Software Configuration Management Plan

for

OpenADR 2.0 Multiple Transport Reference Architecture

Version 1.0

Prepared by the Argonauts Team

Carnegie Mellon University

Master of Software Engineering Program

of April 2012

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Reviewer | Description |
| 1.0.0 | 25 Apr 2012 | Matt Lenzo | Team | Initial SCMP to complement SQAP. |

Table of Contents

1. Introduction 4

1.1. Purpose 4

1.2. Scope 4

1.3. References 4

2. Management 4

2.1. SCM Responsibilities 4

2.2. SCMP Implementation 4

3. SCM Activities 5

3.1. Configuration Identification 5

3.1.1. Baseline Definition 5

3.1.2. Document Artifacts 5

3.1.3. Architecture Design Artifacts 6

3.2. Change Control 6

3.2.1. Change Significance (Versioning Scheme) 6

3.2.2. Statement of Work (SOW) 6

3.2.3. Architecture Drivers Specification (ADS) 6

3.2.4. Architecture Design Document (ADD) 7

3.2.5. Reference Implementation (RI) 7

3.2.6. Ancillary Artifacts 7

3.3. Configuration Status Accounting 7

3.4. Audits and Reviews 7

4. SCMP Implementation Guidance and Policies 7

4.1. Tools 7

4.2. Repository Organization 8

4.3. Wiki and GDocs Organization 9

4.3.1. Client Meeting Minutes and Slides 9

# Introduction

## Purpose

The Software Configuration Management Plan (SCMP) defines the configuration management responsibilities and processes to be used for all the deliverables in the OpenADR 2.0 Multiple Transport Reference Architecture project. Primary project deliverables include:

* Project Statement of Work (SOW)
* Architecture Drivers Specification (ADS)
* Architecture Design Document (ADD)
* Reference Implementation (RI)

## Scope

In addition to the primary project deliverables enumerated in Section 1.1, the following ancillary artifacts shall be maintained within the configuration management system:

* Software Configuration Management Plan (SCMP)
* Software Quality Assurance Plan (SQAP)
* Master Design Plan (MDP)
* OASIS and OADR Alliance specifications and working drafts referenced by project deliverables

## References

Futrell, Shafer, et al. *Quality Software Project Management* – Software Quality Institute Series. Prentice Hall Professional, 2002.

# Management

## SCM Responsibilities

For the configuration items referenced in Section 1.0, all team members are responsible for faithfully executing the SCMP. The Support Lead is the primary authority on SCMP implementation, configuration identification, change control, and status accounting activities. The Quality Assurance Lead is the primary authority on all auditing and review activities associated with SCM compliance. The Chief Architect, Quality Assurance Lead, and Requirements Engineer provide key oversight and review functions for project Configuration Items (CIs).

## SCMP Implementation

Major milestones associated with implementation of this SCMP include:

* M01: Identification of all current Configuration Items (CI)
* M02: Establishing a configuration baseline for CIs that are key deliverables
* M03: Establishing a schedule for periodic SCM reviews and audits in accordance with the project SQAP
* M04: Establishing a maintenance schedule for SCM-related development, test, and support tools

# SCM Activities

## Configuration Identification

### Baseline Definition

Project deliverables have dependencies. For instance, a new version of the ADD must reference key architectural drivers in the ADS. Hence, the ADD and the ADS are both items that constitute a configuration baseline. A baseline may include any or all of the other following elements depending on the project’s current life cycle:

* A summary of changes to associated software or documentation [required]
* Architecture Drivers Specification (ADS)
* Architecture Design Document (ADD)
* Relevant architectural experiments
* Relevant detailed design documents
* Reference Implementation code
* Test and evaluation results to include known faults and failures
* Installation and deployment instructions
* User education material

For instance, Project Release 1.0.0 might include several artifacts:

* ADS version 2.0.1
* ADD version 1.2.1
* RI version 1.0.1

### Document Artifacts

Each document under SCM shall have:

