Software Quality Assurance Plan

for

OpenADR 2.0 Multiple Transport Reference Architecture

Version 1.1.0

Prepared by the Argonauts Team

Carnegie Mellon University

Master of Software Engineering Program

03rd of May 2012

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Reviewer | Description |
| 1.0.0 | 26 Apr 2012 | Matt Lenzo | Team | Initial project SQAP outline |
| 1.1.0 | 03 May 2012 | Team | Team | QA activity estimates included |
| 1.2.0 | 07 Jun 2012 |  |  | Design review checklist  Coding standards document |

Table of Contents

1. Introduction 4

1.1. Purpose 4

1.2. Scope 4

1.3. Applicable References 4

1.4. Acronyms and Definitions 4

2. Quality Goals 4

2.1. Quality Model 4

2.1.1. Product Quality Attributes 5

3. QA Activities for Requirements Artifacts 7

3.1. Requirement Artifacts 7

3.2. QA Activities for ADS 7

4. QA Activities for Design Artifacts 8

4.1. Design Artifacts 8

4.2. QA Activities for ADD 8

4.3. QA Activities for Detailed Design Documents 9

5. QA Activities for Implementation Artifacts 10

5.1. Implementation Artifacts 10

5.2. QA Activities for Production Code 10

5.3. QA Activities for Deployment Guide 12

6. QA Activities for Test Artifacts 12

6.1. Test Artifacts 12

6.2. QA Activities for Test Cases 13

6.3. QA Activities for Test Plans 13

7. QA Activities for Core Processes 14

7.1. Processes 14

7.2. QA Activities Common to all Processes 14

7.3. QA Activities for Stakeholder Management 15

7.4. QA Activities for Planning and Tracking 15

7.5. QA Activities for Software Configuration Management 16

7.6. QA Activities for Software Quality Assurance 16

8. Management 17

8.1. SQA Responsibilities 17

8.2. SQAP Implementation 17

# Introduction

## Purpose

The Software Quality Assurance Plan (SQAP) defines the quality assurance responsibilities, processes, and measures to be used in the OpenADR 2.0 Multiple Transport Reference Architecture project. All hours and budgeted estimates for Quality Assurance activities are for the summer semester only.

## Scope

This plan applies to all artifacts produced and processes instantiated over the course of the project. In particular, the following artifacts are key deliverables that require careful consideration in the quality assurance plan:

* Architecture Drivers Specification (ADS)
* Architecture Design Document (ADD)
* Reference Implementation (RI)

## Applicable References

* IEEE Std. 1012-1998: IEEE Standard for Software Verification and Validation
* IEEE Std. 1044-2009: IEEE Standard Classification for Software Anomalies
* IEEE Std. 1061-1998: IEEE Standard for a Software Quality Metrics Methodology

## Acronyms and Definitions

* Architecture Drivers Specification (ADS)
* Architecture Design Evaluation Workshop (ADEW)
* Architecture Design Document (ADD)
* Reference Implementation (RI)
* Open Automated Demand Response (OpenADR)
* Software Configuration Management Plan (SCMP)
* Software Quality Assurance Plan (SQAP)
* Iteration (Iter.)
* Week (wk)
* Hours (hrs)
* Quality Assurance (QA)
* Development Lead (Dev. Lead)
* Virtual Top Node (VTN)
* Planning Manager (PM)
* Phase Containment Effectiveness (PCE)

# Quality Goals

## Quality Model

As shown in Figure 1, the quality model for this project can be broken down into considerations of product and process quality. Product quality can be measured against both intermediate and final artifacts used in the production of the aforementioned key deliverables; such artifacts might include documents, code, or test cases. On the other hand, process quality can be assessed through artifacts generated as a result of an internal organizational process; such artifacts might include plans, process proposals, checklists, or other communication media.

The quality of key deliverables enumerated in Section 1.2 can be specified through relevant quality attribute characteristics. Product quality attributes have been identified through requirements elicitation techniques such as the Quality Attribute Workshop and Architecture Drivers Elicitation Workshop. Section 2.1.1 describes these product attributes in detail. Other product quality attributes have been derived from the business context and intended use of these key deliverables. Development processes are characterized by Suitability, Evolvability, Manageability, and Efficiency. Section 2.1.2 describes these process attributes in detail.

