Title: Design / Installation Information for Basis of Estimate Database Project	**Version:** #2

Basis of Estimate Database Project Design Document

Team Snakes

Nicholas Bruno

Travis Buff

Peter Cipolone

Dennis Klauder

Phong Tran

Adam Tucker

Title: Design / Installation Information for Basis of Estimate Database Project	**Version:** #2

Contents

Title: Design / Installation Information for Basis of Estimate Database Project	**Version:** #2

1 Purpose

This document presents the Solution Development Lifecycle (SDLC) Architecture/Design Information for the Basis of Estimate Database Project. This is so data from current and previous programs completed by our clients can be stored, accessed and be turned into estimates about future projects. Also, this document has been designed in such a way so that anyone who is new to the project can use this as a guide to understand the database and its interfaces.

2 Scope

For this project, our team is working to create a database that can hold information from multiple pieces of software so that the requirements will fall into a specific scope. First off, the database must be able to take in information from previous projects that MSE has finished, and the information in these projects will be detailed towards values in the database like hours spent, lines of code, program name, product name, build date, end date, and Computer Program Change Requests (CPCR) just to name a few. This information is required to be entered by a CSV file or individually if desired. The database is also required to keep track of this data by relationships between Program, Product and Work Breakdown Structure (WBS) id's. This project will also require an interface so the user can easily input the CSV files into the database. The interface will be designed as a website with a login page, plus there will be restrictive access for non-admin users along with SQL injection prevention measures.

2.1 Exclusions, Assumptions, and Limitations

2.1.1 Exclusions

Now in the development phase, the software requirements for the output of the database have been excluded from the design. The reason for this exclusion is because the requirements of the output are based upon the design and functionality of the database and are currently subject to change under the direction of MSE.

2.1.2 Assumptions

In our design of the database we are assuming the front-end user interface will be a website and not local software. It is also being assumed that, at any point in time, changes can and will be made at the order of Mission Solutions Engineering. It can be assumed that nothing is permanent until the project is completed and turned into

MSE. Lastly, we are assuming a typical user will be an employee who cannot access database or front end code. They can only access the database by functionality provided through the front end.

2.1.3 Limitations

Currently the database will only accept inputs through single insertion via the user interface or a CSV document for one table in the database. What this means is that user interface cannot accept a document that has entities (values linked to a table in the database) that belong to multiple databases. Also, there is not a function that allows a user to edit more than one database entity (table) at a time. Basically, users can only do one piece of data at a time. Some data, once entered, is immutable except by someone who has direct access to the database code base, for example primary keys. Another limitation in regards to insertion is that data can only be inserted to one table at a time. There is currently not any functionality allowing a user to insert multiple data items into multiple tables at once. Lastly, there is currently no functionality to delete information from the database.

In regards to security, this project is also limited in terms of overall protection because there are few security measures implemented. The best current protection of this database is through isolation; This database is locally kept and is currently not connected to any other program or system. The result is that there is little to no interaction with outside dangers. The client works with a great deal of proprietary information so security for this project is difficult to predict. Basic security will be implemented, but specific implementation will be done by the client.

3 Solution Design Overview

3.1 System Architecture

The solution to the problem is to build a relational database using MySQL and a front-end website using PHP with Codeigniter as the framework. We choose Codeigniter framework because it follows strictly the Model View Control (MVC) pattern, which makes the front-end easier to develop and maintain. Using MySQL to program the database is advantageous because the data that will be stored in the database is relational in nature.

3.1.1 Logical View

The database is broken up into three layers of architecture that encompass the program. Security is involved in all aspects of the database from a login screen in the user interface to preventing SQL injection in the database. An admin user is directly involved with the first two layers. The business layers have four programs that propagate the information that will be input into the database. This layer is included in the logical view because of the influence it has on the information in the database. Once an administrator has the information necessary from the business layer they can access the database via the user interface that currently has four sections.

These four sections include but are not limited to signal data insertion per table, a way to search each individual table, edit the information that was searched for and upload CSV files that would put multiple individual data into the database. The user interface is password protected.

The last layer is the database layer where the data entities are stored. These entities are stored in tables that are related to each other. These tables make up the entity relationships that define how each piece of information is related to each other.

