UAH Fit Vault Configuration Management Plan

CPE 656/658 Software Studio

Timothy R. Wilkins

Whit J. Sisulak

Glen L. Riden

James J. Duggan IV

11/11/2015

Revision History

Revision #	Revision Date	Description of Change	Author
0.1	10/4/15	Initial Draft	J. Duggan
0.2	10/4/15	Added SCM Maintenance	G. Riden
0.3	10/11/15	Added SCM Management information and formal release instructions	J. Duggan
0.4	10/20/15	Update Project Scope. Changed document title and filename.	J. Duggan
0.5	11/8/15	Updated scope. Added additional detail about each of the various branches of the git-flow process. Added process for data backup and recover. Detailed plan for non-configuration items.	J. Duggan
0.6	11/11/15	Updated scope. Added hyperlinks shortcuts. Updated build number information section. Added addition training information for customer. Other various edits after review.	J. Duggan
0.7	11/11/15	Updated table of contents.	J. Duggan

Table of Contents

Revision History i
1 Introduction 1
1.1 Purpose 1
1.2 Scope 1
1.3 Definitions, Acronyms, and Abbreviations 2
1.4 References 2
2 SCM Management 3
2.1 Roles 3
2.2 Responsibilities 3
3 SCM Activities 4
3.1 Configuration Items 4
3.2 Configuration Control 4
3.3 Database Backup and Recovery 9
4 SCM Schedules 10
5 SCM Resources 11
5.1 Tools 11
5.2 Training 11
6 SCM Plan Maintenance 12

Software Configuration Plan

1 Introduction

1.1 Purpose

The purpose of this document is to define the software configuration plan for the UAH Fit Vault software projects. This document should be used as a process artifact that defines how to manage project documentation and the system's software. The intended audience for this document includes system developers, testers, customers, and any other stakeholders.

1.2 Scope

1.2.1 Software Overview

The UAH Fit Vault software package will be a web application that will accept medical data from users and display the data in a meaningful way. There are two major components to this software. The first is the data collection tool that is used by the users to upload their medical data that is recorded by one of the supported wearable medical devices. There are three different medical devices supported for this project that record various types of data. The data provided by these devices consists of different file formats, and the data is different from device to device. The software will have to determine the contents of each file and how to process them. The software needs to able to take in files that a user has downloaded from their medical devices, process those files, and store the data in a database. The software should have the ability to process multiple files at a time as well as individual files and allow for an activity to be assigned to them by date and time.

The other major component of the web application is the data analysis tools used to analyze the data that is captured from the data collection tool mentioned above. The software needs to perform data analysis over different intervals of time such as one week, one month, etc. There will need to be some way to manage user access to the various medical data that has been inserted into the database that this software will access. Below are some proposed data analysis ideas that can be incorporated into the project.

- Simple Moving Average
- Data correlation discovery between the multiple devices.
- Simply display data that was uploaded to the customer in a graphical format.
- Calculate the user's activity.

The data analysis possibilities will likely not fully be realized until the project team understands the different types of data that are available. Also, there will need to be collaboration with the customer for additions or changes to the data measurements provided by this software. The web application will have to have different levels of user access which will be defined later in this document.

1.2.2 Software Configuration Management Plan Overview

The Software Configuration Management Plan (SCM) will consist of the following items.

- Documentation Management (*3.2.1.1*, *3.2.2.1*, *3.2.3.1*)
- Software Management (*3.2.1.2*, *3.2.2.2*, *3.2.3.2*)
- Release Process (*3.2*)
- Configuration Management Role Responsibilities (*2*)
- Data Backup and Recovery (*3.3*)

Documentation management will describe how the document artifacts will be controlled, stored, and distributed to the various stakeholders involved in this project. Software management will describe how and where the software for this project will be stored and accessed for releases. It will also describe a branching strategy that will be put in place during development. The release process will explain the different types of releases that can occur. There will be two types of releases: formal and informal. The configuration management roles will also be defined in this document.

1.3 Definitions, Acronyms, and Abbreviations

SCM	Software Configuration Management Plan
SRS	Software Requirements Specifications
SDD	Software Design Document
SDP	Software Development Plan
STP	Software Test Plan
ROM	Rough Order Estimate
Git/GitHub	Source control used by the development team.
Repository	Another term used to indicate where source code is stored.
Branch	Can be viewed as the different file directories in a repository that hold different versions of the software.
Merging	The act of combining repository branches into one branch.
Artifact	Any item that has been generated for use during the development process that is not a configuration item.

