SiteSim V2.0

Client: Dr. Paul Santi

Andrew Hill, Jon Pigg, Paul Susmarski

Summer 2011

Abstract

The BEST SiteSim (Basic Engineering Software Engineering, Site Simulation) is a program designed by John Petrikovitsch in conjunction with Colorado School of Mines Professor Paul Santi. The software is used to provide students with the opportunity to run drilling investigations in both hydrogeologic and geotechnical simulations. The program works by having students determine drilling locations and appropriate tests when doing investigations, and allows them to get cost estimates for these tests. SiteSim is currently out of date, meaning that it is incompatible with newer operating systems and there are bugs within the program. The code is written in an unknown scripting language, and is therefore impossible to update from its current state.

The objective of this project is to recreate the software using the Java programming language. Java ensures that the software will be more portable and compatible with modern hardware, software, and networks. The team has also updated the software to conform to modern day development methodologies while maintaining most of the current functionality that the program offers. The current complex file hierarchy has been replaced with a single modern database. The team recreated all of the algorithms and calculations from SiteSim's current functionality. Work on the project has done such that development can continue after completion of the team's contract.

The team has implemented a functional boring log (a graphical display that has different colors for different soils at different depths when drilling into the earth) that can be accessed through a single site (location where an investigation takes place). The project also includes limited skeleton features and methods (methods and classes that do not perform any function, but have description of their future usage) to easily allow future programmers to complete what needs to be added. A database has also been created that contains information about the sites and allows a user to have their own saved data; however implementation of the database still needs to be completed. The overall project was much larger than initially thought, and thus the scope decreased to having the database and functional boring log to maximize productivity.

Table of Contents

1.0 Introduction

Dr. Paul Santi (the client) teaches a class called Site Investigation for the Geology Department at the Colorado School of Mines. For this class, he utilizes a program called BEST SiteSim (Basic Engineering Software for Teaching, Site Simulation) to introduce students to drilling placement and budgeting for hydrogeological and geotechnical investigations. The software is only available on some of the class computer and contains multiple bugs. Some of the known program problems are as follows:

- Minimal memory leaks
- Requires 32-bit operating system
- Problems with printing, involving the need to save the program state and then load the state in a file and print
- Problems with saving
- The program freezes up at various points of interaction

As it stands, the previous program is no longer compatible with standard Windows operating programs and will fall out of use unless it is updated and reprogrammed.
Our group, Team Recharge, was tasked with developing a new, more stable version of the SiteSim software over a six week period. This new version was created using Java, so that it is more universally compatible with modern computing systems. As computers have greatly advanced since the release of the original program, there is the potential to dramatically improve the educational aspects of the program, as well as its overall functionality, reliability, portability, and maintainability. By upgrading this software we aimed to ease the use of the program, as well as eliminate the hassle of finding compatible computers to run it on. After the addition of these features we have made, it easier for future work to progress.

2.0 Requirements

2.1 Functional Requirements

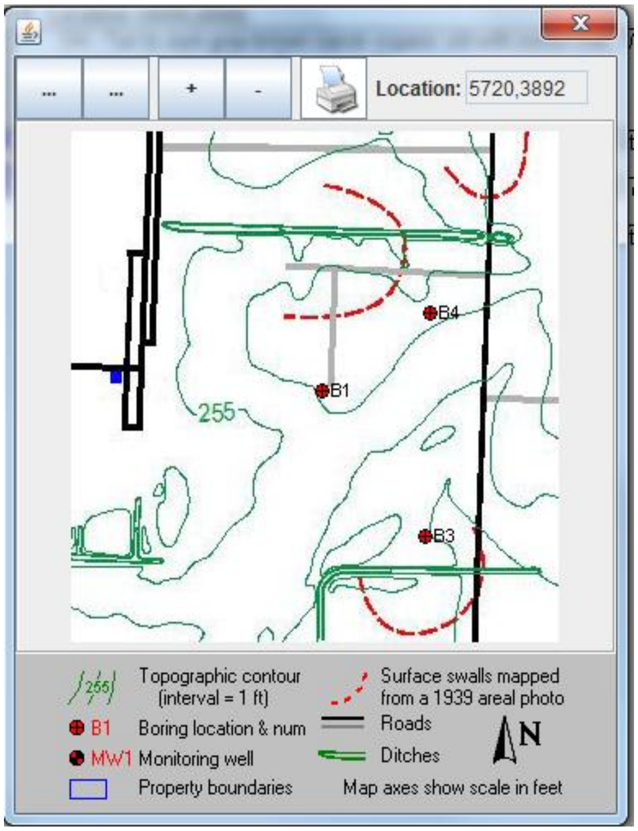
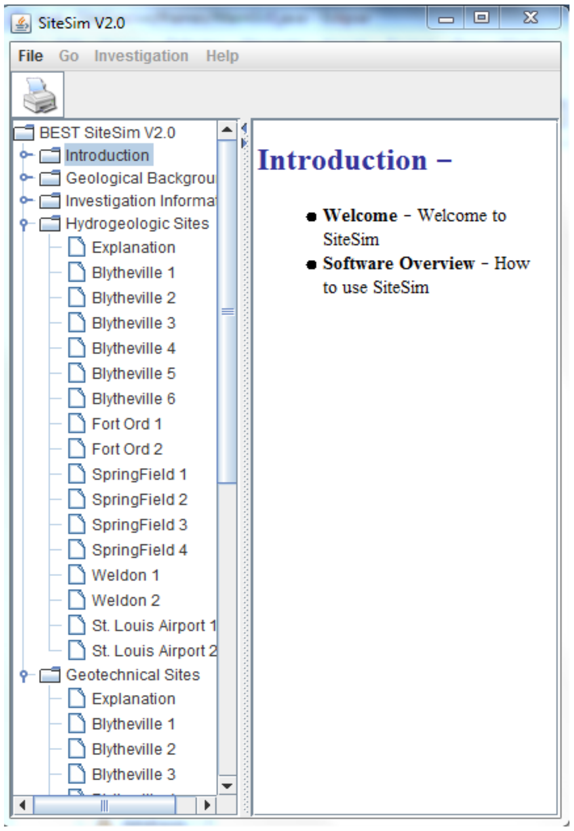


Figure 1. A screenshot of the current map window that is implemented. It displays the borings that have occurred, updates location on mouse over and has placeholder buttons.


SiteSim V2.0

File Go Investigation Help

- BEST SiteSim V2.0
  - Introduction
  - Geological Backgrou
  - Investigation Informa
  - Hydrogeologic Sites
    - Explanation
    - Blytheville 1
    - Blytheville 2
    - Blytheville 3
    - Blytheville 4
    - Blytheville 5
    - Blytheville 6
    - Fort Ord 1
    - Fort Ord 2
    - SpringField 1
    - SpringField 2
    - SpringField 3
    - SpringField 4
    - Weldon 1
    - Weldon 2
    - St. Louis Airport 1
    - St. Louis Airport 2
  - Geotechnical Sites
    - Explanation
    - Blytheville 1
    - Blytheville 2
    - Blytheville 3

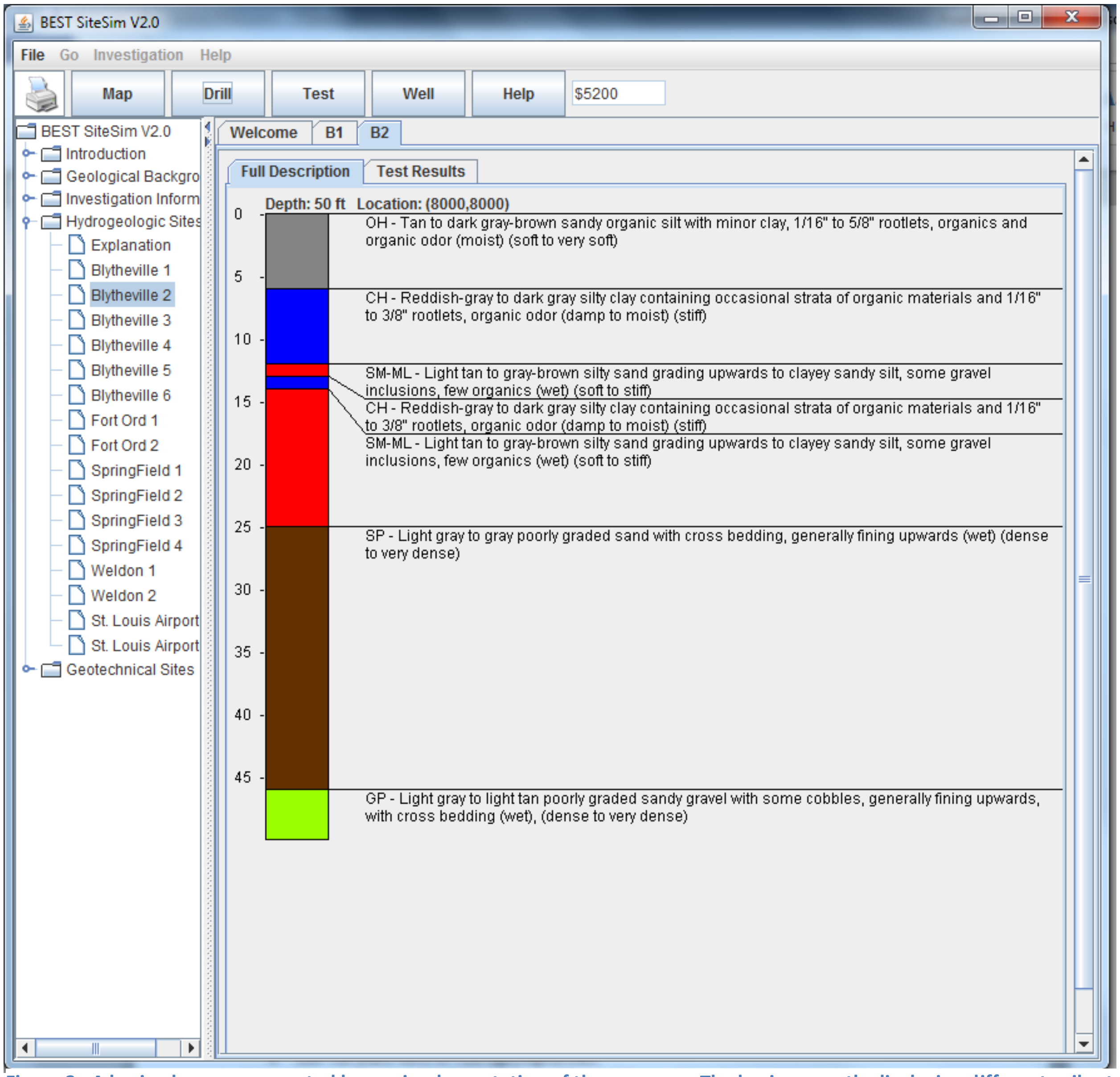
Introduction –

- **Welcome** – Welcome to SiteSim
- **Software Overview** – How to use SiteSim

Figure 2. The list of possible selections within the program. The site selections are fully expanded showing all the possible sites.

