Cyber-Energy Operation Management System
Software Configuration Management Plan
Version <1.5.1>
2/5/2013

Document Control

Approval

The Guidance Team, Dr. Irbis Gallegos, Dr.Yoonsik Cheon, Aditi Barua and the customer, Dr. Ralph Martinez shall approve this document.

Document Change Control

Initial Release:	1.0
Current Release:	1.5.1
Indicator of Last Page in Document:	@
Date of Last Review:	02/04/2013
Date of Next Review:	To Be Determined
Target Date for Next Update:	To Be Determined

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. Irbis Gallegos
Dr. Yoonsik Cheon
Aditi Barua

Customer:
Dr. Ralph Martinez

Software Team Members:
Gabriel Arellano
Chris Duran
Crystal Lopez
John McKallip
Ramon Vega
Matthew Wojciechowski

Change Summary

The following table details changes made between versions of this document

Version	Date	Modifier	Description
1.0	02/01/2013	Crystal Lopez	Initial Document
1.1	02/01/2013	Crystal Lopez	Introduction
1.2	02/03/2013	Gabriel Arellano/Chris Duran	Software Configuration Identification
1.3	02/04/2013	John McKallip/Ramon Vega/ Matthew Wojciechowski	Software Configuration Control
1.4	02/04/2013	Chris Duran	Software Configuration Auditing
1.5	02/04/2013	John McKallip/ Matthew Wojciechowski	Appendix
1.5.1	02/05/2013	John McKallip	Proof read and corrected

TABLE OF CONTENTS

1. Introduction

The Cyber-Energy Operation Management System (C-EOMS) will be used to gather information from several different energy-management sites situated around the UTEP campus. This document has been put together to describe the set of activities that have been developed to manage change throughout the software life cycle of the C-EOM system. These activities systematically control changes to the configuration and maintain the integrity and traceability of the configuration throughout the software system's life cycle

1.1. Purpose and Intended Audience

The primary focus of this document is to describe methods, which identify configuration items, control change requests, and document the implementation of these requests. The objective of the Configuration Management Plan is to limit the impact changes may have on the system. This will enable the development team to remove any unnecessary changes, and to monitor and control any necessary changes. By doing so, it will allow the development of software to continue, despite any large or unimportant changes without substantial tracing and decreasing the amount of development time. The target audiences for this document are the system developers, the client and any other person(s) involved in the development of the C-EOM system.

1.2. Overview

The CMP is divided into the following sections: Software Configuration Identification, Software Configuration Control and Software Configuration Audit. The Software Configuration Identification section discusses the process of dividing the system into distinctive parts, along with providing labels for baselines and their updates. Furthermore this section will provide items that are to be controlled as well as a labeling scheme to be used in managing the controlled items. The Software Configuration Control (SCC) section concentrates on managing changes during the software's lifecycle. There are three components that make up the SCC section, documentation, formally defines a proposed change to the software system, implementing a Configuration Control Board (CCB) which evaluates by approving or disapproving a proposed change to the software system, and procedures which describes the team guidelines or procedures used to manage any configuration items. The Software Configuration Audit section provides a mechanism for determining the degree to which the current configuration of the software system reflects the software system visualized in the baseline and the requirements documentation. To conclude this document an Appendix is provided containing a screen view of the change request form along with a brief tutorial on the GIT software.

1.3 References

[1] Vliet, Hans Van. *Software Engineering: Principles and Practice*. 3rd ed. Chichester, England: John Wiley & Sons, 2008. Print.

[2]"About Versions of the Unicode Standard." About Versions. N.p., n.d. Web. 19 Sept. 2012. http://www.unicode.org/versions.

[3] Cheon, Yoonsik & Gallegos, Irbis. *Software Configuration Management (SCM).* [Microsoft Power Point] EL Paso : s.n., 2013.

[4] "IBM Informix 11.70 Information Center." *IBM Informix 11.70 Information Center*. N.p., n.d. Web. 05 Feb. 2013. http://publib.boulder.ibm.com/infocenter/idshelp/v117/index.jsp?topic=/com.ibm.bar.doc/ids_bar_510.htm

[5] "GIT Tutorial: Commands." N.p., n.d. Web. http://www.siteground.com/tutorials/git/commands.htm

[6] "Git Reference." *Git Reference*. N.p., n.d. Web. 05 Feb. 2013. http://gitref.org/

2. Software Configuration Identification

2.1. Software Configuration Item Identification

During the software engineering process, many items - such as documentation or code - are expected to be produced and modified by the software development team. The following list is all such components, which the team shall create and maintain:

- Source code
- Software Design Document (SDD)
- System Requirements Specification (SRS)
- Meeting reports
- Test Suites
- Graphical User Interface (GUI)
- Requirements Memorandums
- Test Plan

Changing the preceding documents or data shall employ the process detailed in section 3 of this document. Meeting records and requirements memorandums shall not change, and will be kept for reference only.

