SOFTWARE QUALITY PLAN FOR PROJECT

“MODEL BASED TRACEABILITY TOOL”

V 1.3

MARCH 2, 2017

Created by Team 1

(Adivandhya B R, Dhruv, Lavanya R, Ajay, Aparrnaa)

RECORD OF CHANGES

***A** - ADDED M - MODIFIED D - DELETED

VERSION NUMBER	DATE	NUMBER OF FIGURE, TABLE OR PARAGRAPH	**A* M D**	TITLE OR BRIEF DESCRIPTION
1.0	11/01/2017	Entire Document	A	Created the initial document to capture the functional requirements and the quality requirements for the given project
1.1	17/02/2017	1.3	M	Project context is explained in greater detail
1.2	17/02/2017	Added the sections about test verification details	A	Added the sections about test verification details
1.3	02/03/16	Added section on Quality Control Plan	A	Added details on the testing approach, entry and exit criteria, tools, adequacy measures for each functional/quality requirements and testing of artifacts.

TABLE OF CONTENTS

Section **Page**

SECTION 1. INTRODUCTION 2

SECTION 2. REQUIREMENTS GATHERING 4

SECTION 3 – VERIFICATION TECHNIQUES AND TOOLS 18

SECTION 1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to define the Software Quality plan of “Traceability Tool” project.

1.2 SCOPE

In this current version of the document **(v1.0),** we intend to provide: a. functional requirements in the form of use-cases and use-case diagrams; b. quality requirements in the form of raw-scenarios and six-part scenarios categorized based on the dominant quality characteristics and sub-characteristics exhibited by the respective scenario.

1.3 PROJECT CONTEXT

Model-based traceability is a tool which would aid the project teams in pulling the reliable project artifacts for a change request within a minute to meet the client’s SLA period. The tool will be built in such a way that model manager is provided with a user interface using which he will be able to create model, associate properties based on the CMMI level 5 standard and assign user roles and responsibilities. Currently, wrong artifacts are mapped up say for example, business use cases being mapped to system test cases since there are no user level restrictions. Because of this, when there is a change request it gets difficult to track all the right artifacts to the corresponding requirement and it takes anytime between 2 days to 2 hours to manually retrieve the details from the excel sheets maintained for every project. Currently, the linkages are maintained in the excel by associating the related instances (say instance R89 is linked to C87 by using their IDs). Hence, with the model manager tool a desktop based application in place project manager can define the model and assign user roles and restrictions and can easily retrieve the artifacts is less than 5 minutes.

Developers and other team members would be provided with a traceability web UI wherein he can create project instances(artifacts) and link them up. Using the standard model defined in the model manager tool, the project team can create project instances(artifacts) based on their respective roles and responsibilities.

Consider a project model defined in model manager tool is - Requirement à Design à Test Case and it is defined with properties for each element. The model manager would then assign roles and restrictions. Based on this, developer X can create a requirement artifact *“R098 Implement Search Engine”* and link it up with design artifact “*D098 Search Engine Design*” only if he has given the rights to do so.

This application will also provide an option to view traceability and generate project reports which can then be used for creating the patch for a change request. While viewing the traceability, all the related artifacts of base and customized version for each client have to be listed within 5minutes.

Model manager is also provided with Import Project Model facility using which he can copy the project details from the base version and later customize it.

By restricting the users and showing minimum number of artifacts at a point, we are able to achieve reliability to some extent,

- Project manager can benefit by ensuring that unauthorized access to create artifacts could minimize the wrong entries being made in the system.
- Developers do not have the access to create or link up artifacts other than the entity they are assigned to.

SECTION 2. REQUIREMENTS GATHERING

2.1 FUNCTIONAL REQUIREMENTS

The project contains a total of **5** system-wide functional requirements documented as use cases as shown below. In addition, a list of high-level features has been captured below.

