The Report Committee for Marilyn Aldover Gambone

Certifies that this is the approved version of the following report:

QuickLET: A Web-based Tool for Estimating Pollutant Loads in a Watershed

APPROVED BY

SUPERVISING COMMITTEE:

Adnan Aziz, Supervisor

Michael White

QuickLET: A Web-based Tool for Estimating Pollutant Loads in a Watershed

by

Marilyn Aldover Gambone, B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2015

Dedication

I dedicate this report to:

Larry, for his unwavering support throughout this journey,

Jasmine, for inspiring me to do better,

Ian and *Eve,* for always giving me joy and hope,

Dahlia, for helping me appreciate what the present holds, and

Chris, En and *Tyler,* for never letting me forget my roots.

Acknowledgments

I would like to acknowledge my supervisor, Dr. Adnan Aziz, for the guidance in making this project possible and my mentor and reader, Dr. Michael White, for the invaluable help he provided in making me understand the value of our nation's natural resources.

Abstract

QuickLET: A Web-based Tool for Estimating Pollutant Loads in a Watershed

Marilyn Aldover Gambone, M.S.E.

The University of Texas at Austin, 2015

Supervisor: Adnan Aziz

Web-based modeling tools for estimating pollutant loads in watersheds are few in literature. Those that are available for public access often require domain expertise, making them relevant mostly among environmental researchers. Aside from intensive efforts required to gather enormous amount of data, the complexity of the modeling tools themselves hinder their application among non-technical users. Consequently, environmental decision makers often rely on outside consultants to perform watershed assessments for them.

This report presents the Quick Load Estimating Tool (QuickLET), a Web-based tool for estimating pollutant loads in watersheds across the contiguous US. QuickLET empowers users to visualize the effects of land use patterns, cultivated crops, and conservation practices through graphical representation. QuickLET implements an export coefficient approach for predicting the pollutant loads resulting in significant simplification of the estimating process.

Table of Contents

Acknowledgments v

Abstract vi

List of Tables ix

List of Figures x

Chapter 1. Introduction 1

1.1 Problem Definition 1
1.2 Objectives 2
1.3 Organization of this Report 3

Chapter 2. Watershed Modeling 4

2.1 Quantifying a problem using "scoping" estimates 4
2.2 QuickLET's watershed modeling strategy 5
2.2.1 The Export Coefficient approach 5
2.2.2 Empirical modeling 6
2.2.3 National datasets used in QuickLET 6
2.2.4 Simulated data 8
2.2.5 "Distillation" of empirical data for QuickLET 9

Chapter 3. QuickLET's Design and Implementation 10

3.1 Architectural style 10
3.2 Software design pattern 11
3.2.1 Models 12
3.2.2 View 13
3.2.3 Controllers 15
3.3 Design rationale 16
3.3.1 Integrated Development Environment (IDE) 16
3.3.2 Charts and sliders 17
3.3.3 Accordion panel 18
3.3.4 Cascading Stylesheet (CSS) and HTML 19
3.3.5 Web browsers 19
3.3.6 Screen resolution 20
3.3.7 GIS mapping 20

3.4 Database design 21
3.5 Sample QuickLET prediction process 25
3.5.1 Baseline calculations 25
3.5.2 Calculating changes to land use, crops and conservation practices 28
3.5.3 Calculating scenario nutrient loads 32
3.5.4 Load differences between baseline and scenario estimates 34

Chapter 4. Code and Performance Metrics **36**

4.1 Code metrics 37
4.2 Microsoft Applications Insights (AI) 39
4.2.1 Client-side metrics 39
4.2.2 Server-side metrics 40
4.3 Deployment to production server 40
4.3.1 Internet Information Services (IIS) 7.0 40
4.3.2 Deploying QuickLET to IIS 7.0 41
4.3.3 Deploying the database to SQL Server 2012 41

Chapter 5. Conclusion **43**
5.1 Related work 43
5.2 Future work 44
5.3 Lessons learned 45
5.3.1 What went wrong 46
5.3.2 What went right 46

Bibliography **48**

Vita **51**

List of Tables

Table 3.1 Comparison of Web browsers' performance 20

Table 3.2 Fractions table structure 24

Table 4.1 Maintainability Index score 37

Table 4.2 Code metrics results for QuickLET 38

List of Figures

Figure 2.1 HUC8s at 1:25000 resolution in polygon shapefile format 7

Figure 2.2 2009 Cropland Data Layer 7

Figure 3.1 QuickLET's three-layered architectural style 11

Figure 3.2 QuickLET's Model-View-Controller software design pattern 12

Figure 3.4 Screenshot of QuickLET's Home page upon entry to the Web site 13

Figure 3.5 Baseline charts consisting of land use, crop, conservation practices, and nutrient loads 14

Figure 3.6 Scenario charts reflecting changes as users moved the sliders 14

Figure 3.7 Master Web page header and footer common theme 15

Figure 3.8 Accordion panel 18

Figure 3.9 Conterminous US HUC8s converted from polygon to point shapefile format 21

Figure 3.10 QuickLET's Web mapping display 21

Figure 3.11 Entity relationship diagram before normalization 22

Figure 3.12 Entity relationship diagram after normalization 23

Figure 3.17 HUC 12010001 located in Upper Sabine of Region 12 in Texas 25

Figure 3.18 Calculating baseline land use with export coefficients 26

Figure 3.19 Calculating baseline crops with export coefficients 26
Figure 3.20 Calculating baseline conservation practices with export coefficients 26
Figure 3.21 Algorithm for calculating baseline edge-of-field nutrient loads 27
Figure 3.22 Baseline nutrient loads 28
Figure 3.23 UML Activity Diagram for calculating and rendering scenario charts 28
Figure 3.24 Algorithm for calculating scenario land use 29
Figure 3.25 Calculating changes to scenario land use chart 30
Figure 3.26 Algorithm for calculating scenario cultivated crops 31
Figure 3.27 Algorithm for calculating scenario conservation practices distribution 32
Figure 3.28 Scenario box plot charts 33
Figure 3.29 UML Activity Diagram for calculating scenario nutrient loads 33
Figure 3.30 Algorithm for calculating scenario nutrient load values 34
Figure 3.31 Algorithm for calculating differences between baseline and scenario nutrient loads 35
Figure 4.1 Client-side metrics 39
Figure 4.2 Server-side metrics 40

Chapter 1

Introduction

Watershed modeling tools are essential in assessing the impact of changes in land use patterns, crop cultivation and conservation practices on the environment. However, the complexity of existing modeling tools hinder their widespread application among non-expert users. Consequently, decision makers oftentimes require outside consultants to perform watershed analysis for them. Quick Load Estimating Tool (QuickLET) is a Web-based tool developed to address the need for a user-friendly, reliable and cost-effective watershed modeling tool. The ease of reaching a wide range and diverse audience makes the Web an ideal environment for application tools like QuickLET. Web-based application development, however, is fraught with challenges, from rapidly evolving technologies and infrastructure, to lack of standards among different browsers. A particular challenge for building QuickLET was reconciling the demands for faster, platform-agnostic application with demands for richer, process-intensive, and resource-hungry features. Resolving these competing demands provided the underlying motivation for the export coefficient approach that QuickLET employed in its design.

1.1 Problem Definition

Assessing the effects of land use, crop cultivations and conservation practices on the environment has been the focus of sustainable resource management. To fulfill their responsibilities, environmental practitioners and decision makers required

tools that facilitated the monitoring and assessment of the nation's natural resources. The research community pioneered the building of modeling tools, which provided an understanding of the environment and its response to change that would not be possible by any other means. For example, gathering monitoring data on nonpoint pollutant loads is expensive at the watershed level because monitoring is a resource-intensive activity. Monitoring all impaired watersheds would be cost-prohibitive at this national scale. Modeling tools, on the other hand, allow the approximation of natural processes on large spatial scales as well as the prediction of future environmental phenomena [1]. Despite these strengths, the success of modeling tools were confined essentially among the research community. The limited adoption of environmental modeling tools were due largely to the following weaknesses:

- Extensive efforts were required to gather, update and maintain the input data for the models.
- Complexity of the modeling tools required an intermediate or advanced domain knowledge and technical expertise that most users do not have.
- Model construction, data input, calibration, and scenario development required a significant amount of time.

1.2 Objectives

The primary goal of this report is to develop a Web-based application for predicting pollutant loads at the national level using a combination of land uses, types of cultivated crops, and conservation practices. The specific goals of this report are:

1. Develop a dynamic Web-based application that is user-friendly, does not require technical expertise from users, and generates pollutant load estimates rapidly.
2. Develop a dynamic Web-based application using pre-processed and scientifically reliable data from all watersheds within the contiguous US.
3. Design a dynamic Web-based application primarily targeting the PC environment.

1.3 Organization of this Report

The remainder of this report are organized as follows:

- **Chapter 2 - Watershed Modeling.** This chapter explains the basics of watershed modeling and the export coefficient approach.
- **Chapter 3 - QuickLET's Design and Implementation.** This chapter describes QuickLET's architectural design and its implementation using a Model-View-Controller software design pattern. It also discusses the methods for estimating pollutant loads in a watershed.
- **Chapter 4 - Code and Performance Metrics.** This chapter discusses the results of code analysis and performance metrics employed in QuickLET.
- **Chapter 5 – Conclusion.** This chapter provides a conclusion and discusses related work, future work and lessons learned.

Chapter 2

Watershed Modeling

A watershed is an area of land where all of the water that drains off of it goes into the same place [2]. Watershed modeling describes the processes involved in delivering the amount of pollutants in a watershed and identifies their sources [3]. Most watershed modeling tools analyze only portions of the watershed processes due to the complexity of real watershed systems. Balancing the simplicity of model use and the complexity of model representation remains an ongoing challenge in watershed modeling, as Ma, Ahuja and Malone [4] observed:

> On one hand, it is necessary to represent the processes in as much detail as possible for the model to be applicable to a wide range of conditions. On the other hand, it is equally important for a model to be not too complicated so as to discourage its use.

Maintaining the ease of application usage as well as providing scientifically reliable estimates was the approach employed in the design of QuickLET. The following sections explain how QuickLET steered through these complex tasks.

2.1 Quantifying a problem using "scoping" estimates

When a better understanding of an environmental quality problem is desired, a simple model that quickly provides "scoping" or "screening" estimates of the extent and severity of a problem is often sufficient [5]. The ideal condition would be to collect monitoring data since they provide a more reliable information. But oftentimes, a lack of funding and insufficient time prevented the necessary collection of data. Thus, using a scoping modeling tool, like QuickLET, would be a more cost-

effective alternative since one could obtain reasonable estimates of contaminant loads for assessing relative magnitudes and areas of greatest risk with very little cost.

2.2 QuickLET's watershed modeling strategy

QuickLET's purpose for scoping estimates was to predict pollutant loads downstream of a watershed, where all streams and their tributaries converged. To do this, QuickLET extrapolated data from current conditions to potential future conditions. QuickLET relied on representing the general loading pattern within an ecoregion using an export coefficient (EC) method. The following sections explain the EC approach that QuickLET implemented.

2.2.1 The Export Coefficient approach

The EC approach has been successfully used for estimating pollutant exports from small catchment with predominant land use patterns. Do *et al.* [6] asserted that one advantage of the EC approach is its simple model format and spreadsheet-based mode of operation. White *et al.* [7] noted, however, that the small watershed monitoring data on which ECs were based often relied on data collected from distant watersheds with disparate conditions. The discrepancy between actual monitoring data and empirical ones had long been observed in literature. Liu and Lu [8] explained in their study that a great variance between actual ECs and experimental ones exists because nonpoint pollution is influenced by local factors such as geography, climate, soil, land use patterns, and precipitation among others. Despite their weaknesses, ECs are effective tools for predicting nutrient loads on watersheds.

2.2.2 Empirical modeling

To mitigate the uncertainties in ECs, QuickLET used a national database of localized ECs for ecoregions in the US developed by White *et al.* [7]. The database was created using a combination of available monitoring data and empirical modeling using Soil and Water Assessment Tool (SWAT). SWAT is a river basin- or watershed-scale modeling tool for quantifying and predicting the impacts of land management practices on water, sediment, and agricultural yields [9].

