Software Engineering Group Projects – Operating Procedures and Configuration Management Standards

Author: C. J. Price , N.W. Hardy & B.P.Tiddeman
Config Ref: SE.QA.08
Date: 2014-09-23
Version: 1.8
Status: Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1 INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to specify procedures enabling all items produced by a group project to be properly controlled; and to provide a mechanism by which problems and corresponding changes to project items can be recorded.

1.2 Scope

This document describes the procedures, tools and techniques to be used for configuration management, and the procedures to be used for problem reporting and corrective action. It should be read by all members of the project group.

This document assumes the reader is already familiar with the QA Plan [1]. The Project Management Standards [2] should be read in conjunction with this document.

1.3 Objectives

- To describe a practical means of effectively managing all documents, diagrams, source code modules, executable systems and any other significant items that are produced by the project group and which are stored on computer.

- To describe procedures for notifying problems with any of these items,

- To specify the procedures which must be followed when a problem has been discovered: these will cover analysing the problem; identifying the solution, and causing corrective action to be taken.

2 SOFTWARE CONFIGURATION MANAGEMENT

2.1 Introduction

Configuration management is the management and control of all kinds of changes made to a system so that the state of each component is always known. Configuration management begins at the start of the project, and continues throughout development and on into the maintenance phase. It is not concerned just with source code, since all documents, such as the Design Specification, must be kept under control of the configuration management system. The use of GitHub is mandated, students should set up a private account and a user guide will be provided. The configuration management system will allow a user to:

1. retrieve copies of any version of a file, enabling recovery of previous or ‘old’ versions;
2. retrieve copies of any version of a directory structure, with its files, enabling recovery of previous version;
3. check in changes to the file, causing the changes to be recorded and the version number to be incremented;
4. inquire about differences between versions, obtain a log summarising the changes checked in for a particular version and produce a history of the file showing all changes and the users responsible.

Where the following items are produced on a project, they *must* be kept under the control of the configuration management system:

1. List of project deliverables;
2. Requirements Specification;
3. Design Specification, including any accompanying diagrams;
4. Test Specification;
5. User Manual;
6. Maintenance Manual;
7. End-of-Project Report;
8. Java source code and tools (e.g., shell scripts);

9. any other documents produced by the project as appropriate.

The QA Manager will be responsible for adherence to the configuration management procedures, and will directly control access to items in the configuration management system.

Building the software for delivery

In order to make the production of the deliverable system reproducible by others, you should produce and document ways to achieve the following:

1. compile units into class files
2. create a deployable file (e.g. a `jar`)
3. create `javadoc` files.

This may be done via shell scripts. A better solution is to use one or more makefile, ant file or similar to record and execute the commands for building the system. It is recommended that an IDE be used for the project, that is likely to generate such files semi-automatically. All such build scripts must be under version and configuration control. All testing and delivery must be on a system built using such files and controlled, along with the source, as a specified tagged revision under the version control system.

For systems built using languages other than Java, equivalent standards must be established by groups.

2.2 Configuration Items, References and Status

Configuration items are project items which are controlled by the configuration management system, and thus include the items listed in section 2.1 above.

Each configuration item should have a unique *configuration reference* assigned by the Project Leader. It will correspond to the task identifier that produces it, and will have the form:

`SE_groupid_taskid_subtaskid`

where:

1. `groupid` is the unique two digit group number assigned to the group at the start of the project (e.g., '07' for project group 7);
2. *taskid* is an upper case mnemonic uniquely identifying a major task - major tasks will correspond closely to the major activities outlined in section 3 above (e.g. use *PM* for project management, *DS* for design specification, TEST for testing activities);
3. *subtaskid* is a two digit number uniquely identifying each subtask of a major task, starting from 01 and using leading zeros when required (e.g. 03 represents the third subtask of a major task). If it is inappropriate to decompose a major task into multiple subtasks, then that task will have a single subtask labelled 01.

A configuration reference for the test specification (subtask 1 of the TEST task) belonging to project group 5 would thus be: `SE_05_TEST_01`

The QA Manager is responsible for the allocation of configuration references and for the maintenance of a file called `config_refs` in the configuration directory (see section 2.3).

Any item must have one of the following statuses:

1. Draft - the item (e.g., document, code) is currently under development;
2. For review - the item is ready for formal review or for testing;
3. Release - the item has successfully passed its test or review and is thus complete and correct.

Each item will be under the version control system. An item will progress from Draft, through For review to Release but may well return to an earlier status. It may fail its review or a released version may need to be corrected and reviewed again.

2.3 Directory Structures

One aspect of configuration management is the standardisation of the directory structure which hold the project data. This structure is managed by the version control system. Both files and the directory structure may change (typically grow) and this must be managed.

The *QA manager* will ensure that an initial structure is created. All members will check out a working copy. They will work in this structure and commit changes to the repository as appropriate. Working copies will have any generated items (processed documents, compiled code etc.) locally generated. Working files and directories which are not part of the final product (e.g. temporary output files) may also be present.

The directories described below should be present and additional directories may be created as required.

- **docs**. This contains documents produced by the group members. Each document should be in its own directory (see Section 2.2, "Configuration Items, References and Status " above).
- **lib**. This contains any third party libraries used by the project. The build file and/or the code itself ensures access to them. This is language dependent.
- **man**. management documents are stored in this directory. Specifically, there should be a minutes directory (see Section 2.4.3, "Managing Minutes of Meetings " below) and a plan directory. The plan directory will contain the project plan document.
- **src**. This contains the source code for the project, arranged into the relevant directories corresponding to their position in the package hierarchy or in other language defined arrangements. If a project requires more than one program in the system (see [3]) one directory, appropriately named, may be required for each.
- **test.** Test code is stored in this directory. Unit tests will be stored in a structure which reflects src, and test may be a subdirectory of src. Other test harnesses will be stored in directories related to the specific test at the top level of test .