2.1.1 USE CASE DIAGRAM

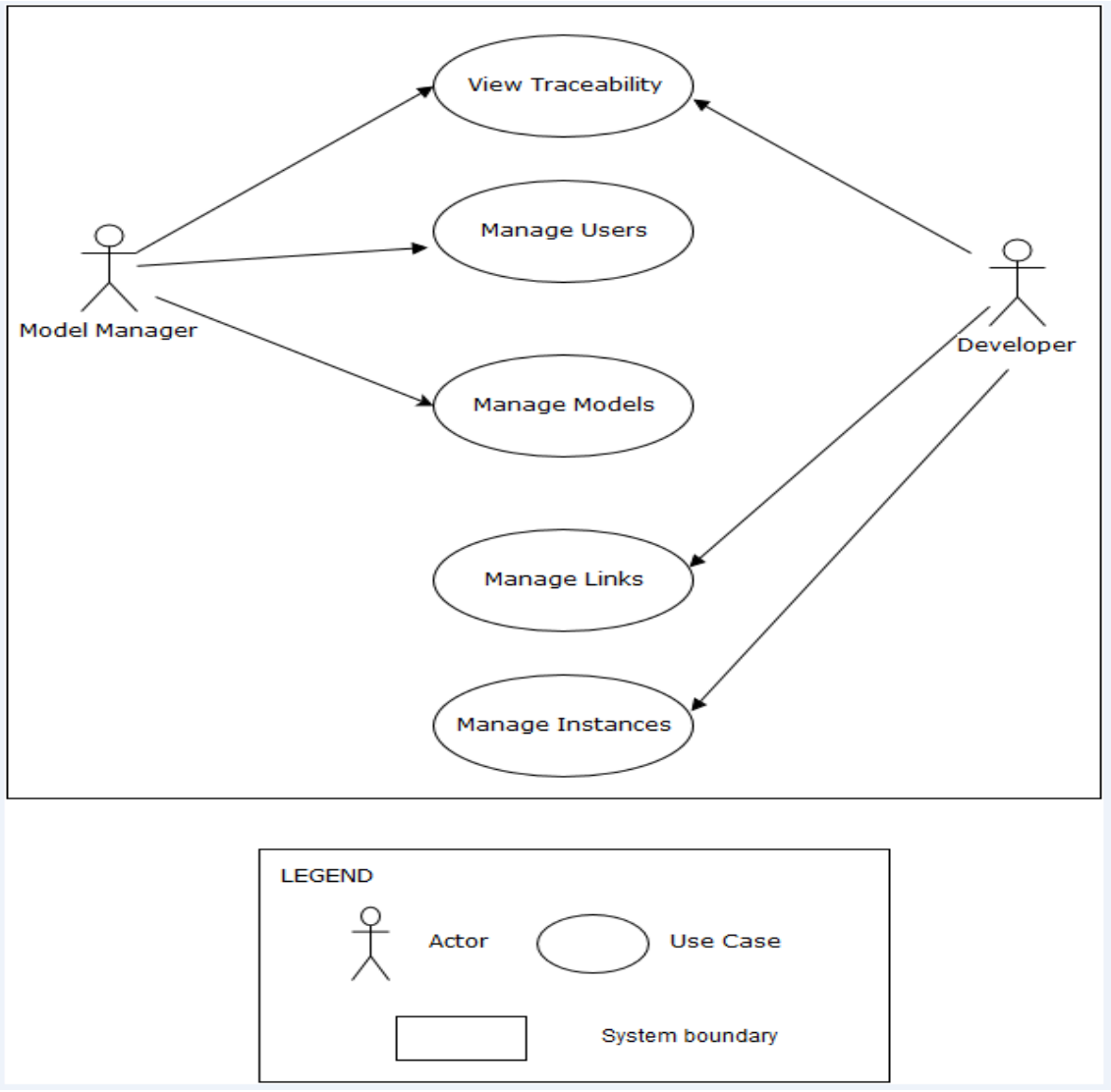

Fig I: Model-based Traceability Use case diagram

2.1.2 DETAILED USE CASE DESCRIPTIONS

- **Manage Models**

Use case name	**Manage Models**
Use case ID	UC #1
Primary actor(s)	Model manager
Secondary actor(s)	None
Brief Description	The model manager can create, update, delete and save a project model. He will be able to associate properties for all the entities defined in the model. This use case gets triggered when the model manager wants to performs one of the below mentioned operations: • He has created a new project, and wants to define a model for that project. • When he wants to create, or edit the properties associated with each entity defined in the model. • He wants to import an existing project model and make necessary modifications. • IF the model manager would like to view the model for an existing project for which he has been assigned, then he can view the model or make any changes to the model entities or the properties of the model entities. • IF the model already created for the project is no longer required, the model manager can delete the model along with its associated properties. The model manager can perform one of the above-mentioned operations.
Pre-conditions	The model manager has access to the Model Manager tool and has the privileges to create a new project or open an existing project for modifications.
Flow of events	Any one of the following alternate flows might get triggered based on whatever operation the model manager wants to perform.
Alternate flows and exceptions	**Create Project Model:** IF the model manager wants to “Define Model” for a new project which he has created, then refer AF #1 **Import/Copy Project Model:** When the model manager wants to “Import Model” for a new project which he has created, then refer AF #2 **Delete Project Model:**

	When the model manager wants to “Delete Project Model” completely, such that he is no longer responsible for maintaining the project. He opens up the corresponding project form the dropdown listed after he gets authenticated. The system provides a delete option which he clicks to completely remove the project from the dropdown listing and make it unavailable for further maintenance. **Save Project Model:** 1. During all the alternate flow mentioned above, the model manager can choose to save the project model for later purpose. The system provides a save option in the UI which the manager can click to persist the model 2. If system does not respond for the save option, corresponding feedback will be provided to the manager and state-of-the-system is maintained.
Post-conditions	The model entities and properties are updated in the system.
Non-behavioral requirements	The model manager must be able to configure the model entities and properties according to his project and be given with an interactive UI for defining/editing the model.
Assumptions	1. The employee chosen as the model manager for the project would be a trusted one and would adhere to CMMI level 5 standards as closely as possible.
Priority	High
Issues	
Source	Stakeholder Meeting MOM- 23/09/2016

- **Alternative Flow 1** for **Use Case 1**

Alternative Name	Create Project Model
ID	AF #1
Primary Actor	Model manager
Secondary Actor	None
Brief Description	This alternate flow is defined for the model manager who wants to "Define Model” for a new project which he has created. He would also associate properties for each entity in the project model.
Preconditions	1. The manager has created a new project 2. The manager chooses to define the model from scratch 3. An empty canvas is presented to the model manager where he will be able to view the list of entities as per CMMI level 5 standard.

	{Note: The system shall provide a model canvas editor with an interactive UI where he can define the project model.}
Flow of events	1. Model manager creates an entity (requirement, design, code, test case, or etc) for the project model by providing the entity name and description. 2. The model manager has to define the properties for the entity created. He can choose to customize the standard CMMI level 5 properties provided by the system or create new properties on his own. {Note: The system shall store the standard properties for every entity which will be made available to the model manager once he has created an entity} 3. Step 2 would end successfully, when the manager has defined all the properties for a particular entity. 4. Repeat step 1 and 2 until all other entities (requirement, design, code, test case, or etc.) to be present in the project would be created. 5. Model manager then appropriately provides the linkages between the entities that are created above. 6. He then saves the project model once he thinks that everything is precise and accurate or when he wishes to resume his work at later point in time. 7. This alternate flow ends successfully when the system provides the feedback that "*The project model has been created successfully*" to the model manager.
Postconditions	The project model is saved in the data store and will be available for the later purpose of defining the project instances and to link them up.
Priority	High
Non-behaviorial requirements	None
Assumptions	1. The model manager has gone through the user manual and knows how to work with the interactive UI to create the project model. 2. This flow would get initiated when a new project request is sent to the model manager
Issues	1. Manager faces an inherent lag while defining the entities or while linking them up. 2. When the project model is not saved successfully then the system has to make sure that the model created by the model manager does not get erased. Corresponding feedback has to be provided to the manager and the system should restore its state automatically.
Source	Stakeholder Meeting MOM- 23/09/2016

