An Implementation of Integrated Visualization & Endpoint Modelling

By

Michael W. Schmidt

A thesis presented to the Honors College of Middle Tennessee State University in partial fulfillment of the requirements for graduation from the University Honors College.

Fall 2018

An Implementation of Integrated Visualization & Endpoint Modelling

Michael W. Schmidt

APPROVED:

____
Dr. Ryan Otter
Dept. of Biology

____
Dr. Phillip Phillips
University Honors College Associate Dean

Abstract

This creative work investigated Integrated Visualization and Endpoint Modeling with the intent to construct a software prototype. Integrated Visual Endpoint Modeling (IVEM) is a retrospective analysis tool that aids in assessing environmental damage. Specifically, IVEM is used to assess particular regions of a contaminated site using field collected data, but data are not directly comparable to one another. At present, a software implementation of IVEM did not exist. This paper details one approach to software implementation of IVEM utilizing C++, PHP, and web technologies to deliver a proof of concept. The works presented here resulted in a functional IVEM web application.

List of Figures, Appendices, Symbols and Equations

π	Greek little Pi. A numerical constant which is approximately 3.14159
e	Euler's number. A numerical constant which is approximately 2.71828
θ	Greek Theta. A variable typically used to represent an angle in radians. A circle has 2π radians.
$\sum_{i=1}^{n} i$	Summation. The current expression is the sum of $1 + 2 + 3 + ... + n$
$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$	Arithmetic Mean.
$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$	Standard Deviation.
$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$	Standard Error of the mean.
$Area = \frac{ab \sin(C)}{2}$	Area of a triangle, Side-angle-side method. (Sides: a, b; Angle: C)

I. Introduction

Environmental disasters will never go away, but retrospective testing methods let us assess damage and provide insight for clean-ups. One chief issue common to both man-made and natural disasters are the resources we choose to allocate to these affected areas. Accurately evaluating affected areas involves numerous tests from several different disciplines, and carried out across different regions (Smith 2013). To challenge matters, collected data does not always agree with similar testing methods. It is often hard to reach a consensus among scientists as to which areas are the most affected since each test describes the scenario differently. The panel scientists report their findings to are not typically composed of scientists which is also problematic.

Integrated Visualization and Endpoint Modelling (IVEM) is a method of standardizing, structuring, and presenting data developed by Dr. Ryan Otter. His approach to processing and visualizing data overcomes the issues faced by scientist and regulatory bodies described above. Even though IVEM solves these problems, creating an IVE model is a time consuming and error prone task. Several statistical values such as arithmetic mean, standard deviation, standard error, and the minimum/maximum values are need for each and every test and region. Trigonometry is also used for generating plots. All of Dr. Otter's IVE models are currently performed on a cumbersome Microsoft Excel spreadsheet.

Computers are able to solve a myriad of problems for their ability to retain large amounts of data and execute hundreds of billions of operations per second on modern hardware. Spreadsheet applications can be excellent tools for solving general data-centric problems, but more complex problems may require a program tailored to a specific need. Computer science is an interdisciplinary field for solving computationally heavy problems in Mathematics, Biology, Physics, Chemistry, and other data driven fields.

A computer program to perform these burdensome tasks presents an excellent opportunity to apply computer science in the environmental toxicology field. IVE models require organizing data, creating statistics that describe the data, manipulation of that data, and generating plots. As of October 2015, no full implementation exists; only a complicated Microsoft Excel document which serves as a proof of concept. Creating a functional IVEM implementation applies computer science to accomplish a task in minutes rather than days. IVEM is an excellent retrospective analysis tool that should be made available beyond the currently time consuming process.

II. Objective

The aim of this project was to develop a functional program that implements IVEM and test this program's results against data provided by Dr. Ryan Otter. In order to accomplish this task, several other smaller objectives first needed to be completed.

The program needs a method to store and handle user collected data appropriately, but in order to do so that data must first be organized. Therefore inputting, storing, and processing data needs to start with users providing well-formatted data. Providing guidelines on how to format data is a requirement in both large- (How to Choose a Data Format) and small- (Best practices for file formats) scale projects. These guidelines must provide a formal system on how this project can expect to receive data, and the details the steps necessary for proper operation. On the application-side, data storage must consider the order and amount of data received. Adding/removing data are user-facing objectives the program should meet, but internally, statistics from this information will need to be calculated, and the data will also need to be sorted accordingly. Information will need to be passed both in and out of this internal arrangement too. This internally arranged structure chiefly needs to represent an area that underwent testing with methods to access and manipulate all of its relevant data.

Creating and calculating results from data is the end goal of this project, but without internal organization, the data won't lead to a direct solution. It is not enough to have data arranged at a site level. The data must undergo standardization by using every recorded measurement at the test level. This is a fundamental requirement for an IVEM implementation. A "master" table that contains every recorded measurement will need to be made to calculate critical parameters. There also needs to be mechanisms to know when processed data is no longer accurate, and How? and When? these changes should occur or get updated. Finally, this table will have to be created or updated alongside each site data object on creation.

This project also aims to create an informative, yet easy to read graph. Providing a legend for the plotted regions, appropriate labels for each test, and sorted tabular results are an essential requirements for a general plot. However, unlike a general plot, IVEM expects and addresses zone ambiguity, so the plot will seek easy identification of these zones, the amount of uncertainty, and its overall significance. It is not an aim of the project to adjust the size of the graph, but it will provide a reasonably sized output for simplicity. Third-party tools might be used to satisfy this requirement.

Finally, general principles related to application development are important too. Development for cross-platform compatibility is a requisite not only graphical output formats, but benefits users on different platforms. A successful implementation also makes provision for expansion and experimental features, so development will also seek code modularity where it can be used most fittingly. Furthermore, in the application's creation opportunities for object oriented paradigms are hoped to be explored. One way to accomplish this goal is to keep "hard-coded" portions to a minimum. Another aim will explore C++ inheritance semantics. The project is intended to be a single use program and will not require anything outside of providing formatted data.

III. Methods & Materials

The project was approached and designed as a web service with three distinct processing layers. The first layer consists of the core IVEM implementation as a C++ back-end program. This program is the "last man" in the process chain and performs all the calculations to generate the final result. The second layer is an interface script. This interface script was written in PHP and acts as an interface layer. On one side, the script executes the back-end program passing a numerical parameter on how to proceed. On the other side, the script provides the graphical interface to the user. The third layer, a web browser, allows the user to send data files to the application via the PHP script, receive the results, and obscures the underlying complexity of the program. See the diagram below.

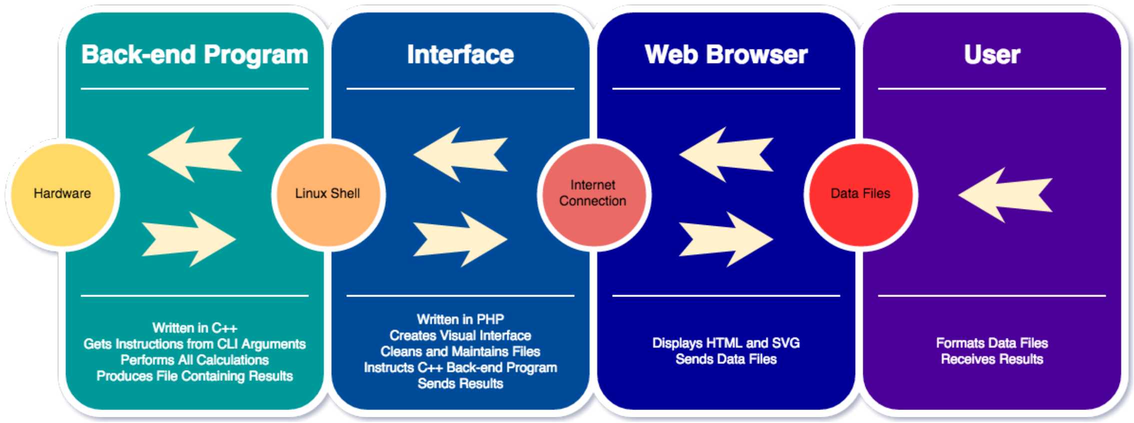

Figure 1. The user formats and provides data files via a web browser. The data files are received by a server that supplied the interface. The interface executes a back-end program to perform the calculations and create a report. When the program terminates, the script renders the report to the user.

Using web technologies avoided many platform-specific issues and greatly simplified the project. In this paradigm, the user is expected to use an HTML 5 conforming web browser rather than a specific device. Building IVEM this way did add an additional requirement, but a trivial one that enabled the application to be truly platform agnostic. It also ensured a uniform experience. Furthermore, this requirement affords IVEM's to be accessible on mobile devices (i.e field tablets) which was not initially considered.

PHP is a scripting language for creating dynamic web content and an excellent candidate for managing the web interface for IVEM. Websites such as Facebook use PHP to fetch and present information on a user-by-user basis (MacKay, et. al. 2017). For this project, PHP responsibilities were rather elementary: It handled user files, ran the backend program, and renders a page with the results. As pointed out by MacKay (et al., 2017), PHP's syntax has the familiarity of C++ which reduced the time needed to produce a working interface. Other technologies such as Scalable Vector Graphics (SVG) and Bootstrap were respectively used for the graphical plot and interface styling in conjunction with the PHP script.

All the code needed to facilitate the service was executed on a server. The PHP script allows a user to upload a file via the web browser and manages a counter to keep track of how many files it receives. Each file the user uploads represents an area of interest and the associated test data as stated in the guidelines. After receiving a file, each file is renamed consecutively starting at 0 (ex. "0.dat", then "1.dat" … and so on). When the script receives the command to process the data, it executes the C++ back-end program with a numeric parameter. For example, let's say '3' was passed as a parameter. This implies that there were three regions that were investigated and three files were uploaded by the user. The backend program now executes while the script waits for the program to complete. Because the script renamed all the user files consecutively, the backend program can expect the files 0.dat, 1.dat, and 2.dat to exist (no more than "3.dat"). The back-end program executes under this assumption, processes each file, and creates a result file before it terminates. Once the PHP script receives the termination signal from the back-end program, it checks for the existence of the result file and finishes the request by sending the results. A reset function deletes the user files, result file, and sets the file counter back to zero.

The backend program performs all the mathematical operations to produce a result. Those operations must be performed on data it retains in quickly accessible memory and not the files themselves; the program parse these files into an object which represents a dataset. The eponymously named object, a `Dataset` object, retains this data and provides an interface to control the data indirectly for operations such as adding, removing, organizing, querying, and manipulating those values as well as generating statistics about the data. A `Site` object retains `Dataset` objects, and `Site` objects interface with each other for instructions on how to manipulate their respective datasets. The underlying container to hold data in a `Dataset` is the STL's list class, and the `Dataset` object directly operates on this list. The STL is a well vetted, open source, ISO specified collection of tools that developers should use for best practices. The STL list was selected for its flexibility in data storage.

As mentioned earlier, each file submitted represents a sampled region and its test data. The backend program's internal representation of these sampled regions are called `SampledSite` objects and are created using C++'s inheritance system (`Site` is the parent class). As each file is parsed, the back-end program instantiates these `SampledSite` objects. Each object has its own properties that only pertain to the region. When the first `SampledSite` object is created, another special object called a `MasterSite` is also created using the inheritance system. As additional sites are added, their data is also merged into the `MasterSite`'s datasets. The `MasterSite` object therefore contains all data from all tests conducted at every sampled region. Once all data is read in, the data is sorted and information is needed for all `SampledSite` objects to normalize their data. The `MasterSite` object provides information on how to do so. Each `SampledSite` object standardizes its datasets using the

`MasterSite`'s information. Finally, with the data in a standardized form, results can be calculated and plotted.

The C++ back-end program needs to output a Scalable Vector Graphics image as part of its duties. Scalable Vector Graphics or SVG is a technology for developing high-quality images using vectors, lines, and shapes. Most, if not all, of SVG's shapes rely on numerical parameters. SVG's use a Cartesian plane coordinate system to render these shapes, but the representation of this plot is better suited in a polar coordinate system. For example, if ***n*** tests were conducted across different regions, then the angle for any endpoint would be some multiple of $\frac{2\pi}{n}$ on a polar system. This computes a θ value. The magnitude then becomes the test's mean or error values.

The resulting values can be connected to form a polygon. The resulting area inside that polygon represents the degree of damage at that region. Each polygon generated shares a common origin from which it was derived from: the origin in the polar system. Using this point, the angle formed from the two neighboring endpoints (θ from above), and the magnitude values as distances, the polygon can be split into triangles. The area for these triangular segments can be computed using the trigonometric side-angle-side formula:

$$Area = \frac{Test1 \cdot Test2 \, sin(\theta)}{2}$$

If we let the E be the standardized values for k tests, and let n represent the number of endpoints or tests, then algorithmically calculating the area of the polygon is just the sum of these triangles:

$$A = \sum_{k=2}^{n} \frac{E_{k-1} \cdot E_k \cdot sin(\frac{2\pi}{n})}{2}$$

This equation fails to calculate the area for a triangle between the n^{th} and 1^{st} endpoint. The missing "slice" can be calculated and added to compute the complete area of the polygon:

$$Area = A + \frac{E_1 \cdot E_n \cdot sin(\frac{2\pi}{n})}{2}$$

Using this approach, the area of all polygons can be computed iteratively. This process was applied for the mean as well as the upper and lower bound standard error polygons. Even though this method is an effective way to determine the area of a polygon, no actual polygon exists yet; the polygon is still a virtual abstraction of the data at this point.

Instead of using a 3rd party library to provide plotting, a less practical method was selected as an exercise. Text holding variables (called "strings") were used to create an actual SVG image from scratch. SVG follows a format that's similar to how HTML marks-up web pages; rather than defining a document pixel by pixel, an element or "tag" is used. Field/value pairs are used to describe the tag's attributes. For this project `Line` and `Path` were the predominately used tags. A possible `line` tag could look like this:

```
<Line x1="1" y1="2" x2="10" y2="20" stroke="black" />
```

The fields x1 and y1 define the starting (x, y) coordinate for the line, and fields x2 and y2 defining the ending (x, y) coordinate. Needless to say, stroke defines the color of the line. Other field/value pairs exist and can be used to provide additional attributes. In this example, a black line runs from the point (1, 2) to (10, 20) on the canvas of the SVG. It uses the top-left corner as the origin in a Cartesian system.

A simple way to adapt IVEM's data is to convert it to the Cartesian plane coordinate system. These conversions are fairly easy and the back-end program can do these conversions on the fly. The conversions are:

$$x = r \cdot cos\ (t \cdot \theta), \quad y = r \cdot sin\ (t \cdot \theta)$$

Using this approach, an (x, y) coordinate can be created from the statistical result (***r***) and a number, say ***t*** for the test number, to get the amount for the angle needed. The implementation present here used partially defined strings as templates and exploited the numerical fields in both `line` and `path` tags to create the shapes needed. For example, the line for an endpoint needs to span out using two pieces of information, an angle and a magnitude from some reference point and direction. Since these rotation and magnitude values were readily from the area calculations, ***r*** and $t \cdot \theta$, this data was reused. The (x, y) coordinate was calculated using the method described above to plot the shapes onto an SVG canvas. `line` tags were used to plot endpoints and the `path` tag was used for the legend, mean, and error bounds. A similar method was used on `text` tags for labeling. Tabular data followed the same approach, but used a table and the area results directly.

