CMSC 447

Software Development Plan (SDP)

[1 Scope 5](#_Toc432620797)

[1.1 Identification 5](#_Toc432620798)

[1.2 System overview 5](#_Toc432620799)

[1.3 Document overview 6](#_Toc432620800)

[1.3.1 Version Modification 6](#_Toc432620807)

[1.4 Relationship to other plans 6](#_Toc432620801)

[2 Overview of required work 7](#_Toc432620803)

[3 Plans for performing general software development activities 8](#_Toc432620804)

[3.1 Software development process 8](#_Toc432620805)

[3.2 General plans for software development 8](#_Toc432620806)

[3.2.1 Software development methods 8](#_Toc432620807)

[3.2.2 Standards for software products 8](#_Toc432620808)

[3.2.3 Reusable software products 9](#_Toc432620809)

[3.2.4 Handling of critical requirements 10](#_Toc432620810)

[3.2.5 Computer hardware resource utilization 10](#_Toc432620811)

[3.2.6 Recording rationale 11](#_Toc432620812)

[3.2.7 Access for acquirer review 11](#_Toc432620813)

[4 Plans for performing detailed software development activities 11](#_Toc432620814)

[4.1 Project planning and oversight 11](#_Toc432620815)

[4.1.1 Software development planning (covering updates to this plan) 11](#_Toc432620816)

[4.1.2 CSCI test planning 11](#_Toc432620817)

[4.1.3 System test planning 12](#_Toc432620818)

[4.1.4 Software installation planning 12](#_Toc432620819)

[4.1.5 Software transition planning 12](#_Toc432620820)

[4.1.6 Management Reviews 12](#_Toc432620820)

[4.2 Establishing a software development environment 13](#_Toc432620822)

[4.2.1 Software engineering environment 13](#_Toc432620823)

[4.2.2 Software test environment 13](#_Toc432620824)

[4.2.3 Software development library 13](#_Toc432620825)

[4.2.4 Software development files 13](#_Toc432620826)

[4.2.5 Non-deliverable software 13](#_Toc432620827)

[4.3 System requirements analysis 14](#_Toc432620828)

[4.3.1 Analysis of user input 14](#_Toc432620829)

[4.3.2 Operational concept 14](#_Toc432620830)

[4.3.3 System requirements 14](#_Toc432620831)

[4.4 System design 14](#_Toc432620832)

[4.4.1 System-wide design decisions 14](#_Toc432620833)

[4.4.2 System architectural design 14](#_Toc432620834)

[4.5 Software requirements analysis 14](#_Toc432620835)

[4.6 Software design 15](#_Toc432620836)

[4.6.1 CSCI-wide design decisions 15](#_Toc432620837)

[4.6.2 CSCI architectural design 15](#_Toc432620838)

[4.7 Software implementation and unit testing 15](#_Toc432620840)

[4.7.1 Software implementation 15](#_Toc432620841)

[4.7.2 Preparing for unit testing 15](#_Toc432620842)

[4.7.3 Performing unit testing 15](#_Toc432620843)

[4.7.4 Revision and retesting 15](#_Toc432620844)

[4.7.5 Analyzing and recording unit test results 15](#_Toc432620845)

[4.8 Unit integration and testing 16](#_Toc432620846)

[4.8.1 Preparing for unit integration and testing 16](#_Toc432620847)

[4.8.2 Performing unit integration and testing 16](#_Toc432620848)

[4.8.3 Revision and retesting 16](#_Toc432620849)

[4.8.4 Analyzing and recording unit integration and test results 16](#_Toc432620850)

[4.9 CSCI qualification testing 16](#_Toc432620851)

[4.9.1 Independence in CSCI qualification testing 16](#_Toc432620852)

[4.9.2 Testing on the target computer system 16](#_Toc432620853)

[4.9.3 Preparing for CSCI qualification testing 16](#_Toc432620854)

[4.9.4 Dry run of CSCI qualification testing 17](#_Toc432620855)

[4.9.5 Performing CSCI qualification testing 17](#_Toc432620856)

[4.9.6 Revision and retesting 17](#_Toc432620857)

[4.9.7 Analyzing and recording CSCI qualification test results 17](#_Toc432620858)

[4.10 CSCI/HWCI integration and testing 17](#_Toc432620859)

[4.10.1 Preparing for CSCI/HWCI integration and testing 17](#_Toc432620860)

[4.10.2 Performing CSCI/HWCI integration and testing 17](#_Toc432620861)