- Alternative Flow 2 for Use Case 1

Alternative Name	Import Project Model
ID	AF #2
Primary Actor	Model manager
Secondary Actor	None
Brief Description	This alternate flow is defined for the model manager who wants to "Import Project Model" from an existing project and wants to customize it for the new project request. Modifications will be done to the entities, or the linkages, or the properties which were already defined in the version that it is derived from.
Preconditions	1. The manager has a new project which is quite similar to the already existing and chooses to import it. 2. The model editor canvas would be populated with the existing project model, properties and linkage details chosen by the manager.
Flow of events	Once the project model is made available in the system for modification, the model manager would perform on of the following operation: 1. Model manager might want to incorporate a new entity into the already existing project model. In which case, he can would define the entity by providing the entity name and description. 2. The model manager would want to modify or define the properties for the new entity that he created in step 1, in which case the system would provide the standard CMMI level 5 properties or create new properties on his own. 3. Model manager might want to edit the already linked entities in the project model. 4. He then saves the project model once he thinks that everything is precise and accurate or in case he wishes to resume his work at later point in time. 5. This alternate flow ends successfully when the system provides the feedback that "*The project model has been saved successfully*" to the model manager.
Postconditions	The project model is saved in the data store and will be available for the later purpose of defining the project instances and to link them up.
Priority	High
Non-behaviorial requirements	None
Assumptions	3. The model manager has gone through the user manual and knows how to work with the interactive UI to import the project model from already existing model. 4. This flow would get initiated when a new project request is sent to the model manager and the process followed is similar to already defined project model.
Issues	When the project model is not saved successfully then the system has to make sure that the model created by the model manager does not get erased.

	Corresponding feedback has to be provided to the manager and the system should restore its state automatically. The exception has to be handled appropriately by the system.
Source	Stakeholder Meeting MOM- 23/09/2016

- **Manage Users**

Use case name	**Manage Users Roles & Permissions**
Use case ID	AF #5
Primary actor(s)	Model Manager, Group Lead
Secondary actor(s)	None
Brief Description	The model manager would want to create user roles, update/delete them whenever required. He would assign access permissions to each entity defined in the project model. Model manager would be the one responsible for creating group leaders for the project and associate the entities under which the group leader would get complete access. For example, model manager X would create group leader Y and associate him to “Design” entity. The group leader would then have the access to create developers for that particular entity and give them read/write access. Group leader has access only to that particular entity and so he is restricted from updating the project model or making any chances to other entities. Under this scenario, he can only add more developers to the entity he is associated with and give them read or write access permission to the developers. This use case requirement is a critical functionality that has to be developed to make the system more reliable and prevent the development team in linking up wrong artifacts. For instance, if developer X has created R089 artifact he would have the permission to edit only that particular artifact. Model manager would provide the user restriction and make developer X to access only that particular entity and instances he has created. This use case gets triggered when the project model is defined and the manager/group lead wants to create the roles and restrictions for each member in the project team so that unauthorized manipulations could be prevented. Model manager/group leader can edit or delete users from the project as well. Once they are deleted they might not to be able to login or use the system but the artifacts created by them would still be under their

	name and it is the responsibility of the manager/group lead to associate a new replacement resource.
Pre-conditions	1. The model manager has created a new project model for a particular project and wants to associate roles and define permissions. 2. When model manager wants to create group leaders and associate them to a particular entity within a project. 3. When group lead wants to associate developers who work under a particular entity and wishes to provide read/write access to them. 4. Or, if model manager wishes to update the user roles and permissions for an already existing project model.
Flow of events	1.The system presents a list of standard roles present within the organization, when the model manager wants to define roles for a particular project. 2. He then selects the roles one by one and saves them. Or, if the model manager wants to custom add new role, the system provides such capability. 3.Once the model manager has defined the project roles, he can assign specific users to these roles. 4.The system now shows the list of users present within the organization. 5.Model manager chooses a particular user and assigns specific role to the user involved in the project. 6. At this point, model manager can assign an entity to the group lead. 7.He then provides read/write permissions for the entities in the model for each and every user. Or, even group lead can create users/developers and assign responsibilities to them.
Alternate flows and exceptions	**Create User Roles:** IF the model manager wants to “Create User Roles” then, the system provides the list of users and standard roles. **Update User Roles:** When the model manager wants to “Update User Roles” then, he can

	delete existing roles or add new roles into the system. **<u>Manage Permissions:</u>** IF the model manager wants to "Define User Permissions" then, the manager provides read and write options for the entities and associate them to the roles defined in the project model.
Post-conditions	The users for a project and their corresponding access permissions to model entities are updated. Based on these least privilege restrictions developer can create instances only under that particular entity for which he has the access.
Non-behavioral requirements	
Assumptions	The project model is created and is made available for the model manager to enable the role-based access permission for the project defined.
Priority	High
Issues	Previously, with the help of excel anybody had access to edit the instance details which led to retrieval of wrong artifacts at a later stage.
Source	Stakeholder Meeting MOM- 23/09/2016, 02/02/2017

- **Manage Instances**

Use case name	**Manage Instances**
Use case ID	UC101
Primary actors	Developers,Group Lead
Secondary actors	-
Brief Description	The developer wishes to create/edit an instance pertaining to an entity.
Preconditions	The model for the particular project has been created by the model manager

	The model manager has defined all the properties pertaining to the entities in the model
	The model manager has defined the roles pertaining to the project
Flow of events	1. The user views all the existing projects and selects a project where he wants to create/edit an instance
	2. The user selects the entity in the model where he wants to create/edit an instance
	3. The user would have the option to either create an instance or edit an instance.
	3. IF the user selects to create an instance, all the properties pertaining to that entity (which would have been defined by the model manager)where he wants to create the instance would be retrieved and shown to the user. The user has to fill the values for the properties defined by the model manager. Once the developer fills all the necessary values, he would be able to save the changes.