1. **Cover Page**: Include the title of the document, project, current version number, preparing team, and date of last revision.
2. **Revision History**: For each revision, describe the revision number, date of revision, responsible author, reviewer name, and a short description of changes in accordance with Table 1.
3. **Standard Naming**: Name each electronic document record as follows: <Name>\_v<Version>[\_<Opt>], where <Name> is the short name of the file (e.g., ADS), <Version> is the change control number explained in Section 3.2, and <Opt> is any optional descriptive word or number that adds semantic meaning to the file name. As an example: ADS\_v1.0\_Spring2012.docx.

Table 1 - Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Reviewer | Description |
|  |  |  |  |  |

### Architecture Design Artifacts

Architecture design artifacts (e.g., views, models, and rationale) may be captured at various levels of fidelity. Since a well-documented architecture is a core project deliverable, this SCMP treats all architectural design artifacts, regardless of fidelity, as CIs. Each architecture design artifact shall have:

1. **Proper Representation**: Whiteboard views may be captured via photograph, but must be legible. Freehand views must use the provided architecture design template and these drawings must be scanned to an appropriate resolution. Views produced using modeling or drawing tools may remain in their native formats.
2. **Standard Naming**: Name each architectural design artifact as follows: <Name>\_v<Version>[\_<Style>]\_<Date>, where <Name> is the short name of the view (e.g., SecurityDesign), <Version> is the change control number explained in Section 3.2, <Style> is an optional flag that describes the view style (permissible styles include Mod, CC, and Alc for module, component/connector, and allocation respectively), and <Date> is the current date in the form YYYYMMDD. As an example: SecurityDesign\_v1.0\_CC\_20120430.png

## Change Control

This section outlines the methods of executing proposed changes to any of the key project deliverables outlined in Section 2 as well as responsibilities of key oversight and approval roles.

### Change Significance (Versioning Scheme)

All project artifacts shall use a common versioning scheme that reflects both maturity and change significance. Specify all artifact versions as follows:

<Major>.<Minor>.<Status>[.<Revision>], where <Major> and <Minor> are arbitrary designators that reflect change significance, <Status> reflects the release status of the artifact (permissible values include 0 => internal use only, 1 => beta, 2 => release), and <Revision> is an optional designator to differentiate minor changes.

### Statement of Work (SOW)

All parties must initiate changes in accordance with the workflow specified in Section 5.0 of the SOW.

### Architecture Drivers Specification (ADS)

The workflow for making changes to the ADS is as follows:

1. Create a new folder under ADS workflow named in accordance with Section 3.1.2. For example, ADS\_v2.0.1.
2. For each section of the ADS that requires changes, create a new file in this folder and specify the affected section in the <Opt> field of the file name. For example, ADS\_v1.9.0\_Section3.2.docx.
3. The Requirements Engineer will review and merge all changes into the new ADS, updating the revision history described in Section 3.1.2. The Requirements Engineer is the final authority for all changes to the ADS.

### Architecture Design Document (ADD)

The workflow for making changes to the ADD is as follows:

1. Create a new folder under ADD workflow named in accordance with Section 3.1.2. For example, ADD\_v1.5.0.
2. For each section of the ADD that requires changes, create a new file in this folder and specify the affected section in the <Opt> field of the file name. For example, ADD\_v1.4.0\_Section10.docx.
3. The Chief Scientist will ensure that all relevant architectural experiments are documented and included as an appendix to the revised ADD.
4. The Chief Architect will review and merge all changes into the new ADD, updating the revision history described in Section 3.1.2. The ADD should also reference its corresponding ADS version. The Chief Architect is the final authority for all changes to the ADD.

### Reference Implementation (RI)

Refer to the Software Quality Assurance Plan (SQAP) for the workflow associated with releasing production code.

