Team 4
Flight Data Analysis Tool
Software Configuration Management Plan

Version 2

12/3/2015

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

Initial Release:	9/2/2015
Current Release:	12/3/2015 – Version 2
Indicator of Last Page in Document:	Pg. 12, “End of Document”
Date of Last Review:	12/3/15
Date of Next Review:	TBD
Target Date for Next Update:	TBD

Distribution List

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Guidance Team Members:	Dr. Salamah Salamah
	Elsa Tai
Team Members:	John Delgado
	Matthew Melendez
	Angel Ortega
	Jesse Tellez
	Michael Rivera

Change Summary

The following table details changes made between versions of this document

Version	Date	Modifier	Description
0.1	8/31/2015	Angel Ortega, Michael Rivera	Initial Release
0.2	9/2/2015	Angel Ortega	Started filling in the blanks (Introduction)
1	9/2/2015	Comcorp	Collaborating and filling out each section
2	12/3/2015	Comcorp	Made corrections. Version 2 Release

Table of Contents

1. Introduction

1.1. Project Overview

The Flight Data Analysis Tool (FDAT) is a browser-based web application that retrieves compressed air traffic data and uses an interface for data analysis, meaningful report creation, and visualization tool usage to assist research of flights and interacting factors. The system will be hosted in a remote server. The system will parse and store flight related messages into a database. The user will be able to query and retrieve data in manners that will be useful for the end-users at the Next Generation Applied Research (NEAR) Lab. In addition, the system will provide security features to avoid disclosing classified data to filtered users.

1.2. Purpose and Intended Audience

The purpose of this document is to establish and maintain the integrity of the software project throughout its life cycle. This document is intended to be used as a protocol for change and version control management. The document will help to control and define software changes in the Flight Data Analysis Tool, and to make sure that those changes are correctly implemented. This document also ensures that every party involved with the project is notified and is aware of the reason of the change. This document will also be able to identify the severity of changes and define a standard for a systematic description of the project. By providing a method of change processing, this document will allow the development team to backtrack changes in case of any mistake in the development process or unnecessary changes applied to patches. This document will ultimately define an inspection guideline to determine the necessity of a change and the Software Configuration Items being affected by the submitted changes.

The document outlines three major sections: Software Configuration Identification, Software Configuration Control, and Software Configuration Auditing. The software configuration identification section includes the naming convention for items, directory structure, backup plan, and processes to manage the items. The software configuration control section includes documentation and process to control changes, configuration control board, approval/disapproval process, configuration management, baseline creation, and check-in/check-out procedures. The software configuration auditing section includes the process, responsible parties, and related documentation.

1.3. References

[1] T. Preston-Werner. (2014, June 18). Semantic Versioning 2.0.0 (2.0.0) [Online]. Available: http://www.semver.org/.

2. Software Configuration Identification

This section provides labels for the baselines and their updates to understand the status of these items as they progress through the development process. The purpose of this section is to define the items that will most likely require revision in case that a change is submitted to the project. This Section also defines the naming convention of future releases and the effect on that naming convention by approved changes. This information includes the changes that will undergo a process of testing, analysis, and approval for implementation and introduction into the project.

2.1. Software Configuration Item Identification

A software configuration item is an artifact or collection of artifacts that must be protected from unnecessary change by the development team to ensure that the project succeeds on schedule and with the highest quality possible. These artifacts must be processed via an approval process in order for changes to be made to a configuration item.

The most obvious configuration item is the code base of the application itself. To protect the validity of the publishable code, two code repositories will be set up using git. Code merged with the production repository must follow the change request process as defined in this document for code.

Along with the code base, the unit tests and other associated code tests will also be considered configuration items.

Physical documents that are shared between two or more members of the development team must follow the change request process as defined in this document for documents. These documents include: the software configuration management plan, software requirements specification, user guides, project standards document, coding standards document, project plans, and all deliverables that will be submitted to the guidance team.