[4.10.3 Revision and retesting 17](#_Toc432620862)

[4.10.4 Analyzing and recording CSCI/HWCI integration and test results 17](#_Toc432620863)

[4.11 System qualification testing 18](#_Toc432620864)

[4.11.1 Independence in system qualification testing 18](#_Toc432620865)

[4.11.2 Testing on the target computer system 18](#_Toc432620866)

[4.11.3 Preparing for system qualification testing 18](#_Toc432620867)

[4.11.4 Dry run of system qualification testing 18](#_Toc432620868)

[4.11.5 Performing system qualification testing 18](#_Toc432620869)

[4.11.6 Revision and retesting 18](#_Toc432620870)

[4.11.7 Analyzing and recording system qualification test results 18](#_Toc432620871)

[4.12 Preparing for software use 18](#_Toc432620872)

[4.12.1 Preparing the executable software 18](#_Toc432620873)

[4.12.2 Preparing version descriptions for user sites 18](#_Toc432620874)

[4.12.3 Preparing user manuals 19](#_Toc432620875)

[4.12.4 Installation at user sites 19](#_Toc432620876)

[4.13 Preparing for software transition 19](#_Toc432620877)

[4.14 Software configuration management 19](#_Toc432620885)

[4.14.1 Configuration identification 19](#_Toc432620886)

[4.14.2 Configuration control 19](#_Toc432620887)

[4.14.3 Configuration status accounting 19](#_Toc432620888)

[4.15 Software product evaluation 19](#_Toc432620891)

[4.15.1 In-process and final software product evaluations 19](#_Toc432620892)

[4.15.2 Software product evaluation records, including items to be recorded 19](#_Toc432620893)

[4.15.3 Independence in software product evaluation 19](#_Toc432620894)

[4.16 Software quality assurance 20](#_Toc432620895)

[4.16.1 Joint Reveiws 20](#_Toc432620896)

[5 Project organization and resources 20](#_Toc432620915)

[5.1 Project resources 20](#_Toc432620917)

# Scope

## Identification

This project is to be a web application which when implemented will allow a user to input preferences into a questionnaire and top locations will then be outputted onto a map. When first accessing the webpage, the user will begin with the “login page.” From the login page, a user can access a “create an account page” or after authentication access their “homepage.” The homepage will allow the user to modify their account information from a “modify account page,” as well as allow the user to see the “map” and access the “questionnaire,” which are defined below.

The “questionnaire” refers to the portion of the project where the user inputs these preferences. This will be accessed from the homepage into a “questionnaire page.” The “ratings” will refer to the numerical input from a user in which they determine which statistics have a higher preference. The “statistics” refer to the set of data that our project will be accessing. The “map” refers to the portion of the project which is the visualization of data. The map is initially displayed from the homepage, but will initially be empty, and be modified as the user takes the questionnaire. After the user completes the questionnaire the map is referenced to be an “updated map” with the continuing definitions being components of the map. The map may also have markers which will be referred to as “pins.” The pins will allow the user to see an image of the location referred to as the “pin image.” In some cases where an image is not available, or if there is additional implementation, a pin may allow the user to see descriptive information on the location which will be referred to as the “pin information.”

This system is to be done in the context of the UMBC CMSC 447 Software Engineering course with Professor Cain. The procedures for this project are to mirror that taught in the course found within the blackboard lecture content. From the class lectures, the software development life cycle should follow that of the procedures found in waterfall as well as agile programming. The documentation for this project should follow a waterfall progress having linear progression such that we should complete the documentation in order. Coding should follow a agile progression such that the coding should be continuously occur, and progression on the development can begin immediately with basic known functionality.

## System overview

The goal of this project is to create a web application which will allow a user to create an account, and from such account be able to evaluate different statistical values such that a list of locations within the United States will be outputted to the user which correlate to the user input. With the list, a map should be presented with pins that correlate to the location and can then be further expanded to output a location image or description.

A SQL database will be used for maintaining the user accounts. Along with this we expect to be using third party software for our output. This would include the use of GoogleMaps for our map output, and GooglePlaces for our image outputs, and GooglePlace for a description output. Further third-party software will be expanded in this section as seems fit throughout the project.

Because the nature of this project is in the scope of a classroom project, there will not be long term maintenance of the project and will be run on a local machine. Throughout this document the term sponsor will refer to John Winder who is the group’s client throughout the project. Similarly, the group refers to the group of developers working on the project including Matthew Hearn, Aaron Lewis, Alex Rochford, Cathy Poore, Ben Kittner, and Steven Heckman. The project will refer to the software and documentation created for this assignment. All project development will be done through GitHub and then developer preference for development environments, debuggers, ect. The web portion of the application will be run using an Angular Framework on a localhost.

