Model-Based Traceability Tool

TEST PLAN DOCUMENT

MODEL Inc.

Contents

[1. Introduction 2](#_Toc476422426)

[2. Test Items 2](#_Toc476422427)

[3. Tested Features 2](#_Toc476422428)

[4. Quality Requirements \* 2](#_Toc476422429)

[5. Testing Strategy and Approach \* 3](#_Toc476422430)

[6. Syntax \* 3](#_Toc476422431)

[7. Description of Functionality 3](#_Toc476422432)

[8. Arguments for Tests 3](#_Toc476422433)

[9. Expected Output 3](#_Toc476422434)

[10. Specific Exclusions 3](#_Toc476422435)

[11. Dependencies 3](#_Toc476422436)

[12. Test-case Success/ Failure Criteria 4](#_Toc476422437)

[13. Pass/Fail Criteria for the Complete Test Cycle 4](#_Toc476422438)

[14. Entrance Criteria/ Exit Criteria 4](#_Toc476422439)

[15. Test Suspension Criteria and resumption requirements 4](#_Toc476422440)

[16. Test Deliverables/ Status Communication Vehicles 4](#_Toc476422441)

[17. Testing tasks 5](#_Toc476422442)

[18. Hardware and software requirements 5](#_Toc476422443)

[19. Problem determination and Correction Responsibilities 5](#_Toc476422444)

[20. Staffing and training needs/assignments 5](#_Toc476422445)

[21. Test Schedules 5](#_Toc476422446)

[22. Risks and Contingencies 5](#_Toc476422447)

[23. Approvals 5](#_Toc476422448)

[24. References 6](#_Toc476422449)

# Introduction

This is a test plan document for the Model-Based Traceability Tool. This document elaborates on how Model Inc. intends to verify and validate their various project artifacts using various verification methods such as analysis, inspection, testing and demonstration.

# Test Items

The test items include all the artifacts related to this project ranging from the requirements document to user manual for the software. The test items are enlisted as follows:

1. Requirements document
   1. Use cases
   2. Scenarios
2. Design document
3. Architecture document
4. Detailed design document
5. Developed code
6. Project related artifacts such as planning, tracking,

# Tested Features

For our tool, the tested features are derived from the specific set of use cases. They are:

1. Manage Models
2. Manage users
3. View Traceability
4. Manage Instances
5. Manage Linkages

# Quality Requirements\*

|  |  |  |  |
| --- | --- | --- | --- |
| SNo. | Use case | Characteristic associated (ISO 25010 quality model) | Sub-characteristic associated (ISO 25010 quality model) |
| 1. | Manage Models | Performance and Efficiency | Time behavior, Resource utilization |
| 2. | Manage Users |  |  |
| 3. | View Traceability |  |  |
| 4. | Manage Instances |  |  |
| 5. | Manage Linkages |  |  |

# Testing Strategy and Approach\*

|  |  |
| --- | --- |
| Use case | Manage Models |
| Test objective | To successfully test the functionality of the creation/update/deletion of model which further includes creation of entities, creation of properties for the corresponding entities and linkage of entities. |
| Verification Method | Testing, Inspection |
| Specifics of the method | 1. **Analysis**: Perform static analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc. 2. **Inspection**: Perform code review with a checklist. The checklist may contain the basic checks like for checking for code modularity, etc. 3. **Testing**: 4. **Black box**: Draw classification tree to identify the equivalence classes from the test aspects. Test cases can be generated using these equivalence classes. 5. **White Box:** Ensure all valid basis paths are traversed so that the execution paths have been tested at least once; 100% branch and statement coverage; white box testing needs to be performed when the model is committed to the database and the program checks if the model is valid before storing it in the database. |
| Testing environment | Development environment |
| Tools required | 1. Testona – Classification Tree 2. ACTS – Generation of test specifications 3. JUnit – Black box testing 4. SonarLint – Static analysis on code 5. Code review facilities of Git Hub can be used. |
| Training Required | 1. Testona: http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html 2. ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html 3. JUnit: http://junit.org/junit4/index.html 4. SonarLint: http://www.seliniumhq.org/docs/ |
| Parameters of Interest | 1. Basis Path coverage 2. Combinatorial Coverage 3. Inspection checklist |

|  |  |
| --- | --- |
| Use case | Manage Properties |
| Test objective | To successfully test the functional correctness of Manage Properties. |
| Verification Method | Analysis, Inspection and Testing |
| Specifics of the method | 1. **Analysis**: Perform static analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc. 2. **Inspection**: Perform code review with a checklist. The checklist may contain the basic checks like for checking for code modularity, etc**.** 3. **Testing**: 4. **Black box**: Draw classification tree to identify the equivalence classes from the test aspects. Test cases can be generated using these equivalence classes. |
| Testing environment | Development environment |
| Tools required | 1. Testona – Classification Tree 2. ACTS – Generation of test specifications 3. JUnit – Black box testing 4. SonarLint – Static analysis on code 5. Code review facilities of Git Hub can be used. |
| Training Required | 1. Testona: http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html 2. ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html 3. JUnit: http://junit.org/junit4/index.html 4. SonarLint: http://www.seliniumhq.org/docs/ |
| Parameters of Interest | 1. Basis Path coverage 2. Combinatorial Coverage 3. Inspection checklist |

