Software Configuration Management Plan

For Database Applications

Document ID: SCMP
Version: 2.0c

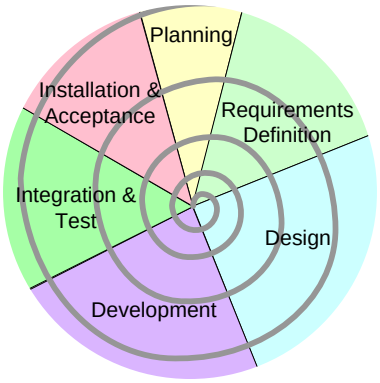


TABLE OF CONTENTS

INTRODUCTION

This document represents the Software Configuration Management Plan for database applications development. Configuration Management (CM) is essentially the process of identifying and assuring retention and control of all of the various artifacts (documents, source code, executables, e-mail, etc.) generated during the Software Development Life Cycle (SDLC).

PURPOSE & SCOPE

The primary objective of the CM process is to establish and manage baselines of product releases. CM for software engineering is defined as:

> A set of requirements, design, source code files and the associated executable code, build files, and user documentation (associated entities) that have been assigned a unique identifier can be considered to be a baseline. Release of a baseline constitutes retrieval of source code files (configuration items) from the configuration management system and generating the executable files. A baseline that is delivered to a customer is typically called a "release" whereas a baseline for an internal use is typically called a "build."[1]

This CM process is tailored to fit database-oriented software development efforts, and is related to the planning and life cycle description documents for the project. This plan provides the criteria and direction necessary for the performance of the CM process during the SDLC and pertains to the products either produced as a result of the development effort or procured as support or reference items.

[1] Chrissis, M. B., Konrad, M., & Shrum, S. (2003). CMMI: Guidelines for Process Integration and Product Improvement (p.164). Boston, MA: Pearson Education, Inc.

METHODOLOGY

The methodology presented here is based on the Software Engineering Institute's (SEI) Capability Maturity Model, Integrated (CMMI) and the Institute for Electrical and Electronics Engineers (IEEE) standards for Information Management. This SCM plan:

- Makes use of the principal project participants as defined in the Vision & Scope chapter of the project planning document.
- Describes the various categories of project artifacts, and how each category is to be controlled.
- Describes how the various project artifacts will be identified, and how that identification will change as each artifact evolves.
- Describes processes for approving and controlling revisions, assuring the availability of the most recent version of each artifact to all project participants, and making previous versions available upon request.

CONFIGURATION ITEMS

Project artifacts that are uniquely identified and placed under version control are known as Configuration Items. Configuration items fall into four general classes:

1. Evolving items, such as documents, which are subject to one or more revisions and new releases during the SDLC.
2. Source items, generally source code and object files used to build a production software application, which are generally numerous and frequently changing.
3. Support items, such as operating systems, of which the project requires certain versions for successful operation.
4. Archive items, such as SQA review forms which generally support decisions made during the SDLC are stored in electronic format for future reference.

PERSONNEL ROLES AND RESPONSIBILITIES

In a database development effort, two principal roles are defined for SCM activities:

1. Primary End-user Representative (PER)
2. Primary Developer Representative (PDR)

The PER acts as the primary point of contact for the end-user community. The PER is responsible for ensuring that significant project-related documents and communications generated by the end-user community are relayed to the PDR for storage and future reference.

The PDR acts as the primary point of contact for the development team. The PDR is responsible for the identification, control, and distribution of project configuration items. The PDR also maintains a central repository of issues, recording new issues raised by project participants and end users.

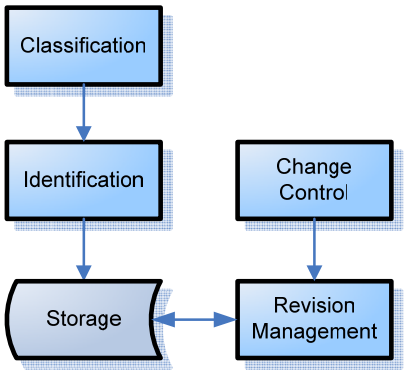
CONFIGURATION CONTROL BOARD (CCB)

The project Executive Sponsor, the PER, and the PDR (at a minimum) comprise the CCB (also known as a *Change* Control Board) for a specific project. The project SPMP describes any additional personnel assigned to the CCB. The CCB is responsible for prioritizing and selecting issues to be resolved during each iteration of the SDLC.