The main goals discussed with the client are to put an emphasis on the web portion of the application. This specifically includes those relating to login databases and the visual presentation of any found data. There should be less emphasis on the algorithm and backend portions of the application.

## Document overview

The goal of this document is to give an overview of the project, as well as explain the development process of the project group. This will include coding practices including but not limited to file headers, function headers, indentation practices, file names, function names, variable names, development environments, used programming languages. Further explained practices will include design and testing processes.

These practices and to be viewed as fluid and as different concerns are raised between the development group or project client this document may be modified as necessary. For this reason, after approval has been given from the client any modifications to the document will be resubmitted for approval and require permission. In a similar manner, if the client wishes for any modifications these should be expressed either through email or verbally which are then approved upon in the form of meeting minutes from the group. With the goal of transparency in mind any modifications to this document will be explicitly noted and noted in the section below.

**1.3.1 Version Modifications**

Version 1.0: Original Document

This includes the paragraph description for each heading which will then be deleted upon approval. Additionally, any empty section is kept for both client and group approval.

Version 1.1: Updated System Overview

Modified the system overview to correspond with the other documentation to reflect the use of Angular framework and running the application on a local host.

Version 2.0 Updated after group meeting on 4/23.

Updated the Identification to be standard with the rest of the documentation. Updated the system overview with the use of Google Places. Updated system specifics to meet request in the group meeting on 4/23. Other changes include small grammatical changes.

Version 2.1 Minor Grammar Changes.

Version 2.2 Removed paragraph skeletal templates.

This should be the final draft. Table of Contents updated.

Version 2.3 Updated details of section 1.2 and 1.3

## Relationship to other plans

The following are referenced documents throughout this document and include the common acronym for referencing the document:

SRS- Software Requirements Specification

SDD- Software Design Description

STD- Software Test Description

SUM- Software User Manual

# Overview of required work

For a list of requirements, see the SRS. There are currently no constraints on the system. The major concept behind this system is to implement a web application. Within this application there should be a require account creation and login process. From here a user can access the homepage of the application. The homepage should interact with the GoogleMaps API to present the continental United States. From here the user should be able to take a questionnaire which outputs counties that match user rated statistics. This should populate the map on the homepage with pins which the user can click to see an image and description of the county. There are currently no requirements related to the maintence of documentation throughout this system development. However, with the goal of transparency in mind, the document overview of each document will include a description of versions for each document. The following are updates, in relationship to the project system life cycle, with reference to the defined version numbers.

As of version 1.0 we are beginning the development life cycle. We are looking into different means of implementation of data visualization, as well as researching data for the project. We have also begun implementing the user account creation and login portion of the project. In terms of documentation we have completed the formation of project requirements and will begin the creation of the project design found in the SRS.

As of version 1.1 we have implemented the basics of the homepage and are beginning to implement the SQL server portion of the project. Also, we have begun the development of the survey/questionnaire portion of the project and will be using data from the Census Bureau. Upon completion of version 1.1 we have completed the SDD.

As of version 2.0 we have a layout for the user account portion of the system. We need to complete the implementation of user accounts with a user avatar. We also need to add the questionnaire portion of the system to the homepage. Finally, we need to implement the location pins. Upon completion of version 2.0 we have completed the STD.

The only defined guideline for dates is the final project presentation which is to be done May 10th. Recommended guidelines for documentation dates were given by Professor Cain. These are expected to be followed by the development team. In addition, all documents will be sent to the client for review as drafts are completed. Final signatures will then be obtained by the development team, from the client during the week of May 10th. There are no security, privacy, standards, or hardware/software interdependence requirements or constraints. There are no program/acquisition strategy or requirement constraints.

# Plans for performing general software development activities

## Software development process

This project is to be viewed as using a mixture between waterfall and agile development processes. The waterfall portion of the project will best correspond to the documentation portions which are to be completed in a systematic order. However, each document is to be viewed as fluid and modified when necessary. Upon modification after client approval, the client will be notified. All documents will receive a final signature upon the week of the project presentations which is the week of May 10th.

The software coding portions of the project are to be viewed with an agile development process such that code should be developed in a manner that is flexible and compartmentalized allowing each software developer the flexibility of coding the portions where they are most experienced. Because of the software development flexibility, the documentation portions of the project will need to be updated as new tools or methods are included in the software.

