# Software Configuration Management Plan

Eindhoven, October 13, 2009 SCMP-2.0.734

Technische Universiteit **Eindhoven** University of Technology

**Where innovation starts**

**Project Manager:**
Wilco Belgraver Thissen, 0514143

**Quality Assurance Manager:**
Jelle Hellings, 0592127

**Senior management:**
Mark van den Brand, HG 5.59
Lou Somers, HG 5.36

**Advisor:**
Erik Luit, HG 7.12

**Customer:**
Natalia Sidorova, HG 7.84

**Project team:**
Roy Berkeveld, 0608170
Gijs Direks, 0611093
Michael van Duijkeren, 0535368
Neal van den Eertwegh, 0610024
Dion Jansen, 0590077
Koen Kivits, 0608715
Sander Leemans, 0608896
Kevin van der Pol, 0620300
Nick van der Veeken, 0587266

*Computer Science, TU/e*

## Abstract

This document is the Software Configuration Management Plan (SCMP) of the Group QIS project. This project is part of the Software Engineering Project (2IP35) and is one of the assignments at Eindhoven University of Technology. The document complies with the SCMP from the Software Engineering Standard, as set by the European Space Agency[1]. This document contains information on the standards to be used for writing the documentation required for this project, as well as information about the processing and storage of these documents.

# Contents

# Document Status Sheet

## Document status overview

### General

| | |
|---|---|
| Document title: | Software Configuration Management Plan |
| Identification: | SCMP-2.0.734 |
| Author: | gdireks, rberkeveld |
| Document status: | Final |

### Document history

| *Version* | *Date* | *Author* | *Reason of change* |
|---|---|---|---|
| 0.0.1 | 08-09-2009 | rberkeveld | Initial version |
| 0.1.138 | 15-09-2009 | gdireks | Included remarks QM made. |
| 0.2.196 | 17-09-2009 | rberkeveld | Included remarks QM made. |
| 0.3.200 | 17-09-2009 | rberkeveld | Included remarks QM made. |
| 1.0.203 | 17-09-2009 | rberkeveld | No change - approved as final. |
| 1.1.696 | 09-10-2009 | rberkeveld | No content change - adapted to recent formatting. |
| 1.2.731 | 13-10-2009 | rberkeveld | As requested by QM. |
| 2.0.734 | 13-10-2009 | rberkeveld | One more QM remark, approved as final. |

## Document Change Records since previous issue

### General

| | |
|---|---|
| Datum: | 2009-10-13 |
| Document title: | Software Configuration Management Plan |
| Identification: | SCMP-2.0.734 |

### Changes

| *Page* | *Paragraph* | *Reason to change* |
|---|---|---|
| (Many) | (Many) | Various markup and grammar details. |
| 12 | 4.1 | Rephrased to match enforced practice. |

# Chapter 1

# Introduction

## 1.1 Purpose

The purpose of this document is to provide rules and guidelines for the storage and of all documents that are created during this project. Additionally it will give naming and layout conventions for all documents and describe the identification of all major documents.

## 1.2 Scope

This document is about:

- Listing the specific documents which need to be written during the course of the project
- Giving naming conventions for these documents
- Providing a structured way to create, store and update the documents
- Providing a standard layout for these documents

This document will not describe the detailed contents of the individual documents.

The Configuration Items (CIs) that will be written during the project are:

- Architectural Design Document (ADD)
- Detailed Design Document (DDD)
- Software Configuration Management Plan (SCMP)
- Software Project Management Plan (SPMP)
- Software Quality Assurance Plan (SQAP)
- Software Requirements Document (SRD)
- Software Transfer Document (STD)
- Software User Manual (SUM)

- Software Verification and Validation Plan (SVVP)
- User Requirements Document (URD)
- Code
- Minutes
- Test plans for Unit Test (UT), System Test (ST), Integration Test (IT) and Acceptance Test (AT)

