Sprint 1 Retrospective

Team 19 – Sentinel Data Vault

Jiho Choi, Zhaoji Jiang, Adam Petty, Dingfu Sun, Thomas Worns

What Went Well?

1) **Communication**
The most important factor of a successful team project is the communication among team members. During Sprint 1, all of our team members were open-minded and willing to share our ideas, even when some of those ideas conflicted with others. This improved the efficiency and productivity of our research and development, which also allowed us to avoid potential misunderstandings. We also coordinated meetings, addressed development issues, and planned changes to the project through a project coordination app called Slack.

2) **Working as a Team**
We split our work into 3 parts: UI implementation, database research and setup, User and DataEntry class implementation. We made all the UI classes work together by following a common format and using the same libraries in order to easily connect individual classes, which prevented conflicts between implementations. By dividing the work in this way, we kept organized and worked efficiently. We tried to make sure everyone had a task to perform at all times, and that work was divided among team members as equally as possible. All team members were willing to help each other with any issues regarding understanding, code, or implementation.

3) **Meeting Format**
Our meetings strictly followed the schedule we set at the beginning of the sprint, and we kept our meeting times and location consistent throughout the sprint. Our meetings began with everyone updating the team with what they progress they had made and what they were currently working on, which included any development issues they were facing. Following the team check-in, we addressed the problems that anyone had and proceeded to do work as a team until we reached a suitable stopping point. This allowed our team to have periods of cooperative coding, which, in turn, enabled us to coordinate styles and fix issues more effectively.

4) **Database Research and Setup**
   This first sprint required a lot of research and learning with regards to the database aspect of the project. No one on the team had any background or experience with Relational Database Management Systems (RDBMS) or Structured Query Language (SQL). This meant that a lot of learning had to take place in this area. For the most part, this progressed smoothly, albeit at a slower pace than expected. We learned the function of databases, how to work with databases using SQL, and how to setup and manage SQL databases using MySQL. We also were able to begin to utilize these to complement our data vault system, such as what tables would be required and what information would need to be stored (including what form it would be in). We were successful in all these areas, however, it was not without its struggles, as is discussed in the section below.

5) **UI Development**
   During Sprint 1, our team was able to build and connect most of the important UI features according to plan. Developing the UI was a challenging task for us since we were not familiar with building Java GUIs. We engaged in a substantial amount of learning about GUIs in Java. Our team was able to successfully combine all the individual GUI classes into one working whole. We used GitHub version control to efficiently track our code. After combining GUI classes with each views, we use action handlers to respond user input in the GUI views. We used iterative development to incrementally evolve the working UI, ensuring that added features were actively working.

What Did Not Go Well?

1) **Sprint Task Planning**
   The progression of our sprint did not adhere to the plan established in our Sprint Planning Document. Originally, we had planned to accomplish more tasks than was realistically possible, such fully implementing the creation and management of data entries for users, along with handling multiple user accounts. As we realized this over the course of the sprint, we made significant adjustments to the Sprint Planning Document and the set of user stories to be implemented. We found ourselves deciding which tasks to prioritize and which new tasks need to be created as we went along.

2) **Database Implementation**

   User Story: As a developer, I want to have a database for storing user and data entry information.

Description of Unsuccessful Task	**Owner**	**Time**
Integrate the chosen RDBMS with the Sentinel Data Vault Java application.	Adam Petty	5

Failed Acceptance Criteria:
- Must have integrated the chosen RDBMS for use with Java and the data vault.

As mentioned above, the team hit a few roadblocks with regards to the implementation of the database and its integration with the main Java application. Most significantly, the entirety of the database research, setup, and integration fell into the hands of a single team member. This meant that all materials related to the database were on a single computer under a single system. Late into the database development during the integration phase, operating system compatibility issues and unavailable features led to a crash that resulted in loss of progress and a working database. This prevented the team from being able to have a complete, working database by the end of the sprint, and forced the team to alter its course of development.

3) **GitHub Organization**
For most of our team members, it was the first time using a version control system, so problems were expected before we actually started. GitHub organization turned out to be the problem we encountered during the sprint. Members pushed .class files to the “bin” folder of our repository and caused several minor merge conflicts. Fortunately, our main source files remained intact through those conflicts. We also experienced a couple mild problems with merging changes into the same file, but those were swiftly resolved with no lasting consequences.

4) **Time Management**
One of the problems that kept coming back repeatedly was poor time management. External factors like other classes, midterms, homework, and Fall Break got in the way of working on our project. We worked around these commitments as best we could, but they still had an impact on the work being done in the sprint and better planning for future sprints is a must. We also encountered some technologies that were time intensive to learn, such as the SQL database and the GUIs in Java.

5) **Making Cohesive Code Decisions**
After finishing each task individually, we spent more time than we would have liked integrating different classes since we didn’t have any coordination plan for these classes. For instance, confusing or conflicting variable naming schemes. Another example would be the initial lack of a cohesive aesthetic for UI during early builds. In some instances, we could have avoided this lack of cohesion if we had decided on a customized style or used system default UI components. Difficulties with integrating different classes was an another time consuming task we could have avoided if we had narrowed in on a design before we started coding.

How Should We Improve?

1) **Sprint Task Planning**
To improve in this area for the next sprint, we will adhere more strictly to the Sprint Planning Document. This will be done by planning out the sprint's tasks better and more realistically understanding the associate time demands. With the benefit of hindsight, this should be much easier to do for the next sprint as we are more familiar with the system, its framework and implementation, and how long specific tasks take to complete. We also want to avoid going back and making significant adjustments to the Sprint Planning Document, and being more confident with our selected set of user stories. Better prioritization of tasks and understanding of individual work efficiency will be key to improving this aspect of the project.

2) **Database Implementation**
Realistically, the responsibility of setting up the database should have fallen onto more than one member of the team, especially since the team as a whole was unknowledgable about databases and how to work with them. This would also have solved the problem of discovering the database's compatibility issues on certain operating systems. Secondly, at all stages during the database research and development, all progress should have been uploaded to the team repository, regardless of its quality or working state. This would ensure that in the event of a loss of files/data/progress (as happened during this sprint), we would be able to recover and adapt more quickly, keeping us more on track with the sprint. Proceeding with the next sprint, we will need to have at least two people working on the database – preferably with Mac OS *and* Windows operating systems – and to strictly save progress in the team repository.

3) **GitHub Organization**
All team members need to be extremely careful before committing any changes to our repository. We should also spend some extra time learning how to use GitHub in a proper and professional way, for example: using a gitignore file to deal with all the auto-generated class files. We can solve our disorganization problem by only pushing the files we are coding to the repository and keeping a clean structure should be our focus when using the version control system. Additionally, organizing our source files into packages should help keep the repository more manageable and understandable.

4) **Time Management**
   We can improve our time management by setting more realistic goals and being aware of upcoming exams, and upcoming breaks. As we become more familiar with the technologies we are using, we will encounter fewer unforeseen learning periods and we will have fewer errors and conflicts. Now that we have a feel for how the sprint process works and how much work we can do in a sprint, we will also be able to work more effectively. These factors will combine to allow us to accomplish more in the time we have. Finally, being more disciplined to set aside 10 hours a week dedicated to project work will help improve time management.

5) **Making Cohesive Code Decisions**
   Before starting actual coding, we will need to better coordinate and make team decisions on design, implementation, and code style. It's imperative that all members are on the same page when it comes to making these decisions. This will save a lot of time when it comes to integrating code.