TU/e

Project FINGERPAINT

SCMP-1.0

Software Configuration Management Plan

Authors:
Tessa Belder (0739377)
Lasse Blaauwbroek (0749928)
Thom Castermans (0739808)
Roel van Happen (0751614)
Benjamin van der Hoeven (0758975)
Femke Jansen (0741948)
Hugo Snel (0657700)

Junior Management:
Simon Burg
Areti Paziourou
Luc de Smet

Senior Management:
Mark van den Brand, MF 7.096
Lou Somers, MF 7.145

Technical Advisor:
Ion Barosan, MF 7.082

Customer:
Patrick Anderson, GEM-Z 4.137

Eindhoven - June 23, 2013

Abstract

This document is the Software Configuration Management Plan (SCMP) of the FINGERPAINT 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-1.0
Author:	Thom Castermans
Document status:	Externally approved

Document History

Version	*Date*	*Author*	*Reason of change*
0.0	02-May-2013	Thom Castermans	Initial version.
1.0	21-June-2013	-	Externally approved.

Document Change Records Since Previous Issue

General

Date:	21-June-2013
Document title:	Software Configuration Management Plan
Identification:	SCMP-1.0

Changes

Page	*Paragraph*	*Reason to change*
-	-	Externally approved.

Chapter 1

Introduction

This chapter explains the purpose of this document as well as what the scope of this document is, that is, how it is related to other documents in the project.

1.1 Purpose

The purpose of this document is to set rules and guidelines to which all project members should adhere. This concerns the versioning, identification and layout of all documents that are created for this project. All major documents should adhere to strict rules, while for other documents, such as files containing code, guidelines are set that are more loose.

This document should be read as a reference. It can be used when a developer or project member is not sure about how to do something as a mainstay.

1.2 Scope

In this project, the following configuration items (CIs) will be produced:

- 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;

- Test plans for a number of phases. In particular:
  - Unit Test Plan (UTP);
  - Integration Test Plan (ITP);
  - Acceptance Test Plan (ATP).
- Product Backlog.

The ESA standard mentions a System Test Plan (STP) as well, but in our case this can be omitted.

1.3 List of definitions

2IP35	The Software Engineering Project
ADD	Architectural Design Document
ATP	Acceptance Test Plan
BCF	Bureau for Computer Facilities
CI	Configuration Item
CM	Configuration Manager
DDD	Detailed Design Document
ITP	Integration Test Plan
SEP	Software Engineering Project
SCMP	Software Configuration Management Plan
SPMP	Software Project Management Plan
SQAP	Software Quality Assurance Plan
SRD	Software Requirements Document
STD	Software Transfer Document
STP	System Test Plan
SUM	Software User Manual
SVVP	Software Verification and Validation Plan
TU/e	Eindhoven University of Technology
QM	Quality Manager
URD	User Requirements Document
UTP	Unit Test Plan

1.4 List of references

[1] ESA, *ESA Software Engineering Standards*. ESA, March 1995.

[2] Group Fingerpaint, "Software project management plan," *SEP*, 2013.

[3] Group Fingerpaint, "Software quality assurance plan," *SEP*, 2013.

[4] Group Fingerpaint, "Software validation and verification plan," *SEP*, 2013.

Chapter 2

Management

This chapter specifies which project members are involved in configuration management. Also, the responsibility a team member has in a function involved with configuration management is explained. Then, some general responsibilities are explained. Finally, a template for the creation of documents is given and some conventions are set about document creation, to ensure a consistent layout.

2.1 Organisation

The team members involved in configuration management are the configuration manager (CM) and vice CM. The project members that have volunteered to fulfil these roles are named in the SPMP [2].

Other group members should always assist the CM and vice CM.

2.2 Responsibilities

The CM and vice CM are responsible for copying documents to the master and archive library at the right moments, as mentioned in chapter 4. They are in general responsible for the contents of the master and archive library. Another task for them is creating and updating document templates, although this task can be delegated.

