TEAM 2: WILL CODE FOR A'S
PROJECT PLAN
MIRROR APPLICATION

CSC-478B Fall 2015

OVERVIEW

1. Project Background and Description

This project is to develop a file backup / mirroring utility designed to help the average computer user to keep a selected set of folders and/or individual files mirrored to a second "mounted" location in the file system. The motivation behind this application is a simple utility that could be easily used with cloud-based storage tools like Google Drive, Microsoft OneDrive, Box, etc. that offer desktop clients. These clients allow those services to appear as local folders to the file system, thus eliminating the complexity for this application to have to deal with networking, etc.

Multiple versions of this application are anticipated to be developed during the project:

- Version 1: The initial version would support basic file mirroring in an on-demand fashion. The user would select a set of files to mirror and a destination for the FileSet to be mirrored to. It is envisioned that the destination would be a folder linked to a cloud storage provider such as Google Drive, etc. Once a FileSet containing at least one file and a destination are selected, the user can then initiate the mirroring operation.
- Version 2: The second iteration of the application would add logging to the system so the user would be able to see the details of which files were mirrored and any errors that occurred during the operation.
- Version 3: The third iteration would add only the capability to prevent the user from initiating an on-demand mirror operation if there is insufficient space at the destination.
- Version 4: The fourth iteration would add the ability for the user to specify folders to include in the mirror operation (not just files). If a folder were selected for inclusion, the application will recursively copy all of the contents, including subfolders, of that folder to the destination.
- Version 5: The fifth iteration would add both scheduling and "incremental" mirroring to the application. With scheduling, the user would be able to schedule a specific time that the operation would commence on a recurring basis. The incremental mirroring capability is the notion that the application would compare the files in the destination with the source FileSet and only copy those files that had changed as evidenced by a combination of the modification timestamp and the file size. The goal of this enhancement is to make the operation faster overall.
- Version 6: The sixth iteration of the application would add the ability to encrypt the files using standard library approaches to encryption such as BCrypt or similar (more research to be done to select specific algorithm and find an open source library to leverage for this). The motivation behind this enhancement is the fact that none of the free consumer cloud storage solutions offer encryption "at rest," thereby potentially exposing your files to hackers should their cloud service be compromised.
- Version 7: The seventh iteration would add the ability to compress the files during the mirror operation. This could be combined with the encryption mentioned for version 6.

- Version 8: This final iteration would add the ability to intelligently inspect the log of the previous run and allow the user to repeat the backup, copying only those files that didn't change or that had errors on the prior run.

2. Project Scope

- The software shall run on a minimum of Windows 7 Home Edition inclusive of the general requirements to run Windows 7 Home.
- The software shall run on a system meeting the minimum requirements for Windows 7 Home from a CPU, storage, and memory point of view. It is not anticipated that this application should require any "measureable" additional compute resources beyond what Windows 7 requires.
- The software will require at least Oracle Java 8 to be installed on the user's computer. Further, Java 8 will need to be configured such that access to the local filesystem is allowed.
- As is evident from the above, the application will be written in Java.
- The user will need to have sufficient disk space to hold the copy of the selected file set.
- If the user chooses to use the application with a cloud storage provider, the user shall install the appropriate client application to enable the local filesystem to interact with the cloud storage like it was a local drive or folder (ie, similar to a "shared drive."
- The application shall have a graphical user interface.
- The application shall function as a self-contained, standalone application that has no external requirements other than as noted above.
- Detailed project requirements with version assignment are presented in Appendix A of this document.

3. Org Chart

A high-level org chart representing the primary duties of each team member is presented below in Figure 1. The team has decided that all functions will be shared among members of the team, but that each team member shall have a primary responsibility as shown in the figure and is thus accountable for that aspect of the project. As an example, while Greg Palen is the primary coder, Ashley Robertson will contribute heavily to the code in the form of all of the user interface code and Zack Burch will contribute all of the testing code (which also fulfills his role as Tester).

As Team Leader, Zack will be responsible for the overall project progression as well as representing the "final vote" where there may be disagreement on a specific item or component. Zack will ensure that all team members are making satisfactory progress towards their assigned goals and will also handle the weekly status report communication.

Documentation is a shared responsibility that is divided roughly along the lines of the team member's primary responsibility. As an example, Greg will handle the majority of the Programmer's Manual since he is the coder, Ashley will handle the majority of the User's Manual as both the Architect and the coder of the UI component of the application, and Zack will cover test documentation.