## 1.3 List of definitions

| | |
|---|---|
| ADD | Architectural Design Document |
| AT | Acceptance Test |
| ATP | Acceptance Test Plan |
| CI | Configuration Item |
| CM | Configuration Manager |
| DDD | Detailed Design Document |
| ESA | European Space Agency |
| IT | Integration Test |
| PM | Project Manager |
| QM | Quality Assurance Manager |
| SCMP | Software Configuration Management Plan |
| SEP | Software Engineering Project |
| SPMP | Software Project Management Plan |
| SQAP | Software Quality Assurance Plan |
| SRD | Software Requirements Document |
| ST | System Test |
| STD | Software Transfer Document |
| SUM | Software User Manual |
| SVVP | Software Verification and Validation Plan |
| SVVR | Software Verification and Validation Report |
| URD | User Requirements Document |
| UT | Unit Test |

## 1.4 List of references

[1] ESA Board for Software Standardization and Control (BSSC). European space agency software engineering standards. February 1991. (ESA PSS-05-0 Issue 2).

[2] Group QIS. Software project manangement plan. Technical report, Eindhoven University of Technology, Computer Science and Engineering, sep 2009.

[3] Group QIS. Software quality assurance plan. Technical report, Eindhoven University of Technology, Computer Science, September 2009.

[4] Group QIS. Software verification and validation plan. Technical report, Eindhoven University of Technology, Computer Science, September 2009.

# Chapter 2

# Management

## 2.1 Organization

The persons that are directly involved in the Configuration Management are the configuration manager and the vice configuration manager. Their names are stated in the Software Project Management Plan[2] (SPMP).

## 2.2 Responsibilities

The configuration manager should provide a neatly working environment for configuration management at all times. Any problems should be reported as soon as possible.

The (primary) configuration manager is the first responsible for configuration management, however he may delegate tasks to the vice configuration manager. Also, whenever the configuration manager is not available or unreachable, the vice configuration manager should take over his tasks. Furthermore, all group members are responsible for their own documents. This includes updating the document status sheet; see also section 5.

## 2.3 Interface management

In case failure of any hardware supplied by the University the CM or vice CM will contact 'Bureau Computer Faciliteiten' (BCF), as they are assumed to have expert knowledge of the hardware configuration of the supplied hardware.

Should the development server fail, the CM should be contacted instead as a custom operating system has been installed. A reboot of the server will not automatically resume all operations, so this should be prevented and only done under supervision of the CM.

## 2.4 SCMP implementation

Contrary to the ESA Software Engineering Standard[1] there won't be a separate SCMP document for each phase of the project. Instead this document will be updated with appendices for every phase of the project. For more information concerning planning of the phases, refer to the SPMP.

## 2.5 Applicable procedures

All the documents are subject to the standards described in the ESA standard and must also adhere to the requirements as described in the SQAP[3] and the SVVP[4].

### 2.5.1 Formal documents

Formal documents are written in LaTeX and should start with the following:

```
%\def\docisfinal{} % Uncomment this line temporarily for Final releases.
% Activate one of the following, or leave for Draft or Final
%\def\docstatus{Internally approved with proposed changes}
%\def\docstatus{Internally approved}
%\def\docstatus{Conditionally approved}
%\def\docstatus{Externally approved}

\input{../style.tex}
% The following line is generated, do not modify!
\svnInfo $IdSVNINFO$

%Variables
\def\doctitle{DOCTITLE}
\def\doctitleabbr{DOCABBR}
\def\docversion{VERSION.\svnInfoRevision}
\def\docid{\textsc{\doctitleabbr-\docversion}}

% Nonstandard settings (extra packages, commands, defenitions)
OTHERDEFS

% Document contents
\begin{document}
\input{../titlepage.tex}

\begin{abstract}
```

With DOCTITLE representing the title of the document, DOCABBR the common abbreviation the document is referenced by, and VERSION the version code of this document. SVNINFO is automatically added by Subversion and should be left blank on initial documents.

