<u>CS1530 – SPRINT 2 DELIVERABLE</u>

<u>PROJECT</u>: Cemetery Plotter

<u>MEMBERS</u>:
Jiajie Dang – Software Engineer
Michael Faller – Software Engineer
Daniel Kindler – Project Manager
William Rossi – Quality Assurance
Xiaokai Wang – Software Engineer

<u>DUE</u>: 13 OCTOBER 2015

2. Description

In this sprint we decided to start implementing the necessary models needed to eventually fulfill user stories one through six. This was somewhat difficult as we have had limited communication with the Synagogue (up until October 9th when we finally established communication) and were unsure of what specifics the cemetery manager required for our software. We imagined ourselves as cemetery manages and thought of different situations that we may find ourselves in. After Dan spoke to the Congregation executives, we were surprised as to how similar the model we predicted was to the actual model required.

Our project manager, Dan, has had a lot of problems communicating with the Synagogue and its executive staff. After two weeks of no email responses, Rob Menes (executive director of Beth Shalom) finally set an appointment with Dan on Friday October 9th. Dan met with Robert, the Executive Director of the Congregation and Lonnie, the Director of the Congregation's cemetery.

Lonnie was a big help. Although Robert was the Executive Director of the Congregation and appeared to have the final say, it was Lonnie who had extensive knowledge of the Cemetery and its current system.

What appeared to be the biggest issue for Lonnie was the move from local storage to the cloud. Lonnie stressed that he would like the information to be available anywhere and everywhere. He wanted to have access to the database when he moved to Florida. We are now considering using a cloud storage solution, such as Parse.com, to host and manage Beth Shalom's database.

After five minutes of talking to Rob, it was very obvious that he had extensive knowledge of computer systems. He had high ambitions for the software that we are to develop. One of his desires was that we parse through the existing systems output files and develop an input clause into our program that allowed for the importing of old files. Another request he had was geological integration of google maps to help the user find grave plots with ease.

Although Dan thought these ideas would improve the external quality of the program, he reminded Rob that the schedule of the project was extremely limited. There are approximately two months of development, meaning about eight sprints. He ensured Rob that the project's primary goal was to provide GUI software for the Congregation that would allow the Cemetery managers to better assess the database. However, Dan maintained that if these goals could be met relatively early, additional features such as file inputs could be considered and even implemented. Rob was happy with the proposition, and even asked if it would be okay if we provided him with source code after the project is completed; we agreed.

Our team struggled with version control this sprint. We are using the IntelliJ IDE to write and run our code, and it turns out that the Git integration with the software is sort of confusing. When a project is created, a "workspace.xml" file, along with multiple other files basically just containing setup information for the project, is created. This workspace.xml file is unique to each IntelliJ user and is updated upon every save, so when we would merge this file to the master branch of our repository we would overwrite another teammate's workspace.xml file. When that teammate pulled the new workspace.xml file, his local copy of the project would malfunction. To resolve this issue, we included the workspace.xml file in our repository's gitignore file.

Additionally, those of us using Git from the command line to pull, push, merge, etc. would run into errors when we would try to update the project within IntelliJ. For some reason, IntelliJ forces us to use Git commands from within the IDE itself in order to effectively accept the changes in our local copies of the project. This caused some frustration, but ultimately we chose to stick with IntelliJ because the solution to the problem was not a big inconvenience to anybody.

We faced numerous challenges writing the code. First and foremost was designing the overall system architecture without having yet spoken to Rob at Beth Shalom. Because we did not have the specifications nailed down, we were making frequent changes to field variables that had a cascading effect on other features we were implementing concurrently. For example, we learned later that the Synagogue uses a pre-existing identification number for each interred person. This required changes in our compareTo, equals, and toString methods, but was overall easier to work with. There was a lot of rewriting of early code. Another difficult feature to implement was saving and loading a file. We opted to create a custom file format. This meant each toString method had to be in a very specific format for reading back into the system and we had to maintain consistent constructors. This again was difficult to set up, but really helped drive the development and to create a consistent codebase.