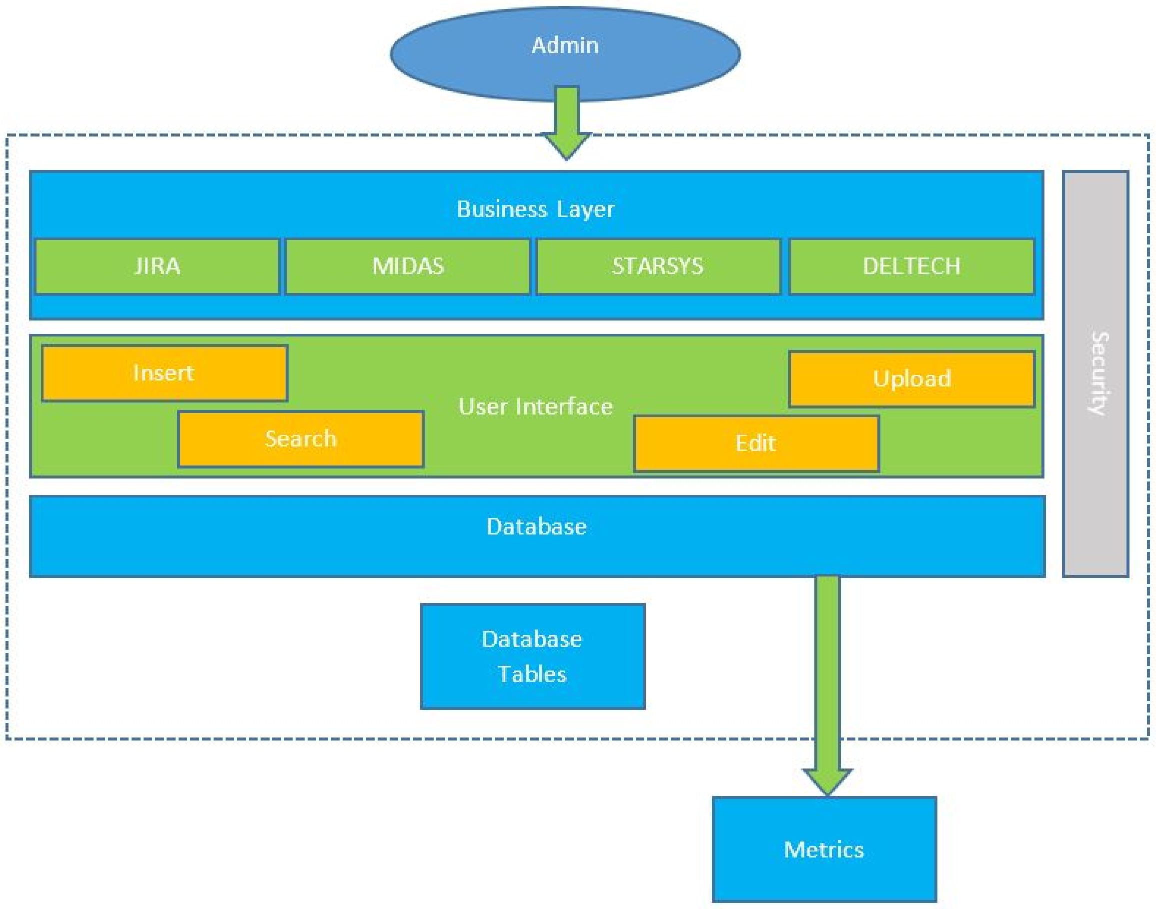

Figure 1 - Logical View

3.1.2 Development View

The project was developed in such a way so that any user is able to search, add, and edit data in the database, but cannot delete data or change the relationship tables in the database. The design accomplishes this by making a user interface that accesses information directly in the database. The user uses the four functions in the user interface to manipulate the data, but cannot change the relationship (you will see later) in the database.