OTHERDEFS may be used for document-specific defenitions, commands, included packages, et cetera.

For the includes to work, these documents must be located in appropriate subfolders of the `/trunk/doc` folder.

# Chapter 3

# Configuration identification

## 3.1 Naming conventions

The name of a file in the trunk should be a good description of its contents, followed by an extension. The entire name should be in lower case. In `/trunk/doc/` this usually is the name of the document (`scmp.tex`, `scmp.pdf`). In `/trunk/src/` this usually is the name of the module (`io.py`, `math.pyo`).

Documents in the master library will have the same name as they had in `/trunk/doc/`. When the master copy gets moved to the archive library, it will be renamed to `<name>-<version>.<ext>`, ie. `scmp-1.2.1.pdf`.

The version will be determined as follows:

- Version: x.y.z
- Inital version: 0.0.revision, the revision will be updated by Subversion automatically.
- After internal changes: x.y.z becomes x.y.revision
- After internal review: x.y.z becomes x.y+1.revision
- After formal review: x.y.z becomes x+1.0.revision

## 3.2 Baselines

Baselines are documents that have been externally reviewed and approved. They will be stored in the Master library (which will be discussed in section 4.1.2). According to the ESA[1] standard new versions of the management documents need to be created for every stage in the project. Because of the small scale of this project the same management documents are used during the course of the project. Information specific for a stage in the project will be added to these documents in the form of appendices.

# Chapter 4

# Configuration control

## 4.1 Library control

This section describes the central storage facility for all CI's. All CI's are stored in at least one of three libraries: the development library, the master library or the archive library. Every CI has a subdirectory of the SVN repository that contains all versions of all files related to the CI.

The general structure of the repository is as follows:

- `/trunk/` Contains everything as a default. The idea is to only commit things that are useful the way they are committed, and to only commit updates that are correct (i.e. they work and are somewhat tested).

  This folder should NOT contain any files that aren't directly useable.

  For anything dangerous or unverified, please see the branches.

  In `/trunk/` itself you should only commit general project information such as a readme (for use of SVN) or contact and group information. All other files should be categorized in subdirectories.

- `/trunk/master/` The master library. This will contain externally reviewed and approved documents only. Only the CM and vice CM may commit in this folder.

- `/trunk/archive/` The archive library as described below. Also contains files delivered by the customer which have historic value (such as old databases). Every time a document in the master library is updated, the overwritten version of that document will be stored here. The version number will be added to the file or directory name.

- `/trunk/doc/*/` All documents which we write or have been provided for use. Only design files (.tex files and images, so no compiled pdf's) are committed.

  This folder represents the document CI's of the development library.

- `/trunk/doc/*/release/` The current "release" version of a document, rendered and finalized (this means the pdf file). After internal review, the pdf file (and the only that file!)

is committed here, after which a tag can be made to `/tags/doc/*/<versionnr>`. This folder is used for tagging purposes and its contents may change arbitrarily.

- `/trunk/internal/` Contains various misc. documents for internal use. Some of them are contact and absence/presence information, references to the systems we use and an overview of important dates and deadlines.
- `/trunk/meeting/` Contains agenda's, minutes and everything else specifically related to the meetings.
- `/trunk/src/` Contains all working code. All code that is yet to be tested should be done in a branch. The exact structure will be explained once more details about the programming environment are clear (such as frameworks and architecture). This will be when we enter the Design phase of the project.
- `/branches/` Contains branches, every directory should be a branch. You can branch everything in `trunk`, you can even branch branches. Make sure the name (folder) is descriptive.

  Note that this folder represents most of the source code part of the development library.

  For further explanations, see our Branching policy (6.1.3).
- `/tags/` Every document that is in any way complete (milestones, etc.) should be tagged with an appropriate version number. The `/tags/doc/*/` will contain internally reviewed documents. Any working, stable version of the code can be committed in `/tags/src/<ID>`.