2.2. Software Configuration Item Organization

We will use the same numbering system to label our documents (e.g. the SRS, the SDD) and software (e.g. source code, test suites). The numbering system will consist of the following naming convention, that is derived from the Unicode standard for versions:
major.minor[.update].[2]

Since the numbering system is the same for both types of configuration items - documents and software - each element from the above convention shall be described separately to clarify how it applies for each set of items.

	Documentation	**Software**
Major	Major numbers change when the paper has been reviewed by a member of the guidance team and their critiques have been rectified.	Major numbers change for software updates that alters how the software works completely (such as a new algorithm) or when the interface has been redesigned.
Minor	Minor numbers change when a section of the document has been added, removed, or rewritten.	Minor numbers change when the software has been modified by adding, removing, or rewriting code that does not alter how the system works or appears.
Update	Update numbers change when a document has been reviewed for formatting, semantic, and grammatical errors.	Update numbers change when an error or bug has been addressed, or when a slight modification of code (such as a string output) has changed.

Likewise, the layout of the directories of documents differs from software. The root directory shall contain three folders: ‘Archive’, ‘Documents’, and ‘Implementation.’ For documents, the ‘Document’ directory shall contain one folder named after each deliverable (SCM, SDD, etc.) as well as a folder named ‘Deliverables’ – this last folder will contain final submitted version of the deliverables. Within each individual deliverables folder, each user shall create a folder labeled as the sections that each member of the software development team is responsible for contributing. For example, someone responsible for section 3 of the SRS would have a folder named /Documents/SRS/Sec3. Along with these section-based folders, rough drafts of the final deliverable shall reside within each deliverable’s folder.

Each member of the software development team shall be delegated a portion of each writing assignment in accordance with the perspective and scope of their role in the software development process. For the sake of congruity, these sections shall be submitted in a timely manner to the V&V to be merged and revised; the V&V therefore will not be given as great of a portion of the work as other team members so that each paper can be carefully revised. Therefore, the folder labeled 'Deliverables' in the root directory shall only be accessed by the V&V for submitting finalized drafts; all other team members shall de facto have only reading privileges to this folder and not writing privileges. At each baseline, which is when the paper has been revised after a critique, the old contents of each deliverable's folder shall be moved to the 'Archive' folder found in the root directory, within an appropriately named folder, such as 'SCM before v2.0'.

For code, the root directory shall contain a separate folder for all implementation files and details names 'Implementation'. These files shall be maintained in adherence to the above mentioned standards. Older files can be moved to separate recursive folders, as long as proper labeling is used for these folders to clarify their contents (e.g. 'older src code files', 'old alg files'). The internal structure to the 'Implementation' folder is irrelevant so long as these practices are in place; for example, code for converting numerical data would be in a folder named 'data conversion.' In turn, file naming within this folder would still be regulated by our standardized naming process.

The folder labeled 'Archive' shall contain files which are not maintained, and thus do not need any intense investment in organization beyond proper naming. Files that do not require maintenance include, but are not limited to: old document versions, meeting records and requirements memos.

Configuration management shall occur by using the open source subversion system Git. Our repository shall reside within their free servers. In order to access the repository, a Git-Hub account will need to be created and each member will have to setup their own interface to connect to Git; the current release that the team is using is 1.8.1.2 found at http://git-scm.com. The repository will be readily accessible to the public with exclusive privileges for committing restricted to the software development team. Team members will readily have access to the repository from any computer that can run a git command. Appendix 5.2 has a tutorial on how to use Git.

As the repository shall be available from any acceptable access point, multiple users can be accessing the repository at any given time. Users will check out the latest version of the repository and commit changes as necessary to an individual branch. If multiple users are modifying the same files, different branches shall be created for each user, with the option to merge these branches back to the main branch created for everyone to access. Merging shall occur only after all other team members modifying the same file(s) have given a confirmation via email or our Google Groups forum. If only one team member is accessing a file, merging can occur only after a confirmation from the Lead Programmer has been received (Matthew Wojciechowski).

To prevent data loss it is imperative to implement a backup scheme. The method of choice will be to first enclose the entire directory and its contents within a zip file, doing so will ensure a Full back up. This approach can be handled natively by Git, using the archive command, which would output the archive as a tar or zip file. Once this is done the zip file will then be transferred to the team's shared Dropbox directory in a folder labeled 'Backup' by the current project leader making the revision. Backups will be labeled using the scheme:

Major. Minor.YYYYMMDD_HHmm.

YYYY would represent the 4 digit Gregorian year, MM the 2 digit month, DD the 2 digit day, HH the 2 digit hour, and mm the two digit minute. In the event that there is a loss or corruption of data the project will be restored from the last back up performed that is available in the Dropbox 'Backup' directory to. The current project lead will perform this recovery process.