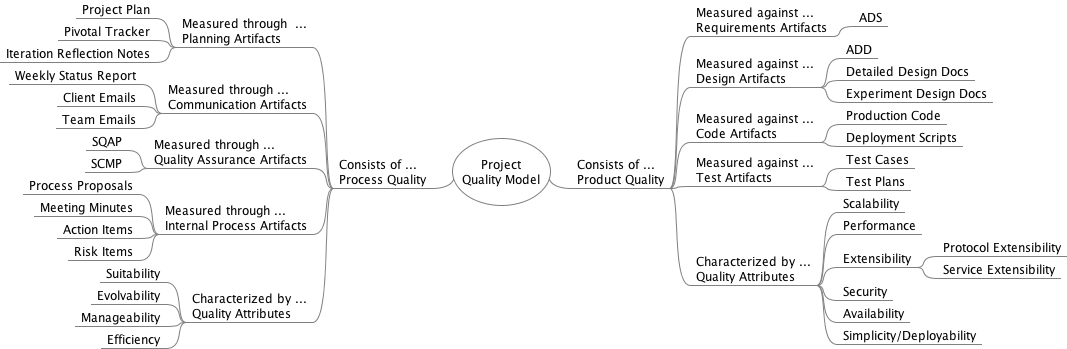


Figure : Project Quality Model

### Product Quality Attributes

Table 1 provides a summary of quality attribute requirements identified during Quality Attribute and Architecture Drivers workshops over the course of the project. While this table only provides a brief description of each quality attribute requirement, full six-part scenarios can be found in the Architecture Drivers Specification (ADS). The client has prioritized each scenario according to its relative importance to the project and business objectives.

|  |  |  |  |
| --- | --- | --- | --- |
| QA# | Priority | Quality Attribute | Description |
| QA01 | High | **Scalability**  ADS Section 6.1.1 | As the number of VENs or end nodes in the DR network increases, the VTN utilization will increase. The VTN should be able to gracefully handle this increased load with no degradation in performance (QA03, QA04, QA05) and no code-level modifications. |
| QA06 | High | **Protocol Extensibility**  ADS Section 6.3.1 | A vendor wants to support an additional transport protocol. The development effort for complying with the provided interfaces and adding support for a new protocol should not exceed one person-day. |
| QA07 | High | **Service Extensibility**  ADS Section 6.3.2 | Services defined in the OpenADR 2.0 profile have changed. The development effort for complying with the provided interfaces and adding support for a new service should not exceed one person-day. |
| QA03 | Medium | **Performance (Outbound Latency)**  ADS Section 6.2.1 | Outbound message latency is measured from the time an event is initiated from the Application Services tier to the time it exits the Transport Services tier. For any one message, the total outbound latency should not exceed 500ms. |
| QA04 | Medium | **Performance**  **(Inbound Latency)**  ADS Section 6.2.2 | Inbound message latency is measured from the time a message or request is received by the Transport Services tier to the time it is processed by the Persistence Tier. For any one message, the total inbound latency should not exceed 500ms. |
| QA05 | Medium | **Performance**  **(Throughput)**  ADS Section 6.2.3 | For both inbound and outbound communication paths, the VTN should be able to process 200 messages or requests per second. That is, when distributing events to VENs, the VTN should be able to sustain a burst throughput of 200 messages per second. When receiving asynchronous requests or responses, the VTN should be able to sustain 200 transactions per second. |
| QA10 | Medium | **Deployability**  **(Cloud Launch)**  ADS Section 6.4.1 | An administrator, familiar with a public or private cloud infrastructure, should be able to launch the VTN reference implementation and initiate a DR event in less than 20 minutes. |
| QA14 | Medium | **Availability**  **(Fault Detection)**  ADS Section 6.6.1 | A fault occurs; faults have been defined using an appropriate fault model. The VTN detects the fault and notifies systems administrators within one minute using either Email or SMS. Fault information contains sufficient detail such that administrators can take corrective action. |
| QA12 | Low | **Security (Confidentiality)**  ADS Section 6.5.1 | An attacker wishes to disrupt or manipulate a DR network in order to interrupt commercial or industrial operations or potentially damage equipment. The VTN does not respond to any illegitimate requests from unrecognized VENs or rogue applications. |
| QA13 | Low | **Availability**  **(Failure Recovery)**  ADS Section 6.6.2 | A failure occurs; failures are traceable to a specific system fault. Manual recovery of any one module can be completed in less than 5 minutes. In the event of failure, no messages are lost in either the outbound or inbound message paths. |

Table : Summary of Quality Attribute Requirements

# QA Activities for Requirements Artifacts

## Requirement Artifacts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact | Inspections | Analysis | Testing | Demonstration |
| ADS | * Requirements Review |  |  |  |

## QA Activities for ADS

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost -Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Review  (35) | Requirements Engineer | After each requirements analysis cycle | Requirements review checklist. This will ask questions such as, Does requirements contain open statements? Are requirements testable? etc.. | **Setup:**  = 5  - Checklist | **Review Cost**:  2 Hrs. /person  **Total Cost:**  \*5 persons  \*3 reviews  = 30 | [Issue Tracker]  [Schema]  [Checklist] | **Metrics:**  - PCE (Reqs)  - Review Rate  - Defect Rate  **Measure:**  - Review Cost  - Artifact Size  - Defects Found | HIGH |

# QA Activities for Design Artifacts

## Design Artifacts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact | Inspections | Analysis | Testing | Demonstration |
| ADD | * Peer Review * ADEW | * Modeling |  |  |
| Detailed Design Docs | * Design Review |  |  |  |
| Experiment Plans | * Approval Review |  |  |  |