The purpose of the CCB in the context of software development and maintenance is to prioritize and select issues to be addressed during a specific SDLC iteration. The CCB is also responsible for determining the timing of support item hardware and software upgrades. Since these upgrades often impact production software viability, the CCB controls the revision cycle for these items.

CONFIGURATION MANAGEMENT PROCESSES

The processes used to manage configuration items fall into five main categories:



1. Classification: Each project artifact is classified as a member of one of the four configuration item classes.
2. Identification: Configuration items are assigned appropriate identifiers.
3. Storage: Configuration items are stored via mechanisms appropriate to their classes.
4. Change Control: Configuration items are modified in response to approved issues (enhancement requests or defect reports).
5. Revision Management: Each configuration item is revised in a manner appropriate to its class; archive items are never revised.

Separate chapters following this introduction address each of these processes.

METRICS

The PDR is responsible for tracking metrics associated with CM activities. The metrics to be captured are defined in the project SPMP.

REFERENCES

The software development lifecycle for this project defines a series of stages; each project stage is defined as a separate operation with specific inputs and outputs. This SCM plan describes how these inputs and outputs will be controlled and managed. The complete software development lifecycle for this project is described in a separate document, available at:

http://www.shellmethod.com/refs/SDLC.pdf

Please refer to the SDLC document for a description of the structure, inputs to and outputs from each of the stages of the SDLC, as well as a comparison to other SDLC models.

Other terms common to the software development process are defined in a Glossary of Software Engineering Terms, available at:

http://www.shellmethod.com/refs/seglossary.pdf

Please refer to this glossary for definitions of the terms used in this document and in the SDLC document.

STANDARDS

The following standards were used as guides to develop this CM process. The standards were reviewed and tailored to fit the specific requirements of database projects using the referenced SDLC.

- ANSI/IEEE 828-1990: Standard for Software Configuration Management Plans
- SEI/CMMI: Configuration Management Process Area

CLASSIFICATION

Project artifacts such as documents, applications, and support/reference files generated or purchased during the SDLC are all referred to as configuration items. As each item is generated or purchased, it must first be classified into one of four configuration item classes: Evolving, Source, Support or Archive.

EVOLVING ITEMS

Evolving items generally take the form of documents, help files, test data and application executables generated during the development process. These items are generally revised one or more times during the SDLC, and as such are considered to be evolving.

Evolving items are usually project stage deliverables, and are generated as a result of collaboration between participants. Although evolving items are in the substantial minority of all items generated or purchased during the SDLC, they require the most control and coordination, and are the subject of the majority of the processes described in this plan. In some cases, the CM tool used for management of evolving items is the same as that used for source items (described below). In either case, the CM tool for evolving items is identified in the Software Project Management Plan (SPMP) for the current project.

SOURCE ITEMS

Source items are generally source code and object files used to build a prototype or production software application. Although source items also have an evolutionary nature, they are normally used only by developers through various development tools. The storage of source items depends heavily on the selected development tool; some are stored as objects in a single repository, other tools may store independent files.

Due to the sheer quantity of these items, as well as the frequency of their revision and the possibility of multiple, simultaneous editors accessing them, source items are generally managed by specific configuration management tools. As such, it is best to utilize the identification schema and configuration management processes

supported by the selected development tools. This plan defers identification and low-level management of source items to a configuration management tool that is part of, or compatible with the chosen development platform for this project. The CM tool for source items is identified in the project SPMP.

SUPPORT ITEMS

Applications, operating systems, development tools and other items purchased to support the SDLC and/or production operations are termed support items. The key term here is *purchased*. Almost every other configuration item is created at some point during the SDLC. Support items are created by other vendors and purchased as necessary. Examples include development tools, database/application/web servers, documentation tools and operating systems. In some cases, software support items are compatible with a specific range of hardware. If this is the case, any hardware items that are bound in this manner to software support items are managed by the PDR as hardware support items.

The reason support items are tracked under this configuration management plan is that the development and/or production operations will most likely depend on their presence or availability. Furthermore, these operations may well be dependent on specific versions of the support items. For example, the application may require a compiler of a specific version or higher, and may be incompatible with lower versions. Under configuration management, the compiler would be identified as a support item with a minimum acceptable version level. Support items are normally managed as physical property items, and are inventoried and stored using tools and storage areas described in the project SPMP.