To perform the simulation, a modeling framework was built using available distributional data for land use, topography, climate, management, structural conservation practices, and soils in the contiguous US. The data served as input parameters for building a SWAT simulation for every land use in every ecoregion within the US. A Monte-Carlo technique was developed to randomly sample the basic input data layers in a statistically valid fashion to further reduce bias in load predictions. The SWAT predictions were stored in the database for subsequent grouping and statistical analysis.

2.2.3 National datasets used in QuickLET

To define a location, individual sample data was assigned a unique 8-digit, United States Geological Survey (USGS) Hydrologic Unit Code (HUC8) [10]. The probability that the sample will be assigned a particular HUC8 was based on HUC8 land area; larger HUC8s have higher probability of being selected. QuickLET obtained the HUC8s from the Watershed Boundary Dataset (WBD) at 1:250,000 scale, available

from US government agencies. Figure 2.1 shows the geographic information system (GIS)-based data consisting of 2,091 HUC8s.

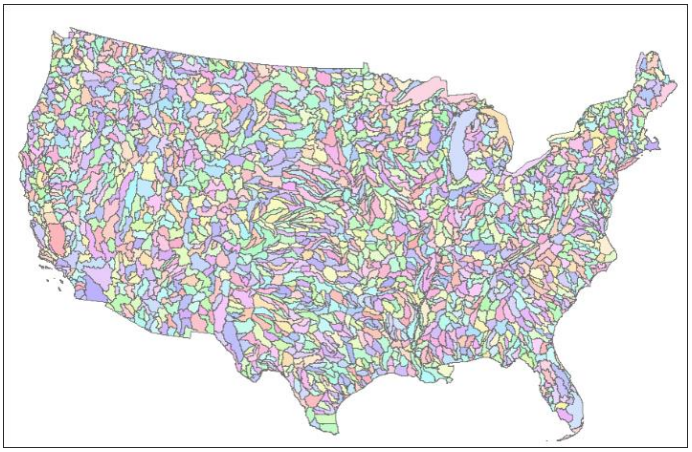

Figure 2.1. HUC8s at 1:250,000 resolution in polygon shapefile format.

Land use data in QuickLET was obtained from 2009 Cropland Data Layer (CDL) [11]. The data was simplified into 123 categories representing both cultivated and non-cultivated land uses. The probability that a sample would be assigned a given land use was based on the area of each land use with the assigned HUC8. Figure 2.2 shows the CDL used in QuickLET.

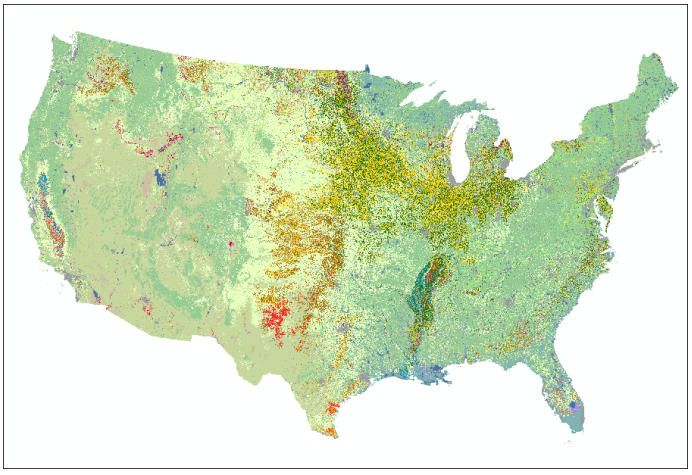

Figure 2.2. 2009 Cropland Data Layer in raster format.

Soils and topography for each sample were selected from a database developed for the Conservation Effects Assessment Project (CEAP) [12]. The CEAP database was created with soils from State Soil Geographic Database (STATSGO) [13], land cover from 2001 National Land Cover Data Sets (NLCD) [14], and Hydrologic Landscape Regions (HLR) of the United States [15].

In addition to the nationally available data, databases were created to store irrigation data in urban or agricultural land uses as well as management data for cultivated and non-cultivated croplands. Samples assigned as cultivated cropland were further attributed with structural conservation practices, which were derived from the CEAP database.

2.2.4 Simulated data

The simulated data that QuickLET used was obtained from the work of White *et al.* [7]. The method used was described by the authors as follows:

> A model constructor program was developed to perform the sampling procedure, construct a representative SWAT model, execute that model, summarize the results, and write those results to a master database. Input data were stored in either a Microsoft Access database or as preprocessed SWAT input files. Methods used to construct the simulations were derived from Texas BMP Evaluation Tool (TBET) (White et al., 2012). Once the model constructor finished a simulation and uploaded the results, the SWAT model was deleted, and the processes repeated drawing another sample until instructed to stop. A dedicated computer cluster of 40 computers (160 processing cores) connected via a private gigabit network running Windows 7 was used. Each machine can handle 8-12 threads of the model constructor software, allowing approximately 450 active instances of the constructor to run simultaneously. The system performs 7 million

simulations per day, and ran for six days, generating database of 45 million samples, representing a wide variety of conditions across the US. The large size of this database, allowed very specific queries with respect to land use, location, soils, or tillage to return a statistically useful number of samples to define a distribution.

2.2.5 "Distillation" of empirical data for QuickLET

A backend database consisting of 45-million rows would not be practical in a multi-user environment like the Web. Although SQL Server has a built-in optimizer that decides when to apply parallelism or serial execution to a query, there is an overhead cost to rebuilding online indexes as concurrent users access the database. Thus, it was necessary to "distill" the database in order to define the statistical distribution of each prediction within each HUC8. A non-parametric distributional definition procedure was applied to the database resulting in sampled data consisting of 639,844 rows. These data define the probability that a pollutant load will exceed a particular value. The sampled data were transformed into the main database supporting all of QuickLET's Web application queries.

Chapter 3

QuickLET's Design and Implementation

The three key considerations in the design of QuickLET were (1) its target users, (2) the system infrastructure on which it was to be deployed, and (3) the goals of the organization for which it was developed. At the outset, QuickLET's intended users were those with little or no experience in using hydrologic and other environmental modeling tools. The design of the Web user interface (UI) was heavily weighted on the usability of the tool and overall user experience. Along with its target users, considerations were given to the system infrastructure on which QuickLET was to be deployed, which required a Windows operating system. Thus, all the components in QuickLET's Web application were envisioned to run on a Microsoft Internet Information Services (IIS) Web server, a .NET Framework application server and a Microsoft SQL Server for its backend database. For maintainability, QuickLET was designed with modular components in order to easily add or remove features as needs arise. The following sections discuss the design decisions and rationale used in implementing QuickLET's features.

3.1 Architectural style

QuickLET's architecture adapted a three-layered style comprising presentation, business, and data layers. This style's distinct feature is the separation of tasks through its layers, thus allowing the implementation of design patterns at

code levels. Figure 3.1 illustrates the logical architectural style employed in QuickLET.

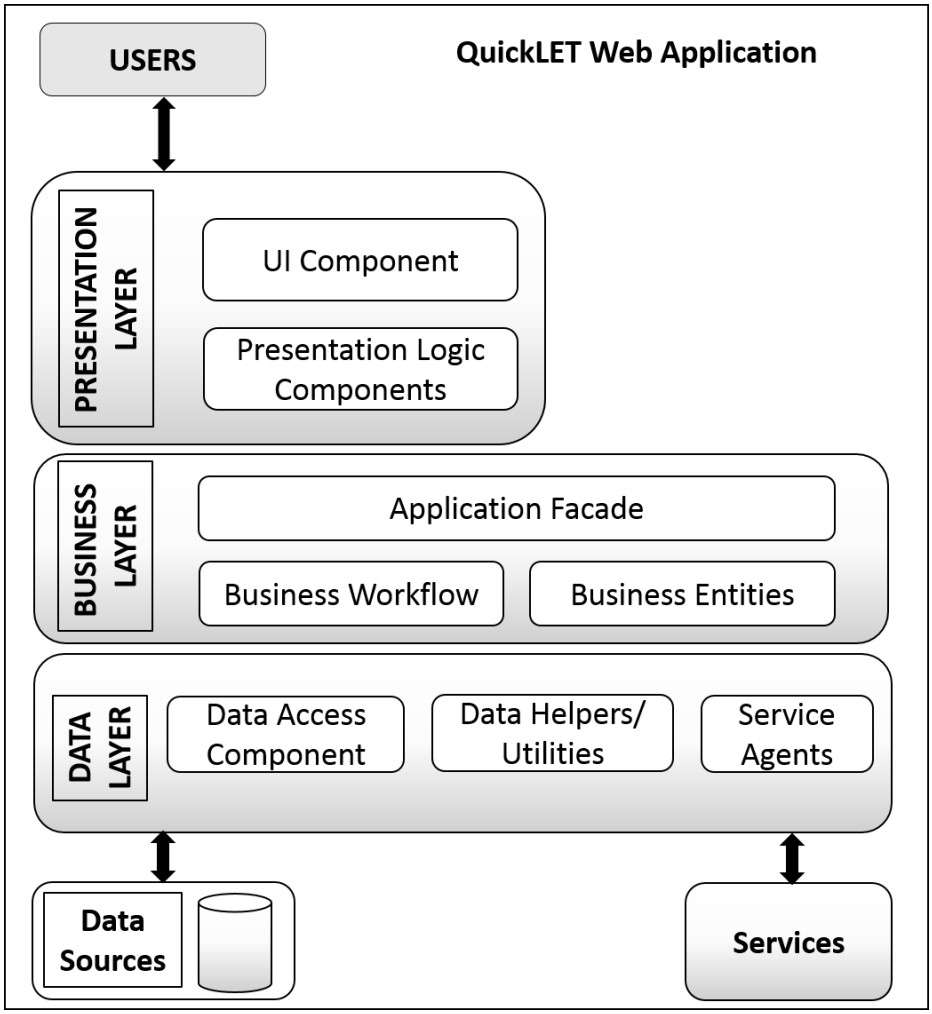

Figure 3.1. QuickLET's three-layered architectural style.

3.2 Software design pattern

With the selection of a three-layered architectural style for QuickLET, the Model-View-Controller (MVC) was a logical choice for a software design pattern. The MVC pattern reinforced the separation of concerns into three distinct areas: the *model*

(data layer), the *view* (presentation layer), and the *controller* (business layer). Figure 3.2 shows the MVC pattern for QuickLET.

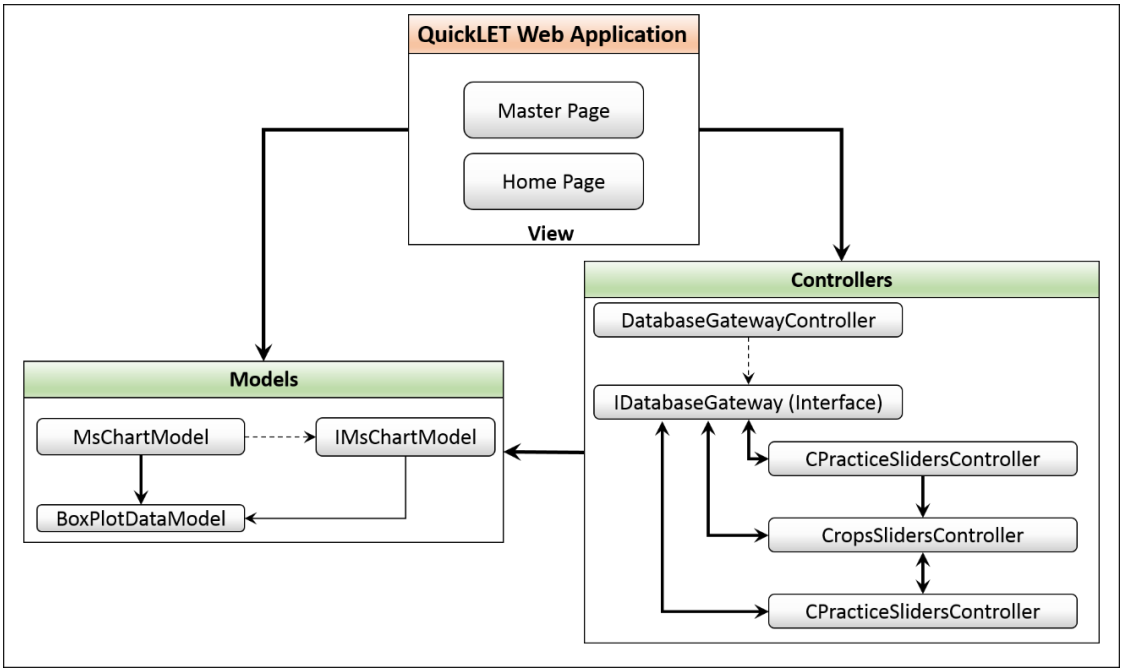

Figure 3.2. QuickLET's Model-View-Controller software design pattern.

