

COS 497: Capstone II



Blue Marble Geographics
Geospatial Data Portal
Code Inspection Report

Team Undershrub
Anthony Jackson - Client Liaison
David Sincyr III - Team Leader
Devin Carter - Support Manager
Grant Shotwell - Developer
Steven Kaplan - Development Manager

Jan. 31st 2022

Geospatial Data Portal
Code Inspection Report

Table of Contents

	<u>Page</u>
1. Introduction	3
1.1 Purpose of This Document	3
1.2 References	3
1.3 Coding and Commenting Conventions	3
1.4 Defect Checklist	5
2. Code Inspection Process	7
2.1 Introduction	7
2.2 Description	7
2.3 Impressions of the Process	8
2.4 Inspection Meetings.	9
3. Modules Inspected	10
3.1 Introduction	10
4. Defects	14
4.1 Introduction	14
Appendix A – Client and Team Review Sign-off	17
Appendix B – Document Contributions	19

Document History

Name	Date	Reason for Changes	Version
David Sincyr	1/31/2022	Initial Creation	1.0
All team members	3/9/2022	Edited whole document	2.0

1. Introduction

Blue Marble Geographics® is a software development company specializing in geographic information systems and geodetic desktop software. Their software is aimed towards all types of users who need a better tool to manipulate and visualize their geospatial data. To better serve their customers, a cloud-based structure is necessary. The dynamic web portal will allow its users to upload their geospatial data to an online database and provide them the ability to share, manipulate, and visualize their data with others. Additionally, users will be able to automate the visualization process using Python scripts through a built-in Python script editor. This is a capstone project for Blue Marble Geographics®, in partial fulfillment of the Computer Science BS degree for the University of Maine.

1.1 Purpose of This Document

This document aims to find defects, identify process improvements, and create corrective actions for all source code. Additionally, it is also to create a protocol for coding and commenting practices, so that source code is formatted correctly and is similar regardless of the developer. The intended audience for this document is the development team, Dr. Terry Yoo, and all stakeholders within Blue Marble Geographics®.

1.2 References

1. Team Undershrub User Interface Design Document
2. Code Inspection Report Template (Adapted from Susan Mitchell)
- 3.

1.3 Coding and Commenting Conventions

Commenting Conventions.

The commenting conventions for this project will be split between “*mainline*” conventions which are followed regardless of the file-type, and “*outlier*” conventions which will only be present in specific file types because of their structural design and differences. There are five current mainline conventions; comments need to have a space between the commenting call and the first word of the comment for organizational purposes, the first word within a comment needs to be capitalized as to be inline with ESLint and Black’s standards, functions need to have an in depth comment (2-3 sentences) above as to explain their purpose, and comments always need to be above the content that they are commenting on.

There are seven outlier conventions present within the project, each referring to a specific category of files that have differing ways to call a comment statement.

- Files that end with **.js** and **.py** need to use the `"/* Text */"` commenting style for larger command blocks over functions to explain their function and purpose. While the simpler `"// Text"` method is to be used for singular comments which do not exceed 79 characters.
- Files that end with **.md** need to use the `"** Text **"` commenting style for larger command blocks over functions to explain their function and purpose. While the simpler `"# Text"` method is to be used for singular comments which do not exceed 80 characters.
- Files that end in **.html** need to use `"<!-- Text -->"` for any and all commenting.
- Files that end in **.ps1** need to use `<# Text #>` for any and all commenting.
- Files that end in **.sh** or **.txt** need to use `"# Text"` for any and all commenting, separating sentences when appropriate following the mainline conventions.
- Files that end in **.json** or **.yaml** need to use `"// Text"` for any and all commenting, separating sentences when appropriate following the mainline conventions.
- Files that end in **.bat** need to use `"rem Text"` for any and all commenting, separating sentences when appropriate following the mainline conventions.
- Files that end in **.css** need to use `"/* Text */"` for any and all commenting, separating sentences when appropriate following the mainline conventions.

Coding Conventions

The main convention that will be used is "upper camel casing" without underscore separation for higher level items such as class names, however the low level items such as variables or functions will be using "lower camel casing" with underscore separations instead. None of the source code lines will go over a maximum of 80 characters in length, else it will need to be broken up in an appropriate place such as a space, comma or another non-digit or non-letter symbol. White space will be used in said code to separate different classes, functions or sections within a function as to thereby separate important aspects or utility within a function.