  This is NOT a place to add files. You should tag existing directories here (`svn copy`). Please read the tagging policy (6.1.3).

### 4.1.1 Development library

The development library contains CI's that are under construction, and CI's that are not official project documents.

### 4.1.2 Master library

The second library, the master library, contains all approved baselines. Only the CM may put documents here and storage may only take place after the document has been reviewed and approved externally. Everybody is free to make (read-only) copies of documents in this library, but these copies may not be put back into this library under any condition.

### 4.1.3 Archive library

The archive library contains CI's that have been externally released and approved. This library will also contain externally supplied files (such as an old databse) which won't change. CI's in this library cannot be changed under any circumstances. Only the CM can put CI's in the archive library, and again as a CI is put into this library it will not be deleted. New versions may only be added when a reviewed and approved version will replace a version in the master library.

## 4.2 Media control

We use subversion as the primary storage media. Our subversion repository is run by the BCF. Backups are also taken care of by the BCF. In addition to this, our own development server performs an `svn update` every 5 seconds, such that the latest revision is always present on some system not in control of the BCF. This server will be used as a backup for the SVN repository as well as running scripts that need to be tested. Every group member has write access to the full repository, but since SVN keeps revision data of all changes, it is trivial to recover past revisions after change has been made. In addition, a usage policy is provided in Section 4.1.

Trac is used for tracking tickets and general project progress. Its contents are backed up by the BCF. No alternate backup procedures have been put in place.

All mail sent via the mailing lists is stored in archives on the listserver. The listserver is run by the Unix committee. We have no information on whether the archives are backed up or not, but everyone receives a copy of all mail sent, so this should be sufficient.

## 4.3 Change control

This section defines the procedure that needs to be followed for making changes to any project document.

### 4.3.1 Development library

The change control system SVN provides will be used to track (or undo) any changes made to source code and documents.

### 4.3.2 Master library

Once a CI is externally approved the CM can put it in the master library. If an author wants to make changes to a document inside the master library, then that author has to take up contact with the QM. The QM will call up a review meeting where all the changes are approved or rejected. More information regarding the change procedure can be found in the SVVP. When changes to a CI are approved the CM will put a copy of the existing version of the CI in the master library to the archive library, and subsequently he will put the new version of the CI in the master library. As the CM is the only one allowed to remove/create new documents in the master library, there is no need for change control.

The master library is located in `/trunk/master`.

### 4.3.3 Archive library

CI's in this library cannot be modified under any condition. New versions may only be added when a reviewed and approved version will replace a version in the master library, or if an external source provides a relevant, nonchanging file (such as an old database). Since documents in this library cannot be modified and may only be added by the CM, there is no need for change control.

The archive library is located in `/trunk/archive`.

# Chapter 5

# Status accounting

CI's in such a state that they can be put in the master library have to be reported 'ready' by the author of that CI. The CM then copies the document to the requested library—hereby possibly removing a previous version of that CI—and record the version tag, and the date of addition in the status-log. This status log consists of two files with the name LOG.TXT which are located in the master library and in the archive library. Each log contains only information about the CI's in the library that the log is in. Furthermore all documents contain a document change record in which all changes with respect to the previous version are recorded.

Reporting a CI as 'ready', can be done by creating a ticket via Trac, listing the files (or folders) affected. Most of the time, this is the CI's release directory. Direct contact with a CM (via email, phone or just talking) is also a valid option to report a CI as ready.

Note: Changes in the development library will only be recorded informally through SVN Commit messages, and possibly tickets.

# Chapter 6

# Tools, techniques and methods

## 6.1 Subversion

Subversion, or SVN, is an open-source revision control system.

### 6.1.1 Clients

There are two common SVN clients available. The command-line client application `svn` (referred to as SVN further in the document) and the graphical Windows-only shell extension known as TortoiseSVN (referred to as TORTOISESVN. Use of both these clients is described. There are other clients, but their usage falls outside the scope of this document. Often simmilar terminology is used, so these should be simmilar in use.