## QA Activities for ADD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Peer Review (18) | Chief Architect | Before design task is moved into **Done** swim-lane on task board | Architecture design review pre-ADEW checklist. | None | **Review Cost:**  1.5 Hrs. /design (max)  **Total Cost:**  \*3 concurrent designs  \*4 design iterations  = 18 | [Checklist] | No assessment – Peer reviews are conducted as a pre-requisite activity for ADEW. Process is light to encourage compliance.  Reviewer name included in all check-in comments. | HIGH |
| ADEW  (60) | QA Lead | At the end of each **Iteration** | Architecture design review checklist | None | **Review Cost:**  3 Hrs. /person  **Total Cost:**  \*5 persons  \* 4 design iterations  = 60. | [Issue Tracker]  [Schema]  [Checklist] | **Metrics:**  - PCE (ADesign)  - Review Rate  - Defect Rate  **Measure:**  - Review Cost  - Artifact Size  - Defects Found | HIGH |
| Modeling  (72) | Chief Scientist | Once per iteration if identified as necessary after issue analysis | Reduce the uncertainty based on the model identified for one particular problem | **Setup:**  6 Hrs. /tool  **Total Cost:**  \*2 tools  \*2 persons  = 24 | **Modeling Cost:**  8 Hrs. /person  **Total Cost:**  \*2 persons  \*2 models  = 48 | [Modeling tools]  [Spreadsheets for calculations] [JMT] |  | MED |
| Experiment Proposals and Approval  (23) | Chief Scientist | Prior to **Planning Meeting** | Experiment-approval checklist; ensure entry-exit criteria, ensure issue traceability, ensure scope; signature from CS | **Setup:**  = 3  - Checklist  - Proposal Form | **Approval Cost:**  0.5 Hrs. /review  **Closeout Cost:**  0.5 Hrs. /review  **Total Cost:**  \*20 experiments  = 20 | [Checklist] [Plan Form] |  | MED |

## QA Activities for Detailed Design Documents

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Design Review  (80) | Quality Engineer | Prior to initiating a Coding WP | Review is conducted on a documented detailed design (UML). Review should focus on how accurately it fits into the architecture and how accurately it satisfies the intended system feature. | **Setup:**  = 5 | **Review Cost:**  12.5 Hrs. /Iter.  **Total Cost:**  \*6 Iter.  = 75 | [Checklist]  [Schema]  [Issue Tracker] | **Metrics:**  - PCE (DDesign)  - Review Rate  - Defect Rate  **Measure:**  - Review Cost  - Artifact Size  - Defects Found | HIGH |

# QA Activities for Implementation Artifacts

## Implementation Artifacts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact | Inspections | Analysis | Testing | Demonstration |
| Production Code | * Peer Review * Fagan Inspection | * Static Analysis * Dynamic Analysis * DSM | * Smoke Test * Unit Testing * Integration Testing * System Testing |  |
| Deployment Guide |  |  |  | * Deployment Demonstration |