- **Manage linkages/links**

Use case name	**Manage linkages/links**
Use case ID	UC #4
Primary actors	Developers
Secondary actors	Group Lead
Brief Description	Other users apart from Project Manager links the instances that are created by them with the instances to which they have read only access permission.
Preconditions	The group lead should have assigned instances for the specific developer.
Flow of events	1. The developer opts to select a element. 2. The system should present the developer with the list of elements and its instances. 3. The developer links an instance of one element to another.
Post conditions	The instances of two elements have been linked.
Priority	High
Alternate flows and exceptions	-
Non-behavioral requirements	The developer should be able to link the instances of element with another within 2 seconds.

	The developer should be able to drag and drop and link instances. Similarly, developer should be able to drag and drop and delete a linkage made by him.
Use case name	**Manage linkages/links**

- **View traceability**

Use case name	**View traceability**
Use case ID	UC #5
Primary actors	Developers, model manager
Secondary actors	-
Brief Description	The users view the traceability of the project instances and the associated linkages between them. Traceability also includes the instances (and linkages) from its parent and/or child project versions if any.
Preconditions	The model has been defined by the manager. Project instances are created and linked by the developers.
Flow of events	1. Developer logs into the model-based traceability tool. 2. The system displays all the projects that the developer is authorized to work on. 3. The developer selects one project. 4. The system provides the instances pertaining to that project along with the instance descriptions. 5. The user can also additionally give the name of a particular instance to view traceability. 6. The user can now view the traceability of the project across modules.
Post-conditions	The traceability of the project is seen.
Priority	High
Alternate flows and exceptions	-
Non-behavioural requirements	The traceability must be easily understood when viewed by the users. Traceability pulls up more records (say 10000 +) how to display the details to the developer/manager such that it is easy to understand and exported into pdf.
Assumptions	The users can view the names of the projects and related instances for guidance.

2.2 QUALITY REQUIREMENTS

We were given ISO 25010 quality model. We have considered all of them and whichever is not relevant for the project, we have pointed out that this quality characteristic/sub-characteristic is not relevant. The quality characteristics and sub characteristics that are relevant have been identified from the project context, use cases and the concrete scenarios that have been given. We mapped the quality characteristics and sub characteristics of the project to the ISO 25010. We have mentioned which of the quality requirements are relevant but not yet documented in six part scenario. In the project, a total of 5 quality scenarios were documented in the form of 6-part scenarios. For each scenario, we mentioned the quality attributes that are associated with the scenario.

1. **Characteristic :** Performance Efficiency

Scenario 1.1:

Scenario Name	**View Traceability**
Scenario	Company X places a CR to the Project Manager. While identifying the impact of CR manually, he loses track of his progress and decides to start from the beginning, thereby more time is incurred to find the CR impact and the manager is not sure if the right artifacts for the change has been identified.
Source of Stimulus	Developer
Stimulus	View the traceability of a project related to a change request
Environment	Runtime environment - normal operations
Element generated	View Traceability component
Response	The system lists the project instances along with their associated linkages.
Response Measure	Within an average of 1 minute

Sub characteristics associated with this scenario: Time behavior

Scenario 1.2:

Scenario	Change requests from clients arrive under normal operations. SSK private limited has found that the bug there is a bug in the code instance C01. Now, they are viewing all the artifacts that will be affected with the bug in C01 within an

	average latency of 1 minute.
Source of stimulus	Clients
Stimulus	change requests
Environment	Runtime environment - normal operations
Element generated	Traceability UI component, View Traceability component
Response	System retrieves the instances linked with change request
Response measure	within an average latency of 1 minute

Sub characteristics associated with this scenario: Time behavior

List of sub-characteristics which are not applicable / applicable but not documented

- Resource utilization: **Applicable but not documented**
- Capacity: **Non Applicable**

2. **Characteristic:** Compatibility: **Non Applicable**

3. **Characteristic:** Usability

Scenario 3.1:

Scenario Name	**Define Model**
Scenario	Project manager creates a model for a project consisting of requirements, design entities and links requirements to design
Source of Stimulus	Project manager
Stimulus	Model definition for a new project
Environment	Runtime environment - normal operations
Artifacts	Model Editor
Response	The system collects the model information and saves it.
Response Measure	Feedback is provided to the manager that the model is defined and saved.

Sub characteristics associated with this scenario: Operability, Learnability, User error tolerance, User interface aesthetics

Scenario 3.2:

Scenario	Tester X attempts to link test case instance T12 with code instance C12 in the runtime environment (under normal operations). The system displays the minimal number of authorized instances to the tester along with Instance descriptions. This allows him to make informed decisions and reduces the chance of wrongly linking by 60%.
Source of stimulus	Developers
Stimulus	attempts to link instances
Environment	Runtime environment - normal operations.
Artifacts	Traceability UI, Link Instances component, Manage Users component
Response	The system shows a minimal number (say, 4 instead of 10 in a page) of authorized instances to the developer along with instance descriptions for him to look through the instances properly before linking them up.
Response measure	60% reduction in a number of users' slips/users.