3. Software Configuration Control

The following sections will detail the software configuration control for the system. The purpose of these sections is to show the mechanisms that will be used to control access to items in the configuration to ensure no unauthorized updates or collisions due to different team members modifying the system simultaneously.

3.1. Documentation

A change request form must be submitted for changes to any file that persists in the repository, in order to ensure proper recording of all proposed changes to the repository. These forms will make the tracking of changes more transparent to the CCB. It will, also, allow the CCB better control over changes to the system and other important documents. Either a developer or the CCB can initiate a request for change.

The change request form, to be filled out by the developer, will initiate the change process and will have the following fields:

- Date, (date request is made)
- Title, (Name of requested change.)
- Originator's Name,
- Priority, (Urgent or routine.)
- Request Description, (Description of changes to be made.)
- Justification, (Reason for change?)
- Impact. (What else will changes effect?)

The authority to approve changes rests in the CCB.

3.2. Configuration Control Board

A configuration control board shall be established to approve and manage modifications to the system. The primary responsibilities of the CCB are to:

- Evaluate proposed changes
- Establish baselines
- Identification of configuration items
- Evaluate and approve baseline changes
- Manage modifications to project requirements
- Authorize additions to baseline directory

The CCB will be composed of all of the members of the development team and under the direction of the board manager. The team member that is the lead (as decided by the guidance team) for the corresponding deliverables shall hold the position of board manager. The board manager will have two votes and general members will have one vote.

The CCB will debate and vote on proposed changes to the system configuration during the board's meetings. A simple majority will decide either to approve or reject the proposed change. Only changes approved by the CCB will made to the system. A change request that is time sensitive can be addressed and voted on through e-mail and other non-face-to-face forms of communication, if the board manger deems the change urgent. The boards' members will approve or disapprove of all changes with the following in mind:

- Consequences of not changing
- Resources needed for change
- Complexity of change
- Ease of integration
- Testing requirements
- Documentation requirements
- Possible alternatives

The CCB manager will be responsible for tasking out changes to the group members. Tasks will be disseminated to the developer that originally wrote the code, whenever possible, to ensure timely modifications, consistency, and continuity.

The V & V will use the change request form to report errors in the code for change. The process for correcting erroneous code is identical to the process for implementing other changes to the system configuration.

3.3. Procedures

All configuration items and files mentioned in section 2.1 will be stored and controlled by a repository. At the start of each step of the development process, each team member will checkout a copy of the whole repository. Once in development, if there are changes to be done on an item and the changes have been approved by the CCB and delegated to a team member then that team member will have priority on that item and any other items related to it. The team member assigned to this change must first re-sync the repository to make sure that the latest version of the item is available. Then the team member will checkout a new branch in the repository. If prompted the team member will have to give a reason for the new checkout. This checkout will allow the team member to work on the item without it being changed by other team members. Once the team member is done changing the item, he or she will merge the checked out branch with the main branch and recommit the item, detailing what has been done to the item in the message dialoged window. The requirements for recommitting and merging the items and branches are as follows:

- If the item is a document, then the CCB must review it and approve it to be a final version.
- If the item is a source code, then:
    - The code must compile without errors,
    - The code is tested and working properly,
        - There are test cases for all modified areas.
    - The CCB has reviewed and approved the code,
    - All necessary changes to documentation have been done.

The team member will commit the changes to the item only if the above requirements have been met and the CCB have approved the changes and results. Only then will the team member be allowed to commit the changes and state the reason why in the repository.

If a team member feels that an item should be reverted to a previous version, then the team member must bring the issue up with the CCB. The CCB will then review the request for change and decide if the change of version is necessary. If the CCB deems it necessary to revert, then this action will be completed in the repository by performing the “revert commit” command. The team member performing this action must give a reason for the revision.

The team member who is responsible for enforcing the rules and requirements listed above is the CCB manager (delegated lead for that part of the project). The lead will be responsible for the following:

- Ensuring that each assigned task is branched out,
- Ensuring that the team members follow the requirements for merging and recommitting changes,
- Ensuring that team members correctly describe and comment the changes upon commit,
- Bringing discrepancies to the attention of the CCB.

To instantiate a new major version of an item, defined in section 2.2, the follow process shall be implemented:

- The CCB has deemed it necessary and has approved the change,
- The team member in charge of the change has created all needed files/directories,
- The team member in charge of the change has branched out a new version of the repository to work with
- The team member in charge of the change has, upon completion, merged the branches,

- The team member in charge of the change has committed the changes and specified the reason why the change was done,
- The team member in charge of the change has documented the change in all appropriate places.