3.2.1 Models

QuickLET models consisted of two classes and an interface for generating charts to display on the client side. The types of charts used to represent changes in the environment were:

- Pie charts - represented land use and crop data series;
- Column charts - represented conservation practices; and
- Box plot charts - represented four types of nutrient loads (water yield, sediment, nitrogen, phosphorus).

3.2.2 View

The view or presentation layer provided the graphical display of the results of controllers' actions. The Web UI controls that were selected to handle a user input consisted of a dropdownlist, charts and sliders. These UI controls enabled the least number of input requirements from the user. In addition, only one Web page handled the queries required to render the graphical display on the client side. This effectively eliminated the need to navigate between Web pages during the entire process.

As a Web application, QuickLET required no user credentials for accessing the tool. Upon entry to the Web site, the Home Web page displayed image placeholders for the charts. Two panels represented the current *(baseline)* and the *scenario* conditions. Figure 3.3 shows a snapshot of the Web site's home page when users first navigate to the Web portal.

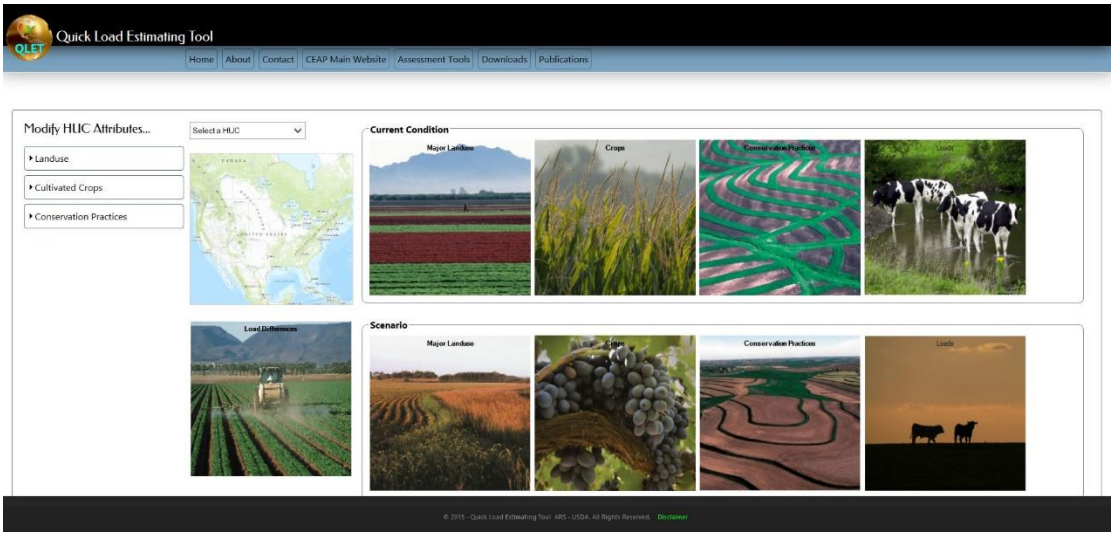

Figure 3.4. Screenshot of QuickLET's Home page upon entry to the Web site.

When users selected a HUC from the dropdownlist control, the image placeholders on the *baseline* panel were replaced by charts representing the HUC8-based output of the simulated model. The initial values of the charts in the *scenario* panel contained the same data as the baseline charts. This was intended to illustrate that no change was made to the baseline condition at this point. Figure 3.5 shows a graphical display of the baseline charts when users selected a HUC.

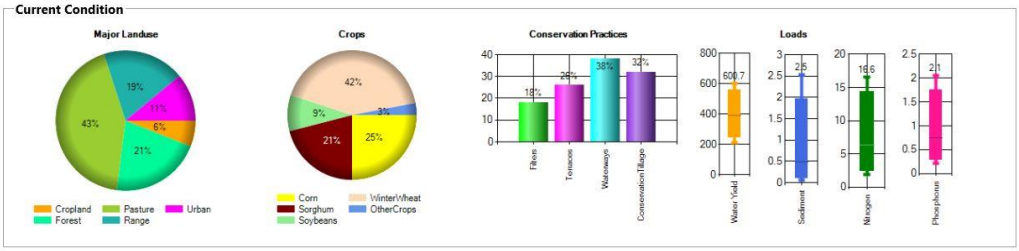

Figure 3.5. Baseline charts consisting of land use, crop, conservation practices, and nutrient loads.

Whenever users modified any of the sliders' values, the scenario panel would re-display the charts to reflect the corresponding change in the charts, i.e., if users moved any of the land use sliders, the scenario land use chart would reflect the change. The same would be true for cultivated crops and conservation practices sliders and charts. Figure 3.6 illustrates changes reflected in the scenario panel charts when users moved a slider.

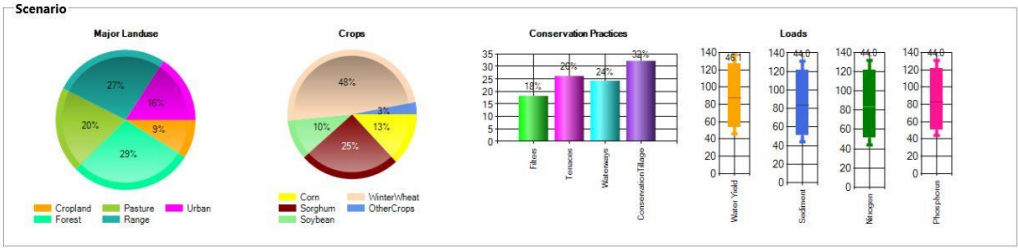

Figure 3.6. Scenario charts reflecting changes as users moved the sliders.

To achieve the same look and feel, QuickLET used ASP.NET's Master Web page template. The Master Web page contained Web UI components common to all the Web pages in the application. Figure 3.7 shows the common theme for the header and footer as implemented in the Master Web page.

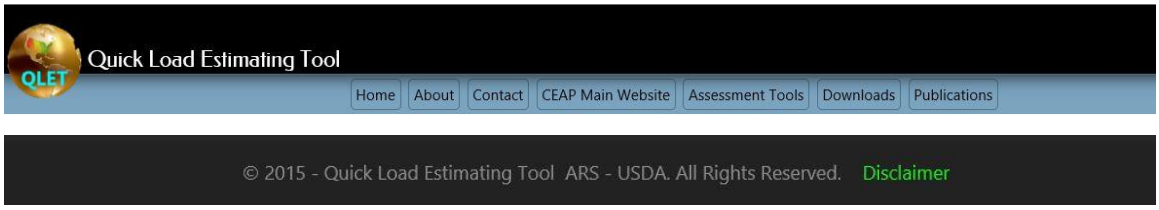

Figure 3.7. Master Web page header and footer common theme.

3.2.3 Controllers

The Controller classes provided the application's functional workflow. The controllers interacted with the models and forwarded requests to the server. One controller, *DatabaseGatewayController,* served as the entry point for all requests to the database. Other controllers responsible for updating the values of charts in the scenario panel of the Home page were:

LandSlidersController. This controller updated the values of the land use chart in the scenario panel every time users changed the value of land use sliders.

CropSlidersController. This controller updated the values of the crop chart in the scenario panel every time users changed the value of the crop sliders.

CPracticeSlidersController. This controller updated the values of the conservation practices chart in the scenario panel every time users changed the value of the conservation practices sliders.

3.3 Design rationale

The design of QuickLET involved weighing various options that would achieve the objectives set out at the beginning of the project. There were design options that would have made QuickLET perform better but were not pursued. Others were initiated at the start of the development phase, but were eventually scrapped due to the length of time needed to learn new methods or languages. Ultimately, QuickLET's design would continue to be re-assessed and its code refactored to take advantage of new features from emerging technologies. The following sections discuss the rationale used in selecting components for QuickLET's design.

3.3.1 Integrated Development Environment (IDE)

Visual Studio (VS) 2013 Ultimate edition was chosen as the IDE for the QuickLET project. The application design phase considered two Web development templates that were available in VS 2013: (1) the built-in MVC template and, (2) the ASP.NET Web form template. The MVC template applied the separation of concerns more strictly than the "classic" ASP.NET template since the model, view and controller were completely de-coupled from each other. The MVC template did not allow any server-side processing, thus, any server-side responses to user interactions with Web UI, such as buttons or text boxes, must be handled explicitly. There were also no stored procedures allowed in the MVC template for accessing the data layer, since it used the Microsoft Entity Framework technology to handle data access [16]. While the MVC template may be the ideal development template to use due to its strict

application of separation of concerns, it required a steep learning curve, especially for a developer used to the classic Web design environment. In the interest of time, the ASP.NET Web form template was chosen as development platform for QuickLET.

3.3.2 Charts and sliders

Several choices were available for the type of charts and sliders to use in the application. There were a number of pleasing commercial-off-the-shelf (COTS) charts and sliders. However, renewing licenses annually was an unsustainable proposition due to budgetary constraints. In addition, using proprietary COTS libraries would create a tight dependency between the external software product and QuickLET, which could be a maintainability issue over time. Initially, jQuery chart and slider API was used but the server-side processing requirements of a data-driven application were too numerous to be handled entirely on the client side. For example, fetching data from multiple tables in the SQL Server backend database and storing them in two-dimensional arrays for further analysis was easily handled using T-SQL database scripting language, but difficult to manipulate using JavaScript or jQuery. In the end, QuickLET implemented the Microsoft System's Data Visualization library for displaying the charts and Microsoft's AjaxControlToolkit for rendering the sliders. Since both are Microsoft products, the integration of the APIs within the Windows platform through the Nuget Package Manager was trivial, as both libraries allowed server-side processing.

3.3.3 Accordion panel

The decision to use accordion panel as a container for the sliders was due to its capability to hold multiple components in a single panel, and at the same time hide them when the components were not in use. This would allow for future addition of other land uses, crops and conservation practices. The accordion panel hid the sliders it contained by contracting, and showed them by expanding when users click it. Figure 3.8 shows snapshots of the accordion panel. The figure on the left is a snapshot of an inactive accordion panel while the figure on the right shows an expanded accordion panel when users select it.

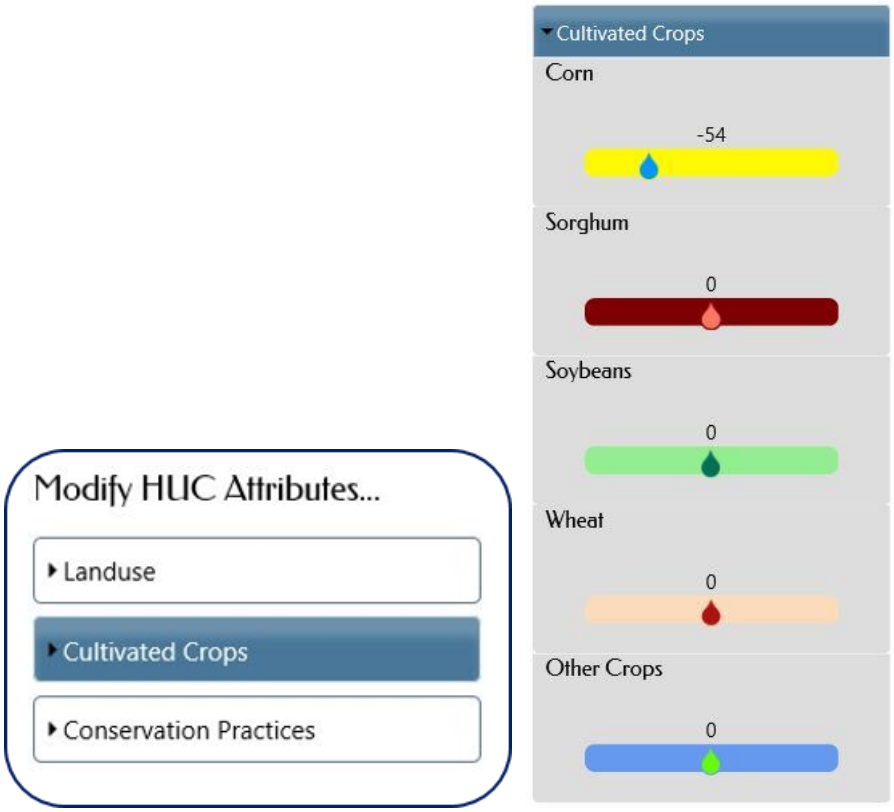

Figure 3.8. Accordion panel.