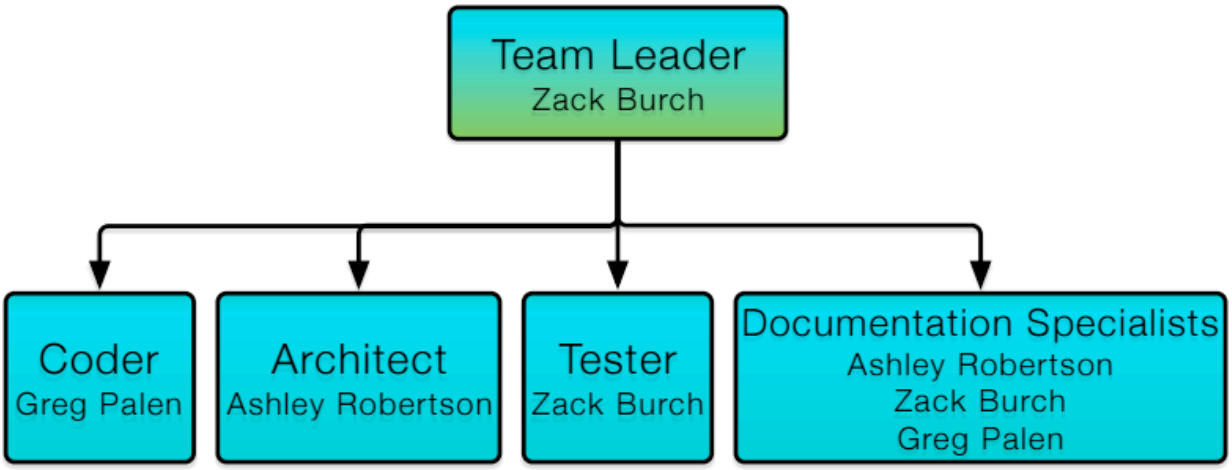
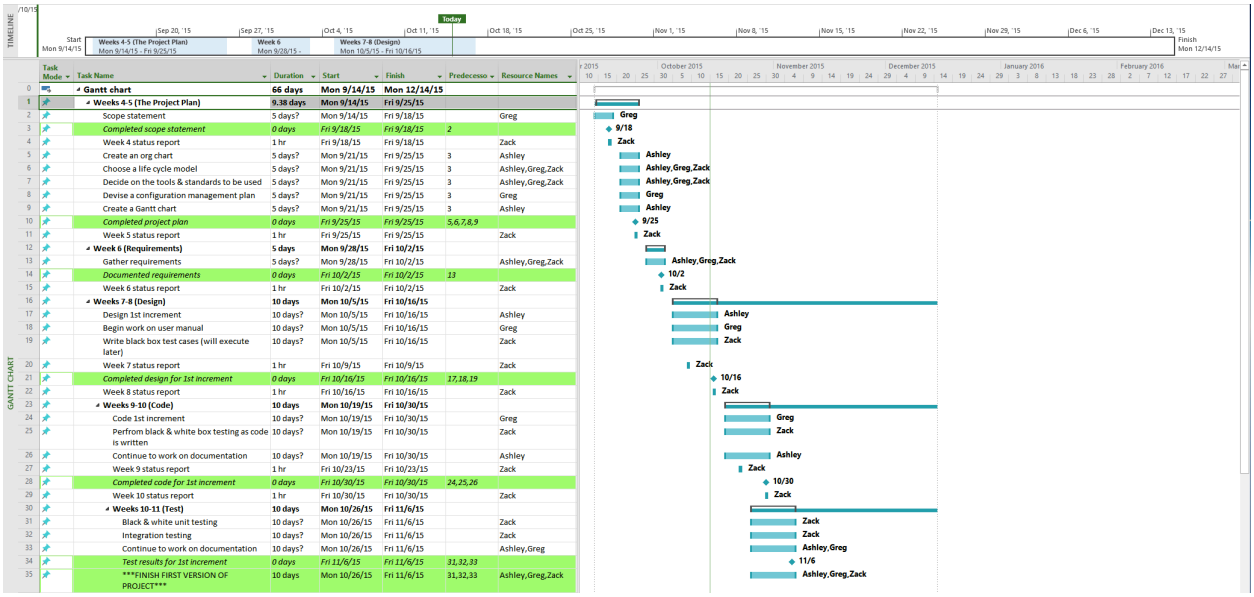