Sub characteristics associated with this scenario: User error protection, Appropriateness, recognizability, User interface aesthetics

List of sub-characteristics which are not applicable

- Accessibility

4. **Characteristic:** Reliability

- Maturity: Non Applicable
- Availability: Applicable but not documented**.**
- Fault tolerance: Applicable but not documented.
- Recoverability: Non Applicable

5. **Characteristic:** Security

Scenario 4.1:

Scenario	Tester X attempts to link requirement instance R12 with design instance D12 in the runtime environment (under normal operations). The system denies such attempts and this threat is overcome by 100%.
Source of stimulus	Developers
Stimulus	Attempts to link unauthorized instances
Environment	Runtime environment - normal operations
Artifacts	Link Instances component, Manage Users component
	component
Response	System denies such attempts
Response measure	Threat is overcome by 100%

Sub characteristics associated with this scenario: Confidentiality

List of sub-characteristics under Security which are relevant but not documented

- Integrity
- Non-repudiation
- Accountability
- Authenticity

6. **Characteristic:** Maintainability:
   List of sub-characteristics which are relevant but not documented

   - Modularity
   - Reusability
   - Modifiability
   - Testability

   List of sub-characteristics which are not relevant

   - Analyzability

7. **Characteristic:** Portability: **Non Applicable**

SECTION 3 – VERIFICATION TECHNIQUES AND TOOLS

1. **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.

Input: Model Name, List of Entities, List of properties associated with the entities and list of linkages.

Output: Creation successful or unsuccessful.

Verification Method: Testing, Inspection.

Specifics of the methods:

- **Analysis:** Perform Static Analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc., in the code.

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like for checking for code modularity, etc.

- **Testing:** In testing we suggest both the Black Box and White Box Testing to test the functionality of Manage Model function.

    - **Black Box:**
        - For instance, the user creates the model, saves the same and reloads to check if the model is created.
        - Use a classification tree to identify the equivalence classes from the test aspects. Some examples of the test aspects could be “number of associated properties”, “duplicate model name” etc. You can then come up with the test cases from these equivalence classes.
    - **White Box:**
        - To ensure that all the valid basis paths are traversed so that the execution paths have been tested at least once.
        - To test for 100% branch and statement coverage.
        - White box testing needs to perform when the model is committed to the data base and the program checks if the model is valid before storing it in the database.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required: As Testona and ACTS are open source tools and as the team has hands on working for the assignments prior we would suggest the same.

- **Testona** - For generating Classification Tree.
- **ACTS** - Generation of Test Specifications.
- **JUnit** - Black Box testing, we suggest this as the code is being developed in JAVA. It is also used for UI testing
- **SonarLint –** To perform static analysis on code.
- **Code review facilities of Git Hub** can be used by the team as they are already using the same.

Training required: Tool training is required. As they are open source tools they are easily available. Following are the links for self-learning and to download.

- **Testona:** http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html
- **ACTS:** http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html
- **JUnit:** http://junit.org/junit4/index.html
- **SonarLint:** http://www.seliniumhq.org/docs/

Parameters of interest:

- **Basis Path Coverage**: Tracked through White Box Testing by the help of Control flow graph.
- **Combinatorial Coverage**: Tracked in 3-way testing - Black Box Testing
- **Inspection Checklist** – Tracked as **No. of defects fixed after the code review/ No. of defects found during code-review based on the check-list**

2. **Use Case:** Manage Properties.

Test Objective: To successfully test the functional correctness of Manage Properties.

Input: Property Name, Property type, Property value.

Output: Successfully set the properties or error due to improper setting of the property/property value.

Verification Method: Analysis, Inspection and Testing.

Specifics of the methods:

- **Analysis:** Perform Static Analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc., in the code.

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like for checking for code modularity, etc.

- **Testing:** In testing we suggest both the Black Box and White Box Testing to test the functionality of Manage Model function.

    - **Black Box:**
        - Use a classification tree to identify the equivalence classes from the test aspects. Some examples of the test aspects could be “number of associated properties”, “duplicate model name” etc. You can then come up with the test cases from these equivalence classes.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required: As Testona and ACTS are open source tools and as the team has hands on working for the assignments prior we would suggest the same.

- **Testona** - For generating Classification Tree.
- **ACTS** - Generation of Test Specifications.

- **JUnit** - Black Box testing, we suggest this as the code is being developed in JAVA.
- **SonarLint –** To perform static analysis on code.
- **Code review facilities of Git Hub** can be used by the team as they are already using the same.

Training required: Tool training is required. As they are open source tools they are easily available. Following are the links for self-learning and to download.

- **Testona:** http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html
- **ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html**
- **JUnit:** http://junit.org/junit4/index.html**:**
- **SonarLint:** http://www.seliniumhq.org/docs/

Parameters of interest:

- **Basis Path Coverage**: Tracked through White Box Testing by the help of Control flow graph.
- **Combinatorial Coverage**: Tracked in 3-way testing - Black Box Testing
- **Inspection Checklist** – Tracked as **No. of defects fixed after the code review/ No. of defects found during code-review based on the check-list**

3. **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.

Input: Instance, List of properties associated with the entity which this instance corresponds to along with which properties are mandatory and which are not.

Output: Creation successful or unsuccessful.

Verification Method: Testing, Inspection, Analysis

Specifics of the method:

- **Analysis:** Perform Static Analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc., in the code.

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like for checking for code modularity, etc..

- **Testing:** In testing we suggest Black Box Testing to test the functionality of Manage Instances

    - **Black Box:** Use a 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 of the test aspects could be “required”, “value” etc. You can then come up with the test cases from these equivalence classes.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required: As Testona and ACTS are open source tools and as the team has hands on working for the assignments prior we would suggest the same.

- **Testona** - For generating Classification Tree.
- **ACTS** - Generation of Test Specifications.
- **Jasmine** – To create and run the javascript tests.
- **SonarLint –** To perform static analysis on code.
- **Code review facilities of Git Hub** can be used by the team as they are already using the same.

