# *Project Save Baltimore*

# -

## Code Inspections Report (CIR)

Version 1.0

Produced For:
Next Century Corporation

Produced By:
Team Indigo (CMSC447):

- Neil Joshi
- Nat Baylon
- Matthew Landon
- Bernie McNamee

Date: April 19, 2016

*Save Baltimore*
Code Inspection Report

## Table of Contents

## 1. Introduction

### 1.1 Purpose of This Document

This document intends to ensure the quality of the tests meets the requirements. The first section describes the coding standard that was implemented by the team. By using common conventions, we hope to create a code base that is readable and usable by all members. We describe the discovered defects in regards to our checklist. Finally, we detail how the process of code inspection occurred and our results. This document is for people interested in the coding styles we used and the current state of the project. Specifically, it could be useful to the client, Next Century, because of their abnormal technical knowledge compared to other clients.

### 1.2 References

Systems Requirements Specification by Team Indigo

System Design Document by Team Indigo

Index of Python Enhancement Proposals (PEPs) by David Goodger & Barry Warsaw
https://www.python.org/dev/peps/

PSR-1: Basic Coding Standard by PHP Framework Interop Group
http://www.php-fig.org/psr/psr-1/

### 1.3 Coding and Commenting Conventions

Team Indigo adopts several conventions to maintain a consistent coding style. To name variables, we follow the camelcase convention, starting the variable name with a lowercase letter and using capitalized letters at the beginning of other words in the variable names. To maintain a good level of clarity in our coding, we make sure we have at least one line of comments per code block. In terms of white space, we don't use more space than necessary, but we make sure there is space between different logical ideas and blocks to aid the clarity of our code. These conventions are standardly used by many coders, and we agreed as a team before hand that by keeping these conventions that we are used to already, we can most easily read each other's code and be able to maintain, edit, critique, integrate, and test it whenever necessary.

An example of our conventions can be found in Example 1 of Appendix A.

1.4 Defect Checklist

| Test # | Category | Description | P/F Criteria | Inspection Comments | P/F? |
|---|---|---|---|---|---|
| 1 | Coding Convention | Well-Commented Code | All logic blocks are commented adequately for the non author to read | Comments have not been completed | F |
| 2 | Coding Convention | Adequate use of whitespace | Sufficient white space is present so it is easy to see what code can be grouped together to do one action | Most code follows this | P |
| 3 | Coding Convention | Logical choice for code blocks/ideas | For each block, the purpose can be stated in one or two sentences | We all have good experience doing this so break down makes sense | p |
| 4 | Coding Convention | Re-using code/ good modularity | Individual modules should have their own functions | Files are broken up into sub actions | P |
| 5 | Coding Convention | Abide by the PEP and PSR-1 coding standards for Python and PHP code | Go through the rules of PEP and PSR-1 and check code for adherence | The PEP and PSR-1 guidelines are very in depth. It is nearly impossible to follow them 100%. | F |
| 6 | Security Oversights | Guarding against SQL injection | Users cannot enter queries and have them be run | No text input from users | P |
| 7 | Memory Problems | Buffer overflow, caused by not checking input length | Test inputs with long queries to ensure no buffer overflow issues | No text input | P |
| 8 | Memory Problems | Buffer overflow, caused by too many sql results | When entire data set is loaded, does anything get damaged | Full dataset read in with no error | P |

| | | | | | |
|---|---|---|---|---|---|
| | | from an executed query | | | |
| 9 | System Flaws | Cron Job fails to run | The script to read data must run at the desired time | This is not automated yet | F |
| 10 | System Flaws | Website can withstand hundreds of users | Test website with many users to see if website crashes or not. | untested | |
| 11 | User Experience Problems | Site takes too long to display data | Load site, to ensure it loads in less than 3 seconds | Map refreshes each time new filter is applied, | F |
| 12 | Functional Problems | Views correctly reflect filters | When a filter changes, the views should be replaced by only the new data | Filter selections are populated with data, but these are not fully connected to data retrieval system yet | F |
| 13 | Functional Problems | Expected results are returned from queries | Given the desired filters, the data should reflect only crimes that fall within the provided parameters | The data retrieval needs to be fixed to allow multiple options for each filter | F |
| 14 | Functional Problems | Filters are populated completely and correctly | Each category that a user can select from to pull more specific events should have all possible options | All distinct values present | P |
| 15 | Functional problems | Ajax calls return the correct data to the views | Calls to database should return the records that correspond to the filters in a json format | All data returned when no filters applied | P |

## 2. **Code Inspection Process**

### 2.1 Description

While we developed our application, we each worked on different pieces of code that we integrated together. While doing this, we met frequently to discuss the architecture and ideas behind our application and stayed on the same page. In order to integrate the code, we examined the frame that we already created and gained a comprehensive understanding of it before writing additional pieces of code. This made it so we had to do minimal guesswork when it comes to understanding each other's code. The different views interact with index, but not with each other, so we just had to make sure index was very good and to understand how it works.

