*Summary of Product*

Our product is a genealogy tree that includes member names, birth and death information, and relationships. The system begins by importing a text file with each person’s information and their relationships to other members in the database. Each individual's information includes person ID, first and last name, suffix, birth date and place, and possible death date and place. The next section includes marriage information displaying the relationship ID, both person IDs, marriage date, and location. Finally, the last input from the text file is children relationships which are imported first using the relationship ID for the parents, then the person ID of the child. Once the initial database is created, there is a menu of functions that the user can perform on the database to find specific members or relationships. First, the user can add a new member to the database when a new marriage takes place or a child is born. They can also search for a person in a few different ways. They can find a person by inputting their first name, last name, or ID number to return the person and their information. There are also a few different relationships that the user can search for within the database. The user can search for parents by typing a person’s ID into the search. This will return both parents of the searched person. They can also search for children by inputting the relationship ID of the parents to return any children they have. Lastly, the user can find sibling relationships by a person ID in the search to return any of their sibling’s names. When the user is done with the genealogy tree, there is also an option to quit the menu.

*Quality*

Throughout the project, quality has always been a focus for our team. At first, we wanted plenty of features that would all do various things so we could show how many features we had. However, as the project progressed, we could not fit all the features we originally thought we could and continued with the focus shifting to the features currently implemented and anything that would be truly valuable. Quantity of features was no longer an issue and we had to focus on the quality of each feature. Attempting to make a bug-free feature takes a lot of time and some features still have bugs in them. As the deadline fast approaches, the decisions were made for what to fix based on the amount of time it would take to fix and how much time we had left. The quality of our product could be higher; however, all software teams should be saying this. We cannot make a perfect product, but with the right focus, we can make an acceptable product that can be supported and fixed along the way.

*Bugs*

Inevitably, our product would suffer from bugs. The only way we could have avoided these would be with infinite time and perfect knowledge of Java. A lot of minor bugs appeared and were easily identified and corrected. Others most likely slipped through and we will never know about them without extremely thorough testing beyond our stated use cases. One major bug that we identified early on was the getWife() and getHusband() method. Unfortunately, none of us found a solution to this bug and, in a real-world situation, our product would not be ready for consumers with this bug in place. If it was launched, a patch for this bug would have to be rolled out very quickly. Every person on a team has a specialty, whether that is redundant or not, except not all specialties will cover every possible situation. For this bug, we should have worked harder to find a solution outside of our own team. Beyond this major bug, our product, according to our use cases, showed it as bug-free as we could get it. The code inspections would help to identify and fix issues that helped with readability and logical or runtime errors. Much like every software team, deadlines decide how many bugs are shipped in the final product. We hope that major issues are not found in the future, and if they are, are easily fixed with minor changes.

*Architecture*

For our architecture, we had several views of it so that we could remain organized with our product and what we were working on. The use-case view, represented by the Use Cases document, allowed us to plan out what functions we wanted our product to have in depth. The Requirements document, which gave us a developmental view, gave us a list of focuses for each cycle and the functions for our product in detail. The process view, represented by the extended state machine model, allowed us to see a depiction of the structure, making it easier to visualize and structure our program. All three structural views allowed us to remain organized and plan for each cycle our tasks and goals. We split up the functions into each of the cycles so that our work would be spread out and even. This allowed us to focus on each task and function more thoroughly. Each of the functions on the documents were divided further into sub-functions, which furthered the quality of our work and allowed us to fine-tune the details.

*Teamwork*

From cycle one to the final cycle our group has learned quite a lot about teamwork and organization during the process of creating our genealogy tree app. In the beginning, we would simply split up each of the tasks and submit them but we realized that it was not enough to do that. Although this saved time it did not produce the best results because we did not understand all of the project’s components. We decided that in order to get the best results we would need to create a group deadline usually a day or two before the cycle deliverable. This would give all the team members enough time to go over each other’s work to ask questions, make suggestions, or improve our work before it was actually due. We also held meetings two days before the presentation so that we could all share our comments with each other and explain any of our feedback. Lastly, we were all open to collaborating and sharing the work. Overall, we improved on how to collaborate together as a team to create our app and write all the documents related to it.

*What we feel good about*

As a team, we feel good about the organization and readability of our requirements documents and use case models from each of our cycles, particularly cycle 3. In addition, we are proud of overcoming the bugs that we faced from cycle 1 to cycle 2, which allowed cycle 3 to be less stressful. We believe the communication throughout the cycles allowed us to work efficiently and check each other’s work. We had many meetings that allowed us to keep on schedule with what we agreed with and able to work through difficulties that we came across. In particular to the code, we feel good about the performance of our program’s runtime and the methods themselves. Of course, with any project, there is room for improvement, but with the time allotted and our heavy workloads outside of just this class, we are proud of what we were able to produce as the final product.

*Possible changes*

If there was more time, our team would’ve liked to explore a more attractive user interface using GUIs rather than having a console-based app. In addition, we ran into a bug that we were not able to fix and would have liked to have been able to form a more complete project. This bug occurred when trying to find one’s parents when one or more does not exist. Other possible changes to the code would be to make it even more readable and efficient. In some areas, there is repeated code and commented out test cases from past cycles that could be replaced with more appropriate code.