Training required: Tool training is required. As they are open source tools they are easily available. Following are the links for self-learning and to download.

- **Testona:** http://www.testona.net/en/webshop/testona-light-free-of-charge/index.html
- **ACTS: http://csrc.nist.gov/groups/SNS/acts/documents/comparison-report.html**
- **Jasmine:** https://jasmine.github.io/pages/docs_home.html

Parameters of interest:

- **Basis Path Coverage**: Tracked through White Box Testing by the help of Control flow graph.
- **Combinatorial Coverage**: Tracked in 3-way testing - Black Box Testing.
- **Inspection Checklist** – Tracked as **No. of defects fixed after the code review/ No. of defects found during code-review based on the check-list**

4. **Use Case**: View Traceability

Test Objective 1: To check if the instances which are linked to a selected instance appear in the view.

Input: Instance.

Output: Linked artifacts appear in the view.

Verification Method: Testing, Inspection, Analysis

Specifics of the method:

- **Analysis:** Perform Static Analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc., in the code.

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like for checking for code modularity, etc.

- **Testing:** In testing we suggest Black Box Testing to test the functionality of View Traceability and UI testing

    - **Black Box:** Use a classification tree to identify the equivalence classes from the test aspects. An example of a test aspect could be “linked instances present or not”. You can then come up with the test cases from these equivalence classes.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required: As Testona and ACTS are open source tools and as the team has hands on working for the assignments prior we would suggest the same.

- **Testona** - For generating Classification Tree.
- **ACTS** - Generation of Test Specifications.
- **Selinium –** To create and run UI test to see if the linked instances show up in the view or not.
- **SonarLint –** To perform static analysis on code.
- **Code review facilities of Git Hub** can be used by the team as they are already using the same.

Test Objective 2: To check if the exported pdf has the required fields which have been selected by the user.

Input: List of linked instances, properties associated with entities corresponding to each instance

Output: The downloaded pdf should contain only the selected properties and selected instances.

Verification Method: Testing, Inspection, Analysis.

Specifics of the method:

- **Analysis:** Perform Static Analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc., in the code.

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like for checking for code modularity, etc..

- **Testing:** In testing we suggest Black Box Testing to test the functionality of exporting a pdf which is part of View Traceability.

    - **Black Box:** Use a classification tree to identify the equivalence classes from the test aspects. An example of a test aspect could be “number of selected properties”. You can then come up with the test cases from these equivalence classes.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required: As Testona and ACTS are open source tools and as the team has hands on working for the assignments prior we would suggest the same.

- **Testona** - For generating Classification Tree.
- **ACTS** - Generation of Test Specifications.
- **Manual checking –** To check the downloaded pdf
- **SonarLint –** To perform static analysis on code.
- **Code review facilities of Git Hub** can be used by the team as they are already using the same.

Parameters of interest:

- **Basis Path Coverage**: Tracked through White Box Testing by the help of Control flow graph.
- **Combinatorial Coverage**: Tracked in 3-way testing - Black Box Testing.
- **Inspection Checklist** – Tracked as **No. of defects fixed after the code review/ No. of defects found during code-review based on the check-list**

5. **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.

Input: Users, User roles, permissions (read, write per entity)

Output: Permissions have been assigned to entities according to the role and users are also assigned with roles in a project.

Verification Method: Testing, Inspection, analysis.

Specifics of the methods:

- **Analysis:** Perform Static analysis can be employed to identify syntactic and semantic errors such as null pointer exception, unclosed connection object checking etc. in the code.
- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like checking for code modularity etc.
- **Testing:** In testing we suggest the Black Box to test the functionality of Manage Users and Permissions function.

    - **Black Box:**
        - The above requirement can be tested by way of 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 and link it up to design instances and vice versa. The analyst should not be allowed to create or link code instances.
        - Consider that a user is allocated the roles of a developer and a tester within a particular project. Say, a developer is allowed to link instances of requirements to code. A tester is allowed to link instances of code to test cases. In this case, the access permissions for the user should be taken on the basis of the currently chosen role and both the roles cannot be considered together.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required:

- **JUnit and Excel**- Black Box Testing, we suggest this as the code is being developed in Java. It is also used for UI testing.
- **Code review facilities of GitHub** can be used by the team as they are already using the same.
- **SonarLint-** To perform static analysis.

Training required: Tool training is required. As they are open source tools they are easily available. Following are the links for self-learning and to download.

- **JUnit:** http://junit.org/junit4/index.html
- **SonarLint:** http://www.sonarlint.org/eclipse/index.html

Parameters of interest:

- **Combinatorial Coverage** corresponding to using Decision Tables- Black Box Testing
- **Inspection Checklist** -Inspection

6.

Scenario:

1. Tester creates a new testcase instance T872. He links the new instance to the requirement instance R872 while he is linking up the instances he thinks that code instance C089 is linked incorrectly and hence changes the linkages.
2. Developer X who is working in project A edits the instance linkage in project B mistakenly because he thinks it is wrong.

Concerns:

1. Data Correctness – Wrong artifacts should not be linked up.
2. Authorization – Only authorized users should be allowed to link up instances.

Test Objective: To successfully test the reliability and security quality aspects for the system.

Input: Project Model, Instances pertaining to two different entities, role of user

Output: Instance linkages are consistent/inconsistent with the model.

Verification Method: Testing, Inspection.

Specifics of the methods:

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like checking for code modularity etc.

- **Testing:** In testing we suggest the following testing methodologies.

    - **Testing for security:**
        - Testing for security can be done using OWASP Security Testing Guide V4.
        - Authorization testing is employed.
            - Testing for bypassing privileges
            - Testing for role/privilege manipulation.

