# CS1530 – SPRINT 4 DELIVERABLE

**PROJECT:** Cemetery Plotter

**MEMBERS:**
Jiajie Dang (djjkobe) – Software Engineer
Michael Faller (mdfaller) – Software Engineer
Daniel Kindler (dkindler) – Project Manager
William Rossi (wjrossi) – Quality Assurance
Xiaokai Wang (daytimebat) – Software Engineer

**DUE:** 9 NOV 2015

## 2. DESCRIPTION

During this sprint we continued to communicate by using Slack. Slack has continued to prove itself as a viable and reliable tool for communication, and we have had no communication related issues via Slack.

One very nifty feature about Slack that we have set up are notifications. We have designated a #notification channel in Slack where web-hook enabled notifications are posted. For example, any time any GitHub action is done, e.g. something is pushed or there is a pull request, a notification is sent to our team via Slack. This makes the whole experience better for us as a team.

We had a minor disagreement about how to implement exiting the program.  We had two different versions being worked on simultaneously.  We choose which to us based on merit.  The implementation that won intercepts the exit signal from multiple sources and deals with all of them in the same method, in the same way.

We did decide to use Trello in the beginning of the semester but never followed through. We believe this is because our communication tool, Slack, was sufficient enough in designating tasks to each team member. As of now we know it is something we should set up, but we have no real incentive to do so since our current process is working.  We did try to do more code reviews and pull requests.  It was a mixed bag for us because we don't have a well-established procedure in place for it.  One person would say they were going to do the review, but then didn't get it to it, or didn't say they were actively doing it, and another person might do it without saying anything and merge it before someone else looked, etc…  We were able to catch a few things so we will keep working at it.

We were hoping to make more progress with the functionality of our application before we got major feedback from Robert and Lonnie. It is imperative that we establish communication within the next week or so to ensure they are happy and content with the current GUI layout that we have devised. Their upcoming feedback and criticism of our process will be a crucial point in our project, and will be a pivotal moment where we begin to shape the end product that Robert and Lonnie are hoping for.

We continued to use our layered approach.  Our previous deliverables had functioning data and visual layers.  This deliverable has those two layers interacting to a large degree.  Data can be viewed and manipulated, and some elements can be changed and saved.  The top-down approach we have taken allows us to implement features and activate GUI elements already in place as we continue developing the program.  It is really convenient because the program continues to work even though lots of things still need to be done.

Isolating the data layer and the visual layer was an interesting task.  Some of the things we did earlier blurred the separation a little too much.  Formatting dates (and money) was one illustrative example.  It made sense for dates to be formatted in the visual layer and stored in the data layer because the dates were entered and displayed in the GUI.  So we had a JFormattedTextBox with a SimpleDateFormat attached to it that made sure we got our nice “MM/DD/YYYY” dates, but it had a lot of side effects.  If you typed anything into it, the user could never totally erase it again.  It just kept putting a date back in, and even then it’s behavior was hard to predict.  On top of that, we remembered the cemetery only has partial information regarding some graves which might only have a year.  So our solution was to split the date fields into separate month, day, and year JTextFields and apply Postel’s principle.  We treat all dates as Strings in the GUI and store it as a formatted Date object in the data layer.  This has the benefit of abstracting away the messy details of formatting and cleaning up our code.

We simplified the compareTo() method of our Person class by adding a unique contactID variable to each person; this resulted in simplification of the unit tests for the compareTo() method, reducing the number of tests from three to one. We restructured the InterredPerson class, adding a new Date object for birth day, month, and year, and death day, month, and year. Because this changed the constructors of the class, the unit tests needed to be entirely rewritten, and new tests needed to be added. We also created a scheme to track and create unique ID numbers for plots, people, and interred people, and these new methods required new tests.

Finally, with the addition of GUI functionality, we figured we would need new tests to make sure that the functionality was working properly. For example, when we click a plot, we want the owner of that plot and the person buried there (and all their relevant details) to be displayed. Initially, this seemed like a great deal of new testing. After some code scrutiny, however, we realized that these new tests would be unnecessary: our GUI was retrieving and displaying values using native Java methods like  setText() and getText(), while these methods were calling our own set() and get() methods in conjunction. Since we already wrote tests for our own methods, we decided against testing Java’s built-in methods, since they have a definitely-more-talented QA team of their own.

## 3. USER STORIES: COMPLETED THIS SPRINT

GitHub Repository: https://github.com/wjrossi/Cemetery-Plotter

1. As a user, I want to be able to view a list of all the sections in the cemetery and select them, so that I can see what plots are in the selected section(s).
2. As a user, I want to be able to see a list of people (interred and contacts) in the selected sections, so that I can access their information.
3. As a user, I want to be able to select a plot, so that I can view all the information associated with that plot.
4. As a user, I want to be able to edit the information for a selected plot, so that I can make changes.
5. As a user, I want to be able to cancel the changes I made while editing, so that I don't have to remember what was there if I don't want to make the change anymore.
6. As a user, I want to be able to save the changes I made to the plot, so that I can make permanent modifications.
7. As a user, I want to be able to save partial dates, because I have partial information about many older graves.
8. As a user, I do not need to store contact information for an interred person, so I only need it for the owner/contact person of a plot.

## 4. USER STORIES: DETAILS

The user stories we completed for this sprint are all about getting functionality into the user interface.  The functionality was dictated by the interface-stub we designed for the last deliverable.  While the data layer was already in place, it still underwent many changes to support the desired implementation of the business logic we are working to develop.  The features we got working allow the user to manipulate objects currently in the cemetery.  We felt it would be easier to add and remove new objects (plots, sections, etc...) once we could handle viewing and modifying what already exists.

## 5. CUSTOMER INPUT

As we said last week, after carefully assessing the needs that Robert and Lonnie have, we have broken down their expectations into three components. We have organized the needs of our customer into components so that we as a team can better assess what we are doing right and wrong. These components serve as a guideline to us as developers, as we consider them to be very important tools during our intensive development process.

1. Cloud Based
    a. As mentioned last week, it is imperative to the Congregation that we make this piece of software usable from any part of the world. Thus, we took measures to ensure that the database was stored in the cloud. Initially we planned to use Dropbox, but have since switched to Parse to store our database files. We will not be storing instances in the Parse database, but rather be storing one text-file which contains all of the information needed for the congregation's operations.
2. User Interface
    a. Again, Lonnie went into great detail about his current system which he claimed was almost as old as he is (he's in his 70s). His biggest concern, aside from the cloud based aspect of the program, was that it is to be a "twenty-first century" program, meaning it has a user-interface. We've already created a very well thought-out and detailed GUI that we hope Robert and Lonnie will both enjoy. We are waiting to receive feedback from the duo.
3. Information
    a. This aspect of the program was the most basic, and was hammered out in our first meeting with Robert and Lonnie. Since then we have had a very good understanding of the data needs of the synagogue, and have accommodated for them accordingly.

## 6. DEFECTS

Bad Date Formats (again)

1. **Reproduction Steps:** Select a Section.  Select a plot.  Click edit.  Type valid month, day, or year numbers in the GUI. Click Update.
2. **Expected Behavior:** The numbers should stay the same in the date fields, or if you put a single digit month, a leading zero would be added.
3. **Observed Behavior:** Each field would be changed to an entire date based on the Unix epoch, something around 1970, except the month, day, or year you inputted would be part of it.
4. **Solution:** We had to use our SimpleDateFormat objects, that “parse” the input date String into Date objects, to “format” the Date back into a String.