3.3.4 Cascading Stylesheet (CSS) and HTML

CSS and HTML were the markup languages used for rendering the View component of the application. QuickLET conformed to HTML 5 and CSS 3 Web standards. The header, menu, body, footer, and the accordion panel were built using a *Project Seven* tool [17]. Although a COTS product, the *Project Seven* CSS API required no annual subscription or expiration of ownership, allowed modifications to the software without penalties, and allowed usage of the API in multiple Web sites. The *Project Seven* tool was a plug-in for DreamWeaver Web design software. Since it used CSS and JavaScript for its code base, it was relatively straightforward to adapt it within the Microsoft development platform.

3.3.5 Web browsers

Internet Explorer (IE) version 11 was the main Web browser used during the development stage. QuickLET provided backward compatibility for previous versions of IE, down to IE6. Other browsers used during the development were Firefox, Chrome and Safari. There were subtle differences among the different browsers but no significant drawbacks were observed. However, QuickLET rendered and ran faster using the Firefox Web browser. Simple performance tests were conducted to compare the four browsers when selecting the dropdownlist and one of the sliders. The tests were conducted in debug mode on the development machine running 64-bit Windows 7 Professional Service Pack 1 with 16 GB RAM and 3.50 GHz CPU. Table 3.1 shows the results of the test.

Web Browser	Version	DropDownList (ms)	Land use (ms)	Crops (ms)	Conservation (ms)
Firefox	v36.0.4	66	831	804	819
IE	v11.0.17	74	901	880	922
Chrome	v42.0.2311.90	75	899	931	921
Safari	v5.1.7	76	889	893	908

Table 3.1. Comparison of Web browsers' performance.

3.3.6 Screen resolution

The optimal desktop screen resolution for QuickLET was 1280 x 720 pixels. Anything lower would cause QuickLET's charts to scroll down the screen. Users could easily adjust the screen resolution by zooming in or out in the Web browser. For smaller screen sizes like those in laptops, the Web site rendered better when zoom percentage was set at 67%.

3.3.7 GIS mapping

QuickLET provided users with a simple GIS mapping display through a Web Service hosted by Environmental Sciences Research Institute's (ESRI) ArcGIS Online [18]. ArcGIS Online allowed upload of shapefiles into the site. However, the size of a shapefile was limited to 10 MB. Some of the HUCs exceeded this size. A shapefile is a proprietary file format developed by ESRI, usually used in GIS. As a workaround to the size limit, the HUCs were converted from polygon to point shapefiles to reduce their sizes. Figure 3.9 shows the data conversion from polygon to point shapefile.

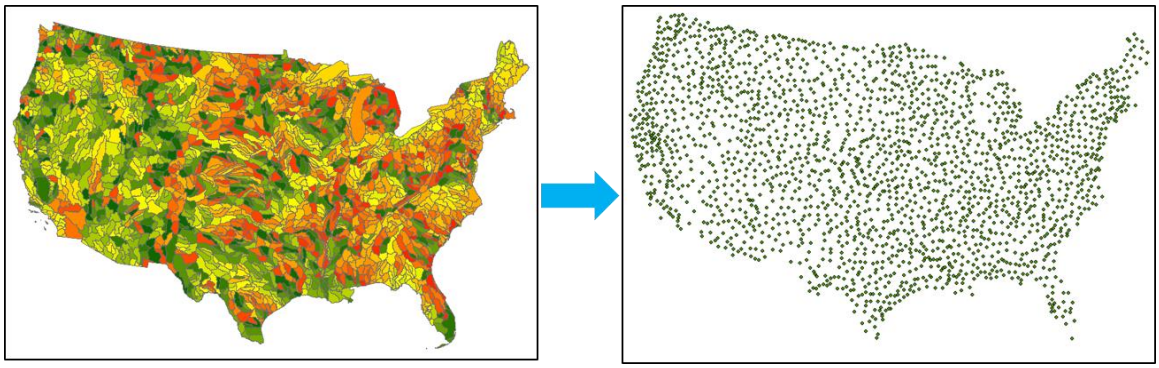

Figure 3.9. Conterminous US HUC8s converted from polygon to point shapefile format.

The points were not visible when the Web page first loaded. Toggling the zoom-in button made the points visible and clicking a point popped-up a window displaying the attributes of the HUC, which included its official name, latitude and longitude. Figure 3.10 shows snapshots of the Web map.

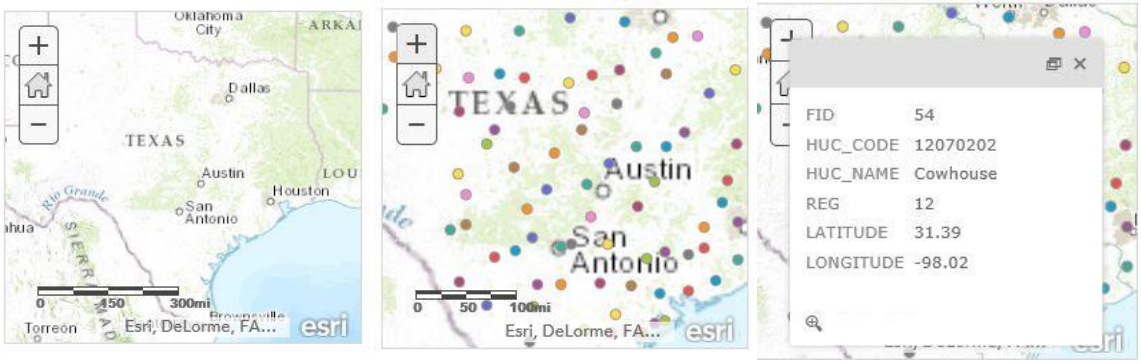

Figure 3.10. QuickLET's Web mapping display.

3.4 Database design

Database performance in SQL Server database is dependent on factors such as number of rows and columns in a table or number of join statements. In some instances, performance benefits from the normalization of tables; for example, if a table contains too many rows (e.g., exceeding a million). There are also occasions

when denormalization would be a better option; for example, if the database contains several tables with very few rows, combining them into one would probably result in better performance than issuing multiple join statements.

Initially, QuickLET's backend database contained a single table *(AllHUC8)* with 639,844 rows. This table had been distilled from the 45 million EC samples derived from SWAT modeling simulations. Despite this greatly reduced number, performance continued to be an issue, exceeding the Google Analytics' performance threshold of 200 milliseconds per server-side processing time [19]. To ease the performance bottleneck, the *AllHUC8* table was normalized. Figure 3.11 shows the entity relationships in QuickLET's database before normalization while Figure 3.12 shows the entity relationships after normalization.

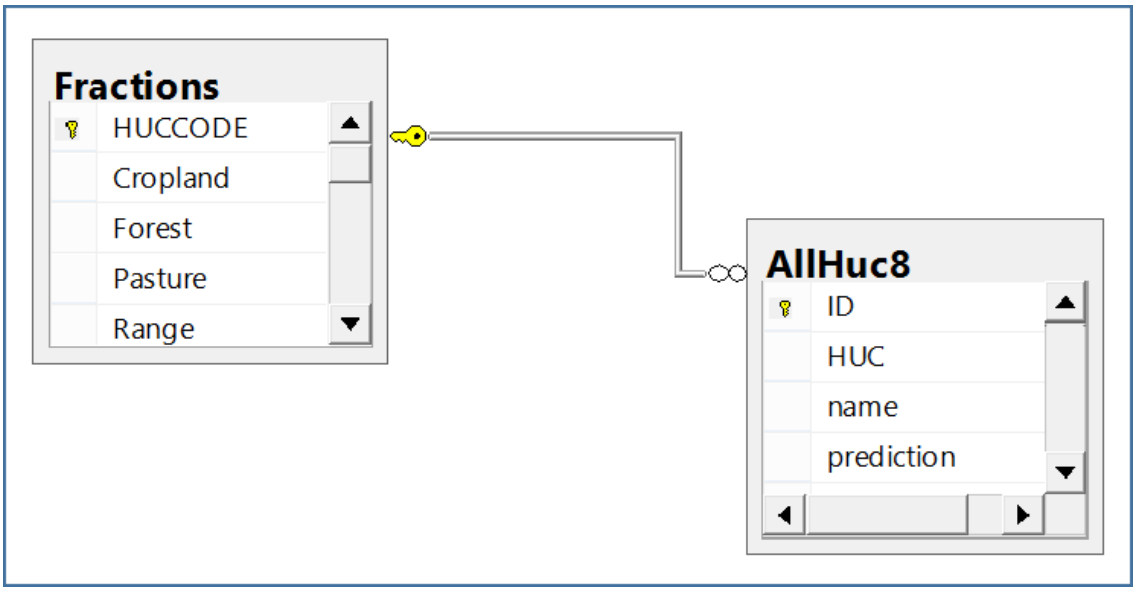

Figure 3.11. Entity relationship diagram before normalization.

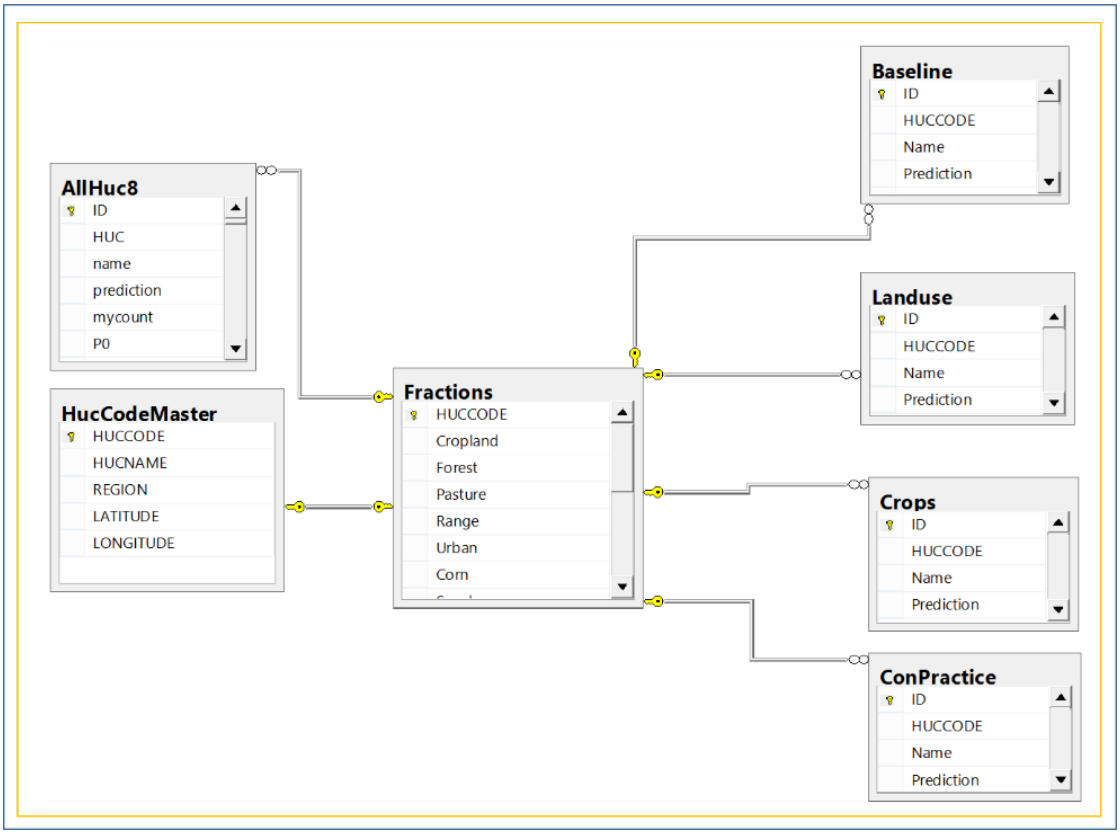

Figure 3.12. Entity relationship diagram after normalization.