## General plans for software development

### Software development methods

There is great flexibility in the software development methods to be used by the group. As such, this portion will be expanded as new software tools and methods are used. Currently, we expect to be using a SQL database for holding user login data, and python for the back-end data analytics and file management. All code and documentation are to be committed and pushed to the GitHub master repository. With the continued goal of transparency any scripts and similar assisting projects are also be included within the GitHub repository for the use of other group members, as well as for viewing by the client.

### Standards for software products

This section deals with the standard of code formatting. It is to be noted that the goal of readability is to be of greater importance than holding to these standards. If there is deviation from the standards a note within the comments should be given with explanation if necessary. In general, all indentation should be done with by using standard 8-character tabs.

In general, single line spacing is to be used for smaller sections of code. If a code portion is more than 5 lines, then there should be a line separating the beginning and end of the segment. A segment is defined as a portion of code that manipulates the same data. This would be typically occurring during a function definition and loop structures. Again, developer judgement should be used for evaluating if deviating from the defined standard would create more readable code.

Header comments should be included at the beginning of any software file, as well as prior to any new function or method. File headers should include the name of the file, the original creator of the file, the date the file was last modified, as well as a brief description of the functionality within the file. This should include a statement of the webpage being created, the functions being defined, or any other aspects that would convey the purpose of the file.

Function headers should include the name of the function developer (if different than the file creator), requirements that the function intends to be satisfying, test associated with the function, inputs to the function, expected outputs of the function, any error handling being done by the function, and a brief description of the function if not satisfied by the inputs and outputs.

Furthermore, if code is being submitted to GitHub with known bugs, the bugs should be described as the ending of the file or function header. Each header must also include a section for another team member to initial, which will serve to represent a code review.

Because all code is to include file and function headers with the name of the code creator, when modifying another group member’s code, a comment should be included either on the line of the modification, or for larger segments of code at the beginning and ending of the code segments with the modifiers’ initials. Each segment of code is expected to have a comment explaining what is occurring within the code segment. Furthermore, if a portion of code is more complex a comment should be included. This could include a URL for referenced website, or an explanation for what is being done.

All naming for files, function, and variables should follow the following standards. A filename should be in the format of “NameOfDocument.ext” such that each word begins with a capitalized letter, including the first. Filenames should allow a not working individual to assume the role of that file from the filename. A function name should take a format of “functionName” such that the first letter is always lowercase, and all other words of the statement begin with a capital letter. Like other names, the function name should demonstrate the purpose of the function. A function name that cannot follow this should most likely be broken into multiple functions for readability. Finally, variable names should be the format of “variable\_name” such that everything is lowercase, and words are separated using an underscore. Variable names should be relevant to the scope of the function they are being implemented within and should thus be different than the name of a global or larger scope variable being passed into a function. For example, if an array is being passed into a modification function, then the global name “array” should not be used within the function, and instead use a name such as “arr” within the function scope. This intends to avoid naming errors, as well as maintain readability and obvious function scope differences within functions. Like the other portions, if a different naming scheme allows for better readability a note within the comments should be provided.

Currently there are no constraints on programming language nor restrictions on code aggregates.

### Reusable software products

#### Incorporating reusable software products

When reusing code is the responsibility of the developer to evaluate the usefulness of the code. This would include any legal restrictions on its usage, as well as evaluating the functionality of the code. Due to the nature of this being a educational application, all reusable code being incorporated into the system should be free for usage. Whenever the developer has a question of the usability of reusable code it is the expectation for the developer to contact the development team and client for any questions regarding ethical behavior in its usage. General practice of the development team should be to remain cautious of using out of house software, and only when confirmed should the code be implemented within the system. When evaluating code it should be expected to search for any published bugs, and evaluate whether bugs affect the implementation within the developing system.

#### Developing reusable software products

Currently there are no requirements for creating reusable software products. However, as a best practice, all code should be written in an easily reusable manner. This would be most applicable to practices of file, function, and variable naming. Similarly, functionality should be expanded such that each function only accomplishes a single task, such that each function could be individually reused. Finally, it is expected that functions do not manipulate an original data set, and instead work on a copy of the data set such that if a process were to be reused there would not be unintentional data manipulation. When this practice cannot be met, for example with file I/O, a note should be made within the comments simply stating that there is access to the original data set.