IV. Results

A case study using fictitious data was prepared and used to validate the IVEM implementation. Fabricated data was compiled by Dr. Ryan Otter as datasets that one might observe in the field (see Appendix A for the data used). This specifically included four contaminated zones and a fifth pre-disaster reference zone. Each zone included five different test sets:

- 7-ethoxy-resorufin *O*-deethylation (EROD), a liver enzyme commonly used as a pollutant marker found in pigs and fish (Zamaratskaia, Vladimir 2009). This test is measured in pmol/mg/min.
- A ground sediment test that measures the concentration of a contaminant in mg per kg of soil.
- A comet assay test that "...has applications in testing novel chemicals for genotoxicity, monitoring environmental contamination with genotoxins, human biomonitoring and molecular epidemiology..." (Collins 2004). This test measures DNA damage as a percentage.
- A gene test that measures the expression of a given gene in response to the environment (Choi and Sang 2007). The test is measured as a simple count of occurrences.
- A concentration test that includes a direct mg/kg concentration of the of a substance in fish.

Each test measured a completely different feature one might wish to observe in a contaminated region. It is important to note that while the sediment and fish test share the same unit of measurement (mg/kg), they were designed to simulate measuring different compounds with different concentration ranges. The rest of the tests have different units of measurements, but all tests cannot be directly compared against each other.

Most datasets contained ten data points per region. The sediment test included twenty data points with the "South Corner" including an additional two points. The

South Corner also included an additional two points in its gene test. The purpose of these additional data tested IVEM against uneven sampling in both test-to-test and region-to-region scenarios. For example, the sediment test contains twenty points where other tests contained ten (test-to-test: The amount of data in each test differs), and the South Corner contained two additional data points in both gene and sediment samples while others sites did not (region-to-region: The amount of data in a region differs from the others). Notice that South Corner specifically contained an additional two data in the sediment test as well which caused both region-to-region and test-to-test amounts to differ simultaneously. Finally, some data were purposely manipulated to be artificially high or low to increase variation which might result from equipment errors, bad data collection, or by sheer chance.

During development, the arithmetic mean, standard deviation, and standard error from the mean were calculated using the back-end program and checked against precomputed values for each region. With each calculation of raw data measuring up correctly, the data and descriptive statistics were standardized. If everything performed accordingly in the implementation, the program should predict normalized values for both data and statistics. Using each dataset as the final testing case, the implementation did predict accurate results. Furthermore, the program correctly predicted the area of each polygon, including the standard error boundaries from manual calculations. Previously, no working model provided a visual output so testing this project presented an opportunity to see a true to life example. Figure 2 is the result from the data contained in appendix A.

Region	+ Error	Mean	- Error
The Pond: West Side	16392.76	15189.77	14014.76
The Pond: East Side	7069.32	6587.14	6121.44
The Pond: North Corner	3282.43	2980.03	2690.35
The Pond: South Corner	386.23	318.18	255.88
The Pond: Reference	10.54	7.06	4.27

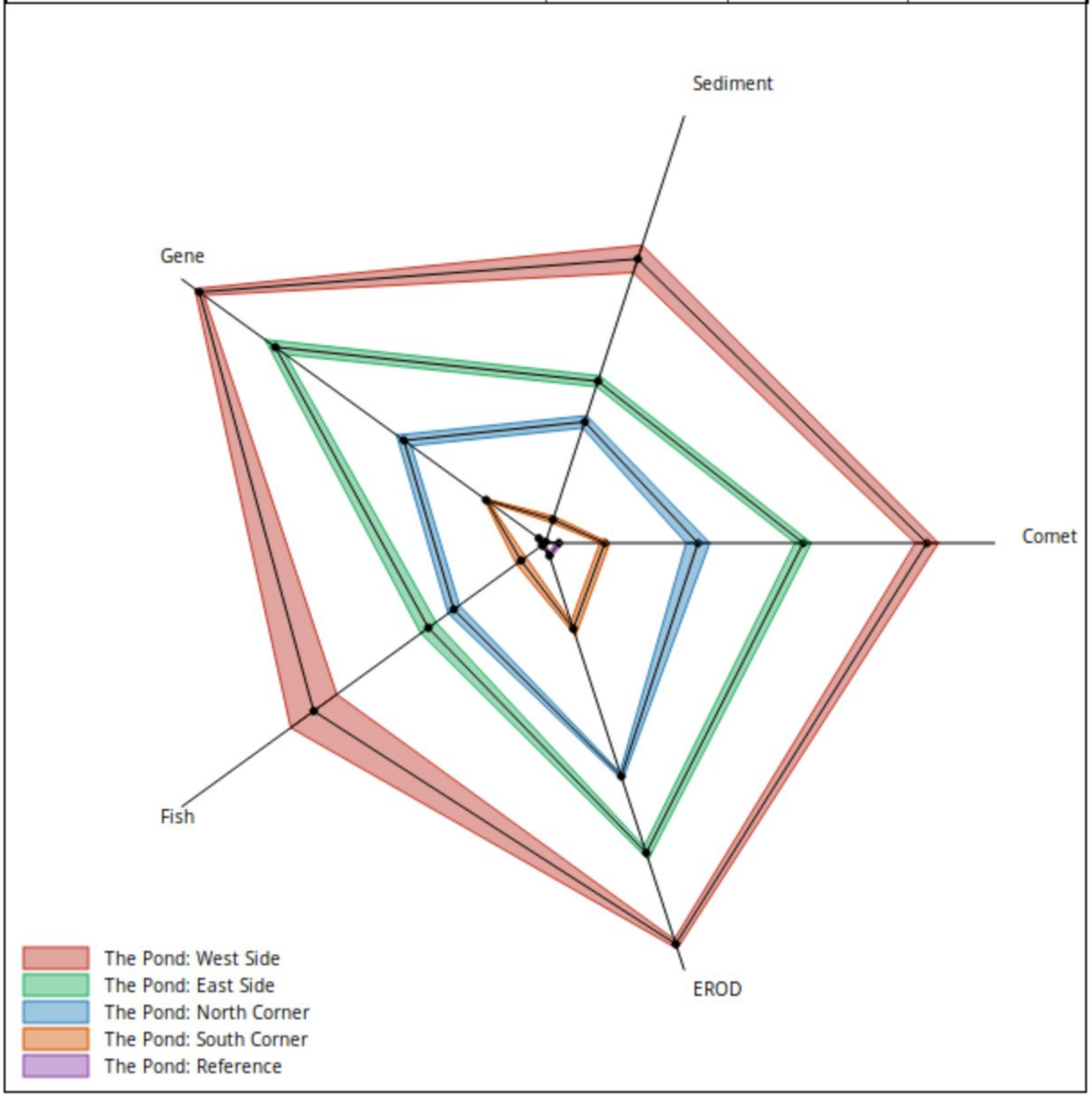

Figure 2. The graphical output that resulted from the data in Appendix A. Each zone was designed to be progressively worse than the last. Notice the large variance in The Pond: West Side's fish test as a result from higher variation. The reference site (inter rmost site) barely registers a polygon due in the result due to background contaminations.

V. Discussion

The application's functionality, interface, and graphical output are both intuitive and straightforward even for non-technical users. The resulting numerical data contains no unit of measurement, and can be thought of as a score where higher values signify worsening conditions. The numerical data is also sorted with respect to mean values and places the worst location at the top of the list. One does not need to know the statistics and math used to create the plot, but the result provides enough details to say that some amount of error can be expected both visually and numerically. Error boundaries are plotted with slightly darker colored lines surrounding the solid black line. This represents the mean measurement from the data with the standard error range filled-in between the colored boundary lines. The plot is designed such that someone can quickly gauge a test, set of tests, or a location as a whole and quickly determine its reliability despite the absence of numerical values. The color-filled error range is also semi transparent so overlapped regions are distinctly visible.

Each test ranged contained different value ranges and the best and worst test did not reach the outermost and innermost values on each endpoint. This was not expected, but after careful examination it was concluded this was an appropriate outcome. Each dataset was normalized using all data for a given test, but a dataset can only contain two extreme values because there can only be one maximum and minimum value in the entire collection. Even if the dataset contains a single entry, that entry is paradoxically both the minimum and the maximum value yet the test still contains a single extreme value.

Furthermore, if a test contains both the minimum and maximum value from the entire dataset, then it can only skew the mean and broaden the error boundary. Therefore plotted data will never reach values zero or one at the extremities of each endpoint. It was also not expected to see a sharp contrast between each site as well. Ancedoctaly manipulating the data to overlap more and more still resulted with plots tending to "pull the zones apart." This too was not expected, but again happens as data is normalized, the property mention above, and the result imposed onto each endpoint.

Moving away from a software prototype, other technologies such as the Python programming language would have dramatically simplified the development of IVEM into a more stable web application. An implementation in Python would merge both the web interface and IVEM program calculations into one cohesive application. Python has several hundred libraries (*Python Package Index*) that can add the functionality needed to replace the current prototype. For example, Python's Flask library is "a micro-framework" that is better suited web framework than the current PHP script. It would combine both the web server, interface, and the app itself into the same code. SciPy and NumPy ("Sci" for science, "Num" for numbers, and "py" for Python) offers an extensive collection of science related functions ("SciPy.org" 2018), and array/matrix data structure with commonly used statistical calculations ("NumPy." 2018). These libraries and methods were not known over the course of the projects development. However, the implementation did use C++'s Standard Template Library (STL) that follows best practices for code portability.

Future work to this implementation could remove some assumptions and limitations made at the model and code level. For example, the code made provisions for inverted data, but did not provide this functionality. In the case of inverted data, lower scores mark increasing damage (on the contrary, normal tests treat higher scores as increasing damage). Currently more testing and development needs to be conducted to remove this limitation. One assumption made is that all data scales linearly, but some tests such as the Richter Scale are non-linear. For example, a value of 3 is ten times the intensity of the value 2, and the value 2 is ten times the intensity of the value 1 in this non-linear scale. The impact of non-linear tests on IVEM is not known, and no work in this implementation exists for non-linear datasets. However, one could either scale the data before entering it, the application could correct it on the user's behalf, or it may simply be the case IVEM can handle non-linear datasets.

Appendix B shows the current file format with the letter "A" after each test name. This "A" designation is currently a fixed parameter and symbolizes that the dataset is in an ascending order (linear and non-inverted), however other letters might symbolize different data formats:

- **I:** Data is inverted and should be treated as such.
- **T:** TAB is the delimiter for each datum.
- **CSV:** The data follows a comma separated value format.
- **L<n>:** The data is in a logarithmic scale. An optional value *n* inside "<" and ">" could specify a custom base.
- **L<e>:** Data follows the natural logarithmic scale. Same as LN or Log_e

At the beginning of this article, an IVEM implementation was explored using common programming tools to develop a proof of concept. This article has shown IVEM implemented as a web application with a graphical interface which is easy to understand. Likewise the output is easy to understand as well. This was a critical part of the model because ultimately IVEM is targeted toward non-scientific entities where a simple but expressive result is needed. To date no implementation of IVEM existed, but this project offers a complete and working example that one might expect using curated but realistic data. This study did not implement some functionality, but instead presented IVEM as a usable, multi-platform, and a highly accessible application. From this prototype, future work should consider programming environments more suitable for web development and frameworks better suited for plotting graphs. A more reliable application would consider Python as a framework to solidify this modeling tool. In order for this project to be taken more seriously, real data could be used and compared with opinions from the scientific community with graphs presented to policy makers.

Appendix A

Raw data for each datasets that produced Figure 2 and associated arithmetic means, standard error, and standard deviation for each test. Standard deviation is based on the sample and not the standard deviation of the population.

The Pond: West Side

Fish (mg/kg)	Gene (count)	Comet (percent)	EROD (pmol/mg/min)	Sediment (mg/kg)	
14	409	42	269	75	145
34	450	57	300	120	98
26	400	50	290	80	97
18	424	47	280	75	86
17	438	46	270	110	112
18	449	43	275	101	104
29	418	51	285	89	103
22	444	49	282	76	100
14	431	47	276	156	100
20	419	53	292	135	101

	Fish	Gene	Comet	EROD	Sediment
Mean	21.200	428.200	48.500	281.900	103.150
Std. Dev.	2.086	17.100	4.528	9.994	22.260
Std. Err.	6.596	5.407	1.432	3.160	4.977

The Pond: East Side

Fish (mg/kg)	Gene (count)	Comet (percent)	EROD (pmol/mg/min)	Sediment (mg/kg)	
10	350	35	220	67	46
12	300	30	215	62	49
13	412	34	210	56	60
14	318	33	202	59	60
9	300	41	200	58	62
8	312	30	215	51	67
8	305	31	245	47	59
13	390	32	235	60	64
10	350	33	222	70	68
11	325	32	226	39	72

	Fish	Gene	Comet	EROD	Sediment
Mean	10.800	336.200	33.100	219.000	58.800
Std. Dev.	2.150	38.955	3.213	13.960	8.679
Std. Err.	0.680	12.319	1.016	4.415	1.941

The Pond: North Corner

Fish (mg/kg)	Gene (count)	Comet (percent)	EROD (pmol/mg/min)	Sediment (mg/kg)	
9	200	18	175	50	42
8	210	15	150	34	51
10	193	13	160	67	32
11	180	20	165	34	51
7	170	20	164	36	29
6	112	19	163	39	50
7	188	18	161	45	62
8	170	23	171	42	54
9	200	26	180	44	43
10	180	28	172	42	34

	Fish	Gene	Comet	EROD	Sediment
Mean	8.500	180.300	20.000	166.100	44.050
Std. Dev.	1.581	27.398	4.619	8.621	9.987
Std. Err.	0.500	8.664	1.461	2.726	2.233

The Pond: South Corner

Fish (mg/kg)	Gene (count)	Comet (percent)	EROD (pmol/mg/min)	Sediment (mg/kg)	
2	100	9	75	8	3
3	90	8	72	10	5
4	80	7	67	12	6
3	70	11	56	17	3
2	60	9	54	8	2
5	75	7	58	6	7
2	85	12	45	5	9
1	89	6	80	15	8
1	85	7	90	20	12
1	85	8	45	1	15
	75			3	5
	75				

	Fish	Gene	Comet	EROD	Sediment
Mean	2.400	80.750	8.400	64.200	8.600
Std. Dev.	1.350	10.550	1.897	15.083	5.205
Std. Err.	0.427	3.045	0.600	4.770	1.164

The Pond: Reference

Fish (mg/kg)	**Gene (count)**	**Comet (percent)**	**EROD (pmol/mg/min)**	**Sediment (mg/kg)**	
0.67	12	1	17	0.1	0.3
0.3	16	2	15	0.3	0.8
0.8	20	4	22	0.5	0.2
0.2	9	3	21	0.7	0.9
0.9	15	2	12	0.4	0.5
0.5	22	1	18	0.8	0.4
0.4	29	4	10	0.2	0.5
0.5	17	5	9	1.0	0.3
0.3	16	3	9	0.45	0.8
0.5	13	2	5	0.67	0.9

	Fish	**Gene**	**Comet**	**EROD**	**Sediment**
Mean	0.507	16.900	2.700	13.800	0.567
Std. Dev.	0.226	5.666	1.337	5.673	0.268
Std. Err.	0.071	1.792	0.423	1.794	0.060

Appendix B

Integrated Visualization & Endpoint Modeling File Specification

The Integrated Visualization & Endpoint Modeling (IVEM) uses plain text files for processing data. Text files with encodings (such as .DOC, .RTF, .ODT) will cause an error in processing and produce incorrect results or cause the program to crash. You can create input files on any operating system from commonly used programs including:

- **Windows machines:** Notepad, Notepad++.
- **macOS**: TextEdit may be used if you remove encode by pressing SHIFT + COMMAND + T or selecting Format > Make Plain Text from the global menu bar.
- **Cross Platform:** The Atom editor is recommend on any operating system if the file is saved with a '.txt' extension.

