



**Applications in Practical High-End Computing - Group Project**

Assignment - "Workflow"

**Test planning**

Supervisor: Dr Stuart Barnes

Authors: Mateusz Gołąb

Csaba Kerti

Jakub Kiełbasa

Zsolt Kollarits

Course: CSTE / SETC

**Table of contents**

[1 Summary 3](#_Toc318802356)

[2 Requirement test 3](#_Toc318802357)

[2.1 Non-functional requirements 3](#_Toc318802358)

[3 Preparing Acceptance Test 3](#_Toc318802359)

[3.1 Trace-ability Matrix for use-cases 3](#_Toc318802360)

[3.2 Test cases 5](#_Toc318802361)

[3.3 Trace-ability Matrix for test-cases 8](#_Toc318802362)

[4 Unit test 9](#_Toc318802363)

[5 Integration Test 9](#_Toc318802364)

[6 References 9](#_Toc318802365)

# Summary

All software probably contains errors. To reduce errors and faults, we tried to use software driven development, and execute test operation after every steps of the software development project.

# Requirement test

It is very important to find faults and misunderstandings at the early stage of a software development project as soon as possible. Furthermore, many projects fail because of the misinterpreted requirements. So it is utmost important to clarify the requirements with the customer.

The [1] web page gives many good ideas how to check and review the requirements of a software project. There is no standard method to execute this test. It is usually a review, made by the stakeholders and the customer should accept it if there is a change.

We checked the consistency of the user requirements, and are they clear enough. It should be clear for a designer or a programmer, who read it at the first time, and doesn’t know the aims of the project. We had many assumptions at the beginning of the project, and we tried to discover them day by day. We contacted with the customer and clarified these assumptions. We read and reviewed the given document many times to clarify missing and hidden requirements.

## Non-functional requirements

The non-functional requirements are very important too. Every non-functional requirement should contain concrete values to measure when the test will be executed. We reviewed the first version of the requirements, and we found some generally formulated sentences, which were modified later. The modifications were coordinated by the customer, to quantify the characteristics (e.g.: number of parallel users, supported operating systems).

We planned to execute the following non-functional test operations:

* Compatibility testing (network, operating system)
* Performance testing
* Security testing

# Preparing Acceptance Test

## Trace-ability Matrix for use-cases

The acceptance test should validate the user needs and requirements. It should be executed at the end of the project, but it is recommended to start the preparations at the beginning.

After we had produced the requirements we have created the use-cases using the UML. The Trace-ability Matrix is a good practice to check that is all the functional requirements are covered by use-cases.

We have created a Trace-ability Matrix to ensure that all requirements should have covered by uses-cases. The columns contain the requirements, while the rows contain the uses-cases. When a uses-case connects with a requirement, we sign it with a dot.

The Trace-ability Matrix is very usual tool to follow the requirements changes. By using them, it is easy to see which use-case should be changed.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | FU1 | FU2 | FU3 | FU3.1 | FU3.1 | FU4 | FU5 |
| UC\_U1 Log in |  |  |  |  |  | ● |  |
| UC\_U2 Log out |  |  |  |  |  |  |  |
| UC\_U3 Stop Simulation |  |  |  |  |  |  | ● |
| UC\_S1 Run Simulation |  | ● |  |  |  |  |  |
| UC\_S2 Upload Simulation parameters |  |  |  |  |  |  |  |
| UC\_S3 Download results of a Simulation |  |  |  |  |  |  |  |
| UC\_S4 Check status/log of Simulation |  |  |  | ● |  |  |  |
| UC\_A1: Add new module to Workflow | ● |  |  |  |  |  |  |
| UC\_A2: Remove module from Workflow | ● |  |  |  |  |  |  |
| UC\_A3: Provide input/output metadata (XSD) of a Module |  |  |  |  |  |  |  |
| UC\_A4: Provide XML file with parameters and commands |  |  |  |  |  |  |  |
| UC\_A5: Provides XML file with recovery configuration |  |  |  |  |  |  |  |
| UC\_A6: Create new User |  |  |  |  |  |  |  |
| UC\_A7: Inactivate User |  |  |  |  |  |  |  |
| UC\_M1: Configure recovery mechanism |  |  |  |  | ● |  |  |
| UC\_M2: Create backup |  |  | ● |  |  |  |  |
| UC\_M3: Recover |  |  | ● |  |  |  |  |

1. Table – Trace-ability matrix for use-cases

Where the user requirements are the followings:

* FU1 User can add/remove arbitrary number of modules into workflow.
* FU2 User can run simulation with uploaded parameters.
* FU3 Recovery system: possibility to restart workflow from the last stable/good point when system crushes.
  + FU3.1 Monitoring of errors: Users can see the exact location of failures.
  + FU3.2 Flexible recovery policy (depending of the expected time of execution, we decide to store data before/after a module or after each iteration)
* FU4 Many users have the possibility to connect to the system simultaneously. But there is only one running program at the same time (users requests' go to queue - serial workflow).
* FU5 User can stop simulation.