The code will then be run through a linting process in order to enforce the various coding and development standards, as well as to detect common mistakes or unused variables. This process will be automated for both the front-end and back-end of the application through the use of Github Actions continuous integration features. The static analysis tool "ESLint" will be used for the front-ends ReactJS elements, while the back-end's Django elements will use the python code formatter known as "Black".

1.4 Defect Checklist

Possible defects will be cataloged in tabular format with five categories for easy navigation and indexing. This cataloging begins with the “defect number,” which is the chronicling index of issues found within the software. The defect number is then followed by the “defect description,” which explains how the defect occurs, whether this is through coding conventions, logical errors, security oversights, or code commenting. The following column, “defect category,” then details the specific issue under its exact category as listed above, only ever being documented as its overarching issue. The “severity” section then describes whether the issue is minor, major, or critical. Minor issues are considered simple without an overarching effect on the project; major issues can affect the functionality or present themselves in specific circumstances, while critical issues can massively affect the project in unsafe ways or will render it non-functioning for a user. The last category, known as “remarks,” is for any comments made by the developers and team members of the project. Comments will range from descriptions on how to fix the issue, further discussions of the danger of the issue, or an explanation on why it occurs.

Defect #	Defect Description	Defect Category	Severity	Remarks
1	Code segment does not have any comments.	Commenting	Minor	Frontend components will use additional comments to explain how different elements work.
2	Comments unclear.	Commenting	Minor	Some comments are limited, and brief, descriptions of what they're meant to describe. Lengthier comments are needed to ensure anyone maintaining the code understands the code.
3	Using wrong operators.	Logic Error	Major	This includes a handful of “grammar-esc” operation issues such as using “!=” rather than “!==” on JS files.
4	Proper camel casing is not used.	Coding Conventions	Minor	Industry standard for good code readability, not used in all places of the project. We need to make a pass and handle the situations where camel casing is not used.
5	White space is not used properly.	Coding Conventions	Minor	Some whitespace should be used to break up functions and comments, but anything over three lines can be excessive. Additionally whitespace use should be consistent across all written code.

6	lines of code go over the 80 character limit.	Coding Conventions	Minor	Complex lines of code should be split up. Splitting up code offers visual clarity, and the ability to explain individual segments of code.
7	User input is not sanitized.	Security Oversights	Critical	User input needs to be sanitized to reduce cybersecurity risk
8	Unused variables present.	Coding Conventions	Minor	Older unused variables from previous tests or functionality are present, these need to be removed at earliest convenience to the developer.
9	Unused imports present.	Coding Conventions	Minor	Some imports are called by left un-used as developers bring functions from other files when making similar ones of the same type.
10	Code is being repeated.	Coding Conventions	Major	Each group member has worked on coding tasks either solo or in pairs. The solutions to minor problems may be shared between tasks, but different individuals/pairs may end up having to solve the same problem with limited help.
11	CSS files are not named the same as its component.	Coding Conventions	Major	Readability and understandability is affected by not naming them uniformly.
12	console.log() code exists.	Coding Conventions	Minor	This will push unneeded information to the user. Can also be a security risk.
13	Error codes are displayed to a user.	Security Oversights	Critical	Error codes or over descriptive error text increase the risk of a cyber attack.
14	Commented out lines of code are present.	Coding Conventions	Minor	Code lines that are not used are a waste of resources and make the source code look unclean.
15	Protected pages do not check for authentication.	Security Oversights	Critical	Protected pages, such as a user's profile, must be only viewable by the user to which it belongs to.
16	Links can be clicked as if the user was editing the text.	Logic Error	Minor	This defect detracts from the user experience and should be avoided.
17	Functions do not follow the same naming scheme across files.	Coding Conventions	Minor	This defect affects readability thereby lessening understandability.

2. Code Inspection Process

2.1 Introduction

This section will detail various aspects of the group's code inspection process. The aspects will include a detailed description of the chosen code inspection process, why it was chosen, the effectiveness of the process, an explanation of issues with the process, and a list of meetings that the group took part in during this process.