When creating input files, the first line represents the name or a region, site, or area of interest and must be unique. Each file represents a site or region with test data following afterwards. Test data must have a name, followed by capital letter 'A' on the next line. The unit of measurement follows the next line. The name will helps the user identify the test on the plot, the 'A' is reserved for future use, and the unit of measurement may be:

- **Single unit:** Specified directly such as Percent, Count, '%', or alpha.
- **Compound units:** No more than three units using a forward slash ('/') as a delimiter. For example mg/kg, or pmol/mg/kg.

The order in which test data is listed in the file does not matter – the application will resolve tests automatically. The letter 'A' must be placed at the end of the unit of measurement for all dataset if they are to be
compared. All test data enter below unit of measurement is delimited by a new line. Data may integers or rational numbers. The word 'END' is placed at the end of the data to signal the end of the dataset. The line after END must be blank before specifying a new test. Here is an example for testing two location using two test:

Filename: "ID 0123-4a.txt"

```
Cannonville Lake, South
EROD
A
pmol/mg/kg
13.2
12.3
. . .
15.2
15
END

Fish Contamination
A
Percent
3
4.2
. . .
3.8
12.3333
END
```

Filename: "123abc.dat"

```
Cannonville Lake, East
Fish Contamination
A
Percent
5.2
4.2
. . .
1.3
4.5
END

EROD
A
pmol/mg/kg
23.2
14.2
. . .
4
17.8
END
```

The file name and extension have no restrictions. You must supply three or more tests in order to