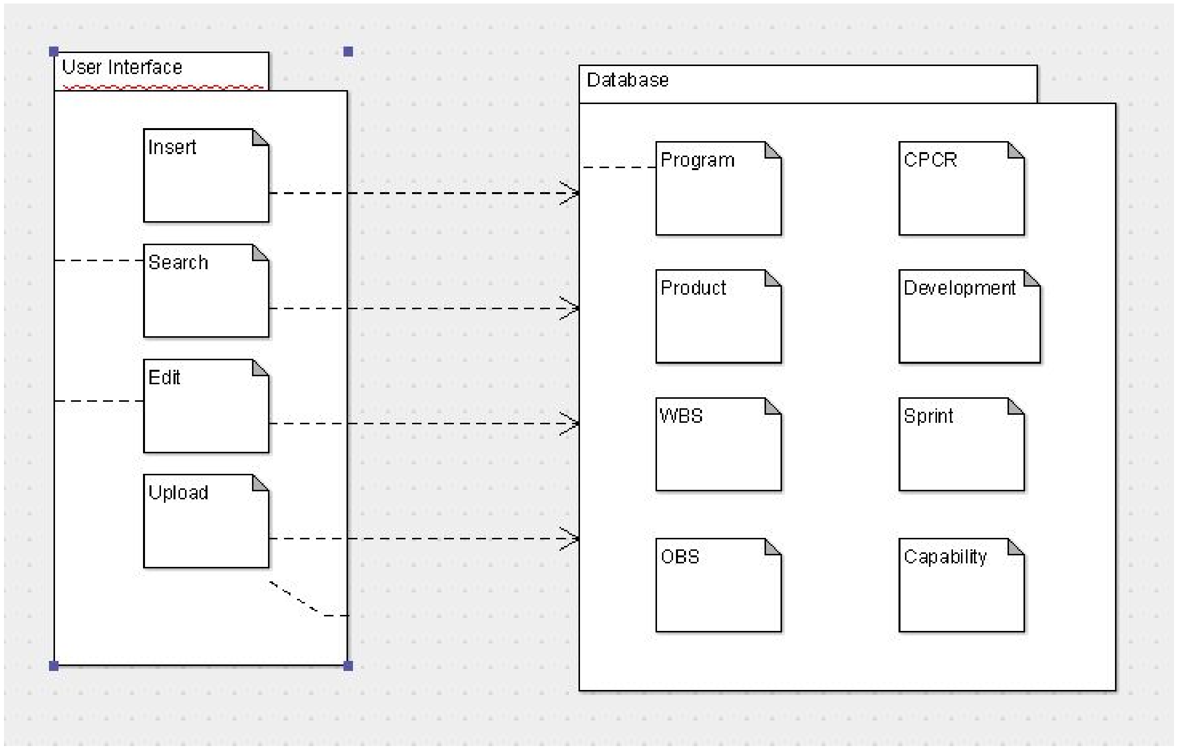

Figure 2 - Development View

3.1.3 Process View

In regards to the user interfaces, three of the four functions behave in a similar manner. The insert, search and edit functions of the user interface all have a pop-up window when the link is clicked. The window contains the individual entities tied to that table. The difference between the insert/search/edit is that in search you are allowed to search the database using any type of data that belongs to the specific table you are searching. Insert does not have any auto incremented fields because those are automatically added to the table when another piece of data is inserted. Edit only allows the user to edit fields that are not primary keys.

Upload is a bit different than the other functions because it involves importing a CSV file. The data in the CSV file does not require a header in the file, though it is allowed.