ARCHIVE ITEMS

Items that are created during the SDLC, but are not evolving or source items are termed archive items. These items are retained for future reference, as opposed to revision. Examples include SQA review forms, project status presentations, copies of electronic mail correspondence, and memoranda of stage exit approval and product acceptance. Archive items are stored in accordance with the mechanism described in the project SPMP.

HARDWARE

Although supporting hardware may be procured for development, test and/or production operations, this plan defers responsibility for most hardware items to the respective property managers of client and developer.

Since minimum acceptable hardware configurations may be defined by the support items described above, some hardware items are managed by the PDR as

support items when this relationship is identified. In this case, the PDR and the property manager coordinate with each other and with the CCB before authorizing the upgrade or replacement of hardware upon which software support items are dependent.

IDENTIFICATION

The methods for uniquely identifying instances of each of the configuration item classes described above are specific to each class. Evolving items are assigned unique identifiers, while source item identification is managed by the selected configuration management tool. Support items carry their own identification and version numbers assigned by their developers, while archive items are identified primarily by name and date.

EVOLVING ITEM IDENTIFICATION

Evolving items fall into two classes: Documents, and software executables / support files.

DOCUMENTS

Document evolving items are assigned unique identifiers that identify the project and (if applicable) component with which they are associated, along with the current revision level. The identifier consists of one to three parts separated by hyphens in the format ACRONYM, ACRONYM-CLASS or ACRONYM-COMPONENT-CLASS.

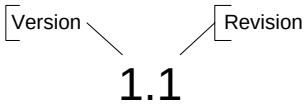
Evolving items that are “umbrella” items (not specific to a single project), such as policies, process descriptions, and guidance are identified with a single acronym, generally derived from the title and/or class of the item. An example is the identifier for this SCMP.

Evolving items that are project-specific, but not associated with a project component use a two-part identifier composed of the project acronym and an acronym derived from the title and/or class of the item. For example, the SPMP for a project named Basic Order Tracking System (BOTS) would carry an identifier of BOTS-SPMP.

Evolving items that are project-specific and associated with a project component use a three-part identifier composed of the project acronym, the component acronym, and an acronym derived from the title and/or class of the item. For

example the requirements document for the BOTS customer component would carry an identifier of BOTS-CUST-SRD.

The version level of each item is maintained as a separate identifier. This allows the primary identifier to be used as part of a file name or URL for access to the most current version without requiring changes to all referencing items. The version level is maintained as a numeric identifier with two components:



1. Version number, which appears to the left of the decimal, and
2. Revision number, which appears to the right of the decimal.

VERSION NUMBER

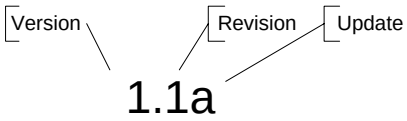
The version number changes only when the core architecture of the item changes, as when an item is completely overhauled, with substantial internal changes. In this case, version 1.1 would become version 2.0.

REVISION NUMBER

The revision number changes when existing content is changed, but the overall structure and flow of the item remains the same. The normal sequence of revision is 1.0, 1.1, 1.2, and so on.

SOFTWARE EXECUTABLES AND SUPPORT FILES

Software executables and support files are generally identified by name and version number, such as “Main DB v1.1a.” The naming convention for each software evolving item is defined by the development team. The version numbering scheme consists of three components:



3. Version number, which appears to the left of the decimal,
4. Revision number, which appears to the right of the decimal, and
5. Update level, consisting of a single trailing character.

VERSION NUMBER

The version number changes only when the core architecture of the software item changes, as when moving from one level of the development tool to another, when an application is completely overhauled, or the user interface changes fundamentally. In this case, version 1.1a would become version 2.0.

REVISION NUMBER

The revision number is updated when new features, functionality, or other content are added or significantly changed. In the normal case, the core architecture or user interface have been extended or limited in some manner. The most common reason for changing the revision number is when adding a new module or other functionality to the software item. The normal sequence of revision is 1.0, 1.1, 1.2 and so on.

