAllExperimentIDsTest : assertion
databaseabstraction_test::retrieveExperimentTest : assertion
databaseabstraction_test::getDataListTest : assertion
databaseabstraction_test::getThumbnailListTest : assertion
databaseabstraction_test::removeExperimentTest : OK
databaseabstraction_test::getAllDataIDsTest : assertion
databaseabstraction_test::retrieveDataTest : OK
databaseabstraction_test::updateDataTest : OK
databaseabstraction_test::retrieveQueueItemTest : error
databaseabstraction_test::updateQueueItemTest : error
databaseabstraction_test::removeQueueItemTest : OK
databaseabstraction_test::getQueueTest : assertion
databaseabstraction_test::removeQueueTest : OK
databaseabstraction_test::retrieveLogTest : error
databaseabstraction_test::addLogTest : assertion
databaseabstraction_test::retrieveLogsTest : assertion

databaseabstraction_test.cpp:38:Assertion
Test name: databaseabstraction_test::getAllSatelliteIDsTest
assertion failed
## Failure Location unknown## : Error
Test name: databaseabstraction_test::retrieveSatelliteTest 
uncaught exception of type DatabaseAbstraction::DatabaseException 
- Identifier not in database.

## Failure Location unknown## : Error
Test name: databaseabstraction_test::updateSatelliteTest 
uncaught exception of type DatabaseAbstraction::DatabaseException 
- Identifier not in database.

databaseabstraction_test.cpp:110:Assertion 
Test name: databaseabstraction_test::getExperimentAtTimeTest 
assertion failed 
- Expression: experimentID == 10

databaseabstraction_test.h:336:Assertion 
Test name: databaseabstraction_test::getAllPayloadIDsTest 
assertion failed 
- Expression: check.size() == s.size()

databaseabstraction_test.cpp:148:Assertion 
Test name: databaseabstraction_test::retrievePayloadTest 
assertion failed 
- Expression: name == "payload4"

databaseabstraction_test.cpp:181:Assertion 
Test name: databaseabstraction_test::updatePayloadTest 
expected exception not thrown 
- Expected: DatabaseAbstraction::DatabaseException

databaseabstraction_test.h:336:Assertion 
Test name: databaseabstraction_test::getPayloadExperimentsTest 
assertion failed 
- Expression: check.size() == s.size()

databaseabstraction_test.cpp:221:Assertion 
Test name: databaseabstraction_test::getPayloadListOfCommandsTest 
assertion failed 
- Expression: check.size() == s.size()

databaseabstraction_test.cpp:234:Assertion 
Test name: databaseabstraction_test::getInternalBookingTest 
assertion failed 
- Expression: (BookingID)16 == id

databaseabstraction_test.h:336:Assertion 
Test name: databaseabstraction_test::getTimeBookingIDsTest 
assertion failed 
- Expression: check.size() == s.size()
Test name: databaseabstraction_test::getAvailableSatellitesTest
assertion failed
  - Expression: check.size() == s.size()

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getAllBookingIDsTest
assertion failed
  - Expression: check.size() == s.size()

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getExternalBookingsOfExperimentTest
assertion failed
  - Expression: check.size() == s.size()

databaseabstraction_test.cpp:327:Assertion
Test name: databaseabstraction_test::retrieveBookingTest
assertion failed
  - Expression: type == Internal

databaseabstraction_test.cpp:375:Assertion
Test name: databaseabstraction_test::updateBookingTest
expected exception not thrown
  - Expected: DatabaseAbstraction::DatabaseException

databaseabstraction_test.cpp:393:Assertion
Test name: databaseabstraction_test::getFirstBookingTest
assertion failed
  - Expression: id == 16

databaseabstraction_test.cpp:407:Assertion
Test name: databaseabstraction_test::accountHasRightsToSatelliteTest
assertion failed
  - Expression: result == true

databaseabstraction_test.cpp:433:Assertion
Test name: databaseabstraction_test::accountHasRightsToPayloadTest
assertion failed
  - Expression: result == true