    - **Testing for reliability:**
        - Reliability testing involves testing for consistency of data and integrity of the system.
        - Soak testing is employed to test the reliability of the system. Here, the system is subjected to continuous operation for around 8 hours to test the consistency of the model when subjected to multiple modifications over this period of time.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required:

- **JMeter** for soak testing.

- **Code review facilities of GitHub** can be used by the team as they are already using the same.
- **ZED Attack Proxy Project –** To test for system reliability.

Training required: Tool training is required. As they are open source tools they are easily available. Following are the links for self-learning and to download.

- **JMeter:** http://jmeter.apache.org/usermanual/index.html
- **ZED Attack Proxy Project** https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project#tab=Features

Parameters of interest:

- (Number of regressions in the current model when compared to the previous version of model)/ (Number of modifications to the model): This parameter is used to track the state of the model to check for data correctness and verify if the links and permissions for model entities are reflected.
- **Inspection Checklist** -Inspection

7.

Scenario:

Company X places a CR to the Project Manager. While identifying the impact of CR manually, he loses track of his progress and decides to start from the beginning, thereby more time is incurred to find the CR impact and the manager is not sure if the right artifacts for the change has been identified.

Concern: Retrieval of artifacts pertaining to a change request within the stipulated time (1 minute).

Test Objective: To check whether choosing an instance of an entity yields the instances linked to the chosen entity within a minute. The view should also be populated with these linked instances within the allocated minute.

Input: Instance for which traceability is to be identified

Output: Linkages to instances in the same project, base project, domain project.

Verification Method: Testing, Inspection.

Specifics of the methods:

- **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like checking for code modularity etc.

- **Testing:** In testing we suggest the Black Box to test the functionality of the scenarios given.

    - **Black Box:**
        - **Performance testing:**
            - Employ performance testing to identify if the artifacts pertaining to a change request are retrieved within a minute. The end to end latency is considered, that is, time taken to fetch the response when

a request is initiated and the time taken to render the response onto the UI.

Testing environment: We suggest Development environment for all the testing and inspections.

Tools required:

- Use the browser profiler to find the end to end latency.

Training required: Tool training is required. As they are open source tools they are easily available. Following are the links for self-learning and to download.

- **Chrome Profiler Tutorial:** https://developers.google.com/web/tools/chrome-devtools/rendering-tools/

Parameters of interest:

- End to end latency

8.

Scenario :

Developer X wishes to modify the artifacts' linkage. He manually checks the list of artifacts and modifies the linkages by using version number to map the association.

Concerns:

- Ease of use – The provided user interface should be easy to use.

Test Objective: To measure how easy it is to use the user interface.

Input: This user interface.

Output: Accepted or change requests suggested by stakeholder.

Verification Method: Testing, Demonstration

Specific of the method:

- **Testing:** We suggest A/B testing to understand how the various users feel about the UI.
- **Demonstration:** Demonstrate to the user how to use the implemented interface.

Tools Required: none

Training Required: none

Testing Environment: Production Environment

Parameters of Interest:

- Customer satisfaction Is measured by the number of change requests suggested by the customer for the implementation of this scenario. If the number is zero and the customer accepts the implementation, then the implemented scenario is easy to use. Greater the number of change requests, the less easy it is to use.

Mapping between use case and quality scenarios:

Use Case	Scenario	Concern	Quality Attribute
Manage Links	Tester creates a new testcase instance T872. He links the new instance to the requirement instance R872 while he is linking up the instances he thinks that code instance C089 is linked incorrectly and hence changes the linkages.	Data Correctness – Wrong artifacts should not be linked up.	Reliability
Manage User Roles and Permissions	Developer X who is working in project A edits the instance linkage in project B mistakenly because he thinks it is wrong.	Authorization – Only authorized users should be allowed to link up instances.	Security
View Traceability	Company X places a CR to the Project Manager. While identifying the impact of CR manually, he loses track of his progress and decides to start from the beginning, thereby more time is incurred to find the CR impact and the manager is not sure if the right artifacts for the change has been identified.	Retrieval of artifacts pertaining to a change request within the stipulated time (1 minute).	Performance
Manage Links	Developer X wishes to modify the artifacts' linkage. He manually checks the list of artifacts and modifies the linkages by using version number to map the association.	Ease of use – The provided user interface should be easy to use.	Usability

SECTION 4- QUALITY CONTROL PLAN:

Categorical Classification:

Based on the set of features that can be delivered to the customer initially within the available time period, the use cases have been classified into two categories of criticality namely **High** and **Nominal**.

Based on the criticality assigned to a use case, the following types of static analysis should be conformed.

Static Analysis ID	Type of SA	Tool
1	Style Checking	CheckStyle
2	Pattern Matching	Control Flow Graph
3	Control Flow Analysis	Factory
4	Data Flow Analysis	Findbugs
5	Formal Verification	Manual

The use cases and their corresponding criticalities are listed below.

Use case ID	Use case name	Criticality	Business Rules Testing	UI Testing
1	Manage Models	High	Yes-Modules developed	Yes-Modules developed
2	Manage Instances	High	Yes-Modules developed	Yes-Modules developed
3	Manage Properties	High	Yes- Modules not developed	Yes- Modules not developed
4	View Traceability	Nominal	Yes- Modules not developed	Yes- Modules not developed
5	Manage user roles and permissions	Nominal	Yes- Modules not developed	Yes- Modules not developed

Use cases:

1. **Manage Instances:**

Modules pertaining to the use case:

Use Case	UI Modules	Service Modules	Database Modules

Manage Instances	CreateInstances.html ShowInstances.html AddInstances.html EditInstances.html	FetchEntities FetchInstances UpsertInstances	Entity collection Instance collection

Types of testing to be performed:

Unit Testing:
Unit testing can be employed for the standalone service modules. That is, for a particular service, we can provide a request in the desired format and check the obtained response for correctness.