Appendix C

```
<<<<<< colors.dat
#27ae60
#2980b9
#8e44ad
#c0392b
#d35400
#f39c12
<<<<<< END


<<<<<< dataset-summary.cpp
//#include <boost/function.hpp>

#include "dataset-summary.h"
#include "dataset.h"

DatasetSummary::DatasetSummary(Dataset * dset)
     : m_dsref(dset) {
     d_init();
     m_dsref = (dset) ? dset : nullptr;
}


DatasetSummary::DatasetSummary(sDatasetPtr dset) {
     d_init();
     m_dsref = (dset.get()) ? dset.get() : nullptr;
}
```

```
void DatasetSummary::d_init() {
     m_testname = &Dataset::name;
     m_testid   = &Dataset::id;
     m_testunit = &Dataset::unit;

     m_inverted = &Dataset::inverted;
     m_n        = &Dataset::size;

     m_mean     = &Dataset::mean;
     m_stddev_p = &Dataset::stddev_p;
     m_stddev_s = &Dataset::stddev_s;
     m_stderr   = &Dataset::std_error;

     m_min      = &Dataset::minimum;
     m_q1       = &Dataset::q1;
     m_median   = &Dataset::median;
     m_q3       = &Dataset::q3;
     m_max      = &Dataset::maximum;

     m_iqr      = &Dataset::iqr;
     m_range    = &Dataset::range;
}


TestSummary::TestSummary(Dataset* dset, SampledSite* site)
     : DatasetSummary(dset) {
     t_init();
```

```
    m_siteref = (site) ? site : nullptr;
}


TestSummary::TestSummary(sDatasetPtr dset, SampledSite* site)
: DatasetSummary(dset) {
    t_init();
    m_siteref = (site) ? site : nullptr;
}


void TestSummary::t_init() {
    m_n_upbound          = &Dataset::n_upper_bound;
    m_n_mean             = &Dataset::norm_mean;
    m_n_lwbound          = &Dataset::n_lower_bound;

    m_standardized       = &Dataset::standardized;
    m_normalize          = &Dataset::normalize;
}
<<<<<< END


<<<<<< dataset-summary.h
#ifndef DATASUMMARIES_H
#define DATASUMMARIES_H
/*
#include <string>
#include <memory>
```

```cpp
#include <boost/function.hpp>

#include "dataset.h"
#include "sitesample.h"

class Site;
class DatasetSummary;

typedef std::shared_ptr<Dataset> sDatasetPtr;
typedef DatasetSummary * DatasetSummaryPtr;

using std::string;
using boost::function;

// Standard info thats in every dataset should, to be handed off to a site.
// this acts as a thin layer between accessors up the class structure.
class DatasetSummary {
public:
     DatasetSummary() {};
     DatasetSummary(Dataset * dset);
     DatasetSummary(sDatasetPtr dset);

     string test_name() const   { return (m_dsref) ? m_testname(m_dsref)    : ""; }
     int test_id() const        { return (m_dsref) ? m_testid(m_dsref)      : -1; }
     string test_unit() const   { return (m_dsref) ? m_testunit(m_dsref)    : ""; }

     bool inverted()  const     { return (m_dsref) ? m_inverted(m_dsref)    : false; }
     int n() const              { return (m_dsref) ? m_n(m_dsref)           : 0; }
```

```
    double mean() const        { return (m_dsref) ? m_mean(m_dsref)       : 0; }
    double std_dev_s() const   { return (m_dsref) ? m_stddev_s(m_dsref)   : -1; }
    double std_dev_p() const   { return (m_dsref) ? m_stddev_p(m_dsref)   : -1; }
    double std_err() const     { return (m_dsref) ? m_stderr(m_dsref)     : -1; }

    double minimum() const     { return (m_dsref) ? m_min(m_dsref)        : 0; }
    double q1() const          { return (m_dsref) ? m_q1(m_dsref)         : 0; }
    double median() const      { return (m_dsref) ? m_median(m_dsref)     : 0; }
    double q3() const          { return (m_dsref) ? m_q3(m_dsref)         : 0; }
    double maximum() const     { return (m_dsref) ? m_max(m_dsref)        : 0; }
    double iqr() const         { return (m_dsref) ? m_iqr(m_dsref)        : 0; }

    double range() const       { return (m_dsref) ? m_range(m_dsref)      : 0; }

protected:
    Dataset* m_dsref;

    // Dataset hooks
    function<string(Dataset*)> m_testname;
    function<int(Dataset*)>    m_testid;
    function<string(Dataset*)> m_testunit;

    function<bool(Dataset*)>   m_inverted;
    function<int(Dataset*)>    m_n;

    function<double(Dataset*)> m_mean;
    function<double(Dataset*)> m_stddev_s;
```

```
    function<double(Dataset*)> m_stddev_p;
    function<double(Dataset*)> m_stderr;

    function<double(Dataset*)> m_min;
    function<double(Dataset*)> m_q1;
    function<double(Dataset*)> m_median;
    function<double(Dataset*)> m_q3;
    function<double(Dataset*)> m_max;

    function<double(Dataset*)> m_iqr;
    function<double(Dataset*)> m_range;

private:
    // Helper Methods
    void d_init();
};


// extend the DatasetSummary class to include site specific data.
class TestSummary
    : public DatasetSummary {

public:
    TestSummary(Dataset*, SampledSite*);
    TestSummary(sDatasetPtr, SampledSite*);

    string site_name() const   { return (m_siteref) ? m_sitename(m_siteref)     : ""; }
    int site_id() const        { return (m_siteref) ? m_siteid(m_siteref)       : -1; }
```

```
    double upper_bound() const  { return (m_dsref) ? m_n_upbound(m_dsref)        : 0; }
    double norm_mean() const    { return (m_dsref) ? m_n_mean(m_dsref)           : 0; }
    double lower_bound() const  { return (m_dsref) ? m_n_lwbound(m_dsref)        : 0; }

    bool standardized() const   { return (m_dsref) ? m_standardized(m_dsref)     : false; }
    void normalize() const      { if (m_dsref) m_normalize(m_dsref); }

protected:

private:
    void t_init();
    Site* m_siteref;

    // Test & Site Specific hooks
    function<double(Dataset*)> m_n_upbound;
    function<double(Dataset*)> m_n_mean;
    function<double(Dataset*)> m_n_lwbound;
    function<bool(Dataset*)>   m_standardized;
    function<void(Dataset*)>   m_normalize;

    // Site Specific
    function<string(Site*)>    m_sitename;
    function<int(Site*)>       m_siteid;

    function<double(Site*)>    m_lwe_outcome;
    function<double(Site*)>    m_est_outcome;
    function<double(Site*)>    m_upe_outcome;
```

```
};
*/
#endif // !DATASUMMARIES_H
<<<<<< END


<<<<<< dataset.cpp
// Preprocessor System Directives
#include <iostream>
#include <cmath>
#include <iomanip>
#include <string>

// User Defined Directives
#include "dataset.h"
#include "types.h"

// Using directives
using std::cout;
using std::endl;
using std::cin;
using std::setw;
using std::setprecision;
using std::abs;

// Implementation

// Constructors & Destructor
```

```
Dataset::Dataset() {
     m_size = 0;
     m_unitcnt = 0;

     init_stats();
     stale_stats();

     m_inverted = false;
     m_normtype = SELF;
}


Dataset::Dataset(bool descending) {
     m_size = 0;
     m_unitcnt = 0;

     init_stats();
     stale_stats();

     m_inverted = descending;
     m_normtype = SELF;
}


Dataset::Dataset(const Dataset &to_copy) {
     m_data = to_copy.m_data;
     m_size = to_copy.m_data.size();
     m_inverted = to_copy.m_inverted;
```

```
    m_name = to_copy.m_name;
    unit_setter(to_copy.unit());

    init_stats();
    stale_stats();

    m_inverted = to_copy.m_inverted;
    m_normtype = to_copy.m_normtype;
    //calc_stats();
}


Dataset::Dataset(vector<double> data, string name, string unit, bool inverted) {
    vector<double>::const_iterator iter;
    DataNode node;

    name_setter(name);
    unit_setter(unit);

    init_stats();

    m_inverted = inverted;
    m_normtype = SELF;

    node.norm_value = 0;
    for (double d : data) {
        node.value = d;
```

```
            m_data.push_back(node);
        }

        m_size = m_data.size();
        stale_stats();
        sortdata();
        normalize();
        rank();
}


Dataset::~Dataset() {

}


// Public Observers
string Dataset::unit() const {
        string ret_string = m_units[0];

        if (m_unitcnt > 1) {
            ret_string += "/";
            ret_string += m_units[1];
        }

        if (m_unitcnt > 2) {
            ret_string += "/";
            ret_string += m_units[2];
```

```
    }
    return ret_string;
}

// Main Interface
void Dataset::print() {
    cout << "Dataset: " << Dataset::name();

    if (m_normtype == EXTERNAL)
        cout << " (STANDARDIZED)";

    cout << endl;

    cout << "Values |     Unit |   Rank |  N. Value |" << endl;

    for(DataNode data : m_data ) {
        cout << setw(6) << data.value;
        cout << " | " << setw(8) << unit() << " | ";
        cout << setw(6) << data.rank << " | ";
        cout << setw(9) << data.norm_value << " |" << endl;
    }
    cout << endl;

    cout << setw(12) << "Mean: ";
    cout << setprecision(6) << setw(7) << mean();
    cout << setw(14) << "St Dev: ";
    cout << setprecision(6) << stddev_s();
    cout << endl;
```

```
    cout << setw(12) << "N. L Err: ";
    cout << setprecision(6) << setw(7) << Dataset::n_lower_bound();
    cout << setw(14) << "St Err: ";
    cout << setprecision(6) << Dataset::std_error();
    cout << endl;

    cout << setw(12) << "Nm. Mean: ";
    cout << setprecision(6) << Dataset::norm_mean();
    cout << endl;

    cout << setw(12) << "N. U Err: ";
    cout << setprecision(6) << Dataset::n_upper_bound();
    cout << endl;
    cout << setw(12) << "Range: " << Dataset::range() << endl;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
}


double Dataset::basis() {
    if (stale_basis) {
        Dataset::c_basis();
        stale_basis = false;
    }
    return m_basis;
}
```

```
double Dataset::mean() const  {
     if (stale_mean) {
          Dataset::c_mean();
          stale_mean = false;
     }
     return m_mean;
}


double Dataset::median() const {
     if (stale_median) {
          Dataset::c_median();
          stale_median = false;
     }
     return m_median;
}


double Dataset::iqr() const {
     return abs(q3() - q1());
}


double Dataset::q1() const  {
     if (stale_q1) {
          Dataset::c_q1();
          stale_q1 = false;
     }
```

```
        return m_q1;
}

double Dataset::q3() const {
        if (stale_q3) {
                Dataset::c_q3();
                stale_q3 = false;
        }
        return m_q3;
}

double Dataset::minimum() const {
        if (m_inverted) {
                if (stale_max) {
                        c_max();
                        return m_intmax;
                }
        } else if (stale_min)
                c_min();

        return (m_inverted) ? m_intmax : m_intmin;
}

double Dataset::maximum() const {
        if (m_inverted) {
```

```
        if (stale_min) {
            c_max();
            c_min();
            return m_intmax;
        }
    } else if (stale_max)
        c_max();

    return (m_inverted) ? m_intmin : m_intmax;
}


double Dataset::stddev_p() const {
    if (stale_stddev_p) {
        Dataset::c_stddev_p();
        stale_stddev_p = false;
    }
    return m_stddev_p;
}


double Dataset::stddev_s() const {
    if (stale_stddev_s) {
        Dataset::c_stddev_s();
        stale_stddev_s = false;
    }
    return m_stddev_s;
}
```

```
double Dataset::std_error() const {
     if (stale_stderror) {
          Dataset::c_stderr();
          stale_stderror = false;
     }
     return m_stderr;
}


double Dataset::sum() const {
     if (stale_sum) {
          Dataset::c_sum();
          stale_sum = false;
     }
     return m_sum;
}


double Dataset::up_mean_err() {
     if (m_inverted)
          return mean() - std_error();
     else
          return mean() + std_error();
}
```

```
double Dataset::lw_mean_err() {
     if (m_inverted)
          return mean() + Dataset::std_error();
     else
          return mean() - Dataset::std_error();
}


// Public normalized/standardized data access
double Dataset::norm_mean() {
     double value = mean() - m_normval;
     return abs(value / m_normbasis);
}


double Dataset::norm_median() {
     double value = median() - m_normval;
     return abs(value / m_normbasis);
}


double Dataset::norm_q1() {
     double value = q1() - m_normval;
     return abs(value / m_normbasis);
}


double Dataset::norm_q3() {
```

```
    double value = q3() - m_normval;
    return abs(value / m_normbasis);
}


double Dataset::n_upper_bound() {
    double value = up_mean_err() - m_normval;
    return abs(value / m_normbasis);
}


double Dataset::n_lower_bound() {
    double value = lw_mean_err() - m_normval;
    return abs(value / m_normbasis);
}

//  Public Mutators
void Dataset::normalize() {
    m_normtype = SELF;
    norm_helper();
}


void Dataset::standardize(double ref_value, double ref_basis) {
    m_normtype = EXTERNAL;
    m_normval = ref_value;
    m_normbasis = ref_basis;
    norm_helper();
```

```
}


void Dataset::insert(double to_insert) {
     DataNode ins;

     ins.value = to_insert;
     ins.norm_value = 0;

     // TODO:  Insert in place to avoid whole set sorting: O(N) vs O(n log n)
     m_data.push_back(ins);
     sortdata();
     m_size++;
     rank();
     stale_stats();
     norm_helper();
}


void Dataset::merge(const Dataset &_merge) {
     list<DataNode>::const_iterator iter = _merge.m_data.begin();
     for (; iter != _merge.m_data.end(); iter++)
          this->m_data.push_back(*iter);

     m_data.sort([](const DataNode & a, const DataNode & b) { return a.value < b.value; });
     m_size = m_data.size();
     stale_stats();
     rank();
```

```
    norm_helper();
}


void Dataset::remove(int pos) {
    // NYI
}


void Dataset::rank() {
    list<DataNode>::iterator iter = m_data.begin();
    int rank = 1;
    for (; iter != m_data.end(); iter++, rank++)
        iter->rank = rank;
}


void Dataset::unit_setter(string unit) {
    char reader;
    m_unitcnt = 1;

    for (size_t i = 0; i < unit.size(); i++) {
        reader = unit.at(i);

        if (reader == ' ')
            continue;
        else if (reader == '/') {
            m_unitcnt++;
```

```
                continue;
            }

            if (m_unitcnt > 3)
                break;

            m_units[m_unitcnt - 1] += reader;
        }
}


void Dataset::calc_stats() {
    stale_stats();
    c_basis();
    c_sum();
    c_mean();
    c_median();
    c_q1();
    c_q3();
    c_stddev_s();
    c_stddev_p();
    c_stderr();
}


// private
void Dataset::c_mean() const {
    if (m_size != 0)
```

```
            m_mean = sum() / m_size;
        else
            m_mean = std::numeric_limits<double>::quiet_NaN();
}


void Dataset::c_stddev_s() const {
        list<DataNode>::const_iterator iter = m_data.begin();

        double* nums = new double[m_size];
        double tempsum = 0;
        c_mean();

        for (int i = 0; iter != m_data.end(); iter++, i++)
            nums[i] = pow(iter->value - m_mean, 2);

        for (int i = 0; i < m_size; i++)
            tempsum += nums[i];

        tempsum /= (m_size - 1);
        m_stddev_s = sqrt(tempsum);
        delete[] nums;
}


void Dataset::c_stddev_p() const {
        list<DataNode>::const_iterator iter = m_data.begin();
```

```
    double* nums = new double[m_size];
    double tempsum = 0;
    c_mean();

    for (int i = 0; iter != m_data.end(); iter++, i++)
        nums[i] = pow(iter->value - m_mean, 2);

    for (int i = 0; i < m_size; i++)
        tempsum += nums[i];

    tempsum /= (m_size);
    m_stddev_p = sqrt(tempsum);
    delete[] nums;
}


void Dataset::c_stderr() const {
    m_stderr = stddev_s() / (sqrt(m_size));
}


// m_min will hold the absolute minimum in the set
void Dataset::c_min() const  {
    if (m_inverted)
        m_intmin = m_data.rbegin()->value;
    else
        m_intmin = m_data.begin()->value;
    stale_min = false;
```

```
}

void Dataset::c_q1() const {
      bool plus_one_form = false;
      int n, midpoint;
      double temp_q1;
      list<DataNode>::const_iterator iter = m_data.begin();

      if (m_size % 2) {                                              // odd length total set
            if (((m_size - 3) % 4)) {
                  n = (m_size - 1) / 4;
                  n--;
                  plus_one_form = true;
            }
            else                                                     // of 4n+1
                  n = (m_size - 3) / 4;

            if (plus_one_form) {
                  for (int k = 0; k < n; k++)
                        iter++;

                  temp_q1 = (iter->value * .25);
                  iter++;
                  m_q1 = temp_q1 + ((iter->value) * .75);
            } else {
                  for (int k = 0; k < n; k++)
                        iter++;
```

```
                temp_q1 = (iter->value * .75);
                iter++;
                m_q1 = temp_q1 + ((iter->value) * .25);
            }
        } else {                                                        // even length
            midpoint = m_size / 2;
            if (midpoint % 2) {                                         // odd n on lower half
                midpoint /= 2;

                for (int k = 0; k < midpoint; k++)
                    iter++;

                m_q1 = iter->value;
            } else {
                midpoint /= 2;
                midpoint--;

                for (int k = 0; k < midpoint; k++)
                    iter++;

                temp_q1 = iter->value;
                iter++;
                temp_q1 += iter->value;
                m_q1 = (temp_q1 /= 2);
            }
        }
        stale_q1 = false;
```

```
}


void Dataset::c_median() const {
     list<DataNode>::const_iterator iter = m_data.begin();
     int midpoint = (m_size / 2);
     bool is_odd = (m_size % 2 == 1) ? true : false;

     if (!is_odd)
          midpoint--;

     for (int i = 0; i < midpoint; i++)
          iter++;

     if (is_odd) {
          m_median = iter->value;
     }
     else {
          m_median = iter->value;
          iter++;
          m_median = (iter->value + m_median) / 2;
     }
}


void Dataset::c_q3() const {
     bool plus_one_form = false;
     int n, midpoint;
```

```
double temp_q3;
list<DataNode>::const_reverse_iterator iter = m_data.rbegin();

if (m_size % 2) {                                   // odd length total set
     if ((m_size - 3) % 4) {                        // of the form 4n+1? (then mod 4 == true)
          n = (m_size - 3) / 4;                     // swapped
          plus_one_form = true;
     }
     else                                           // 4n+3 form
          n = (m_size - 1) / 4;

     if (plus_one_form) {
          for (int k = 0; k < n; k++)
               iter++;

          temp_q3 = (iter->value * .25);                 // Apply weights
          iter++;
          m_q3 = temp_q3 + ((iter->value) * .75);     // Set Q3
     } else {
          for (int k = 0; k < n; k++)
               iter++;

          temp_q3 = (iter->value * .75);
          iter++;
          m_q3 = temp_q3 + ((iter->value) * .25);
     }
} else {                                            // even length
     midpoint = m_size / 2;
```

```
            if (midpoint % 2) {                           // odd n on lower half
                  midpoint /= 2;

                  for (int k = 0; k < midpoint; k++)
                        iter++;

                  m_q3 = iter->value;
            } else {                                      // even n on lower half
                  midpoint /= 2;
                  midpoint--;

                  for (int k = 0; k < midpoint; k++)
                        iter++;

                  temp_q3 = iter->value;
                  iter++;
                  temp_q3 += iter->value;
                  m_q3 = (temp_q3 /= 2);
            }
      }
      stale_q3 = false;
}

// m_max will hold the absolute maximum in the set
void Dataset::c_max() const  {
      if (m_inverted)
            m_intmax = (m_data.begin())->value;
      else
```

```
            m_intmax = (m_data.rbegin())->value;
      stale_max = false;
}


void Dataset::c_sum() const {
      m_sum = 0;
      list<DataNode>::const_iterator iter = m_data.begin();
      for (; iter != m_data.end(); iter++)
            m_sum += iter->value;
      stale_sum = false;
}


void Dataset::init_stats() {
      m_normval = 0;
      m_normbasis = 0;

      m_basis = 0;
      m_iqr = 0;
      m_mean = 0;
      m_median = 0;
      m_q1 = 0;
      m_q3 = 0;
      m_intmin = 0;
      m_intmax = 0;
      m_stddev_s = 0;
      m_stddev_p = 0;
```

```
    m_stderr = 0;
    m_sum = 0;
}

void Dataset::stale_stats() {
    stale_basis = true;
    stale_mean = true;
    stale_median = true;
    stale_max = true;
    stale_min = true;
    stale_q1 = true;
    stale_q3 = true;
    stale_stddev_p = true;
    stale_stddev_s = true;
    stale_stderror = true;
    stale_sum = true;
}

void Dataset::sortdata() {
    if (m_inverted)
        m_data.sort([](const DataNode & lhs, const DataNode & rhs) { return lhs.value >
rhs.value; });
    else
        m_data.sort([](const DataNode & lhs, const DataNode & rhs) { return lhs.value <
rhs.value; });
}
```

```
void Dataset::norm_helper() {
    if (m_normtype == SELF) {
        m_normbasis = basis();
        if (m_inverted) {
            c_max();
            m_normval = m_intmax;
        }
        else {
            c_min();
            m_normval = m_intmin;
        }
    }

    if (m_inverted) {
        for (DataNode d : m_data)
            d.norm_value = abs((d.value - m_normval) / m_normbasis);
    }
    else {
        for (DataNode d : m_data)
            d.norm_value = (d.value - m_normval) / m_normbasis;
    }
}


void Dataset::c_basis() {
    m_basis = Dataset::range() / 100;
```

```
        if (m_basis == 0)
            m_basis = 1;                          // Don't divide by zero

        stale_basis = false;
}
<<<<<< END


<<<<<< dataset.h
#ifndef DATALIST_H
#define DATALIST_H

// External Preproccessor Directives
#include <string>
#include <memory>
#include <list>
#include <vector>

// Internal Preprocessor Directives
#include "types.h"

// Using Statements
using std::shared_ptr;
using std::list;
using std::vector;
using std::string;
```

```
// Forward Declaritions and typedef's
class Dataset;
struct DataNode;
typedef std::shared_ptr<Dataset> sDatasetPtr;
typedef Dataset* DatasetPtr;

class Dataset {
public:
     // Constructors, Destructor
     Dataset();
     Dataset(bool descending);