### Handling of critical requirements

#### Safety assurance

There are currently no safety requirements associated with this project. However, if safety requirements are added, they should be evaluated with in the highest requirement priority. Due to the nature of the system, a pure web application, we do not expect physical safety to be an issue.

#### Security assurance

There are currently no security requirements associated with this project. However, if security requirements are added, they should be evaluated with the highest requirement priority. The system being designed is to include a login portion which requires interaction with a SQL server. Best practices should be made to limit the potential vulnerabilities within the system. Any developer who has questions regarding preventing SQL vulnerabilities should express their questions to the development team for advice, and any potential vulnerabilities should be expressed to the client and documented.

#### Privacy assurance

There are currently no privacy requirements associated with this project. However, if privacy requirements are added, they should be evaluated with the highest requirement priority. Due to the nature of the system being run locally, we do not expect privacy to be an issue.

#### Assurance of other critical requirements

There are no other critical assurance requirements. For a list of other requirements see the SRS.

### Computer hardware resource utilization

There are currently no hardware resources that would need to be monitored for the use of the system due to the nature of the system being a local web-based application.

### Recording rationale

There are currently no explicit measures in place for recording rationale. It is expected that as occurrences arise they be documented with email correspondence, and when verbally expressed during a meeting, meeting minutes should be sent out within 48 hours of the meeting and reviewed by all group members and the client within the next 48 hours. If this cannot be done a simple response noting that the minutes were received with an expected timeframe of when the minutes will be read should be stated. This should be sent within the group within the scope of the group members.

Key decisions will be any decision, specifically within the realm of design, which modifies the functionality of a pre-built CSCI component. CSCI components for the system will be described within the SDD. If a CSCI component has been completed, and for integration testing it is found to need modification, then the development team is expected to review the necessary modifications. If these require changes to requirements or design, the modifications should be documented and summarized in the documentation version summary. This should then be sent to the client immediately.

### Access for acquirer review

The client should be granted read-only access to the group’s GitHub repository with the goal of being able to review any code. Any notes on the code should be sent to the group in the form of an email message or be verbally expressed in the next group meeting which should then follow the normal meeting minute procedure. Due to the nature of this project being an academic project, there would be no need to evaluate physical facilities. There are no requirements related to acquirer review.

# Plans for performing detailed software development activities

## Project planning and oversight

### Software development planning (covering updates to this plan)

To plan for development a completion of SDD will be required, as well as review from the development team. This should include descriptions of the database setup, code that will be reused, as well as any continued research that needs to be done. All system decisions should be discussed within the team, and then sent to the client for approval. Before implementing new system, the development team should meet with client for a design review.

To prepare for development, the development team should research reusable code or any tools which would ease in the development of the system. This should then be evaluated by the development team to determine usefulness. Once approval has been given by the development team, then the tools shall be documented within the SDD. If development has begun, and a new tool is required, this will reflect a new subversion within the SDD.

### CSCI test planning

CSCI design documentation should be found within the SDD. This should include an outline of what the different CSCI components will be and basic descriptions of how they will be developed. CSCI decisions should be made by the developer(s) working on them and then shared with the development team for approval. This will then be sent to the client through the documentation. The CSCI components should also be designed with the testing portion of the code in mind which can be found in the STD.

To prepare for testing, the development team shall write and preform unit test as portions of code are developed. These tests should also output a visual Pass/Fail to the developer, which should be stored as a Boolean True/False value within the code. A final component test should check that all tests returned a value of True, with a final Pass/Fail visual. Upon completion of the CSCI component, the unit test print tests should be included, however the visual outputs should be commented, such that only the final Pass/Fail is presented. This is to prepare for system testing.

### System test planning

Details of the testing of the system can be found within the STD. All CSCI components should be designed with the goal of unit and integration testing in mind. Upon completion of all CSCI components then system testing should be done. All system test should be done as a cross reference to the requirements described in the SRS. Basic of preparing for the system test are described above within the CSCI component testing. Assuming a CSCI component has passed all unit and integration testing. Then there should be a Pass/Fail visual output for each CSCI component. The system tests should be the check that the final CSCI tests all returned True.

### Software installation planning

This project is not intended to be installed on a device outside of the development group. However, to ease group development and testing of the system, there should be README files throughout each the project describing any setup to run a piece of code. These files should be included within their respective directories, with the inclusion of a system-wide README in the home directory. Software installation should be described within the STD within instructions on what software is needed, and instructions for the installation.

### Software transition planning

This system will not need any transition planning since the system is entirely software.

