Software Configuration Management Plan and Procedures Template

Version 2.0

Authors

[Name]	[Name]	[Name]

Version history

		Draft		**Final**	
Description	*Version*	*Draft Date*	*Author*	*Approval Date*	*Approver*

Table of contents

1. Introduction

This document provides the comprehensive plan and procedures for the <*Project Name*> Software Configuration Management (SCM) as well as the project library. The SCM system will be used to manage (i.e. store, version, provide change history) for the software product deliverables and project documentation (i.e. plans, status reports). This document will be revised and republished to remain consistent with other project documentation such as:

- Change Control Plan;
- Testing Plan(s);
- Defect Management Plan;
- Quality Management Plan and Policies;
- Project Subcontractor Management Plan;
- Quality Management Audit and Report requirements.

1.1. Purpose

<Modify this section to indicate purpose for the project – in general this document is very comprehensive therefore modest-size projects may be able to reduce this plan to the scale of their project >.

This plan provides the basis for performing software configuration management (SCM) and project library activities involved in the <*insert engagement name*> engagement. The purpose of the SCM plan is to define the processes, procedures, and conventions used in the organisation, management and protection of software assets and project documentation. Detailed procedures are provided to help ensure the integrity of the versioned assets is maintained throughout the project life cycle.

This document does not address installation/configuration of the project's development, test or other computing environments.

1.2. Audience

The engagement team must understand and comply with the procedures set forth in this document. The primary audience includes the individual developers, testers, management, organisations, and project team members who need access to or work within the SCM system.

All project participants are responsible for reading and understanding the content of this document. SCM personnel and project management will use the procedures in this document to manage the implementation of the engagement configuration as well as the project library.

1.3. Scope

This SCM plan is used as the basis for performing the configuration management activities, including document and repository management, software configuration management, and release management for the <*insert engagement name*> engagement. It identifies the configuration management roles, and responsibilities that include interfaces to other organisations or systems.

<This plan may be written for an initial release, a single release, and/or concurrent multiple releases including maintenance driven by a production release – describe here what this plan applies to.>

The SCM procedures, conventions, and processes supporting the various phases of the product's life cycle are described. They manage versions and releases of:

- Project library items – which includes all artefacts (e.g. files such as project plans, etc.) produced during the lifetime of the project that are not part of the software product deliverable;

- Software product items – which includes all files that are part of the software product deliverable (i.e. source code, scripts, help and other release/product documentation, etc.). The scope of the SCM is summarised in Figure 1 below.

<The figure and subsequent paragraphs may require modification based on how quickly the change control/configuration management discipline is introduced in the project.>

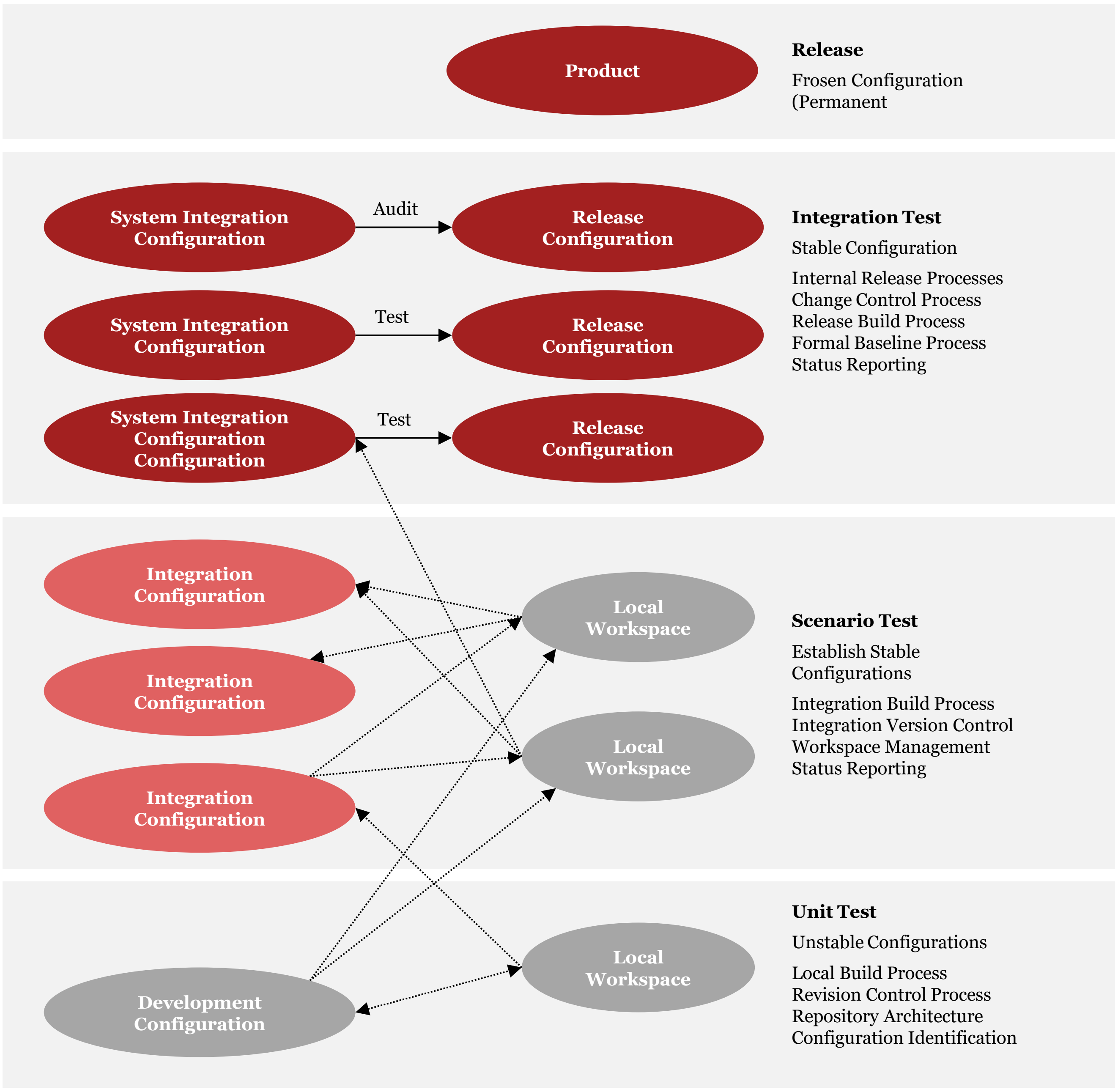

Figure 1: Software Configuration Management Overview

Unit test and scenario test are the first two phases where developers assemble the components into a co-operating system of processes. The infrastructure required to support these stages are predominantly within the revision control, workspace management and software build capabilities.

Once stable, and all features are verified as correctly functioning, the integration test begins and the integration configuration comes under additional change control. A change management database is used to help prioritise and track changes to the configuration. In general, as the software proceeds through the test stage's more formal procedures, conventions and processes are followed. Each software release is maintained within the SCM repository.