     Dataset(const Dataset &to_copy);
     Dataset(sDatasetPtr to_copy) : Dataset(*to_copy) {};
     Dataset(vector<double> data, string name, string unit, bool descending = true);
     ~Dataset();

     // Strict Observers
     bool empty() const { return (m_size == 0); }     // Returns if list is empty
     int size() const { return m_size; }              // Returns num of elements in list

     bool inverted() const { return m_inverted; }
     bool standardized() const { return (m_normtype == EXTERNAL); }   //
     string name() const { return m_name; }           // Returns name of dataset
     int id() const { return m_id; }
     string unit() const;                             // Returns unit of dataset

     // Amendable Observers
     double up_mean_err();                        // returns mean + std error
```

```
double mean() const;                        // returns mean
double lw_mean_err();                       // returns mean - std error

double stddev_p() const;                    // returns std dev of a population
double stddev_s() const;
double std_error() const;                   // returns std error of the mean

double minimum() const;                     // returns smallest value
double q1() const;                          // returns first quartile
double median() const;                      // returns median
double q3() const;                          // returns third quartile
double maximum() const;                     // returns highest value
double iqr() const;                         // returns Q3 - Q1

// Observer Mutators
void print();                           // Modifies stat values
double basis();                         // returns basis_unit: Range / 100

double range() { return abs(maximum() - minimum()); }  // returns max - min

double n_upper_bound();
double norm_mean();
double n_lower_bound();

double norm_q1();
double norm_median();
double norm_q3();
double norm_iqr() { return abs(norm_q3() - norm_q1()); }
```

```
    double sum() const;                                  // returns the sum of the values

    // Mutators
    void name_setter(string name) { m_name = name; }      // Set the name of the dataset
    void id_setter(int _id) { m_id = _id; }
    void unit_setter(string units);                      // Set the unit of the dataset

    void insert(double to_insert);
    void insert(const DataNode &to_insert) { insert(to_insert.value); }
    void merge(const Dataset &merge);
    void merge(sDatasetPtr _merge) { merge(*_merge); } // Merge & Sort a DataSet by shared_ptr
    void remove(int position);                           // Remove data (*NYI*)

    void normalize();
    void standardize(double ref_value, double ref_basis);

    void calc_stats();                                   // Refresh all internal stats

private:
    mutable bool stale_mean;                             // Stale Data flags
    mutable bool stale_stddev_p;
    mutable bool stale_stddev_s;
    mutable bool stale_stderror;

    mutable bool stale_min;
    mutable bool stale_q1;
    mutable bool stale_median;
```

```
        mutable bool stale_q3;
        mutable bool stale_max;

        bool stale_basis;

        mutable bool stale_sum;
        bool m_inverted;

        int m_size;                                          // Number of items in this list
        int m_unitcnt;
        unsigned int m_id;

        double m_basis;                                 // Internal observer data references
        double m_normval;                               // value to norm. / stdz with
        double m_normbasis;                             // basis value to norm/stdz with

        mutable double m_intmin;
        mutable double m_intmax;

        mutable double m_mean;
        mutable double m_stddev_p;
        mutable double m_stddev_s;
        mutable double m_stderr;

        mutable double m_q1;
        mutable double m_median;
        mutable double m_q3;
        double m_iqr;
```

```
    mutable double m_sum;

    string m_name;                   // The name or type of test conducted
    string m_units[3];               // Stores compound units (mg/L/hr)

    NORM_T m_normtype;

    list<DataNode> m_data;                                  // The dataset

    // Helper Methods
    void c_mean() const;
    void c_stddev_s() const;
    void c_stddev_p() const;
    void c_stderr() const;

    void c_min() const;
    void c_q1() const;
    void c_median() const;
    void c_q3() const;
    void c_max() const;

    void rank();
    void c_basis();      // Calcs basis_unit or uses basis_override if normalization_override

    void c_sum() const;
    void stale_stats();                                     // Flag resetter
    void init_stats();
```

```
    void norm_helper();
    void sortdata();
};
#endif
<<<<<< END
```

```
<<<<<< ETP-main.cpp
// Preprocessor Directives
#include <iostream>
#include <vector>
#include <list>
#include <iomanip>
#include <fstream>
#include <cassert>
#include <string>
#include <time.h>

// User Defined Directives
#include "types.h"
#include "dataset-summary.h"
#include "dataset.h"
#include "site.h"
#include "sitesample.h"
#include "sitemaster.h"
#include "integrator.h"
#include "plotter.h"
#include "box-whisker.h"
```

```
// Using statements
using std::cout;
using std::string;
using std::vector;
using std::list;
using std::cin;
using std::endl;
using std::setprecision;
// Function Prototypes

// Main
int main() {

    clock_t begin, end;
    double time_spent;

    vector<string> paths;
    paths.push_back("site1.dat");
    paths.push_back("site2.dat");
    paths.push_back("site3.dat");
    //paths.push_back("site4.dat");
    //paths.push_back("site5.dat");
    //paths.push_back("site3.dat");

    begin = clock();
    Integrator inte(paths);
    inte.configure();
```

```
        inte.show_results();
        end = clock();

        //string test = string();
        //BoxWhisker::box_whisker(test, 20, 40, 70, 85, 88, 300);
        //cout << test;

        time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
        cout << "done in: " << time_spent << " seconds" << endl;
        return 0;
}
<<<<<< END


<<<<<< HTMLTable.cpp
#include "HTMLTable.h"
/*
using std::to_string;

HTMLTable::HTMLTable(int rows, int cols, string id)
        : m_rows(rows), m_cols(cols) {

        m_table.reserve(rows + 1);
        m_headerset = false;

        m_buffer = "<table id='" + id + "'>\n";
        for (int i = 1; i <= cols; i++) {
                m_buffer += "<colgroup class='" + id + "-" + to_string(i) + "'>< / colgroup>\n";
```

```
        }
};

HTMLTable::~HTMLTable() {
}

void HTMLTable::set_header(const list<string> & headings) {
//      _html += "<thead>\n";
//      _html += "<tr class='header'>\n";
//      _html += "<th class='header-label'><span>Region</span></th>\n";
//      _html += "<th class='header-label'><span>+ Error</span></th>\n";
//      _html += "<th class='header-label'><span>Mean</span></th>\n";
//      _html += "<th class='header-label'><span>- Error</span></th>\n";
//      _html += "</tr>\n";
//      _html += "</thead>\n";

        m_table[0] = headings;
}

void HTMLTable::set_row(int row, const list<string> & entries) {
        if (row > m_rows || row == 0)
```

```
            return;
      else
            m_table[row] = entries;
}


void HTMLTable::string_entry(string entry, string & out) {
      out.clear();
      out += "<td class='site-name'>";
      out += entry;
      out += "</td>\n";
}
*/
<<<<<< END


<<<<<< HTMLTable.h
#ifndef HTML_TABLE_H
#define HTML_TABLE_H
/*
#include <string>
#include <vector>
#include <list>


using std::string;
using std::vector;
using std::list;
```

```
class HTMLTable {
public:
    HTMLTable() {};
    HTMLTable(int rows, int cols, string id);

    ~HTMLTable();

    void set_header(const list<string> & headings);
    void set_row(int row, const list<string> & entries);

    static void string_entry(string entry, string & out);
    template <typename T> static void number_entry(T num, string & out) {
        string temp = to_string(num);
        std::size_t pos;

        out.clear();
        out += "<td class='num-result'>";
        pos = temp.find('.');
        if (pos != string::npos) {
            pos += 3;
            temp = temp.substr(0, pos);
        }
        out += temp;
        out += "</td>\n";
    }

private:
```

```
     bool m_headerset;
     int m_rows;
     int m_cols;

     string m_buffer;
     vector<list<string>> m_table;
};

*/
#endif
<<<<<< END


<<<<<< integrator.cpp
#define _USE_MATH_DEFINES

// System Directives
#include<math.h>
#include<iomanip>

// User Defined Directives
#include "integrator.h"
#include "plotter.h"
#include "sitesample.h"
#include "sitemaster.h"

// Using Statements
using std::cout;
```

```
using std::endl;
using std::sin;
using std::setw;
using std::setprecision;

// Take a vector of file paths, create the integrator object that
// holds all the sites, the datasets of those sites, and creates a
// master listing from those sites.
Integrator::Integrator(const vector<string> &paths) {
     vector<s_SampledSitePtr>::const_iterator sites_iter;
     shared_ptr<SampledSite> sampled_site;

     m_sitecount = 0;
     stale_conf = true;

     for (const string path : paths) {
          sampled_site = s_SampledSitePtr(new SampledSite(path));
          if (find_site(sampled_site->name()) == -1) {
               m_sitenames.push_back(sampled_site->name());
               m_sites.push_back(sampled_site);

               m_sitecount++;
          }
          sampled_site.reset();
     }
     // Constructor MASTER reference.
     sites_iter = m_sites.begin();
     m_master = s_MasterSitePtr(new MasterSite(**sites_iter));
```

```
    sites_iter++;
    while (sites_iter != m_sites.end()) {
        m_master->add_data(**sites_iter);
        sites_iter++;
    }
    m_usable_ids.reserve(m_master->test_count());

    assign_ids();
}


Integrator::~Integrator() {
    m_sites.clear();
    m_sitenames.clear();

    m_master.reset();
}


void Integrator::print_sites() const {
    for (const s_SampledSitePtr site : m_sites)
        site->print_tests();
}


void Integrator::print_site(string site_name) const {
    for (const s_SampledSitePtr site : m_sites) {
        if (site->name() == site_name) {
```

```
                site->print_tests();
                break;
            }
        }
}


int Integrator::get_testcount() {
    if (stale_conf)
        configure();

    return m_master->test_count();
}

// Mutators

// standardize every test in a site
void Integrator::standardize(string site_name) {
    for (const DatasetPtr test : m_master->getMasterSets())
        standardize(site_name, test->name());
}


void Integrator::standardize(string site_name, string name_setter) {
    vector<s_SampledSitePtr>::iterator site_iter = m_sites.begin();
    DatasetPtr refer = m_master->get_test(name_setter);

    double r_value = refer->minimum();
```

```
    double r_basis = refer->basis();

    while (site_iter != m_sites.end()) {
        if ((*site_iter)->name() == site_name) {

            (*site_iter)->standardize(name_setter, r_value, r_basis);
            break;
        }
        site_iter++;
    }
}


void Integrator::normalize(string site_name, string name_setter) {
    vector<s_SampledSitePtr>::iterator site_iter = m_sites.begin();
    while (site_iter != m_sites.end()) {
        if ((*site_iter)->name() == site_name) {
            (*site_iter)->normalize(name_setter);
            break;
        }
        site_iter++;
    }
}


void Integrator::normalize(string site_name) {
    vector<s_SampledSitePtr>::iterator iter;
```

```
    for (iter = m_sites.begin(); iter != m_sites.end(); iter++) {
        if ((*iter)->name() == site_name)
            (*iter)->normalize();
    }
}


void Integrator::add_site(string path) {
    s_SampledSitePtr new_sampled(new SampledSite(path));
    add_site(new_sampled);
}


void Integrator::add_site(s_SampledSitePtr site) {
    if (Integrator::find_site(site->name()) == -1) {
        m_sites.push_back(site);
        m_sitenames.push_back(site->name());
        m_master->add_data(site);
        // if sites are standardized, they aren't now !!
        // THATS A BIG PROBLEM!
        // TODO: keep a list of names and only restandardize those sites.

        // OR... you can handle this dynamically...
        m_sitecount++;
    }
    stale_conf = true;
}
```

```
void Integrator::remove_site(string name) {
     int index = find_site(name);

     if (index == -1)
          return;
     else {
          m_sites[index].reset();
          m_sitecount--;

     }
     stale_conf = true;
}


void Integrator::configure() {
     vector<s_SampledSitePtr>::iterator site_iter = m_sites.begin();
     vector<DatasetPtr>::iterator ref_iter;
     vector<vector<TestResults>>::iterator iter;

     int unique_count = m_master->test_count();

     int test_id;
     double r_val;
     double r_basis;

     if (!m_tesres_adj.empty())
          m_tesres_adj.clear();
```

```
    determine_coverage();

    while (site_iter != m_sites.end()) {
        for (ref_iter=m_master->m_tests.begin();ref_iter!=m_master->m_tests.end();ref_iter++) {
            test_id = (*ref_iter)->id();
            if (is_usable(test_id)) {
                m_master->std_refrc(test_id, r_val, r_basis);
                (*site_iter)->standardize(test_id, r_val, r_basis);
            }
        }
        if ((*site_iter)->standardize_cnt() > 0) {
            (*site_iter)->generate_payloads();
            m_tesres_adj.push_back(*(*site_iter)->get_payloads());
        }
        site_iter++;
    }

    for (iter = m_tesres_adj.begin(); iter != m_tesres_adj.end(); iter++)
        calculate_site(*iter);

    stale_conf = false;
}


void Integrator::show_results() {
    if (stale_conf)
        configure();
```

```
    m_plotter = new Plotter(&m_tesres_adj,
          &m_results,
          m_theta,
          m_usable_ids.size(),
          &(*m_master));

    m_plotter->plot_results();
    delete m_plotter;
    m_plotter = nullptr;
}

// Private
void Integrator::assign_ids() {
    vector<s_SampledSitePtr>::iterator sites;
    int test_count = m_master->test_count();
    int test_id = 0;
    string name;

    for (int i = 0; i < test_count; i++) {
          m_master->id_enumerator(i, name);           //get name based on id
          for (sites = m_sites.begin(); sites != m_sites.end(); sites++) {
                (*sites)->id_setter(test_id);
                (*sites)->id_enumerator(i, name);
          }
          test_id++;
    }
}
```

```
void Integrator::set_theta(int points) {
     m_theta = (2 * M_PI) / points;
     m_sine_theta = sin(m_theta);
}

void Integrator::calculate_site(vector<TestResults> &test_results) {
     double u_area = 0;
     double m_area = 0;
     double l_area = 0;
     double str_u, str_m, str_l;
     double lag_u, lag_m, lag_l;
     double lead_s;

     string site_name;
     int site_id;

     ResultsPtr site_result;
     vector<TestResults>::const_iterator iter = test_results.begin();
     site_name = iter->site_name();
     site_id = iter->site_id();
     lag_u = str_u = iter->upper_error();
     lag_m = str_m = iter->mean();
     lag_l = str_l = iter->lower_error();
     iter++;
```

```
        // Partial area calculation for the polygon
        while (iter != test_results.end()) {
            lead_s = iter->upper_error();
            u_area += .5 * lag_u * lead_s * m_sine_theta;
            lag_u = lead_s;

            lead_s = iter->mean();
            m_area += .5 * lag_m * lead_s * m_sine_theta;
            lag_m = lead_s;

            lead_s = iter->lower_error();
            l_area += .5 * lag_l * lead_s * m_sine_theta;
            lag_l = lead_s;
            iter++;
        }
        u_area += .5 * lag_u * str_u * m_sine_theta;
        m_area += .5 * lag_m * str_m * m_sine_theta;
        l_area += .5 * lag_l * str_l * m_sine_theta;

        // That 0 needs to be fixed
        site_result = new Results(site_name, site_id, u_area, m_area, l_area);

        m_results.push_back(*site_result);
        delete site_result;
        site_result = nullptr;
    }
```

```
void Integrator::determine_coverage() {
     vector<int>::iterator id_iter;   // an iterator to walk through the master site
     vector<DatasetPtr>::const_iterator master_iter = (m_master->getMasterSets()).begin();
     vector<s_SampledSitePtr>::const_iterator site_iter = m_sites.begin();

     while (master_iter != m_master->getMasterSets().end()) {
          m_usable_ids.push_back((*master_iter)->id());
          master_iter++;
     }

     while (site_iter != m_sites.end()) {
          id_iter = m_usable_ids.begin();
          while (id_iter != m_usable_ids.end()) {
               if (!(*site_iter)->test_present(*id_iter))
                    id_iter = m_usable_ids.erase(id_iter);
               else
                    id_iter++;
          }
          site_iter++;
     }
     set_theta(m_usable_ids.size());
}


bool Integrator::is_usable(int _id) const {
     vector<int>::const_iterator iter;
     for (iter = m_usable_ids.begin(); iter != m_usable_ids.end(); iter++)
          if (_id == *iter)
```

```
                return true;
        return false;
}

int Integrator::find_site(string name) const {
        list<string>::const_iterator iter = m_sitenames.begin();
        int i = 0;
        while (iter != m_sitenames.end()) {
                if (*iter == name)
                        return i;
                i++;
                iter++;
        }
        return -1;
}

/*
void Integrator::maximize_accuracy() {

} */
<<<<<< END

<<<<<< integrator.h
#ifndef INTEGRATOR_H
#define INTEGRATOR_H
```

```
#define _USE_MATH_DEFINES

// System Directives
#include <math.h>
#include <iostream>
#include <memory>
#include <algorithm>
#include <set>

// User Defined Directives
#include "site.h"
#include "sitemaster.h"
#include "sitesample.h"
#include "types.h"

// Using Statements
using std::set;
using std::string;
using std::shared_ptr;

class Integrator;
class Plotter;

typedef Integrator * IntegratorPtr;
typedef shared_ptr<Integrator> s_IntegratorPtr;
typedef Plotter * PlotterPtr;

class Integrator {
```

```
public:
    // Constructors & Destructor
    Integrator() {};
    Integrator(const vector<string> &paths);
    ~Integrator();

    // Observers
    void print_sites() const;
    void print_site(string site_name) const;
    void print_master() const { m_master->print_tests(); }
    void show_results();
    //int get_sitecount() const { return m_sitecount; }


    // Observer/Mutator
    int get_testcount();                        // will configure() on demand.

    // Mutators
    void standardize(string site_name);
    void standardize(string site_name, string name_setter);
    void normalize(string site_name, string name_setter);
    void normalize(string site_name);
    void add_site(string path);
    void add_site(s_SampledSitePtr site);
    void remove_site(string name);

    void configure();
private:
```

```
    bool stale_conf;

    int m_sitecount;
    double m_theta;
    double m_sine_theta;

    // Site Data
    s_MasterSitePtr m_master;
    vector<s_SampledSitePtr> m_sites;

    // Names & Ids
    list<string> m_sitenames;
    vector<int> m_usable_ids;

    // Result Data
    list<Results> m_results;
    vector<vector<TestResults>> m_tesres_adj;
    vector<vector<VerboseResultsPtr>> m_verres_adj;

    PlotterPtr m_plotter;

    // Helper Methods
    void assign_ids();                  // assign ids to sites and tests
    void set_theta(int points);         // calculate theta based on how many data points
    void calculate_site(vector<TestResults> & test_results);
    void determine_coverage();          // Use all (the most) sites

    bool is_usable(int ID) const;
```

```
    int find_site(string name) const;     // search the vector of site names, -1 if not found
};
#endif
<<<<<< END


<<<<<< plotter.cpp
#include <stdio.h>  /* defines FILENAME_MAX */
#include <string>
#include <cassert>
#include <fstream>
#include <algorithm>
//#include <windows.h>

#include "plotter.h"
#include "integrator.h"
#include "box-whisker.h"
#include "svg.h"
#include "types.h"

using std::cerr;
using std::endl;
using std::to_string;
using std::ifstream;
using std::ofstream;
using std::sort;
```

```
Plotter::Plotter(vector<vector<TestResults>> * tesres_adjlist,
     list<Results> * results,
     double theta,
     int endpoints_cnt,
     const MasterSitePtr master)
          : m_ORIGIN_X(300),
          m_ORIGIN_Y(300) {

     m_endpoints = endpoints_cnt;

     m_theta = theta;
     m_radius = 250;

     m_adj_list = tesres_adjlist;
     m_results = results;
     // verbose results here

     // sort results
     m_results->sort([](const Results & lhs, const Results & rhs) { return lhs.mean() >
rhs.mean(); });

     import_colors();
}

void Plotter::plot_results(string out_filename) {
     const char * out_filename_cstr = out_filename.c_str();

     vector<vector<TestResults>>::const_iterator sites = (*m_adj_list).begin();
```

```
vector<string>::const_iterator color = m_colors.begin();

string buffer = string();
ofstream out_file;

out_file.open(out_filename_cstr);
if (!out_file.good()) {
     cerr << "Could not create " << out_filename << "." << endl;
     return;
}

html_header(buffer);

buffer += "<div class='wrapper'>\n";
// START OF RESULTS TABLE
result_table_hdr(buffer);
result_table_rows(buffer);
result_table_ftr(buffer);
out_file << buffer;
buffer.clear();
// END OF RESULT TABLE

// START OF SVG
SVG::header(buffer, 600, 600);
for (const vector<TestResults> & site : *m_adj_list) {
     if (color == m_colors.end())
          color = m_colors.begin();
     draw_site(buffer, *color, site);
```

```
            draw_legend(buffer, *color, site[0].site_name());
            color++;
        }
        draw_axis(buffer);