Figure 1 - Team Organization

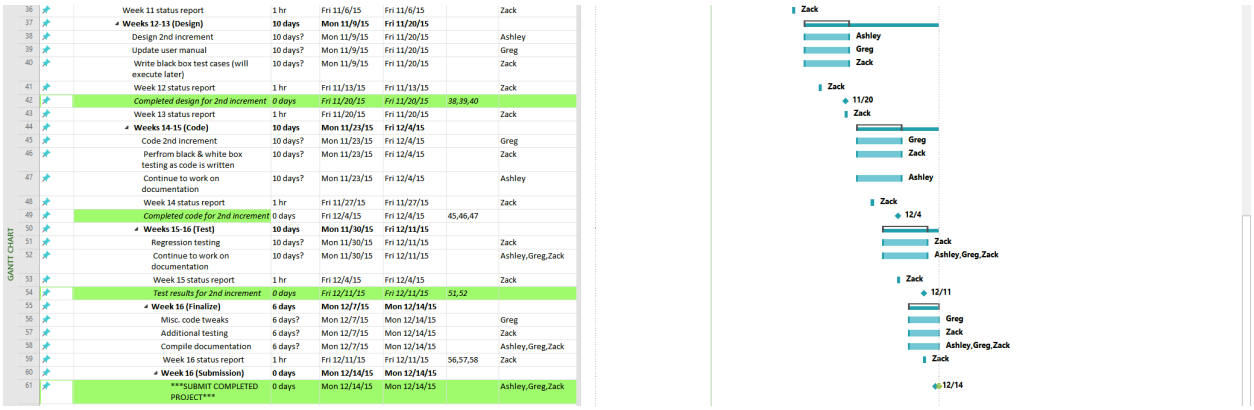
4. High-Level Project Schedule (Gantt Chart)

The overall project schedule is depicted below in Figure 2. Milestone tasks are identified as well as key subtasks for each milestone. Where reasonable and appropriate, specific team members are identified as responsible for some of the sub-tasks. This chart represents an iterative approach to project development that parallels the version-oriented development that will be followed, so some tasks will be repeated at points in time as each iteration evolves.



	Task Mode	Task Name	Duration	Start	Finish	Predecesso	Resource Names
0		Gantt chart	66 days	Mon 9/14/15	Mon 12/14/15		
1		Weeks 4-5 (The Project Plan)	9.38 days	Mon 9/14/15	Fri 9/25/15		
2		Scope statement	5 days?	Mon 9/14/15	Fri 9/18/15		Greg
3		*Completed scope statement*	*0 days*	*Fri 9/18/15*	*Fri 9/18/15*	*2*	
4		Week 4 status report	1 hr	Fri 9/18/15	Fri 9/18/15		Zack
5		Create an org chart	5 days?	Mon 9/21/15	Fri 9/25/15	3	Ashley
6		Choose a life cycle model	5 days?	Mon 9/21/15	Fri 9/25/15	3	Ashley,Greg,Zack
7		Decide on the tools & standards to be used	5 days?	Mon 9/21/15	Fri 9/25/15	3	Ashley,Greg,Zack
8		Devise a configuration management plan	5 days?	Mon 9/21/15	Fri 9/25/15	3	Greg
9		Create a Gantt chart	5 days?	Mon 9/21/15	Fri 9/25/15	3	Ashley
10		*Completed project plan*	*0 days*	*Fri 9/25/15*	*Fri 9/25/15*	*5,6,7,8,9*	
11		Week 5 status report	1 hr	Fri 9/25/15	Fri 9/25/15		Zack
12		Week 6 (Requirements)	5 days	Mon 9/28/15	Fri 10/2/15		
13		Gather requirements	5 days?	Mon 9/28/15	Fri 10/2/15		Ashley,Greg,Zack
14		*Documented requirements*	*0 days*	*Fri 10/2/15*	*Fri 10/2/15*	*13*	
15		Week 6 status report	1 hr	Fri 10/2/15	Fri 10/2/15		Zack
16		Weeks 7-8 (Design)	10 days	Mon 10/5/15	Fri 10/16/15		
17		Design 1st increment	10 days?	Mon 10/5/15	Fri 10/16/15		Ashley
18		Begin work on user manual	10 days?	Mon 10/5/15	Fri 10/16/15		Greg
19		Write black box test cases (will execute later)	10 days?	Mon 10/5/15	Fri 10/16/15		Zack
20		Week 7 status report	1 hr	Fri 10/9/15	Fri 10/9/15		Zack
21		*Completed design for 1st increment*	*0 days*	*Fri 10/16/15*	*Fri 10/16/15*	*17,18,19*	
22		Week 8 status report	1 hr	Fri 10/16/15	Fri 10/16/15		Zack
23		Weeks 9-10 (Code)	10 days	Mon 10/19/15	Fri 10/30/15		
24		Code 1st increment	10 days?	Mon 10/19/15	Fri 10/30/15		Greg
25		Perfrom black & white box testing as code is written	10 days?	Mon 10/19/15	Fri 10/30/15		Zack
26		Continue to work on documentation	10 days?	Mon 10/19/15	Fri 10/30/15		Ashley
27		Week 9 status report	1 hr	Fri 10/23/15	Fri 10/23/15		Zack
28		*Completed code for 1st increment*	*0 days*	*Fri 10/30/15*	*Fri 10/30/15*	*24,25,26*	
29		Week 10 status report	1 hr	Fri 10/30/15	Fri 10/30/15		Zack
30		Weeks 10-11 (Test)	10 days	Mon 10/26/15	Fri 11/6/15		
31		Black & white unit testing	10 days?	Mon 10/26/15	Fri 11/6/15		Zack
32		Integration testing	10 days?	Mon 10/26/15	Fri 11/6/15		Zack
33		Continue to work on documentation	10 days?	Mon 10/26/15	Fri 11/6/15		Ashley,Greg
34		*Test results for 1st increment*	*0 days*	*Fri 11/6/15*	*Fri 11/6/15*	*31,32,33*	
35		***FINISH FIRST VERSION OF PROJECT***	10 days	Mon 10/26/15	Fri 11/6/15	31,32,33	Ashley,Greg,Zack

