Project Save Baltimore

-

Testing Report (TR)

Version 1.0

Produced For:
Next Century Corporation

Produced By:
Team Indigo (CMSC447):

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

Date: April 28, 2016

Save Baltimore
Testing Report

Table of Contents

1. Introduction

1.1 Purpose of This Document

This document explains the method of testing utilized for Save Baltimore. It is intended for people who will be evaluating this project. It will describe the methods of how the tests were carried out and what the results are. This information will reinforce the quality of the program and provides a secondary means of verification.

1.2 References

Systems Requirements Specification by Team Indigo

System Design Document by Team Indigo

2. **Testing Process**

2.1 Description

Our team decided to implement a bottom up testing approach. In this design, each component is tested independently with unit tests. Each developer is responsible for testing their own code they wrote because they know the way it works. Although the writers of code can miss problems in the code because they see what they thought they wrote. This issue is resolved through later testing phases. After the developers verify that their own code works, groups of components are merged together and tested. This process is iterated until a full system is tested and verified.

This is a web application which resulted in adaptations to the testing framework. We all had a central repository which contained the entire site. This means that the current site was running on all member's system so, it was easier to catch bugs. So, first the team began by testing each visualization with no filtering. Each page was written by a separate member so, these formed the components of the system. They all went through testing to make sure that the data is represented accurately and as we desired. Integration testing was done in the form of filter integration. We had to connect all of the visualizations to a common data retrieval system to allow all of the tabs to be updated from the same crime events. These tests consisted of layering multiple filters to the program and verifying that the data could come out of the database given many different filter combinations.

2.2 Testing Sessions

Date	Location	Time Started	Time Ended	Performed By	Use Case Covered*
4/5/16	Online	7:00 pm	8:00 pm	All	F: 2, 3 NF: 1, 3, 5
4/12/16	Online	7:00 pm	9:00 pm	All	F: 3, 4 NF: 2, 5, 6
4/19/16	Online	7:00 pm	8:30 pm	All	F: 1, 2, 4, 5 NF: 4, 6, 7, 8, 10
4/26/16	Library	7:00 pm	10:00 pm	All	F: All NF: All

* Use Case Covered = Number of functional / non-functional requirement (see SRS)

2.3 Impressions of the Process

Overall the testing process was very effective in finding errors and improving code functionality. Testing was done on completed components, and as more components were added, they were tested in the next session. Flaws were often found on edge cases, during stress testing (testing max/min DB queries, etc). Through the testing process, we were able to fortify our system to better handle data flow and user-system interaction. Before putting the system through testing, we had flaws with incorrect data being displayed and minor database query flaws. As a result of testing we were able to create a more robust system, able to handle much more.

The best units of the program would be the map view and chart views, mainly because of their simplicity. All they need to do is take in some data and visualize them the same way every time. Google provides very robust APIs for these views that prevent many flaws from ever becoming too prominent.

The unit that filters and passes the data to the different views is fairly complex, and because of the amount of data, performance is important. After the associated coding inspection, many of this component's flaws became even more prominent (Test #12 & 13 in Defect Checklist) The histogram also has the potential for many flaws that aren't fully discovered yet. This is because it was a recent addition that hasn't been tested well, even though it is a very complex component.

The choice of the best module has changed from the backend to the charts/map view. Recently, the charts and map were mostly finalized, however the data pulling mechanism needs to accept multiple inputs and needs to be able to deals with these. We have several multiple selection fields where the user is able to pull all events pertaining to these specifications. Then, the query had to be built such that the result had all of the requirements. In addition, there is a time interval to search for, thus adding even more complexity.

On the other hand, the maps and charts are simple on our side. These pages use the Google apis to display data. These interfaces provide simple mechanisms to show data. All we were required to do was format the crime data in a particular way and throw this into a graph.

3. **Test Results**

Use Case: F1: User Requests Filtered Data (Tester: Matt)

Equivalence Partition: User can only select from data available in the DB

1. Test Case 1: Make sure that form elements are being created based on data in the DB
    a. This is to ensure that the user will select appropriate information for DB queries.
    b. Summary: Functionality to propagate the filter form worked as planed. The user is able to make queries according to the DB. Inputs are then sent to be queried and returned as a JSON obj.
2. Test case 2: select one option from each category and making sure that all visualizations update correctly
    a. This was the beginning of the tests for filters to provide a baseline of functionality to build on
    b. Summary: individual filters work to limit data to all visualizations
3. Test case 3: combine multiple options over more than one category
    a. This helped test cases that are less likely to happen in normal testing
    b. Summary: Provides an in-depth test to try to account for all possible combinations possible with the filter menu

Use Case: F2 Heat Map updates (Tester: Bernie)

Equivalence Partition: Data come from user queries - can only be from DB data

1. Test Case 1: Log user query results to a file to validate the map updating
    a. Do this to make sure that the map is displaying the correct data.
    b. Inputs: JSON form the database. After user has entered query filter
    c. Outputs: Map display
    d. Summary: We found some queries coming back null (typo error). We were also able to show that the data was being plotted in the correct way.
2. Test Case 2: Stress Test - Try complex queries, in addition to the displaying the full DB on the map
    a. Use to make sure the map can handle all points with speed and agility
    b. Inputs/Outputs: from before
    c. Summary: With this test we found that the data to be too large for the server memory to handle. After talking with the customers we came to the solution of displaying only a fraction of the most recent data to the screen until more server memory can be added

Use Case: F:3 View charts (Tester: All)

Equivalence Partition: data matches user selections and is within the given query

1. Test Case 1: Log user query results to a file to validate the charts updating
    a. Do this to make sure that the charts are displaying the correct data.
    b. Inputs: JSON form the database. After user has entered query filter
    c. Outputs: Map display
    d. Summary: We found some queries coming back null (syntax error). We were also able to show that the data was being plotted in the correct way.