- The program has the following features:
  - Includes a brief tutorial that explains how to use it.
  - The frame is divided into 3 sections: a button panel, a list of all nodes selectable, and a panel that displays information about the current node.
  - Displays a 2-Dimensional representation of the site in a map window, see Figure 1.
  - Displays a list of pre-defined sites. See Figure 2 for the representation of this list in the current program.
- Site type may be geotechnical or hydrogeological.
  - Hydrogeological sites are a subset of geotechnical sites, including all the information of a geotechnical site with the addition of being able to add wells to a given boring log.
- Each individual site begins with a budget for the investigation, called the balance.
- The user can perform a drilling on either type of site.
  - User can select location for drilling from a 2D map of the site. See Figure 1 for how this is implemented in the current program.
  - User can specify the target depth for the drilling. Some sites have a revolution of a foot while others have a resolution of five feet.

- Program provides an estimated cost for each drilling.
- Selecting various sites on the map to drill should yield results, whereas selecting the same location twice (e.g., by two different students) should yield the same result. The program should displays soil properties based on the soil types and their definitions.
- Drilling results are shown as a color-coded boring log, which allows the user to see different types of soils and details about the soils. See Figure 3 for how the current program displays this information.



Depth: 50 ft Location: (8000,8000)

OH - Tan to dark gray-brown sandy organic silt with minor clay, 1/16" to 5/8" rootlets, organics and organic odor (moist) (soft to very soft)

CH - Reddish-gray to dark gray silty clay containing occasional strata of organic materials and 1/16" to 3/8" rootlets, organic odor (damp to moist) (stiff)

SM-ML - Light tan to gray-brown silty sand grading upwards to clayey sandy silt, some gravel inclusions, few organics (wet) (soft to stiff)

CH - Reddish-gray to dark gray silty clay containing occasional strata of organic materials and 1/16" to 3/8" rootlets, organic odor (damp to moist) (stiff)

SM-ML - Light tan to gray-brown silty sand grading upwards to clayey sandy silt, some gravel inclusions, few organics (wet) (soft to stiff)

SP - Light gray to gray poorly graded sand with cross bedding, generally fining upwards (wet) (dense to very dense)

GP - Light gray to light tan poorly graded sandy gravel with some cobbles, generally fining upwards, with cross bedding (wet), (dense to very dense)

Figure 3. A boring log as represented by our implementation of the program. The log is currently displaying different soils at different depths with their appropriate descriptions.

- User can print relevant information on the panel that's currently being viewed.
  - For sites that are loaded, users will be able to view previous borings done on the site and view the results.

2.2 Non-functional Requirements

- Development Language
  - Java was used due to the portability it offers.
  - Eclipse was used because it is a best in class Integrated Development Environment (IDE) for Java.
  - Java Data Base Connectivity (JDBC) was used to interface with database.
- Database
  - SQLite is a Structured Query Language (SQL) that was used because it is a self-contained, serveless, zero-configuration, transactional database engine. This allowed the storing of files in a much more organized manner.
  - User was investigated (size and concurrent connections), however was not a concern due to local drive accessing.

3.0 Design

Our system is an application that resides on a single computer. Upon opening the application a graphical interface appears allowing the user access to three different branches of a tree; reference Figure 2 for the layout. One branch gives the user information on the program itself and the different drilling sites. Another branch is used for geotechnical work, allowing the user to simulate drillings and to perform contaminant/soil tests on a given site at specified locations. The final branch is for hydrogeological work, allowing the user to simulate drillings, contaminant/soil tests, and wells to be placed at a given boring. Drilling currently functions for Blytheville sites (1 through 6) in both hydrogeological and geotechnical investigations, displaying relevant information about the soils that are found at specific depths and locations, reference Figure 1.

We have added several skeleton features during our project that represent functionality beyond the scope of our project, but are important for the overall usability of the program. We also adhered to agile practices which allow code to be easily expanded upon with minimal changes to the existing system.

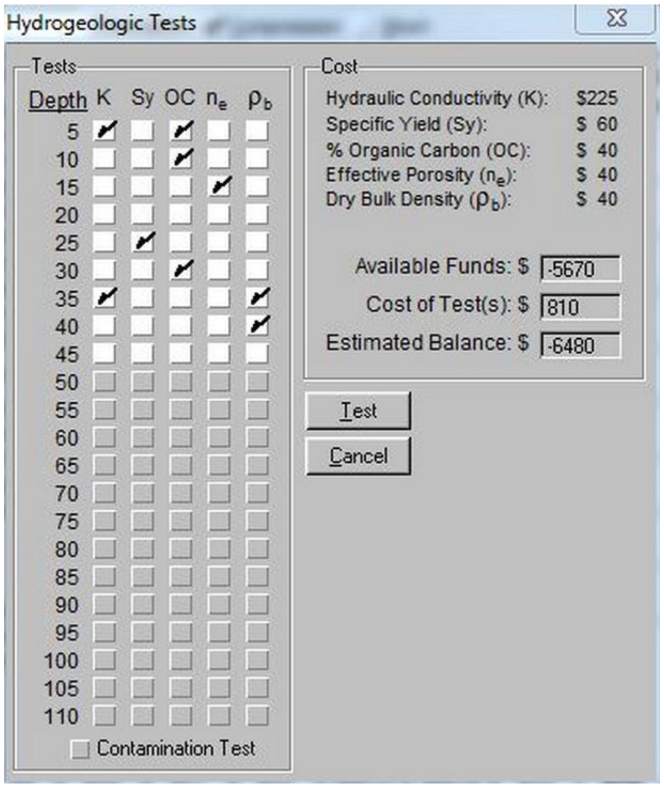
3.1 System Testing

Our application needs testing of the components and calculations that are performed. These have been defined as JUnit tests and acceptance tests, which are found in the following lists.

3.1.1 JUnit Tests for Calculations

- **testRandomNumberGrabber()** – This test runs through a for loop to assure that we are only grabbing random numbers between -1 and 1. The function that is called returns a double value that will be used in calculations. Math.Random was used to generate numbers as it provides a simple and easy way to acquire random numbers.
- **testPerformGeologicalCalculation()** – This test currently assures that our program successfully performs the calculation that is used throughout the program. The calculation $(A + (B*Z)) + ((C + (D*Z))*R)$ requires five parameters. A, B, C, and D are all acquired from the definition file in the given order for a specific property (see Appendix B), and Z represents the current depth. This

calculation is performed at every depth that the user specifies when selecting tests. See Figure 4 for how these are selected.



Depth	K	Sy	OC	n_e	ρ_b
5	✓		✓		
10			✓		
15				✓	
20					
25		✓			
30			✓		
35	✓				✓
40					✓
45					
50					
55					
60					
65					
70					
75					
80					
85					
90					
95					
100					
105					
110					

Contamination Test

Hydraulic Conductivity (K):	$225
Specific Yield (Sy):	$ 60
% Organic Carbon (OC):	$ 40
Effective Porosity (n_e):	$ 40
Dry Bulk Density (ρ_b):	$ 40

Available Funds: $ -5670

Cost of Test(s): $ 810

Estimated Balance: $ -6480

Figure 4. The test window popup that appears within a site. This is from the original program and has full functionality.

- **testUpdateBalance()** – This test sets the balance to a given value then performs a deduction, thus confirming that the implementation of the BalanceCalculator (tracks the balance for any given site) is correct. There are only deductions from the balance in the SiteSim program.

3.1.2 Acceptance Tests

Since a lot of our application is graphical interfacing, it was important to layout simple acceptance tests that we could adhere to as we programmed. The list of acceptance tests that we successfully pass:

- Can the user open the map window?
- Can the user print a boring log?
- Can the user open the drilling window?
- Can the user manually modify the coordinates and depth in the drilling window?
- Does the cost estimate change with varying depths in the drilling window?
- Can the user open the test window?
- Can the user open the well window?
  - Can the user modify the top and bottom depths of the well to be placed?
  - Does the cost estimate update with the change in depths?
- Does selecting a drilling location on the map provide the correct coordinates in the drilling dialogue?
- Is the database properly storing all data?
- Are the boring logs being properly generated?

3.1.3 JUnit Tests for Database and JDBC

Unit tests were created to test both the database integrity and the Database Access Objects (DAOs). The DAOs are classes that utilize JDBC in order to communicate with the database from our Java program. Unit tests were written for most of the DAOs that ultimately tested the appropriate create, read, update, and delete commands on each table.

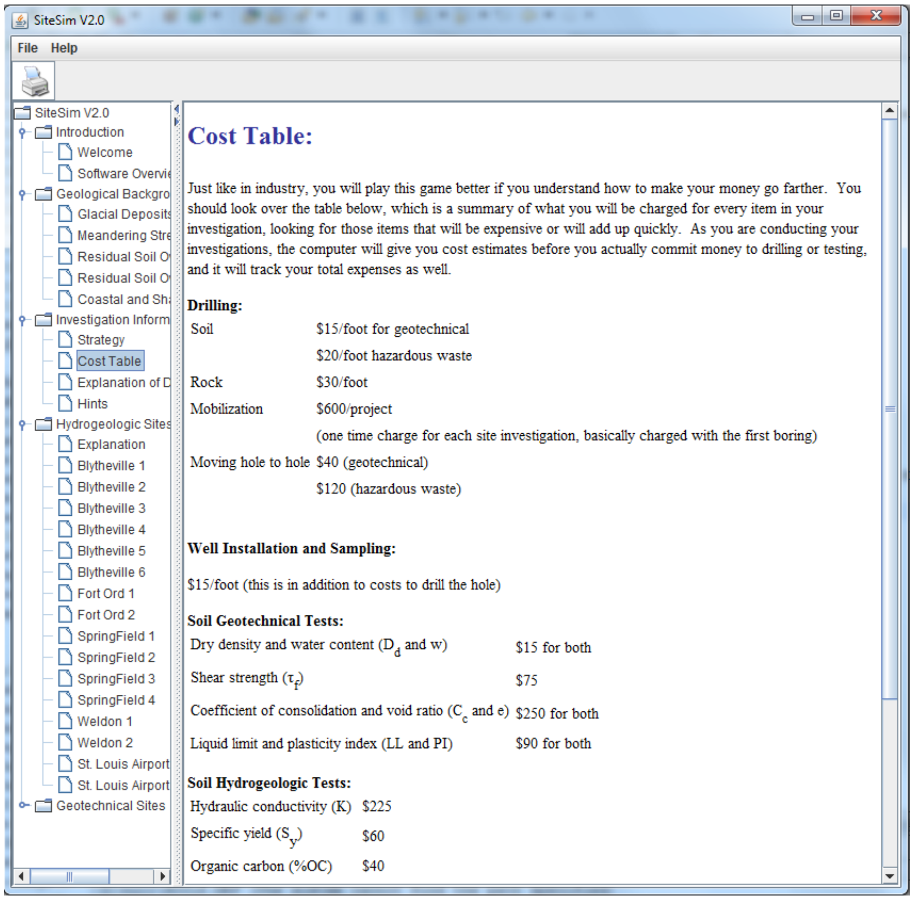
3.2 Use Case Scenarios

Appendixes A-1 through A-5 represent the flow of execution that the program should be able to undertake when complete. The entire program was laid out using these “work flow diagrams” because the previous program has all the functionality that the client wanted, so it was important to capture all possible choices a user makes while using the old software. While the diagrams show the entire workflow of the program, our scope was limited to creating the ability to drill within the program and displaying a valid boring log.