The CM is primarily responsible for configuration management, although he or she can delegate tasks to the vice CM, in which case the vice CM is responsible. Whenever the CM is (temporarily) not available, the vice CM should take over the tasks of the CM, including the responsibility for these tasks.

Finally, all project members are responsible for the documents they work on. This means that they update the document status sheet and make sure the latest version of the document they work(ed) on is available in the development repository (refer to chapter 4, section 4.1.1). When multiple group members work on the same document, they share the responsibility and additionally are responsible for the combined consistency of the document. Also, they should make sure that the repository remains in a "workable" state. That is, they should solve possible merge conflicts together.

2.3 Interface Management

The FINGERPAINT application will be developed using an external virtual server provided by the BCF. In case of failure of this server, the CM will contact BCF and let them resolve the issue. BCF is in this case supposed to have expert knowledge and have the means to resolve issues.

In general, the CM can help other project members when they have trouble with some software that is used (refer to chapter 6). However, the CM may delegate this task to other group members who have more expertise on the subject.

2.4 SCMP Implementation

In this project, we will have only one SCMP document, contrary to what is described in the ESA standard [1]. Thus, this document will not contain a planning for every phase of the project. Instead, refer to the SPMP for the planning of the phases.

2.5 Applicable Procedures

Every non-code document has to be created using LaTeX and should use the `fingerpaint.cls` document class by declaring the following in the beginning of the document (with other values for the options of course):

```
\documentclass[%
    pathtobase=..,%
    titlefull={Full Document Title},%
    titleabbr=FDT,%
    version=0.1]{fingerpaint}
```

Every document should start with a `\maketitle{}` call, followed by an abstract and then the `\tableofcontents`. This to ensure a consistent style among all documents.

Each document should have a "main" `.tex`-file that `\input`s separate `.tex`-files that each contain a chapter (or appendix). This enables project members to work on the same document more efficiently, as working on different chapters will not cause any merge conflicts.

So, a standard "main" `.tex`-file should look like as is shown in figure 2.1.

As noted in section 2.3, the CM can assist when project members experience problems with LaTeX.

Finally, all documents are subject to the standards described in the ESA standard [1] and must also adhere to the requirements as described in the SQAP [3] and the SVVP [4].

```
\documentclass[%
    pathtobase=..,%
    titlefull={Full Document Title},%
    titleabbr=FDT,%
    version=0.1]{fingerpaint}

\begin{document}

  \maketitle{}

  \begin{abstract}...\end{abstract}

  \tableofcontents

  \input{history.tex}
  \input{chapter1.tex}
  \input{chapter2.tex}
  ...

\end{document}
```

Figure 2.1: Example of how a document that uses the `fingerpaint.cls` document class should look like.

Chapter 3

Configuration Identification

In this chapter, a versioning scheme is set. All documents created for the FINGERPAINT project should adhere to this scheme.

3.1 Naming Conventions

All documents have a unique identifier. This identifier is `title abbreviation-version`, for example `URD-0.1`. The initial version of every document is `0.0`. Then, after every formal review, the version number is bumped up with `0.1`. A document that has been reviewed three times thus has version number `0.3`. Only when the client or management has approved a document, the version number is bumped up to `1.0`. Basically, the version number will not change after that, but it is theoretically possible that after that, some more changes are required and versions `1.x` are created. After a second final approval (note that the fact that there is a *secondary* approval that is *final* already indicates that this is an exceptional situation), the version will become `2.0`, et cetera.

Changes noted in the document status sheet in every document only mention changes since the last version. Older versions of the document can be found in the master (and archive) library (which will be discussed in more detail in chapter 4), so all changes leading to the current version of a document can at all times be retrieved. In practice, even between-version changes can be retrieved from the development library, but this functionality will probably not be needed.

3.2 Baselines

A baseline is a document that has been reviewed and approved externally. Baselines are stored in the master (and archive) library, as discussed in chapter 4. As described in the ESA standard [1], the CM makes sure that any version of every document can be directly downloaded from or rebuilt from the various libraries.