When we finish making a view, we individually make sure our own code is up to the group's coding standards and individually test the blocks of code and relevant functions that we created. Then, when we are satisfied with our own code and behaves in the way we designed it to work, we talk to the team about what it does, show each other how it is used and what types of input/output to expect.

From that point, the members who didn't work on developing the new functionality examine the code, gain an understanding of how it works, and try to find vulnerabilities and holes in each other's logic. Since people tend to believe their own logic is correct, when other eyes see the code, the other pair of eyes may have a different mindset about what is correct, thus potentially seeing things that the author of the code didn't notice. After we individually scrutinize the code and think of test cases and walk through the code, we commit the code to the master branch and then it is a piece of the overall structure.

### 2.2 Impressions of the Process

Our inspection process continually happened while writing the code. It has been evolving to where it is now. At first, we set off to do our own tasks, but individually had some trouble. This prompted the need for us to inspect each other's code. Our code-inspection process evolved organically, and where we currently are, we are fine with how we implement our code. Also, the more we collaborate while coding, the more insight we can gain from each other so that we can maximize our productivity.

The best modular unit in our program is the backend: data.php. It is a simple but powerful and important piece of our architecture, and we have looked it over many times because all the data comes from it. We believe that this file is strong and dependable. The worst modular unit in the program used to be the map, because we had many bugs trying to get it to load correctly. We all looked at it and gave the author feedback to help him get it to work well. Now, the map is up to par with our coding standards, and it has undergone rigorous testing.

As we continue developing, we are realizing that the more collaboration and eyes that go over the code, the more errors we can catch and so the process of developing is becoming fast and comfortable.

2.3 Inspection Meetings

We all took notes on the topics discussed, so there we were all scribes.

| Meeting # | Date | Location | Time Started | Time ended | Participators | Units tested | Moderator |
|---|---|---|---|---|---|---|---|
| 1 | 3/30/2016 | Library | 8:45 PM | 10:00 PM | Matt, Bernie, Neil, Nat | DB, Script, Index, pie chart | Matt, Nat |
| 2 | 4/6/2016 | Library | 8:45 PM | 10:30 PM | Matt, Bernie, Neil, Nat | Backend, map | Matt |
| 3 | 4/13 | Library | 8:45 | 10:00 | Matt, Bernie, Neil, Nat | Map, filters | Bernie |

3. **Modules Inspected**

| Module | Description | Differences between planned and actual implementation, algorithms, data structures, i/o | Comments |
|---|---|---|---|
| Filters | User specifies which data they want to see | None | |
| Backend (data.php) | Returns requested info from the db in JSON form | None | |
| Map View | Displays heat density map, populated by the backend | None | |

| | | | |
|---|---|---|---|
| Pie Chart | Displays occurences of items in a filter, populated by the backend | None | For Pie Chart and Line Graph: We are still deciding which exact filters we'd want to display, but we have working examples of the filters. |
| Line Graph | Displays occurrences of a filter over time, populated by the backend | In Production | |
| Histogram | Above the time slider filter, shows a frequency view of crime occurrences. | In Production | This module was not in the SDD because it was a new requirement given to us at the midterm presentation. |
| Table | Shows the data in a tabular view | None | |

## 4. Defects

| Module with Defect | Description | Defect Category |
|---|---|---|
| Data retrieval | The module that pulls data does not incorporate multiple filter option | Correctness with respect to functional specs |
| Filters refreshing views | Views are not connected to filtering so all three do not change after filter | Functional correctness |
| Map refreshes each filter | Each time data is filtered, the entire =map flickers | Functional correctness |

## Appendix A - Coding and Commenting Conventions

Example 1: With a logical idea, we use whitespace before and after the idea and make sure we comment it to explain what the purpose of the code is. Variables named, such as graphFrame, start with a lowercase letter and have an uppercase letter for every following word.

```
//force an onClick event on the hidden 'div' to call redraw() chart
document.getElementById('graphFrame').contentWindow.document.getElementById('dataDiv').c
lick();
```

### Appendix B – Team Review Sign-off

| | |
|---|---|
| Nat Baylon | 4/19 |
| Matthew Landen | 4/19 |
| Neil Joshi | 4/19 |
| Bernie McNamee | 4/19 |

### Appendix C – Document Contributions

Identify how each member contributed to the creation of this document. Include what sections each member worked on and an estimate of the percentage of work they contributed. Remember that each team member must contribute to the writing (includes diagrams) for each document produced.

Nat: parts of 1.3-4
Matthew: parts of 1.3-4
Neil: 1.1, 1.2, 1.4, some 3
Bernie: some 3, some 1.3, 1.4