## QA Activities for Production Code

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Smoke Test  (18 hrs) | QA Lead | Before each release; tests are incorporated into CI tools | Tester identifies applicable test cases for each release and configures the smoke test | None | **Test Cases:**  1 hr/Iter.  **Verification:**  2hr/Iter.  **Total Cost:**  \*6 iter.  = 18 hrs | [CI Server]  [Scripting]  [Schema]  [Issue Tracker] | **Metrics:**  - PCE (Testing)  **Measure:**  - Defects Found | HIGH |
| Static Analysis  (58 hrs) | QA Lead | Continuously | SA tool installed and configured for each workstation; Daily build produces reports that should be reviewed for issues | **Setup:**  = 8 hrs  - Install  - Configure  - Tweak | **Aggregate Cost:**  1 hr/day  **Total Cost:**  \*50 days  = 50 hrs | [PMD]  [Findbugs]  [Schema]  [Issue Tracker] | **Metrics:**  - PCE (Testing)  - True Positives  - False Positives  **Measure:**  - Defects Found  - False Positives | MED |
| Dynamic Analysis  (32 hrs) | QA Lead | As needed (targeted) | Targeted testing for problems associated with concurrency or memory leaks | **Setup:**  = 8 hrs  - Install  - Configure  - Tweak | **Budgeted Cost:**  2 hrs/week  **Total Cost:**  \*12 weeks  = 24 hrs | [TBD] | [TBD] | MED |
| Peer Review  (102 hrs) | Dev. Lead | Before Coding task is movedinto **Done** swim-lane on task board | Peer reviews code prior to check-in; reviewer name included in check-in comments | **Setup:**  = 2 hrs  - Checklist | **Review Cost:**  2 hrs/day  **Total Cost:** \*50 days  = 100 hrs | [Checklist] | No assessment – Peer reviews are conducted as a pre-requisite activity for further testing. Process is light to encourage compliance.  Reviewer name included in all check-in comments | HIGH |
| Fagan Inspection  (124 hrs) | QA Lead | Once per iteration prior to release testing | 1. Planning 2. Preparation 3. Inspection 4. Follow-up | **Setup:**  = 4 hrs  - Checklist  - Defect Schema | **Planning:**  30 min/person  **Prep/Inspection**:  150 LOC/hr (300 LOC)  100 LOC/hr (300 LOC)  3.5 hrs/person  Total  \*5 persons  \*6 Iter.  = 120 hrs | [Checklist]  [Schema]  [Issue Tracker] | **Metrics:**  - PCE (Coding) - Review Rate  - Defect Rate  **Measure:**  - Review Cost  - Artifact Size  - Defects Found | MED |
| Unit Testing  (380 hrs) | QA Lead | Continuously | All code modules must have accompanying unit test cases | **Setup:**  = 5 hrs  - JUnit | **Aggregate Cost:**  1.5 hrs/person/day  **Total Cost:**  \*5 persons  \*50 days  = 375 hrs | [JUnit]  [Schema]  [Issue Tracker] | **Metrics:**  - PCE (Testing)  **Measure:**  - Defects Found | HIGH |
| Integration Testing  (20 hrs) | QA Lead | Continuously and before each release; tests are incorporated into CI tools | Integration tests designed and incorporated into CI tools | **Setup:**  = 8 hrs | **Test Cases:**  2 hrs/Iter.  **Total Cost:**  \*6 Iter.  = 12 hrs | [CI Server]  [Scripting]  [Schema]  [Issue Tracker] | **Metrics:**  - PCE (Testing)  **Measure:**  - Defects Found | HIGH |
| System Testing  (50 hrs) | QA Lead | As scheduled | Complete system and architecture is verified against its key QAs and functional requirements | None | **Test Planning:** 3 hrs/test  **Test Design:**  10 hrs/test  **Test Execution:**  12 hrs/test  **Total Cost:**  \*2 system tests  = 50 hrs | [Schema]  [Issue Tracker] | **Metrics:**  - PCE (Testing)  **Measure:**  - Defects Found | HIGH |
| Design Structure Matrix (DSM)  (12 hrs) | Chief Architect | Weekly | Architect checks current build for conformance to architectural design | **Setup:**  = 6 hrs  - Configuration | **Conformance Check:** 30 min/week  **Total Cost:**  \*12 weeks  = 6 hrs | [Lattix]  [Issue Tracker]  [Schema] | **Metrics:**  - PCE (Coding)  **Measure:**  - Defects Found |  |

## QA Activities for Deployment Guide

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Deployment Demonstration  (14 hrs) | Dev. Lead | Whenever the deployment guide is updated; before each release | Tester walks through guide step-by-step and ensures instructions are accurate, accessible, and understandable | **Setup:**  = 2 hrs  - Checklist | **Review Cost:**  2 hrs/update.  **Total Cost:**  \*1 update/Iter. \*6 Iter.  = 12 hrs | [Checklist]  [Issue Tracker]  [Schema] | **Metrics:**  - PCE (Testing)  **Measure:**  - Defects Found | HIGH |

# QA Activities for Test Artifacts

## Test Artifacts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Artifact | Inspections | Analysis | Testing | Demonstration |
| Test Cases |  | * Test Coverage * Mutation Analysis |  |  |
| Test Plans | * Test Approval |  |  |  |

## QA Activities for Test Cases

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Test Coverage and Mutation Analysis  (18 hrs) | QA Lead | Continuously through continuous integration environment | Automated daily "build and smoke test" will generate code coverage reports; this information will be used to assess the quality of test cases, QA Engineer and PM will use these reports to make decisions on test resource allocation | **Setup:**  = 6hrs | **Report Analysis:**  15min/day  **Total Cost:**  \*48 days  = 12 hrs | [CI Server] [Cobertura Plugin]  [Maven] | - Statement coverage  - Mutation score | HIGH |

## QA Activities for Test Plans

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Test Approval  (13 hrs) | QA Lead | Prior to initiating a Verification WP | Test plan checklist: Does plan explain what is under test, approach, report format, resources, risks, cost; QA and CA sign off | **Setup:**  = 5hrs | **Review Cost:**  1 hr/plan  **Total Cost:** \*8 tests  = 8 hrs | [Checklist]  [Test Form] | # tests within estimated time cost; # test incidents (e.g. go over free web tier rates, break something, etc.) | HIGH |

# QA Activities for Core Processes

## Processes

|  |  |  |
| --- | --- | --- |
| Process | Relevant Artifacts | Activity |
| Stakeholder Management | Client Communication Logs | * Scorecard Assessment * Meeting Review |
| Software Cfg. Management | SCMP | * Scorecard Assessment * SCM Audit * SCMP Review |
| Software Quality Assurance | SQAP | * Scorecard Assessment * SQA Audit * SQAP Review |
| Planning and Tracking | Project Plans  Work Packages (WP) | * Scorecard Assessment * Entry/Exit Review * Data Analysis |
| Requirements Management | Problem Definition Proposal | * Scorecard Assessment |
| Risk Management | Risk Management Proposal | * Scorecard Assessment |
| Design Process | Design Proposal | * Scorecard Assessment |