UPDATE CHARACTER

The update character is appended or incremented when the only change to the software item is to correct one or more defects, without the addition of any new functionality. Version 1.1 of the software would become v1.1a, 1.1b, and so on. This updating is over-ridden when a combination revision, involving bug fixes and new feature additions, is performed. In such a case, the software revision number is incremented and any update indicator is dropped, as in v1.1b to v1.2.

SOURCE ITEM IDENTIFICATION

As described in the previous chapter, source item identification is managed by the selected configuration management tool for this project, which is identified in the project SPMP. Source item CM tools are vendor supplied software; refer to the vendor documentation for a description of source item identification and management.

SUPPORT ITEM IDENTIFICATION

Support items are identified by common product name and the version number range necessary to support successful development or production operations. For example, a text editor may be upgraded during the course of the project from version 2.1 to version 2.2a; the version range for this configuration item would show 2.1 - 2.2a. This is important for after-the-fact retrieval of archived project information; documentation and source items are best handled with known compatible versions of their parent applications.

ARCHIVE ITEM IDENTIFICATION

Archive items are generally miscellaneous support documents and records of communication that are stored for later reference. These items are stored as described in the project SPMP. Each item is identified by file name and modification date as returned by the operating system. In the event an identically named item is already present in a target subdirectory, the new document file name will be appended with a sequential number to prevent naming conflicts.

STORAGE

Each configuration item class has different storage requirements. The evolving and source items have the most complex storage requirements, while archive items are the simplest to manage.

EVOLVING ITEMS

After identification, evolving items are stored in the subdirectory specific to the stage in which they are produced. A copy of the item is then placed on the project Web Server, and a link established from the project homepage to the new item. If the new item is an update or revision of an existing item, the old item is replaced on the Web Server by the new item under the same hyperlink. This insures that the most recent version of each project deliverable is available to all participants.

The current and all prior versions of each evolving item are stored in the project CM tool as defined in the project SPMP. Standard check-in/check-out controls prevent more than one person from editing an evolving item.

SOURCE ITEMS

Source items are generally created by members of the development team and immediately introduced to the source item CM tool as defined in the project SPMP. The tool handles storage, identification, versioning and check-in and check-out operations. Any member of the development team, as well as the PDR, may introduce source items to the source item configuration management tool. This is the only case where the PDR does not have to be involved in the identification or storage of new configuration items.

SUPPORT ITEMS

The PDR logs each support item as it is received from a vendor into the support items log for the project. After the software has been appropriately installed, the original media used to perform the installation, along with any necessary license key information, is placed into a physically secure storage area for later retrieval

if necessary. In most cases, a software copy of the installation executable is stored on a central fileserver.

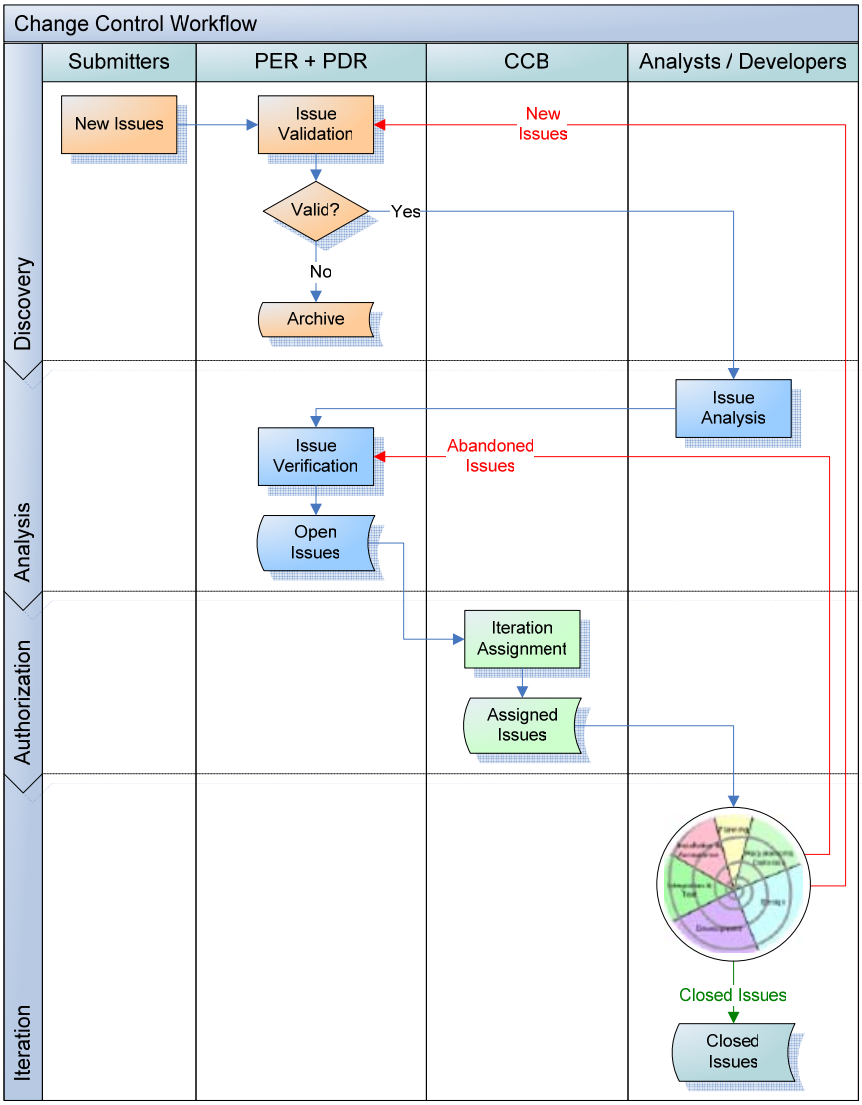
Support items may only be upgraded on production systems after the concurrence of the CCB. This rule is in place to make sure the CCB is involved in the decision to upgrade items that may impact the viability of current production software.