1.4. Constraints and assumptions

<Identify constraints limiting SCM activities and any assumptions. Include any risks, or known potential issues, and how they have been addressed. Sources of information include project planning sessions and the Statement of Work.

Constraints may include using organisation tools, procedures, having to work within an embedded environment, transition from an existing environment, limitations on software licenses, training availability/schedule, budget and/or other resource constraints.>

This…

1.5. Approach

Configuration control of project artefacts (i.e. plans, baselines, documents, hardware, software, and deliverables) throughout the project life cycle includes the following major processes:

- Configuration Administration Criteria:

<The project team and major stakeholders should agree upfront on the criteria (e.g., standards/procedures) used to determine which project artefacts are subject to configuration management, the configuration identification conventions associated with these artefacts, and the library control mechanisms for maintaining the current released versions of these artefacts.>

- Change Management:

<Refer to the project's change control plan for details.>

- Reporting:

<Reporting activities are performed continually throughout a project to provide management and team the current status of change requests during the project life cycle. Refer to the project's Change Control Plan.>

- Data and Backup Restoration: restoration activities are performed as needed to provide the project team and/or the organisation with the last known validated project artefact that was entered into the project library

2. Organisation

2.1. Structure

< If the project is of sufficient scale to require an SCM group describe the roles/people of the organisation and the relationship to other organisations. Show the SCM relationship to project management, technical team, and any other users of the SCM services, which may include organisation personnel and/or sub-contractors. Define roles for any persons dedicated to project library functions and/or quality record retention within the SCM or other organisation.

Define the interfaces among groups, designating points of contact and responsibilities including auditing and status reporting.

If SCM is performed within another group (not a group dedicated to configuration management) explain by whom and how it functions and serves all members of the team – with smaller teams it may be a developer, or it may be part of the project management or quality team with larger teams. The organisation could also potentially provide the function.

Refer to the Project Organisation Chart.>

2.2. Authorisation

<Provide a general high-level description of the authorisation for configuration management activities. If this is already documented in the change control plan, project plan or elsewhere simply provide references.

Refer to the procedures section of this document for details – as the procedures include authorisation steps.>

2.3. Roles and responsibilities

<Refer to the Project Organisation Chart if roles are detailed in that document and document only complimentary information here.

Define the tasks and responsibilities of the SCM staff and of any associated groups including:

- *Ensuring that the configuration management process is followed consistently throughout the project life cycle;*
- *Participating in project planning activities, and in the establishment of standards and procedure;*
- *Interfacing to other organisations.*

Provide information on the individuals involved, the roles and responsibilities of each individual, and from where those individuals are assigned (i.e. division, service, or contractor) by completing the Roles and Responsibilities Table below. Include as much information as practical and add additional text in the surrounding paragraphs to describe any complexities.>

Roles and Responsibilities Table

Role	Responsibilities	Name	Organization
Project Manager	Ensures compliance with SCM processes, procedures and conventions Change Control Board (CCB) Member		Project Management
Development Lead	Administrator of development tools Ensures that development artefacts (source code and design documents etc.) are under revision control Ensures the integrity of the software revision control Contact for development for SCM problem resolution CCB Member		Development Team
Testing Lead	Administrator of Test Tools Ensures that test artefacts are under revision control Coordinates regression tests of software release builds delivered from SCM Engineer Verifies change requests implemented in release builds CCB Member		Testing Team
SCM Engineer	Administrator of the SCM Tools Administrator of software build management Administrator of version control/baseline management Ensures the integrity of release configurations Contact for management authorising builds to environments *<May serve as project librarian or support the project's librarian>*		Project Management Team < *often a member of the development team* >
Release Coordinator	Administrator of change control processes and Change Control Board (CCB) meetings Ensures the integrity of the change management database CCB Member		Project Management
Documentation Lead	Administrator of document change process Ensures that documentation is under revision control CCB Member		Software Product Documentation Team
Configuration QA	Administrator of baseline audit process Reports audit results to Project Management Ensures the integrity of the release baselines		QA team < *may be member of project management, or testing team* >

3. Resources, training and schedule

3.1. Staffing

<Determine staff estimates, staff assignments, and required skills for the configuration management functions to cover the life of the engagement. If already documented elsewhere (i.e. project management plan) simply state so and refer to the documents.>

3.2. Additional required resources

<Refer to the SCM Environment section later in this document. Point out any high priority or high-risk resource needs. Also identify anything not described elsewhere in this document (i.e. space, product documentation).>

3.3. Training

<Define the training requirements necessary to support configuration management activities for the engagement, and the availability of training. Describe any mandatory training required for different roles (developer vs. others). If the SCM environment is also supporting the project library describe training/documentation other project members will need.

Training will likely include: courses available from tool vendors, peer training (one developer instructing another), and project training sessions (the projects SCM engineer instructing developers on tool use and project conventions).>

3.4. Schedule

<The schedule of the configuration management review, audits and any on-going activities should be related to the major activities and associated milestones. Refer to the project schedule if it is documented there and include just major or extraordinary milestone activities here.>

SCM Schedule Table

Activities	Target Date	Assigned Staff
<Develop SCM Plan>		

4. Standards, procedures, and policies

4.1. Standards

<List any standards or external requirements that the SCM must meet. This may include supporting a CMMI level and/or audits conducted to ensure compliance with a standard. It may also include standards adopted by the project. Audits often require proof of existence – describe any record keeping that will be done to provide evidence of compliance.>

4.2. Procedures overview

<List here the major SCM activities that must be supported by this plan. Each listed procedure must be documented in this or a companion document. Identify where the procedure is/will be documented. Include any organisation specific procedures that may already be in place and must be followed.>

The SCM procedures and where they are documented is described in the following table.

SCM Procedures

Procedure	Short Description	Reference
Change control		
Revision Control		
Data Recovery and Backup		

The SCM procedures will also support the project library and describe how the SCM system is used to support baseline and change control of project documents (plans).

4.3. SCM policies

<Describe the general project policy regarding the access, use and ownership. Include:

- *Compliance and consistency with the project Change Control Plan (for example when will a new document first be put in the SCM system (should be before distributed for review);*
- *What the library is used for, all deliverables etc. and storage priorities;*
- *Checking artefacts out of the library and return to the library;*
- *Who manages policies and access rights for the library;*
- *Address deliverables from subcontractors under Configuration Management's control;*
- *Address commercial products to include licensing rights;*
- *Address ownership of the library contents consistent with the T&C's in the SOW;*
- *If the project entails changes to an operational system, define how maintenance changes to that system are uploaded into the project repository, and then merged into the various baselines;*
- *Include the data backup and restoration policies;*
- *Any policies required by the organisation;*
- *Address any privacy or confidentiality issues >.*

5. SCM environment

The scope of the SCM environment is defined to the content and implementation required to support the SCM standards, procedures and policies.