## QA Activities Common to all Processes

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Scorecard Assessment  (12 hrs) | QA Lead | Before weekly status meeting | QA Lead polls each team member on key process areas and captures their responses in a scorecard | **Setup:**  = 4 hrs  - Scorecard | **Capture Cost:**  20min/week  **Analysis Cost:**  20min/week  **Total Cost:**  \*12 weeks  = 8 hrs | [Scorecard] | None | HIGH |

## QA Activities for Stakeholder Management

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Meeting Review  (7 hrs) | Team Lead | Before and after each client meeting | Team Lead uses checklist to ensure agenda is published, notes collected, action items assigned | **Setup:**  = 1 hr  - Checklist | **Review Cost:**  30min/meeting  **Total Cost:**  \*1 meeting/week  \*12 weeks  = 6 hrs | [Checklist] | None | HIGH |

## QA Activities for Planning and Tracking

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| Entry/Exit Review  (124 hrs) | PM | Before any task is moved into **WIP** swim-lane on task board  Before any task is moved into **Done** swim-lane on task board | PM uses a checklist to ensure that a WP has ETVX, owner, and is sufficiently broken-down  **[See Section 9.0]** | **Setup:**  = 2 hrs  - Checklist | **Start Cost:**  15min/WP  **Closeout Cost:**  15min/WP  **Total Cost:**  \*40WP/Iter.  \*6 Iter.  = 122 hrs | [Checklist] | None | **HIGH** |
| Data Analysis  (28 hrs) | PM | For each WP and task, members must log their time;  PM will analyze data at end of each iteration, before reflection meeting | **Metrics:**  - Earned Value  - Estimation Acc.  **Measure:** | **Setup:**  = 16 hrs  - Scripts  - Record | **Capture Cost:**  1hr/Iter.  **Analysis Cost:**  1hr/Iter.  **Total Cost:**  \*6 Iter.  = 12 hrs | [Toggl Tracker]  [Automated Scripts]  [Basecamp Data] | None | **HIGH** |

## QA Activities for Software Configuration Management

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| SCM Audit  (8 hrs) | QA Lead | End of each iteration, before reflection meeting | QA uses checklist and SCMP to identify non-compliance issues and record on Argo Issue Tracker | **Setup:**  = 2 hrs  - Checklist | **Review Cost:**  1hr/Iter.  **Total Cost:**  \*6 Iter.  = 6 hrs | [Checklist]  [Issue Tracker] | None | **MED** |
| SCMP Review  (5 hrs) | Dev. Lead | After each major update to the SCMP | Team reviews the SCMP for discrepancies, errors, or omissions, and approves | None | **Review Cost:**  0.5hr/person  **Total Cost:**  \*5 persons  \*2 updates (assm.)  = 5 hrs | None | None | **HIGH** |

## QA Activities for Software Quality Assurance

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Activity  (Total Cost - Hrs.) | Responsible  Engineer | When | Extent/Method | Upfront  Cost (Hrs.) | Recurring  Cost (Hrs.) | Tools  Training | Assessment | Priority |
| SQA Audit  (8 hrs) | QA Lead | End of every iteration, before reflection meeting | QA uses checklist and SQAP to identify non-compliance issues and record on Argo Issue Tracker | **Setup:**  = 2 hrs  - Checklist | **Review Cost:**  1hr/Iter.  **Total Cost:**  \*6 Iter.  = 6 hrs | [Checklist]  [Issue Tracker] | None | **MED** |
| SQAP Review  (5 hrs) | QA Lead | After each major update to the SQAP | Team reviews the SQAP for discrepancies, errors, or omissions, and approves | None | **Review Cost:**  0.5hr/person  **Total Cost:**  \*5 persons  \*2 updates (assm.)  = 5 hrs | None | None | **HIGH** |

# Management

## SQA Responsibilities

Software Quality Assurance is a responsibility of all team members. The Quality Assurance Lead plays a pivotal role in ensuring compliance with the activities outlined in this document as well as maintaining an appropriate QA budget with the help of the PM.

## SQAP Implementation

Major milestones associated with implementation of this SQAP include:

* M01: All budgeted QA activities have been accounted for in the project schedule
* M02: QA Activities for core processes identified in Section 7 have been instantiated
* M03: QA Activities for requirements artifacts identified in Section 3 have been instantiated
* M04: QA Activities for design artifacts identified in Section 4 have been instantiated
* M05: QA Activities for test artifacts identified in Section 6 have been instantiated
* M06: QA Activities for implementation artifacts identified in Section 5 have been instantiated

# ETVX Checklist for Tasks

The following checklist shall be used before a task can be moved from **Backlog** to **Work in Progress**. Entry, Task, Verification, and Exit criteria should be checked before starting work on this task and verified before completing the task.