The ESA standard prescribes that new versions of management documents need to be created for every phase of the project. However, as the FINGERPAINT project is a relatively small project, we will have only one version of every document, including management documents, for the complete project. Phase-specific information will be added in the form of appendices to documents if needed.

Chapter 4

Configuration Control

This chapter describes how we handle different versions of CIs and where they are stored: we introduce the concept of libraries. Moreover, we describe how different libraries interact and what the role of the CM is in the management of these libraries.

4.1 Library Control

All CIs that are created for the FINGERPAINT project have to be stored somewhere. We call a place where CIs are stored a *library*. There are three libraries, that will be discussed in more detail in this section.

4.1.1 Development Library

The development library is the library where all CIs are stored initially. Documents in this library are generally under construction and can thus change a lot. From the development library, *all* versions of a CI that are stored in it can be retrieved and thus, every modification to a CI can be undone at any time.

In practice, the development library is split up in multiple Git repositories. Git is discussed in more detail in chapter 6. We have the following repositories:

- `project-code`: This repository contains all code that makes up the FINGERPAINT application.
- `project-docs`: This repository contains all documentation that is potentially relevant for the client. That is, all CIs that are related to the FINGERPAINT application and *not* to the SEP that led to the creation of that application are stored here.
- `sep-docs`: This repository contains all documentation that is strictly relevant to SEP. That includes the roles that project members fulfil, testing plans, this CI and the SQAP for example.

The idea of this split is that we have the possibility to hand over the software to the client by simply providing the first two repositories in the above list. These repositories will then be "clean" in the sense that they only contain the code and its documentation, nothing that is purely SEP-related.

All repositories are stored on and accessible through GitHub (discussed in section 4.2 and chapter 6).

4.1.2 Master Library

In the master library, CIs that are externally approved are stored. It may of course be the case that a CI is stored in the development library, but not in the master library, when there is no externally approved version (yet). In practice, this library is a folder on the website of the FINGERPAINT application[1]. Contrary to the development library, that only contains code and `.tex`-files that can be compiled to PDF files not present in the library, the master library only contains code and PDF files that are the result of running LaTeX on files from the development library.

The master library also contains a page that presents an overview of all documentation-related CIs stored in it. Furthermore, there will be a link to the development library *at that version*, so that the source files of CIs in the master library are easily accessible as well. Code is accessible through a link to the development version.

A CI can be placed in the master library only with permission of the CM and only after the CI has been reviewed and approved externally. Documents in the master library can be downloaded freely: this is the reason why the master library is accessible on a website. However, putting CIs in the master library is, again, something that can happen only with permission of the CM.

CIs cannot be deleted from the master library, but can be replaced with a newer version. In that case, the older version is moved to the archive library.

4.1.3 Archive Library

Like the master library, the archive library is a folder on the website of the FINGERPAINT application. Thus, this library is accessible through the same website as the master library. A main difference with the master library is that documents are stored in a folder one level deeper, in a folder with the name of the version. Thus, the structure in the archive library will be similar to what is shown below:

- Code
  - 0.1
  - 0.2
  - ...
- Documentation
  - URD
    - 0.1
    - 0.2
    - ...
  - SRD
    - 0.1
    - 0.2
    - ...

[1] `http://fingerpaint.github.io/`

– ...

The structure of the master library resembles the one of the archive library, but does not have (and does not need) the version folders. This is because the master library only contains the latest externally accepted version of every CI.

CIs may only be added to this directory after they have been externally reviewed and approved, as described in the SQAP [3] and SVVP [4].

4.2 Media Control

The libraries mentioned in section 4.1 are, as also mentioned there, stored on GitHub. GitHub is a commercial service that can be used freely as long as the repositories hosted by them are publicly accessible. They have offline encrypted backups of all repositories, that can be used in case of complete failure. Also, because of the distributed nature of Git, every project member has a local copy of every library on his/her computer.