Use Case: F:4 Tabular & sortable view of the Dataset (Tester: Neil)

Equivalence Partition: All rows of the table fall within the requested filters

1. Test Case: Log user query results to a file to validate the table updating correctly
    a. Do this to make sure that the table are displaying the correct data.
    b. Inputs: JSON form the database. After user has entered query filter
    c. Outputs: Map display
    d. Summary: We found some queries coming back null (syntax error). We were also able to show that the data was being plotted in the correct way.
2. Test Case 2: Sort columns by hand in phpMyAdmin and compare with table sorting

a. Used to ensure that the table sorts correctly
b. Inputs/Outputs: from before
c. Summary: Table was able to sort correctly on column header clicks. These were the expected results

Use Case: F:5 Admin updates dataset (Tester: Nat)

Equivalence Partition: Grabbing only NEW data from the online crime dataset

1. Test Case: Compare new date Timestamp with the one in our system DB (from the online crime dataset compare the last date received with the newest data of our system DB)
    a. We do this to make sure that the data coming in is most up-to-date
    b. Summary: We were able to import data chronologically, from our last most recent import - such where the results as expected.

Use Case: NF:1 Browser Compatibility (Tester: All)

1. Test case 1: Opened site in chrome, firefox and IE to verify the similarity
    a. Tested to ensure platform compatibility with the most commonly used browsers.
    b. Summary: HTML 5 and all JS apis ran as expected on the browsers tested. We would have liked to test others (Safari, Opera), however we had limited access to them.

Use Case: NF:2 Responsive UI (Tester: All)

Equivalence Partition: needed to find the highest amount of data that would not freeze the site

1. Test case 1: increased the number of records that were pulled out and stop when the frames were no longer responsive.
    a. Did this to stress the server capacity
    b. Summary: We found the limit to be around 50,000 DB rows in order for the site to still be operational. We may increase server size if time permitted.

Use Case: NF:3 Convenient Map (Tester: All)

Equivalence Partition: Map can be navigated by the technically unsavvy

1. Test case 1: Test covered in external demo to non technical user
    a. Important for system to be intuitive to non-developers, so average users can navigate map with ease

b. Summary: Google maps api showed to be very intuitive, and was able to be easily navigated. We used individuals in their 80' (team member's grandparents) to test. Results were as expected.

Use Case: NF:4 Stable System (Tester: All)

Equivalence Partition: System should not crash throughout stress testing

1. Test case 1: Once the record count limit was agreed upon, the system did not crash with the data given
    a. We need this because an unstable website is unacceptable
    b. Summary: Our previous experiences with databases and SQL proved helpful in creating a stable system.

Use Case: NF:5 Friendly UI (Tester: All)

Equivalence Partition: design makes sense to user

1. Test case 1: Showed the UI to Lourian to get feedback
    a. Wanted to get the opinion of someone with experience in this field
    b. Summery: She provided helpful recommendations that will improve the look and feel of the site
2. Test case 2: got opinion from outside friends about the system usability
    a. See how someone without knowledge of the project and how they interacted with the app
    b. Summery: saw real users and how they thought to interact with the site so we could improve the experience

Use Case: NF:6 Site Always Up (Tester: All)

Equivalence Partition: No error should shut down the entire site

1. Testing: This was tested throughout all the other tests
    a. This was done to prevent ensure fatal errors and worst case scenarios.
    b. Summary: Throughout our testing we came upon many errors that prevented interaction with the DB and inabilities to display data. However the site itself never crashed, and errors since then have been fixed. This was as expected, since the scope/size of our project would not likely ruin the servers

Use Case: NF:7 Site Security (Tester: All)

Equivalence Partition: No SQL injection should be possible to modify data

1. Test case 1: There is no option to enter free text data so, this is not possible
    a. This is important because modifying the data for a data visualization website could break the whole system
    b. Summary: This turned out to be a non-issue, thanks to the simplicity of the system, and our proper planning

Use Case: NF:8 Data Updates

*This use case eventually became the functional requirement detailed in (F:5)

Use Case: NF:9 Maintenance (Tester: All)

Equivalence Partition: System should be modular enough to ease maintenance]

1. Test case 1: This was verified during the coding inspection, we checked that there were clear divisions.
    a. Important because maintenance is usually the longest part of the SDLC, so effort should be given to make it easier.
    b. Summary: Our project was initially organized in such a way that updating would be easy throughout the SDLC.

Use Case: NF:10 User Profiles (Tester: All)

*This function was decided not to be critical and not added to the application

Appendix B – Team Review Sign-off

Nat Baylon	4/28
Matthew Landen	4/28
Neil Joshi	4/28
Bernie McNamee	4/28

Appendix C – Document Contributions

Nat: some 2.2, some 3
Matthew:1,2.1, some 3
Neil: 2.2, some 2.3, some 3
Bernie: some 2.3 and some 3