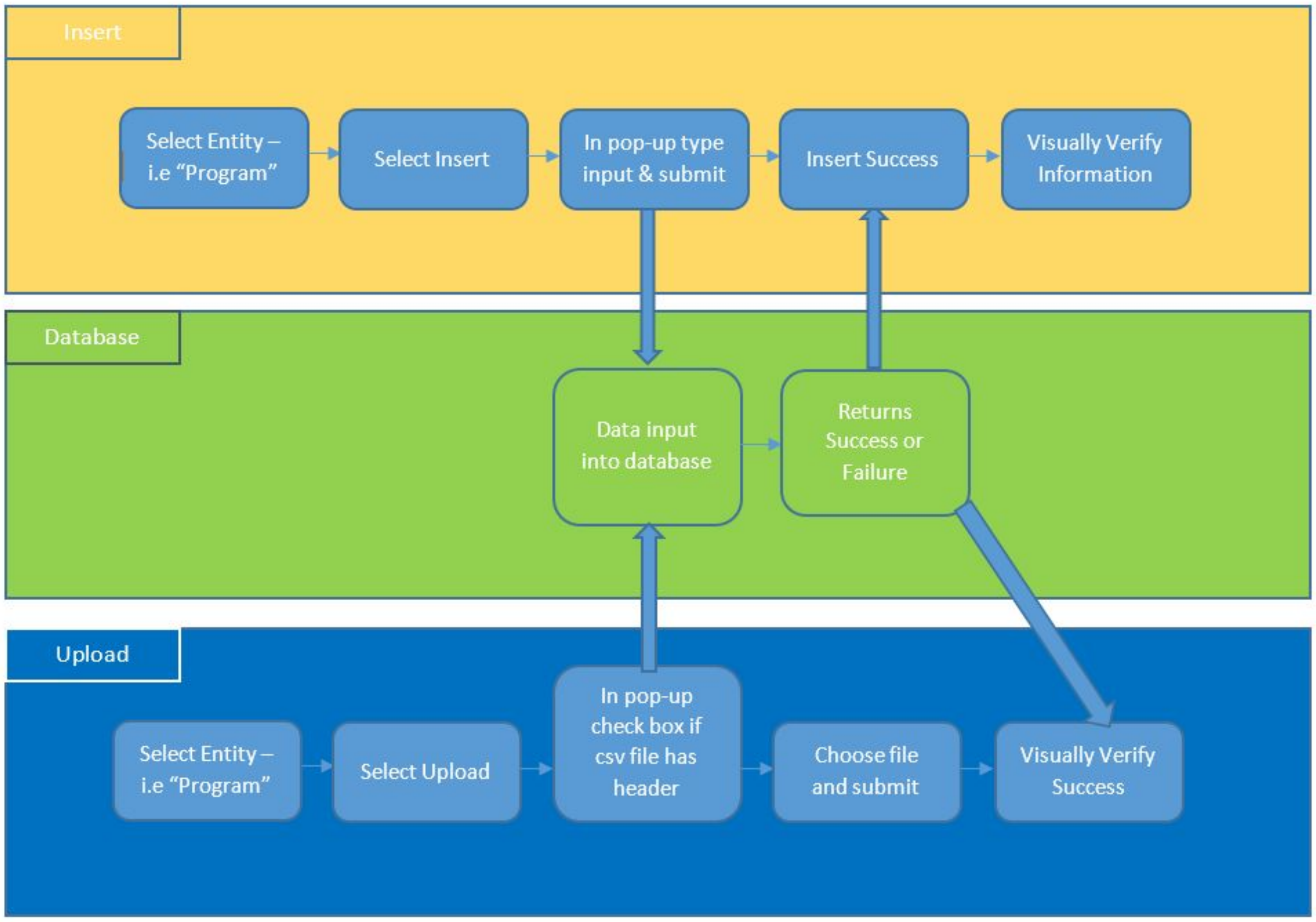
Insert
Select Entity – i.e "Program"
Select Insert
In pop-up type input & submit
Insert Success
Visually Verify Information
Database
Data input into database
Returns Success or Failure
Upload
Select Entity – i.e "Program"
Select Upload
In pop-up check box if csv file has header
Choose file and submit
Visually Verify Success

Figure 3 - Process View

4 Technical Architecture

4.1 Hardware Inventory, Specifications and Locations

4.1.1 Servers

As it stands our development team is using the Elvis server provided by Rowan to host both the database and website. The specs of the server are as follows:

Ram: 16 gigabytes

CPU: eight Intel xeon(R) E5-2680 v3 @ 2.50 ghz

As based off the knowledge of the server in use, any other system that has similar specs as the Elvis server that we are using should be able to integrate the website and database.

4.2 Additional Information

There are four programs that are used frequently by the client at their office. These programs are depicted in the logical view and will be implemented in the future in some manner in the database.

5 Configuration Specification

5.1 Operating system

Currently the application server is using Fedora release 24 with regional setting is en_US, time zone is EST.

5.2 PHP/ Codeigniter configuration

```
Configuration for Codeigniter is located in application/config.php.
        $config['base_url'] = <website address>;
        $config['encryption_key'] = <encryption key>;
Database credential for Codeigniter is located in application/database.php.
        $db['default'] = array(
                'dsn'      => '',
                'hostname' => '',
                'username' => '',
                'password' => '',
                'database' => '',
                'dbdriver' => 'mysqli',
                'dbprefix' => '',
                'pconnect' => FALSE,
                'db_debug' => (ENVIRONMENT !== 'production'),
                'cache_on' => FALSE,
                'cachedir' => '',
                'char_set' => 'utf8',
                'dbcollat' => 'utf8_general_ci',
                'swap_pre' => '',
                'encrypt' => FALSE,
                'compress' => FALSE,
                'stricton' => FALSE,
                'failover' => array(),
                'save_queries' => TRUE);
```

5.3 Configuration Specification Limitations

The remaining configuration is currently unknown because the client works with proprietary information at a secured site. Any configuration needed is difficult to specify because of this fact.