1.4 References

IEEE Std 828-1998, IEEE Standard for Software Configuration Management Plans

Git-Flow Branching Model (http://nvie.com/posts/a-successful-git-branching-model/)

Atlassian Branching Tutorials (https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

2 SCM Management

2.1 Roles

- Team Member
  The team member role describes all the members of the team that creates or edits documents or source code that require configuration management.

- Release Manager
  The release manager is in charge of performing source code merges in preparation of a new release.

- Test Manager
  The test manager is responsible for ensuring that the software has completed testing prior to releasing.

2.2 Responsibilities

The configuration management process is the responsibility of the entire team. All team members regardless of role are required to follow the configuration procedures laid out in this document. Team members are responsible for ensuring that the documents and source code that they create are properly uploaded to the current repository. They are also responsible for communicating to the rest of the team when new documents are available for review or new source code is ready to test and merge back into a stable branch.

The release manager is responsible for reviewing all changes to the software prior to merging back up into a stable branch in the repository. This is to ensure that that the merge does not introduce conflicts and to determine the likelihood of the changes introducing new bugs into the stable branches. The release manager will be responsible for finalizing all software releases and ensuring that the correct version of the software is pulled for new releases. The release manager will be responsible for pulling software and delivering it to the customer upon release. The release manager will ensure that all pertinent documentation for setting up and installing the software is given to the customer during a software release.

The test manager is responsible for ensuring the software satisfies the customer's requirements prior to the software release. The test manager is also responsible for giving final signoff of any test procedures that are required to run against the software prior to release. The test manager is responsible for the over test plan for the software. The release manager will not be permitted to release software to a customer without permission from the test manager.

3 SCM Activities

3.1 Configuration Items

- SRS – Microsoft Word document
- SDD – Microsoft Word document
- SDP – Microsoft Word document
- STP – Microsoft Word document
- SCM – Microsoft Word document
- ROM - Microsoft Word document
- Data Collection Software Source Code – Source code files that include the following file types: .cs, .sln, and .csprog.
- Data Analysis Web Application Source Code – Source code files that include the following file types: .cs, .cshtml, .js, .config, .sln, and .csproj.

3.2 Configuration Control

The configuration items listed in section 3.1 will be controlled using both the UAH Canvas web application and GitHub. There will be two types of releases that each can apply to each configuration item. The first is an informal release. An informal release occurs every week on during which document updates are submitted to the various stakeholders on the project. The second type of release is a formal release. This release will occur at the culmination of the project period when all the documents and source code are turned over to the customer.

3.2.1 Informal Release Process

3.2.1.1 Documentation

Each member of the team will be responsible for uploading any documents that they have generated or modified to either Canvas or GitHub. Canvas discussions are used to notify the other members of the team of new or updated documents and where they can be found. The documents are then reviewed by the members of the team and any comments or changes are provided. If there are no required changes, then the changes are merged into the document by the team member that has authored that document. In the case that the document author is not available the team member that made the change is responsible for modifying the document with the new changes. At this point the document should be updated in the GitHub master branch. This is where the informal releases will be pulled for submission to a stakeholder. Each document shall have a revision history section in order to keep track of changes made to each document. The revision history table shall contain a version number, a summary of the changes made to the document, and the author of the changes.

3.2.1.2 Source Code

The configuration control plan for source code will be modeled after Vincent Driessen's branching model which is also known as git-flow. The process involves maintaining two branches master and develop. The master branch will contain the different software releases that occur during the project lifetime. The develop branch is the main development branch that contains all the new features and/or improvements that will go into the next release. When a new feature or an improvement needs to be implemented a new feature branch will be created. Upon the completion of the feature or improvement the branch will be merged into the develop branch and then deleted. Before making a release the develop branch will be merged into a new release branch that is meant for final testing before the release. Once testing is completed any and all changes made in the release branch are merged back into both the master and develop branches and a new tag is created on the master branch that will be represented by the current version number of the software. See the references section for links to more information about the git-flow process.