        SVG::footer(buffer);

        buffer += "</div>\n";

        out_file << buffer;
        buffer.clear();
        // END OF SVG

        // END OF HTML
        out_file << buffer;
        buffer.clear();
        out_file.close();

        //ShellExecute(NULL, "open", out_filename_cstr,
        //    NULL, NULL, SW_SHOWNORMAL);
}


// Helpers: Output Formating
void Plotter::html_header(string & _html) {
        // Create necessary HTML specs
        _html += "<!DOCTYPE html>\n";
        _html += "<html>\n";
        _html += "<head>\n";
```

```
    // styles and fonts
    _html += "<link rel=\"stylesheet\" type=\"text/css\" href=\"styles.css\">\n";
    _html += "<link href='https://fonts.googleapis.com/css?family=Droid+Sans+Mono'
rel='stylesheet' type='text/css'>\n";
    _html += "<link href='https://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet'
type='text/css'>\n";

    _html += "</head>\n";
    _html += "<body>\n";
}


void Plotter::result_table_hdr(string & _html) {
    _html += "<div class='table-wrap'>";
    _html += "<table id='results-table'>\n";
    _html += "<colgroup class='site-col'></colgroup>\n";
    _html += "<colgroup class='lower-col'></colgroup>\n";
    _html += "<colgroup class='mean-col' id='important'></colgroup>\n";
    _html += "<colgroup class='upper-col'></colgroup>\n";
    _html += "<thead>\n";
    _html += "<tr class='header'>\n";
    _html += "<th class='header-label'><span>Region</span></th>\n";
    _html += "<th class='header-label'><span>+ Error</span></th>\n";
    _html += "<th class='header-label'><span>Mean</span></th>\n";
    _html += "<th class='header-label'><span>- Error</span></th>\n";
    _html += "</tr>\n";
    _html += "</thead>\n";
```

```
}


void Plotter::result_table_rows(string & _html) {
     list<Results>::const_iterator results = m_results->begin();

     string temp;
     int pos;

     _html += "<tbody>\n";
     while (results != m_results->end()) {
          _html += "<tr>\n";
          _html += "<td class='site-name'>";
          _html += results->site_name();
          _html += "</td>\n";

          _html += "<td class='num-result'>";
          temp = to_string(results->upper_error());
          pos = temp.find('.');
          pos += 3;
          temp = temp.substr(0, pos);
          _html += temp;
          temp.clear();
          _html += "</td>\n";

          _html += "<td class='num-result'>";
          temp = to_string(results->mean());
          pos = temp.find('.');
```

```
        pos += 3;
        temp = temp.substr(0, pos);
        _html += temp;
        temp.clear();
        _html += "</td>\n";

        _html += "<td class='num-result'>";
        temp = to_string(results->lower_error());
        pos = temp.find('.');
        pos += 3;
        temp = temp.substr(0, pos);
        _html += temp;
        temp.clear();
        _html += "</td>\n";

        _html += "</tr>\n";
        results++;
    }
    _html += "</tbody>\n";
}


void Plotter::result_table_ftr(string & _html) {
    _html += "</table>\n";
    _html += "</div>\n";
}
```

```
void Plotter::html_footer(string & _html) {
     _html += "</body>\n";
     _html += "</html>\n";
}


// Helpers: Macro SVG generator methods
void Plotter::draw_axis(string & buffer) {
     double end_x;
     double end_y;
     double theta = 0;

     vector<TestResults> & tests = m_adj_list[0][0];
     vector<TestResults>::const_iterator label_iter = tests.begin();

     for (int i = 0; i < m_endpoints; i++) {
          ptoc(m_ORIGIN_X, m_ORIGIN_Y, m_radius, theta, end_x, end_y);
          SVG::line(buffer, m_ORIGIN_X, m_ORIGIN_Y, end_x, end_y, "stroke='#000000'
stroke-width='1\'");
          theta += m_theta;
     }

     theta = 0;
     for (int i = 0; i < m_endpoints; i++) {
          ptoc(m_ORIGIN_X, m_ORIGIN_Y, m_radius + 15, theta, end_x, end_y);

          SVG::text(buffer, label_iter->test_name(), end_x, end_y, "fill='#000'");
```

```
            label_iter++;
            theta += m_theta;
        }
}

void Plotter::draw_section(string & buffer,
        const string & color,
        int offset,
        const TestResults & curr,
        const TestResults & next) {

        vector<double> xs;
        vector<double> ys;
        double theta = offset * m_theta;
        double end_x;
        double end_y;

        double mean_x1, mean_y1;
        double mean_x2, mean_y2;

        string extra;

        extra = "fill='";
        extra += color;
        extra += "' fill-opacity='0.45' stroke='";
        extra += color;
        extra += "' stroke-width='1'";
```

```
    ptoc(m_ORIGIN_X, m_ORIGIN_Y, curr.mean() * 2.5, theta, mean_x1, mean_y1);
    ptoc(m_ORIGIN_X, m_ORIGIN_Y, next.mean() * 2.5, theta + m_theta, mean_x2, mean_y2);

    // Get first corner
    ptoc(m_ORIGIN_X, m_ORIGIN_Y, curr.upper_error() * 2.5, theta, end_x, end_y);
    xs.push_back(end_x);
    ys.push_back(end_y);

    // Get second corner
    ptoc(m_ORIGIN_X, m_ORIGIN_Y, next.upper_error() * 2.5, theta + m_theta, end_x, end_y);
    xs.push_back(end_x);
    ys.push_back(end_y);

    // Get third corner
    ptoc(m_ORIGIN_X, m_ORIGIN_Y, next.lower_error() * 2.5, theta + m_theta, end_x, end_y);
    xs.push_back(end_x);
    ys.push_back(end_y);

    // Get fourth corner
    ptoc(m_ORIGIN_X, m_ORIGIN_Y, curr.lower_error() * 2.5, theta, end_x, end_y);
    xs.push_back(end_x);
    ys.push_back(end_y);

    SVG::path(buffer, xs, ys, extra);
    draw_mean(buffer, mean_x1, mean_y1, mean_x2, mean_y2);
}
```

```
// Really F'n optimized...
void Plotter::draw_site(string & buffer,
     const string & color,
     const vector<TestResults> & tests) {

     vector<TestResults>::const_iterator test_iter = tests.begin();
     const TestResults & start = *test_iter;
     const TestResults * lag = &start;
     const TestResults * curr = &tests[0];

     int offset;
     int test_cnt = tests.size();

     for (offset = 1; offset < test_cnt; offset++) {
          curr = &tests[offset];
          draw_section(buffer, color, offset - 1, *lag, *curr);
          lag = curr;
     }
     draw_section(buffer, color, offset - 1, *curr, start);
}


void Plotter::draw_legend(string & buffer,
     const string & color,
     string site_name) {

     static int placeHolder = m_results->size();
```

```
    int start = 600 - (placeHolder * 15);

    string extra = "stroke='" + color + "' ";
    extra += "fill='" + color + "' ";
    extra += "fill-opacity='0.45' ";

    SVG::rectangle(buffer, 10, start, 36, 12, extra);
    SVG::text(buffer, site_name, 55, start + 10);

    placeHolder -= 1;
}


void Plotter::draw_mean(string & buffer,
    double x1,
    double y1,
    double x2,
    double y2) {
    SVG::point(buffer, x1, y1, 2, "stroke='#000000'");
    SVG::line(buffer, x1, y1, x2, y2, "stroke='#000000' stroke-width='1'");
}


// Helpers: General plotter methods
void Plotter::import_colors() {
    ifstream colors_file;
    string color = "colors.dat";
```

```
    colors_file.open(color.c_str());
    colors_file >> color;

    while (colors_file) {
        m_colors.push_back(color);
        colors_file >> color;
    }
}


void Plotter::ptoc(double start_x, double start_y, double radius, double angle, double & end_x,
double & end_y) {
    end_x = (start_x + (radius * cos(angle)));
    end_y = (start_y + (radius * sin(angle)));
}
<<<<<< END


<<<<<< plotter.h
#ifndef PLOTTER_H
#define PLOTTER_H

#ifndef _USE_MATH_DEFINES
#define _USE_MATH_DEFINES
#endif // !_USE_MATH_DEFINES

// System Directions
```

```
#include <string>
#include <vector>
#include "math.h"

// Other Directives
#include "types.h"
#include "sitemaster.h"
#include "integrator.h"
#include "svg.h"

// Typedefs
class Plotter;
typedef Plotter* PlotterPtr;

// Using Statements
using std::vector;
using std::cos;
using std::sin;


class Plotter {
public:
     Plotter() : m_ORIGIN_X(0), m_ORIGIN_Y(0) {};
     Plotter(vector<vector<TestResults>> * tesres_adjlist,
          list<Results> * show_results,
          double theta,
          int test_cnt,
          const MasterSitePtr master);
```

```
    ~Plotter() {};

    void plot_results(string out_filename="output.html");

private:
    int m_endpoints;

    const double m_ORIGIN_X;
    const double m_ORIGIN_Y;

    double m_theta;
    double m_radius;

    vector<vector<TestResults>> * m_adj_list;
    list<Results> * m_results;
    vector<string> m_colors;

    // Helpers: Output Formating
    void html_header(string & _html);
    void result_table_hdr(string & _html);
    void result_table_rows(string & _html);
    void result_table_ftr(string & _html);
    void html_footer(string & _html);

    // Helpers: SVG 'plotter' methods
    void draw_axis(string & output);
    void draw_section(string & output,
```

```
            const string & color,
            int offset,
            const TestResults & curr,
            const TestResults & next);
        void draw_site(string & out,
            const string & color,
            const vector<TestResults> & tests);
        void draw_legend(string & buffer,
            const string & color,
            string site_name);
        void draw_mean(string & output,
            double x1,
            double y1,
            double x2,
            double y2);

        // Helpers: General plotter methods
        void import_colors();
        void ptoc(double centerX,
            double centerY,
            double radius,
            double angle,
            double & end_x,
            double & end_y);
};
#endif
<<<<<< END
```

```
<<<<<< site.cpp
// System Directives
#include <iostream>
#include <fstream>
#include <cassert>
#include <string>
#include <memory>
#include <algorithm>

// User Defined Directives
#include "types.h"
#include "site.h"
using std::cout;
using std::endl;
using std::swap;
using std::string;
using std::vector;
using std::ifstream;
using std::shared_ptr;
using std::sort;

int Site::get_n(string test_name) {
     DatasetPtr temp = get_test(test_name);
     return temp ? temp->size() : -1;
}

// Inherited Methods
```

```
void Site::add_test(const Dataset& add_data) {
     DatasetPtr temp = new Dataset(add_data);
     m_tests.push_back(temp);
     sort_tests();

     delete temp;
     temp = nullptr;
}


void Site::add_test(DatasetPtr add_data) {
     DatasetPtr new_set = new Dataset(add_data);
     m_tests.push_back(new_set);
     sort_tests();
}


void Site::merge_test(const Dataset &add_data) {
     shared_ptr<Dataset> temp(new Dataset(add_data));
     vector<DatasetPtr>::iterator iter;
     for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
          if ((*iter)->name() == temp->name())
               (*iter)->merge(temp);
}


void Site::merge_test(DatasetPtr add_data) {
     vector<DatasetPtr>::iterator iter;
```

```
    for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
        if ((*iter)->name() == add_data->name())
            (*iter)->merge(add_data);
}

DatasetPtr Site::get_test(string test_name) const {
    for (DatasetPtr dataset : m_tests)
        if (dataset->name() == test_name)
            return dataset;
    return nullptr;
}

bool Site::collision(DatasetPtr dataset) const {
    vector<DatasetPtr>::const_iterator iter;
    for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
        if ((*iter)->name() == dataset->name())
            return true;
    return false;
}

void Site::sort_tests() {
    bool swapped = true;                                    // uses bubble sort
    int start = 0;
    while (swapped) {
        swapped = false;
```

```
            start++;

            for (size_t i = 0; i < m_tests.size() - start; i++) {
                if (m_tests[i]->name() > m_tests[i + 1]->name()) {
                    swap(m_tests[i], m_tests[i + 1]);
                    swapped = true;
                }
            }
        }
}
<<<<<< END


<<<<<< site.h
#ifndef SITECORE_H
#define SITECORE_H

// System Directives
#include <iostream>
#include <vector>
#include <list>
#include <memory>

// User Defined Directives
#include "dataset.h"
#include "types.h"

// Using Statements
```

```cpp
using std::vector;
using std::list;
using std::shared_ptr;

// Forward Declaritions
class Site;

// Typedef and enumations
typedef shared_ptr<Dataset> sDatasetPtr;
typedef shared_ptr<Site> s_SiteCorePtr;

class Site {
public:
     Site() {};                                       // Constructors & Destructor
     ~Site() {};

     virtual void print_tests() = 0;                  // Observers
     virtual void describe() = 0;
     int get_n(string test_name);
protected:
     vector<DatasetPtr> m_tests;                      // Datasets

     // To remove
     list<VerboseResultsPtr> m_descriptions;          // Descriptive stats of those sets

     // Operations
     void add_test(const Dataset &add_data);     // Single, unique DataSet object by reference
     void add_test(DatasetPtr add_data);         // Single, unique DataSet object from shared ptr
```

```
    void merge_test(const Dataset &add_data);   // Merge to an existing DataSet, by reference
    void merge_test(DatasetPtr add_data);       // Merge to an existing DataSet, by shared ptr
    virtual void remove_test(string testname) = 0;

    // Search
    DatasetPtr get_test(string test_name) const;
    bool collision(DatasetPtr test) const;

    void sort_tests();
};
#endif
<<<<<< END


<<<<<< sitemaster.cpp
// System Directives
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cassert>
#include <string>
#include <memory>

// User Defined Directives
#include "types.h"
#include "sitemaster.h"

// using statements
```

```
using std::cout;
using std::endl;
using std::setw;
using std::ifstream;
using std::string;
using std::shared_ptr;
using std::vector;


// Constructors & Destructor
MasterSite::MasterSite(const MasterSite &copy_in) : stale_descript(true) {
     vector<DatasetPtr>::const_iterator cpy_iter = copy_in.m_tests.begin();
     vector<DatasetPtr>::iterator msite_it;

     while (cpy_iter != copy_in.m_tests.end()) {
          (collision((*cpy_iter))) ? merge_test(*cpy_iter) : add_test(*cpy_iter);
          cpy_iter++;
     }
     sort_tests();
     id_tests();
}


MasterSite::MasterSite(const SampledSite& copy_in) : stale_descript(true) {
     vector<DatasetPtr>::const_iterator cpy_iter = copy_in.m_tests.begin();

     while (cpy_iter != copy_in.m_tests.end()) {
          (collision((*cpy_iter))) ? merge_test(*cpy_iter) : add_test(*cpy_iter);
```

```
            cpy_iter++;
      }
      sort_tests();
      id_tests();
}


MasterSite::MasterSite(string path) : stale_descript(true) {
      bool desc;
      char order;

      string name_setter;
      string test_unit;
      string s_reader;

      ifstream InputFile;
      vector<double> data;

      // Procedure
      InputFile.open(path.c_str());
      getline(InputFile, s_reader);                      // Clear site name
      assert(InputFile && "Could not be read.");
      while (getline(InputFile, name_setter)) {          // set name
            InputFile >> order;                           // ordering
            toupper(order);
            InputFile.get();                               // clear \n

            desc = (order == 'A') ? false : true;
```

```
            getline(InputFile, test_unit);                // unit
            getline(InputFile, s_reader);                 // priming data read

            while (s_reader.at(0) < 58) {
                  data.push_back(stod(s_reader));
                  getline(InputFile, s_reader);
            }                                             // Got all data

            getline(InputFile, s_reader);                  // Clear "END"

            DatasetPtr temp = new Dataset(data, name_setter, test_unit, desc);

            if (!collision(temp)) {                        // don't add datasets with same name
                  m_tests.push_back(temp);
            }
            data.clear();
      }

      InputFile.close();
      sort_tests();                                       // Sort all the Datasets names for this
site
      id_tests();
}


MasterSite::MasterSite(vector<string> paths) : stale_descript(true) {
      for (string path : paths) {
```

```
            shared_ptr<SampledSite> temp(new SampledSite(path));
            this->MasterSite::add_data(temp);
            temp.reset();
      }

      sort_tests();
      id_tests();
}


MasterSite::~MasterSite() {
      for (DatasetPtr test : m_tests) {
            delete test;
            test = nullptr;
      }

      for (VerboseResultsPtr result : m_descriptions)
            delete result;

      m_tests.clear();
      m_descriptions.clear();
}


// Observers
void MasterSite::print_tests() {
      vector<DatasetPtr>::iterator iter;
```

```
    cout << "~~~~~~~~~~~   MASTER  ~~~~~~~~~~~" << endl;
    for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
        (*iter)->print();

    cout << "=== END ===   " << "  === END ===" << endl;
    cout << endl << endl << endl;
}


void MasterSite::describe() {
    VerboseResultsPtr _push;
    for (DatasetPtr test : m_tests) {
        _push = new TestDescriptor("MASTER",
            -1,
            test->name(),
            test->unit(),
            test->id(),
            test->size(),
            test->up_mean_err(),
            test->mean(),
            test->lw_mean_err(),
            test->minimum(),
            test->q1(),
            test->median(),
            test->q3(),
            test->maximum());
        m_descriptions.push_back(_push);
    }
```

```
}

void MasterSite::describe(int test_id, VerboseResultsPtr & _result) {
     for (VerboseResultsPtr result : m_descriptions)
          if (result->test_id() == test_id) {
               _result = result;
               return;
          }
     _result = nullptr;
}

int MasterSite::test_size(int test_id) const {
     for (DatasetPtr test : m_tests)
          if (test->id() == test_id)
               return test->size();
     return -1;
}

void MasterSite::std_refrc(int test_id, double & val, double & basis) {
     vector<DatasetPtr>::iterator iter;
     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
          if ((*iter)->id() == test_id) {
               val = (*iter)->minimum();
               basis = (*iter)->basis();
               break;
```

```
            }
        }
}


void MasterSite::std_refrc(string ds_name, double & val, double & basis) {
        vector<DatasetPtr>::iterator iter;
        for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
                if ((*iter)->name() == ds_name) {
                        val = (*iter)->minimum();
                        basis = (*iter)->basis();
                }
        }
}


DatasetPtr MasterSite::get_test(string name) const {
        vector<DatasetPtr>::const_iterator iter;
        for (iter = m_tests.begin(); iter != m_tests.end(); iter++)
                if ((*iter)->name() == name)
                        return *iter;
        return nullptr;
}

// Mutators
void MasterSite::remove_test(string name) {
```

```
        sort_tests();
        id_tests();
        return;
        // Not Implement, take the set, remove the values from the Linked List
}

void MasterSite::add_data(string path) {
        SampledSite temp(path);
        vector<DatasetPtr>::iterator iter = temp.m_tests.begin();
        // check to see if master site has no data, create the first set
        if (this->m_tests.size() == 0) {
                this->m_tests.push_back(*iter);
                iter++;
        }
        // add (or merge) all (or the rest) of the remaining sets.
        while (iter != temp.m_tests.end()) {
                (collision((*iter))) ? merge_test(*iter) : add_test(*iter);
                iter++;
        }
        sort_tests();
        id_tests();
}

void MasterSite::add_data(SampledSite &merge) {
        vector<DatasetPtr>::iterator iter = merge.m_tests.begin();
        // check to see if master site has no data, create the first set
```

```
	if (this->m_tests.size() == 0) {
		this->m_tests.push_back(*iter);
		iter++;
	}
	// add (or merge) all (or the rest) of the remaining sets.
	while (iter != merge.m_tests.end()) {
		(collision(*iter) ? merge_test(*iter) : add_test(*iter));
		iter++;
	}
	sort_tests();
	id_tests();
}


void MasterSite::add_data(s_SampledSitePtr merge) {
	vector<DatasetPtr>::iterator iter = merge->m_tests.begin();
	// check to see if master site has no data, create the first set
	if (this->m_tests.size() == 0) {
		this->m_tests.push_back(*iter);
		iter++;
	}
	// add (or merge) all (or the rest) of the remaining sets.
	while (iter != merge->m_tests.end()) {
		(collision((*iter))) ? merge_test(*iter) : add_test(*iter);
		iter++;
	}
	sort_tests();
	id_tests();
```

```
}


void MasterSite::id_enumerator(int test_id, string &name_setter) {
     vector<DatasetPtr>::const_iterator iter;
     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
          if ((*iter)->id() == test_id) {
               name_setter = (*iter)->name();
               break;
          }
     }
}

// Private
void MasterSite::id_tests() {
     int test_id = 0;
     vector<DatasetPtr>::iterator iter;
     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
          (*iter)->id_setter(test_id);
          test_id++;
     }
}
<<<<<< END


<<<<<< sitemaster.h
#ifndef MASTERSITE_H
#define MASTERSITE_H
```

```cpp
// System Directives
#include <iostream>
#include <vector>
#include <memory>

// User Defined Directives
#include "dataset.h"
#include "site.h"
#include "sitesample.h"
#include "types.h"

// using statements
using std::vector;
using std::shared_ptr;

// forward declarations
class SampledSite;
class MasterSite;

// typedefs
typedef shared_ptr<Dataset> sDatasetPtr;
typedef shared_ptr<SampledSite> s_SampledSitePtr;
typedef MasterSite* MasterSitePtr;
typedef shared_ptr<MasterSite> s_MasterSitePtr;

class MasterSite : public Site {
public:
```

```
    friend class Integrator;
    MasterSite() {};