2.2 Description

The first component of our code inspection process is our use of pair programming. Going into this project, we all had our own differing skill sets that would be useful in developing the project and its documentation. That said, there were still gaps in knowledge. An example of this: the majority of the group lacked experience developing code in React. For this reason a multitude of different tasks were worked on together over Discord voice calls, that way group members could assist each other as well as learn from each other. This is not to say we pair programmed every single task and module, but some significant hurdles were overcome via the use of this style of cooperation between group members. Needless to say, the development of the project would not have been the same without the communication that was used in different instances. Some examples of pair programmed modules/tasks included the frontend login page, the frontend project list view, the frontend user settings page as well as the backend for login/signup.

For our automated testing we made use of linting; which was incredibly useful in reviewing our code for stylistic errors, which was something that was originally difficult to do with the limited timeframe of the development process. ESLint and stylelint are two specific Github actions (for Linting) that we have used (and are still using) to analyze our javascript and CSS code (respectively). It should be noted that it's because we chose to use standard style and coding rules that this automated analysis works for our project. In addition to the automated method for analyzing code, we had a limited number of manual code inspections. Due to numerous issues that have come up over the semester, our code inspections are fewer than what would perhaps be ideal. Nonetheless, we have had inspections of code on the frontend and backend. That said, none of these have included the full team (as can be seen on the document) due to time constraints. Pair programming, and general inter-group cooperation, has been our crutch for moving past this constraint.

As a final note for our inspection process, we directly utilized our prior documentation to guide development and analysis of the different elements of our

project. The frontend, in particular, was an important place to keep an attentive watch and direct approach. While we can automate testing the style of our code, as well as the correctness of different functions, the visual elements of our frontend need direct comparisons to our documented material. While there is potential to automate this process, especially when checking for the appropriate use of Blue Marble Geographic colors or fonts, we lacked the experience and time to develop quick and easy automated methods for such testing. It should be noted that not all frontend visualizations were viewed synchronously. Sharing images over Discord, or simply sharing with individuals one at a time over a Sprint, occurred when asynchronous communication was the best (if only) immediate option.

2.3 Impressions of the Process

Pair programming was of immense use at different stages of the development tasks. For a project like this one, where the majority of group members are picking up new skills and knowledge, there's immense utility in being able to share what's being learned through active application. The simple act of having to explain something to a peer develops a deeper understanding for both individuals. When addressing how to solve a problem, two different mindsets can also be of extreme use, allowing for the sharing of different methods and solutions. The simple act of coding, and problem solving as a duo is also particularly motivating. After having engaged in pair programming, the use cases are far more apparent and understandable.

To summarize impressions on automated inspection: the immense utility of automating an inspection process becomes far clearer under an applied use case. One of the great weaknesses, and general constraints, for this project has been time. While the majority of documentation work met each set deadline, the amount of completed code does not meet previous expectations. Time constraints could have similarly been of great impact to our inspection process, if not for the level of automation that was utilized. Especially when it comes to the analysis of code and comment styling, automated tools are an incredibly efficient way of handling analysis. When using standard style and coding rules, tools like Lint provide the exact amount of precision and clear cut responses needed for analysis. The immediate feedback is also useful for allowing group members to fix code and style errors, without needing a formal review involving the majority of group members. Genuinely, it cannot be stressed enough how useful a tool like Lint can be when your greatest constraint is time.

Some of our best work occurred while we utilized a combined effort on the project. Examples of this were the login, logout, and sign up. Login, logout, and sign up all featured immense communication in terms of developing the backend and frontend in respect to each other. The frontend additionally needed to be such that it was easy for others to understand, and with the visualization element showcased

to the whole group for analyzing to make sure we followed our documentation. The cooperation utilized for these modules showcases what we would have wanted to do for every single module, if time was more permitting to work in such a way.

2.4 Inspection Meetings.

Date	Location	Start Time	End Time	Attending Individuals	Role	Code Unit Covered
13th of February in 2022	Virtual - Discord	10:00	10:45	Stephan, David, Anthony, Grant, Devin	Author - David Inspector - Stephan	REST API, base React Components
17th of February in 2022.	Virtual - Discord	8:00	10:00	Stephen, David	Author - David Inspector- Stephen	Header. Footer. NavBar. Login Page.
21st of February in 2022	Virtual - Discord	8:00	11:00	Anthony, Stephen, Devin, David	Author - Anthony, Devin Inspector - Stephen	Login Page Project List.
1st of March in 2022	Virtual - Discord	8:30	9:00	Stephen, David	Author - Stephen Inspector - David	File Upload Send data to an S3 bucket.
4th of March in 2022	Virtual - Discord	9:30	9:45	Stephen, Grant	Author - Grant Inspector - Stephan	Jest unit tests

3. Modules Inspected

3.1 Introduction

This section describes all of the modules that are or need to be inspected within the software. A list of uninspected modules will be supplied at the end of this section that details the module name and a description of its functionality. For all inspected modules, the name, functionality, and a description of where it fits into the design and architecture in the SDD will be detailed. If there are any differences between the modules current design and the proposed design within the SDD, then a comparison between the two will be detailed and the reasoning behind the change will be explained.