3.4. Baseline Creation

Baseline control is a tool used to keep track of revision control. Our development team will be using the GIT file repository system, which can make baseline control easier with the following attributes:

- Ability to rename and update item and/or files when appropriate
- Ability to place comments on any item submissions
- Separate file directories, which can keep items and deliverables organized and separate

The development team will keep track of version control using this baseline creation. The version numbering for the baseline will follow a strict procedure to keep track of version control. For an example, "documentV1.0mw" will act as a baseline. This document will be major version 1, and was created by Matthew W. Now after a change was made to the document, to where it was relabeled as "documentV1.24jm". The documents change was not a major change so it stays as version 1; however, two modules and four modifications were added to the original document updated by John M. Due to the document experiencing a major change, the document is now labeled "document2.0rv. This shows there was a major change and the developer that modified the document was Ramon V. During each submission of each item, the developer who is updating the baseline will leave a brief commit message using the "-m" function tools that GIT allows. Using this tool will briefly describe the changes made to document.

4. Software Configuration Auditing

The preceding sections have defined the standard for development that is the Software Configuration Plan. This section will entail the auditing process that will ensure that the standards previously stated within this document are adhered to, in order to provide a valid and verifiable implementation.

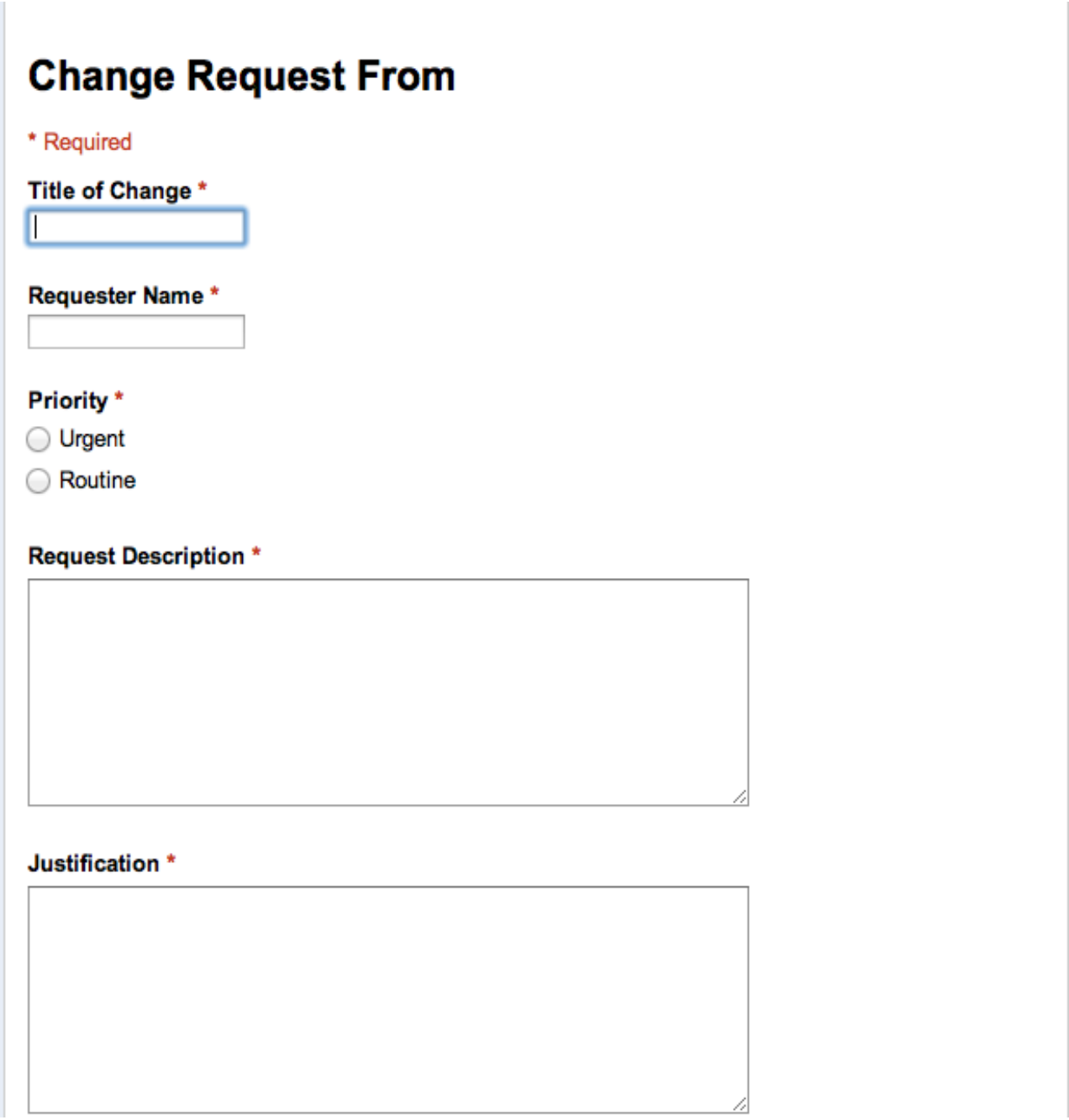
The CCB manager, the current project lead, will appoint two additional team members to serve as the CAB (Configuration Auditing Board). For any change requests made by the CCB, it will be the duty of the Auditing board to audit the change. Before an audit is made the Auditing board will ensure that the change request has been submitted and verify the request form submission correctly represents the team's decision. Once that is established an audit can commence. The Audit consists of these four parts:

- Verifying correct documentation of change requests
- Ensuring that SCP was adhered to.
- Verifying configuration has been correctly committed.
- Verifying that backups have been made and committed

If this auditing standard is correctly adhered to then the team can be ensured that the development will proceed in a consistent manner.