AllHUC8 table was normalized into five tables: *Baseline, Landuse, Crops, HucCodeMaster,* and *ConPractice. AllHuc8* table included crops other than those covered in this report. To further reduce the number of rows in the *Crops* table, crops that were not part of this report were eliminated from the table. The final normalized *Crops* table contained 94,095 rows and included five crop classifications: *corn*, *sorghum*, *soybeans*, *wheat*, and *other crops*. No adjustment was made to the *Landuse* table. The final normalized *Landuse* table also contained 94,095 rows and included five land use classifications: *cropland*, *forest*, *pasture*, *rangeland*, and *urban*.

Like the *Crops* table, practices not covered by this report were eliminated from the normalized *ConPractice* table to reduce the number of unneeded rows. This resulted in the final normalized *ConPractice* table, containing 150,550 rows representing four conservation practice classifications: *filter strips*, *terraces*, *waterways*, and *conservation tillage*.

The normalized *HucCodeMaster* table contained unique data for each HUC8 and was used to populate the dropdownlist Web UI control. There were 2,091 unique rows in the table, representing each of the HUC8 in conterminous US.

The *Baseline* table contained the combined data from the normalized tables of *Crops*, *Landuse* and *ConPractice* tables totaling 338,740 rows.

The *Fractions* table represented the ECs for the HUC8s. The *Fractions* table served as the "heart" of the export coefficient methodology, and all the normalized tables, including the main sampling table, *AllHuc8*, had a one-to-many relationship with the *Fractions* table. Table 3.2 shows the table structure for *Fractions* table.

Columns	Data Type
HUCCODE	(PK, FK, nvarchar(8), not null)
Cropland	(float, null)
Forest	(float, null)
Range	(float, null)
Urban	(float, null)
Corn	(float, null)
Sorghum	(float, null)
Soybeans	(float, null)
WinterWheat	(float, null)
OtherCrops	(float, null)
Filters	(float, null)
Waterways	(float, null)
Terraces	(float, null)
ConservationTillage	(float, null)

Table 3.2. Fractions table structure.

3.5 Sample QuickLET prediction process

Watershed modeling is composed of complex processes involving numerous factors in the environment. Its primary objective is to predict the amount of pollutants that find their way into streams and other water bodies as water flows and nutrients are carried downstream the watershed. To illustrate how QuickLET handled its prediction functionalities, a concrete example is given below, using HUC8 with ID "12010001".

3.5.1 Baseline calculations

We start with the location of HUC 12010001, which is in hydrologic Region 12. Figure 3.17 shows its exact location in the State of Texas. The multi-colored HUCs on the left represent the different watersheds in Region 12 while the single HUC on the right represents HUC 12010001.

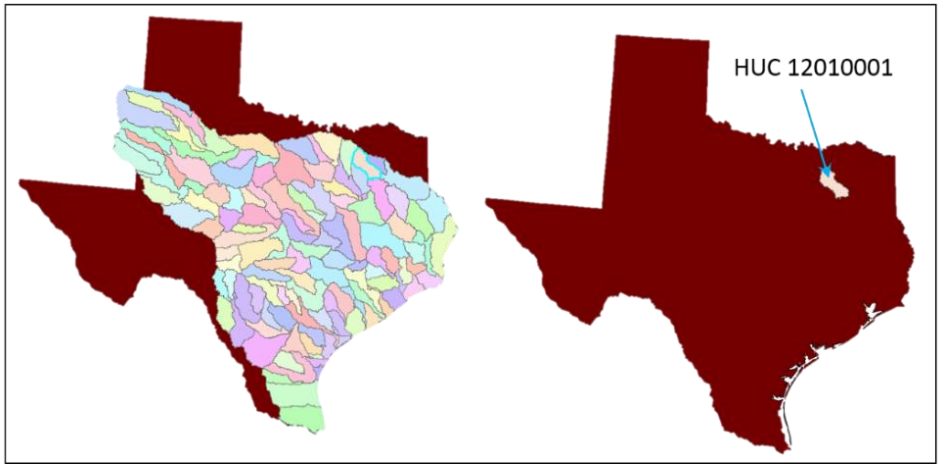

Figure 3.17. HUC 12010001 located in Upper Sabine of Region 12 in Texas.

Given HUC 12010001, QuickLET queried the *Fractions* table in the database and fetched the data matching the given ID. The data was used as input parameters

to calculate the distribution of land uses, crops and conservation practices found in the HUC. Figures 3.18 through 3.20 illustrate this process.

HUCCODE:	12010001	Percent
Cropland	0.0621530531324346	6%
Forest	0.20524385408406	21%
Pasture	0.426695083267248	43%
Rangeland	0.194785884218874	19%
Urban	0.111122125297383	11%

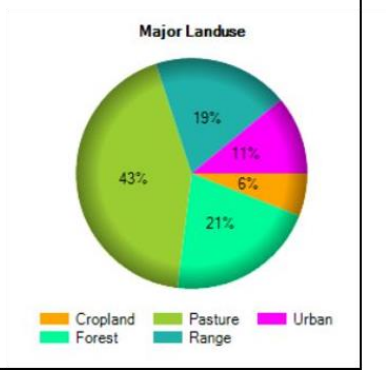

Figure 3.18. Calculating baseline land use with export coefficients.

HUCCODE:	12010001	Percent
Corn	0.250398724082935	25%
Sorghum	0.212918660287081	21%
Soybeans	0.0893141945773525	9%
Winter Wheat	0.41866028708134	42%
Other Crops	0.0287081339712918	3%

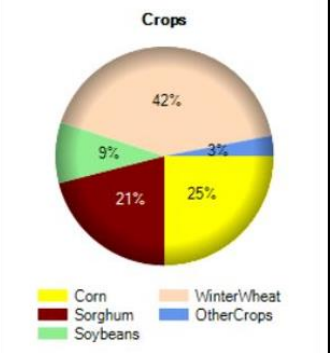

Figure 3.19. Calculating baseline crops with export coefficients.

HUCCODE:	12010001	Percent
Filters	0.181020733652313	25%
Waterways	0.25518341307815	21%
Terraces	0.384370015948963	9%
Tillage	0.321371610845295	42%

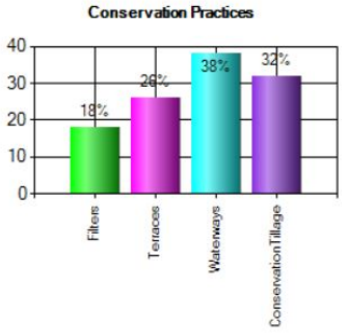

Figure 3.20. Calculating baseline conservation practices with export coefficients.

An algorithm was developed to calculate the edge-of-field nutrient loss—the amount of water, sediment, nitrogen, phosphorus—carried downstream from the HUC. QuickLET employed units of measure for water yield in millimeter per year; sediment loading in milligram per hectare per year; and nitrogen and phosphorus in kilogram per hectare per year. The results of calculating the current nutrient loads in the HUC were displayed in the form of box plot charts. Figure 3.19 shows the algorithm for estimating the nutrient loads while Figure 3.22 illustrates the resulting box plot charts.

```
1   public List<BoxPlotDataModel> CalculateBaselineNutrientLoads(string huccode)
2   {
3       IDatabaseGatewayController gwc = new DatabaseGatewayController();
4       DataTable baselineDt = new DataTable();
5       baselineDt = gwc.GetBaselineBoxPlotDataByHuc(huccode);
6       double?[] b_lulcFrac = new double?[5];
7       b_lulcFrac = gwc.GetBaselineLanduse(huccode);
8       double?[,] B_Sum = new double?[4, 101];
9
10      //Add landuses and other crops together
11      for (int i = 0; i < 4; i++)
12      {
13          for (int j = 0; j < 101; j++)
14          {
15                  B_Sum[i, j] = (B_Sum[i, j] == null) ? 0 : B_Sum[i, j];
16                  B_Sum[i, j] = B_Sum[i, j] + (baselineDt.Rows[i][j] as double?) * b_lulcFrac[i];
17          }
18      }
19      List<BoxPlotDataModel> load = new List<BoxPlotDataModel>
20      {
21          new BoxPlotDataModel("Water Yield", B_Sum[3,2], B_Sum[3,98], B_Sum[3,5], B_Sum[3,95], B_Sum[3,50]),
22          new BoxPlotDataModel("Sediment", B_Sum[0,2], B_Sum[0,98], B_Sum[0,5], B_Sum[0,95], B_Sum[0,50]),
23          new BoxPlotDataModel("Nitrogen", B_Sum[1,2], B_Sum[1,98], B_Sum[1,5], B_Sum[1,95], B_Sum[1,50]),
24          new BoxPlotDataModel("Phosphorus", B_Sum[2,2], B_Sum[2,98], B_Sum[2,5], B_Sum[2,95], B_Sum[2,50])
25      };
26      return load;
27  }
```

Figure 3.21. Algorithm for calculating baseline edge-of-field nutrient loads.

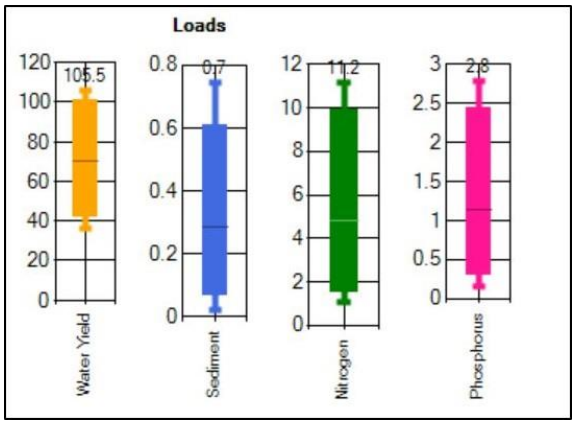

Figure 3.22. Baseline nutrient loads.

3.5.2 Calculating changes to land use, crops and conservation practices

Moving the sliders represented changes in land use, crops or conservation practices. The baseline values were used for comparison when evaluating the effects of these changes on nutrients, sediment loading and water yield on streams or water bodies within the HUC. Estimates ranged from minimum and maximum to low, high, and median values. Figure 3.23 shows the UML Activity Diagram reflecting changes in the scenario conditions.

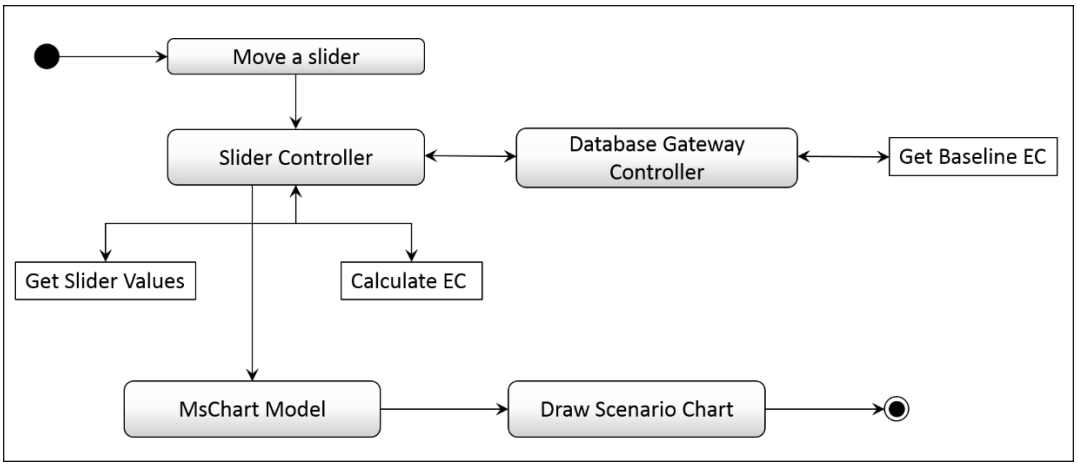

Figure 3.23. UML Activity Diagram for calculating and rendering scenario charts.