5.1. Contents

<Describe what types of files will be kept in the SCM repository, as well as any files that will not be. Some tools cannot accept certain types of files or may require additional encoding to accept them. Point out any limitations or constraints on contents. Point out any limitation on number of revisions, revision numbering etc. Comment on any automated assistance for branch and merge functions provided by the tool, and how and when they will be used.>

One of the purposes of revision control is to allow access to all of the elements in the repository. Another purpose is to ensure that all changes to elements do not get lost or overwritten. Users of the revision control system should be able to see the revision history of all elements, compare the differences between revisions, and merge revisions together to form new revisions. They should also be able to retrieve any version.

The purpose of this section is to identify components to be put under revision and ultimately baseline control. This includes:

- Source Code;
- Documentation/specifications;
- Design artefacts;
- Test artefacts;
- Project library items (plans etc.);

Configuration Item Table

Configuration Item	**Element**	**Extension**
Source Code	Java Source File	*.java
	Sequel Script	*.sql
	Java Server Pages	*.jsp
	Java Script	*.js
	HTML File	*.html
	Image File	*.jpg
	Image File	*.gif
	Shell Script	*.sh
	Oracle Table Constraints File	*.con
	Oracle Index File	*.ind
	Oracle Stored Procedure Body File	*.spb
	Oracle Stored Procedure Definition File	*.spd
	Oracle Table Definition File	*.tab
	EJB Deployment Descriptor	*.xml
	Java Properties File	*.properties
	Build Script	*.xml
Design Artefacts	Rose Model File	*.mdl

Configuration Item	Element	Extension
Architecture	Design document	*.doc
Test Artefacts	Test Suite	*.xls
Test tracking	Metrics	*.xls
Core Internal Documentation	Project Schedule (MS Project)	*.mpp
	Test Plan	*.doc
	Software Configuration Management Plan	*.doc
	Business Requirements Specifications	*.doc
	System Requirements Specifications	*.doc
	Detailed Design Document	*.doc
User Documentation	User Documentation	*.doc
Engagement Team Documentation	Project Handbook	*.doc

5.2. Attributes

Each item in the repository will be maintained with the following attributes:

- Name;
- Identifier;
- Identity of the person who initially created the item;
- Date/time the item was initially created;
- Revision number – for each revision the following attributes will be available
  - Identity of the person who applied the version;
  - Date/time version the revision was done;
  - Revision description (as entered at the time of the revision – may include CR number).
- A history of the revisions will be available. The history will permit retrieval of any version as well as all the version attributes.

5.3. Identification and traceability

<Describe how the project will conduct positive identification and traceability. For example will each change be delivered against a CR number, how and where will that CR number be entered and tracked. How/what will link the CRs to revisions in the repository?

How will a file be confirmed to be the correct version when outside the repository (Typically this requires version ids in the actual files?

How will code revisions/other artefacts be traced to requirements? >

5.4. Release identifier/software version structure

<Contains the conventions the project will use to identify releases including the format of the id for major, minor and fix releases (patches).>

5.4.1. Release definition

The following convention will be followed for software release numbers:

- **Major Release** – a release that contains significant new functionality or technology based enhancement. Most major releases are anticipated to require a complete installation procedure and may require data migration/conversion. Significant new tests must be developed and a full regression test will be required. Revised product documentation and new training classes will be required, and an increase in the support activities should be anticipated. Major Releases will also be known as single digit releases (e.g. 1.0, 2.0);
- **Minor Release** – a release that contains a new feature or technology enhancement. Minor releases will typically be applied as an upgrade rather than a complete reinstallation. Some new tests must be developed and minimally a partial regression test will be required. Revised product documentation and modification to training classes will be required, and a modest increase in the support activities should be anticipated. Minor releases will be designated two digit releases (1.1, 1.2). The minor release number is reset to sero for a new major release. There will usually be more than one minor release associated with each major release (e.g. 2.0, 2.1, 2.2, 3.0, 3.1). Minor release numbers should not be skipped. For example, the minor release after 2.1 is always 2.2, not 2.5;
- **Maintenance Release** – a release in which the broadest change is a number of bug fixes and very minor enhancements to existing features that have been packaged together in a release. Minimal new tests must be developed and a modest regression test will be required. Minimal revisions to product documentation and modification to training classes will be required, and no increase in the support activities should be anticipated. Maintenance Releases will be applied as minor upgrades and identified as three digit releases (1.1.1, 1.2.2);
- **Patch Release** – a software patch to be applied to remedy a very specific change request of limited scope that required an expedited delivery (cannot wait until the next one, two or three digit release). Patches require unscheduled development, test and operations resources and should be limited to avoid impact to resources/schedule. Minimal regression testing is required and there should be no impact to documentation, training, and support activity. All patches will be delivered with four digits (e.g. 1.2.0.1).

Each type of release may contain changes of lesser scope (i.e. a major release in addition to significant new functionality may address many change requests).

5.4.2. Format

A release identifier is of the form **MM.mm.ii.pp.bb[aaa]**

MM is the major release number

mm is the minor release number

ii is the maintenance release number

pp is the patch release number

bb is the build number

aaa is an optional alpha field

Fields are changed by incrementing, i.e., there should be no gaps in the numbering. MM, mm, pp, ii and bb are one or two digit numeric with optional leading seros. aaa is a 1-3 character alpha only field. Each software version should have a unique release identifier. Typically, for a given release there will be several software deliveries (SCM builds) delivered to test. The fifth field tracks the software build. The build number is reset to one whenever the version number (MM, mm, ii or pp) changes.

The optional alpha identifier is used for software builds with the same functionality that differ in some other manner such as a build targeting one of several operating environments (Sun OS, Linux, etc.).

5.4.3. Release media and format

<Contains identification of the media and format of the media of the release delivery.>

Major and minor releases will typically be delivered via CD-ROM. Maintenance and patch releases will be made available via a secure server access. *<Provide Instructions here for secure server access here. >*

The format of the release will be a tar file identified by a release identifier.

5.5. Software and hardware environment

5.5.1. Tools

The SCM system is typically comprised of three separate but preferably integrated systems. The configuration management (sometimes called source control or version control) system provides revision/version control and workspace management. The change management system provides change request tracking. The build system provides automated product manufacture (provides dependency based compilation, setting of file attributes, preparation for release packaging). This many also be linked to an automated deployment system, particularly where the new software needs to be deployed to numerous servers or desktops.

SCM Tools

Servers	**Platform**	**Function/Applications**
Tools	Revision/version/workspace management	*nn* – Floating
Tools	Change request management	*nn* – Floating
Tools	Build Management	No License Required

5.5.2. Repository architecture

<This is an overview of the software/hardware configuration in use for the repository. Depending on the complexity of the software product and project library and how the SCM system is organised there may be a need to describe the "folder" structure (i.e. where is each type of artefact kept in the repository). >

5.5.3. Source (revision/version/workspace) control

ClearCase version *X.X* residing on the *AAA* servers is established to manage file access control, baselines, and workspace management.