Incomplete Modules:

Module Name	Functionality	Expected Date of Completion
Script_Editing	This module would allow a user to edit an uploaded script on the website, modifying and changing their self-made functions.	3 - 16 - 22
File_Search	This module would allow a user to query the system to search for a particular file in their database by its name.	3 - 21 - 22
Remove_File	This module would allow a user to query the system and request that the system remove a specific file from their database.	3 - 21 - 22
Script_List	This module would allow the user to view all of their uploaded scripts in a database-esc format.	3 - 11 - 22
Parse_Metadata	This module would automatically pull metadata from a user's uploaded file.	3 - 21 - 22
Download_File	This module would allow a user to download a previously uploaded file from their database to their system.	3 - 21 - 22
Save_metadata	This module will allow a user to be able to search for existing data within the database.	3-21-22
User_groups	This module will allow users to create groups so that they can work on the same project together	3-21-22
Upload_Form	This module creates a space in which users can take files off their desktop and upload them into the systems file database.	3 - 11 - 22
Associate_User	This module associates (links) a user's created profile with a folder within an S3 Bucket.	3 - 11 - 22
User_Settings	This module is a page within the project which allows a user to modify key settings such as their account information.	3 - 11 - 22

Inspected Modules.

Module Name	Functionality	Description	Changes
Footer_Component	This component exists on all pages. It will display the "Blue Marble Geographics" logo. The BMG logo can be interacted with to bring the user to the about page.	This is a component within the UIDD (2.2.7) and a part of the frontend architecture.	No changes.
Reusable_Button	This component allows for button consistency and code reuse	This is a component within the UIDD (2.2.6) and a part of the frontend	No changes.

		architecture.	
React_Navigation	This component allows for a user to quickly navigate the data portal	This component is a part of the frontend architecture of the SDD.	No changes.
Header_Component	This is a fixed component that will allow the displaying of important information.	This fits into the UIDDD column description for all pages. (UIDD 2.1 - 2.1.1).	No changes.
Project_List_View	This component allows for a user to be able to view projects	This component is a part of the frontend architecture of the SDD. The design for it is also stated in the UIDD (2.1.5). Additional UIDD information under section 2.2.4 of the UIDD.	Functionally, no difference. Visualization wise, different. The visualized page lacks the body structure from the UIDD, instead existing as two floating and separated components. Note: this change needs to be reversed.
Upload	Allows user to upload project data to be saved in the database	This module is a part of the backend architecture detailed in the SDD in section 2.1	No changes.
About_Page	Explanatory page that describes Blue Marble Geographic, what it does, and the purpose of the data portal.	This module was added to inform users of Blue Marble Geographics.	Only change was to add this to the frontend.
Home_Page	Default frontend landing for any user that accesses the data portal.	This module is a part of the frontend architecture listed in section 2.1. This module is also discussed in the SDD within sections 2.2 and 3.2.	No changes.
LoginLogout_Page	Frontend page that allows a user to login to their account so that they can utilize portal features that require an account. Similarly allows individuals to logout.	UIDD (frontend) component of this module is listed in 2.1.2. For the SDD, login/logout information is listed under section 2.1 (Architectural Design). Logout, in particular, is listed in section 3.2.	Signing in with Google not included as originally intended. Google auth is currently in a weird state of constant documentation changes and development. We elected to drop the intended inclusion of Google's services for login.