Figure 2 - Project Schedule Gantt Chart



36	Week 11 status report	1 hr	Fri 11/6/15	Fri 11/6/15		Zack
37	**Weeks 12-13 (Design)**	**10 days**	**Mon 11/9/15**	**Fri 11/20/15**		
38	Design 2nd increment	10 days?	Mon 11/9/15	Fri 11/20/15		Ashley
39	Update user manual	10 days?	Mon 11/9/15	Fri 11/20/15		Greg
40	Write black box test cases (will execute later)	10 days?	Mon 11/9/15	Fri 11/20/15		Zack
41	Week 12 status report	1 hr	Fri 11/13/15	Fri 11/13/15		Zack
42	*Completed design for 2nd increment*	*0 days*	*Fri 11/20/15*	*Fri 11/20/15*	*38,39,40*	
43	Week 13 status report	1 hr	Fri 11/20/15	Fri 11/20/15		Zack
44	**Weeks 14-15 (Code)**	**10 days**	**Mon 11/23/15**	**Fri 12/4/15**		
45	Code 2nd increment	10 days?	Mon 11/23/15	Fri 12/4/15		Greg
46	Perfrom black & white box testing as code is written	10 days?	Mon 11/23/15	Fri 12/4/15		Zack
47	Continue to work on documentation	10 days?	Mon 11/23/15	Fri 12/4/15		Ashley
48	Week 14 status report	1 hr	Fri 11/27/15	Fri 11/27/15		Zack
49	*Completed code for 2nd increment*	0 days	Fri 12/4/15	Fri 12/4/15	45,46,47	
50	**Weeks 15-16 (Test)**	**10 days**	**Mon 11/30/15**	**Fri 12/11/15**		
51	Regression testing	10 days?	Mon 11/30/15	Fri 12/11/15		Zack
52	Continue to work on documentation	10 days?	Mon 11/30/15	Fri 12/11/15		Ashley,Greg,Zack
53	Week 15 status report	1 hr	Fri 12/4/15	Fri 12/4/15		Zack
54	*Test results for 2nd increment*	*0 days*	*Fri 12/11/15*	*Fri 12/11/15*	*51,52*	
55	**Week 16 (Finalize)**	**6 days**	**Mon 12/7/15**	**Mon 12/14/15**		
56	Misc. code tweaks	6 days?	Mon 12/7/15	Mon 12/14/15		Greg
57	Additional testing	6 days?	Mon 12/7/15	Mon 12/14/15		Zack
58	Compile documentation	6 days?	Mon 12/7/15	Mon 12/14/15		Ashley,Greg,Zack
59	Week 16 status report	1 hr	Fri 12/11/15	Fri 12/11/15	56,57,58	Zack
60	**Week 16 (Submission)**	**0 days**	**Mon 12/14/15**	**Mon 12/14/15**		
61	***SUBMIT COMPLETED PROJECT***	0 days	Mon 12/14/15	Mon 12/14/15		Ashley,Greg,Zack