ClearCase manages a repository called a Versioned Object Base (VOB) that holds all controlled component deltas, revision change history and baselines. ClearCase offers local workspaces called views that allow developers to work independently, but still access the various integration configurations. There are two different types of views supported. Dynamic views are always fixed on the current version, or the last revisions checked in. SnapShot views allow developers to isolate their work from the activities in the integration configuration until they choose to update.

5.5.4. Build system

<A brief description of the tools used should be in this section. >

This project will use Jakarta Ant supplemented with shell scripts to perform automated product builds.

5.5.5. Change control

<A brief description of the tools used should be in this section.>

ClearQuest version *N.N* will be used to manage all change requests, defect reports and resolutions. ClearQuest will be configured to support the procedures and state diagram described in section 6.

5.5.6. Infrastructure

<The infrastructure required to support software configuration management typically starts with a network/server/workstation topology that allows access to all users of the system. In addition tools will need to be installed to support the SCM activities and processes. The text below is an example of what typically should be described. >

Software developers will be working on PC workstations running Windows 2000. Software will be developed, built and unit tested on Windows 2000 workstations. Requirement writers, designers, and technical writers will be using NT based tools on PC Workstations. A single NT *X.X* server will be used as the rational tool license and ClearCase registry server. In addition, a shared directory will be setup on a Windows 2000 server to hold the ClearQuest project and database files.

A single Unix Solaris *N.N* server will support a single ClearCase repository

Server configuration

Number	Servers	Platform	Function/Applications
1	Sun xxx-1 *(ID here)*	Solaris N.N	ClearCase VOB Server ClearCase View Server
1	PC Server *(ID here)*	Windows NT 4.0 Server	Rational Tool License Server ClearCase Registry Server
1	PC Server *(ID here)*	Windows 2000 Server	ClearQuest Database

Servers will be accessed via a organisation workstation.

Workstation Environment

Number	Workstations	Platform	Function/Applications
nn	PC Workstations	Windows 2000 Workstation	Development machine ClearCase Organisation (View Server) ClearQuest Organisation Word Processor Xterm Emulator
N	PC Workstation	Windows 2000 Workstation	Windows Build Machine

5.5.7. Environment access

<Describe here who will have access to the environment and any procedures necessary to get access (use of VPN, logins, passwords). Describe all limitations/security restrictions on access. If dial-in or remote access is available describe the necessary procedure, if it is not available or access is restricted to specific networks or sub-networks describe these restrictions.>

6. Procedures

<Arrange the sections below to the manner in which the projects procedures will be modified through the life cycle. In each section highlight the differences from the prior section.

Refer to the project's change control plan for the engagement change process and then define any configuration management specific details using those sub-sections below that are relevant to the project and delete any unneeded sub-sections. If an automated tool is used for configuration management, identify the aspects of the tool that support the functions.>

6.1. Change control procedures

The change control process is defined in the change control plan. The following defines those processes that apply to configuration management.

When a release enters the multi-unit test phase, all changes will be under change control and require a change request (CR) to be opened and its progress tracked until verified by a successful test and closed. A Change Control Board (CCB) will approve all changes. The CCB will meet periodically to review the severity and determine the priority of all non-closed CRs. Changes to be included in the next release build will be determined and authorised by the CCB. Developers will hold source changes in their local workspaces and will not check them in until authorised by the CCB. The release configuration used for the build will be baselined with a unique release/build identifier and will be associated with the authorised CRs.

Documents and artefacts managed by the SCM system as part of the project library will be similarly managed. The baselined version of the document/artefact will in the SCM system and subsequent changes will be driven through approved CRs consistent with change control procedures.

6.1.1. Release coordination

The release coordinator is responsible for administrating the CCB meetings and managing the change control processes. In general, the release coordinator is responsible for ensuring the change request management system is always current and accurate.

Release Content Process

Step	Role	Action
1	Release Coordinator	Schedule CCB meeting and inform CCB members
2	Release Coordinator	Generate and distribute CR status report to CCB members prior to the CCB meeting
3	Change Control Board	Review, prioritise, assign, and authorise CRs
4	Release Coordinator	Emails a request to check In code to the lead developer listing authorised CRs
5	Release Coordinator	Email a request for baseline and build to the SCM Engineer listing authorised CRs and the release/build identifier for the build.
6	Release Coordinator	Updates CRs typically to the "Assigned", "Duplicate", or "Postponed" state.

6.1.2. Change Control Board (CCB)

As the release nears completion, the CCB will meet more frequently until it becomes daily for the final two weeks. In general, the CCB performs important decision-making activities.

Prioritise Change Requests – A report is generated before each CCB meeting listing all non-closed CRs reported against a specific release. This report will include CR severity and a description of the problem. The CCB will review to ensure that the proper severity has been assigned and the impacts are understood. Once impact is understood, in addition to CRs of lower severity, each severity 1 and 2 CR will be prioritised and scheduled based upon available resources and customer expectations.

Identify Duplicate Change Requests – The CCB will review new "Submitted" CRs to determine if they are describing a prior reported problem. Duplicates will be moved to the "Duplicate" state.

Assign High Priority Change Requests to Engineers – Once the priority is understood, the CCB will assign "Submitted" high priority CRs to engineers supporting the units that need to be modified.

Review Change Requests that Can't be Fixed – CRs that engineers cannot fix because they can't reproduce the problem (or are limited by design constraints etc.) will need to be reviewed by the CCB for a decision to determine next steps.

Authorise Change Requests to be Implemented into Future Baselines – "Resolved" CRs that have been fixed and verified by engineers in their local workspaces will be reviewed and authorised by the CCB to be implemented into the next or future builds.

Authorise Audit – To understand the quality of all the configuration items (source code, documentation etc.), the CCB may authorise a formal baseline and release of all products to Quality Assurance to validate.

Postpone Change Requests – Low impact or CRs associated with functional enhancements may be put on hold or postponed by the CCB to a future release/build.

6.1.3. Change request type

A change request can be opened by anyone and will be one of two types.

Defect (bug) – A defect is a source implementation that does not satisfy a release requirement defined in the product's specifications.

Enhancement – An enhancement is an improvement to existing functionality or new functionality. Prior to implementation an enhancement will require new or revised requirements, and corresponding changes to designs, test cases, and product documentation.

6.1.4. Change request severity and priority

Severity level is a means of assessing and documenting the impact of a CR to the recipient (customer or testing) of the release/build. The severity level gives restoration or repair priority to problems causing the greatest impact. For example, if a CR is blocking the execution of many test cases and therefore jeopardising the test schedule it will be assigned a high severity. The availability of a work around that is satisfactory to the recipient may result in a downgrading of the severity level.