5. Appendix

5.1. Change Request Form



Change Request From

* Required

Title of Change *

Requester Name *

Priority *

Urgent

Routine

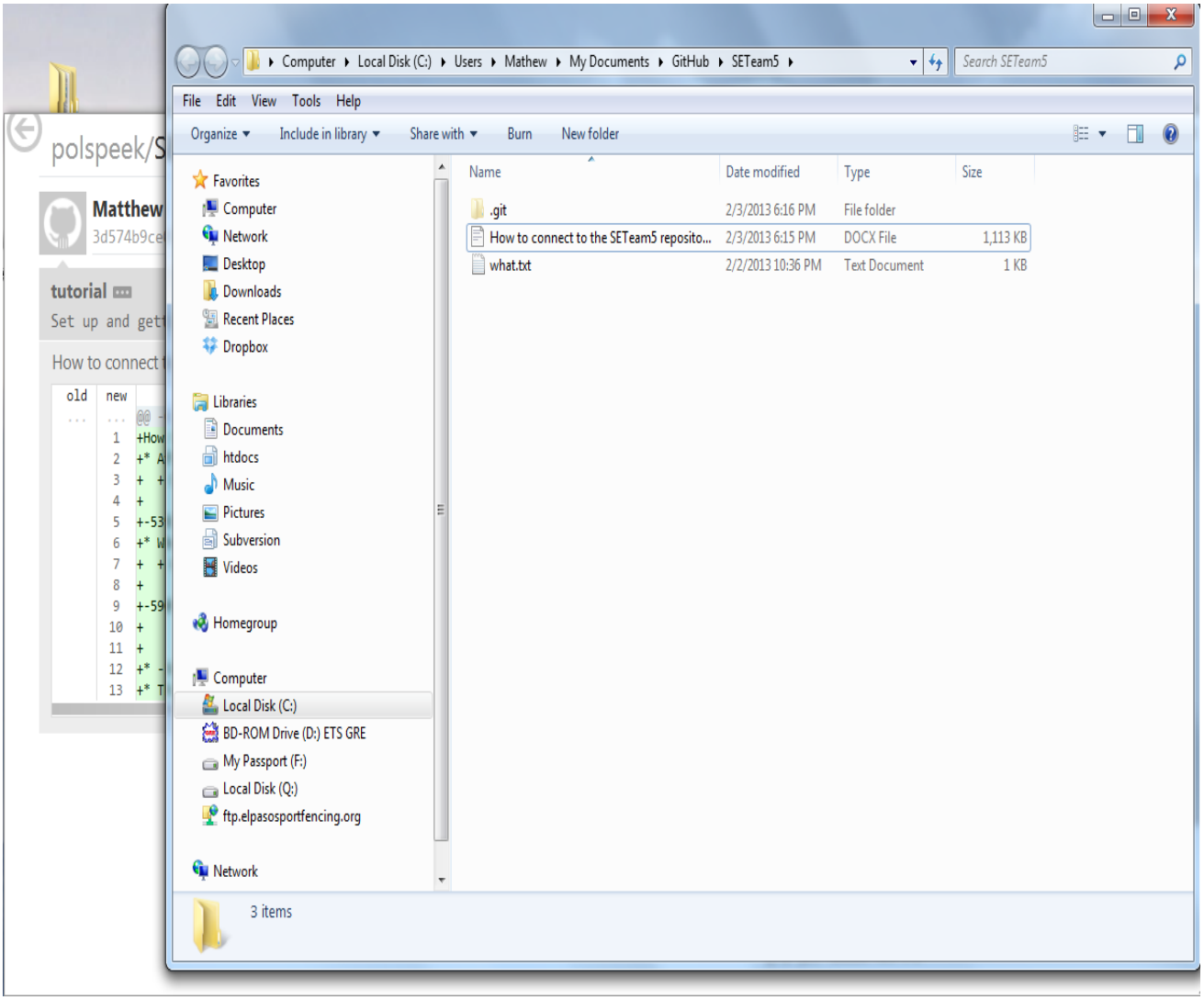
Request Description *

Justification *

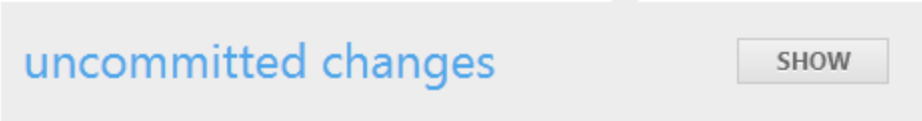
5.2. GIT Tutorial

This tutorial assumes that the user has already downloaded and installed Git and GitHub, made an account with GitHub, and has access to a repository.