3.3 The Graphical User Interface (GUI)

The basis of the program is to allow students to do a simulation of a site investigation, so it was important to get a graphical interface working that can be extended to include all functionality. The client was very pleased as to how the previous program worked, thus much of the previous design and layout was maintained. The list of key graphical displays that our implementation of SiteSim supports is:

- Textual Information about the program and any site in the form of a display window on the right side of the split pane. See Figure 5 for how this is displayed in the program.
- A print button that can be clicked to have whatever is currently being displayed in the right side of the split pane printed.
- A map button that can be clicked, creating a map window that allows the user to create borings, see Figure 1 for the map window display.



Cost Table:

Just like in industry, you will play this game better if you understand how to make your money go farther. You should look over the table below, which is a summary of what you will be charged for every item in your investigation, looking for those items that will be expensive or will add up quickly. As you are conducting your investigations, the computer will give you cost estimates before you actually commit money to drilling or testing, and it will track your total expenses as well.

Drilling:

Soil	$15/foot for geotechnical
	$20/foot hazardous waste
Rock	$30/foot
Mobilization	$600/project
	(one time charge for each site investigation, basically charged with the first boring)
Moving hole to hole	$40 (geotechnical)
	$120 (hazardous waste)

Well Installation and Sampling:

$15/foot (this is in addition to costs to drill the hole)

Soil Geotechnical Tests:

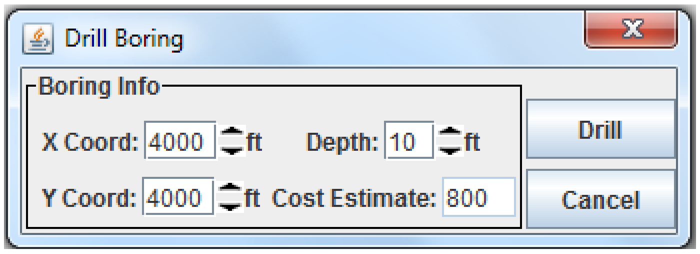
Dry density and water content (D_d and w)	$15 for both
Shear strength (τ_f)	$75
Coefficient of consolidation and void ratio (C_c and e)	$250 for both
Liquid limit and plasticity index (LL and PI)	$90 for both

Soil Hydrogeologic Tests:

Hydraulic conductivity (K)	$225
Specific yield (S_y)	$60
Organic carbon (%OC)	$40

Figure 5. A view of the current program displaying information associated with the node that is currently clicked. In this case, the node is the cost table.

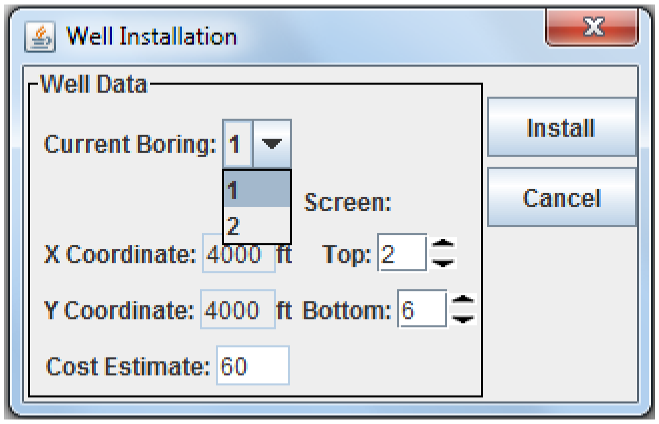
- A drill window that appears with the appropriate fields for drilling. See Figure 6 for how our program displays the drill window. This drill window needed to detect the boundaries of the X and Y position on the map, estimate the cost as the depth changed, and prevent the user from entering depths that were out of bounds.



X Coord: 4000 ft Depth: 10 ft
Y Coord: 4000 ft Cost Estimate: 800

Figure 6. The drill window as it is displayed in our implementation of the SiteSim program.

- After drilling a boring log is displayed with the information on the soils of that site, with a scale to see where (in feet) the soils occur. See Figure 4 for how this is shown in the program.
- A well window button appears and can be clicked in hydrogeological sites that allow the user to see cost estimates, modify the top and bottom depths of the well, and select any boring they want to install a well on in the current site. See Figure 7 for the window that appears for when a well needs to be installed. It has the same type of boundary detection for depths that the drill window has.



Current Boring: 1
Screen:
X Coordinate: 4000 ft Top: 2
Y Coordinate: 4000 ft Bottom: 6
Cost Estimate: 60

Figure 7. The well window that appears for hydrogeological sites upon clicking the well button from the main interface.

- A help window appears upon clicking the help index tab that displays information on how to use the program. The formatting of the text in the window is very similar to how it is displayed in Figure 5, the help information is displayed in a separate window.

All other graphical displays in the program have very minimal functionality. The items that do appear, but are simply place holders are the test window, the test results on the boring log, and the zoom, pan, and cursor buttons on the map window as seen in Figure 1.

3.3 UML

Figure 8 is a UML that was created for our new SiteSim program. While it doesn't include all the classes, it provides a basic representation of the communication that is occurring in the new program. The UML provides insight to how the dependency inversion principle was utilized to keep the program easily extendable.

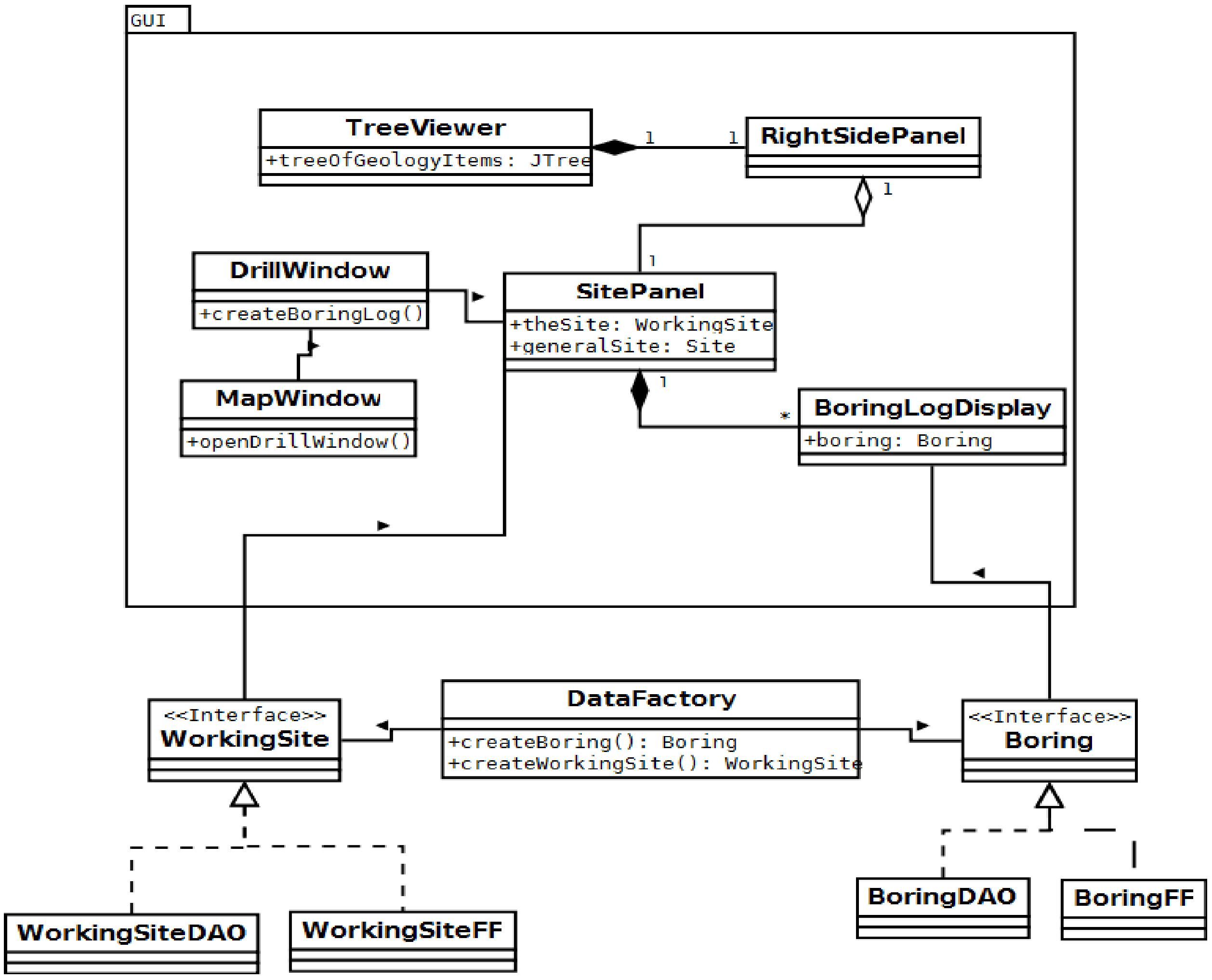


Figure 8. A UML diagram of the communication between the GUI and the data needed to populate the GUI. The UML doesn't include all classes, but represents the way the GUI will in general communicate with the files/database.