A **Severity 1** defect means that a test or a production application is down has experienced data corruption or a major portion of functionality is inoperable blocking operations. Severity 1's typically require an immediate resolution via an emergency patch. Characteristics of severity 1's include:

- Critical Impact;
- High visibility;
- Affects production commitment;
- Major loss of functionality;

- Large user community impact;
- Unacceptable performance impact;
- Problem cannot be bypassed;
- No viable workaround available.

A **Severity 2** defect impacts a portion of significant functionality, with little/no data loss. Severity 2 defects need to be addressed in the very short term, most often measured in days depending upon the service agreement.

- Serious Impact;
- Moderate visibility;
- Moderate to large number of users or orders are affected;
- Seriously slow performance;
- Severity 1 may be downgraded to 2 if there is an acceptable work around.

A **Severity 3** defect is prioritised according to resource availability, business priority and user priority. Severity 3s are scheduled for future builds and/or maintenance releases.

- Moderate acceptable loss of functionality;
- Low visibility;
- Low business impact;
- Minimal poor performance;
- Severity 2 may be downgraded to 3 if there is an acceptable work around;
- Enhancements or new feature requests.

A **Severity 4** defect is noted for a problem with little/no functional loss. Defects classified as 4s may be typos in documentation, optional functionality which is currently not being used etc. *<Some classification schemes do not include a severity 4. >*

6.1.5. Change request state transition

CRs will be use the state transition model (see Figure 2 below) via ClearQuest.

Periodically the CCB will review submitted CRs and assign the high priority tasks to engineers to work **(assigned)**. If a new CR is determined to be a duplicate of an existing one, it will be assigned a duplicate state. If a new defect CR has low impact or does not satisfy requirements of the current release, it may be moved to a postponed state to remove it from reports and put it in a holding state or scheduled for a future release.

Once a developer begins to work on a defect, the developer will update the CR state to opened. If a fix has been implemented (built, tested and verified) in a local workspace, the engineer will update the CR state to resolved to indicate it is ready to be included in a future build.

The CCB authorises (schedules) CRs to be implemented into the next release/build and the lead engineer will ensure the associated source code is checked in appropriately. The release coordinator will move the authorised CRs to a ready for build state. The SCM Build engineer will create a baseline and verify the source code required to satisfy the authorised CRs have been included in the baseline. After the baseline has been built, the product packaged and handed off to test, the SCM engineer will move the authorised CRs to a ready for test state. The test engineer will move verified CRs to the closed state. Implemented CRs that cannot be verified will be rejected and moved back to the opened state until a resolution can be found as shown in Figure 2.

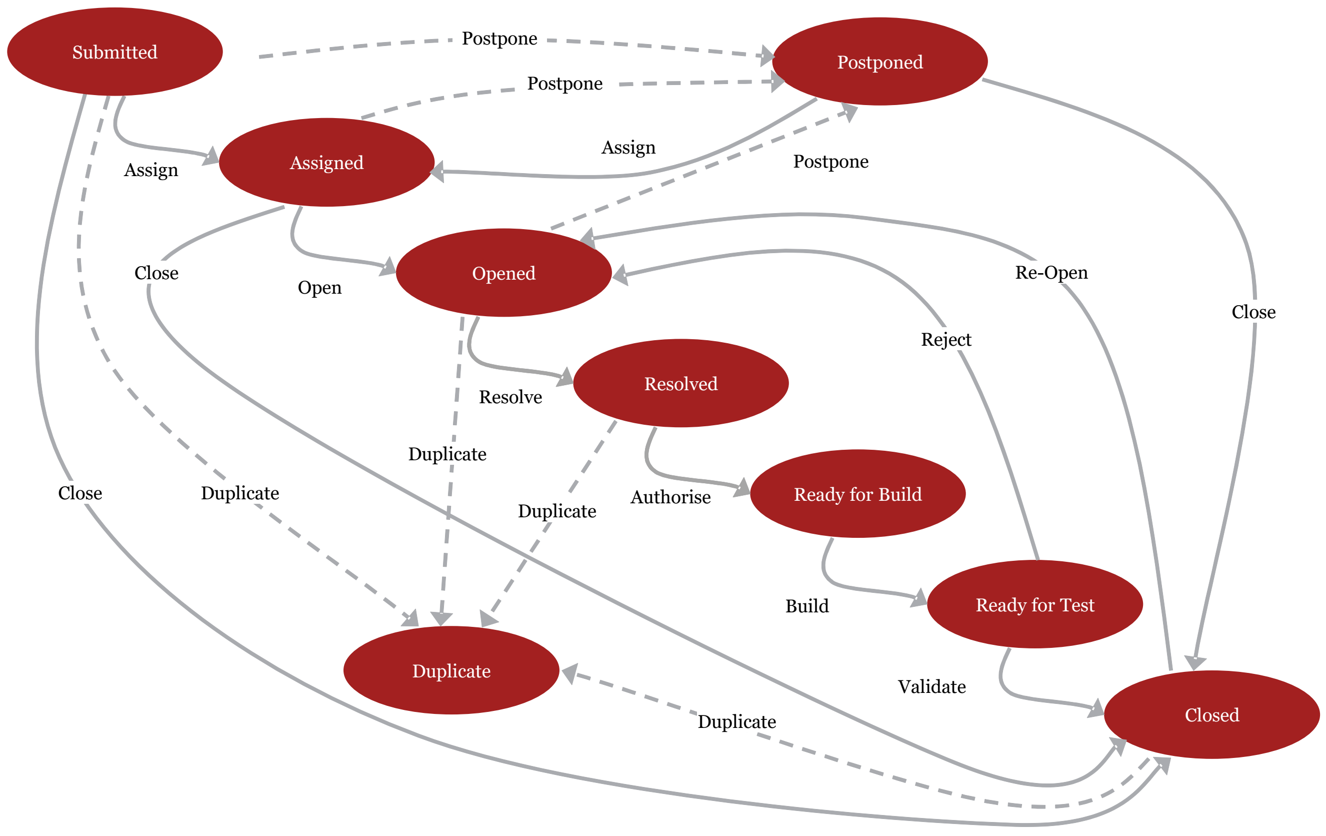

Figure 2: CR State Transition

State	Description
Submitted	A new CR is submitted.
Assigned	CCB assigns designated CRs to appropriate developer
Opened	Assigned Engineer has started working on the CR
Resolved	Developer has tested solution and is ready to check in changes
Build	CCB has authorised CRs to be implemented into next release build
Test	SCM has included CRs in release build and delivered to test
Duplicate	CCB determines this is a duplicate CR
Postponed	CCB decides to postpone work on CR
Closed	CR has been verified by test or has been postponed indefinitely

6.2. Revision control procedures

< This section describes the procedures by which check-in/check-out is conducted.>

Revision control ensures changes to items do not get lost or overwritten and provides access to all of the units in the repository. Users of the system are able to see the revision history of all units and compare the differences between revisions.

The following sections describe the revision control procedures to be used during the various test stages.