GANTT CHART

5. Tools and Standards

Software Development Process Model

The Software Development Process Model that will be followed is best described as a combination of the early aspects of the Waterfall approach and then transition to an Agile/Scrum approach for the actual development effort.

From a Waterfall point of view, the Project Requirements will be defined first, followed by a high-level architecture, which is then followed by the design of key classes and interactions. Upon conclusion of these activities, the project then transitions to an Agile-oriented approach for the iterative development of the code base. Borrowing from the concepts of Scrum, the team will follow a weekly Sprint approach where specific tasks and goals are selected for completion during a weekly meeting. Due to the distributed nature of the team members and the difficult scheduling constraints that employment represents to this project, a daily stand-up meeting will not be held. Rather, as appropriate, a combination of Slack and Waffle will be used by team members to communicate progress and issues (details on these applications are below).

Within each Sprint, particularly in the earlier phases of the project, the team will seek to validate the requirements, design, and architecture decisions made during the Waterfall part that preceded. It is desired to “lock down” on those elements as soon as practicable during the actual development to minimize delays later in the project. In that regard, the first phase that is being described in a Waterfall manner is actually subject to some small modification, but again the goal is to think through those aspects during the Waterfall phase with as much care as possible to minimize the potential to have to revisit those decisions.

Development Tools and Applicable Standards/Guides

In an effort to maximize team productivity and minimize rework due to workspace variations, a number of tools and standards are described below for use in the project. Every effort should be made to adhere to the standards and guides that are discussed, but it is understood that exceptions can occur though they do need to be minimized. From a tooling point of view, the tools described are to be considered mandatory for use due to the risk that variation poses to the project schedule should varying development environments introduce inconsistencies or incompatibilities.

Mandatory Tools and Standards

a) Communications: team communications will be handled mainly via Slack (www.slack.com). Channels will be created within Slack to focus communications on specific topics such as Project Ideas, Code, General, Requirements, etc. This provides a forum for team discussion of various topics and will also provide a history of those discussions.
b) Development Tools

a. Git will be used for version control. A Master branch will contain only final release code by version. The Develop branch will be used for the current version being worked on, and feature branches will be used for issue- or feature-specific development. See Section 6 - Configuration Management Plan below for more details particularly as it relates to the required branching scheme to be used with Git along with semantic versioning requirements.
b. As Git is a distributed version control system, Github will be used as a central repository for the entire code base. All code will be pushed up to Github on a regular basis so that all team members have access to the latest code. A Git Pull should be performed prior to starting work each time so that any committed changes made by other team members can be integrated.
c. Github Issues (an issue tracking capability of Github that is integrated with the repo itself) will be used for both feature and issue management such as features lists, bug reports, etc. Feature and requirement tasks will be created in Github Issues to assist in planning sprints and assigning work to specific team members (see the next paragraph regarding Waffle for how issues will be viewed and interacted with).
d. A free web service named Waffle (http://www.waffle.io) will be used for organization of tasks and issues. Waffle integrates with Github Issues in a bi-directional fashion and allows Waffle to depict logged issues in the context of a Scrum-based board-view with each issue or feature represented as a card in a column. Several columns will be defined that link the Github Issues Tags to stages of work such as “Icebox” (a grass-catcher list of all work to be completed over the life of the project), Questions, Backlog, Ready, In Progress, and Done. The combination of Waffle and Github also support assigning issues to specific team members for resolution as well as a chat capability to discuss issues and solutions within the context of the specific issue.