The DataFactory was used to keep the GUI not directly dependent on the data classes. The two different data classes are FF and DAO; FF are flat file classes that access the files directly off the hard drive, while the DAO classes access the newly created database. This allows for any implmentation to take place with the program, as it can easily be loaded on to a single computer or (if possible) a webserver that will utilize the database.

3.4 Database Schema

The program utilizes a local SQLite database, although this database is currently not attached to the GUI. In the future it is expected that the program will utilize a MySQL database on the web. A database was chosen to model the information for the following reasons:

- To separate data access logic from application logic
- To meet the requirements of a server based program more readily if necessary
- To store all data in a consistent, widely supported, and up to date format

A database Entity Relationship Diagram (ERD) and schema were created early on in development. These have continued to change as necessary throughout the project up to, and including, the final week. To see the top level ERD, schema, and relevant information, refer to Appendix D-1. The remainder of Appendix D describes each entity in more detail.

4.0 Implementation Details and Results

4.1 Design Decisions

The program was replicated using Java, however minimal changes were made to the design because the client was overall pleased with how the old program functioned and looked. The only graphical change that was proposed was the way boring logs were displayed. The old boring logs (see Appendix C for the old program's display of boring logs) were poorly formatted in relation printing and viewing, resulting in disjointed information, meaning there is no way to connect the tests with what depths they were performed at. We then proposed using tabbed panes, with one pane displaying the full information of the soils, while the other displays the information about the test performed, along with the graph of the boring (with short descriptions). This created a better format for printing and allowed for easier connection between the boring and performed test.

A problem we ran into during the session is that we originally proposed the project be designed using a Java web framework. This involved a week of research and spending time with a proposed framework named JVx. JVx is a Swing based framework that allows implementation of Java source code on a website. However, at the end of the week we discovered that this framework had no documentation for deployment on a web server (at least in English) and this was put aside for future work. Thus the decision was made to change design from a web based program to a local machine program based on only Java and Swing components.

We also made the decision to move from MySQL to SQLite. While both are SQL based, SQLite provided a much more straightforward interface when accessing information stored on a single computer/hard drive. MySQL was originally proposed for the thought of porting the program to a webserver. While this is still the ultimate goal of the project, SQLite was a convenient choice in order to run the program on a local machine.

4.2 Scope

The original scope defined by our team was to have a basically functioning program that had all the features of the original program. However, after the first three weeks of research and design, it became evident that to finish a basic replica of the old program (all buttons working with all tests results and wells being displayed on the boring log), the project would have to extend beyond the six weeks of field session. The extension of the project was already planned; however after only a week of only coding, it became evident that we could only achieve a functioning drill feature that displayed a boring log and a map of the site where the drilling took place. Even this took time and we decreased the scope to only include one functioning site for drilling, information that is displayed about the sites, and how to use the program.

As part of the original scope it was proposed that a functional database, with all of the old files loaded onto it, would exist. However, after a week of working with the database, we realized that we weren't using the most optimal approach and had to rethink how the database would be accessed. In addition, without the source code, we did not know the order that the program was accessing the files. Thus, we had to fashion a structure that was logically utilizing the files. This resulted in a basic database that had the result sets planned out, as well as a way to access the old file structure (where everything resides in folders on the hard drive) allowing for the program to take any direction that is chosen in future work (whether it be simply a local program or a web server based program as preferred by the client).

4.3 Results

As mentioned in the design section, a lot of the program was graphical interaction and displays for the users. The following list is the acceptance tests and their results in the current state of the program:

- *Can the user open the map window?* – The map window currently opens with the map of the site, updates coordinates as the mouse moves over the map, and displays a legend. It also allows the user to click on a location on the map to open the drilling window.
- *Can the user print a boring log?* – Currently, the user can print a boring log, however the formatting of the print still cuts off the edges of the soil descriptions.
- *Can the user open the drilling window?* – The drilling window can be open with all the fields of the previous program being displayed and a drill and cancel button also function.
- *Can the user manually modify the coordinates and depth in the drilling window?* – The coordinates and depth can be modified either by clicking the up or down buttons associated with each of the fields or by typing in their own values.
    - *Does the cost estimate change with varying depths?* – The cost estimate currently changes with any change in depth, whether the depth is typed in or incremented using the buttons.
- *Can the user open the test window?* – The test window can be open with placeholder checkboxes representing tests that can be performed.
- *Can the user open the well window?* – The well window opens with all the previous fields of the SiteSim program, however a installing a well does nothing.
    - *Can the user modify the top and bottom depths of the well to be placed?* – The top and bottom coordinates can be modified much like the drill window fields; it is also detected when the top of the well is deeper than the bottom of the well, displaying an error.
    - *Does the cost estimate update with the change in depths?* – The cost estimate does change with depths, calculated by the difference between the top and the bottom of the well. Very similar to the way the drill window changes cost.
- *Does selecting a drilling location on the map provide the correct coordinates in the drilling dialogue?* – This test passes; the location that is clicked on the map currently displays the coordinates of the clicked location in the drill window.
- *Is the database properly storing all data?* – Currently the database doesn't store all data; however the database can successfully create, read, update and destroy entries. The database also has appropriate tables for all information needed in the program.

- *Are the boring logs being properly generated?* – Boring logs that display soil information at their appropriate depths are currently being generated. However all information related to testing the soils and installing wells still need to be implemented.

5.0 Conclusions

The program, SiteSim V2.0, successfully performs drilling in Blytheville and displays the relevant information on this site given any location and depth within the site bounds. SiteSim V2.0 also displays all the textual information that was available in the previous version of the program, which includes information about the sites, how to use the program, and descriptions of the various geological terms and tests. Very minor bugs remain with the current state of the program, and those that couldn't be solved within the project's time frame have been documented within the code for future work. We've also added skeleton methods and code that allowed parts of the program to function, but don't actually do anything relevant. By including the skeleton code, it makes it easier for future programmers to recognize what needs to be added. The database has also been implemented and can successfully read and write information, however, the program doesn't currently utilize the database in order to display the information for a boring log, as it is there for future extensions. The program currently loads files directly from the hard drive that were used by the previous SiteSim version to display information.

5.1 Lessons Learned

Throughout the project, it became very obvious that using subversion was critical to accomplish work. Often times we each had different schedules and couldn't always meet in person, so it was critical to commit often and commit small amounts of code, which made it easier to deal with collisions and quickly read through what was recently added to the project.

Another lesson learned was the importance of interfaces and their creation early on in the design of the software. Without them, we ended up creating two separate entities, the database and the GUI, that acted independently and had no way to easily connect the GUI to the database once it was complete. Thus, extra time was spent during the final week of the project to get the two entities to work together and easily switch between the files on the hard disk that were used, and the database that was planned to be used.

The importance of communication and understanding how different modules of the system are being implemented is also important. While the code from another person is often there to view, it may not always be clear to how to utilize the code from a different perspective; or the other user may end up using the code improperly.

6.0 Future Work

Work is planned to continue after the completion of the six week field session, where one of the members of the team will continue the project. One of our goals of the project during these six weeks is to lay out what would need to be done for the entire program to function, since we focused on getting the drilling feature working for a single site. It is also important for us to keep the program expandable so that work can continue. This section defines the requirements of the fully implemented program.

6.1 Functional Requirements

The following list contains all the features that should be available upon completion of the program. This allows future programmers to check-off features that need to be added.

- Map Screen includes options to pan, zoom, drill, and print
- User can place wells on hydrogeological sites.
    - User can select location for well from a 2D map of the site.
    - User can specify start depth and end depth.
    - Program will provide an estimated cost for the well.
- User can choose to test any boring for contaminants.
    - User selects the boring for the tests.
    - User specifies tests to be performed. The possible tests are contaminant testing, dry density and water content, shear strength, coefficient of consolidation and void ratio, and liquid limit and plasticity index.
    - Program provides estimated cost for test.
    - Test results will be displayed on the boring log.
    - Program will interpolate contamination result based on nearby contaminants. The interpolation would involve a weighted linear interpolation if between more than two source contaminants.
- Costs for running tests, drillings, or well placements should be deducted from the user's balance. The balance should be prominently displayed. If the user's balance drops below $0, the balance will continue to go into the negatives. Color coding will be used so an instructor walking the room can easily spot negative balances.

6.2 Non-Functional Requirements

The client had expressed that the program would eventually be available on a webserver. It should be investigated as to how the program would be ported to a webserver in future work. We had investigated a framework, but found that there was no documentation for the deployment of the framework. This short list of nonfunctional requirements is for the webserver when it is implemented.

- Must support a Java web framework.
- Must support a SQL database.
- Network load needs to be investigated (size and bandwidth).

6.3 Acceptance Tests

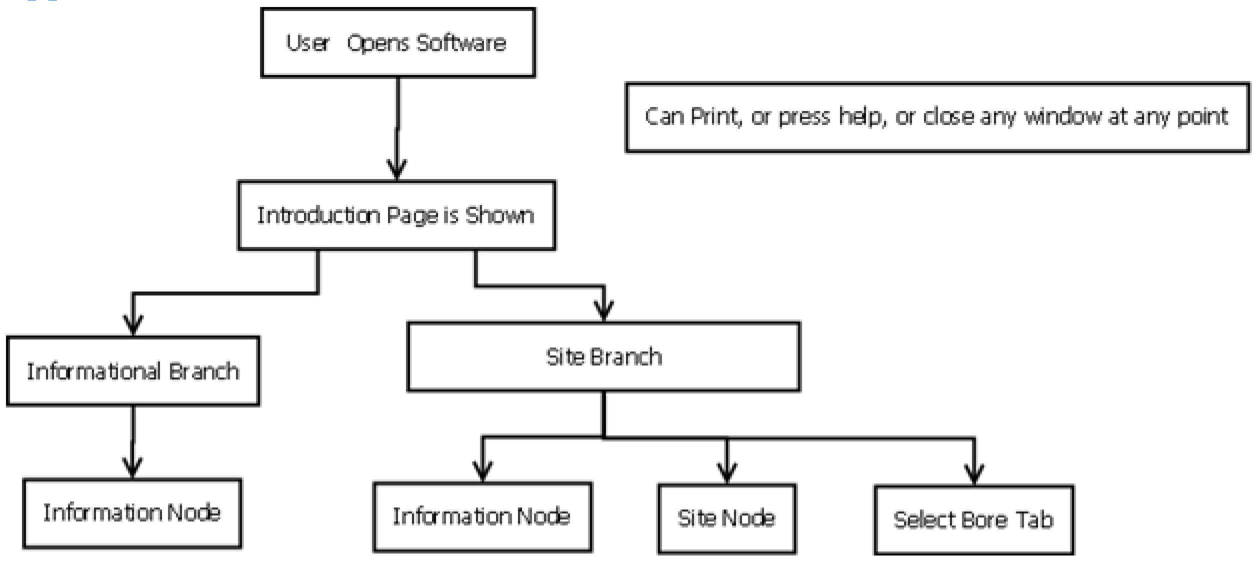
This list of acceptance tests are features that cannot easily be tested using JUnit tests, however are important features of the old program that still need to be implemented.