Profile_Page	Frontend page that allows signed in users to view their information and projects.	UIDD (frontend) component of this module is listed in 2.1.4.	No changes.
SignUp_Page	Frontend page that allows for individuals to become verified users with visible accounts.	UIDD (frontend) component of this module is listed in 2.1.2. In the SDD signup is listed under sections 3.1 and 3.2. Brief notes also exist in SDD section 2.2 (pages 14 and 15 of the doc).	No changes.
UserSetting_Page	Page that allows a user to adjust their password, email, username, pfp, and any preferences added following the data portal's initial development.	The user settings page was not initially included in the UIDD or SDD. Documentation referenced user settings as if it was planned for, and in the back of our minds we all seemed to agree that such a page should exist. That said we neglected to include any framer work or notable description for this page. Considering this page is required from the perspective of user experience, we made the jump to include documentation as well as a module/task for the page.	No change due to the fact documentation was added as development occurred.
Dropdown	Frontend collection of navbar menu items condensed such that they are only visible when moused over. Reduces navbar clutter.	Frontend component, listed in the UIDD under section 2.2.7.	No changes.
MenuItems	Frontend buttons contained within the Navbar and Dropdown, these buttons allow individuals to jump to a different page on the data portal.	UIDD (frontend) component of this module is listed in 2.2.2.	No changes.
Navbar	Component of the frontend view located at the top of all pages. The Navbar contains almost all menu buttons that	UIDD (frontend) component of this module is listed in 2.2.2. Navbar reference in formation can	No changes.

	allow for jumping between pages on the data portal. Having all buttons in one place offers clarity to the user (for how to navigate the portal). (Note: the navbar does not contain the button to the about page).	also be found in sections 2.2 and 3.2 of the SDD.	
LoginFile	Frontend component that handles user login by collecting user input.	This is part of the frontend architecture in the SDD, section 2.2: Login Page	No changes.
PrivateRoute	Frontend component that enforces authentication for pages that requires a user to be authenticated	This is part of the backend in the SDD, section 2.2: Authentication controller.	No changes.
SignFile	Frontend component that collects user input and passes it to the backend for registration to the system.	This is part of the frontend architecture in the SDD, section 2.2: Signup Page	No changes.
User_Views	Backend component that handles web requests from the frontend to create a response.	This is part of the backend in the SDD, section 2.2: Authentication controller	No changes.
User_Urls	Backend component that stores the various urls pertaining to users and authentication	This is part of the backend in the SDD, section 2.2: Django Web Server	No changes.
User_Serializers	Backend components that allow for the conversion of data to python data types to then be rendered into JSON format..	This is part of the backend architecture for Django in the SDD in section 2.2: Django Web Server	No changes.
User_Models	Backend component that sets up a custom user module that creates users and maps their information to the database.	This is part of the backend in the SDD, section 2.2: Django Web Server	No changes.
User_Apps	Backend component that stores the app names and labels for the databases.	this is part of the backend in the SDD, section 2.2: Django Web Server	No changes.
User_Permissions	Backend component that allows only owners of an object to edit it.	This is part of the backend in SDD in section 2.1.	No changes.

App.js	Frontend component that contains the main view, handles the routing and all imports to form the user view.	Frontend Architecture in section 2.1 of the SDD	No changes.
--------	--	---	-------------

4. Defects

4.1 Introduction

This section will report all defects that have been identified within the software through a tabular format. This will also include all non-compliance issues such as coding or commenting development standard deviations. Modules will be broken down and analyzed, noting defects which will be categorized by the different violations. This includes but is not limited to; correctness, coding convention, commenting convention and user friendliness. Additionally, there will be another category known as “other” which will be used for defects which do not conform to any of the previously noted violations.

Defect #	Module Name	Defect Violation Description	Defect Category
1	Upload.js	The function “fileData” on line #33 does not have an appropriate header comment in the /* Text */ format, instead using two instances of “// Text”.	Commenting Conventions
2	App.js	Removed unused imports	Coding Conventions
3	App.js	Removed unused lines of code	Coding Conventions
4	CSVReader.js	There is a commented out line of code on line #6.	Commenting Conventions
5	SignUp.js	This function’s “SignUp” and “Sign” on lines #7 and #14 are not correctly described in detail in their above comments.	Commenting Conventions
6	Pages.css	None of the functions are correctly commented on pages.css, each of the current comments are spaced in incorrectly or are on the same line as the function.	Commenting Conventions
7	Settings.css	Needs comments to better explain the structure of page divisions (there’s a visual explanation for it on hand, but it needs to be	Commenting Conventions