    MasterSite(const MasterSite &copy_in);          // Copy a Master Site
    MasterSite(const SampledSite &copy_in);         // Copy in a Sampled Site
    MasterSite(string path);                        // Construct from a single site data file
    MasterSite(vector<string> paths);               // Construct from a vector of paths!
    ~MasterSite();

    // Observers
    void print_tests();
    void describe();
    void describe(int test_id, VerboseResultsPtr & result);
    int test_count() const { return m_tests.size(); }
    int test_size(int test_id) const;

    void std_refrc(int test_id, double & val, double & basis);
    void std_refrc(string name_setter, double & val, double & basis);

    // These really need to be reworked
    const vector<DatasetPtr>& getMasterSets() const { return m_tests; }
    DatasetPtr get_test(string name_setter) const;

    // Mutators
    void remove_test(string setname);
    void add_data(SampledSite &merge);          // Add or merge into this site
    void add_data(s_SampledSitePtr merge);      // Add or merge s_SiteCorePtr into this site
    void add_data(string path);                 // Add or merge file path.
    void id_enumerator(int test_id, string &name_setter);
```

```
private:
     bool stale_descript;
     void id_tests();
};
#endif
<<<<<< END


<<<<<< sitesample.cpp
// System Directives
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cassert>
#include <string>
#include <memory>

// User Defined Directives
#include "types.h"
//#include "site.h"
#include "sitesample.h"
//#include "sitemaster.h"

// using statements
using std::cout;
using std::endl;
using std::setw;
```

```
using std::ifstream;
using std::string;
using std::shared_ptr;
using std::vector;

// Constructors & Destructor
SampledSite::SampledSite(string path) : stale_descript(true) {
     bool desc;
     char order;

     string test_name;
     string test_unit;
     string s_reader;

     ifstream InputFile;
     vector<double> data;

     // Procedure
     m_stdcount = 0;

     InputFile.open(path.c_str());
     getline(InputFile, this->site_name);
     assert(InputFile && "could not be read.");

     while (getline(InputFile, test_name)) {                    // Name of test
          InputFile >> order;                                   // Ordering (A/D)
          toupper(order);
          InputFile.get();                                      // Clear '\n'
```

```
        desc = (order == 'A') ? false : true;

        getline(InputFile, test_unit);                          // Unit of measure
        getline(InputFile, s_reader);                           // Prime the stream

        while (s_reader.at(0) < 58) {
            data.push_back(stod(s_reader));
            getline(InputFile, s_reader);
        }                                                       // Data acquired
        getline(InputFile, s_reader);                           // Clear "END"

        // Create a dataset
        DatasetPtr dataset =  new Dataset(data,
            test_name,
            test_unit,
            desc);

        if (!collision(dataset))                                // Add uniques tests only
            m_tests.push_back(dataset);

        // Prepare for next iteration
        data.clear();
    }

    // Clear stream
    InputFile.close();
    sort_tests();
```

```
}


SampledSite::~SampledSite() {
     for (DatasetPtr d_ptr : m_tests) {
          delete d_ptr;
          d_ptr = nullptr;
     }

     for (VerboseResultsPtr result : m_descriptions)
          delete result;

     m_tests.clear();
     m_descriptions.clear();
}


// Observers
void SampledSite::print_tests() {
     cout << "===========   " << site_name << "  ===========" << endl;

     for (DatasetPtr test : m_tests)
          test->print();

     cout << "=== END ===   " << site_name << "  === END ===" << endl;
     cout << endl << endl << endl;
}
```

```
void SampledSite::describe() {
     VerboseResultsPtr _push;

     if (!stale_descript)
          return;

     for (DatasetPtr test : m_tests) {
          _push = new TestDescriptor(name(),
               site_id(),
               test->name(),
               test->unit(),
               test->id(),
               test->size(),
               test->up_mean_err(),
               test->mean(),
               test->lw_mean_err(),
               test->minimum(),
               test->q1(),
               test->median(),
               test->q3(),
               test->maximum());
          m_descriptions.push_back(_push);
     }
}


bool SampledSite::test_present(int test_id) const
```

```
{
	vector<DatasetPtr>::const_iterator iter;
	for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
		if ((*iter)->id() == test_id)
			return true;
	}
	return false;
}

vector<TestResults> const * SampledSite::get_payloads() {
	return (m_payloads.size() == 0) ? nullptr : &m_payloads;
}

// Mutators
void SampledSite::id_enumerator(int test_id, string name_setter) {
	vector<DatasetPtr>::iterator iter;

	for(iter = m_tests.begin(); iter != m_tests.end(); iter++)
		if ((*iter)->name() == name_setter) {
			(*iter)->id_setter(test_id);
			break;
		}
}

void SampledSite::remove_test(string name) {
```

```
    vector<DatasetPtr>::iterator iter;
    for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
        if ((*iter)->name() == name) {
            if ((*iter)->standardized())
                m_stdcount--;
            //delete iter;
            //iter = nullptr;
            m_tests.erase(iter); // does this free memory??!!??!!
            break;
        }
    }

    // you've remove data, you need update master site's set too...
    // this is really bad.
}


void SampledSite::normalize() {
    vector<DatasetPtr>::iterator iter;
    for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
        if ((*iter)->standardized()) {
            (*iter)->normalize();
            m_stdcount--;
        }
    }
}
```

```
void SampledSite::normalize(string name_setter) {
     vector<DatasetPtr>::iterator iter;
     for (iter = m_tests.begin(); iter != m_tests.end(); iter++)
          if ((*iter)->name() == name_setter) {
               (*iter)->normalize();
               m_stdcount--;
          }
}


bool SampledSite::standardize(string dataset_name, double ref_min, double ref_basis) {
     vector<DatasetPtr>::iterator iter = m_tests.begin();

     while (iter != m_tests.end()) {
          if ((*iter)->name() == dataset_name && !(*iter)->standardized()) {
               (*iter)->standardize(ref_min, ref_basis);
               m_stdcount++;
               return true;
          }
          iter++;
     }
     return false;
}


bool SampledSite::standardize(int test_id, double r_val, double r_basis) {
     vector<DatasetPtr>::iterator iter = m_tests.begin();
```

```
    while (iter != m_tests.end()) {
        if ((*iter)->id() == test_id && !(*iter)->standardized()) {
            (*iter)->standardize(r_val, r_basis);
            m_stdcount++;
            return true;
        }
        iter++;
    }
    return false;
}

void SampledSite::generate_payloads() {

    vector<DatasetPtr>::iterator dataset_iter = m_tests.begin();
    m_summaries.reserve(m_tests.size());

    for (DatasetPtr dataset : m_tests) {
        //m_summaries.push_back(dataset.get());
    }



    string site_name = this->name();
    int site_id = this->site_id();
    string name_setter;
    string test_unit;
    int test_id;
```

```
    double ue, m, le;

    PayloadPtr payload;
    vector<DatasetPtr>::const_iterator iter;

    for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
        if ((*iter)->standardized()) {
            // create the payload data
            name_setter = (*iter)->name();
            test_unit = (*iter)->unit();
            test_id = (*iter)->id();
            ue = (*iter)->n_upper_bound();
            m = (*iter)->norm_mean();
            le = (*iter)->n_lower_bound();
            // store the payload
            payload = new TestResults(site_name,
                test_id,
                name_setter,
                test_unit,
                test_id,
                ue,
                m,
                le);

            m_payloads.push_back(*payload);

            delete payload;
```

```
                payload = nullptr;
            }
        }
}


// Private
DatasetPtr SampledSite::get_test(string name) const {
    vector<DatasetPtr>::const_iterator iter;
    for (iter = m_tests.begin(); iter != m_tests.end(); iter++)
        if ((*iter)->name() == name)
            return *iter;
    return nullptr;
}
<<<<<< END


<<<<<< samplesite.h
#ifndef SITESMPL_H
#define SITESMPL_H

// System Directives
#include <iostream>
#include <memory>
#include <vector>

// User Defined Directives
#include "dataset.h"
```

```
#include "site.h"
#include "types.h"
//#include "sitemaster.h"

// using statements
using std::string;
using std::vector;
using std::shared_ptr;

// Forward Declaration
class MasterSite;
class SampledSite;

// Typedefs
typedef shared_ptr<Dataset> sDatasetPtr;
typedef shared_ptr<MasterSite> s_MasterSitePtr;
typedef SampledSite* SampledSitePtr;             // Redefine types for raw pointers
typedef shared_ptr<SampledSite> s_SampledSitePtr;

class SampledSite : public Site {
public:
     friend class Integrator;                    // Integratior needs to use id_sets();
     friend class MasterSite;                    // Master needs access to m_sets for merge

     SampledSite() : m_stdcount(0) {};           // Constructors & Destructor
     SampledSite(string path);
     ~SampledSite();
```

```
    // Observers
    void print_tests();
    void describe();

    // to remove
    int site_id() const { return m_id; }
    string name() const { return site_name; }

    int test_count() const { return m_tests.size(); }
    int standardize_cnt() const { return m_stdcount; }
    bool test_present(int test_id) const;

    vector<TestResults> const * get_payloads();

    // Mutators
    void set_name(string site_name) { site_name = site_name; }
    void id_setter(int test_id) { m_id = test_id; }
    void id_enumerator(int test_id, string name_setter);

    void remove_test(string name_setter);

    void normalize();                           // Normalize All Datasets
    void normalize(string name_setter);         // Normalize a Dataset
    bool standardize(string name_setter, double ref_min, double ref_basis);
// Returns true is dataset was Normalized
    bool standardize(int test_id, double r_min, double r_basis);
    void generate_payloads();
```

```
private:
     bool stale_descript;
     int m_id;                                   // This site's ID
     int m_stdcount;

     string site_name;                           // This site's name
     vector<TestResults> m_payloads;

     vector<DatasetSummary *> m_summaries;

     DatasetPtr get_test(string name_setter) const;
     // Returns a shared pointer to a dataset, nullptr if not found
};
#endif
<<<<<< END


<<<<<< styles.css
body {
  font-family: 'Open Sans', sans-serif;
}

svg text{
  font-family: 'Open Sans', sans-serif;
  font-size: 10px;
}
/* Wrappers */
```

```css
.wrapper {
  width: 600px;
  border: 1px solid #000;
}

.table-wrap {
  width: 600px;
  border: 1px solid #000;
}

svg {
  font-family: labels;
}

/* Results Table */
#results-table {
  width: 100%;
  margin:0 auto;
  border-collapse: collapse;
}

.header {
  height: 12px;
}

.header-label {
  background-color: #4C9ED9;
  font-size: 12px;
```

```
  color: #FFF;
}

.site-col {
  width: 300px;
}

.site-name {
  font-size: 12px;
  font-weight: 100;
  padding: 2px;
  padding-left: 5px;
}

.num-result {
  border-left: 1px solid #555;
  font-family: 'Droid Sans Mono';
  font-size: 12px;
  text-align: right;
  padding: 2px;
  padding-right: 5px;
}

tr:nth-child(even) {
  background-color: #d9ebf7;
}
<<<<<< END
```

```
<<<<<< svg.cpp
// System Directives
#include <string>
#include <vector>

// Other Directives
#include "svg.h"

// Using Statements
using std::vector;
using std::to_string;


void SVG::footer(string & svg) {
     svg += "</svg>\n";
}


void SVG::header(string & svg,
     int width,
     int height) {
     svg += "<svg version='1.1' baseProfile='full' width='" + to_string(width) + "' ";
     svg += "height='" + to_string(height) + "' xmlns='http://www.w3.org/2000/svg'>\n";
}


void SVG::line(string & buffer,
```

```
    double str_x,
    double srt_y,
    double end_x,
    double end_y,
    string extra) {
    buffer += "<line ";
    buffer += "x1='" + to_string(str_x) + "' y1='" + to_string(srt_y) + "' ";
    buffer += "x2='" + to_string(end_x) + "' y2='" + to_string(end_y) + "' ";
    buffer += extra;
    buffer += " />\n";
}

void SVG::path(string & buffer,
    const vector<double> & xs,
    const vector<double> & ys,
    string extra) {

    vector<double>::const_iterator x_it = xs.begin();
    vector<double>::const_iterator y_it = ys.begin();
    // Create path tag and attribute d...
    buffer += "<path d='";
    buffer += 'M' + to_string(*x_it) + ' ' + to_string(*y_it);

    while (x_it != xs.end() && y_it != ys.end()) {
        buffer += " L" + to_string(*x_it) + ' ' + to_string(*y_it);
        x_it++;
        y_it++;
```

```
    }
    buffer += " Z' ";
    buffer += extra;
    buffer += " />\n";
}


void SVG::point(string & buffer,
    double x,
    double y,
    double r,
    string extra) {
    buffer += "<circle cx='";
    buffer += to_string(x);
    buffer += "' cy='";
    buffer += to_string(y);
    buffer += "' ";
    buffer += "r='";
    buffer += to_string(r);
    buffer += "' ";
    buffer += extra;
    buffer +=" />\n";
}


void SVG::rectangle(string & buffer,
    double x,
    double y,
```

```
      double w,
      double h,
      string extra){

      buffer += "<rect x='" + to_string(x) + "' y='" + to_string(y) + "' ";
      buffer += "width='" + to_string(w) + "' height='" + to_string(h) + "' ";
      buffer += extra;
      buffer += "/>\n";
}


void SVG::text(string & buffer,
      string label,
      double x,
      double y,
      string extra) {

      buffer += "<text x='";
      buffer += to_string(x);
      buffer += "' y='";
      buffer += to_string(y);
      buffer += "' ";
      buffer += extra;
      buffer += '>';
      buffer += label;
      buffer += "</text>\n";
}
<<<<<< END
```

```
<<<<<< svg.h
#ifndef SVG_H
#define SVG_H

#include <string>
#include <vector>

using std::string;
using std::vector;

namespace SVG {
     void footer(string & svg);
     void header(string & svg,
          int width,
          int height);
     void line(string & ins_line,
          double str_x,
          double str_y,
          double end_x,
          double end_y,
          string extra = "");
     void path(string & ins_path,
          const vector<double> & x,
          const vector<double> & y,
          string extra = "");
     void point(string & ins_point,
```

```
            double pos_x,
            double pos_y,
            double r,
            string extra = "");
      void rectangle(string & buffer,
            double x,
            double y,
            double w,
            double h,
            string extra = "");
      void text(string & ins_text,
            string label,
            double x,
            double y,
            string extra = "");
};

#endif
<<<<<< END


<<<<<< types.cpp
// System directives

// User Defined Directives
#include "types.h"
```

```
Results::Results(string site_name,
      int site_id,
      double u,
      double m,
      double l) {

      m_site_name = site_name;
      m_site_id = site_id;

      m_up_err = u;
      m_mean = m;
      m_lw_err = l;
}


TestResults::TestResults(string site_name,
      int site_id,
      string name_setter, // test
      string test_unit,
      int test_id,
      double u,
      double m,
      double l) :
      Results(site_name, site_id, u, m, l) {

      m_test_name = name_setter;
      m_unit = test_unit;
```

```
    m_test_id = test_id;
}


TestDescriptor::TestDescriptor(const TestResults & t_results, double min, double q1, double med,
double q3, double max)
    : TestResults(t_results) {

    m_min = min;
    m_q1 = q1;
    m_median = med;
    m_q3 = q3;
    m_max = max;
}


TestDescriptor::TestDescriptor(string site_name,
    int site_id,
    string test_name,
    string test_unit,
    int test_id,
    int n,
    double ue,
    double mean,
    double le,
    double mini,
    double q1,
    double med,
```

```
      double q3,
      double maxi)
      : TestResults(site_name, test_id, test_name, test_unit, test_id, ue, mean, le) {

      m_n = n;
      m_min = mini;
      m_q1 = q1;
      m_median = med;
      m_q3 = q3;
      m_max = maxi;
}
<<<<<< END


<<<<<< types.h
#ifndef TYPES_H
#define TYPES_H

#include <iostream>
using std::string;


// Forward Declarations
struct DataNode;
class Results;
class TestResults;
class TestDescriptor;
class Dataset;
```

```
class DatasetSummary;

typedef DataNode* DataNodePtr;                    // Redefined type for Pointers.
typedef TestDescriptor* VerboseResultsPtr;       // Remove
typedef TestResults* PayloadPtr;                  // Remove
typedef Results* ResultsPtr;
typedef DatasetSummary* SummaryPtr;

enum NORM_T { SELF, EXTERNAL };

struct DataNode {
     int rank;

     double norm_value;
     double value;
};

class Results {
public:
     Results() {}
     Results(string site_name,
          int site_id,
          double u,
          double m,
          double l);
```

```
    string site_name() const { return m_site_name; }
    int site_id() const { return m_site_id; }

    double upper_error() const { return m_up_err; }
    double mean() const { return m_mean; }
    double lower_error() const { return m_lw_err; }

protected:
    string m_site_name;
    int m_site_id;

    double m_up_err;      // Final data (Areas)
    double m_mean;
    double m_lw_err;
};


class TestResults : public Results {
// Once constructed, unmodifiable.
public:
    TestResults() {}
    TestResults(string site_name,
        int site_id,
        string test_name,
        string test_unit,
        int test_id,
        double u,
```

```
            double m,
            double l);

      // Observers
      string test_name() const { return m_test_name; }
      string test_unit() const { return m_unit; }
      int test_id() const { return m_test_id; }

      // Inherited members serve as single data points (Distances).

protected:
      string m_test_name;        // Descriptor
      string m_unit;
      int m_test_id;
      int m_site_id;
};


// Depreciate this
class TestDescriptor : public TestResults {

public:
      TestDescriptor(const TestResults & t_results,
            double min,
            double q1,
            double med,
            double q3,
            double max);
```

```
    TestDescriptor(string site_name,
        int site_id,
        string test_name,
        string test_unit,
        int test_id,
        int n,
        double ue,
        double mean,
        double le,
        double mini,
        double q1,
        double med,
        double q3,
        double maxi
    );

    double mini() const { return m_min; }
    double q1() const { return m_q1; }
    double median() const { return m_median; }
    double q3() const { return m_q3; }
    double maxi() const { return m_max; }

private:
    int m_n;
    double m_min;
    double m_q1;
    double m_median;
    double m_q3;
```

```
    double m_max;
};

#endif
<<<<<< END


<<<<<< index.php
<?php
    error_reporting(E_ALL);
    ini_set('display_errors', 1);
    require 'include/header.php';

    $fileIter = new FilesystemIterator('uploads/', FilesystemIterator::SKIP_DOTS);
    $fileCount = iterator_count($fileIter);

    $target_dir = "uploads/";

    $target_file = $target_dir . $fileCount . '.dat';

    if (isset($_POST['upload']) && !file_exists($target_file)) {
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file))
            header('location: ' . $_SERVER['PHP_SELF']);
       else {
        ?>
            <div class="palette palette-pomegranate">
                    ERROR: NO FILE SELECTED
            </div>
```

```
            <?php
           }

       } else if (isset($_POST['compute'])) {
            exec('ETP/a.out');
       } else if (isset($_POST['reset'])) {
            exec('rm -rf uploads/*');
            header('location: ' . $_SERVER['PHP_SELF']);
       }
?>

<body>
       <div class="container">
            <div class="row">
                 <h1>Ecotox Project Demo</h1>
                 <small>Written By Michael Schmidt in C++, PHP</small>
            </div>
            <div class="row">
                 <h6>Input your data files</h6>
                 <small>
                      <?php
                           printf("%d File(s) loaded.", $fileCount);
                      ?>
                 </small>
            </div>
            <div class="row">
                 <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post"
enctype="multipart/form-data" >
```

```
                    <div class="input-group">
                         <span class="input-group-btn">
                              <input type="file" name="fileToUpload" id="fileToUpload"
class="btn btn-primary" />
                         </span>
                         <input type="submit" name="upload" value="upload" class="btn btn-info"
/>
                         <?php
                              if ($fileCount > 2)
                                   echo '<input type="submit" name="compute" value="compute"
class="btn btn-primary" />';

                              if ($fileCount > 0)
                                   echo '<input type="submit" name="reset" value="reset"
class="btn btn-warning" />';
                         ?>
                    </div>
               </form>

          </div>
     </div>
</body>

<?php
require 'include/footer.php';
?>
<<<<<< END
```