- Can the user save and load an investigation?
- Can the user print any screen?
- Can the user install a well of any appropriate size and depth?
- Can the user perform tests on any existing boring log?

- Can the map be panned and zoomed

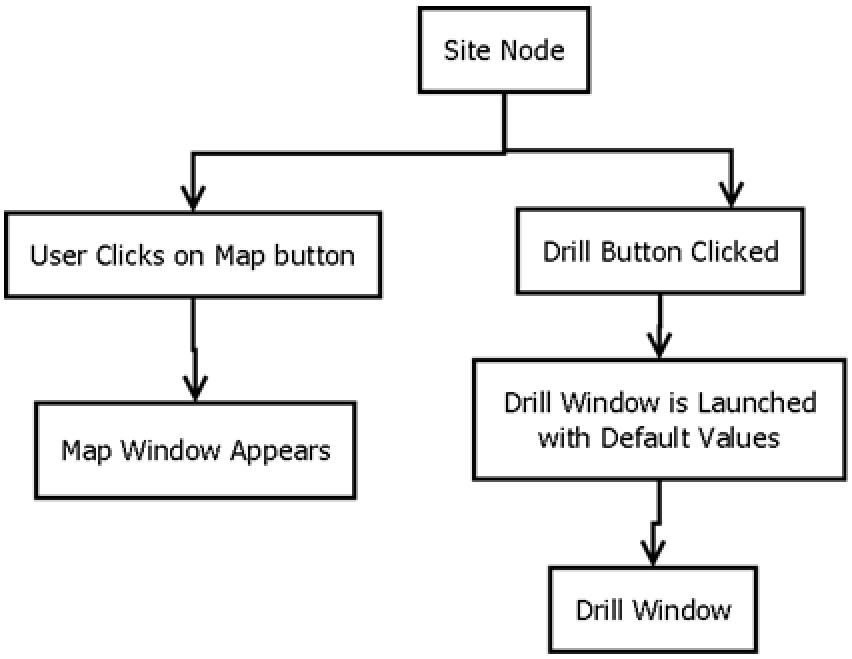
Appendix

Appendix A-1



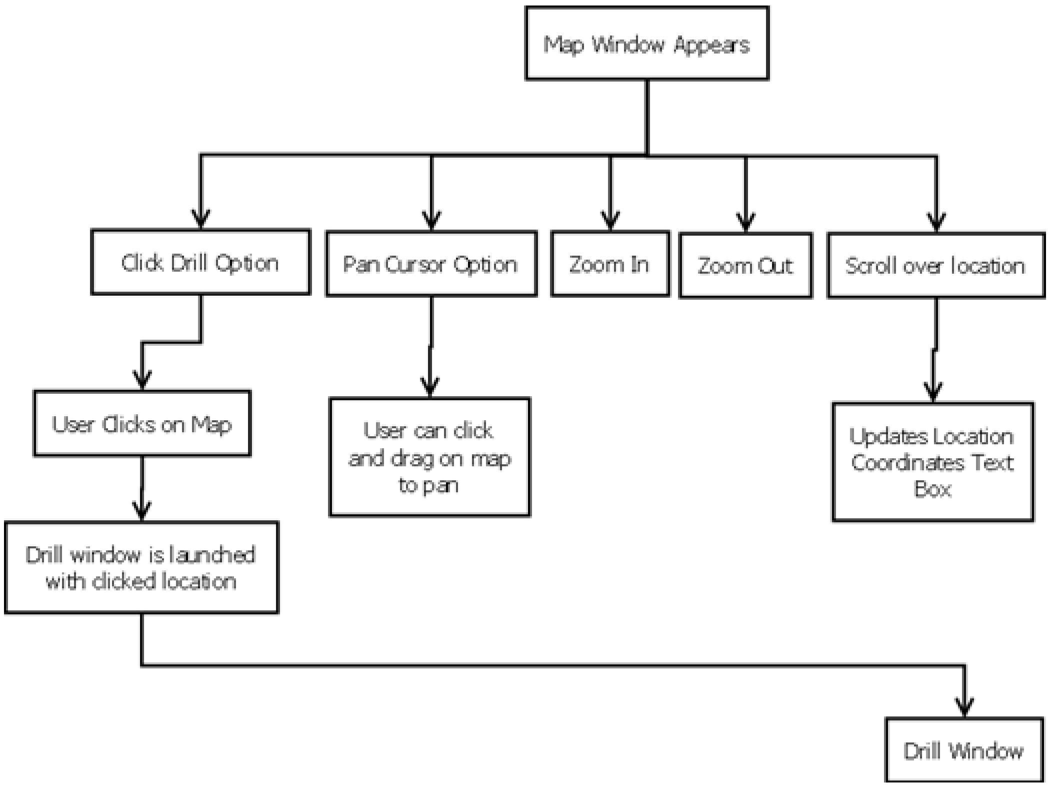
This flow represents the choices the user can make upon first opening the program. From here the user is directed to choose a node from the tree on the left side of the program. The tree is a list of all the possible selections for a user and a node being one of the selections. Information nodes simply display text, while the site branch accounts for the majority of the interaction. This is continued on Appendix A-2 and A-4.

Appendix A-2


Site Node
User Clicks on Map button
Map Window Appears
Drill Button Clicked
Drill Window is Launched with Default Values
Drill Window

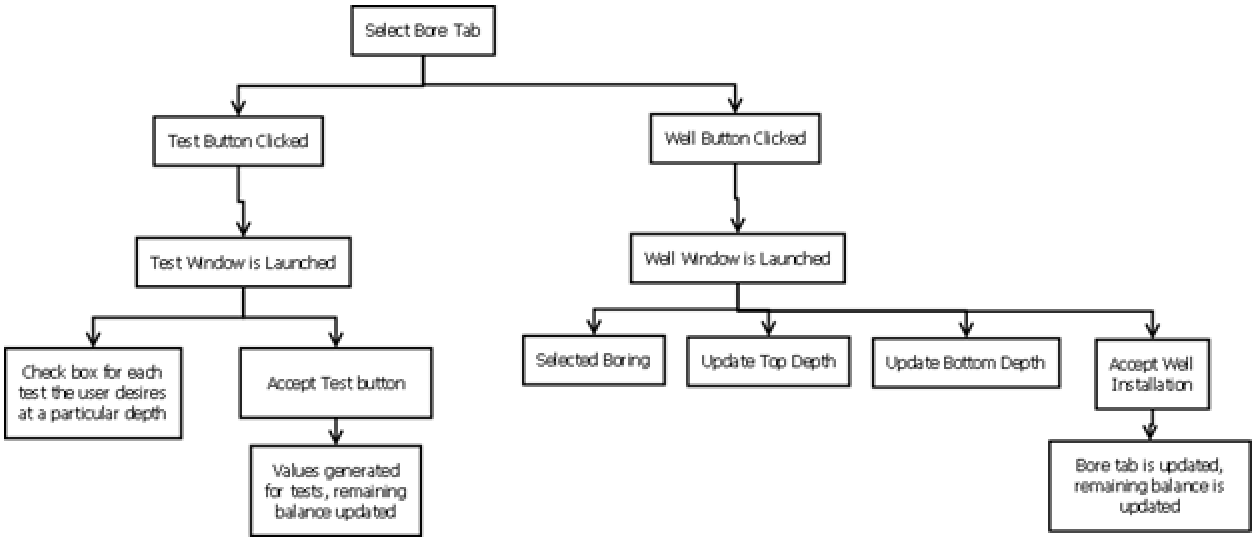
This represents the flow of interaction the user could take upon opening up site node from the site branch. The interaction includes the buttons that are available at the top of the GUI. This is a continuation from Appendix A-1 and is continued on Appendix A-3 and A-5.

Appendix A-3


Map Window Appears
Click Drill Option
Pan Cursor Option
Zoom In
Zoom Out
Scroll over location
User Clicks on Map
User can click and drag on map to pan
Updates Location Coordinates Text Box
Drill window is launched with clicked location
Drill Window

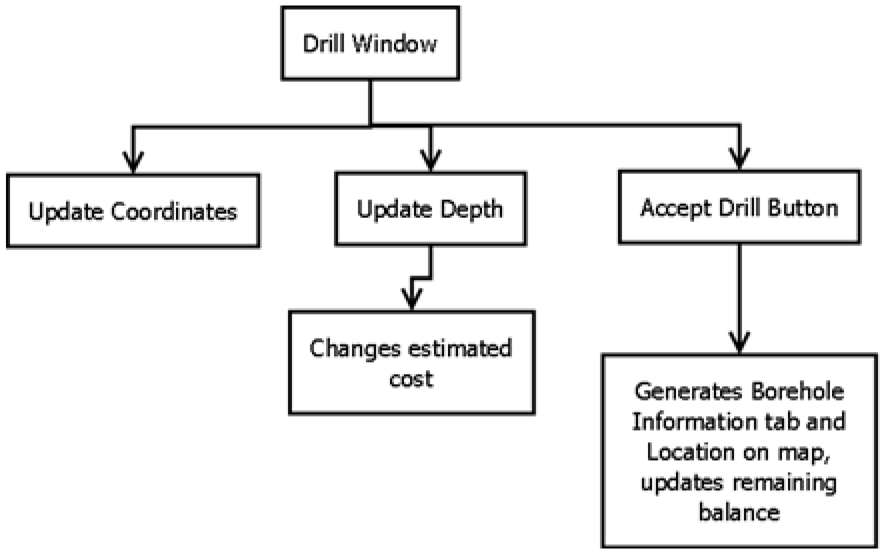
This figure is representative of the interaction the user can take with the map window. All the boxes one level below the Map Window Box are button options for the user to click. This is a continuation from Appendix A-2 and is continued on Appendix A-5.

Appendix A-4



This figure represents the flow of user interaction possible upon opening a bore tab from within a site. This is a continuation from Appendix A-1.