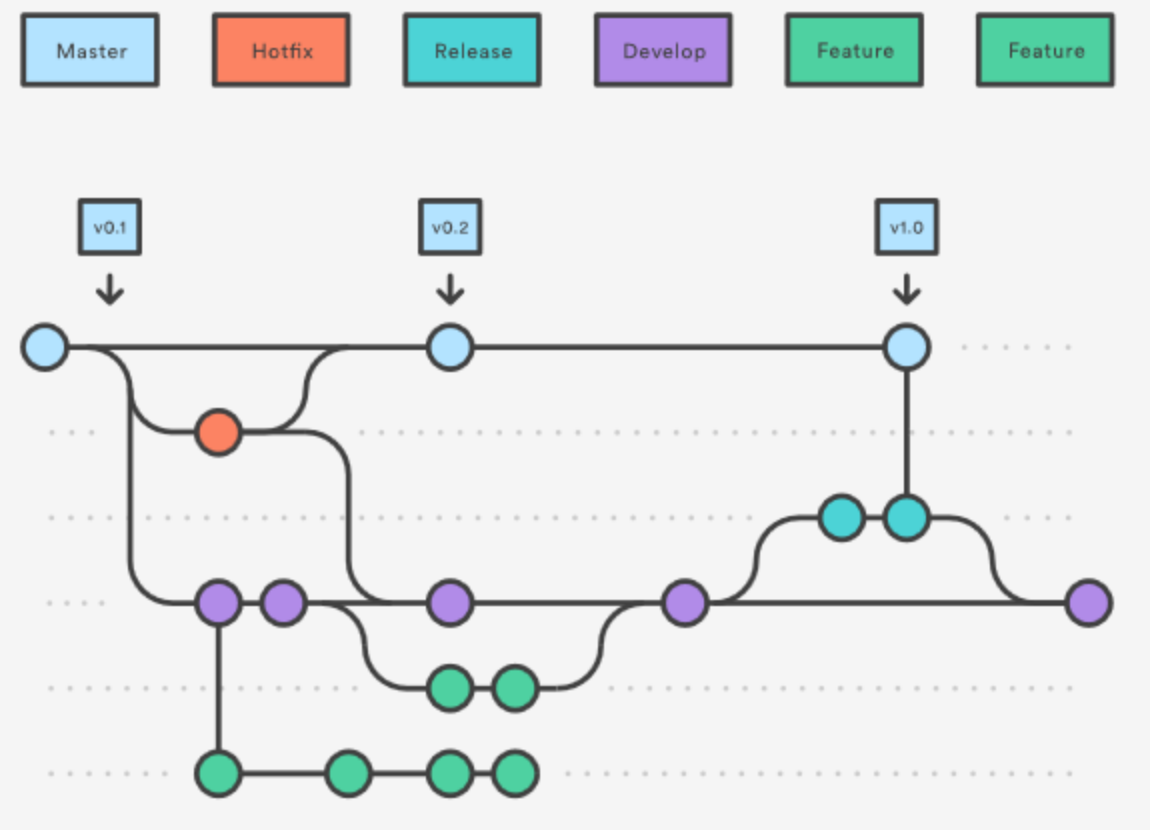

Figure 1 – Git-Flow Atlassian Branching Tutorial

Below each of the branches discussed in this section will be described in further detail. These sections should aide in a better understanding for the git-flow process.

3.2.1.2.1 Master Branch

The master branch contains the version of the software that can be released to a customer. The master branch will only be merged into after testing has been completed successfully on a current test branch that is pending release. The master branch is created at the beginning of the project and exists for the duration of the project. The process of merging into the master branch is synonymous with the process of releasing software the customer. The process is detailed below.

1. Merge the current release (master branch) into the develop branch.
2. From the develop branch create any number of feature branches in order to satisfy the requirements of the software release.
3. Create a release branch from the develop branch once development is completed.
4. The complete software test plan shall be executed on this release branch.

5. Any defects found shall be fixed and committed to this branch.
6. Once the software test plan has been executed successfully and no defects have been found, the release branch is merged to the master branch. At the time of the merge a version number tag shall be assigned to the master branch illustrating the version number of the release.

3.2.1.2.2 Develop Branch

The develop branch contains all of the features and improvements made to the software during a release cycle. The develop branch is a stable branch that results from merging all of the changes. The develop branch is meant to be a staging area for the new features and improvements that are ready for testing for the pending release.

3.2.1.2.3 Release Branch

The release branch is used for testing prior to release. During testing any defects that are found are fixed and committed to this branch. After testing is completed successfully the release is finalized by merging the release branch into the master branch and tagging the master branch with a new version number. Full details of the process are listed in the master branch section 3.2.1.2.1.

3.2.1.2.4 Feature Branch

Features branches are created when a new feature is being designed for the system. The purpose of the feature branch is to encapsulate the feature allowing developers an easier time to develop a feature without disturbing the master branch. The process of creating a feature branch is as follows.

1. A request for a new feature has been made.
2. The Release Manager creates a branch off of the current develop branch for any developers to use to create the new feature.
3. Developers commit changes to the branch as they develop the feature.
4. Upon completion of the feature, the Release Manager and/or another developer that did not develop the feature will pull the current feature branch for review. (Note: This allows for review of just this feature and does not confuse the reviewer with other changes that have been made to the system. This is one of the main advantages to feature branching).
5. After any changes are made and committed to the feature branch, the Release Manager will merge the feature branch back into the develop branch.
6. The Release Manager will then delete the feature branch.