### 6.1.2 Installation

TORTOISESVN can be downloaded from the TortoiseSVN website[1].

SVN is mostly used on systems which have some form of package management. You can install it directly via your operating system, or you can get it from the Subversion website[2].

### 6.1.3 Operations

We go into detail on some operations here, with remarks specific to our set-up.

For those new to SVN, after checking out, you need to update your repository to pull the most recent changes. You also need to commit changes before they can become visible to others. These and other more general operations are properly explained at the TortoiseSVN website.

[1]`http://tortoisesvn.tigris.org/`
[2]`http://subversion.tigris.org/`

### Checking out

A one-time operation, meant to populate your drive with the contents of the repository. Afterwards, you can perform all listed commands on the repository. Your local copy is called a working copy. Changes will only be propagated after you commit them.

Where a username and password are requested, please provide the credentials you received by mail.

For SVN: `svn checkout https://svn.win.tue.nl/repos/SEP2010-1/ svn`

For TORTOISESVN: Open Microsoft Windows Explorer in the target folder, right-click it, click CHECKOUT, fill in the URL `https://svn.win.tue.nl/repos/SEP2010-1/` and further authentication information and confirm by clicking OK.

### Branching

Whenever development starts on a source code module currently present in `/trunk/src/`, you should branch. Whatever you do, do not commit changes directly in `/trunk/src/`, trunk should never break. Branching is generally done on the complete source tree.

Branching for documents is not needed, unless modifications are made which might break stuff, such as introducing new layouts or odd packages.

Whomever breaks anything in `/trunk/` will be publically humiliated.

In brief, a branch is a copy of (part of) `/trunk/` that you can modify at your leasure. In branches you can perform the ugliest of ugly modifications, and commit as often as you like. Just make sure that when all modifications are complete, everything is coherent, and an improvement on what we previously had.

Branching can be done by performing an `svn copy`. The procedure is as follows:

SVN: `svn copy trunk/src branches/blamybranch`

TORTOISESVN: Right-click on the folder to branch, choose SVN $->$ Branch/Tag. Select an appropriate (new) folder in `/branches/` and you can perform any edits on the folder you just filled.

Note that these branches are cheap copies. They do not take any additional space on the server, even though the files are physically duplicated on your own hard drive. You do not need to worry about making too many branches.

To limit local disk usage to some degree, we should branch either `/trunk/src` and `/trunk/doc` instead of the entire `/trunk` folder. You are, however, free to make branches of everything, should you really want to.

Whenever `/trunk` changes, your branch will not automatically be changed. To adapt these changes in your branch, you'll have to merge the current trunk with your branch. This is usually done like this:

SVN: From the current branch folder: `svn merge ../../trunk/src`

TortoiseSVN: Right-click, SVN $->$ Merge.

Make sure that during a merge, the working copy is updated and all changes that you might have added are commited. Whenever a merge fails and you need to revert, all changes which haven't been commited yet would be discarded. Also, merge as early and often as possible to prevent later conflicts!

Afterwards, when development on a branch has completed sufficiently, it can be merged the other way. Just remember to only do this after a review process has determined that everything still works and that the changes are acceptable. Before you merge to trunk, you should also merge all recent additions to your own branch such that your new branch effectvely becomes the new trunk, and a merge to trunk becomes trivial since there are no more conflicts.

Note: do not work for too long on a single branch. At some point, a branch might differ significantly from the current trunk, becoming a nearly unmergeable pile of changes. Make sure you merge recent changes to and from trunk often. Just make sure changes are verified and keep modifications relatively small and quick.

You should commit directly after making a branch, and also just before and after a merge, with appropriate commit messages. Often, you'll want to update some tickets in Trac as well.

After you are done with a branch, and after it has been approved and merged with `/trunk`, you should remove it from the repository to free local space and reduce clutter.