|  |  |
| --- | --- |
| Use case | Manage Instances |
| Test objective | To successfully test the functionality of the creation/updation/deletion of values of the properties associated with an instance. |
| Verification Method | Analysis, Inspection and Testing |
| Specifics of the method | 1. **Analysis**: Perform static analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc. 2. **Inspection**: Perform code review with a checklist. The checklist may contain the basic checks like for checking for code modularity, etc**.** 3. **Testing**:    1. **Black box**: Draw classification tree to identify the equivalence classes from the test aspects. In this case, this will pertain to the properties of the instance. Some examples could be “required”, “value”, etc. You can then come up with the test cases from these equivalent classes. |
| Testing environment | Development environment |
| Tools required | 1. Testona – Classification Tree 2. ACTS – Generation of test specifications 3. Jasmine – To create and run the Javascript tests 4. SonarLint – Static analysis on code 5. Code review facilities of Git Hub can be used. |
| Training Required | 1. Testona: http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html 2. ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html 3. Jasmine: https://jasmine.github.io/pages/docs\_home.html |
| Parameters of Interest | 1. Basis Path coverage 2. Combinatorial Coverage 3. Inspection checklist |

|  |  |
| --- | --- |
| Use case | View Traceability |
| Test objective | 1. To check if the instances which are linked to a selected instance appear in the view. 2. To check if the exported pdf has the required fields which have been selected by the user. |
| Verification Method | Analysis, Inspection and Testing |
| Specifics of the method | 1. **Analysis**: Perform static analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc. 2. **Inspection**: Perform code review with a checklist. The checklist may contain the basic checks like for checking for code modularity, etc**.** 3. **Testing**:    1. **Black box**: Draw classification tree to identify the equivalence classes from the test aspects. In this case, this will pertain to the properties of the instance. You can then come up with the test cases from these equivalent classes. |
| Testing environment | Development environment |
| Tools required | 1. Testona – Classification Tree 2. ACTS – Generation of test specifications 3. Selinium – To create and run UI test to see if the linked instances show up in the view or not. 4. Manual Checking – To check the downloaded pdf 5. SonarLint – Static analysis on code 6. Code review facilities of Git Hub can be used. |
| Training Required | 1. Testona: http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html 2. ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html |
| Parameters of Interest | 1. Basis Path coverage 2. Combinatorial Coverage 3. Inspection checklist |

|  |  |
| --- | --- |
| Use case | Manage User roles and permissions |
| Test objective | To test the functional correctness of Manage Users by ensuring that users are associated for a set of roles in a project and their access permissions for the entities have been set. |
| Verification Method | Analysis, Inspection and Testing |
| Specifics of the method | 1. **Analysis**: Perform static analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc. 2. **Inspection**: Perform code review with a checklist. The checklist may contain the basic checks like for checking for code modularity, etc**.** 3. **Testing**:    1. **Black box**: The above requirement can be tested by using Decision Tables. This allows us to specify conditions/rules for testing. For example, an analyst is provided read and write permissions for the requirements and design entities. This implies that he can create requirement instances |
| Testing environment | Development environment |
| Tools required | 1. Testona – Classification Tree 2. ACTS – Generation of test specifications 3. Selinium – To create and run UI test to see if the linked instances show up in the view or not. 4. Manual Checking – To check the downloaded pdf 5. SonarLint – Static analysis on code 6. Code review facilities of Git Hub can be used. |
| Training Required | 1. Testona: http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html 2. ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html |
| Parameters of Interest | 1. Basis Path coverage 2. Combinatorial Coverage 3. Inspection checklist |

# Syntax\*

# Description of Functionality

# Arguments for Tests

# Expected Output

# Specific Exclusions

# Dependencies

# Test-case Success/ Failure Criteria

# Pass/Fail Criteria for the Complete Test Cycle

# Entrance Criteria/ Exit Criteria

# Test Suspension Criteria and resumption requirements

# Test Deliverables/ Status Communication Vehicles

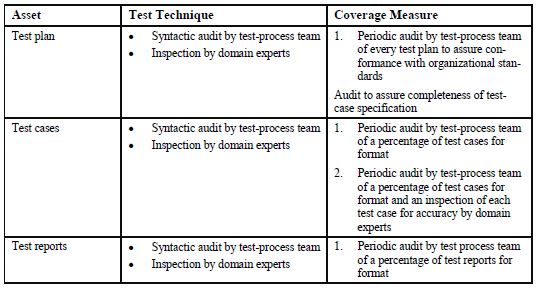


Figure 4: Test Assets [[1]](#_References)

# Testing tasks

# Hardware and software requirements

# Problem determination and Correction Responsibilities

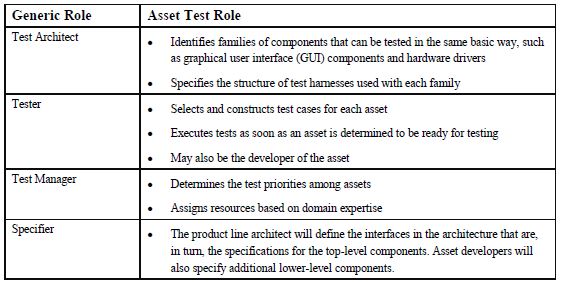


Figure 5: Roles and responsibilities in Assets testing [[1]](#_References)

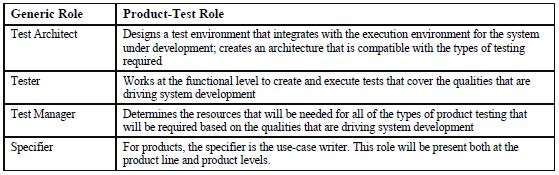


Figure 6: Roles and responsibilities in Product testing [[1]](#_References)

# Staffing and training needs/assignments

# Test Schedules

# Risks and Contingencies

# Approvals

# References

1. McGregor, John. "Testing a software product line." (2001).