2.2. Item Naming Convention

2.2.1. Code Versioning Scheme

The development of code shall follow the semantic versioning approach based on the standard defined in [1].

2.2.2. Document Versioning Scheme

Documents shall begin with the initial version number of 0.1, and the minor version number will be incremented with every change. Once submitted to the guidance team, the major version number will increase by 1, and the minor version number will reflect the number of edits since the submission version.

2.3. Software Configuration Item Organization

The two code repositories will be structured as so: the baseline repository will be called the *production* repository, while the second repository will be called the *development* repository. The development repository is where the development team will collaborate to develop and test the system. In order for code to be transferred to the production repository, the code must pass all unit tests that are to be carried out by the development team. After passing all tests and is reviewed by the team, the change request may be submitted and processed by the assigned configuration manager. Version numbers will be incremented according to the semantic versioning 2.0 [1] standard.

Document and diagram artifact baselines will be stored in a baseline folder in smart cloud. These files are to be kept separate from the working files.

2.4. Directory Structure

Code that is saved in the code repository will contain at least the following high-level folders: models, views, and controllers. The models folder will contain all of the data representations of the project and will contain subfolders to describe the subsystems/modules of the system as needed. Data connections will also be stored in the models folder under the “data” subdirectory.

The views folder will contain all of the files needed to handle the user interface. Views will also be placed in their respective subdirectories based on the subsystem or module that they support.

Controllers will be placed in the controllers folder, where the interactions between the users will be mapped to a specific view and a set of models.

Documents will be stored within the IBM Smart Cloud system that was provided for the class to use. Under the Team 4 group, each document will have its own folder where the different revisions and baseline of the documents can be accessed.

Change request forms will have their own folder in smart cloud, with three more subdirectories inside where change requests can be sorted based on their status: pending, approved, and closed.

2.5. Backup Plan

The code repository will be hosted on a private GitHub repository. There will be a main integration branch, separate from development branches. Each developer has their own copy of the code for development, but all developers can also access the integration branch. Documents will also be backed up in a DOCS/ directory in the code repository in a similar manner, so that the documents are also backed up along with the code in a distributed environment.

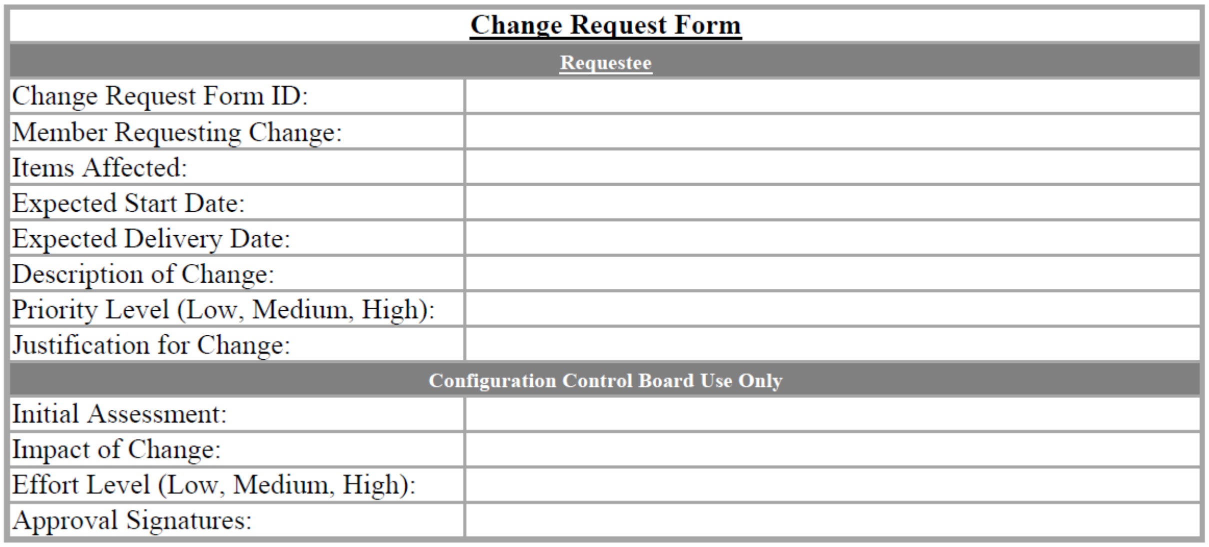
GitHub is a reliable service and the development team does not expect any serious downtime from the hosting service. However, if GitHub is to go down, an Amazon Free Tier server can be set up to act as a temporary repository, and act as the new remote repository until GitHub resumes regular service.