### Following and updating plans, including the intervals for management review

Periodic reviews should be done with the development team and the client. The development team is expected to meet once a week to discuss current completion of the system as well as any development issues. Furthermore, the development team should include progress updates to the client. These should include notes of any completed CSCI components or documents. Finally, in person meetings should occur at least three times through the entirety of the project including at least one meeting to discuss requirement, one meeting as a prototype test, and one meeting for an acceptance review and document signing. Further meetings are encouraged and required for within the development team.

## Establishing a software development environment

### Software engineering environment

Currently there are no requirements focused on the development environments and usable libraries. The development team has chosen to use the Angular framework for the web development. Also, an Oracle MYSQL server will be used for holding the database. Each developer may use the development environment that they choose, assuming the code works within the testing environment.

### Software test environment

The requirements for this system state that the web application will be run on a local host. For this reason, a README file should be included for what software is needed to setup the environment. This should be included within the system’s home directory. Furthermore, all automation test should be included within the same test file, so that all test may be run in a single automated test.

### Software development library

All files should be included within the GitHub master directory. This will serve as the location to store all code and documentation. The home directory should be broken down into categories of Census\_Data, Code, Documentation, Meeting\_Minutes, and Programs. There should also be the Angular README file, as well as a SETUP READE file. Code for the actual applications should be found within the backend and frontend folder of the Code directory. The Programs directory should only contain executable files for the installation process of the system. The Documentation directory should contain the original templates, as well as each modified version of each document. The Meeting\_Minutes directory should include a word document describing the meeting available for modification, as well as a pdf version which is to be sent from the development team to the client. The Census\_Data directory should include all data obtained from the Census Bureau database, as well as any code which was used to modify the data.

### Software development files

All developed files should be included within the GitHub master directory, and furthermore included in there appropriate subdirectory. The frontend directory should include all code, documentation, and README files corresponding to the web application of the system. The backend directory should include all code, documentation, and README files corresponding to the data analysis portion of the application. The documentation directory should include any files which will be included in the final deliverable binder to be signed by the client. The meeting minutes directory should include any files documenting meeting that took place and should be sent to the client upon there completion.

### Non-deliverable software

All software will be delivered with the final project. These files will be within their appropriate folder described within section 4.2.3.

## System requirements analysis

### Analysis of User Input

User input should be mostly evaluated within the user login portion of the code and the questionnaire portion of the system. This will include validation of appropriate username and password usage within the login portion of the system. The questionnaire portion of the system will need to validate that the user has entered appropriate input so that any backend portions of the system may perform their appropriate calculations. This is further expanded in the SRS.

User input will also be evaluated during the questionnaire portion of the system. The user rating should be evaluated as integers which will then be sent to the backend portion of the application.

### Operational Concept

Currently there are no requirements associated with the operational concept outside of the project being a web application.

### System Requirements

System requirements will be found within the SRS as well as the Requirements Traceability section.

## System design

### System-wide design decisions

There are currently no requirements or contractual clauses related to system-wide design decisions. This has been left to the development team to do as they choose fit. Current system wide design system is primarily related to the use of a database for the user login portion of the system. The user account is shared through multiple CSCI components and will need to be accessible throughout the system. Further information related to the system wide design can be found within the SDD.

### System architectural design

There are currently no requirements or contractual clauses related to system architectural design. Each CSCI component will have a description found within the SDD describing the components implementation. The design decisions for each CSCI component are left to the developer(s) of that component, which will then be approved by the development team.

## Software requirements analysis

The software requirements may be found within the SRS. All requirements will be reviewed by the development team to ensure feasibility and then the SRS will be sent to the client for approval. If the client does not provide explicit requirements that the development team will create the set of requirements for the system prior to writing the SRS and will seek approval for the requirement list. These will then serve as the set of requirements for the SRS.

## Software design

### CSCI-wide design decisions

CSCI wide decisions will be discussed by the development team prior to implementation. After the development team has decided, then these decisions will be documented within the SDD which will be sent to the client for review.

### CSCI architectural design

CSCI architectural decisions will be discussed by the developer(s) of the CSCI component prior to implementation. These decisions will be delivered to the development team, and if approval is given implementation of the component may begin. These decisions will be documented within the SDD which will be sent to the client for review.

### CSCI detailed design

CSCI detailed decisions will be determined by the developer(s) of the CSCI component and documented within the SDD if the decision affects any architectural system. Documentation of details within a CSCI component should be documented within the comments of the code. If modifications are made to the SDD this will be sent to the client for review. If the documentation is limited to the comments, then review will be done during code review within the development team.

