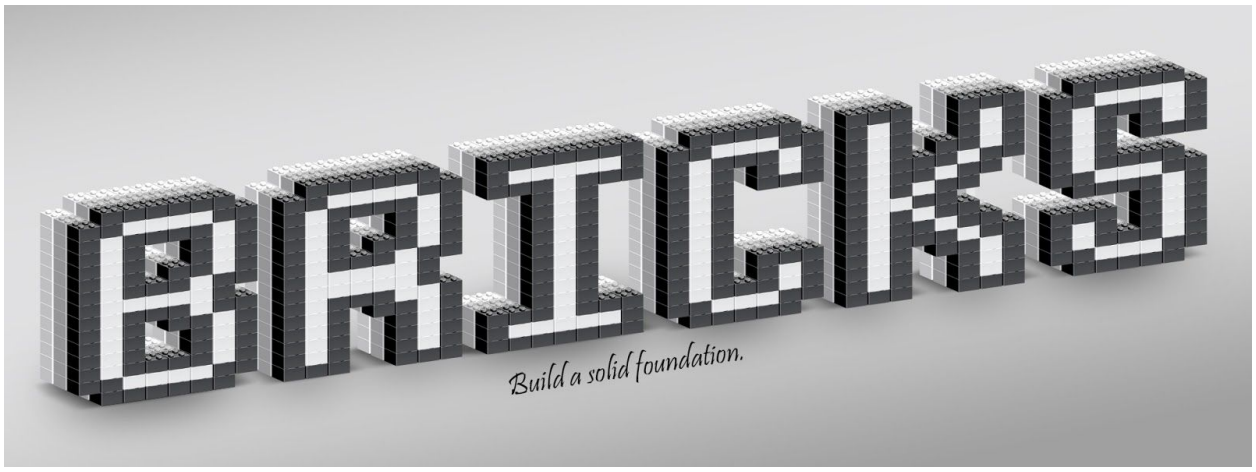
Code Inspection Report

The Redistrictinator

Client: Geoffrey Weiss



BRICKS

Benjamin Jeremenko

Jacob Philip

Sumanth Neerumalla

Zachary Elliott

Francis Kato

Nathaniel Fuller

Table of Contents

1. Introduction

1.1 Purpose of This Document

The purpose of this document is to provide an overview of the coding and commenting conventions of the group, a list of defects found during the inspection process, and the process of inspection used. This document is intended for the customer, the initial development team and any future team that may build on this project.

1.2 References

I. *System Requirements Specification Document,* The Redistrictinator

II. *System Design Document*, The Redistrictinator

III. *User Interface Design Document*, The Redistrictinator

1.3 Coding and Commenting Conventions

The coding and commenting conventions agreed upon were created from guidelines that were used in previous computer science classes as well as coding habits that were commonly used between most of us. The rationale behind creating our own coding conventions was that outside guides were often unnecessarily long and detailed for the type of project this is and that it would be simpler to create a handful of simple rules that we all already followed instead of going through several pages of standards to make sure a guide was followed correctly.

The coding conventions were used to keep the programs consistent. This included variable and function naming, indentation methods, and avoiding confusing and long statements. For the commenting conventions, they revolved around making sure the comments were written in a consistent style and that they could always be expected in certain areas of code. The details for the coding and commenting conventions can be found in *Appendix A*.

1.4 Defect Checklist

Coding Conventions	**Complex statements that can be broken down into easier to understand pieces**
	Variable name are lowercase or use underscores
	Function name are lowercase or using underscores
	Maintain proper indentation. No excessive indenting, even if it doesn't result in errors.
	Avoid excess whitespace
	Used spaces for indentation instead of tabs
	Remove inaccessible code (ie: no extra code after return statements)
Logic Errors	**Ensure that the function does only what it states it will do**
	Pages must load correctly and website has to handle traffic.
Commenting	**Inline comments**
	Over/under commenting function descriptions
	No comments on loops
	Too few comments in file
	Incorrect comments or vague comments
	Poor or missing comments on functions

Security Oversights	**Only relevant files are served to users**
	Ensuring that the libraries used in our project don't have vulnerabilities
	Ensure that no one except for authorized users can access the production server. This means private key access only
	Ensure that only authorized users have the permissions to modify the relevant folders and files on the server
	Avoid installing any unnecessary packages or software
	Avoid opening any ports that are not required to be open
	Our website is available over http, this means that it is vulnerable to a Man In The Middle attack, should our users ever become targets.