## Entry

*What criteria must be met before this task can be started?*

* Does this task have any prerequisite tasks or dependencies?
* Are required tools and/or resources available?
* Does this task contain an estimate in person-hours?
* Does this task contain a unique tracking number and tag?
* Are there higher priority tasks that should be completed first?

## Task

*What constitutes “Done” for this task?*

* Are output criteria and artifacts clearly defined?
* Where are output artifacts maintained?
* Are quality checklists applicable to this task? (e.g., Design Checklist)

## Verification

*By what means can we determine that this task is “Done right”?*

* *Does this task require an additional QA activity such as “Peer Review?”*

## Exit

*What are the outputs of this task?*

* *Are all output criteria met?*
* *Are artifacts checked-in and versioned according to the SCMP?*

Note: If a task is rejected in Quality Check, it should receive one strike against it. Upon three strikes, the task should be reviewed by the Planning Manager to ensure that the ETVX criteria is still appropriate for the task.

# Requirements Review Checklist

The following checklist shall be used for requirements reviews; specifically, reviews of the Architecture Drivers Specification.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Documentation Standards | |  | Consistency and Clarity | |
| 1. | Have consistent documentation standards been followed? |  | **13.** | Does the ADS agree with any higher-level specifications? |
| 2. | Are all figures and tables labeled and referenced? |  | **14.** | Are requirements free of duplication or conflict? |
| 3. | Are all acronyms, units, and measures clearly defined? |  | **15.** | Is each requirement written in clear, precise, and unambiguous English? |
| 4. | Are all requirements written consistently and at the same level of detail? |  | **16.** | Does each requirement have only one interpretation? |
| 5. | Are all architectural drivers ranked according to priority? |  | **17.** | Is each requirement verifiable through testing, demonstration, review, or analysis? |
| 6. | Are requirements written to support downstream design and test activities? |  | **18.** | Are there measureable verification criteria for both functional and quality attribute requirements? |
| Completeness and Correctness | |  | Traceability | |
| 7. | Are cross-references to other architectural drivers minimized? |  | **19.** | Is each requirement uniquely and correctly identified (e.g, UC##, BC##, TC##, QA##) |
| 8. | Are all entities (stakeholders or users) clearly defined? |  | **20.** | Is each requirement traceable to its source (e.g, client discussion, specification, whitepaper, or workshop) |
| 9. | Are there any known architectural drivers not captured? |  | Other | |
| 10. | Does each use case clearly indicate pre and post conditions? |  | **21.** | Is each requirement in-scope for this project? |
| 11. | Are business and technical constraints properly specified? |  | **22.** | Are all requirements free from language specific to design and implementation solutions? |
| 12. | Are quality attribute requirements properly specified? |  |  |  |
| 12. | Has the full lifecycle support been specified, including maintenance? |  |  |  |

# Design Review Checklist

The following checklist shall be used for design reviews; specifically, reviews of any detailed design artifacts that are generated prior to implementation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Functional Design | |  | Consistency and Clarity | |
| 1. | Does each entity have a single and clearly defined purpose?   * Do all methods perform one and only one action? * Is abstraction necessary for future modifiability? * Is the entity as simple as possible, but not simpler? |  | **8.** | Does the design follow known design patterns?   * Are design patterns used correctly? * Are there more appropriate design patterns? * Are design patterns used cited for reference? |
| 2. | Are entities loosely coupled with tight cohesion?   * Do entities have little knowledge of other entities? * Do entities only reference information required for their purpose? * Is there architectural justification for tight coupling? |  | **9.** | Are entities and relationships appropriately named?   * Do class names follow language conventions? * Do interface names follow language conventions? * Do methods, properties, and accessors follow language conventions? |
| 3. | Are all required inputs provided and specified outputs produced for each entity? |  | **10.** | Are designs captured using appropriate tools or notation such that the design can be communicated to other developers? |
| Architectural Conformance | |  | Traceability | |
| 4. | Does the design conform to the static architecture?   * Are “uses” relationships correct? * Are “allowed to use” relationships correct? * Are classes allocated to appropriate packages? |  | **11.** | Is each design entity traceable to a functional or quality attribute requirement? |
| 5. | Does the design inhibit any key quality attributes?   * Performance? * Scalability? * Reliability? |  | **12.** | Has the design made any undocumented architectural decisions? |
| Defensive Design | |  | Other | |
| 6. | Does the design account for error conditions and exceptions? |  |  |  |
| 7. | If the design is for an API, is the API designed such that it is easy to use correctly and difficult to use incorrectly? |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Defect Classification Schema

## GitHub Issue Tracker

GitHub Issue Tracker shall be used to maintain a list of defects across the project. Defects shall be categorized according to the Source or Origin of the error/defect, the Phase or Activity in which the error/defect was discovered, as well as the Type of defect with description.