## Test cases

We created test cases to test the correctness of each operation and to examine how they affect each other. We made many test case variants for each operation to reach better test coverage.

We chose the following form to describe each test case (we will fill it with concrete values)

|  |  |
| --- | --- |
| Title | Content |
| Test Case ID |  |
| Description |  |
| Input |  |
| Steps | 1.  2.  3… |
| Dependencies |  |
| Expected result |  |

Following test cases are connected to general User activities:

|  |  |
| --- | --- |
| Test Case ID | Description |
| TC\_0001 | login with a valid scientist login name and password |
| TC\_0002 | login with a valid administrator login name and password |
| TC\_0003 | try to login with an invalid login name |
| TC\_0004 | try to login with valid login name with invalid password |
| TC\_0005 | logout working properly |
| TC\_0100 | scientist tries to stop own simulation |
| TC\_0101 | scientist tries to stop not own simulation |
| TC\_0102 | administrator tries to stop arbitrary simulation |
| TC\_0103 | user tries to stop an empty workflow |

Test cases connected with scientist activities:

|  |  |
| --- | --- |
| Test Case ID | Description |
| TC\_0200 | scientist tries to start a simulation with appropriate parameters |
| TC\_0201 | scientist tries to start a simulation with invalid parameters |
| TC\_0202 | scientist tries to start a simulation with empty parameters |
| TC\_0203 | scientist tries to start a simulation when the WorkflowQueue empty |
| TC\_0204 | scientist starts a simulation when the WorkflowQueue doesn’t empty |
| TC\_0300 | scientist tries to download the result of the simulation |
| TC\_0301 | scientist tries to download the result when there is no result |
| TC\_0400 | scientist tries to check the status/log of own simulation |
| TC\_0401 | scientist tries to check the status/log of not owned simulation |
| TC\_0402 | administrator tries to check the status/log of any simulation |

Test cases connected with administrator activities:

|  |  |
| --- | --- |
| Test Case ID | Description |
| TC\_0500 | add new module to an empty workflow |
| TC\_0501 | add new module before an existing module |
| TC\_0502 | add new module after an existing module |
| TC\_0503 | insert module to an existing workflow |
| TC\_0504 | add new start module, when the WorflowQueue is not empty |
| TC\_0505 | add new module when a simulation is not running |
| TC\_0506 | add new module when a simulation is running |
| TC\_0600 | remove an alone module from the workflow |
| TC\_0601 | remove module from the middle of the workflow |
| TC\_0602 | remove module from the end of the workflow |
| TC\_0603 | remove module from the beginning of the workflow |
| TC\_0604 | remove module when the simulation is running |
| TC\_0605 | remove module from the beginning when the WorflowQueue is not empty |
| TC\_0700 | provide valid input and valid output metadata of a Module |
| TC\_0701 | provide invalid input and invalid output metadata of a Module |
| TC\_0702 | provide valid input and invalid output metadata of a Module |
| TC\_0703 | provide invalid input and valid output metadata of a Module |
| TC\_0800 | provide XML file with parameters and commands |
| TC\_0801 | provide inappropriate structure of XML with parameters and commands |
| TC\_0900 | provide XML file with recover configuration |
| TC\_0901 | provide inappropriate structure of XML with recover configuration |
| TC\_1000 | try to create an user which is already existing in the system |
| TC\_1001 | try to create an user whit invalid details |
| TC\_1002 | try to create a scientist |
| TC\_1003 | try to create an administrator |
| TC\_1100 | inactivate an administrator |
| TC\_1101 | inactivate a scientist |
| TC\_1102 | inactivate a scientist who have a simulation in the WorkflowQueue |
| TC\_1103 | inactivate a scientist who have a running simulation |
| TC\_1104 | inactivate a user who is online in the system |

Test cases connected with WorflowManager, RecoveryManager and DatabaseManager

|  |  |
| --- | --- |
| Test Case ID | Description |
| TC\_1200 | Workflow/Recovery Manager start to work without configuration (conf.XML) |
| TC\_1201 | Workflow/Recovery Manager start to work with configuration |
| TC\_1300 | Module execution , when the Recovery Manager should create a backup |
| TC\_1301 | Module execution , when the Recovery Manager should not create a backup |
| TC\_1302 | Test that if there was not backup under the iteration, then at the end of the iteration the RecoveryManager should create it |
| TC\_1303 | Try to create backup when the Database is unreachable |
| TC\_1400 | The system recover the last stable point successfully |
| TC\_1401 | The system can’t recover the last stable point |
| TC\_1402 | Artificial execution where after the first failed execution and the successfully recovered stable point the execution is successful. |
| TC\_1403 | Artificial execution where after the first two failed execution and the successfully recovered stable point the execution is successfully |
| TC\_1404 | Artificial execution for what happened when all the recovery attempt fail. |