Calculating the scenario land use involved values from the land use sliders and ECs from the *Fractions* table. The values of the sliders and the ECs were stored in separate arrays. An algorithm for performing the calculations is shown in Figure 3.24.

```
1   public DataTable CalculateScenarioLanduse(double?[] lulcSliders, string huccode)
2   {
3       double?[] p_lulc_frac = new double?[5];
4       double? S_Area = 0;
5       IDatabaseGatewayController gwc = new DatabaseGatewayController();
6       double?[] baseLulc = new double?[5];
7       baseLulc = gwc.GetBaselineLanduse(huccode);
8
9       //Compute sum of land use sliders
10      double? sliderSum = 0;
11      for (int i = 0; i < 5; i++)
12      {
13          sliderSum += (lulcSliders[i] / 100 + 1) * baseLulc[i];
14      }
15      //Loop through landuses
16      for (int i = 0; i < 5; i++)
17      {
18          p_lulc_frac[i] = ((lulcSliders[i] / 100 + 1) * baseLulc[i]) / sliderSum;
19      }
20      //Format p_lulc_frac to percent with zero decimal place
21      //Create DataTable to hold land use values
22      DataTable scenLulc = new DataTable();
23      scenLulc.Columns.Add("LULC");
24      scenLulc.Columns.Add("Fraction");
25      //Store p_lulc_frac into scenLulc
26      return scenLulc;
27  }
```

Figure 3.24. Algorithm for calculating scenario land use.

The result of the calculations was used to create a table that was passed as parameter to the chart model for rendering the scenario land use pie chart. Figure 3.25 shows the pie chart generated by a change in land use. The slider was moved to its 25th position.

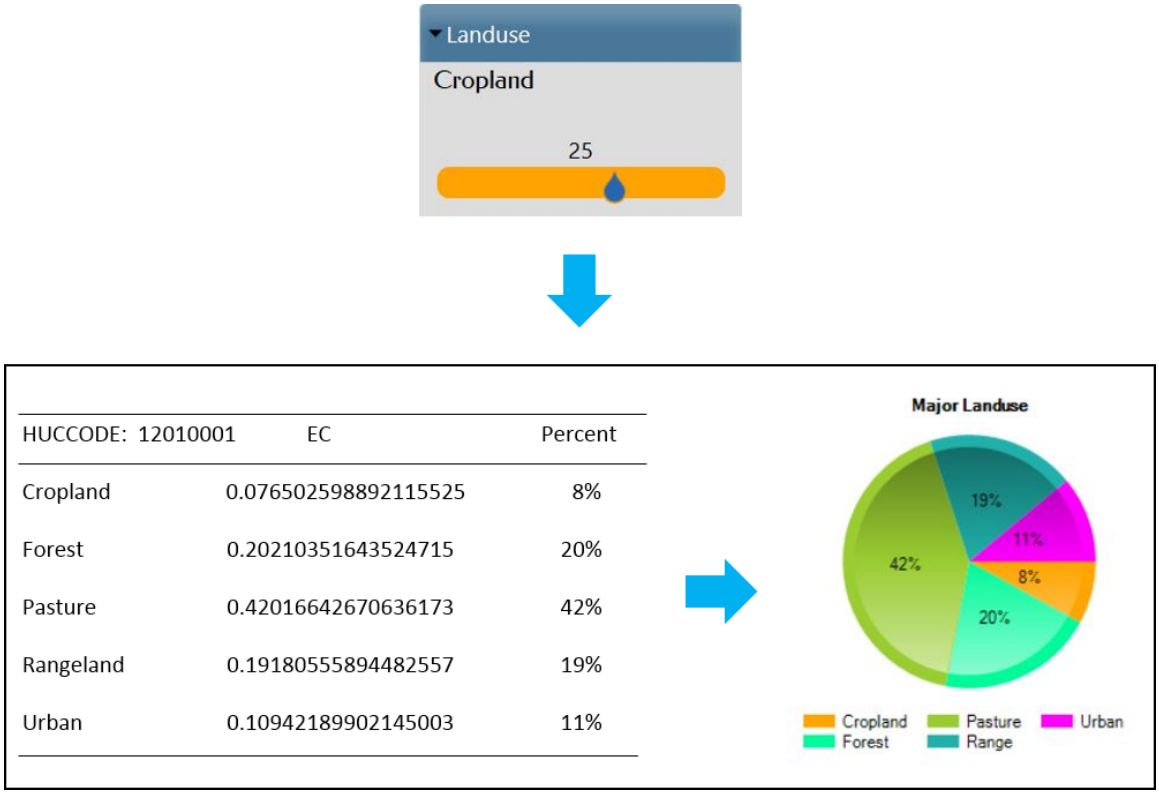

Figure 3.25. Calculating changes to scenario land use chart.

Similar algorithm and methodology were used to calculate changes to the scenario types of cultivated crops and conservation practices. Figures 3.26 and 3.27 show the algorithms used to calculate the respective changes in cultivated crops and conservation practices.

```
1   public DataTable CalculateScenarioCropsBySlider(double?[] cropSliders, string huccode)
2   {
3       IDatabaseGatewayController gwc = new DatabaseGatewayController();
4       double?[] b_CropsFrac = new double?[5];
5       b_CropsFrac = gwc.GetBaselineCrops(huccode);
6
7       //Compute pre-adjusted fractions
8       double?[] p_crop_frac = new double?[5];
9       p_crop_frac = ComputeCropsPreAdjustFractions(cropSliders, huccode);
10      //Compute sum of pre-adjusted fractions
11      double? sumPreAdj = 0;
12      foreach(double? dbl in p_crop_frac)
13      {
14          sumPreAdj += dbl;
15      }
16      //Compute scenario fractions
17      double?[] s_crop_frac = new double?[5];
18      double? sumCropFrac = 0;
19      for (int i = 0; i < b_CropsFrac.Length; i++)
20      {
21          s_crop_frac[i] = p_crop_frac[i] / sumPreAdj;
22          sumCropFrac += s_crop_frac[i];  //this should add to 1 or 0.99
23      }
24      //Format s_crop_frac percent to zero decimal place
25      //Create DataTable to hold s_crop_frac values
26      DataTable dtcrop = new DataTable();
27      dtcrop.Columns.Add("Crops");
28      dtcrop.Columns.Add("Fraction");
29      return dtcrop;
30  }
```

Figure 3.26. Algorithm for calculating scenario cultivated crops.

```
1   public DataTable CalculateScenarioCp(double?[] cpSliders, string huccode)
2   {
3       //This creates a DataTable used for updating S_cp chart
4       IDatabaseGatewayController gwc = new DatabaseGatewayController();
5       DataTable cpdata = new DataTable();
6       cpdata = gwc.GetCPWithPracticeData(huccode);
7
8       //Calculate new CP values
9       double?[] b_cpFrac = new double?[4];
10      b_cpFrac = gwc.GetBaselineCp(huccode);
11
12      double? new_filter_frac = 0;
13      new_filter_frac = b_cpFrac[0] + cpSliders[0] / 100;
14      //Set new_filter_frac limits to range 0-1
15      double? new_terrace_frac = 0;
16      new_terrace_frac = b_cpFrac[1] + cpSliders[1] / 100;
17      //Set new_terrace_frac limits to range 0-1
18      double? new_waterways_frac = 0;
19      new_waterways_frac = b_cpFrac[2] + cpSliders[2] / 100;
20      //Set new_waterways_frac limits to range 0 -1
21      double? new_tillage_frac = 0;
22      new_tillage_frac = b_cpFrac[3] + cpSliders[3] / 100;
23      //Set new_tillage_frac limits to range 0 -1
24
25      //Format new values to percent with zero decimal place
26      //Create DataTable to hold new values
27      DataTable dtcp = new DataTable();
28      dtcp.Columns.Add("CP");
29      dtcp.Columns.Add("Fraction");
30      return dtcp;
31  }
```

Figure 3.27. Algorithm for calculating scenario conservation practices distribution.

3.5.3 Calculating scenario nutrient loads

Box plot charts were used to display the estimated nutrient loads because they are non-parametric: they display a range of values without making any assumptions of the underlying statistical distribution [20]. QuickLET selected values from the simulated data at the 2nd, 98th, 5th, 95th, and the 50th percentiles and assigned them as

minimum, maximum, low, high, and median values respectively. Figure 3.28 shows sample box plot charts for the scenario conditions when a land use slider was moved to the 25th position.

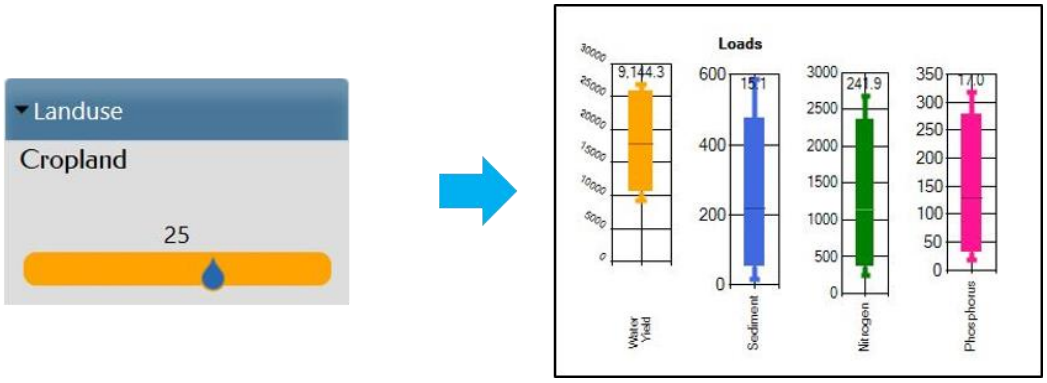

Figure 3.28. Scenario box plot charts.

Figure 3.29 shows a UML Activity Diagram with steps for calculating the nutrient loads due to changes in scenario conditions.

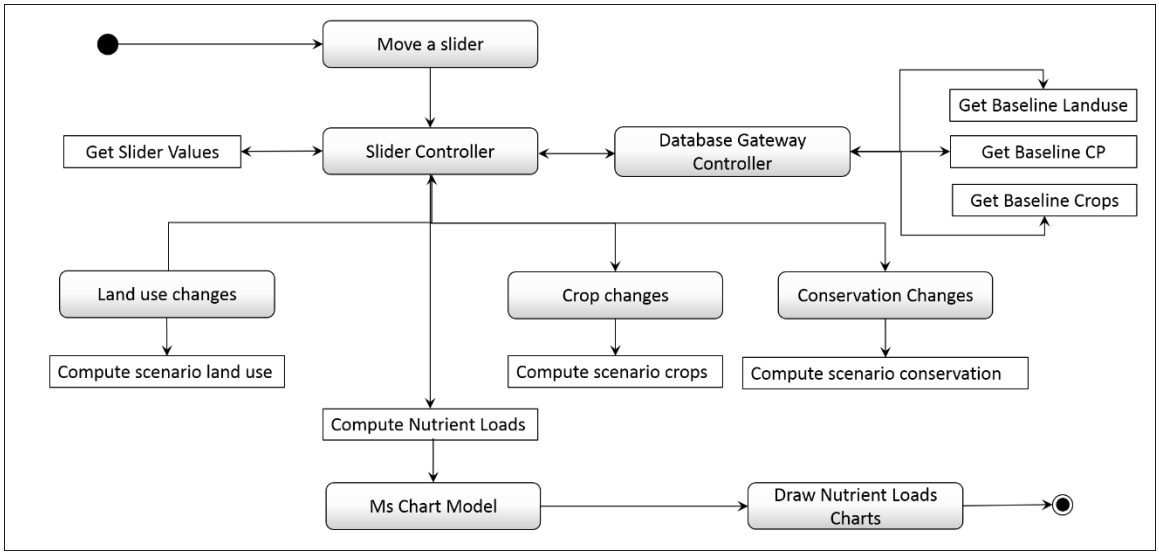

Figure 3.29. UML Activity Diagram for calculating scenario nutrient loads.