### Origin Tags

The following are valid tags to indicate the origin of a defect. That is, where and when this defect was injected.

* **Requirements Defect**
* **Architecture Defect**
* **Documentation Defect**
* **Design Defect**
* **Code Defect**
* **Test Defect**

### Activity Tags

The following are valid tags to indicate how a defect was discovered or removed.

* **Architecture Design**
* **ADEW**
* **Experimentation**
* **Detailed Design**
* **Design Review**
* **Coding**
* **Code Review**
* **Testing**

### Defect Type

Defect type information is maintained in the issue title. The format of this title is as follows:

MODULE NAME: Class\_Name/TYPE, where MODULE NAME is the defective module, and TYPE is a description of the defect type. The following are valid defect types. In general, keep the Type as short as possible and defer details to the description field. As an alternative, you may simply use the ClassName of the file under question.

**Example Design Defect Types**

* **Scalability** – An architecture design defect pertaining to scalability design
* **Service Extensibility** – An architecture design defect pertaining to service extensibility design
* **Protocol Extensibility** – An architecture design defect pertaining to protocol extensibility design
* **Availability** – An architecture design defect pertaining to availability design
* **Performance** – An architecture design defect pertaining to performance design
* **Security** – An architecture design defect pertaining to security design
* **Deployability** – An architecture design defect pertaining to deployability design

**Example Code Defect Types**

* **Documentation**: Problems with comments or messages provided to the user
* **Syntax**: Spelling, punctuation, typos, or improper instructions
* **Interface:** Method and procedure calls or references
* **Checking**: Boundary condition checking, input validation, and error messages
* **Data:** Data structures, content, and problems with data model implementation
* **Functional**: Logic problems, loops, recursion, computational, or improper implementation
* **Environment:** Design, test, or other support system problems such as version control, build, or packaging

### Defect Description

A description of the defect should provide more information about its source, trigger (cause to reproduce), or explanation.

# GQM: Team and Project Goals

## Team Goal #1 – Sustainable Pace

### Goal Statement

To analyze resource usage for the purpose of ensuring a sustainable work environment with respect to person-hours from the viewpoint of the individual developer in the context of Summer 2012.

### Questions

* Are team members working their required 48 hours per week?
* Is work distributed evenly and are team members utilized equally?
* Are team members working a minimum number of hours per day to meet the 48-hour requirement?

### Metrics

* Resource Utilization (0 – 100%) per resource per day
* Resource Burndown (48 hours – 0 hours) per week

### Reporting and Implementation

* QA Lead will send out a report of the current resource utilization by EOD Monday and EOD Wednesday and EOD Friday
* Toggl shall be used for time data collection
* Automated scripts shall be used for report generation

## Team Goal #2 – Pull Together

### Goal Statement

To assess team cohesion for the purpose of maximizing productivity and happiness with respect to project and team factors from the viewpoint of the entire team in the context of Summer 2012.

### Questions

* What is the team’s attachment to its project?
* What is the team’s social connection?
* What is each individual’s attachment to the project?
* What is each individual’s attachment to the group?

### Metrics

* 14-Point Group Environment Questionnaire

### Reporting and Implementation

* Team takes questionnaire at the end of each iteration prior to Iteration Reflection

## Project Goal #1 – Reach Period of Certainty

### Goal Statement

To assess confidence in the architecture design for the purpose of meeting project objectives with respect to outstanding issues from the viewpoint of the Chief Architect and client by end of Summer.

### Questions

* Are all systemic requirements satisfied through a design?
* How many outstanding issues are associated with high-priority quality attribute requirements?
* How many outstanding issues are associated with low and medium-priority quality attribute requirements?
* At the current rate of closing issues, will the team reach its milestone?

### Metrics

* Number of outstanding Architectural Issues
* Mean Age of outstanding Architectural Issues (Days)

### Reporting and Implementation

* Team evaluates metrics weekly during each status meeting
* Metrics should provide some insight on whether risk mitigation is required

## Project Goal #2 – Complete Core Reference Implementation

### Goal Statement

To evaluate the core reference implementation for the purpose of ensuring architectural conformance with respect to key quality attribute requirements from the perspective of the QA Lead in the context of Summer 2012.

### Questions

* To what extent does the reference implementation deviate from its architecture from a static perspective?
* To what extent can the team ensure that code-level defects do not exist in the reference implementation?
* To what extent does the reference implementation demonstrate its intended behavior?

### Metrics

* Phase Containment Effectiveness
* Number of Outstanding Defects

### Reporting and Implementation

* Team evaluates metrics weekly during each status meeting
* Metrics should provide some insight into ability to deliver reference implementation

## Other Goal #1 – Overhead Activities under 30%

### Goal Statement

To analyze time data for the purpose of ensuring overhead activities do not exceed 30% with respect to person-hours from the viewpoint of the team in the context of each Iteration.