6.2.1. Unit test

During the unit test phase, developers will be using an open unreserved checkout/merge approach to source control. All developers will check out source files with the "–unreserved" option. Many developers will be able to check out a file at one time. Each workspace will be a dynamic view pointing to the tip revisions (/main/LATEST) so that each developer is constantly up-to-date with the current version. The first developer to check in an unreserved checkout will create the new revision. All other unreserved checkouts will need to merge the new tip revisions with their modified files in their workspace, build, and verify before they can check in their changes. Since the integration configuration is not under change control, developers are free to check in changes as soon as they think they are stable.

Unit Test

Step	Role	Action
1	Developer	Creates dynamic view from current version with login id in name (see Workspace Naming Conventions)
2	Developer	Checks out code with "-unreserved" option
3	Developer	Modify code, build, unit test, and verify
4	Developer	Checks for new revisions of checked out code, merges into the workspace, build and verify again
5	Developer	Checks in modified code before next build

6.2.2. Scenario, integration and acceptance test

During the scenario and integration test stage, developers will be using a controlled reserved check-in/check-out approach to source control. Only one developer will be able to check out a file at a time. Each workspace will be a snapshot view that points to the baseline the CR was reported against, isolating the specific CR task from the rest of development. Since the configuration is under change control developers will hold back their changes in their workspaces until the lead developer requests a check in (see Release Processes section).

Multi-Unit, System and Acceptance Test

Step	Role	Action
1	Developer	Creates snapshot view from the current version with login id and CR number in name (see Workspace Naming Conventions)
2	Developer	Checks out code with "-reserved" option
3	Developer	Modify code, build, unit test, verify
4	Developer	Wait for the lead developer to call for view name

6.3. Workspace procedures

<This section describes the conventions used for versioning of workspaces. >

During unit test, the simplest views will be used that only accesses the tip revisions (current version). During other test stages, view names will indicate what task (change request) users are working on. After the software is permanently base lined and deployed, views should indicate the baseline the user is working on as well as the task (CR number).

Convention: *{aaa* | *<userloginID>}_{int* | *r}<version>[.<build>]_cr<number>*

Aaa	A generic name for views used for Administrative or Build purposes
<userloginID>	The login id for the user creating the view
Int	If working on pre-release version use *int*
R	If working on released version use *r*
<version>	Version Number indicating baseline
<build>	Number attached to full Version Number indicating build number (12)
cr<number>	Change request number if working on specific task

6.3.1. Unit test

During Unit Test, developers should be using personal dynamic views named using the **<userloginID>_int<version>** format. For example, initially each should be working on the next release (1.1) and accessing the tip revisions (/main/LATEST) as modifications and new elements are checked in. To indicate the user is working on a pre-release configuration, a "**_int<version>**" naming convention (i.e. **jsmith_int1.1**) should be used.

Once the integration configuration has been formally baselined, verified by test and deployed, views should indicate work is being done on one of the next release configurations. Some developers may be working on the next functional release (i.e. **jsmith_int2.0**), some may be working on the first interim or maintenance release (i.e. **jsmith_int1.2**), and some may be working on the first patch for emergency purposes (i.e. **jsmith_int1.1.1**).

During this stage the SCM engineer will setup a dynamic view to build the current version of the configuration every night that will follow a generic ***aaa*_int<version>** naming convention (i.e. ***aaa*_int1.1**).

6.3.2. Multi-Unit, System and Acceptance Test

Developers will address defects as they are found by test and scheduled via the CCB. Since developers may be working on more than one defect (or enhancement) at a time, the snapshot view name should indicate the tasks they are working on. Since all changes should be under change control, the view name will indicate the CR numbers being worked. If a developer is working on a defect the Snapshot View name will follow the **<userloginID>_int<version>_cr<number>[_cr<number>]** convention. For example, if a developer is working CR 01267 a new view called **jsmith_int1.1_cr01267** should be created. If more than one CR is worked in a view, each CR number should be appended to the view name (i.e. jsmith_int1.1_cr01267_cr01287).

The SCM engineer will create a new snapshot view based on the new baseline that will follow a generic ***aaa*_int<version>. <build>** convention (i.e. ***aaa*_int1.1.0.15**) to correspond with the version number of the…

6.3.3. Maintenance release for production support

If a developer is assigned to fix a defect in a released configuration, the snapshot view name should indicate they are working on a released configuration by using the "**_r<version>**" naming convention. For example, working on CR 01389, would result in a view called **jsmith_r1.1_cr01389**. Again, multiple CRs can be worked in each view by appending an additional "**_cr<number>**" (i.e. jsmith_r1.1_cr01389_cr01297).

The SCM engineer will create a new snapshot view based on the new baseline that will follow the generic ***aaa*_r<version>.<build>** convention (i.e. ***aaa*_r1.1.0.21**) to correspond with the baseline version number. Maintenance build numbers continue to be incremented from the last system test baseline that produced the deployed baseline.

6.4. Version control procedures

During software development there may be at least three different "versions" of the software at any given time that will need to be managed. A "developer's workspace" represents a version of the software that should be based on the integration configuration, but with some local changes that should not impact other developers. An "integration configuration" is constantly changing as developer's check-in changes from their local workspaces and they become visible to the rest of development. A "release configuration" is a frozen snapshot of the integration configuration that can be delivered to test.

The purpose of version control is to separate developers and development teams so they do not impact each other, provide a way to integrate changes, as well as identify consistent software release/build baselines and correlate them to the specific life-cycle phases.

6.4.1. Unit test

During unit test engineers will be working on the tip revisions of the integration configuration. The purpose of version control during this phase is recoverability. The tip revisions will be labelled with a current version string just before the night integration build. The last current baseline will be relabelled with a last version string. Integration base lining allows generation of "diff" reports between that tell engineers which files and revisions are new since the last baseline so they can identify revisions that may have caused any unstable conditions.

Convention: {Current | Last}

Current	Tip revisions – baseline used to produce current build
Last	Baseline used to produce the last integration build

As the configuration begins to stabilise, semi-permanent baselines may be captured in case future configurations become less stable and it becomes preferable to recover a past configuration. Comparisons to past stable baselines may help teams identify revisions that were checked in that may have caused any unstable conditions.

Convention: int_<number>

int_	Indicates a pre-release baseline
<number>	First integration baseline is **int_00**, incremented for each subsequent baseline

Once a new repository is created and the primary directory structure and file elements have been either imported or checked in, an initial baseline will be labeled **(int_00)** that will be used as a base for comparison for future baselines. These integration labels can be removed once multi-unit test begins and a more formal baseline procedure is used.

6.4.2. Scenario, integration and acceptance test

Once the scenario test begins, a formal baseline procedure using version numbers will be used to improve reproducibility and traceability. After developers have checked in source code for the CCB authorised CRs, the tip revisions of the integration configuration will be labelled with the version and release build number.