ARCHIVE ITEMS

Archive items are stored in the appropriate subdirectory of the archive items directory for the project. The archive items directory is available on a central fileserver. The archive items directory may be further divided into topical subdirectories at the discretion of the PDR.

CHANGE CONTROL PROCESS

The change control process manages the acquisition, verification, approval, and execution of *externally identified* issues (enhancement requests or defect reports) identified during application development and/or production operations. The change control workflow consists of four actors and four phases:



EXTERNALLY-IDENTIFIED ISSUES

The issue management process for this project is restricted to those issues identified by outside parties (reviewers, testers, end-users, etc.) as opposed to those identified by members of the development team. Externally identified issues are either defect reports or enhancement requests.

DEVELOPER-IDENTIFIED ISSUES

The development team will naturally generate a large number of issues, but these issues are typically progress-related issues focused on the current iteration. Examples of these issues include the need to gain access to a database schema, finish a specific screen, refine a reporting format, or load a specific set of test data. These issues are developer-derived and do not need validation, analysis, or verification. Instead, developer-identified issues are maintained separately and are part of the internal working artifacts shared within the development team.

ACTORS

The actors associated with change control include:

1. Submitters, who are generally end users, reviewers or testers,
2. The project PER and PDR acting in concert to perform verification and validation of new issues,
3. The project CCB, which controls the assignment of issues to SDLC iterations, and
4. The project development team, which performs issue analysis and executes the SDLC iteration.

PHASES

Each issue flows through four phases:

1. Discovery, where new issues identified during production or SDLC iterations enter the workflow and are validated,
2. Analysis, where the effort associated with the resolution of each issue is researched,
3. Authorization, where the CCB assigns verified issues to specific SDLC iterations for resolution, and
4. Iteration, where the development team develops and/or modifies the documentation and code as necessary to resolve each issue.

DISCOVERY PHASE

Issues may be discovered by end users during production operations or by members of the development team during software development. New issues are submitted via email, written reports, or telephone conversations, and stored in a repository as described in the project SPMP. In any case, each new issue is eventually received by the PER and PDR for validation. To validate an issue, the PER and/or PDR:

1. Determines the issue is not associated with an infrastructure element, such as network access,
2. Determines the issue is not a duplicate, and
3. Reproduces the issue in the case of a defect report.

New issues that fail to meet the above validation criteria are placed in an archive of invalid issues. New issues that pass the criteria are sent to the development team for analysis.

ANALYSIS PHASE

Valid new issues are examined by appropriate members of the development team to identify the root cause of the issue, as well as the configuration items that would have to be changed, and how they would be changed in order to close the issue. The development team members then estimate the level of effort required to make the defined changes. This information is added to the issue, which is then passed back to the PER and PDR for verification. During verification, the PER and PDR assign a priority level to the issue based on the severity of the defect or the desirability of the enhancement. The issue is then assigned a status of “Open” and stored for review by the CCB.