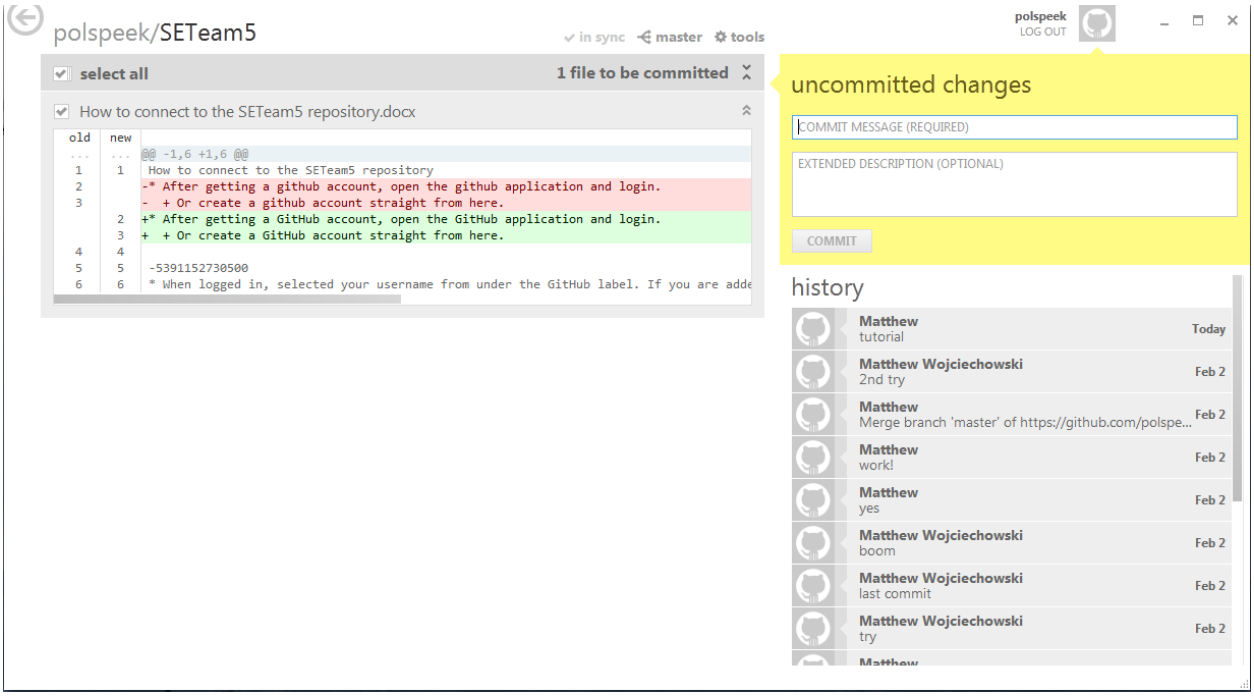
1) **Create and upload files**. When the user has cloned and opened his/her repository there is a folder created usually in MyDocuments>GitHub>'Name of Repo'. In it, there is a hidden folder called '.git'. It is in this folder where all files in the repo will be stored and accessed.



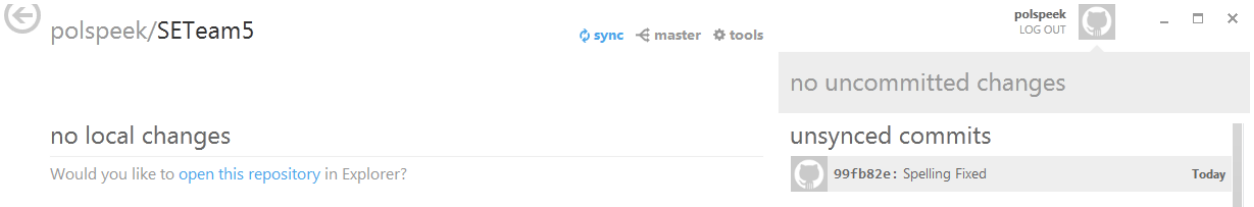
2) **Committing files**. Once the user has saved a file in the repo folder, the user should return to the GitHub Graphical User Interface (GUI). There the user will be prompted to commit all uncommitted changes.