#### Tagging

By tagging part of the repository, you indicate that that version of what is tagged will be used for something. Before tagging, always make sure to create a ticket detailing the purpose of the tag, such that the people that are going to work with it can see the specifics.

Making a tag is just another `svn copy` into the `/tags` folder, just like with branching, but unlike with branching, contents of a tag should never be changed.

## 6.2 Trac

Trac is an issue tracking system that integrates nicely with Subversion. Whenever we say a ticket should be made, Trac is used for this.

Trac allows for assigning tickets to specific milestones or components. Milestones are deadlines to be made, or certain important points in the development of the project. Components are parts of the project that can be worked on (certain documents or pieces of code).

You can access Trac via the following URL: `https://svn.win.tue.nl/trac/SEP2010-1`.

Whenever a new component or milestone is identified that is not yet listed in Trac, a CM should be notified who will add the it in Trac.

Whenever Trac starts running out of unresolved issues, you should start testing and reviewing. You can then make some tickets with things you'd like to see changed. Those tickets will be assigned to group members, who are then tasked with resolving them. Don't worry about creating tickets for minor things or things you are not quite sure of, those are discarded just as easily as you can write them, and at least someone will verify your claim.

Open tickets will be resolved by the group members to whom the tickets are assigned. Ticket updates are closely related to changes on SVN, therefore SVN allows for the updating of tickets by writing special SVN Commit messages. Any commit message which is of the form `<command> #<ticket number>` is an instruction to Trac. The commands `refs`, `fixes`, `closes` are understood. As an example, the commit message "`Changed blah and foo to do this or that. Fixes #10 and #12, and refs #12.`" would close #10 and #12, and add a note to #12. Note that fixes and closes perform the exact same operation internally.

## 6.3 Programming language

The language that has been chosen for web application development is Python. Coding standards are available in the SQAP document[3].

The specific version of Python we will be using is 2.6.2. A language reference is available at the Python documentation website[3]. A general introduction to web programming in Python is also available[4].

We will also be using the Django Web framework[5] for Python. We will be using version 1.1 of this framework. Documentation is available[6], as well as a general tutorial[7].

## 6.4 Mailing list

The mailing list is used for general announcements which are either urgent or of interest to everyone. Every group member is subscribed, managing subscription details can be done via `http://listserver.tue.nl/mailman/listinfo/sep2010-1`. The mailing list creates weekly archives which can be viewed via another page[8].

The mailinglist is set up using Mailman, meaning you can configure by either sending commands or by visiting the configuration website. External documentation on Mailman is available from the Mailman website[9].

[3] `http://docs.python.org/reference/`
[4] `http://wiki.python.org/moin/WebProgramming`
[5] `http://www.djangoproject.com/`
[6] `http://docs.djangoproject.com/en/dev/`
[7] `http://docs.djangoproject.com/en/dev/intro/tutorial01/#intro-tutorial01`
[8] `http://listserver.tue.nl/mailman/private/sep2010-1/`
[9] `http://www.gnu.org/software/mailman/index.html`

## 6.5 Latex

All documents will be written using the LaTeX typesetting system in combination with BibTeX. We assume that each group member knows LaTeX and has a working environment. If not, more information can be requested from the CM's.

The non-default LaTeXpackage SVNINFO should be installed to ensure correct compilation of the documents.

# Chapter 7

# Supplier control

Refer to the SQAP for the demands that are placed on tools supplied by external sources.

# Chapter 8

# Records collection and retention

Since we are using SVN as the primary storage facility, complete deletion of documents is impossible. This means that all documents will be kept for the whole duration of the project. In general, documents may be modified and deleted from the repository as group members see fit, with the following exceptions:

- Documents in the master library (`/trunk/master`) should never be modified by anyone other than a CM.
- Documents in the archive library (`/trunk/archive`) should never be deleted or modified by anyone. Only a CM can add to these directories.
- Tags (in `/tags`) may never be deleted.

The only reason to override this policy is when a severe error has been made. And only a CM may perform such corrections.