3.2.1.2.5 Hotfix Branch

Hotfix branching is only used in the event that a critical defect is found in the released software that cannot wait to be fixed until the next release. The steps for completing a hotfix are described below.

1. Customer reports a defect in the software and requests it fixed.
2. The Release Manager creates a new hotfix branch from the current master branch.
3. A developer troubleshoots and fixes the defect in the hotfix branch and commits their changes.
4. The test team executes an updated test plan to verify the defect has been resolved and no other defects have been introduced into the system.
5. The Release Manager merges the changes back into the master branch, tags the software with a new version number, and releases the hotfix to the customer.

3.2.2 Formal Release Process

3.2.2.1 Documentation

During a formal release all of the documentation for this project will be collected from the GitHub repository master branch. The files will then be written to any media of the customer's choice. If the customer does not provide any media for delivery the documentation will be written to a CD-ROM and delivered to the customer.

3.2.2.2 Source Code

All source code released to a customer during a formal release must come from the master branch from the project's GitHub. The source code will be provided with all other project documentation. Before providing the software source code to the customer the test manager will run through the project test plan to verify the software is correct.

3.2.3 Version Numbering

3.2.3.1 Documentation

Version numbers will constructed from a major and minor number. These will be combined with periods between each in the following fashion:

X.X (major.minor)

Major Number: Major number is incremented every time the document is considered to be a final project.

Minor Number: Minor number is incremented every time an edit is made to the document leading up to the final project.

3.2.3.2 Source Code

Version numbers will constructed from a system, major, minor, hotfix, and build number. These will be combined with periods between each in the following fashion:

X.X.X.X.X (system.major.minor.hotfix.build)

System Number: System number will rarely change as this would mean that the entire software product had been recreated or the software changed so radically that it could be considered a new product. This number will only increment after a formal release to the customer.

Major Number: Major number is incremented for each release completed in the git-flow process. This will occur when the all the new features and improvements in the develop branch are merged into the master branch.

Minor Number: Minor number is incremented every time a feature branch is merged into the develop branch.

Hotfix Number: Hotfix number is incremented only if a critical bug is identified and a fix needs to be sent to a user prior to the next release and will include the fix for this bug.

Build Number: Build number is used to determine what build version of the software is being used. Mostly, it is used to see what build is on a test server during the testing and development stages. During testing if changes are made to fix defects, there needs to be an update to the software that is being tested against. In order to keep track of which version of software is being tested against the build number is incremented to differentiate between the versions of software in the release branch that are being deployed to the test environment.

3.2.4 Non Configuration Items

There may be several other artifacts that are generated during the design, development, and testing process. These items are not strictly identified in this document. Testing artifacts that are generated during various times at this process to outline testing strategies or guidelines for the various unit tests, integration tests, or system tests will be saved in GitHub. These documents will not take part in any document configuration process detailed in this document. These documents will not be delivered to the customer during a formal release. However if the customer requests additional testing information, we will provide these artifacts to the customer.

3.3 Database Backup and Recovery

The system requires the use of a database to store all of the patient medical data that is uploaded into the system. There will be no automated backup utility provided with the configuration items. This section will outline a recommendation for the system administrators for a database backup and recovery process. The database and web application will be run on a Windows Server virtual machine. It is recommended that once a week or at any other period determined by the customer that an image of the virtual environment be created and stored on another external media from the computer system. In the event that the machine becomes corrupted or no longer usable, and a new machine is required, the environment can be recreated by loading the image backup that was created.

There is one major risk to data recovery. If the customer's system administrators do not have a regular backup plan and process, then they risk losing all new patient data uploaded since the last backup image was created. It is recommended that a backup be performed at least once a week when the system is active and patients are actively uploading data into the system.

During development the backup and recovery recommendations listed above will be the process used during development. The one change made to this process is that the development team will only backup on a monthly basis not weekly.