Testing for this sprint was not very difficult. As it stands, there isn't any user input besides a "cemeteries.db" file that contains all the necessary data we are trying to store about our cemetery. Coming up for tests for this version of our product was a fairly simple task because of this. Testing the Set and Get methods of the different classes was important. To ensure that the Section class was working properly, we tested that you could both add and remove Plots from a section. We also tested that each class’s toString() methods were working properly. Other than that, there was not much to test at this time. We made the class variables public to expose them to the test classes, but we are worried this will lead to a leaky abstraction, so we plan to switch them back to private and utilize the set and get methods for verification.

Our object-oriented approach was to look at the cemetery from an is-a and has-a perspective. A cemetery has named sections. Sections have numbered plots. Plots can have an owner and/or an interred person. Plots, interred people, and owner people all have their associated data fields which need to be accessible and mutable. We want sections, plots, interred people, and other people to all be stored in lists for lookup and management functions. We used ArrayLists for now, but anticipate refactoring to use HashMap’s for their constant-time lookup capability.

3. User Stories Completed in Sprint 2

GitHub repository: **https://github.com/wjrossi/Cemetery-Plotter**

1. As a user, I want the plots to be organized by section and number in order to better locate plots.
2. As a user, I want to be able to categorize plots as unused, used, available/ready, reserved/purchased so that I can manage usage of the land.
3. As a user, I want to be able to list the names of those who are interred in the cemetery to find what plot they are located in.
4. As a user, I want to be able to record a list of all the plots belonging to a person so that I can search for all the plots they reserved or own.
5. As a user, I want to be able to update the status of a plot and add, remove, or modify its associated information so that I can keep accurate records.
6. As a user, I want to be able to load and save the data representing the cemetery so that I can make permanent changes and backup the data externally.

4. User Stories Details

The user stories we completed were our highest priority user stories from the first sprint. They were high priority because they will allow for more advanced functionality later, like a GUI. As we fleshed out our object-oriented approach, we had to modify some of our goals for the individual stories to ensure we were focusing on representing the underlying structure of a cemetery rather than the management functions of the driver program. They were also modified as we got new information from the customer. The final user story regarding loading and saving of data was also deemed critical for this sprint. We want to model the cemetery as it exists and maintain its state. This means we must be able to work with a set of data. We are currently investigating more robust, stable methods for this for a future sprint. Starting with the data layer will allow us to integrate the application layer next.

5. Testing: Defects & Fixes

A simple defect testing uncovered was the Section class reporting an incorrect size. The test in question was SectionTest.testGetSize(). The test created a Section object with a size of 1. Naturally, we expected it would return a size of 1, but we neglected to add a plot to that section's list of plots. However, we saw getSize() returned 0. It appears the size() of ArrayList returned 0 to Section.getSize() because it tracks the number of items in the list, rather than the constructed size of an empty list. We determined this was the preferred behavior and modified testGetSize() to add a plot to the section before testing the size.

We also had some minor defects regarding compilation and execution of the test classes. This was mostly due to conflicts with IntelliJ and git not playing nicely together. One person would modify configuration files and they would get shared in the repository, and then it would cause another problem on someone else's computer. In the end, we solved this mostly by modifying the .gitignore file.

Another minor defect we had was using the Date data type or the new Calendar data type to store our burial date and purchase date variables. This was due to different versions of Java we discovered we were using. Parts of the Date class were deprecated in favor of Calendar. Eventually, we figured out how to use Date objects in conjunction with the SimpleDateFormat class to parse Strings using a date format pattern to resolve the issue.

One major defect we faced was with our save/load functions. In theory if we made no changes after loading cemetery.db, then it should be the same when the program exits and saves the file, but it had a different size after saving. This required a lot of debugging and code tracing as it was the result of compound errors relating to the toString methods as well as incorrectly formatted data. Once it was loading and storing correctly, another defect was revealed. It was reporting an incorrect number of people in the cemetery. Each plot can have an owner and/or an interred person, so either can be null and we had no way to account for that in our file format structure.