The algorithm for calculating the nutrient loads due to changes in scenario conditions is shown in Figure 3.30.

```
1   public List<BoxPlotDataModel> UpdateScenarioNutrientLoads(double?[] lulcSliders, double?[]
2           cropSliders, double?[] cpSliders, string huccode)
3   {
4       IDatabaseGatewayController gwc = new DatabaseGatewayController();
5       LanduseSlidersController ils = new LanduseSlidersController();
6       DataTable allBaselineDt = new DataTable();
7       allBaselineDt = gwc.GetAllBoxPLotDataByHuc(huccode);
8       double?[] b_lulcFrac = new double?[5];
9       b_lulcFrac = gwc.GetBaselineLanduse(huccode);
10      // Calculate nutrient loads
11      double?[,] croplandMultiplier = new double?[5,101];
12      croplandMultiplier = ils.GetCroplandMultiplier(cropSliders, lulcSliders, huccode);
13      double?[,] filterMultiplier = ComputeCpFilterMultiplier(cpSliders, huccode);
14      double?[,] terraceMultiplier = ComputeCpTerraceMultiplier(cpSliders, huccode);
15      double?[,] waterwayMultiplier = ComputeCpWaterwayMultiplier(cpSliders, huccode);
16      double?[,] tillageMultiplier = ComputeCpTillageMultiplier(cpSliders, huccode);
17      double?[,] S_Sum = new double?[4, 101];
18
19      for (int i = 0; i < 4; i++)
20      {
21          for (int j = 0; j < 101; j++)
22          {
23              S_Sum[i, j] = S_Sum[i, j] + (allBaselineDt.Rows[i][j] as double?) * b_lulcFrac[0] *
24                  croplandMultiplier[i, j] * filterMultiplier[i, j]
25                  * terraceMultiplier[i, j] * waterwayMultiplier[i, j] * tillageMultiplier[i, j];
26          }
27      }
28      List<BoxPlotDataModel> load = new List<BoxPlotDataModel>
29      {
30          new BoxPlotDataModel(wtryield, sediment, nitrogen, phosphorus)
31      };
32      return load;
33  }
```

Figure 3.30. Algorithm for calculating scenario nutrient load values.

3.5.4 Load differences between baseline and scenario estimates

Load differences represented the percent increase or decrease in pollutant loads between the baseline and scenario. To calculate the percentage differences, sums of nutrient loads from both baseline and scenario conditions were calculated

and values from the 50th percentile (median) were used for the load differences chart.

Figure 3.31 shows the algorithm for calculating the load differences.

```
1   public DataTable CalculateLoadDifferences(List<BoxPlotDataModel> scenbp,
2           List<BoxPlotDataModel> basebp, string huccode)
3   {
4       List<BoxPlotDataModel> diffbp = new List<BoxPlotDataModel>()
5       {
6               new BoxPlotDataModel(),
7               new BoxPlotDataModel(),
8               new BoxPlotDataModel(),
9               new BoxPlotDataModel()
10      };
11      //Get median values
12      diffbp[0].Q2_med = ((scenbp[0].Q2_med - basebp[0].Q2_med) / basebp[0].Q2_med); ); //Water Yld
13      diffbp[1].Q2_med = ((scenbp[1].Q2_med - basebp[1].Q2_med) / basebp[1].Q2_med); ); //Sediment
14      diffbp[2].Q2_med = ((scenbp[1].Q2_med - basebp[2].Q2_med) / basebp[2].Q2_med); //Nitrogen
15      diffbp[3].Q2_med = ((scenbp[3].Q2_med - basebp[3].Q2_med) / basebp[3].Q2_med); //Phosphorus
16
17      //Format diffbp to percent with zero decimal place
18      diffbp[0].Q2_med = Convert.ToDouble(String.Format("{0:0}", diffbp[0].Q2_med * 100));
19      diffbp[1].Q2_med = Convert.ToDouble(String.Format("{0:0}", diffbp[1].Q2_med * 100));
20      diffbp[2].Q2_med = Convert.ToDouble(String.Format("{0:0}", diffbp[2].Q2_med * 100));
21      diffbp[3].Q2_med = Convert.ToDouble(String.Format("{0:0}", diffbp[3].Q2_med * 100));
22
23      //Create a DataTable to hold diffbp data
24      DataTable diffDt = new DataTable();
25      DataRow drow = diffDt.NewRow();
26      diffDt.Columns.Add("type");
27      diffDt.Columns.Add("diff");
28      //Store diffbp into diffDt and return
29      return diffDt;
30  }
```

Figure 3.31. Algorithm for calculating differences between baseline and scenario nutrient loads.

Chapter 4

Code and Performance Metrics

The role of software measurement in software development life cycle cannot be overstated. Some described how software measurements could be used to estimate projects and monitor their progress [21] while others observed that software measurements could be used to gauge the quality and maintainability of software [22]. There were those who viewed the importance of software measurements from management objectives of prediction, progress and process improvement [23]. Despite the significant role that software measurements play in ensuring software quality, very few of the software metrics in industry have been examined closely from a measurement method perspective due to lack of agreed-upon frameworks of verification and validation [24]. For Web application projects, the lack of consensus on software measurement standards makes it difficult to find appropriate and reliable benchmarks. In the context of QuickLET, there were no historical data to derive credible Web application metrics. In addition, the various types of Web browsers and lack of Web browser standards made establishing metrics an intricate exercise since what worked with one browser may not work with another. Despite these limitations, certain measurements and metrics were applied to QuickLET to gauge its performance and assess the structure of its source codes. Tests were performed using Visual Studio 2013 Ultimate's built-in code analyzer. No actual tests were performed on the production server. However, a Web application

monitoring tool from Microsoft, *Application Insights (AI)*, was integrated into Visual Studio to allow the monitoring of QuickLET from both the client and server sides. The results of the code analysis and AI monitoring metrics are discussed below.

4.1 Code metrics

Visual Studio's built-in code analyzer was used to calculate the code metrics for QuickLET. The analyzer include the following metrics:

- Maintainability Index (MI) - represents the relative ease of maintaining the code. The maintainability index score is always in the range 0–100. A high value means better maintainability. MI ranking is shown in Table 4.1.

MI score	Rank	Maintainability
100 – 20	A	Very high
19 – 10	B	Medium
9 – 0	C	Extremely low

Table 4.1. Maintainability Index score.

- Cyclomatic Complexity –measures the structural complexity of the code. The code analyzer calculates the number of different code paths in the flow of the program. A high value means a program has complex control flow, requires more tests to achieve good code coverage and is less maintainable.
- Lines of Code (LOC) – indicates the approximate number of lines in the code. The LOC usually does not include comments or blank spaces, but Visual Studio does not provide an exact definition of this metric.

- Depth of Inheritance – indicates the number of class definitions that extend to the root of the class hierarchy. Deeper hierarchies signify a program that is difficult to understand.
- Class Coupling – measures the coupling to unique classes through parameters.

The overall code analysis results for QuickLET is shown in Table 4.2.

Components Measured	Maintainability Index	Cyclomatic Complexity	Lines of Code	Depth of Inheritance	Class Coupling
Models	82	73	334	1	24
Controllers	75	345	857	1	31
QLETWeb	91	156	635	5	73

Table 4.2. Code metrics results for QuickLET.

QuikLET scored high in Maintainability Index. However, its Cyclomatic Complexity and Class Coupling scores were also very high, which warranted further investigation as to which codes were responsible for the high marks. A look at the models showed that *MsChartModel* was mainly responsible for the complexity in the code. A further inspection revealed that since the model performed the drawing behavior for all 15 charts, it was incurring the burden of the code paths for all drawing method calls. The Controller classes also showed high complexity across the board. Any future rework or refactoring must therefore investigate further the results of this analysis, especially those with high complexity scores.

4.2 Microsoft Application Insights (AI)

Metrics for monitoring client-side requests and server-side responses were measured using Microsoft's *Application Insights* (AI) tool, which provided telemetry services via Windows Azure. The tool was added to the production server's Web application. AI was also integrated within the Visual Studio IDE, which required a Windows Azure subscription account. Since QuickLET was not yet officially launched at the time of this writing, the measurements collected reflect mostly testing data. The sample metrics, however, helped to assess the current "health" of the application. Following are some of the measures collected.

4.2.1 Client-side metrics

The client-side metrics measured the performance in the client browser, e.g., how fast the Web page loaded and which part of the process took the most time. The topmost metrics, *Client Processing Time* and *Send Request Time,* are the key indicators for this metric.

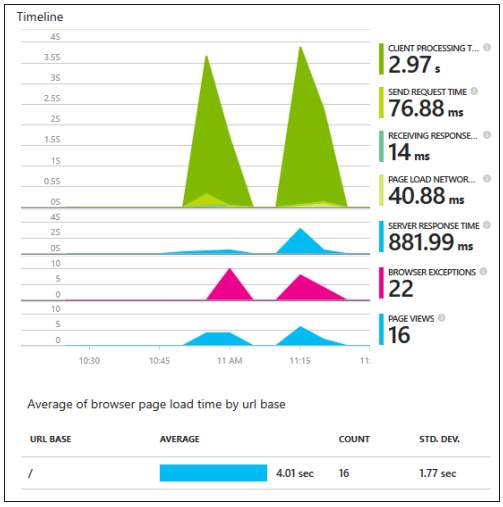

Figure 4.1. Client-side metrics.

4.2.2 Server-side metrics

These metrics measured performance or load at the production server. Figure 4.2 shows the metrics for QuickLET's hosting server.

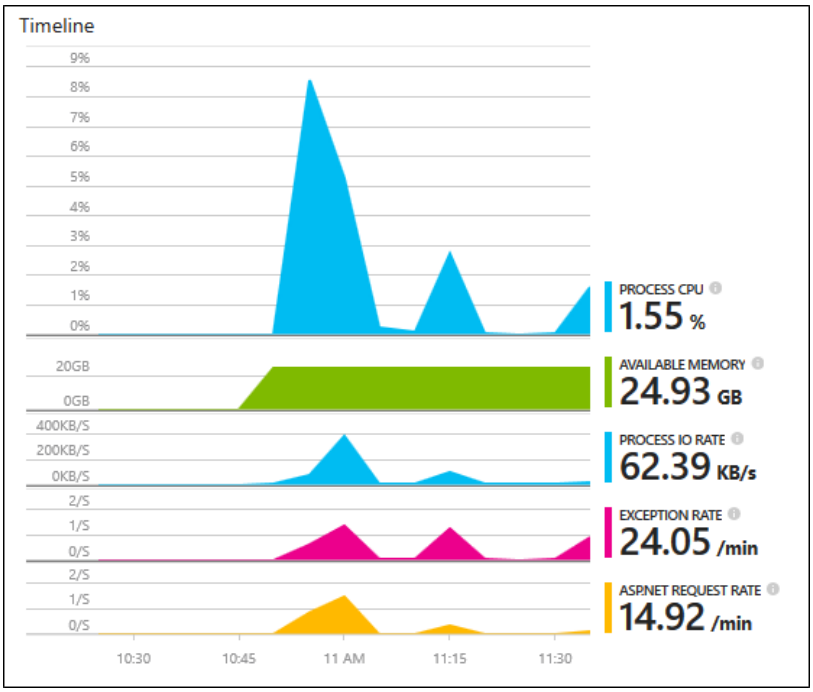

Figure 4.2. Server-side metrics.

4.3 Deployment to production server

Deployment to a production server was a three-step process, following the three-layered architectural design: (1) Web application deployment; (2) database deployment; and (3) creating a package and publishing it to IIS Web server.