Appendix A-5



This figure shows the interaction a user can take from selecting the drill button and the drill window appearing. This is a continuation of the flow from both Appendix A-2 and A-3.

Appendix B

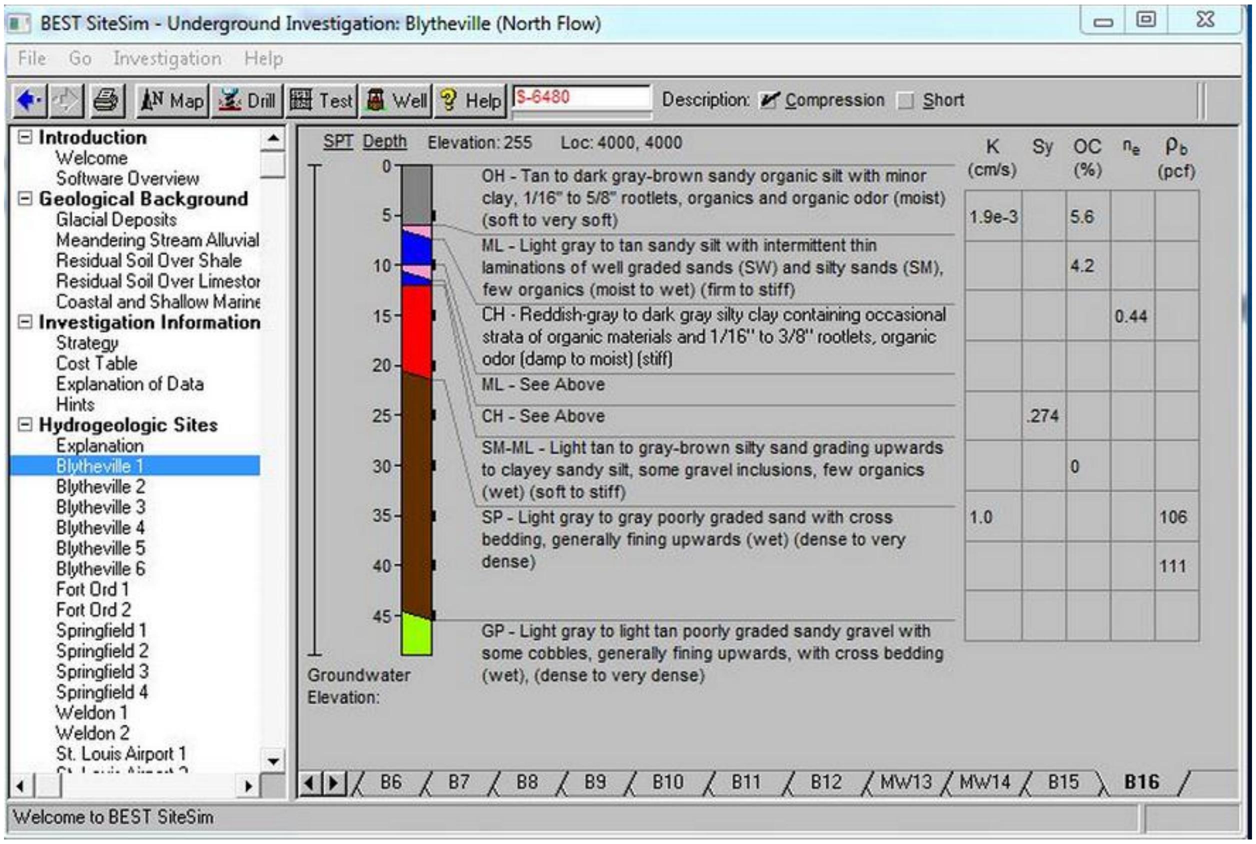
The following text is an excerpt from a definition file of the SiteSim program. It contains descriptions and parameters for the geological calculations. Each calculation takes the parameters in order, for example to find K at depth 5 ft., it would take 0.0011 = A, 0 = B, 0.0009 = C, 0 = D, and 5 = Z in the equation (A + (B*Z)) + ((C + (D*Z))*R). The last number is the type of distribution that was used to acquire these data points.

```
[Soil2]

USCS = OH
desc = Tan to dark gray-brown sandy organic silt with minor clay, 1/16" to 5/8" rootlets, organics and organic odor (moist) (soft to very soft)
sdesc = Gray-brown sandy organic silt
color = 132,132,132
pattern = 64
grad = 3,5,6,7,8,9,10,-1
rock = False
SPT = 3, 0.25, 2, 0, 0
Dd = 75, 1, 7, -0.5, 1
w = 90, 2.0, 15, -0.75, 2
Tf = 200, 25, 200, 0, 1
Cc = 0.23, -0.006, 0.05, -0.001, 1
e = 2.1, 0, 0.45, 0, 1
LL = 140, -1, 18, -0.5, 1
PI = 80, 10, -0.25, 1
K = 0.0011, 0, 0.0009, 0, 2
```

Sy = .08, 0, .02, 0, 1
%OC = 9, -0.75, 2, -0.125, 1

Appendix C


BEST SiteSim - Underground Investigation: Blytheville (North Flow)

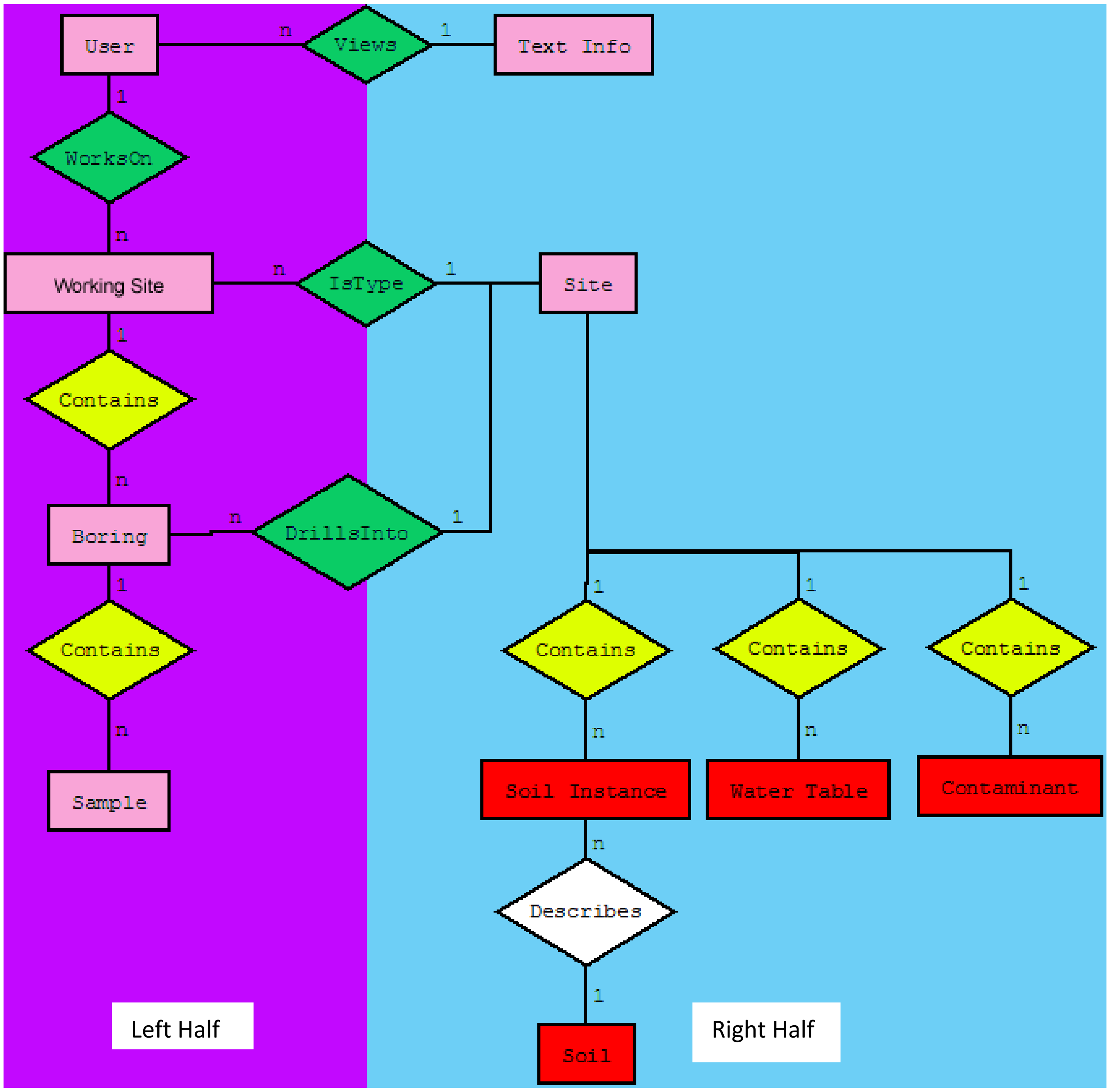
SPT Depth Elevation: 255 Loc: 4000, 4000

Depth	Description	K (cm/s)	Sy	OC (%)	n_e	ρ_b (pcf)
0–5	OH - Tan to dark gray-brown sandy organic silt with minor clay, 1/16" to 5/8" rootlets, organics and organic odor (moist) (soft to very soft)	1.9e-3		5.6		
10	ML - Light gray to tan sandy silt with intermittent thin laminations of well graded sands (SW) and silty sands (SM), few organics (moist to wet) (firm to stiff)			4.2		
15	CH - Reddish-gray to dark gray silty clay containing occasional strata of organic materials and 1/16" to 3/8" rootlets, organic odor (damp to moist) (stiff)				0.44	
20	ML - See Above					
25	CH - See Above		.274			
30	SM-ML - Light tan to gray-brown silty sand grading upwards to clayey sandy silt, some gravel inclusions, few organics (wet) (soft to stiff)			0		
35	SP - Light gray to gray poorly graded sand with cross bedding, generally fining upwards (wet) (dense to very dense)	1.0				106
40						111
45	GP - Light gray to light tan poorly graded sandy gravel with some cobbles, generally fining upwards, with cross bedding (wet), (dense to very dense)					

Groundwater Elevation:

This is a boring log from the previous version of the program. The different colors on the program represent different simulated soils and display a scale to see what depths different soils can be found. On the right side is a list of the tests that can be performed and any results that were found when the test was performed at a specific depth. It has all the features the client wants that is needed to be implemented in the new version.