Entry Criteria:

- The use cases or the scenarios associated with the given code artifact have been created and approved by the client.
- The specific code module has been completed.
- The code module has passed the Static Analysis
    - High: (1,3,4)
    - Nominal: (1)

Exit Criteria:
All the generated test cases have been executed in both 2-way testing and white-box basis path testing (for high criticality requirements).

Testing Approach:
We perform black box as well as white box testing

- Generate the classification tree to model the Input domain using Testona.
- Using ACTS, enter the parameters from the classification tree and generate the Test specification for a 2-way combination.
- Filter out the infeasible combinations through constraints.
- Generate the test cases for the given test specification.
- If the requirement is of high criticality, then generate the control flow graph from the code artifact using "Control Flow Graph Factory" tool.
- Using the CFG, calculate the number of basis paths and generate the test cases to traverse all the basis paths.
- Execute the test cases and record the pass rate.
- Correct the defects and re-run the test cases until all the test cases have passed.

Measure of Adequacy:

- All the test cases generated from the test specification have passed for the given code artifact.
- For high criticality requirements, the generated test cases must ensure 100% feasible basis path coverage.

Tools:

- Testona is used to generate the classification tree.
- ACTS is used to generate the Test Specification.
- The test cases are executed manually.

Responsibilities:
The unit testing is performed by the developer who wrote the given code module.

Justification:

- The Unit testing is applicable for all the requirements with any criticality as this ensures the functional correctness of the written code early in the development process.
- We suggest 2-way combinatorial testing instead of 3-way or higher due to the budget and time constraints which leaves limited time for testing.

Integration Testing:

Entry Criteria:

- The specific code modules to be tested have passed unit testing.
- Interface documentation which shows how the two components which are to be integrated communicate with each other.

Exit Criteria:

All the generated test cases have been executed.

Testing Approach:

- Here the data flow between the components being integrated are to be tested.
- Generate the test cases to test the communication between two components.
- Execute those test cases and verify if the communication between the components is what is expected. For example: When integrating a front-end component with a service component, verify if the data which is passed from the front end to the service is in the expected format. Also the communication between the service and the database i.e. what is finally being stored or retrieved from the database is to be verified.

Measure of Adequacy:

- All the test cases which cut across the components being tested have passed for the given code artifacts.

Tools:

- Testona is used to generate the classification tree.
- ACTS is used to generate the Test Specification.
- The test cases are executed manually.

Responsibilities:

This is to be performed by the developer. He has to make sure that the coded interfaces work as expected.

System Testing:

Entry Criteria:

- The acceptance criteria.
- The integrated system which needs to be tested should be completed.
- Integration testing has been passed.

Exit Criteria:

The functionality as described by the acceptance criteria in the form of use cases have been executed.

Testing Approach:

This will be a black box test to test the functionality of the system. Walkthrough the use cases present and execute them in the system.

Measure of Adequacy:
The requirements which were specified as part of the acceptance criteria are satisfied.

Tools:

- Testona is used to generate the classification tree.
- ACTS is used to generate the Test Specification.
- The test cases are executed manually.

Responsibilities:
This testing should be performed by the tester.

Justifications:
Here we suggest a different person to perform the test in order to avoid any biases which the developer may have. This prevents complacency of the developer from setting in.

2. **Manage Models:**

Modules pertaining to the use case:

Use Case	UI Modules	Service Modules	Database Modules
Manage Models	EntityManager LinkManager PropertiesManager RolesManager UserManager RightClickPane NotificationPane	ProjectListService SaveProjectService RetrieveStdEntityService FetchModelService SaveModelService	Entity collection Project collection Role Collection

Types of testing to be performed, Entry Criteria, Exit Criteria, Testing Approach, Measure of adequacy, Tools, Responsibilities and Justification
Whatever is mentioned above for Manage Instances holds true for this use case.

Soak Testing:

Entry Criteria:

- The acceptance criteria.
- The integrated system which needs to be tested should be completed.
- Integration testing has been passed.
- Knowledge on how to use Jmeter (this is the tool suggested by us)

Exit Criteria:

Test report generation and analysis are complete.

Testing Approach:

- Determine the workflow of a normal user of the system.
- Using this estimate the number and type of requests made per user session.
- Estimate the number of users on normal usage hours per day (or take it from the acceptance criteria if present).
- Feed this information to jmeter and execute the tests constituting this workflow for an extended period of time(say 8 hours, again get this from the acceptance criteria if present).
- Generate a report
- Analyze the report to ensure that it meets the acceptance criteria.

Measure of Adequacy:

- System is able to perform without issues for the estimated number of users using the system for a prolonged period of time.
- The requirements which were specified as part of the acceptance criteria are satisfied.

Tools:

- JMeter is the suggested tool for Soak Testing.

Responsibilities:

This testing should be performed by the tester.

Justifications:

We suggest Jmeter because it is the only open source tool available free of cost.

Performance Testing:

Entry Criteria:

- The acceptance criteria.
- The integrated system which needs to be tested should be completed.
- Integration testing has been passed.
- Knowledge on how to use Profiler (this is the tool suggested by us)"

Exit Criteria:

Profiling is complete and bottlenecks if any are identified.

Testing Approach:

- Using the profiler, check whether all the relevant elements are loaded within the time specified by the acceptance criteria.
- If the elements are not loaded within the specified time, identify those elements which are taking time more time to load and flag them as the bottleneck which need to be improved."

Measure of Adequacy:

System must be able to perform that particular activity within estimated time explicitly stated in the acceptance criteria.

Tools:

Chrome Profiler is the suggested tool for Performance testing.

Responsibilities:

This testing should be performed by the tester.

Justifications:

We suggest Profiler because it is the easily available.