The following guideline will be used for numbering application releases. The release number is comprised of five fields:

Convention: <major>.<minor>.<maintenance>.<patch>.<build>

This guideline is meant to achieve the following objectives:

- Provide a common release identifier across all applications to minimise miscommunication. For example, if even a minor change is made to the software being delivered to test, the associated released products should have a different release identifier;
- Provide a means to reference releases at different levels of detail, according to the audience. Using the same release identifier and a date is not acceptable. For example, an organisation may speak in general about release 3.2, whereas the development tar file used to install a specific version of that release might be identified as 3.2.1.9.0;
- Provide a structured identification scheme that includes information as to the scope of change between two baselines. For example, there is a greater functionality difference between 2.2.1.9 and 4.0.2.7 than between 3.2.1.9 and 3.2.1.10.

6.5. Special control procedures

<Describe any out of the ordinary measures that require special control or interfaces for the SCM.

Such procedures may come into play if more than one organisation or location are involved in a single development effort. For example, access rights may vary. Also a different group performing a stage of work in a different location may be using different SCM procedures/systems for a portion of the life cycle and deliver interim works into this environment.

If there are no special control situations state so, rather than delete this section. >

6.5.1. Subcontractor control procedures

< Describe the configuration management reviews, audits, processes and standards to which all subcontractors must adhere. State where they are the same as the normal project procedures and specifically describe all differences.

Also identify the frequency and use of configuration management audits of subcontractor performance and deliverables, including acceptance criteria for subcontractor-produced deliverables. Refer to the Quality Management sub-work stream and quality management plan for additional information.

If there are no subcontractors on the project and therefore this section does not apply, state so rather than deleting this section. >

6.6. Product builds/release procedures

<Describes the mechanisms for how builds and releases are maintained. This will typically require a description for each stage of the testing process. >

6.6.1. Unit test

During unit test, developers will create dynamic views that point to the tip revisions (current version). Local Builds will be performed in dynamic views according the instructions

<Depending how long a single build takes there may be variances in procedure – nightly build? >

6.6.2. Scenario, integration and acceptance test

Once the lead developer has informed the SCM engineer that all code has been checked in, the SCM Engineer will create a snapshot view that includes the build number in the name and labels the tip revision with the requested release and build number. Release builds will be performed in snapshot views according the instructions. If the build is successful, the final steps should be the automatic copy of the deliverable products to a release area and a compressed archive file generated.

<Replace the contents of the table below with a description of your build products. >

Build Products

Release Media	Release Directory	Archived Elements	Ext
***aaa**_<version>.sip*	*aaa\bin*	*Executable Files*	**.bat, *.ksh*

The build products release tar file will follow an **<appname >_<version number>.sip** naming convention (i.e. *aaa*_**0101001211.sip**).

Convention: <appname>_<version number>.sip

<appname>	Project acronym (i.e. aaa)
<version_number>	Ten Digit Version Number including build number (i.e. 0101001211)
<version_number>	Ten Digit Version Number including build number (i.e. 0101001211)

6.6.3. Release to field

Once testing has completed the final release configuration, the application and product documentation will be handed off to project management for final packaging and release to the organisation.

Final Release Package

Release Media	Deliverable Components	Ext
Master CD – Application	Archived build products	*.sip
	Manifest File (inventory)	*.doc
Soft Copy	Release Letter	*.doc
Documentation Soft Copy	Product documentation	*.doc

6.7. Status reporting procedure

<Describe the mechanisms and reports that are used to communicate status and provide a documentation trail for CRs and builds. It may be necessary to maintain a review/audit Log and collect planning and actuals for performing the configuration management activities. Reference the Project Status Report and the Configuration Audit Report template in the Project Management and Quality Management sub-work streams respectively.>

The project has identified three reports generated from the SCM systems to help communicate current status as well as provide a trail of documentation describing the differences between baselines.

6.7.1. Change request status report

The release coordinator will generate a "Change Request Status" report from the change management system just before each CCB meeting. This report should list all open CRs with their current state, severity, priority, assigned to fields and a brief comment describing the reason for the requested change. The CCB members will use this up-to-date report to make decisions about priority, who to assign new CRs, and schedule CRs into builds/releases.

6.7.2. Release configuration baseline release notes

"Release notes" will be generated for each baseline created during test stages (except unit test) by the SCM engineer. This report will list the authorised CRs (and associated comments) implemented in the baseline as well as a list of files (and new revision history comments) that have changed since the last baseline. The CR information is generated from a query of the change management system for all the authorised CR numbers. This report should include at minimum the CR number and the respective resolution description. The file/revision information is generated from a baseline compare ("diff") report from the revision/version control system. This report includes the file names, revision numbers, and the revision history comments of files that have changed since the last baseline. The developers will have listed the authorised CR number the check-in is trying to satisfy in the revision history comments.

This information should be combined into a single text report and distributed. This report provides baseline traceability and associates the change control processes with the revision/version control and build systems.

6.7.3. Configuration build log and status report

Each build will capture the display output into a log file.

Multi-unit test night builds will capture the display output of the build in a log file. This log file will be scanned for error messages and this output as well as the location of the log file are emailed to the development team with a "good" or "bad" build status. If a bad status, developers will look at the log file and determine which files and revisions are creating the unstable condition and correct the problem.

6.7.4. Practitioner reports

<Normally engagement team members will need either a report or query capability to support their work – describe that here. Examples follow:

- *All need a report of what is assigned to them with its disposition (type, severity, state, etc.);*
- *Managers need summary reports by person for their direct reports to manage commitments;*
- *Project managers need summary reports on outstanding CRs by severity to help manage organisation satisfaction.>.*

6.8. Data recovery and backup procedure

<Include the data backup and restoration procedures. Identify the manner and frequency of backups, which organisations are responsible, and what data is included. Are backups done nightly and between what hours? Are they incremental or full? When are the full backups done? What is the retention of the backup media? Specify the tool for backing up and restoring the data. Specify authorisations and the procedure to use for requested a restore. If there is a separate documented procedure perhaps from the admin org refer to it.

Also describe any procedures/policies the project may adopt for maintaining versions, builds, log files, status reports and other interim products.>

6.9. Release procedures

6.9.1. Internal release process to test

This process describes the steps required to authorise, baseline, build and release a source code configuration to test for verification during test (multi-unit, system test). The purpose of this process is to provide verification that the current stable source code configuration continues to satisfy all performance and functional requirements (regression testing) and that authorised CRs have been successfully implemented.

6.9.1.1. Build authorisation

The CCB reviews, prioritises, and authorises the CRs that will be implemented in the next baseline. The release coordinator informs the lead developer to ensure the source code associated with the authorised CRs is properly checked in. The release coordinator requests a formal baseline and build from the SCM engineer and emails the version symbol to use and a list of the authorised CRs. The release coordinator updates the authorised CRs to the "ready for build" state.