6 Solution Design Specification

6.1 Software Description

6.1.1 Database

The database is needed to be a repository for multiple products and programs whose data is related. Data from each will be entered into the database and stored based on their relationships. The current structure of the database and the way it has been designed is based from meetings with the customer and their feedback on previous designs. The information that is stored in the database has also been set up in the same manner.

6.1.2 Database Interface - Website

The website is the user interface which interacts with the database. Relational tables in the database are modeled in the website under different tabs. Each tab has functions that allow an admin user to input data into the database, search the database, edit data and upload CSV files.

6.2 Coding Standards

6.2.1 MySQL

Currently, the only coding standard is the database must be written in MySQL. The current version is mysql Ver 14.14 Distrib 5.7.15, for Linux (x86_64) using EditLine wrapper.

6.3 Solution Data, Information View, and Data Requirements

6.3.1 Database and collections of data

Data is collected into the base and stored in multiple tables. Each table has a relation to another table as shown in the Entity Relationship diagram below (Figure 4). Each table has the data which describes its name, for example the Program Table is named because it contains a program_id, code and name. The data in each table consists of attributes that are important to describe a program. The purpose of other tables is to connect tables which have a common relationship. For example, in Figure 4 OBS is a relational table which connects Program, Project and WBS by their separate id's.

6.3.2 Data Types

Data types are based logically depending if they need to contain qualitative or quantitative data. Integers are the most common type used for quantitative data. Varchar is used for any type of qualitative data needed to be stored in the database.

6.4 Module Description

6.4.1 Database Modules

The database tables are the modules in the database. The entity relationship diagram (Figure 4) shows how the tables are separated, but also how they are connected. There is a hierarchical overhead of 3 tables Program, Product, and WBS respectively.

6.4.1.1 Program - This contains the program information needed to identify a unique program stored in the database.
6.4.1.2 Product - Contains the collection of the different types of products that can be used in a program.
6.4.1.3 WBS - Contains the collection of services that can be associated with a unique product to program cardinality.
6.4.1.4 CPCR - Used to identify a unique CPCR event that occurred on a unique OBS relationship.
6.4.1.5 Development - Contains information used to compute metrics on a unique OBS relationship.
6.4.1.6 Sprint - Used to identify when a CPCR was active during a sprint. Each sprint is representative of two to four-week period.
6.4.1.7 Capability - Used to identify the type of capability that a CPCR can be described as.
6.4.1.8 Relationship Tables **-** Links the entity tables together to create a relationship of a set of entities. Where there is cardinality between two or more tables, it requires the primary keys of the desired tables to be used as foreign keys in the relationship table. For example, the OBS table is the overhead relationship table which links a unique set of program, product, and wbs objects. Through OBS, you can then link any type of development and CPCR information to the resulting cardinality.

6.4.2 Stored Procedures for Reports

The stored procedures located within the database are a batch of SQL statements which allow convenient use of commonly used SQL statements .This was done for the creation of reports made for the website and could be useful for future reports that might be made at a later date.

6.4.3 User Interface Modules

The user interface's four modules are functions that allow a user to talk to the database. The four functions can be broken into two types of functions: singular data queries or batch insertion through CSV upload. Singular data queries include the insert, search and edit functions. These functions allow a user to view or manipulate one entity at a time in a singular table. Batch insertion involves the upload function which allows a user to insert columns of data into the database at once, but only for a singular table.