2. Code Inspection Process

2.1 Description

The code inspection process began with a review of the code and the commenting conventions that were decided upon. Next, the author or authors for each module wrote down any logical errors and broken code instances that were present in the code they were working on. After the authors recorded the errors that they knew were present in their modules, two members of the group went through the files to see if any other logical errors existed as well as checking if coding and commenting conventions were followed. The defects that were found during this review were dealt with in one of two ways. If the defect found related to coding and commenting

conventions, then they were corrected immediately. If the defect was more complex, the author of the program was made aware and the defect was noted in the team communications software. This process is repeated for every module in the project to keep consistency of coding conventions and to help keep errors to a minimum.

This process was not necessarily ideal and a more thorough testing procedure could have been used if not for time constraints. Due to the iterative process that we used to develop our project, and the multiple unknown factors that resulted in software development slowdowns, our code review process was not as thorough as it could have be had we known exactly what we were doing from day one. Our approach to solving technical problems and the software tools we used changed throughout the course of product development, and this result in more sparse, periodic review until closer to the delivery date. We decided that the members who were still writing their code would list the defects that were known and actively being worked on while members that were more free would test the code and check for more general defects like readability and good commenting.

2.2 Impressions of the Process

The code inspection process was effective in helping find and fix smaller defects having to do with coding and commenting conventions. However, due to much of the code still being worked on, more errors could have possibly been introduced after inspection. To improve the process for next time we would need to have all the modules completed so the whole group could meet and complete the inspection process together. We purposefully aimed for core functionality requirements in our initial design documents, so that we would have something solid to deliver, but due to last minute bugs in our districting algorithm, we ended up not being able to test that component as thoroughly as we would have liked.

The best modular unit in our program would be the user interface. This is because for this project the designed user interface is very simple, moving users between a very limited number of pages which leaves less possibilities for flaws. Instead of writing everything from scratch, our front end designer used the industry proven angularjs library to develop our website. We kept elements and pages to a minimum in order to leave the user with only the things they needed to

use our tool. This meant fewer components on the front end to worry about, and fewer issues that came up during its development process.

Our infrastructure and dev ops was a component in our project that did not wildly outperform but it is one we are relatively happy with. The use of Flask as our webserver meant that we did not have to write our own networking code, and so the code review for our backend infrastructure was very efficient. Additionally, using AWS for our server infrastructure meant that doing a security review and checking that the machine was locked down was done easily from the AWS dashboard..

The worst unit in terms of possibility of flaws would be the algorithm. This is because the code and logic behind it is more complex and it is easy to overlook small flaws in the code if they are not easily visible from the output. This was a pain point in our project from the very beginning, and if we had a chance to do this project again, we would have focused on this more from the beginning of the project.

2.3 Inspection Meetings

As all of our code was not completed on time for this document (namely the algorithm), the inspection meetings held were informal and took place over online chats that lasted for extended periods of time. All major units of code were covered, which were the algorithm used to create the new districts, the user interface of the web page, and the implementation of Google Maps with the overlay for showing the districts. A scribe was not needed as all our conversations were recorded by the app being used and for each major unit, the authors of that unit were leading out and showing the defects.

3. Modules Inspected

I. Algorithm
   A. This module reads in data (zip codes, population, boundary coordinates) and outputs the state data separated into the specified number of districts.
   B. The algorithm module is part of the back-end of 'The Redistrictinator' and is specifically part of the "Fetch Data" and a bit of the "Generate Map" sections of the SDD. It prepares the data from the server and sends it to the Mapping module to be displayed.

II. Google Maps
   A. This module takes the data from the algorithm and displays it over a Google Maps view of the specified state.
   B. The Google Maps module is a part of the front and back-end in the SDD, specifically a part of the "Generate Map" and "Get Map Data" sections. It takes the newly made districts from the Algorithm module and creates an overlay for the new districts to be displayed on Google Maps on the front end.