4 SCM Schedules

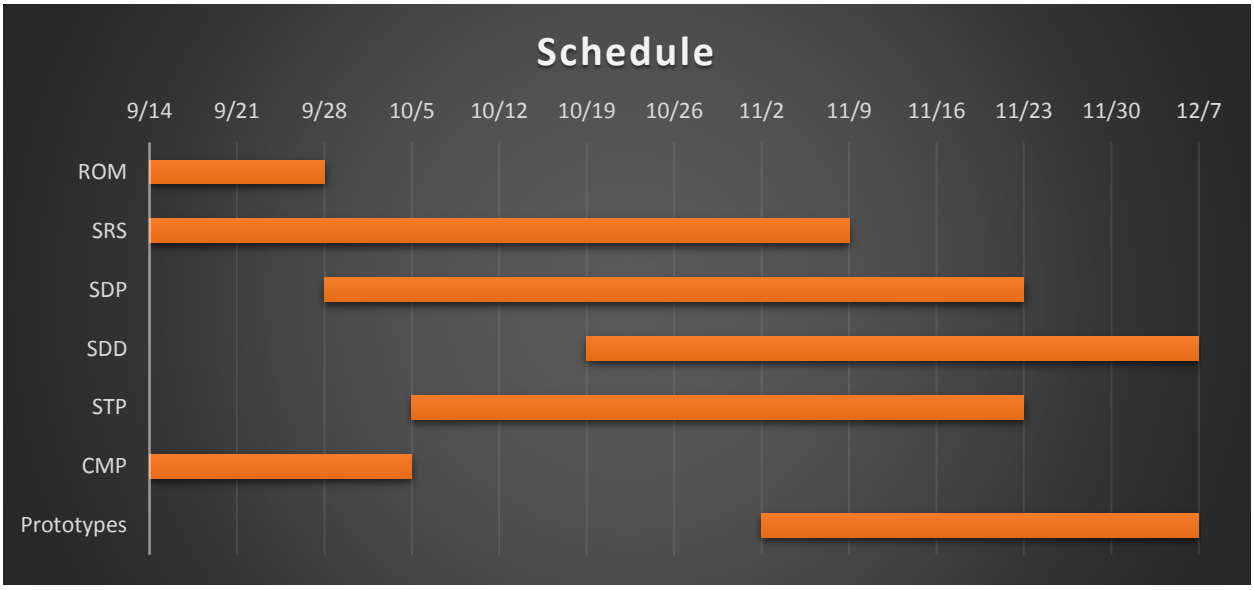

Figure 2: SCM Schedule

The preceding diagram illustrates the tentative schedule for this project. The overall goal is to complete the initial version 1.0 by the dates above. During the identified schedule times each document may go through several minor or major revisions before being finalized. After December 7th all documentation will be considered frozen for the duration of the project term and not changed until after a formal release has occurred and new changes have been proposed by the customer. The dates above are not firm dates but have been set to help establish a base line for configuration item initial releases. The only scheduling requirement that must be met is that all configuration items be released by December 7, 2015.

The schedule does not include the following details. The SDD is blocked by the completion of the SRS. This means that the SDD cannot be completed without a complete SRS. Work on the SDD document however can begin before the SRS is completed The prototype is meant to be two working pieces of software with limited functionality based on the requirements laid out in the SRS and the design completed in the SDD. Upon completing the prototypes, and initial beta version of the software project shall be available.

The planned formal release does not have an exact date as this point but is planned tentatively for the end of April 2016. At this time the project will be considered completed and all documentation and source code will go through a formal release process and be delivered to the customer.

5 SCM Resources

5.1 Tools

- Visual Studio 2015
- Git Plugin for Visual Studio (latest version)
- Git Desktop Application (latest version)
- UAH Canvas (active version)
- Microsoft Word 2010+

5.2 Training

5.2.1 Developers

There may be training required for member of the team that have not used any of the tools mentioned in the previous section. In the event that training is required a group meeting will be held to provide cross training to any member of the team that needs training in one of the areas listed above. There may also be online tutorials that may be referenced for additional training.

Training will either be provided face-to-face or through an online video conference tool. Screen sharing will be required to adequately facilitate the training exercise. The trainer will also have to have a working microphone allowing for verbal communication in order for the training to be more efficient.

Upon completion of a new software release any new features or major improvements will require cross training. The team member that developed the feature or improvement shall provide training to the other members of the team prior to release of the feature or improvement.

5.2.2 Customer

Customer training will be provided after a formal release. Training will be provided on setting up and installing the system on the virtual machine provided. The development team will provide training to the customer on the usage of the system if the customer requests. In the event that after a formal release the customer requires training on the tools used to develop or test the software, the development team will provide email consultation or references to assist in training.

6 SCM Plan Maintenance

Configuration plan maintenance will be performed/reviewed at the start of each project phase (requirements, design, implementation and testing). If the CM plan is changed at the start of a project phase, the plan will be distributed to the team. The updated CM plan will include a history of the changes made, who is responsible for monitoring the plan, how changes to the plan will be approved, and how the changes will be made and distributed.