4.3.1 Internet Information Services (IIS) 7.0

The Microsoft IIS 7.0 in Windows Web Server 2008 R2 provided the platform for administering and hosting the QuickLET Web application. A *web.config* file stored the IIS settings that are specific to the QuickLET Web application. IIS 7.0 has five main

components consisting of application pool, worker process (w3wp.exe), http.sys, svchost.exe, and configuration store [25].

An IIS application pool is a grouping of URLs that is routed to one or more worker processes. Since several Web applications may share one or more worker processes, application pools provide a convenient way to manage a Web infrastructure. QuickLET was setup to run on an application pool having a unique name so that it is isolated from problems in other application pools.

An IIS worker process is a Windows process responsible for handling requests sent to a Web server for specific application pool. The *http.sys* listens to requests in kernel mode and passes it to the *svchost.exe* through the World Wide Web publishing service (w3svc). The configuration store is the storage unit for all *web.config* files, ASP.NET settings, and other configuration.

4.3.2 Deploying QuickLET to IIS 7.0

The QuickLET Web application was deployed to IIS 7.0 using a publishing profile created with Visual Studio 2013. A domain name was previously obtained before deploying the Web application and all security requirements cleared by the IT administrator.

4.3.3 Deploying the database to SQL Server 2012

Database deployment to Microsoft SQL Server 2012 Standard edition was performed by copying the database from the development machine into the production server. This method required stopping the SQL Server and SQL Server

Agent services from both machines and restarting them after copying. The *web.config* file was updated based on the production server settings. The database may be deployed before or after the deployment of the Web application. After the deployment of the database, necessary user permissions were configured.

Chapter 5

Conclusion

This report presented QuickLET's Web application design and the export coefficient (EC) approach it implemented in delivering pollutant load estimates within a watershed. The three-layered architecture and MVC software design pattern on which QuickLET's framework was built upon weighed various options in selecting the Web UI controls that were necessary to accomplish the development objectives. QuickLET used a final database of over 600,000 EC samples. The samples represented combinations of land use, soils, location, climate, time, topography, and conservation practices. Most watershed modeling tools that exist today have limited scope, largely because watershed data are localized in nature. The national scope of QuickLET's pre-processed data, randomly drawn from 45 million SWAT simulations, provided QuickLET with a unique predictive capability not found in today's existing Web-based watershed modeling tools. QuickLET's modeling uncertainty is acceptable for initial scoping. If the level of uncertainty exceeds the decision maker's tolerance, additional rigor and more resource-intensive analysis may be required.

5.1 Related work

Web-based modeling tools assessing the impact of land use, crop cultivation, or structural conservation practices on pollutant loads are difficult to develop owing to the complex nature of environmental modeling. Currently, few websites provide these types of tools for general public use. Among the few that exist, is the Spatially

Referenced Regressions on Watershed (SPARROW) developed by the US Geological Survey (USGS). SPARROW integrates monitoring data with landscape information on national, regional, and basin-wide levels [26]. The tool, designed for water-resources managers across the United States, requires domain knowledge of water quality assessments. A Web-based decision support system has been developed to allow the use of SPARROW predictions by the public. The *My Waters Mapper* tool available from the US Environmental Protection Agency's (EPA) website provides static information on water quality assessments for a specified geographic location [27]. Purdue University's Load Duration Curve (LDC) tool enables the collection of water quality data and provides suggestions for best management practices scenarios for reducing annual pollutant load [28]. The LDC tool derives data from EPA and USGS through direct links to those websites. The Soil Nutrient Assessment Program (SNAP) tool, developed by the Agricultural Research Service (ARS) for the Texas State Soil and Water Conservation Board, assists Texas farmers in planning for their next year's fertilizer inputs [29]. The Small Watershed Nutrient Forecasting Tool (SWIFT), also developed by the ARS, provides the same estimation methodologies as QuickLET, except on a larger scale instead of at HUC8-level [30].

5.2 Future work

Although QuickLET, in its present design and implementation, fulfilled the objectives it set out to do, some areas of improvement remain. Among these, in order of priority, are as follows:

- *Re-design the architecture using VS 2013 MVC template*. The poor performance test results and high code complexity scores were due essentially to its current architecture. The tight coupling QuickLET had with its database would be avoided with the use of enhanced MVC pattern since data models could be re-created easily, without affecting its code base.
- *Add a Web service.* Adding a Web service that exposes QuickLET's EC method for estimating pollutant loads would enable other Web sites or users to access this functionality, without having to obtain, process and maintain the associated data.
- *GIS geoprocessing.* Providing thematic maps for viewing changes in the scenario landscape would provide added visuals to users. However, this enhancement should be weighed against additional efforts needed to create a separate View page due to the limited screen space to accommodate it in the current Home page.

5.3 Lessons learned

This section takes a look at all the steps and missteps taken throughout the life cycle of this project. It has been a long journey and reflecting upon what went right and what went wrong may help another software engineer who chooses to venture into the wide world of the Web.

5.3.1. What went wrong

- *Lesson 1. "Normalize until it works; denormalize until it doesn't."*

  We waited too long before normalizing the database. This should have been done during the planning stage, before any actual coding began. It set us back time wise and made the architectural design an arduous task.

- *Lesson 2. Good codes come from good tools.*

  We used Visual Studio Ultimate very late in the life cycle of the project. As such, we did not use its code analyzer and architectural design tools until late in the development phase. Still, the tools rescued us many times from the doldrums of code purgatory.

- *Lesson 3. Understand the domain.*

  About 70% of time was spent trying to understand the world of watershed modeling. Countless times we thought we finally nailed what QuickLET was meant to do, only to realize we were on the wrong path. There are no fast answers to this predicament; just a lot more time needed to research the domain, a luxury that we did not have.

5.3.2 What went right

- *The right IDE and fast machines.*

  Having a robust development environment and fast machines were by far the most important elements that made developing QuickLET an interesting and enjoyable journey.

- *The right project.*

  Developing a work-sponsored project provided the best of both worlds, since we were able to work on the project during work hours as well as at home.

- *The right programming language.*

  C# continued to prove itself capable with each project we undertake, and QuickLET was no exception.

- *The right support group.*

  The Microsoft Developer Network forums for C#, ASP.NET and SQL Server had been outstanding in their responses to tough questions. Most of all, there was always someone responding directly to questions within 24 hours, often, even less than that.

Bibliography

[1] P. Quay and W. Frangos, "Model Limitations," 23 February 2010. [Online]. Available: http://serc.carleton.edu/quantskills/methods/models/limits.html. [Accessed 13 April 2015].

[2] "Watersheds," 6 March 2012. [Online]. Available: http://water.epa.gov/type/watersheds/whatis.cfm. [Accessed 13 April 2015].

[3] "Modeling," 23 September 2013. [Online]. Available: http://www.usawaterquality.org/themes/watershed/research/modeling.html. [Accessed 13 April 2015].

[4] L. Ma, L. R. Ahuja and R. W. Malone, "Systems Modeling for Soil and Water Research and Management: Current Status and Needs for the 21st Century," *Transactions of the ASABE,* vol. 50, no. 5, pp. 1705-1713, August 2007.

[5] "Watershed Modeling Tools," 14 April 2015. [Online]. Available: http://cfpub.epa.gov/watertrain/pdf/modules/WshedModTools.pdf. [Accessed 14 April 2015].

[6] H. T. Do, S.-L. Lo, P.-T. T. L. a. P. Chiueh and W.-T. Shang, "Optimal design of river nutrient monitoring points based on an export coefficient model," *Journal of Hydrology,* vol. 406, pp. 129-135, 2011.

[7] M. White, D. Harmel, H. Yen, J. Arnold, M. Gambone and R. Haney, "Development of Sediment and Nutrient Export Coefficients for US Ecoregions," 2015. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/jawr.12270/abstract. [Accessed 2 April 2015].

[8] M. Liu and J. Lu, "Solution of export coefficients of nitrogen from different land-use patterns based on Bayesian analysis," *Water & Science Technology,* vol. 68, no. 3, pp. 632-640, 2013.

[9] J. Arnold, R. Srnivasan, R. S. Muttiah and J. Willams, "Large Area Hydrologic Model Development and Assessment Part 1: Model Development," *Journal of the American Water Resources Association,* vol. 34, no. 1, pp. 73-89, 1998.

[10] "HUC," 6 March 2014. [Online]. Available: http://water.usgs.gov/GIS/huc.html. [Accessed 8 March 2015].

[11] D. M. Johnson and R. Mueller, "The 2009 Cropland Data Layer," *Photogrammetric Engineering & Remote Sensing,* vol. 76, no. 11, pp. 1200-1201, November 2010.

[12] USDA-NRCS, "Conservation Effects Assessment Project (CEAP)," [Online]. Available: http://www.nrcs.usda.gov/wps/portal/nrcs/main/national/technical/nra/ceap. [Accessed 14 April 2015].

[13] USDA-NRCS, *State Soil Geographic Database (STATSGO) Data User's Guide.,* Washington, DC: US Government Printing Office, 1992.

[14] C. Homer, H. L. Yang, W. B. and M. Coan, "Development of a 2001 National Landcover Database for the United States," *Photogrammetric Engineering and Remote Sensing,* vol. 70, no. 4, pp. 829-840, 2004.

[15] "Hydrologic landscape regions of the United States," 31 January 2015. [Online]. Available: http://water.usgs.gov/lookup/getspatial?hlrus. [Accessed 14 April 2015].

[16] Miller, Rowan, "MSDN," 2015. [Online]. Available: https://msdn.microsoft.com/en-us/data/jj590134. [Accessed 20 March 2015].

[17] "ProjectSeven," 2015. [Online]. Available: http://projectseven.com/. [Accessed 20 March 2015].

[18] "ArcGIS Online," ESRI, [Online]. Available: http://www.esri.com/software/arcgis/arcgisonline/. [Accessed 20 March 2015].

[19] "Google Analytics," 2015. [Online]. Available: https://support.google.com/analytics/answer/1257084?hl=en. [Accessed 23 March 2015].

[20] "Box plot," 6 April 2015. [Online]. Available: http://en.wikipedia.org/wiki/Box_plot. [Accessed 15 April 2015].

[21] R. Grady, "Successfully Applying Software Metrics," *Computer,* vol. 27, no. 19, pp. 18-25, 1994.

[22] C. Don, D. Ash, B. Lowther and P. Oman, "Using Metrics to Evaluate Software System Maintainability," *Computer,* vol. 27, no. 8, pp. 44-49, 1994.

[23] S. U. Farooq, S. Quadri and N. Ahmad, "Software Measurements and Metrics: Role in Effective Software Testing," *Internation Journal of Engineering and Science and Technology,* vol. 3, no. 1, pp. 671-680, 2011.

[24] A. Abran, Software Metrics and Software Metrology, California: Wiley, 2010.

[25] A. Jana, "Deploying ASP.NET Websites on IIS 7.0," 5 September 2008. [Online]. Available: http://www.codeproject.com/Articles/28693/Deploying-ASP-NET-Websites-on-IIS. [Accessed 2015 31 March].

[26] "DSS," 14 September 2014. [Online]. Available: http://cida.usgs.gov/sparrow/dss.jsp. [Accessed 8 March 2015].

[27] "MWM," 21 June 2013. [Online]. Available: http://watersgeo.epa.gov/mwm/. [Accessed 8 March 2015].

[28] "LDC," 2014. [Online]. Available: https://engineering.purdue.edu/~ldc/. [Accessed 8 March 2015].

[29] "SNAP," 2015. [Online]. Available: http://research.brc.tamus.edu/snap/About. [Accessed 8 March 2015].

[30] "SWIFT," 2015. [Online]. Available: http://swift.brc.tamus.edu/Default. [Accessed 8 March 2015].

Vita

Marilyn Aldover Gambone attended the Hawaii Pacific University, where she obtained her degree of Bachelor of Science in Computer Science and graduated Cum Laude in August 2005. She is currently employed at Texas A&M AgriLife Research as GIS specialist and application developer.

Email address: mgambone@live.com

This report was typed by the author.