3. Software Configuration Control

The purpose of the Software Configuration Control is to provide a detailed mechanism for preparing, evaluating, and approving or disapproving all change proposals to the configuration items throughout the life cycle. The purpose of this section is to identify what mechanisms will be used to control access to items in the configuration in order to prevent unauthorized updates and collisions between team members working on the system simultaneously.

3.1. Documentation

Changes to configuration of sensitive items will require approval by the Configuration Control Board. The top of the Change Request Form, Figure 1, will be filled out by member requesting change before presenting the change for approval to the Board. The Board will then complete the Change Request Form. The Board will also need to make a decision on whether or not the specific change is approved or not.



Change Request Form	
Requestee	
Change Request Form ID:	
Member Requesting Change:	
Items Affected:	
Expected Start Date:	
Expected Delivery Date:	
Description of Change:	
Priority Level (Low, Medium, High):	
Justification for Change:	
Configuration Control Board Use Only	
Initial Assessment:	
Impact of Change:	
Effort Level (Low, Medium, High):	
Approval Signatures:	

Figure 1: Change Request Form

3.2. Configuration Control Board

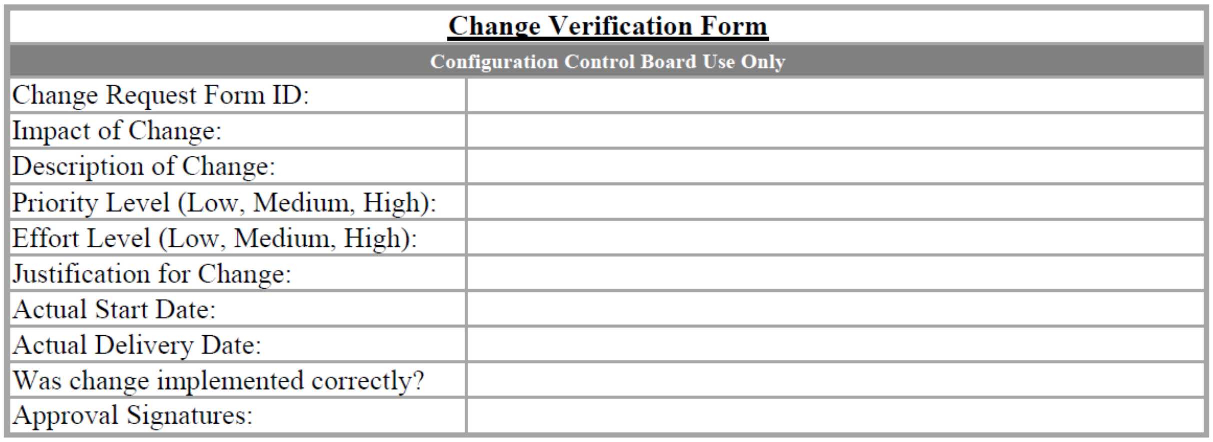
The Configuration Control Board will consist of each of the team members. Any changes being made to the SCIs will be requested through the Change Request Form, and reviewed by at least 3 members of the Board. The lead for the item will be the first member of the Board, unless he is the one requesting the change. The remaining members will be chosen from the development team. No member of the Board should be requesting the change. The Change Request Form will be placed in a folder on IBM SmartCloud marked "Pending Requests". The change must be approved by majority vote by these three members, which will consist of the lead member for the item that is to be changed and two other members. The two other members will be decided ahead of time, when the SCI is initially created. If the one proposing the change is the lead member for that item, then three other members will decide whether to grant the change request. If a major change is being proposed, all five members will vote on whether the change be implemented. If the change is approved, it will be marked as "Approved".