Appendix D-1



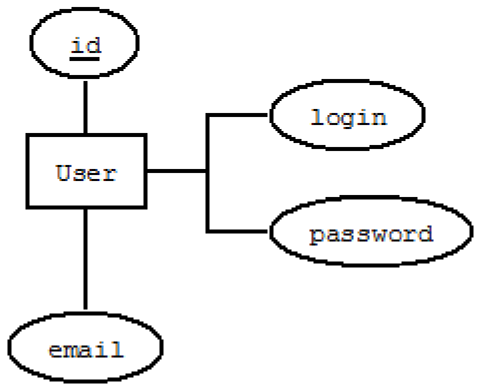
This figure represents the relationships for how the data will interact during the program's usage. There are ten entities in the figure represented by bolded boxes. These entities can be thought of as a conceptual grouping of information.

It should be noted that entities on the left half of the figure represent tables that will be edited during use of the program to represent the user's current work. Entities on the right half of the figure represent tables of general information accessed by all users. The right half tables never change.

The diamonds represent relationships in the form of verbs. The numbers and letters on each diamond represent cardinality. For example a user works on many site instances, but a site instance has only 1 user.

Appendix D-2

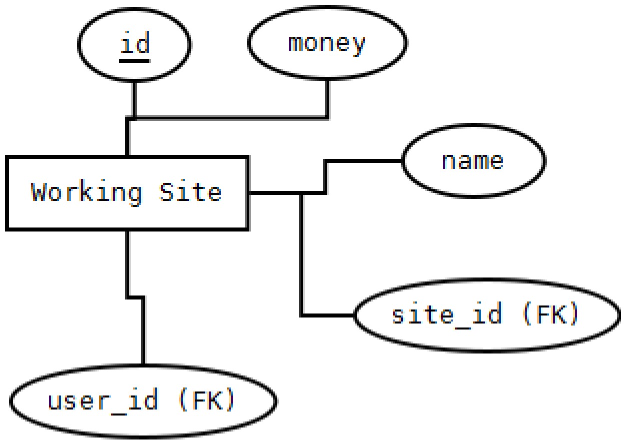
User: This entity stores information of the user that logs in to the website. It includes attributes such as login and password. This entity is currently not used, but is included for the future web integration.



```
DROP TABLE IF EXISTS "user";
CREATE TABLE user(
           id INTEGER PRIMARY KEY,
           login VARCHAR(32) NOT NULL,
           password VARCHAR(32) NOT NULL,
           email VARCHAR(32)
           );
```

Appendix D-3

Working Site: This entity is responsible for holding information pertaining to a particular working instance of a site(ie Blytheville1, FortOrd2, etc.). This can be thought of as the primary save files for the user.

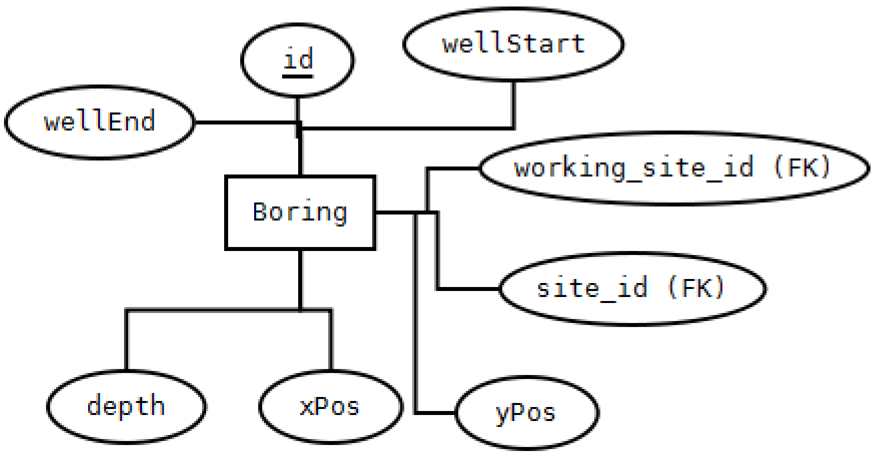


```
DROP TABLE IF EXISTS "working_site";
```

```
CREATE TABLE "working_site" ("id" INTEGER PRIMARY KEY  NOT NULL ,
"name" VARCHAR, "site_id" INTEGER NOT NULL , "user_id" INTEGER NOT
NULL , "money" INTEGER NOT NULL );
```

Appendix D-4

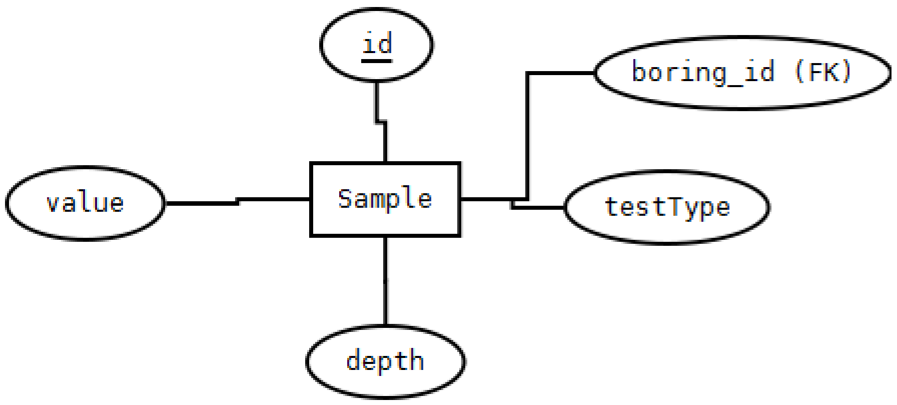
Boring: This entity represents the various borings that a user can create at a site. It has information such as x location, y location, a reference to where it was drilled in the earth model, and how deep it was drilled.



```
DROP TABLE IF EXISTS "boring";
CREATE TABLE "boring" ("id" INTEGER PRIMARY KEY  NOT NULL ,
"working_site_id" INTEGER NOT NULL , "site_id" INTEGER NOT NULL ,
"xPos" INTEGER NOT NULL , "yPos" INTEGER NOT NULL , "depth" INTEGER
NOT NULL , "wellStartDepth" INTEGER NOT NULL , "wellEndDepth" INTEGER
NOT NULL );
```

Appendix D-5

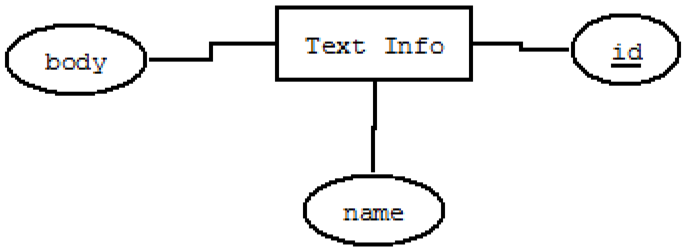
Sample: This entity represents which samples have been purchased for a particular boring. It contains the depth the sample was bought at, the type of test, and the value of the test.



```
DROP TABLE IF EXISTS "sample";
CREATE TABLE "sample" ("id" INTEGER PRIMARY KEY  NOT NULL ,
"boring_id" INTEGER, "testType" VARCHAR, "depth" INTEGER, "value"
DOUBLE);
```

Appendix D-6

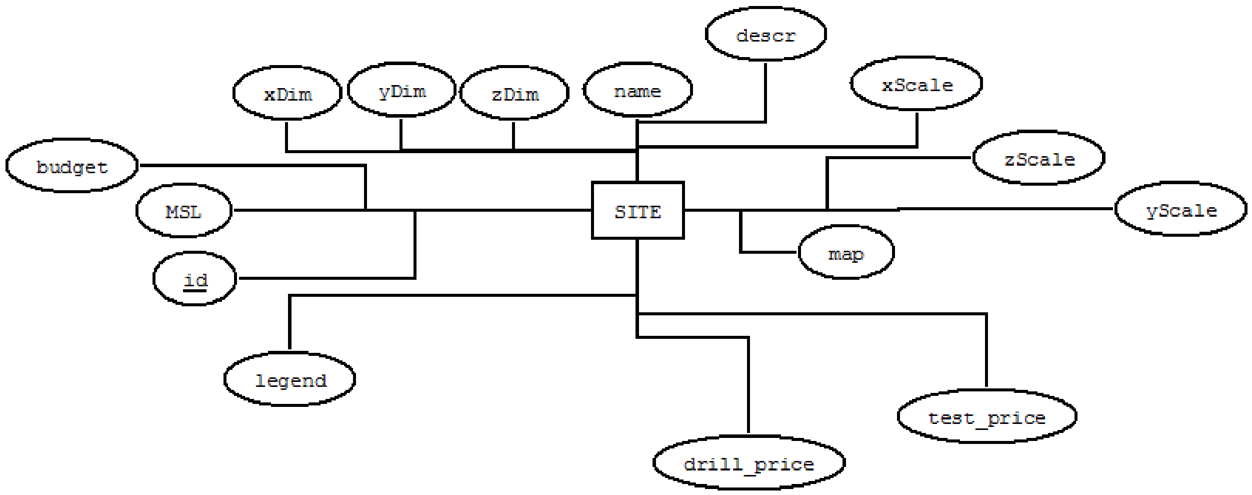
Test Info: This entity stores text that is displayed to the user in several of the program panels. The information is stored as HTML so that the formatting is handled correctly.



```
DROP TABLE IF EXISTS "text_info";
CREATE TABLE "text_info" ("id" INTEGER PRIMARY KEY  NOT NULL , "name"
VARCHAR NOT NULL , "body" TEXT);
```