AUTHORIZATION PHASE

The CCB evaluates open issues to determine the SDLC iteration, or product release, that will address the development effort needed to close the issue. Issues can be assigned to the next iteration, a future iteration, or left in an unassigned state as determined by the needs of the end user community, the resources available from the development team, and the funding available from the project sponsor. Issues with a status of “Assigned” are also tagged with the iteration (also known as the target release version) of the software in which they are to be addressed.

ITERATION PHASE

Each iteration of the SDLC is focused on the accomplishment of a specific set of goals, for either the application or for a specific component. The CCB authorizes

the initiation of an SDLC iteration. At this point, the planning stage of the SDLC is kicked off and the SPMP or CIP (as applicable) is developed and approved.

NEW ISSUES

During design and development, the team may identify additional issues. As the scope of work is fixed for the current iteration, these additional issues are submitted as "New" issues to be addressed in later iterations. The project team *may* resolve a new issue in the current iteration, but this is an unusual action requiring concurrence of the CCB and will generally be recorded as a stage reversion.

ABANDONED ISSUES

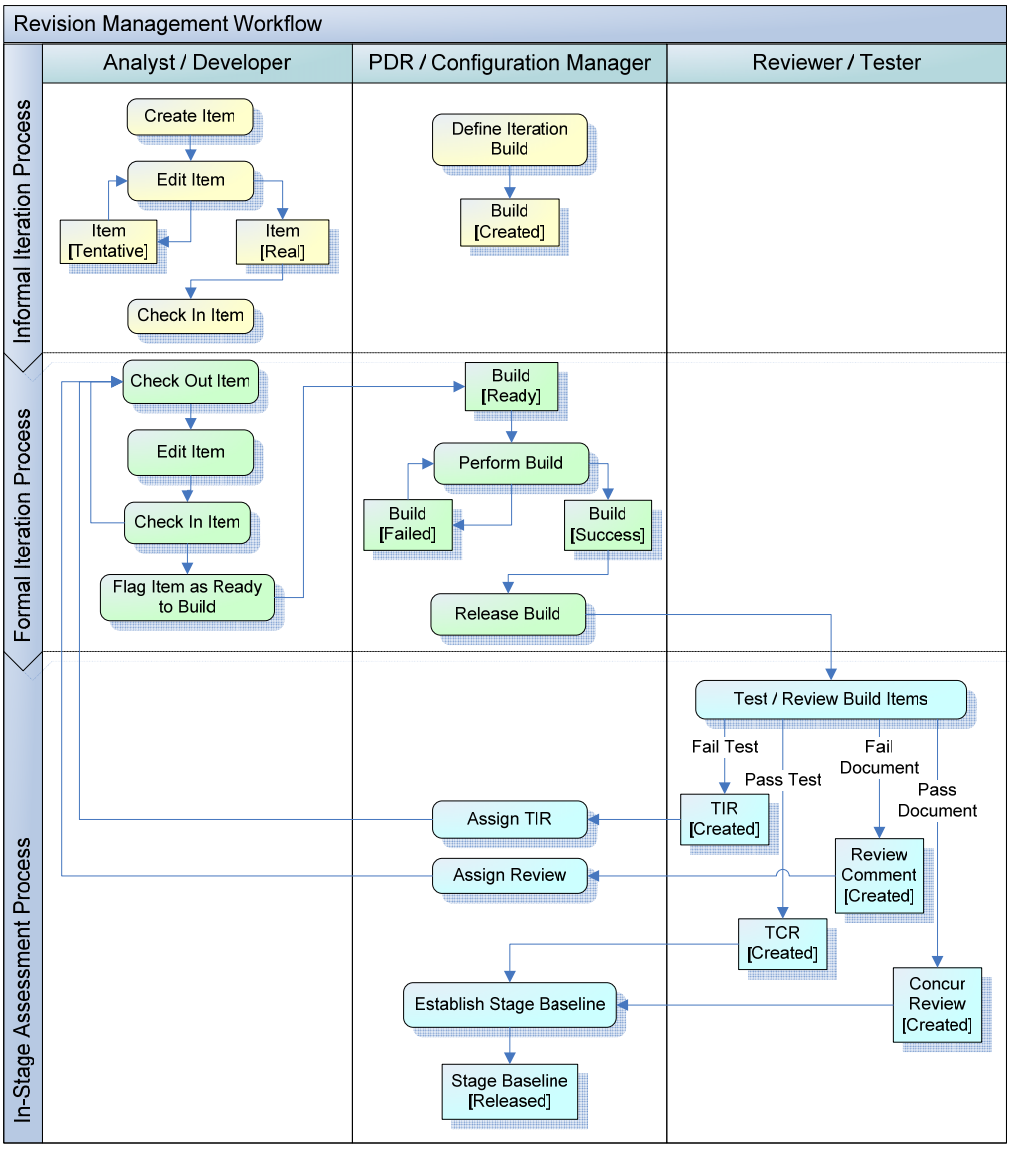
As the team moves through the iteration, certain issues may be acknowledged as unworkable given the current resource and/or schedule constraints. With the permission of the CCB, these issues are abandoned. The scope of work is removed from the iteration, and the issue is returned for verification by the PER and PDR. Note that issue abandonment may also result in a stage reversion. Refer to the SDLC for a discussion of stage reversion.