### Ancillary Artifacts

Ancillary artifacts such as the SCMP and the SQAP are generally not “collaboratively owned” documents and do not require stringent workflow processes for merging and reviewing changes from multiple team members. As such, the change management authority identified in each of these documents shall ensure SCM compliance in accordance with Section 3.1.2 for documents.

### Source code (Include only production code and unit tests)

Production code and related unit tests is the core of system. This section contains the basic configuration rules for managing the source code. Branching and labeling are described under a separate section.

Standard source code related rules

* The depth of the namespace should not exceed 6 levels. Extending this limit needs everyone’s agreement.
* All namespace names and package names should only contain lower case letters.
* All acronyms/abbreviations can have any number of letters in them, but all should be in lower case
* Unit tests should go into the same package, but under different folders.
* Test should class name should have the word “Test” appended to the class it's testing

## Branching and labeling of source code

The Development Lead is responsible for making sure branching; especially labelings are done according to the policies defined in this document.

### Branching

Following steps explain how the branching has to be done and maintained in the system.

* Development lead creates “remote” branches in the Git for every major component/ layer of the system. Following branches are speculated according to the current architecture of the system
  + Core – Contains all modules for core layer
  + OADR service
  + Dispatcher service
  + Receiver service
  + Persistence service
  + XMPP protocol gateway
  + HTTP protocol gateway
  + AMQP protocol gateway

Note: These remote branches will be created as and when these components are being developed according to the project plan.

* Development lead (or delegated resource) is responsible for merging all remote branches at the end of iteration (or as when planned work of that iteration is completed).
* No developer is allowed to work on the “master” branch. Instead they will work on component branch created by the development lead. However developers are free to create any number of local branches from the remote branch. Developers may checkout different branches as needed, so they can switch and work on different components of the system.

### Labeling

Labeling will be done using “tag” feature of the Git. Development lead (or delegated resource) is responsible for creating relevant label at reaching a particular milestone of the project.

Following labeling scheme will be used while labeling the source code.

<Major>.<Minor>.<Patch>[.<Build>], where:

<Major> - Start at 0 until all components and functionalities are in place. Incremented only upon introduction of a new feature.

<Minor> - Start at 1 with the initial component/ feature of the system. Incremented every time a new component/feature update happens.

<Patch> - Start at 0 and increment only if there is a patch provided for the relevant minor version.

<Build> - Optional, start at 0. Incremented only when there are multiple builds related to the same patch.

## Configuration Status Accounting

The Development Lead shall brief the team on a weekly basis regarding any or all of the following events. No formal report is necessary, but any issues raised during status accounting should be tracked to resolution.

* Status of proposed, current, and completed revisions to CIs
* Status of SCM-related development, test, and support tools
* Status of compliance and process issues associated with SCM implementation

## Audits and Reviews

The Quality Assurance Lead shall conduct all audits and reviews of the SCMP implementation. As explained in Section 2.1, this authority also sets the schedule and frequency of all SCM-related quality assurance activities. Details regarding scope, procedures, issue deposition, and problem resolution are provided in the project SQAP.

# SCMP Implementation Guidance and Policies

This section is intended to convey practical guidance and policies regarding implementation of the SCMP. New processes, tools, or policies should be included in this section of the document.

## Tools

This project makes use of the following SCM and collaboration tools:

* **GitHub Argo Repository**: All primary and ancillary CIs enumerated in Section 1 shall be maintained in the client’s Argo repository. Refer to Section 4.2 for organizational guidance.
* **GitHub Internal Repository**: All academic proposals, meeting notes, and internal process-related documents that are not needed for collaboration with the client shall be maintained in the internal repository. Refer to Section 4.2 for organizational guidance.
* **Dropbox**: Shared links, articles, or interesting research materials may be kept in Dropbox. Project-specific artifacts such as status meeting slides or process documents may remain in a temp directory for up to two weeks, at which point they must be committed to a repository or purged. The Development Lead is responsible for ensuring compliance with this policy.
* **Google Docs and Wiki Pages**: Internal artifacts, which require real-time collaboration and do not need explicit change control, are good candidates for Google Docs or the project Wiki. Examples include meeting notes, academic proposals, brainstorming, and polling. However, the internal project Wiki is the authoritative index for these artifacts. No project-related artifact shall reside on Google Docs without a corresponding link from the project Wiki. Refer to Section 4.3 for organizational guidance.