Refer to chapter 6 for more information about GitHub and how the libraries can be accessed through it.

4.3 Change Control

In this section, we discuss who can change the contents of the various libraries.

4.3.1 Development Library

Every group member is allowed to change any CI in the development library. This means that any group member can create new files, edit existing files and delete files from the development library. There are two reasons why we allow this: first of all, the size of this project is relatively small. The entire group is working in the same room, so consultation can be done efficiently. Secondly, when a project member makes an error, this can be restored at any time because we use Git. Git also handles conflicts that may occur when two team members change the same file.

The general structure of the development library can however not be changed by group members: this is something the CM is responsible for. This is fair, as the CM is chosen by all group members.

4.3.2 Master Library

Files in the master library can only be changed by the CM, thus there is no need for version control. When a team member wants to change something to a CI in the master library, he/she has to contact the QM. The QM can then call up a review meeting, in which the proposed changes are discussed and either approved or rejected. More information on this can be found in the SVVP [4]. When changes to the CI under discussion are approved, the CM will copy the new version to the master library from the development library. The old version in the master library is then moved to the archive library. Note that in this procedure, the version number of the CI is bumped with `1.0`.

There is an exception possible to the above situation: if the only changes to a CI are non-fundamental layout changes, then the CM can simply update the document in the master library. This includes changing the font, changing the display style of the project name, et cetera. Note that these should not occur normally, as the layout is decided on at the start of the project.

4.3.3 Archive Library

Files in the archive library cannot be changed, only new files can be added, as described in the SVVP [4]. The only person allowed to do this is the CM. Thus, no change control is needed here. Note that files that are added to this library should come from the master library. The files in the master library should at that point be replaced by a newer version, that has been approved externally.

Chapter 5

Status Accounting

In this chapter, we discuss how the configuration status of CIs is documented and reported in a clear way to group members and management.

As described in chapter 4, there are three libraries in which a CI can reside. The CM is responsible for moving files from one library to another, thereby following strict rules. To report the status of all CIs, the CM will maintain a page on the website of the FINGERPAINT application. This page will list all CIs and for every CI a table indicating its status. Such a status could look as follows:

Version	*Library*	*Date into master*	*Comments*
current	Development	-	-
0.3	Master	07-May-2013	-
0.2	Archive	03-May-2013	-
0.1	Archive	25-Apr-2013	-

Note here that the top line will always be present, even when for instance in above example, the current version is version 0.3. This is because changes in the development library are not tracked, but we want to make it explicit that there is a version sitting there, for completeness.

Chapter 6

Tools, Techniques and Methods

In this chapter, the various tools are discussed that are used in the FINGERPAINT project. Also, some methods to keep the various libraries tidy are discussed.

6.1 Tools

The FINGERPAINT application is a web application that runs locally in the browser. The client has indicated it should be implemented in HTML5. At the moment, HTML5 is a sort of "umbrella definition" for a lot of technologies. In practice, what we will do is implement the application in JavaScript, making heavy use of the canvas element and everything it supports. Note that currently, the canvas element supports quite a lot of features: simple things like drawing rectangles, but also drawing curves and more complex shapes (polygons) is supported. Even creating effects like blur or inverting colours belongs to the possibilities. Finally, 3D-drawing is supported. We will not make use of the latter, but the idea is clear: a canvas is the right basis to build the FINGERPAINT application on.

To make developing the software easier, we use a couple of frameworks. This enables us to document the code more easily and more clearly (including generating documentation from the comments in the code). Additionally, it enables automated testing. This is explained below.

6.1.1 Git

All code and documentation is stored in Git repositories. Git is a distributed, lightweight version control system. It enables all team members to work efficiently in parallel on the same documents/code base, even to some extent on the same file. Apart from enabling working in parallel, it also provides an implicit backup system, as every team member has a local copy of a repository.