##Failure Location unknown## : Error
Test name: databaseabstraction_test::accountAddPayloadTest
uncaught exception of type DatabaseAbstraction::DatabaseException
  - Invalid query result

##Failure Location unknown## : Error
Test name: databaseabstraction_test::accountRemovePayloadTest
uncaught exception of type DatabaseAbstraction::DatabaseException
  - Precondition failed, registered != true

##Failure Location unknown## : Error
Test name: databaseabstraction_test::accountAddExperimentTest
uncaught exception of type DatabaseAbstraction::DatabaseException
  - Precondition failed, registered != true
CHAPTER 5. TEST REPORTS

##Failure Location unknown## : Error
Test name: databaseabstraction_test::accountRemoveExperimentTest
uncaught exception of type DatabaseAbstraction::DatabaseException
- Precondition failed, registered != true

databaseabstraction_test.cpp:589:Assertion
Test name: databaseabstraction_test::retrieveAccountTest
assertion failed
- Expression: type == Administrator

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getAllCommandIDsTest
assertion failed
- Expression: check.size() == s.size()

databaseabstraction_test.cpp:717:Assertion
Test name: databaseabstraction_test::retrieveCommandTest
assertion failed
- Expression: commandParam.minValue.i == 0

databaseabstraction_test.cpp:685:Assertion
Test name: databaseabstraction_test::updateCommandTest
expected exception not thrown
- Expected: DatabaseAbstraction::DatabaseException

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getAllExperimentIDsTest
assertion failed
- Expression: check.size() == s.size()

databaseabstraction_test.cpp:771:Assertion
Test name: databaseabstraction_test::retrieveExperimentTest
assertion failed
- Expression: name == "experiment8"

##Failure Location unknown## : Error
Test name: databaseabstraction_test::updateExperimentTest
uncaught exception of type DatabaseAbstraction::DatabaseException
- Command with given paramType not in database

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getDataListTest
assertion failed
- Expression: check.size() == s.size()

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getThumbnailListTest
assertion failed
- Expression: check.size() == s.size()

databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::getAllDataIDsTest
assertion failed
- Expression: check.size() == s.size()

```cpp
##Failure Location unknown## : Error
Test name: databaseabstraction_test::retrieveQueueItemTest
uncaught exception of type DatabaseAbstraction::DatabaseException
- Identifier not in database.
```

```cpp
##Failure Location unknown## : Error
Test name: databaseabstraction_test::updateQueueItemTest
uncaught exception of type DatabaseAbstraction::DatabaseException
- Identifier not in database.
```

```cpp
databaseabstraction_test.cpp:1048:Assertion
Test name: databaseabstraction_test::getQueueTest
assertion failed
- Expression: s.size() == check.size()
```

```cpp
##Failure Location unknown## : Error
Test name: databaseabstraction_test::retrieveLogTest
uncaught exception of type DatabaseAbstraction::DatabaseException
- Identifier not in database.
```

```cpp
databaseabstraction_test.cpp:1138:Assertion
Test name: databaseabstraction_test::addLogTest
assertion failed
- Expression: check.size() == 1
```

```cpp
databaseabstraction_test.h:336:Assertion
Test name: databaseabstraction_test::retrieveLogsTest
assertion failed
- Expression: check.size() == s.size()
```

### 5.10 Controller test

controller_test::constructorTest : OK

### 5.11 Queue test

queue_test::constructorTest : error
queue_test::queueItemTest : error

```cpp
##Failure Location unknown## : Error
Test name: queue_test::constructorTest
setUp() failed
- uncaught exception of type std::logic_error
- basic_string::_S_construct NULL not valid
```

```cpp
##Failure Location unknown## : Error
Test name: queue_test::queueItemTest
setUp() failed
- uncaught exception of type std::logic_error
- basic_string::_S_construct NULL not valid
```
5.12 Scheduler test

scheduler_test::createBooking : OK
scheduler_test::uploadExperimentTest : OK