   New issues can be created either in Waffle (the preferred creation tool as it’s easier) or the Github Issues tab on Github. By adding directly in Waffle, the user can see the issue on the board immediately.
e. The Eclipse IDE (Mars.1 Java EE edition) will be used for development. This IDE is a requirement. However, the user is free to also install eGit and Mylyn with the Github Issue Connector to be able to integrate with Git, Github, and Github Issues all without leaving the IDE user interface. These additional plug-ins are not required.
f. JUnit 4 will be used for testing.
g. Java 8 SE (Oracle JDK 8 specifically) will be used for all code development. The Project Settings will be configured such that all developers are using the same revision of Java 8. The revision in effect at the commencement of the project should be kept through project completion to maximize consistency. Any new versions of Java that happened to be released during the course of development should be tested in a separate clone of the project.
h. DCP Setup Maker (https://sourceforge.net/projects/devcompack/files/latest/download) will be used to create an Installer program suitable for installing the application into a Windows environment.

c) File Storage
   a. Code
      i. All code will be stored on Github under the “official” repository at https://github.com/codingvirtual/CSC478B/
      ii. The project will follow the “Gitflow Workflow” (see https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow for a detailed explanation/tutorial of this workflow as well as the Configuration Management Plan section of this document.
   b. Documentation and Related Files: all other files and documents besides “code” will be stored in a central Google Drive folder that is shared among team members. To roughly mimic the Gitflow Workflow for non-code files, two folders will be used to store project documentation and ancillary files:

i. Master – this folder will contain the final version of the documentation will be tagged to a specific release.
ii. Develop – this folder will contain work-in-progress documents. When a document is complete and ready for inclusion in a release, it will be moved into the master folder and a version number will be appended to the document name.

d) Mandatory Documentation Standards
   a. All FINAL documentation (other than code comments and the Programmer's Manual, which will be developed using Javadoc) will be created and maintained in Microsoft Word format. Effort will be made to ensure a consistent look and feel between different documents in an effort to try to create somewhat of a "brand" appearance. Appropriate documents such as the User's Manual will also be made available via public website (http://codingvirtual.github.io/CSC478B). Within Microsoft Word, the "Track Changes" feature should be enabled to facilitate change management of these documents.
   b. In addition to Word-formatted final documents, there will be a PDF version of the final documentation available as well.
   c. Microsoft Project will be used to develop a Project Plan and associated Gantt chart.
   d. The Programmer's Manual documentation will be generated using Javadoc comments within the source code which are then converted into HTML. The resulting HTML will be available via public website (http://codingvirtual.github.io/CSC478B).
      i. Package-info.java files will be created to house package-level overview comments.
      ii. A single overview.html file will be created to house the overall application overview summary comments.
      iii. Methods should be properly documented with the appropriate Javadoc comments including a summary of the method's functionality with traceability to specific requirements if applicable, the type, name and description of each formal parameter, and the type and description of any returned data.
      iv. Any code that is capable of throwing an exception needs to also have an @throws Javadoc comment explaining the exception(s) thrown and as appropriate, the likely causes for those exceptions when they are specifically related to the operation of the affected code or the operation of any client that may be using the method.
      v. Class-level Javadoc comments shall include the overall purpose and functionality the class provides along with applicability to requirements either in summary or to specific requirements where appropriate. The primary author of the class should also be defined.

Preferrred Tools and Standards

a. Alternate Documentation Tools
   a. The Draw.io Chrome Extension (or desktop application if preferred; www.draw.io) is the preferred tool for the creation of all technical, architectural, and design diagrams including all UML diagrams. The UML 2.0 standard should be followed for UML diagrams as much as possible.
      i. Final diagrams should be saved not only in native format but also in JPEG format for easy inclusion into any Word or HTML documents that may require the diagram.
   b. Any word processor can be used for draft documentation but when ready for inclusion into a Candidate Final Document, files must be converted by the author to Word format before submitting to the Documentation Specialist that will be handling final document assembly and formatting.
b. Style & Programming Standard
   a. Javadoc doc comments will follow the Javadoc section of Google's Java style guide (see section 7 of https://google.github.io/styleguide/javaguide.html#s7-javadoc). Additionally, the overall style described in the above Google style guide will be used in source code with particular attention to Section 4 (Formatting) and Section 5 (Naming), though strict compliance will not be enforced.

b. Variables, methods, and test methods should all be named to communicate as much as possible the nature of the element. In particular, test methods should be named using the "given_when_then" structure that is loosely defined in the context of Behavior-Driven Development.
c. Packages will be created to group related classes to improve cohesion and should be named such that the overall function of the package is communicated in the name of the package.
d. Each source code file will contain appropriate Javadoc headers indicating the purpose of the file and traceability to requirements at a summary level.