Changes will be judged based on a number of criteria. Some examples include, but are not limited to:

- Validity (change makes sense)
- Ease with which it can be implemented

- Impact it has on other modules/documents

Once the change is approved, it will then be carried out by the member who is most familiar with the area of code/documentation. Once the change is made, the Board will meet again and review the change to ensure it has been implemented correctly. At this point, the Change Verification Form, Figure 2, will be filled out. If the change was not implemented correctly, the member in charge of it must resubmit the change request. After the Board has deemed that the change was implemented correctly, it will be integrated into the working repository. The version number of the SCI will be updated to reflect the change.



Change Verification Form

Configuration Control Board Use Only

Change Request Form ID:	
Impact of Change:	
Description of Change:	
Priority Level (Low, Medium, High):	
Effort Level (Low, Medium, High):	
Justification for Change:	
Actual Start Date:	
Actual Delivery Date:	
Was change implemented correctly?	
Approval Signatures:	

Figure 2: Change Verification Form

If an error or defect is found by V & V, a special Error Report Form may be created and sent to the members of the Board. The lead member for that artifact as well as the one who wrote it is responsible for responding to the form and fixing it. The fix will not need to be formally requested.

3.3. Configuration Manager

Each artifact will have an assigned configuration manager, or document lead, who is charged with the task of monitoring their assigned configuration items. The default configuration manager for an artifact will be the assignment lead for the artifact. If no lead is assigned or the lead petitions for a different member to take on the responsibility due to a pre-existing heavy workload, a new document lead will be assigned by majority vote of the development team.

3.4. Procedures

In the process of a change being submitted by a stakeholder, the team will undergo a series of steps to determine the approval or denial of a proposed change.

1. First, the form should be submitted to the document manager, who in turn will file this form in the SmartCloud folder for Pending Changes
2. After the team is aware that a change is pending, the group of three people excluding the proposer will gather and discuss the reasoning of the proposer and add the insight as to the advantages and disadvantages of the added change. This also includes the necessity of the change.
3. After the information if spun and talked about, the team will take a vote and majority will dictate if a change goes through or not.
4. If the change goes through, the development team lead will assign the change to the person in charge of the item in question and make a change in the development environment for testing purposes. If the change does not go through, the form gets filed under the denied requests folder in SmartCloud.

5. Once the implementation is added, the versioning of the project will change accordingly depending on the features removed or changed.

If one of the changes that were proposed and denied, the Board must then provide a ‘concrete’ reasoning for it, such as an obstacle to move on with the project. These steps are independent of a timing convention, so it is not appropriate to include time limits. Reasoning regarding the final decision MUST go in the request form before being archived. This will eliminate unfair decisions and allow for further investigation regarding denied features or new features.

3.4.1. Baseline Creation

The objective of establishing a baseline is to define a basis for further system life cycle process activity and allow reference to, control of, and traceability among configuration items and to requirements. It serves as the common reference that all system development activity is built on and dictates to the development team the changes that are to be implemented.

- A baseline must be presented for every single Software configuration item described in the sections above. The baseline for the development area of the project will define a guideline for the development team which will aid in the lifecycle of the development process of the software.
- The regular baseline will directly be presented through the production repository and will include every working feature in the project and will be presented formally as another item in this document.

In addition to the functionality of the software, the baseline will include every document that is linked to the functionality to enhance the tracing capability.