Appendix D-7

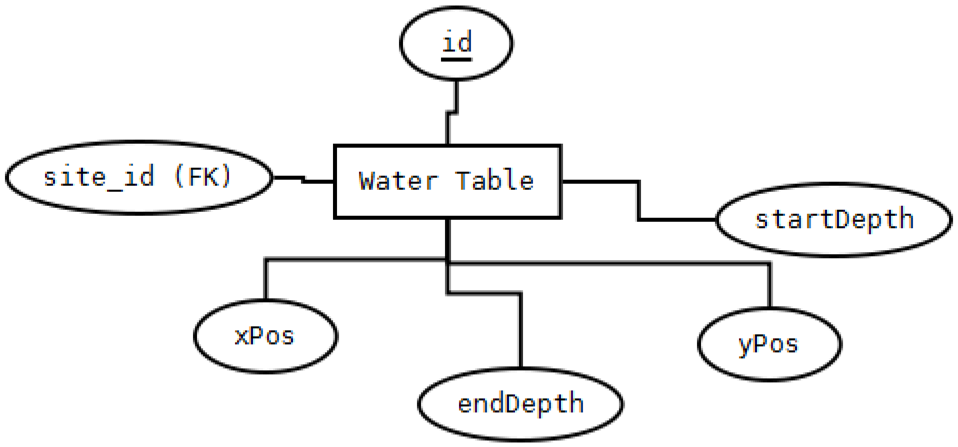
Site: This entity represents general information about the site, including name, scaling, map reference strings, and drilling and testing costs at the site.



```
DROP TABLE IF EXISTS "site";
CREATE TABLE "site" ("id" INTEGER PRIMARY KEY  NOT NULL , "name"
VARCHAR NOT NULL , "description" VARCHAR NOT NULL , "xScale" INTEGER
NOT NULL , "yScale" INTEGER NOT NULL , "zScale" INTEGER NOT NULL ,
"testPrice" INTEGER NOT NULL , "drillPrice" INTEGER NOT NULL , "MSL"
INTEGER NOT NULL , "budget" INTEGER NOT NULL , "xDim" INTEGER NOT NULL
, "yDim" INTEGER NOT NULL , "zDim" INTEGER NOT NULL , "map" BLOB,
"legend" BLOB);
```

Appendix D-8

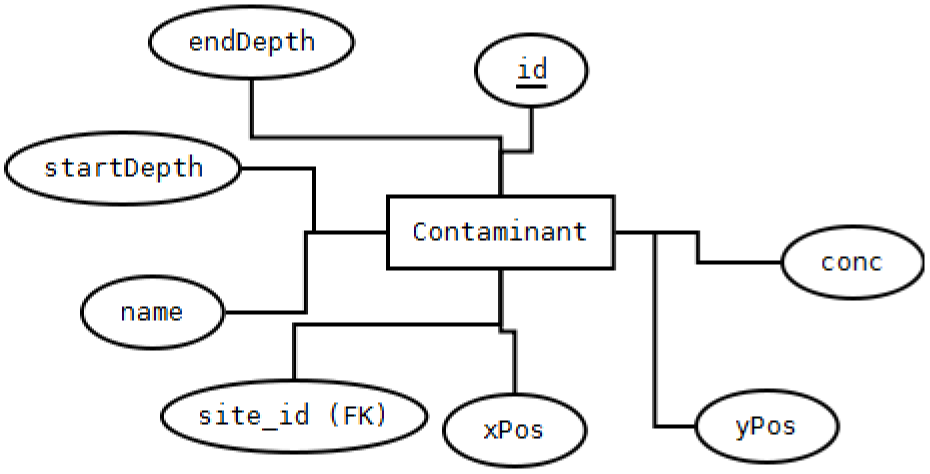
Water Table: This entity stores water table information. This information is retrieved via the x and y position attributes when needed.



```
DROP TABLE IF EXISTS "water_table";
CREATE TABLE "water_table" ("id" INTEGER PRIMARY KEY  NOT NULL ,
"site_id" INTEGER NOT NULL , "startDepth" INTEGER NOT NULL ,
"endDepth" INTEGER NOT NULL , "xPos" INTEGER NOT NULL , "yPos" INTEGER
NOT NULL );
```

Appendix D-9

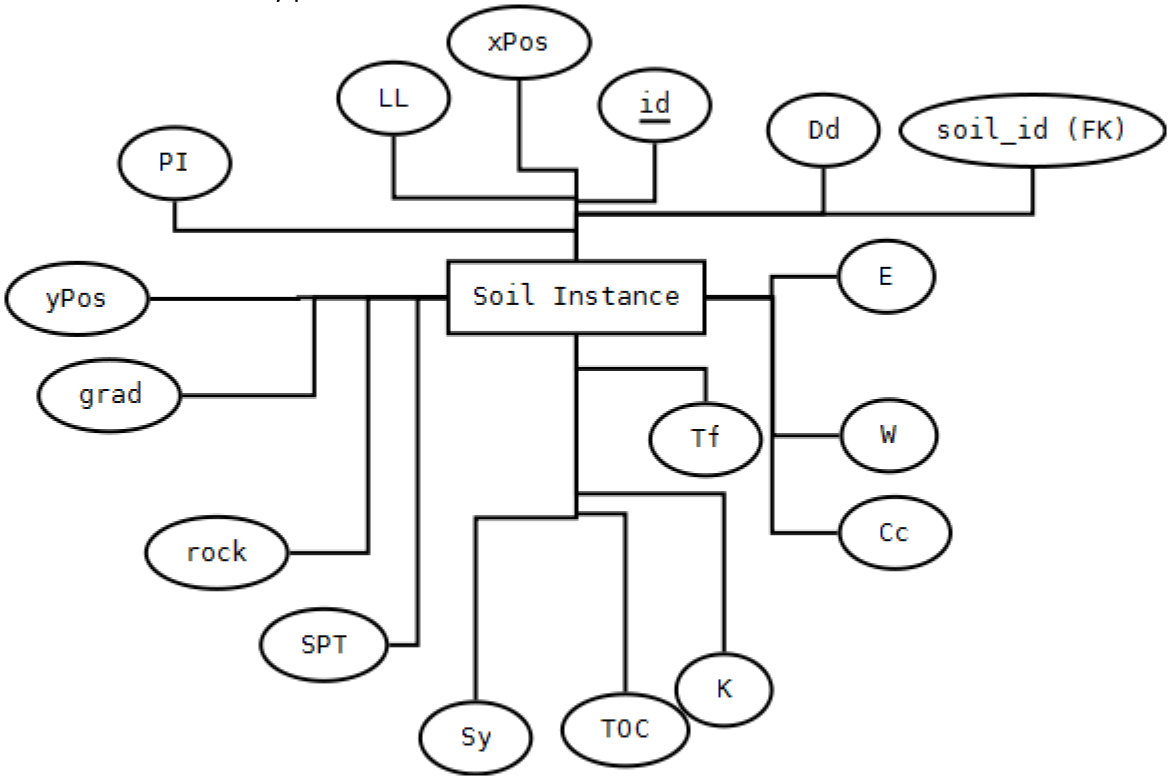
Contaminant: This entity stores contaminant information. This information is retrieved via the x and y position attributes when needed.



```
DROP TABLE IF EXISTS "contaminant";
CREATE TABLE "contaminant" ("id" INTEGER PRIMARY KEY  NOT NULL ,
"site_id" INTEGER NOT NULL , "name" VARCHAR NOT NULL , "conc" DOUBLE
NOT NULL , "startDepth" INTEGER NOT NULL , "endDepth" INTEGER NOT NULL
, "xPos" INTEGER NOT NULL , "yPos" INTEGER NOT NULL );
```

Appendix D-10

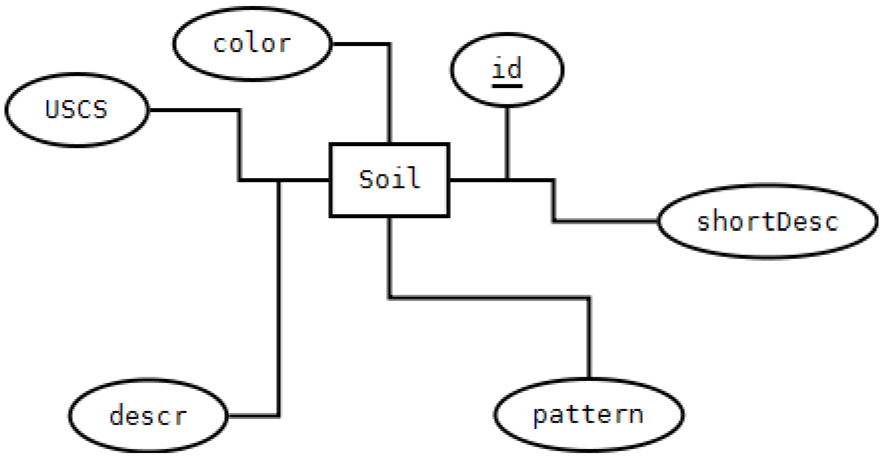
Soil Instance: This entity stores soil instance information. The difference between soil and soil instance is that soil instance describes a soil at a particular position, whereas soil describes general soil information. Soil Instance therefore has a foreign key to soil instance. This information is retrieved via the x and y position attributes when needed.



```
DROP TABLE IF EXISTS "soil_instance";
CREATE TABLE "soil_instance" ("id" INTEGER PRIMARY KEY  NOT NULL ,
"site_id" INTEGER NOT NULL , "soil_id" INTEGER NOT NULL , "Dd" DOUBLE,
"E" DOUBLE, "W" DOUBLE, "Tf" DOUBLE, "Cc" DOUBLE, "K" DOUBLE, "TOC"
DOUBLE, "Sy" DOUBLE, "SPT" DOUBLE, "rock" BOOL, "grad" INTEGER, "PI"
DOUBLE, "LL" DOUBLE, "startDepth" INTEGER NOT NULL , "endDepth"
INTEGER NOT NULL , "xPos" INTEGER NOT NULL , "yPos" INTEGER NOT NULL
);
```

Appendix D-11

Soil: This entity stores general information about the soil, as well as how it is displayed to the user.



```
DROP TABLE IF EXISTS "soil";
CREATE TABLE "soil" ("id" INTEGER PRIMARY KEY  NOT NULL , "name"
VARCHAR NOT NULL , "USCS" VARCHAR NOT NULL , "descr" VARCHAR NOT NULL
, "shortDesc" VARCHAR NOT NULL , "pattern" INTEGER NOT NULL , "color"
INTEGER NOT NULL );
```

Glossary

Boring - The act of drilling a hole, tunnel, or well in the earth.

Boring Log - A 2-dimensional representation of a boring, the graph has color flags for different types of soils.

DAO - Data Access Object, a design pattern that provides an abstract interface to some databases.

Database ERD - Entity Relationship Diagrams, conceptual schema used to model a relational database.

Flat File - Files of various formats that must be loaded directly from a hard drive to be used.

GUI - Graphical user interface, this is the part of the program that a user will generally interact with.

Hydrogeological - The area of geology that deals with the distribution and movement of groundwater in the soil and rocks of the Earth.

IDE - Integrated Development Environment, a software application that provides comprehensive facilities to computer programmers.

Geotechnical - Relating to the behavior of earth materials.

SQL - Structured Query Language, a database computer declarative language designed for managing data in relational database management systems.

SQLite - A software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

JDBC - Java Database Connectivity, an API for java that defines how a client may access a database.

JVx - An open source framework for java that facilitates the development of database applications.