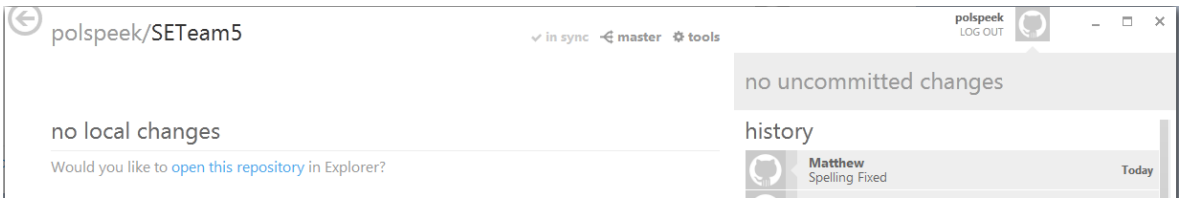
The user must enter a description in the yellow box to commit. Also, there is an addition field for more details on the file.



3) **Syncing**. When all the uncommitted changes are committed it is necessary to sync the repository. There is a sync button on the top of the page. When the user clicks on that, the repository will begin to sync.

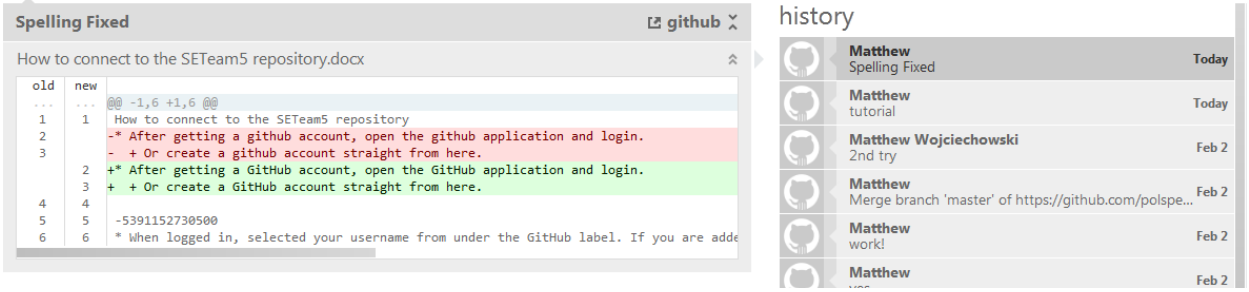


If successful, the button will change to “in sync”. Now all the new files and changes will be available to all users of the repository.

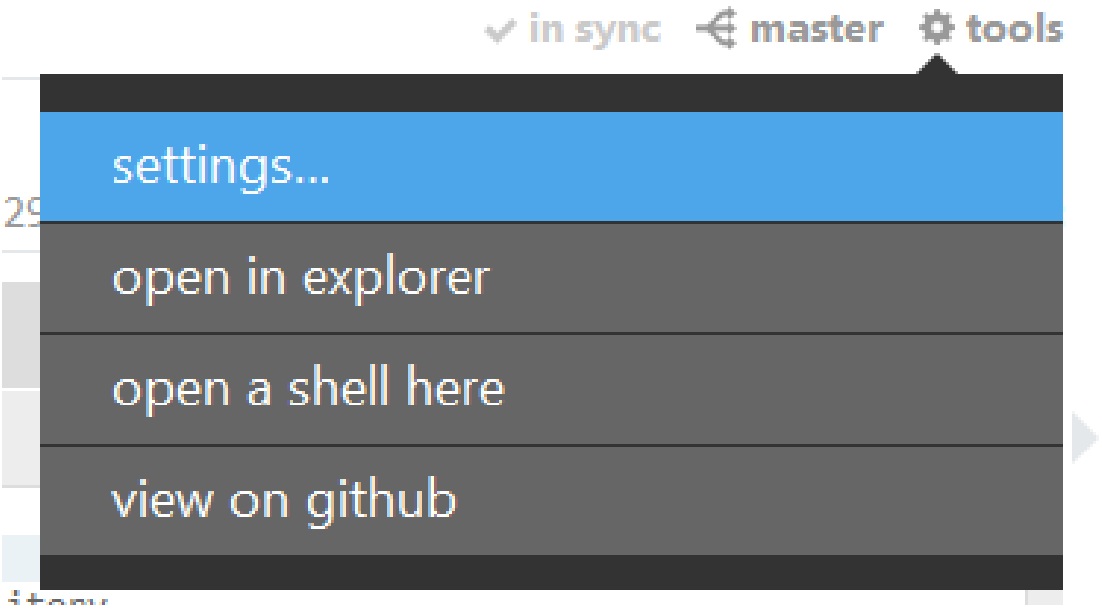


4) **Refreshing**. The repository will not re-sync automatically. The user must sync, or press F5 to refresh, to obtain any new changes to the repository.