IDEs will typically manage files directories at this level. For example, NetBeans will use a directory called nbproject; Eclipse will have a number of "dot" files. These must be under version control. The IDE will normally manage this. Directories generated by the IDE from source must not be under version control. For example, `build` or `dist` directories.

Released versions of deliverables should be tagged. Major changes to released versions should be made in branches. The QA manager is responsible for managing the structure of the branches and tags directories to support this.

2.4 Managing Documents

2.4.1. Managing Documents/Code During Their Initial Development

When documents are initially created, they should be developed in the appropriate place in the author's working directory. The author will repeatedly edit the file, and check it into the repository as convenient and when ever sharing with other developers is necessary.

When the document is ready for review or testing, its status is updated to *For review* an it is checked into the repository. Review or testing is the carried out on an identified revision number to ensure that all work is carried out on the same version. When it has been approved, the status is updated to Release. If it is not approved, its status may be reduced to Draft while changes are made, or a new revision may be created if minor changes require only one edit.

2.4.2. Updating Documents/Code

Team members may all have working copies of any or all of the repository. They are responsible for ensuring that they have the correct version. Major changes are likely to take place through version control branch. When this branch is merged to the trunk, developers may need to update. Developers may be working with an older version (parallel engineering) and a tagged version will normally then be employed.

2.4.3. Managing Minutes of Meetings

Minutes of meetings will be stored in the man/minutes. directory. There must be one file per meeting, and the file name must have the following form:

y*yyy-mm-dd_minutes*

(plus any file extension) where:

- *yyyy* is the year;
- *mm* is a two digit number representing the month, using leading zeros where required (January = 01);
- dd is a two digit number representing the day of the month using leading zeros where required.

e.g., the minutes of the regular weekly meeting of the project group held on 6th January 2012 might be saved in a file called 2012_01_06_minutes.txt. The advantage of using this naming convention is that when the file names are listed, they can be made to appear in order of the date.

See [4] for a description of the format of minutes.

2.5 Managing Code

The procedures to be used for managing code having *Draft* or *Release* status are the same as those for managing documents, except that:

1. exploratory code for personal development an risk management may be stored outside the repository;
2. code to be allocated Draft status must have passed an informal check against the Design Specification and must adhere to the Coding Standards [5] and must compile and must have passed unit tests;
3. code to be allocated Release status must have passed a software verification review [6] and any relevant tests specified in the Test Specification [7].

3 PROBLEM REPORTING AND CORRECTIVE ACTION

Problems relating to configuration items, including specifications, design diagrams, and code must go through a formal problem reporting and action process. This should be managed through the online project management / bug reporting / ticketing system built into GitHub ticketing system.

3.1 Online Reporting and Corrective Action

Project teams should run all reporting and corrective action online through the project management tools built into github. In order to do so, teams should use the usual conventions of the system. These include setting up milestones, creating issues for unresolved bugs or design change requests. It is suggested that one member of the group familiarises themselves with the system and then explains and demonstrates it to the group tutor and the rest of the group. The QA manager should monitor the use of the system and ensure that it is being used correctly. For example, testers and developers should submit issues for identified bugs or change requests. The project manager (or designated deputy) should approve or reject the issue and if accepted assign it to a developer. When the developer has implemented the fix or feature it should be submitted for testing before final approval (issue closed by project manager or deputy).

REFERENCES

[1] *Software Engineering Group Projects* . QA Plan. C. J. Price and N. W. Hardy. SE.QA.01. 1.9. Release.
[2] *Software Engineering Group Projects* . Project Management. C. J. Price. SE.QA.02. 1.9. Release.
[3] *Software Engineering Group Projects* –Design Specification Standards. C. J. Price, N.W.Hardy and B.P.Tiddeman. SE.QA.05A. 1.7. Release
[4] *Software Engineering Group Projects* . General Documentation Standards. C. J. Price and N. W. Hardy. SE.QA.03. 1.7. Release.
[5] *Software Engineering Group Projects* . Java Coding Standards. C. J. Price, A. McManus, and N.W. Hardy. SE.QA.09. . 1.7. Release.
[6] *Software Engineering Group Projects* . Review Standards. C. J. Price. SE.QA.07. . 1.6. Release.
[7] *Software Engineering Group Projects* . Test Procedure Standards. C. J. Price and N. W. hardy. SE.QA.06. 1.8. Release.

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	30/10/00	Document given complete rehaul	CJP
1.1	N/A	18/07/02	Changed after review with Graham Parker - PRFs removed from process - filestore structure simplified	CJP
1.2	N/A	24/09/03	Added ref to Ant.	CJP
1,3	N/A	26/09/04	BN9 - resolve who issues CCFs	CJP
1.4	N/A	12/09/08	Changed document template to be Aber Uni	CJP
1.5	N/A	10/10/10	Unpdated to Docbook. SVN added. Consequential changes, including "For Review".	NWH
1.6	N/A	01/10/12	Reverted to word doc. Adjusted for possibility of using online project management.	BPT
1.7	N/A	25/09/13	Adjusted for use of github. Removed references to CCFs and replaced with brief discussion of github issue tracking system	BPT
1.8	N/A	23/09/14	§2.3 expanded to include multiple programs	NWH