## Software implementation and unit testing

### Software implementation

All software is intended to be developed in a manner that will allow for unit testing, and as such each function should be limited, and complete a single testable task. Furthermore, a unit test should be written for each developed function. Unit test should be automated.

### Preparing for unit testing

Following the above procedure, functions should be easily testable. Similarly, each developer is expected to create their own unit test as they develop code segments. These tests should be included within a test file in their respective directory. The test should be callable from a single command line argument found within the directories README file.

### Performing unit testing

Upon completion of each code segment it is expected that the portion of code is to be unit tested. The unit test should be included within a test file found in each directory. Using the command line arguments found within the README, the test should be performed. Upon successful unit tests, the command to run the directories unit test should be included in above directories tests. This should continue up until the system unit test file is reached.

### Revision and retesting

When a code segment is modified, unit testing shall be performed again. Similarly, if there is failed integration testing, a test for the failure should be developed. If a test is failed, this should be shared with the development group.

### Analyzing and recording unit test results

All unit testing should present some type of visualization for the developer demonstrating that the code portion has “passed” the unit test. As the code segment becomes more well developed then a single output can be used for the accumulation of unit tests for an extended portion of code.

## Unit integration and testing

### Preparing for unit integration and testing

All code should be developed within the context of what they will be integrated into, and as though of a portion of the entire system. Integration testing serves to test the macrocode portions. Ignoration testing should be done if all unit tests for the code segment are passed. Furthermore, it is expected that all code units will then be a part of the CSCI and should be developed within the CSCI context.

### Performing unit integration and testing

Integration testing should begin once all the components of a code segment pass the individual unit test. The integration test should test all possible “routes” of the code and would be most associated with the loop structures of the code. In the context of a web application the integration testing should check the links between pages as well.

### Revision and retesting

Integration testing should check that previous unit tests are not being failed, as well as test the overall system. Failed unit tests should be documented with the developer of the failed portion. If a unit test for a different system is being broken by a different CSCI component, then the new component should be inspected. In the case of a poor unit test for the previously developed component, the development team will collaborate on which component should be redesigned.

### Analyzing and recording unit integration and test results

All integration testing should present some type of visualization for the developer demonstrating that the code portion has “passed” the integration test. As the code segment becomes more well developed then a single output can be used for the accumulation of integration tests for an extended portion of code

## CSCI qualification testing

### Independence in CSCI qualification testing

CSCI testing is intended to be primarily done by the development team in a manner that states overall success or failure of the CSCI component. Successful CSCI testing would be able to be publicly released and used in a manner that would demonstrate new usability within the web application. For the release, acceptance testing with the client would be required.

### Testing on the target computer system

Since the application will be locally run, all tests will be done in a local web environment. There is no “target computer” for this system.

### Preparing for CSCI qualification testing

CSCI qualification testing should begin to be approached in integration testing and should be the overall goal of the software component. Successful tests should test all possible code paths, as well as handle user input errors.

### Dry run of CSCI qualification testing

All CSCI qualification testing is expected to be done by the software development group prior to testing with the client. These test results should be known prior to product demonstrations with the intention of limiting portions of the code that are not currently implemented.

### Performing CSCI qualification testing

CSCI qualification testing should be done upon completion of an individual CSCI component. Upon a successful dry-run, a meeting should be planned with the client to demonstrate the functionality which should then receive approval or disapproval. Completion of a CSCI allows for the next development cycle for creating new functionality within the application.

### Revision and retesting

Upon the development of a new CSCI component, the previous test should be checked to ensure that previous tests do not fail. This would similarly be tested within the dry-run as well as client demonstrations.

### Analyzing and recording CSCI qualification test results

Each CSCI should be associated with at least one requirement on the SRS. The specific requirement to CSCI association will be recorded in the requirements table of the SDD document.

## CSCI/HWCI integration and testing

### Preparing for CSCI/HWCI integration and testing

CSCI integration testing should be done upon completion of the unit testing of a CSCI component. Integration testing should include the unit test of the other CSCI components. To do this each CSCI component should include a single automated test file. Instructions for how to run the file from the command line should be included in a separate README file.

### Performing CSCI/HWCI integration and testing

CSCI integration testing should require simply calling the overall systems test script. This should be found within the home directory of the system. As CSCI components complete their CSCI testing, then the system test script should be updated with what script to call to test the CSCI component. Overall system integration test should be run from the home directory.