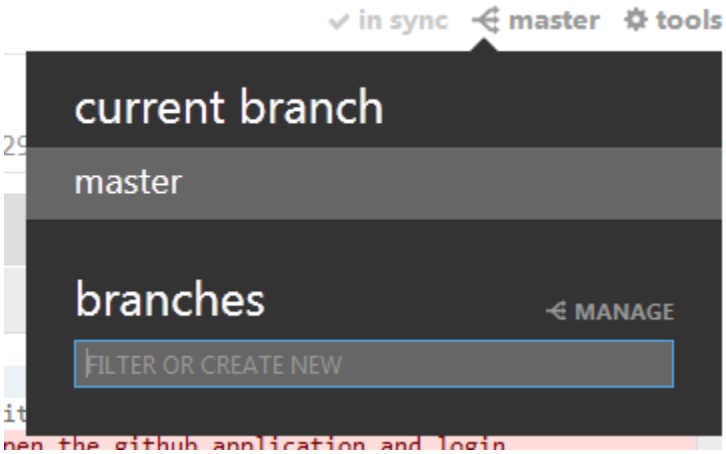
5) **History**. On the right side of the page there is a list of all the changes that have happened to the repository. These include: what changes were done, who committed the changes and when the changes were committed.



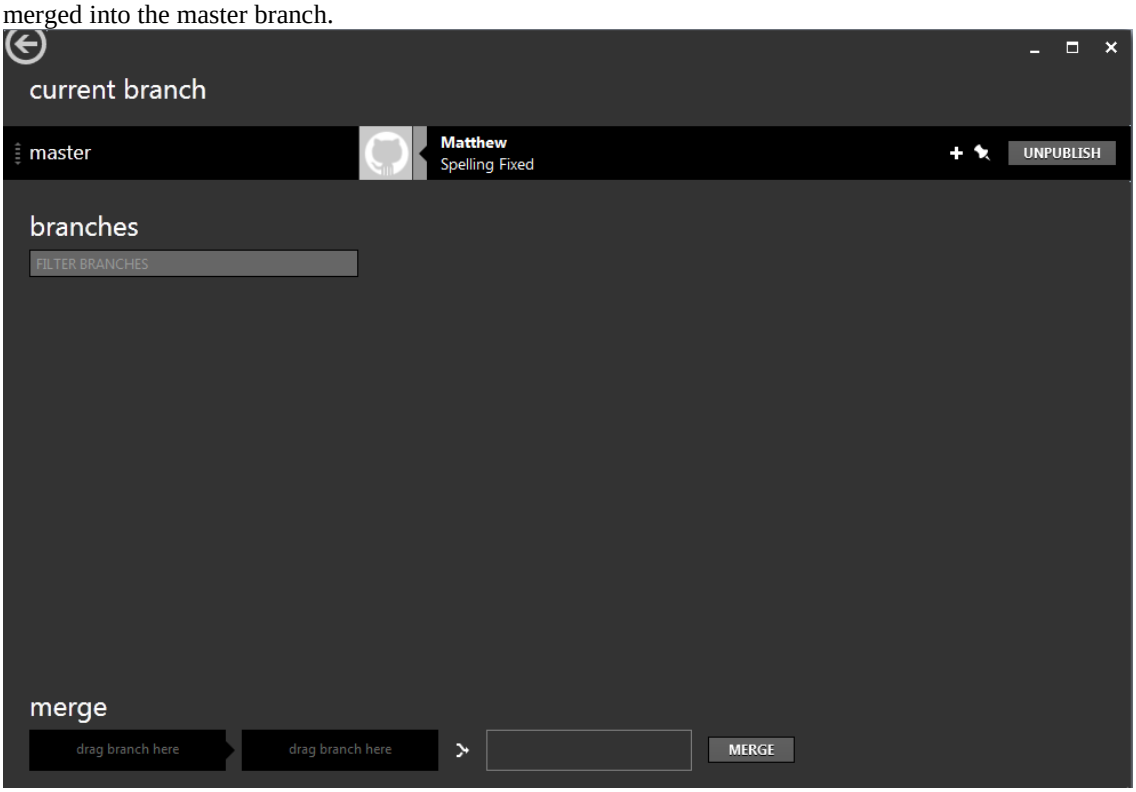
6) **Tools**. There is an icon on the top of the page called “Tools”. Clicking on that will bring up a menu that allows you open the repository folder, open a shell (CMD), and go to GitHub.com and view the repository there.



7) **Branching**. Git works off of branching and merging. What this means is that the user can checkout a new branch of the repository and work on his/her stuff in that branch and committing will have no effect on other user checkouts of the repo.
    a. To create a branch, the user should click the button called “master” at the top of the page and enter the name of the new branch the user wants.



    b. To merge the new branch back to the master branch, click the “new branch” icon (previously ‘master’) and select manage. Then drag the new branch to the bottom of the page where it says merge. Drag the master branch to the 2nd slot. Select merge. The branches should be merged into the master branch.



8) **Command Prompt**. GitHub and Git rely heavily on the command prompt.
    a. To open the CMD, click on Tools and select open shell.
    b. Here you can type in commands that will do most and more of the stuff that is available through the GUI.
    c. Two reference pages for CMD commands: http://www.siteground.com/tutorials/git/commands.htm and http://gitref.org/

@ END OF DOCUMENT