3.4.2. Check-In/Check-Out Procedures

For team working procedures, the team will be utilizing GitHub for development processes involving collaborative efforts. Github provides the possibility of uploading module by module and document every version change with logs. The steps for editing or creating a module are the following:

1. Every user will have access to a Production and a Development environment within GitHub where code can be uploaded.
2. If a user were to edit a file, the commit message should read the purpose or the change that was achieved in the edit. Committing without merging first could result in files being lost, so it is very necessary to merge then commit to the repository. These actions are mostly in the development environment since production environment requires something else mentioned later.
3. Every user will have a copy of the code in their IDE. However, every time that code is accessed an update must be called to retrieve any possible changes that could be overwritten. This will allow an ease of access for every other member of the team.
4. After the update is called, the user is welcome to edit as they please, or create modules.
5. When committing, if a merging error occurs, close look at the changes is required to avoid buggy or non-working code. This will make the software more usable and everybody will know that the change produced many changes.
6. With that said, any future edits will follow the first steps in order to avoid any confusion and any mistake when uploading.

4. Software Configuration Auditing

4.1. Process

To ensure that the current configuration of the software system mirrors the software system pictured in the baseline and the requirements documentation we shall review the requirements for the software. Creating a matrix that will allow our team to create a checklist of the requirements that will need to be implemented and ensure that everything that is in the software serves a purpose and is needed. This matrix can also be updated as needed to ensure that the software being developed is up to par with the current standards of the customer. We shall also conduct unit testing on the source and executable code modules. We shall review the design of the software and conduct acceptance testing to see that the integration of the software is complete. The last part of the process is going to see the deployment of the software is completed and successful. After ensuring that the deployment is successful the client will be provided the complete documentation of the software including a user guide, a signed report that shows exactly how each requirement was met, and the process for how maintenance will be handled and expansion of the software if needed.

4.2. Responsible Parties

The responsible parties for conducting this audit will be Comcorp Systems internal Verification and Validation department and then approved by the University of Texas at El Paso department of Computer Science guidance team for software engineering.

4.3. Related Documentation

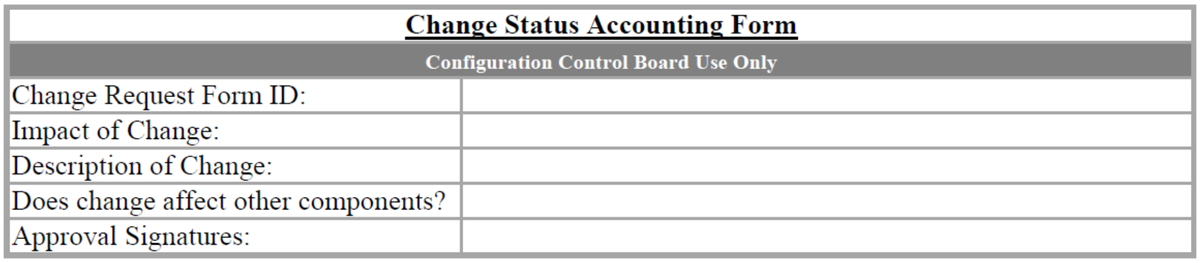
The development team created a checklist to serve as a mechanism to determine to which degree the current configuration of the software system mirrors the software system pictured in the baseline and the requirements documentation.

The checklist helps determine ideological and actual status of the software system configuration. The checklist will help determine conformity between design and implementation.

5. Software Configuration Status Accounting

5.1. Process

To ensure changes did not affect other components of the software configuration, the development team developed the Change Status Accounting Form. This form list the impact of change, a description of this change, external components affected, and approval signatures. The form can be seen in Figure 3.



Change Status Accounting Form

Configuration Control Board Use Only	
Change Request Form ID:	
Impact of Change:	
Description of Change:	
Does change affect other components?	
Approval Signatures:	

Figure 3: Change Status Accounting Form

6. End of Document

End of Document