### Revision and retesting

Upon failure of a CSCI component or system test within the CSCI integration test, there should be a review on failing CSCI component. The development group will then collaborate on how to redesign the CSCI component.

### Analyzing and recording CSCI/HWCI integration and test results

Assuming the test for the smaller component of the system have been correctly added to the CSCI integration test, then the CSCI integration results should just be pass/or fail print statements. The integration test should simply be the done via checking these print statements.

## System qualification testing

### Independence in system qualification testing

Upon completion of each CSCI component, previous testing should be done to ensure the overall system is not being modified. To ensure independence, all requirements will be written in a format for ease in evaluation. Meeting the given requirements would be equivalent to passing the system qualification testing.

### Testing on the target computer system

This system will run on a local environment so there is no further target computer system outside that of the development team.

### Preparing for system qualification testing

The system test should be formatted such that each test meets a specific requirement listed within the SRS. A successful system qualification test would represent a successful system. This would include meeting all requirements.

### Dry run of system qualification testing

All system tests are expected to be run by the group prior to a client meeting with documentation on what requirements are currently being met and which requirements currently need to be met.

### Performing system qualification testing

System qualification tests are expected to be done with the client present, and upon successful tests, the final signatures should be completed on the SRS documentation.

### Revision and retesting

If a portion of the system is modified after system testing, then the SRS document should be modified with the new testing date as well as a description of what modifications are being made to the system.

### Analyzing and recording system qualification test results

All system tests should be recorded within the STR. This should include a description of any failures, and modification which will be necessary for the system.

## Preparing for software use

### Preparing the executable software

All software will be locally run and therefore executable software will be locally owned and executed. There should be no need to prepare executable software outside of the CSCI and system testing.

### Preparing version descriptions for user sites

Version descriptions will be maintained within the SUM. System version number will be kept in a format of system version primary(#).secondary(#). A new primary index will indicate a new CSCI component. A secondary index will indicate a modification to a CSCI component.

### Preparing user manuals

The user manual will be maintained in the SUM. The SUM should be made using the README files which should be documented as the development team use new outside software. This will include descriptions for how to setup the environment and a dependency list.

### Installation at user sites

There should be no need to install at a user site since the project will be run from the development team’s computers. An installation guide can be found within the STD as well as the SETUP Readme found in the home directory of the GitHub repository.

## Software configuration management

Each version should be associated with completion of a CSCI element. Due to the scope of the project, configuration control should not be needed.

## Software product evaluation

### In-process and final software product evaluations

In-process product evaluation should be done upon completion of each CSCI component and receive a signature within the SRS upon successful demonstration. The final software product is to be completed on 5/10 within the classroom environment. All tests should be done against the requirements laid out in the SRS.

### Software product evaluation records, including items to be recorded

Product evaluation should be completed with CSCI and system testing, and any modification or bugs should be reported and documented within the context of each test.

### Independence in software product evaluation

The software product is to be evaluated for approval by the client to avoid development group influence.

## Software quality assurance

### Software quality assurance evaluations

Software quality assurance should be evaluated with each CSCI and system demonstration.

### Software quality assurance records, including items to be recorded

To maintain software quality the group intends to use practices of code review as well as the use of unit, integration, CSCI, and system testing.

### Independence in software quality assurance

Quality assurance will be independent using code review, and then final approval from the client.

## Joint technical and management reviews

### Joint management reviews, including a proposed set of reviews

The scope of technical reviews will be simple initials from code reviews. Each unit should be independently reviewed by another group member which will be kept in the header section of individual units. There will be no management reviews for this project since there is no management. These signatures will be obtained during the last week of the progress from the project client. The development team will be expected to outreach to the client during this time period.

# Project organization and resources

## Project resources

For this project our group will have 6 members. Each member is expected to spend approximately 5-8 hours per a week on the project throughout the school semester. This time may need to be flexible depending on due dates and request by the client. Each member is expected to participate in the documentation, software development, unit testing, integration testing, system testing, and code review processes. Furthermore, we expect everyone to maintain a specific role within the project. These roles should include document lead, person of content, frontend developer, backend developer, tester, and a flex person. We are all college students at UMBC with no need for security clearances for this project. Each member is expected to perform all portions of the project. For transparency, each document will include the name of the developer for that code segment. Development is expected to be done by the group where they please which could include but is not limited to the UMBC campus, personal living spaces, or any public location.