6.1.2 GitHub

Git is a distributed system in which a central server is needed. This is the "main" repository, to which a number of project members can connect (this is called *cloning*, every project member has a *clone* of the "main" repository on his/her machine). GitHub is a (commercial) service that can host the "main" repository for us. As long as the repository is public, the service is free to use. The client has expressed that the project can be public, so we can use

this free service from GitHub. Apart from providing "simple" hosting, GitHub also comes with a nice-looking web interface with which anyone can browse the repository. Also, there is a Wiki and bug-tracking system. Finally, GitHub provides a mechanism for hosting a website by means of creating a repository with a special name. We use this to build a website on which we can present some progress information to the client and maybe documentation or other project-related things.

6.1.3 Google Web Toolkit (GWT)

The GWT[1] is a toolkit that provides two important things for us:

- possibility to develop (complex) browser-based applications productively;
- optimised JavaScript generated by the GWT provides the user with a high-performance end product.

A great advantage of the GWT to us is that it enables developers to create web applications without requiring the developer to be an expert in browser quirks or JavaScript. A developer simply needs to know Java (which is well-known and easy to learn) to understand the code of the application. The heart of the GWT, namely, is a compiler that can convert (GWT-enabled) Java code into JavaScript. The GWT even generates different versions of the JavaScript for the five most widely used browsers[2], optimising performance and preventing browser quirks from having effect on the application.

Finally, the GWT includes a plug-in for Eclipse and an extension for Chrome to profile the application, which makes developing again more productive. When using the plug-in, it is possible to deploy the web application on a local server, keeping the developing cycle very short. If a new feature is implemented, a "production version" of the web application can be uploaded to a server.

6.1.4 Selenium

From the start of the project, we want to be able to test the software automatically. As we are developing a web application here, an important part is user experience. Of course, one of the most important facets of the user experience is the visual facet: what does it look like? However, it is very labour intensive to test this manually: that would require a tester to open the application in a number of browsers and perform the same sequence of actions in every browser. Still, probably just a subset of all available browsers will be tested. It is simply impossible to test the application in all browsers on desktop Windows, Mac and Linux machines and on mobile iOS and Android machines - which are not even *all* platforms available on the market.

Selenium[3] is a library that can aid here. It is available in multiple programming languages. We use the Java-variant. Selenium provides a means of creating and executing tests in a lot of browsers, including mobile browsers for iPhone and Android. A test in this context is literally a sequence of actions *as the user could perform them.* It is possible to click elements, type text, submit forms, drag-and-drop, simulate clicks, et cetera.

[1] The homepage of the GWT can be found at `https://developers.google.com/web-toolkit/`.
[2] Internet Explorer, Chrome, Firefox, Safari and Opera.
[3] The homepage of Selenium can be found at `http://docs.seleniumhq.org/`.

The code base of Selenium also includes a server. This server can be used to run actual tests on a remote machine, while the program that is issuing the tests runs on a local machine. Commands/actions are sent to the server through a tunnel, the server executes those actions in a browser of choice and returns the results to the local machine, when asked for. This functionality is used by us to be able to run tests from any machine on a number of different platforms.

6.1.5 Servers

The above described Selenium server will run on a virtual server provided by BCF. This server runs on Microsoft Windows 7. Apart from testing, we also run our application on a server. This will be a different server, that will be provided by the client. The application is deployed on a Jetty server (see section 6.1.6), that listens on port 80, the default HTTP port. This implies that our application can be reached directly via the browser on its IP-address or on its domain name, if that is configured correctly by the client.

6.1.6 Jetty

Jetty[4] is a web server and `javax.servlet` container that runs on Java. It supports web sockets, is open source and used to power the Google AppEngine. When using the Eclipse GWT plug-in, you can run develop mode, in which case a Jetty server is run locally to quickly deploy your code. Using Jetty from the range of servers available was a logical choice for us: if everything works when run locally on Jetty, it will work on another Jetty server as well (if the server on which Jetty runs is configured correctly of course).