Step	Role	Action
1	Release Coordinator	Schedule CCB meeting and inform CCB members about location and time
2	Release Coordinator	Generate and distribute CR Status Report prior to the CCB Meeting to CCB members
3	Change Control Board	Review, prioritise, and authorise which CRs in the resolved state will be implemented into the next build
4	Release Coordinator	Email a request to check In code to the lead developer listing authorised CRs
5	Release Coordinator	Email a request for formal baseline and build to SCM engineer listing authorised CRs and the version identifier
6	Release Coordinator	Update authorised CRs to "ready for build" state.

6.9.1.2. Development check-in

The lead developer emails a list of the authorised CRs to the development team. Developers that worked on authorised CRs respond back to the lead developer with the name of the workspaces used to work on and fix the authorised CRs. The lead developer (or designated developer) opens each workspace and updates with the current version; builds and verifies.

The lead developer (or designated engineer) then checks in the associated source code and lists the CR number first in the comments (CR#####). The developer runs Check for locks utility in each workspace to ensure all code associated with authorised CRs are checked in. The lead developer informs the SCM engineer when all of the code is checked in.

Step	Role	Action
1	Lead Developer	Forwards check In request to development team requesting name of workspaces with fixes for authorised CRs
2	Developers	Reply with name of workspaces containing fixes for authorised CRs
3	Lead Developer	Assigns workspace verification to experienced developers
4	Developer	Updates workspace with current version; builds and verifies fix
5	Developer	Checks in modified source code listing CR number first in the comments
6	Developer	Runs Check For Locks utility to ensure all code is checked in
7	Lead Developer	Email SCM engineer when all code is checked in

6.9.1.3. Formal baseline of source code

The SCM engineer generates a text query report from the change management system using the authorised CRs. The SCM engineer locks the repository and labels the tip revisions of the integration configuration with the authorised version identifier. The SCM engineer generates a text baseline compare report from the version control system listing files and revisions (including revision comments) checked in since the last baseline. Any checked in revisions that do not list an authorised CR number in the comments will cause the SCM engineer to contact the lead developer to verify or fix. The SCM engineer combines the CR query with the baseline compare report to produce the baseline release notes.

Step	Role	Action
1	SCM Engineer	Receives request to build and waits for code check In confirmation
2	SCM Engineer	Generates a text query report from the change management system using the "ready for build" state
3	SCM Engineer	Verifies that CR report matches authorised CRs listed in request.
4	SCM Engineer	Receives code check In confirmation and locks repository
5	SCM Engineer	Labels the tip revisions with authorised baseline version identifier.
6	SCM Engineer	Generates a text baseline compare report listing the file and revision differences between the current baseline and the last one
7	SCM Engineer	Verifies that all new revisions list an authorised CR in it's comments
8	SCM Engineer	Alerts lead developer if revisions missing CR number or have non-authorised CRs in comments and waits for verification or fix
9	SCM Engineer	Unlocks repository and removes baseline if revisions need to be added or deleted and waits for code check in confirmation – return to step 3
10	SCM Engineer	Unlocks repository once baseline has been verified

6.9.1.4. Build from Baseline

The SCM engineer creates a clean workspace and updates with the source revisions labeled with the authorised version identifier. The SCM engineer builds the application from sources providing a non-debug version that deposits the release products in a release directory structure that mimics the installation structure. If the build terminates with errors, the SCM engineer contacts the lead developer and emails the error messages describing where the build broke. If revisions need to be removed or added to the baseline, the current baseline is removed and the baseline process is started over.

Step	Role	Action

1	SCM Engineer	Creates new workspace from new baseline with build number in name
2	SCM Engineer	Builds software for all platforms and monitors for errors
3	SCM Engineer	Alerts lead developer if build terminates with errors and waits for fix
4	SCM Engineer	Unlocks repository, removes baseline, baseline compare report, and build view if revisions need to be added or deleted and waits for code check In confirmation before starting baseline process over

6.9.1.5. Package build products and hand-off to test

If the build is successful, the SCM engineer updates the authorised CRs to the "ready for test" state. The SCM engineer finds the compressed archive file image of the product release area generated by the build and puts it on an FTP site that can be accessed by the test engineer.

The SCM engineer combines the text reports generated from the change management and version control systems to produce the baseline release notes. The SCM engineer emails a build complete message to project manager, lead developer, and test engineer with attached release notes and the location and name of the compressed file.

Step	Role	Action
1	SCM Engineer	Puts the build generated compressed archive file images of the release products on an FTP site the test engineer can access
2	SCM Engineer	Combine CR query with baseline compare report to produce the baseline release notes
3	SCM Engineer	Emails "build complete" message to project manager, lead developer, and test engineer with the location and name of the build package. Release notes should be attached

6.9.1.6. Verification

The test engineer uses FTP to access and copy the compressed archive files to the test machine. The files are uncompressed and extracted into the test area as root.

Step	Role	Action
1	Test Engineer	Generates a text query report from the change management system using the "ready for test" state
2	Test Engineer	Verifies that CR report matches authorised CRs listed in release notes
3	Test Engineer	FTPs the compressed archive file to the test machine, uncompress and extracts into the test release area as root
4	Test Engineer	Verifies authorised CRs were satisfied and moves them to the closed state
5	Test Engineer	Moves unverified CRs back to opened state
6	Test Engineer	Creates new CRs with submitted state for new defects discovered

6.9.2. External release process to organisation environment

The Internal Release Process to Product Test will be followed to produce the build products to be handed off. In addition to the build products, documentation will also need to baselined, verified and released to the deployment team. The hand-off process will need to be negotiated and documented according the needs of the deployment team. Thus this process will follow the same procedure as described in the prior section and include additional steps to accommodate the more formal and final packaging of the release to the organisation environment.

<Describe here the differences, additional steps for a organisation packaging, and install. These additional steps may depend upon the activities planned for the organisation environment that may include:

- *A "delta test" or testing that must be performed in a organisation environment because the test requires resources only available in the organisation environment;*
- *Acceptance test;*
- *Formal delivery of a new release>.*

<The following section contains possible additional paragraphs you may wish to include in your document. Use any or all of these paragraphs, or if none, then delete the entire section.

Any of the information below may be provided in a list in the body of this document, as a reference to an appendix, or as a reference to another document.>

7. Additional information

7.1. Acronyms, abbreviations and definitions

<Provide an alphabetical listing of acronyms, abbreviations, terms and definitions needed to understand this document.>

7.2. Open issues and future considerations

<If there are known issues, risks or considerations: describe, give timeframe, possible resolution>

7.3. References and related documents

<List the title, version/publishing date of referenced documents, websites, or other relevant references. If copyrighted documents are referred to, the copyright information must be appropriately referenced.>

8. Appendices

<Appendices may be used to provide information published separately for convenient document maintenance, such as classified data, or for providing supplemental material. The main body of the document should contain at least one reference to each Appendix. Appendices are listed in alphabetical progression (A,B,C).>