III. User Interface
   A. This module allows the user to navigate the web app and displays the working algorithm and altered Google Maps map on the web page.
   B. The User Interface module is a large part of the front-end section of the SDD diagram. It takes the Google Maps map with the overlay showing the new districts and displays them on the web page with the "Generate Map Button".

There are not noticeable differences in our resulting code and our SDD due to our SDD being minimal

4. Defects

Module Name	Defect Category	Description
Algorithm	Functional correctness	Zip codes are not entirely contiguous when displayed.
Algorithm	Functional correctness	Last district is smaller than the rest
Mapping	Functional correctness	Excess lines displayed between districts.
Mapping	Displaying districts	Combining the KML of zip codes into districts results in outlines of the zipcodes.
Mapping	Displaying districts	The map doesn't fill in districts with a solid color, and doesn't separate districts by color
Data	Incomplete	Missing data for all zip codes.
Web Server	Functional correctness	Output from backend must be manually copied over to html to display map and districts.
Web Server	Functional correctness	We don't have a web domain, and are relying on noting down the ip address every time the server boots up
User interface	Functional correctness	Certain browsers like IE 11 are having trouble displaying entire content of the site.
Front-end	Pages loading slowly	While navigating through the site, the pages aren't loading fast enough.
User interface	Missing content	Certain pages are yet to be fully finished and integrated into the site.
Algorithm	Commenting	Certain parts of the algorithm are only understood by the team member that wrote it, and those portions are not fully commented.
Front end	Coding convention	There are minimal comments for the front end source code, will be difficult to understand if someone new takes over.

Appendix A - Coding and Commenting Conventions

- Aim for one comment for every 7 lines of code
- At minimum, one comment explaining a loop
    - # prints out the numbers 0,1,2,3,4
    - for x in range(5):
        - print(x)

- Functions should have a short comment explaining its use
    - # prints out happy birthday
    - def happy_birthday(person):
        - print("Happy Birthday!")
- No inline comments
    - # Correct
    - # stores a person
    - string person

    - #Incorrect
    - string person #stores a person
- Tabs used for indentation instead of spaces
- Variable and function names are lowercase and if two words an underscore separates it
    - # prints out happy birthday
    - string person
    - def happy_birthday(person):
        - print("Happy Birthday!")
- Global variables are written in all uppercase
    - # stores the total number of districts for MD
    - int TOTAL_DISTRICTS

- Function names should be descriptive
    - # Correct
    - def happy_birthday(person):
    -     print("Happy Birthday to you!")

    - # Incorrect
    - def foo(person):
    -     print("Happy Birthday to you!")
- Avoid extraneous whitespace in general aka no spaces in between commas
    - # Correct
    - def happy_birthday(person1,person2):
    -     print("Happy Birthday from ",person1,"to ",person2)

    - # Incorrect: there is extra space between the commas
    - def happy_birthday(person1, person2):
    -     print("Happy Birthday from ", person1,"to ", person2)
- Avoid compound statements
    - # Incorrect: multiple statements in one line are difficult to read
    - if (var1 == 80 or var1 == 443 or (1024 <= var1 <= 65535)):
    -   print("Good")

Appendix B – Team Review Sign-off

All of the following team members have reviewed this document and agree to the content and format.

Team Members:

Benjamin Jeremenko: ____  Date: ___11/28/17_____

Comments: _____

Jacob Philip: _____  _____ Date: ___11/28/17_____

Comments: _____

Sumanth Neerumalla: _____  __ Date: ___11/28/17_____

Comments: _____

Zachary Elliott: _____  ____ Date: ___11/28/17_____

Comments: _____

Francis Kato: ____  _____ Date: ___11/28/17_____

Comments: _____

Nathaniel Fuller: _____  _____ Date: ___11/28/17_____

Comments: _____

Appendix C – Document Contributions

Benjamin Jeremenko:	Introduction, Code inspection process	30%
Jacob Philip:	Introduction, Modules inspected	40%
Sumanth Neerumalla:	Defect checklist, Module defects, Impressions of Process	50%
Zachary Elliott:	Defect checklist, Defects	45%
Francis Kato:	Defects, Appendices	25%
Nathaniel Fuller:	Code inspection process, Modules inspected	45%