Example Details of the TC\_1402 test case:

|  |  |
| --- | --- |
| Title | Content |
| Test Case ID | TC\_1402 |
| Description | Artificial execution where after the first failed execution, and the successfully recovered stable point, the execution will successful. |
| Input | Appropriate module which fails at the first time and successful at the second time. |
| Steps | 1. Administrator creates a new workflow with this module as a start module.  2. Administrator sets the appropriate XML and XSD files.  3. Scientist starts the simulation with appropriate parameters. |
| Dependencies |  |
| Expected result | The execution of the module should fail. After the Recovery Manager recovery the workflow and the second execution should be successful. |

## Trace-ability Matrix for test-cases

The Trace-ability Matrix is also usable to check that at least one test-case exists for every use-cases. The columns contain the use-cases, while the rows contain the test-cases. When a test-case connects with a use-case, we sign it with a dot.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | UC\_U1 | UC\_U2 | UC\_U3 | UC\_S1 | UC\_S2 | UC\_S3 | UC\_S4 | UC\_A1 | UC\_A2 | UC\_A3 | UC\_A4 | UC\_A5 | UC\_A6 | UC\_A7 | UC\_M1 | UC\_M2 | UC\_M3 |
| TC\_0001 | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0002 | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0003 | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0004 | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0005 |  | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0100 |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0101 |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0102 |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0200 |  |  |  | ● | ● |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0201 |  |  |  | ● | ● |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0202 |  |  |  | ● | ● |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0203 |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0204 |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0300 |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0301 |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |  |
| TC\_0400 |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |
| TC\_0401 |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |
| TC\_0402 |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |  |
| TC\_0500 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0501 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0502 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0503 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0504 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0505 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0506 |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |  |
| TC\_0600 |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |
| TC\_0601 |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |
| TC\_0602 |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |
| TC\_0603 |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |
| TC\_0604 |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |
| TC\_0605 |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |  |
| TC\_0700 |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |
| TC\_0701 |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |
| TC\_0702 |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |
| TC\_0703 |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |  |
| TC\_0800 |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |
| TC\_0801 |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |  |
| TC\_0900 |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |  |
| TC\_0901 |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |
| TC\_1000 |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |
| TC\_1001 |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |
| TC\_1002 |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |
| TC\_1003 |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |  |
| TC\_1100 |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |
| TC\_1101 |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |
| TC\_1102 |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |
| TC\_1103 |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |
| TC\_1104 |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |  |
| TC\_1200 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |
| TC\_1201 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |  |
| TC\_1300 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |
| TC\_1301 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |
| TC\_1302 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |
| TC\_1303 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |  |
| TC\_1400 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |
| TC\_1401 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |
| TC\_1402 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |
| TC\_1403 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |
| TC\_1404 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ● |

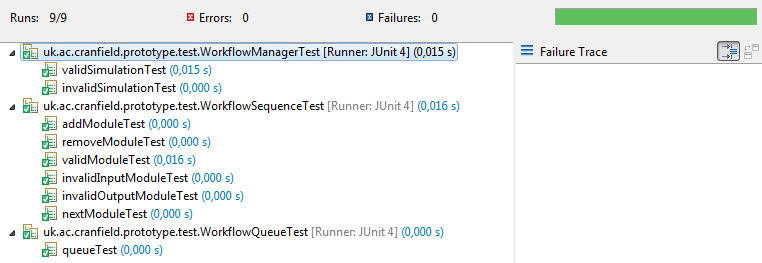
2. Table - Trace-ability matrix for test-cases

# Unit test

We choose the Java programing language for the implementing our system. JUnit is an easy to use environment for Java, to create repeatable tests. So it was convenient to choose the JUnit as a testing environment.

We planned to create JUnit testes for every unit and component.

Many mock classes have been created to ensure the excitability of the prototype. Furthermore these mock classes are very important to execute the unit testes. We have created unit testes for the implemented classes which provide full functionality. Few testes have been created in each unit test to cover the every class functionality. The following diagram shows the result of unit test execution.



# Integration Test

We plan to test the bigger components of the application, and examine how they work together. The modules which we plan to test in the integration test:

* WorkflowManager
* RecoveryManager
* WorkflowSequence

JUnit is not limited for unit tests, so we decided to use JUnit where it is a possible in the system.

A few scenarios what we plan to examine:

1. The Workflow gets knowledge about the unsuccessful execution. It asks the RecoveryManager to get the last stable point, who forwards it to Database Manager. Finally the WorkflowManager should get the last backup from the DatabaseManager through the RecoveryManager.
2. After a successfully execution, if the RecoveryManager decide to store the result it sends it to the DatabaseManager and the backup will be store in the database.

# References

|  |  |
| --- | --- |
| [1] | <http://www.softwaretestinghelp.com/how-to-test-software-requirements-specification-srs/>, 2012 |
| [2] | Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black : Foundations of software testing |
| [3] | Brian Hambling : "SOFTWARE TESTING - An ISTQB–ISEB Foundation Guide", 2nd Edition, 2010 |
| [4] | Rex Black, Jamie Mitchell: Advanced Software Testing Vol. 3, 2011 |