Sprint 1 Retrospective
Reservoir Planning Tool

Team 16: Clayton Marshall, Drew Atkinson, Jonah Heeren, Vritant Bhardwaj

What went well during sprint 1?

1. **Selecting right number of user stories.**

   We completed a large number of user stories within this sprint including a large section of the user's workflow including the main input and output of the algorithm. From this point on, we will be refining the user experience and adding additional features. We thought we might have been overestimating the time that we would be spending on each of the stories, but because some of us had to learn different technologies like Node.js and SQL, this allowed us time to research these and deal with issues as they presented themselves.

2. **Organizing Individual Tasks**

   We excelled in planning out our individual tasks and assigning them according to individual strengths. For example, Jonah has the most experience with MySQL so he took over creating scripts to initialize and update the database. In addition, Vritant is interested in UI and UX, so he designed the front end of the website and handled the user flow and graphing functions. Except for a few challenges, described later, most tasks were finished within the time that we estimated.

3. **Learned asynchronous programming**

   One challenge that comes along with using Node.js as a backend framework is understanding its workflow. Asynchronous programming was something most of us understood from an outside perspective, but had never encountered it in the wild. After writing the Transforming Drainage Project algorithm we quickly realized that the return statements were running before the database was finished querying. We had to rewrite this along with other database calls throughout our program in the form of a promise in order to get correct return output. This took some time to understand, but we are now well versed in how this process works and will be able to fix any future encounters with ease.

4. **Version Control and Communication**
   Many of the members of the group were not familiar with git, and through this sprint learned version control. We maintained multiple branches for each user story, corresponding to the tasks that were part of it, and constantly used Pull Requests. Pull Requests made sure that each member's branch was checked by the others before it merged onto the master branch. Along with Github, we had consistent communication on Slack, and helped each other to solve errors that were encountered.

What did not go well?

1. **Algorithm confusions with project owners**

   A large part of our program is leaning on the output of the Transforming Drainage Project's algorithm. The project owners had given our team documentation on how the application should work along with pseudo-code for the algorithm. We faced a few challenges when writing the algorithm due to a lack of detail in their documentation. We frequently had to ask questions on slack to clarify details and eventually had one of our team members meet with them specifically to discuss specifics about the implementation of the algorithm.

2. **Setting up MySQL server on machines**

   Setting up individual MySQL servers and databases on each machine was a time consuming process as we did not have a server and database set up in the cloud. There were multiple unique problems on different operating systems which led to an increased unnecessary usage of time. Permission errors while accessing databases, coupled with errors in the installation of MySQL was something we did not anticipate, and will keep in mind of such possible cases in the future.

3. **Synchronous v. Asynchronous Database Setup**

   The ideal implementation of our database setup task was that it would be done in node to maintain continuity with the rest of the project. Attempting to do this asynchronously with node turned out to be a small disaster. The main issue was trying to keep a vague sense of synchronicity while within the context of CSV event listeners. The CSV event listeners which fired asynchronously created a huge issue in the context of managing open CSV files. Anytime a DB call was made to insert a new row, it would be pushed into the event loop and the stack would continue to fill up with constant time tasks relating to parsing the CSV. In the end, this resulted in keeping thousands of files open at the same time which would inevitably overflow node's 1.76 GB heap and cause the program to crash. One solution would've been to read all of these files into a matrix and then do DB inserts after parsing all CSVs. However, we were concerned about performance on what was already a decently large dataset and in the end decided that using python/synchronous programming would be a more efficient and readable solution.

How will we improve?

1. **Communication with the project partners**

   Our team did a good job of communicating with each other during the first sprint, but there could be some improvement in our communication with our project owners on both ends. In the upcoming sprints we want to plan weekly or biweekly meetings that we think will help clear up any confusion we have about the deliverables of the application. This will also help us during the testing process as well since no one in our group is an expert in crop drainage science.

2. **AWS Setup**

   We plan to fix a lot of the technical issues associated with developing on our local machines by beginning to deploy our site on Amazon Web Services. This way we can point our local machines to a central database which will house a single version of the data we are working with. This will solve file differences on our local machines and give us an environment closer to what we will have when we deploy our website finally. In addition, we will deploy the web server in our next sprint, which will mean we will have a live server to test our changes on. Also, the members of our team haven't spent too much time working with Amazon Web Services, so it is a good idea to do this earlier rather than later in case we run into issues.

3. **Reliable testing on all new code written**

   Currently we have no automated testing in our project and this is an issue for robustness and peace of mind. From now on, all new code that is testable will be required to have the appropriate testing accompany pull requests. Additionally, we will work to retroactively add testing to our already written modules.