### Questions

* Do overhead activities account for more than 30% of the total effort across an Iteration?
* What overhead activities are the major contributors to the overall overhead budget?
* Are automation and process enhancements effective mechanisms for reducing project overhead?

### Metrics

* Percent of iteration effort spent on overhead activities
* Top three overhead activities according to proportion

### Reporting and Implementation

* Team evaluates metrics weekly during each status meeting
* Team evaluates metrics during the iteration reflection meeting

# Coding Standards for Java

## Introduction and Purpose

This section introduces the coding standards that need to be followed and practiced by the developers of the OpenADR 2.0 Multiple Transport Reference Architecture project. Enforcing coding standards has numerous benefits such as improved code readability, enhanced quality, increased modifiability, and ease of maintenance. The following set of coding standards is specific to the Java programming language; which is the programming language selected to implement the Reference Implementation (RI) of the project. Every developer, before writing the production code for the RI, should go through this section in detail, and must follow them as the commandments for coding. Also, every reviewer, during the peer review activity, must ensure that the developer has fully adhered to the coding conventions enlisted in this section of the SQAP. The Quality Assurance Lead/Quality Process Engineer is responsible for overseeing such aforementioned activities. Any non-compliance with the set coding standards should be documented in GitHub issue tracker as a ‘documentation’ or a ‘syntax’ defect, depending upon the nature of the non-compliance. Such defects should be actively tracked by the Quality Assurance Lead during formal inspections, and semi-formal walkthroughs, and resolved by the responsible developer/engineer.

## Commandments for Coding in the Java Programming Language

### Naming Conventions

* **Packages:** All packages should be named in lower case to avoid conflict with the names of classes and interfaces. Packages should be created for a group of classes of related functionality. Consider writing a **package.html** file in each directory outlining the purpose, and structure of the package. Start every package name with **org.mse.argonauts**. An example package name would be ‘package org.mse.argonauts.mypackage.’
* **Classes:** All classes should be named in upper camel case. Try using nouns to name a class, because they usually represent something in the real world. An example class name would be ‘class Party’
* **Interfaces:** All interfaces should be named in upper camel case. All interfaces should begin with the letter ‘I’ to distinguish classes with their interfaces. Interfaces tend to have a name that describes an operation that a class can do. An example interface name would be ‘interface IComparable’
* **Methods:** All methods should be named in mixed case, i.e., lower camel case. Use verbs to describe what the method does. An example method name would be ‘void showSupportedProtocols.’
* **Variables:** All variable should be named in mixed case, i.e., lower camel case. The names should represent what the value of the variable represents. An example variable name would be ‘string firstName.’ It is permissible to use short names for variables that are short lived. Examples of such variables would be the variables used in ‘for’ loops.
* **Constants:** All constants should be named in upper case. An example constant name would be ‘static final string DEFAULT\_PROTOCOL.’

### Documentation and convention comments

* **Java Doc Style:**  The in-built Java Doc style feature of the Java SDK can be used to generate documentation for the project.
* **Block comments:** They should be used to describe the classes, methods, data structures, and algorithms. Block comments should be used at the beginning of each class file, and also before each method, describing the purpose of the class file and the method respectively. The block comment for a method should include the return type and the input parameter types. An example of a block comment would be as follows:

/\*

\* Here is a block comment.

\*/

* **Single-Line Comments:** They should appear on a single line, and should be used to the level of the code that follows. If a comment can’t be written on a single line, it should follow the block comment format. An example of a single line comment would be as follows:

if (condition) {

/\* Handle the condition. \*/

...

}

* **Trailing Comments:** They should be very short comments, and should appear on the same line as the code they describe. However, they should be shifted far enough to separate them from the code statements. More than one trailing comment in a chunk of code should be intended to the same tab settings. An example of a trailing comment would be as follows:

if (z == 2) {

return TRUE; /\* special case \*/

} else {

return FALSE; /\* non-special case \*/

}

* **End-Of-Line Comments:** They are denoted by the ‘//’ delimiter. This delimiter begins a comment that continues to the newline. It shouldn’t be used in consecutive multiple lines for text comments. However, it can be used in consecutive multiple lines for commenting out sections of code. Examples of end-of-line comments would be as follows:

if (numberOfProtocols > 1) {

// Select the default protocol.

...

}

else

return false; // Explain why here.

//if (numberOfProtocols > 1) {

//

// // Select XMPP protocol.

// ...

//}

//else

// return false;

### Other Conventions

* The variables in a bean (entity) class must be private, and have accessors, i.e., getters and setters.
* Use the ‘Ctrl + F’ feature of Eclipse for standard indentation, formatting, and line wrapping.
* Avoid abbreviations in naming packages, classes, interfaces, methods, etc.
* Variables with a larger/global scope should have long names, and variables with a smaller/local scope should have short names.
* *is* prefix should be used for boolean variables and methods. Examples of this include isSet, isOpen, etc.