		translated in full). Other components were not originally commented on, but have been since.	
8	Navbar.js	There is an unused line of code found on line #8 which imports an unused library.	Coding Conventions
9	Loginfile.js	There is currently an issue with how the signup button is presented on line #30 → #32, when viewed by the user the button is twice as large as it should be and has filler text.	Other
10	LoginFile.js	There is currently a lack of comments describing the functionality of LoginFile.	Commenting Conventions
11	Sign.css	The formatting parameters within the file are not properly labeled from lines #152 → #162.	Commenting Conventions
12	Button.js	This component originally was not reusable but was changed to be able to handle different button types.	Coding Conventions
13	urls.py	an unused url was present and removed	Coding Conventions
14	views.py	An unused view file	Coding Conventions
15	Auth.css	The formatting parameters within the file are not properly labeled from lines #164 → #183.	Commenting Conventions
16	Search.js	This file is not currently labeled with anything to describe what it does.	Commenting Conventions
17	UserSettings.js	The main branch version of this file still contains the basic "lorem ipsum?" text used for filler in documents. (Currently the completed file is in a different branch).	Other
18	About.js	This file is not currently labeled with anything to describe what it does.	Commenting Conventions
19	ProjectListView.js	Currently the visualization does not match the standardized format seen in the UIDD between descriptions and the representative framer image.	Correctness
20	Dropdown.js	The commenting on this file is placed within brackets and isn't spaced correctly.	Commenting Conventions
21	UserSettings.js	Currently lacks a special image/button to denote that the pfp can be changed	User Friendliness

		(something like a small camera icon).	
22	General Frontend	Currently only the simpler frontend pages have mobile support built in. While this is not required of us, it is still something Blue Marble will inevitably need (and thus something we should have structure for as a courtesy).	User Friendliness
23	General File Naming	Some files are incorrectly named with the first letter being lowercase rather than uppercase.	Correctness
24	UserSettings.js	The initial placeholder for data portal language adjustment was an input box rather than a button. While the ability to change display language isn't part of our design goals, the structure is there as a courtesy to BMG (although an input box still isn't useful!)	Correctness

Appendix A – Client and Team Review Sign-off

Within this section of the CIR, you will find the signatures, names and date of the documental approvals for both Blue Marble Geographics Chief Technology Officer and each acting member of "Team Undershrub ". These signatures acknowledge and authenticate the approval and review of this document. This authentication includes the overall content, formatting, identification of contributed material and development directions presented within. These signatures must be updated alongside the date and comments when there is a subsational update to the document, which in this case includes any formatting, editing, grammar and change of materials within.

Name: Victor Minor	Date:
Signature:	

Comments: Client

Name: Anthony Jackson

Date: March 9th, 2022

Signature:



Comments:

Name: David Sincyr III

Date: March 9th, 2022

Signature:



Comments:

Name: Devin Carter


Date: March 9th, 2022

Signature:



Comments:

Name: Stephen Kaplan	Date:
Signature:	
Comments: Was not around to sign	

Name: Grant Shotwell	Date: March 9th, 2022
Signature: 	
Comments:	

Appendix B – Document Contributions

Grant Shotwell 0%	
Stephen Kaplan 0%	
Devin Carter 33%	<ul style="list-style-type: none"> • Wrote section 2.2 of the document. • Wrote section 2.3 of the document (with one paragraph completely rewritten as per received comments). • Added and described some of the defects seen in 4.1 (Namely the settings/usersettings defects as well as general frontend and naming). • Added modules to the inspected modules section of the

	<p>document (second half of 3.1).</p> <ul style="list-style-type: none"> • Filled in information for 3.1 inspected modules that were added by others (targeted frontend modules). • Worked on some of section 1.4's remarks.
<p>David Sincyr III 34%</p>	<ul style="list-style-type: none"> • Set up initial document • Added Title page, appendixes A, B, C from previous documents • Wrote all introduction sections • Set up and filled in possible defects table • Created and filled in inspection meeting table • Created the table of contents • Wrote sections 1, 1.1, 1.3 coding conventions • Created and filled in section 4.1, 3.1 • General editing of entire document
<p>Anthony Jackson 33%</p>	<ul style="list-style-type: none"> • Wrote the coding and commenting conventions for sections 1.3 • Writing and editing for section 1.4 "defect checklist". • Adding remarks and some additional defects within sections 1.4 table. • Editing for section 2.2 "descriptions" • Editing for section 2.3 "impressions of the process" • Created the initial tables for section 3's "incomplete" and "inspected" modules. • Added and described some defects to section 4.1. • General Editing on the format and display of documents.