Bibliography & References

"8 Paths." *Paths – SVG 1.1 (Second Edition)*, www.w3.org/TR/SVG/paths.html.

"9 Basic Shapes." *Basic Shapes – SVG 1.1 (Second Edition)*, www.w3.org/TR/SVG/shapes.html.

"Area of a triangle - "side angle side" (SAS) method." *Area of a triangle (Side-Angle-Side method) - Math Open Reference*, www.mathopenref.com/triangleareasas.html.

"Best Practices for File Formats." *Stanford Libraries*, library.stanford.edu/research/data-management-services/data-best-practices/best-practices-file-formats.

Browse : Python Package Index, pypi.python.org/pypi?%3Aaction=browse. Retrived Mar. 2018

Choi, Jung Kyoon, and Sang Cheol Kim. *Genetics*, Copyright © 2007 by the Genetics Society of America, Apr. 2007, www.ncbi.nlm.nih.gov/pmc/articles/PMC1855137/.

Collins, A R. "The comet assay for DNA damage and repair: principles, applications, and limitations." *Molecular biotechnology.*, U.S. National Library of Medicine, Mar. 2004, www.ncbi.nlm.nih.gov/pubmed/15004294.

"Flask home." *Welcome | Flask (A Python Microframework)*, flask.pocoo.org/.

"How to Choose a Data Format - Silicon Valley Data Science." *Silicon Valley Data Science*, 21 Sept. 2017, svds.com/how-to-choose-a-data-format/.

"Is it a good practice to always scale/Normalize data for machine learning?" *Cross Validated*, stats.stackexchange.com/questions/189652/is-it-a-good-practice-to-always-scale-normalize-data-for-machine-learning.

MacKay, Jory, and Jory MacKayEditor at Crew. Writer. Reader. Coffee drinker. "Building Facebook today: Is PHP still relevant in 2017?" *Crew.co*, 29 Jan. 2017, crew.co/blog/is-php-still-relevant-in-2017/.

Mark Otto, Jacob Thornton, and Bootstrap contributors. "Bootstrap." · *The most popular HTML, CSS, and JS library in the world.*, getbootstrap.com/.

"NumPy." *NumPy - NumPy*, numpy.org/.

"SciPy.org." *SciPy.org*, Enthought, 2018, scipy.org/.

Smith, Keith. *Environmental hazards: assessing risk and reducing disaster*. Routledge, 2013.

"The Computer Language Benchmarks Game." *The Computer Language Benchmarks Game*, benchmarksgame.alioth.debian.org/.

Williams, Sarah. *Normalizing Census Data : Census : 11.520*, 1 Oct. 2002, web.mit.edu/11.520/www/labs/lab5/normalize.html.

Zamaratskaia, Galia, and Vladimir Zlabek. *Sensors (Basel, Switzerland)*, Molecular Diversity Preservation International (MDPI), 2009, www.ncbi.nlm.nih.gov/pmc/articles/PMC3345832/.