CLOSED ISSUES

The normal conclusion of this process is the completion of the iteration, resulting in the closure of all remaining issues assigned to the iteration.

REVISION MANAGEMENT PROCESS

The revision management process manages the creation and editing of individual configuration items as well as the combination of items into one or more builds. Builds that pass all required reviews and/or tests become stage baselines.



ACTORS

The actors associated with revision management include:

1. The analysts and/or developers associated with the development team,
2. The PDR, acting in the role of Configuration Manager, or a specifically assigned Configuration Manager, and
3. Reviewers and/or testers charged with evaluating stage work products.

IN-STAGE PROCESSES

Each stage of the SDLC is composed of a standard set of pre-defined stage processes. The stage processes that include revision management activities include:

1. Informal iteration, where new configuration items are created (often from templates or prototypes), and checked in to the appropriate CM system as defined in the project SPMP.
2. Formal iteration, where configuration items are finalized for review and/or test, and incorporated into a release build for the stage.
3. In-stage assessment, where review and/or testing is conducted and the build is either rejected for re-work or accepted as a stage baseline.

INFORMAL ITERATION PROCESS

The informal iteration process is focused on elicitation and refinement of requirements and project artifacts, resulting in the generation of configuration items that may or may not become candidates for the stage build. For this reason, new items are classified as either “Tentative” or “Real.” This status becomes important in the next stage process.

The Configuration Manager is responsible for setting up the project and stage structure in the CM system, which includes establishing the specifications for the stage build. This process is further described in the vendor’s operating manual for the selected CM system, as defined in the project SPMP.

FORMAL ITERATION PROCESS

The formal iteration process is focused on refining and finalizing stage deliverables. The analysts and/or developers involved in the generation of stage deliverables check out, edit, and check in items until they consider the items ready for inclusion in the stage build. The Configuration Manager performs the stage build when all items are flagged as ready for the build. The formal iteration process concludes when the build is successful.

IN-STAGE ASSESSMENT PROCESS

The in-stage assessment process focuses on reviewing and/or testing (as appropriate) the stage deliverables. In the case of reviews, the end result is either a concurring review or a non-concurring review with associated comments. In the case of testing activities, each test case execution results in the generation of either a Test Completion Report (TCR), or a Test Incident Report (TIR).

If one or more non-concurring reviews and/or TIRs are generated, the Configuration Manager evaluates the item and assigns an analyst or developer to correct the problem. The stage reverts to the formal iteration process when the review comments and/or TIRs are passed to the designated analyst or developer. At this point, the build is revised to include one or more of the changed items and the stage deliverables are re-built after the analysts and/or developers have finished revising the appropriate configuration items.

Note: Several builds may be executed before all TIRs are fully resolved. For example, one build may be devoted entirely to resolving a set of related TIRs, while another build may address a single TIR.

If all reviews and/or tests are successful, only Concurring reviews and/or TCRs are generated and passed to the Configuration Manager. Once all reviews and TCRs are in, the Configuration Manager promotes the build to a released stage baseline. At this point, the stage transitions to the Stage Exit process, as defined in the SDLC.