6.1.7 LaTeX

All documents will be generated from LaTeX source files. LaTeX is a tool to create professionally typeset documents. We use it for a couple of reasons. First of all, the source files are plain text. This allows us to have good version control of the source. Secondly, we like to have good-looking documents, which we can create without expert knowledge about typesetting using LaTeX. Lastly, LaTeX allows us to use a single file that defines the style for all documents. This ensures a consistent layout across all documents.

6.2 Techniques and Methods

In this section, we will discuss some methods we apply to keep the Git repositories tidy and the project manageable.

6.2.1 Committing

Committing changes is something that developers do a lot, so the conventions are simple, because a developer does not want to do something complex a lot. With this in mind, we have come to the following list of recommendations:

- Make every relevant change to a repository a single commit. Do not combine multiple changes in a commit. This makes reverting changes easier.

[4]The homepage of Jetty can be found at `http://www.eclipse.org/jetty/`.

Example A commit wherein both a document and the general layout are changed, is not allowed. These should be two separate commits.

- Always write a concise yet descriptive commit message for every commit. This makes it easier to read through the commit history and find relevant commits. Note that "concise" does not mean "at most two sentences". You can definitely explain in some detail what you did. Just do not repeat the code you added, because it will be visible what you changed in your commit when looking at it in detail.
- Refrain from committing binary files. These files will change a lot (probably) with every change, which does not work well in general.

6.2.2 Tags and Branches

In Git, it is possible to *tag* a repository at any moment. A tag is simply a reference to the repository at a certain point in time, with a label. You can list all tags that are present in a repository and easily revert to the point in time the tag was made. GitHub even shows all tags in a drop down menu on the website, so browsing the repository at the time a tag was created is easy. We use this feature to tag the repository whenever a new version of a CI is created. The tag should then have the identifier of that CI as a label, if the CI has a label. For example: `URD-0.3`. The code in the repositories does not have an explicit identifier. When the code reaches a stable state, which should be at the end of each sprint, a tag with label `v[version]` should be created, for example `v0.1`. After each sprint, the version will be bumped with `0.1`. After the last sprint of the project, the version will be bumped to `v1.0`.

Since we have separate repositories, the above strategy will result in the following:

- The repository `project-code` will only contain code, no documentation and thus only tags of the form `v[version]`;
- The repository `project-docs` will only contain documentation and thus only tags of the form `[title abbreviation]-[version]`;
- The repository `sep-docs` will only contain documentation and thus only tags of the form `[title abbreviation]-[version]`.

Chapter 7

Supplier Control

The tools listed in 6.1 are all supplied by external suppliers, in some way or another. Technically, Git is an open source project and thus not really supplied by anyone, but let us call that external as well, as no project member worked on Git.

We can then make the following overview:

Tool	*Discussed in Section*	*Supplier*
Git	6.1.1	Public Domain
`http://git-scm.com/`		
GitHub	6.1.2	GitHub
`https://github.com/about`		
GWT	6.1.3	Google
`https://developers.google.com/web-toolkit/`		
Selenium	6.1.4	Various Contributors
`http://docs.seleniumhq.org/about/contributors.jsp`		
Server	6.1.5	BCF
`http://www.win.tue.nl/bcf/`		
Jetty	6.1.6	Eclipse Foundation
`http://www.eclipse.org/org/`		
LaTeX	6.1.7	Leslie Lamport/CTAN
`http://www.ctan.org/`		

The above suppliers are all trusted by us, either because we have previous experience with software from the supplier or because we have tested the software in the research phase of the project and found that the software does what we want.

In general, when we consider using software from a supplier we trust, we just use the software. When we do not know the supplier, we test the software and if the software satisfies our tests and needs, only then we will use the software.

Chapter 8

Records Collection and Retention

In the development version, files can be deleted by any group member. Of course, all members need to agree on this decision, but theoretically, anyone can delete any file. Git retains files even after they are deleted, so files can be recovered at any time if needed.

Files in the master library can only be replaced with a newer version. At the same time, the old version will move to the archive library, where files cannot be removed. So, files placed there will be retained for the entire project.