6. Configuration Management Plan

As mentioned in Tools and Standards, Git will be used for Version Control and, ostensibly, for Configuration Management, with Github being the official centralized storage repository for all code. The approach to version and configuration control is thoroughly documented as this link: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

The general approach is as follows:

1. The "master" branch is reserved strictly for releases.
2. The "develop" branch will contain the in-progress development towards the next release. No commits should be done directly to the develop branch if at all possible.
3. For each feature/task/issue a developer is working on, the developer will create a specific branch for that feature named using the following naming convention: "feature-<feature_name>#<related-issue-number>." For example, the User Interface feature branch might be named "feature-ui#5" because the branch houses a feature (in this case the "UI feature" specifically) and it's linked to Issue #5 in the Github Issues tracker (this also facilitates integration with Waffle; when a branch is created in this manner, Waffle automatically moves the issue's status to "In Progress" and when the branch is merged via Pull Request back to the Develop branch, Waffle will automatically mark the issue as Done).
4. When a developer determines a feature is ready for code review, the developer will commit and push the code to the origin repository on Github and submit a pull request via Github. The other developers should review the code and document any concerns or questions under the Comments for the issue the branch is related to.
5. The team will review the submitted code and upon approval, the lead coder will merge the feature branch into the "develop" branch.
6. The process above repeats until all features for a given release are merged with the "develop" branch.
7. At this time, integration and system testing will commence on the "develop" branch
8. Upon completion of satisfactory testing, the develop branch will be merged with the master branch and that branch will then be tagged as a release.

Semantic Versioning will be used for version numbering with the first digit indicating the Release Number, the 2nd digit indicating the patch level, and if present, the 3rd digit will signify a specific build.

APPENDIX A – DETAILED PROJECT REQUIREMENTS BY VERSION

In the table below, each significant planned feature of the project is described and also assigned two different sets of identification numbers. In the first column, a Specification Number is identified for each feature or requirement. Documentation for any code or testing should specifically identify the Specification Number(s) that the documentation or code traces to.

Specification numbers are in dotted-decimal form that also communicates version-related information as well as prioritization.

1. Digit 1 indicates the Targeted Release the specification relates to.
2. Digit 2 indicates the Targeted Code Version the specification is assigned to. Targeted Code Version is incremented if and only if there are multiple versions planned that ultimately lead to a Release (such as when features are being built-up one at a time).
3. Digit 3 indicates the Priority of the specification or requirement AS IT RELATES TO OTHER ITEMS within the same VERSION. Priority numbers reset to 1 when the Targeted Code Version is incremented.
4. Digit 4 indicates the Task Number and is generally only relevant when there are several sub-tasks or sub-requirements that collectively make up a fully qualified single requirement or feature.

Semantic Version numbers are also dotted-decimal in nature and have a relationship to the Specification Number as well (rather, the Specification Numbering really has the relationship with the Semantic Versioning approach).

1. Digit 1 indicates the Release number of the code. This shall start with Release 1, which will only be generated when all of the requirements and tasks that make up Version 1 are completed (that is, when all of the code for Version 1 is complete, it becomes Release 1).
    a. “Release 0” is not really a Release, but rather represents the in-progress development towards Release 1. For this reason, the reader will note that the Semantic Version is one digit shorter than the Specification Number as the first digit of the Specification Number indicates the release that will include that specification.
    b. This digit ONLY increases when the underlying application functionality is modified in such a way that the next Release is in some way incompatible with the previous release. This could be due to internal compatibility such as changes to key interface method signatures or due to external compatibility such as file format changes, preference changes, or new functionality that is not backwards-compatible.
2. Digit 2 indicates the version number of the release.
    a. This digit increases when new functionality is introduced, such as a new feature, that does not affect the backward compatibility of existing code that came before it.
    b. This digit also resets to 1 when a Release is generated (in other words, when the Release number is incremented, the version number resets to 1).
3. Digit 3 seeks to indicate the overall progression of the development effort by relating to last priority addressed by the version. A quick review of the Semantic Version column and it’s relation to the Release, Version, and Priority columns will reveal the relationship between these elements.
4. Effort to maintain the “integrity” of the first two digits should be maintained. The 3rd digit, if used, should align with the Specification Number for consistency rather than an increment-based approach.
5. The Semantic Version is what will be used as the Tag Name for a Git Tag applied to a specific point in time on the develop and master branches so that a developer can get back to that state relatively easily by searching for the tag in the repository.