6.4.3.1 Insert/Search/Edit

Insert will be used to describe the main functionality of each function and how to manipulate individual data in the database. Each function has a link in the user interface that, when clicked, will pull up a pop-up window. The pop-up window is populated depending on the current table selected by the user. For example, if a user was in the product section of the UI, the pop-up would contain the same number of text fields that are in the table of product, which are currently product name, code and id. The information typed into the text fields will be inserted if insert is selected, and will insert data into the database. Search retrieves information from the database and only then allows the user to edit the information if necessary. One important point to make is that a user can only edit once the information needed is found in the database by searching for the data. Once a user has typed the information in the text fields, they will click submit and information from the database is retrieved or manipulated depending on the function. Figure 5 shows the sequence the information takes once submit is clicked. The result is information is retrieved or manipulated and the result is displayed on the screen and can be viewed by the user. An example for inserting would be a user wants to input a new program into the database. The user clicks on insert and waits for the pop-up window. The user types the information required for a product (Figure 4) and clicks submit. The text goes through the program controller then into the model inserting the data into the database. The inserted program id is returned to indicate success and retrieve the information from the database indicating success and the information now in the database.

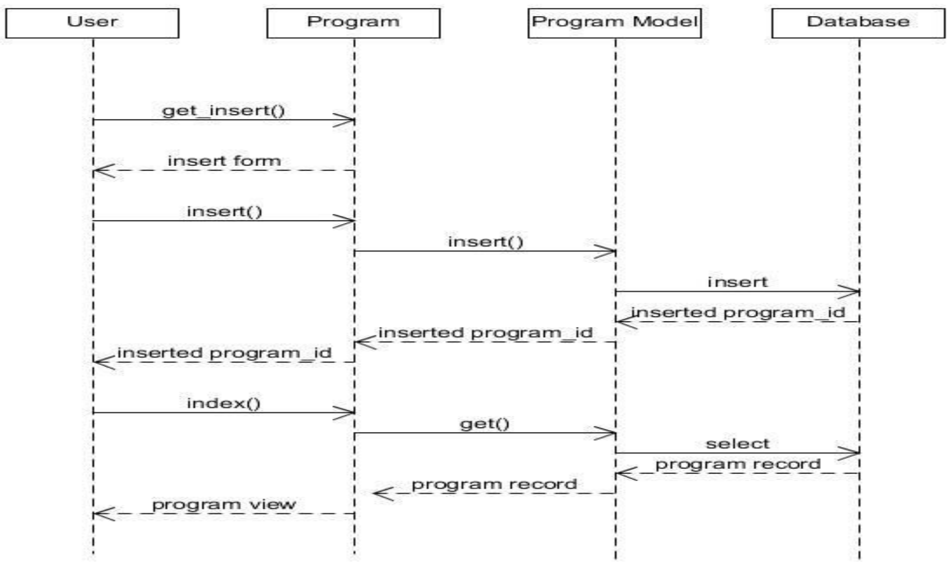

Figure 5 - Program Insert Sequence Diagram

6.4.3.2 Upload

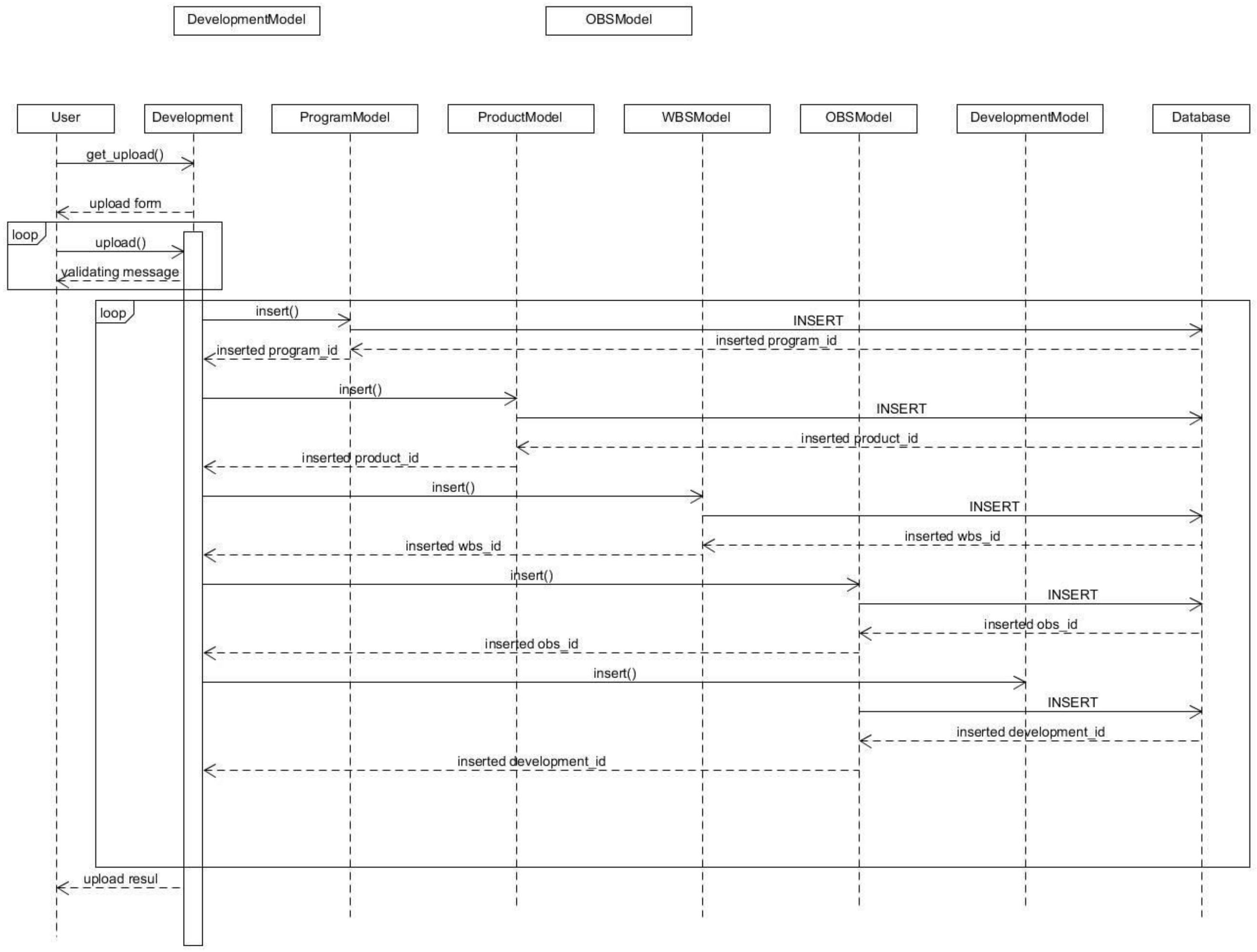

Figure 6 - Development Upload Sequence Diagram