## Repository Organization

The README.org file is the organizational keystone to all GitHub repositories. This file shall reside at the root of each directory containing child nodes. Refer to Figure 1 for an example. This organizational file describes the contents and structure of each directory and provides online guidance for developers trying to understand the organization of the repository. Figure 2 provides a template for all README.org files. The GitHub web interface recognizes the \*.org extension and will format this index file as depicted in Figure 3.

|  |
| --- |
| Example Directory Structure |
| +-/  | --README.org  | +-subdirectory  | | --README.org  | | +-child1  | | +-child2 |

Figure 1 - Directory Structure Example

|  |
| --- |
| Example README.org File |
| #+TITLE: Configuration Control  \* Directory Structure  - \*directory\_name\*: Description of contents ...  - \*directory\_name\*: Description of contents ...  \* Workflow  The workflow section is optional; if a workflow is associated with this directory, reference the corresponding section from the SCMP. For instance, the folder containing the ADS should reference Section 3.2.3 of this document.  \* Naming Conventions  The naming conventions section is optional; if developers should follow a specific naming convention for contents in this folder (not already specified by a workflow), explain it here. For instance, “all folders shall be named YYYYMMDD\_Review corresponding to the date of a particular design review.” |

Figure 2 - README.org Template

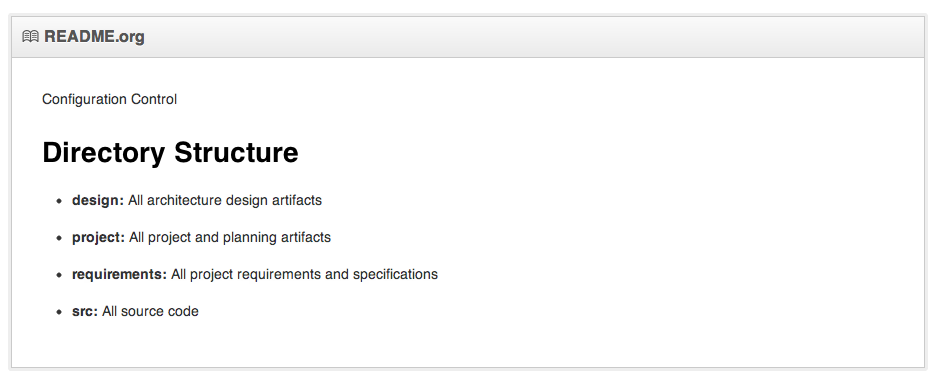


Figure 3 - Example README.org formatting

## Wiki and GDocs Organization

The Home.org page of the internal Wiki is the central organizational mechanism for all other Wiki and GDocs artifacts. From this page, a developer should be able to navigate to any active and collaboratively edited page. Non-terminal pages (that is, pages that simply act as gateways or menus to other pages like Home.org) must provide a description of its contents next to the provided link. For example: - Status Meeting Minutes: Weekly meeting minutes and links to slides. If a page naming convention is used, specify it at the top of the page. For instance, “Name all meeting minutes pages using the long date format.”

### Client Meeting Minutes and Material

Since client meetings represent collaboration between teams, meeting notes should be maintained in the GitHub Argo repository along with any slides or supporting documentation. These notes may be maintained in Wiki or document format. However, if maintained in Wiki format, links to corresponding support material should be included.

### Status Meeting Minutes and Material

Since status meetings are primarily internal events, meeting notes should be maintained in the GitHub internal repository along with any slides or supporting documentation. These notes should be maintained in Wiki format and links to corresponding slides or support material should be included.