Specification Number	Semantic Version #	Requirement	Release	Version	Priority	Task #	
1.1.1.1	0.1.0	The user must be able to create a list of files that the application is to operate on.	1	1	1	1	
1.1.2.1	0.1.1	The user shall be able to specify a destination folder or drive for the backup operation to store the backup into.	1	1	2	1	
1.1.3.1	0.1.2	The user must be able to modify the list of files to operate on prior to backup. After backup, the user can modify the list and save as a new backup.	1	1	3	1	
1.1.4.1	0.1.3	The user shall have the capability of executing a backup on demand.	1	1	4	1	
1.1.5.1	0.1.4	The user must be notified of the status of any backup (failure or success).	1	1	5	1	
1.1.6.1	0.1.5	The installer shall check the target machine for compliance with the minimum system requirements defined in Section 1 above.	1	1	6	1	
1.1.6.2	0.1.5	Is at least Java 8 installed?	1	1	6	2	
1.1.6.3	0.1.5	Is the host operating system at least Windows 7?	1	1	6	3	
1.1.6.4	0.1.5	Does the installation location have enough space to hold the installation?	1	1	6	4	Upon completion, Release 1.0 happens
1.1.7.5	0.1.6	The user shall name a backup, and the backup shall be saved in a folder with the chosen name at the root of the destination.	1	1	7	5	
2.1.1.1	1.1.0	The system shall log the successful copying of each individual file.	2	1	1	1	
2.1.2.1	1.1.1	Each log entry will contain a date/time stamp.	2	1	2	1	
2.1.3.1	1.1.2	The application shall record in the log any file system errors that occur when trying to copy. If the error is a file error, it shall be logged and the system will continue with the next file in the backup set (meaning an error should not abort the entire backup if at all possible).	2	1	3	1	
2.1.1.1	1.1.0	The user shall be able to view the logs in the user interface.	2	1	1	1	Upon completion, Release 2.0 happens
3.1.2.1	2.1.1	The application shall prevent the backup operation if there is not sufficient space to store the source files in the destination location.	3	1	2	1	Upon completion, Release 3.0 happens
4.1.1.1	3.1.0	The user must be able to create a list of folders that the application is to operate on.	4	1	1	1	
4.1.1.2	3.1.0	The system shall log each successfully copied folder. A folder shall only be deemed completely copied if all of its contents (recursively) have been successfully copied.	4	1	1	2	
4.1.2.1	3.1.1	The user must be able to modify the list of folders to operate on.	4	1	2	1	
4.1.3.1	3.1.2	If the user specifies a folder, the complete contents of the folder shall be copied including hidden files.	4	1	3	1	Upon completion, release 4.0 happens

5.1.1.1	4.1.0	The user must be able to schedule a backup operation to occur on some recurring interval.	5	1	1	1	
5.1.2.1	4.1.1	Backup operations shall happen in the background and should minimize the impact to the user's machine.	5	1	2	1	
5.1.3.1	4.1.2	The application shall use the file modification date and file size of the source file to determine if the file has been changed since the last backup. ONLY changed or new files will be copied. Existing files that have had no change shall be skipped.	5	1	3	1	
5.1.4.2	4.1.3	The log shall note if files/folders were skipped due to no change detected.	5	1	4	2	Upon completion, Release 5.0 happens
6.1.1.1	5.1.0	The system shall support the capability to apply various optional transformations to the backup during the operation (such as encryption and compression)	6	1	1	1	
6.1.1.1	5.1.0	The user shall be able to optionally encrypt the backup.	6	1	1	1	
6.1.1.2	5.1.0	The backup will be encrypted using a user-defined passphrase	6	1	1	2	
6.1.1.3	5.1.0	The encryption will be handled via open-source encryption libraries	6	1	1	3	Upon completion, Release 6.0 happens
7.1.1.1	6.1.0	The user shall be able to optionally compress the backup.	7	1	1	1	
7.1.1.2	6.1.0	Compression will be handled via open-source compression libraries	7	1	1	2	
7.1.1.3	6.1.0	Transformations must be able to be combined (ie, ability to both compress and encrypt the backup)	7	1	1	3	Upon completion, Release 7.0 happens
8.1.1.1	7.1.0	If a backup operation does not complete entirely, the user shall be given the option to repeat the backup. If the user elects to complete the backup immediately, the backup will only copy those files that did not successfully complete on the immediate prior backup.	8	1	1	1	Upon completion, Release 8.0 happens