Upload is specific functionality requested by the clients requiring the database to be able to accept CSV files. The actual insertion into the database works like insert and is completed in a loop. The pop-up window comes up when a user clicks on upload and has a few options. One option is to indicate if the CSV contains any column names which are not necessary for insertion. Next the user can choose a file by browsing their local computer directory and locating the CSV file. At that point the upload becomes multiple single insertions. The difficulty is that each table in the ER Diagram (Figure 4) needs separate insertion methods because each table is unique. Development and cpcr uploading is a different than other entities upload because a development or a cpcr record required an existing obs_id in the obs table. So before doing an insert in development and cpcr, we need to run an insert function for program, product, wbs, and obs tables. The insert function will check if there is an existing record containing the same data as the CSV record. If found, it returns the existing record id, else it will insert the record and return the id. An example would be if a user wanted to upload a development CSV file, the controller will go through every record and check for required criteria for the different fields such as numeric, max length, and not null before the data is processed.

6.4.3.2.1 Error Checking in function upload

Error checking is a part of the upload functionality that checks if the csv file being uploaded has the correct data required for its table and the data is in the correct format. Each table described in 6.4.1.1 thru 6.4.1.8 has error checking functionality. The error checking is very simple process by putting the tables names into an array with the data input from the columns in the csv file. The function upload_validation tests each piece of data checking if it matches the rules. For example, a product_name rule could allow a max of only 25 characters and if there is more than 25 the function will return an error for the entire file stating the product_name piece of data caused the error and needs to fixed. Data from the csv file is only stored in the database if all the data passes the validation checks. This helps by ensuring that not only some data was stored into the database which could be difficult to figure out, but also which data was or wasn't stored. If all data passes the validation the upload function then and only then stores the data in the database. The report returns the number of successful inputs into the database showing the numbers of rows stored.

6.4.3.3 Report Generation

Report Generation is the main functionality that was requested by the customer. The report generation is divided into three parts. These parts are the user interface to show the report, the data controller, and the query to pull the data. The user interface contains two main parts: the post filter form and the data table which will show the report after the user submits the filter. The HTML code for the user interface can be found at application/View/report/. This is done to give the result table a professional look.

Each report is controlled by two different methods pertaining to the ReportController. The first is function responsibility, it is used for calling and populating data to the User Interface. This function is triggered when the user clicks on the report href link on the report menu or called by the second function which will be mentioned later in this paragraph. The second function's responsibility is gathering the data from postfilter after the user clicks the submit button. This function will format data in a correct format and call Model function which will run the query to pull data for the report,it will then send the result data to the User Interface by calling the first function with the result as a parameter. In order to maintain readability of the code, we name the second function the same as the first function but add "_get_result" at the end. For example: the first method for Development Cost Report is "development_cost" then the second method name is "development_cost_get_result"

The query could be a procedure or a custom query which is executed using a function of the Codeigniter Model which is located at application/Model. For example, the query of Development Cost Report will be found inside function development_cost_report of application/Model/DevelopmentModel.php.

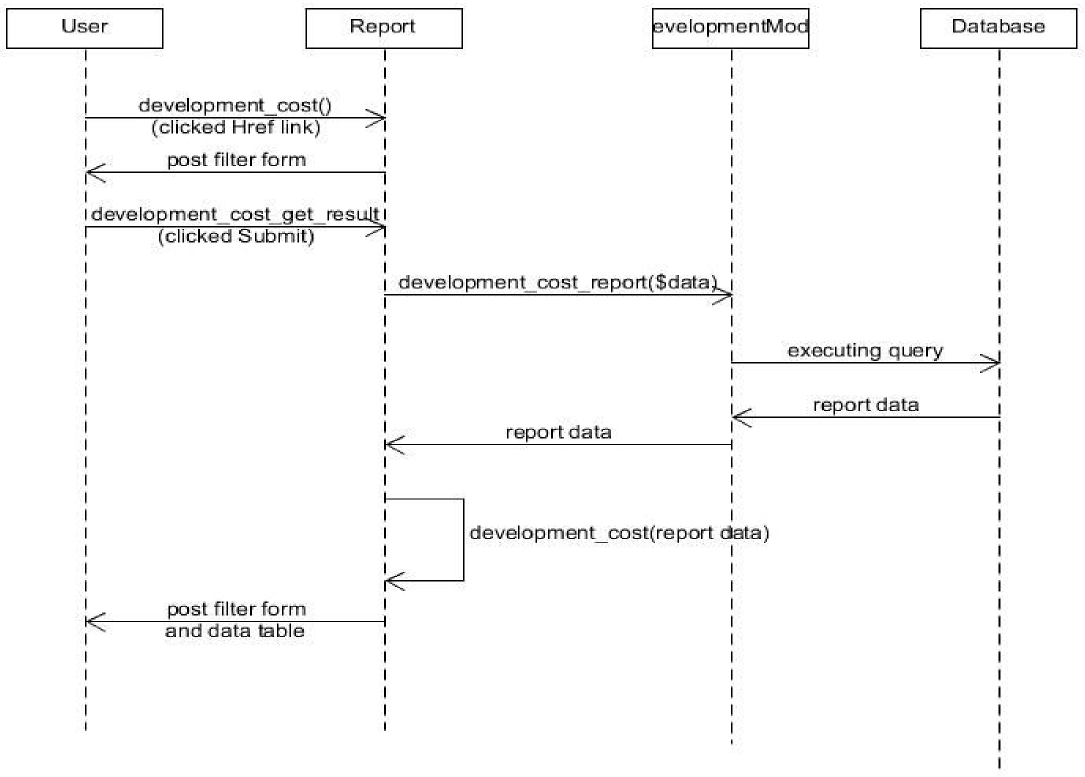

Figure 7 - Development Cost Report Sequence Diagram

7 Roles and Responsibilities

Role	Responsibilities
Maintain Database	Ensure the integrity of the database is optimal as possible
Maintain User Interface	Update the UI as needed when entities are removed or added to the database

8 Supporting References

"There are no supporting references specific to this document"

9 Revision History

Version	Version Date	Revisions
1.0	11/23/2016	Initial Document Release
2.0	12/20/16	Update with new functionality including error checking and report generation