Reflection Report on Software Engineering

Team 14, Reach
Aamina Hussain
David Moroniti
Anika Peer
Deep Raj
Alan Scott

1 Changes in Response to Feedback

1.1 SRS and Hazard Analysis

There were several changes made to the SRS and hazard analysis as a result of feedback from Dr. Smith, our TAs, peers, and supervisors.

SRS:

- Peer feedback
  - Update the individual product use cases to be fully black-box use cases. We did this by only focusing on what the use case is, and not describing how it could be done. This ensures the design is not constrained in the SRS (see Issue 12, this commit and this commit for more details).
  - Improve NFR-1 to make it less ambiguous (see Issue 14 and this commit for more details).
  - Improve NFR-12 to make it less ambiguous (see Issue 15 and this commit for more details).
  - Added user characteristics to make the target users more clear (see Issue 13 and this commit for more details).
- Dr. Smith and supervisors - After the revision 0 demo, both parties had very useful feedback, which we implemented in many places throughout the app and documentation.
  - Add a map to be able to see exactly where the trial/study is being held (see Issue 138 and this commit for more details). This was recommended by our supervisors.

- Add a disclaimer/title to the profile creation page to make the purpose of the page more clear (see Issue 155 for more details). This was recommended by both Dr. Smith and our supervisors.
  - Add the ability to filter the studies/trials by distance (see Issue 178 and this commit for more details). This was recommended by both Dr. Smith and our supervisors.

- Other
  - Add FR-4, FR-5, FR-11 to the waiting room as they were not implemented.

Some other useful issues to give more insight into the changes we have made to the SRS documentation are provided below:

- Issue 251

Hazard Analysis:

- Peer feedback/TA feedback
  - Added some critical assumptions. This addresses feedback provided to us by our peers, and also feedback from the TA via the graded rubric (see Issue 35 and this commit for more details).
  - Added a definition of failure. This also addresses feedback provided to us by our peers and the TA (see Issue 36 and this commit for more details).
  - Add failure mode for email generation. This also addressed feedback provided to us by our peers and the TA (see Issue 37 and this commit for more details).
  - Make SR-6 less ambiguous by improving the wording of the requirement. This addresses feedback provided to use by our peers (see Issue 39 and this commit for more details).
  - Removed the notification subsystem as it is currently out of scope. This addresses feedback provided to us by our peers (see Issue 38 and this commit for more details).
  - Added traceability to FRs in the FMEA table (see Issue 40 and this commit for more details).

Some other useful issues to give more insight into the changes we have made to the Hazard Analysis documentation are provided below:

- Issue 250

1.2 Design and Design Documentation

There was very little feedback given for this particular section of the project, we generally performed well in this area. However, the changes made will be denoted below:

MG:

- Peer/TA Feedback.
  - Added all Functional and Non-Functional requirements to the traceability matrix in response to TA feedback. See this commit for more details.
  - Corrected some minor sentence structure issues in the document in response to TA feedback.

MIS:

- There was no feedback given for this deliverable, grammar was checked and minor flow changes were made.

System Design:

- There was no feedback given for this deliverable, grammar was checked and minor changes were made.

1.3 VnV Plan and Report

VnV Plan:

- Peer/TA Feedback.
  - Added missing abbreviations to the symbols section in response to peer review (see Issue 45 and this commit for more details).
  - Added addition characteristics for the typical user tester in response to peer review (see Issue 46 and this commit for more details).
  - Added reference to the grading rubric in response to peer review (see Issue 47 and this commit for more details).
  - Added clarification on what a greater than 7 score would mean in terms of testing in response to peer review (see Issue 48 and this commit for more details).
  - Added an open ended additional concerns question to the usability survey in response to peer feedback (see Issue 49 and this commit for more details).

  - Fixed some formatting issues including removing some whitespace from the document in response to TA feedback through the marking rubric (see Issue 255 and this commit for more details).
  - Improved clarity of NFR and FR tests in response to TA feedback through the marking rubric (see Issue 256 and this commit for more details).
  - Adding missing reference to hazard analysis in the relevant documents section in response to TA feedback through the marking rubric (see Issue 257 and this commit for more details).
  - Added more specific plan for nondynamic testing in response to TA feedback through marking rubric (see Issue 258 and this commit for more details).

VnV Report:

- TA Feedback (No peer review was received for this deliverable).
  - Expanded on the changes made due to testing, in response to received feedback on the marking rubric (see Issue 254 and this commit for more details).3
  - Added a screenshot of the failed linting in the maintainability section to provide convincing evidence of the test failing, in response to received feedback on the marking rubric (see Issue 253 and this commit for more details).

2 Design Iteration (LO11)

We approached the design, development, and implementation of this project using an iterative approach. We started off by coming up with the project requirements, and then splitting them into two groups: requirements that we must have and requirements that we would like to have time-permitting. Taking a look at the must-have requirements, we realized that our entire project relied on whether or not we could actually use the clinicaltrials.gov API. Since this was a huge priority, we decided for the proof of concept demo we would prove we could successfully utilize the external API.

Once we were able to verify the API worked, we officially started the development phase of the project. We worked on implementing the must-have requirements to the best of our abilities. We did this by creating tasks (using issues in GitHub) that we needed to get done in order to fully implement each requirement. For each individual task, we made the necessary code changes, did some minor testing locally, made further based on the results of the tests, merged the changes, and closed the issue for the task. In this case, we were iteratively developing on a smaller scale (on a task-by-task basis).

Once we completed implementing all these smaller tasks in order to fully implement the must-have requirements, we deployed the app and conducted some more thorough and integrated testing. This testing also included conducting a usability test with multiple users; one of the users was our supervisor. Throughout testing, we made note of things that did not work, needed improvement, or was redundant. Taking these notes, along with user feedback we received from the usability survey and our supervisors, we came up with some more tasks that we needed to complete in order to reflect our notes and feedback. This was a large-scale development iteration (of the overall integrated application).

We then repeated the process once more by making necessary changes for each task, deploying the app, and once again fully testing the overall application. Gathering our last round of feedback, we made the necessary changes and deployed the app once more before our final demo. This was where we stopped for now, but in order to keep improving our app and dealing with any bugs that we may discover or suggestions from our supervisors, we would continuously repeat this iterative process.

3 Design Decisions (LO12)

Due to time limit, we ended up deciding to not complete/implement some of the less important tasks/requirements. Thankfully, we had accounted for this time constraint at the beginning of the project when we were creating the development plan. We had already split the requirements into two groups: must-have requirements and nice-to-have requirements. This allowed us to prioritize the must-have requirements due to the time constraint, and put the nice-to-have requirements in the waiting room.

We were also greatly constrained by the fact that we were using the clinicaltrails.gov API. Fetching and filtering trials were forced to be done based on the restrictions imposed by the API. In fact, midway through the project, the external API was updated. Since our project was extremely reliant on this API, we were forced to refactor a lot of our back-end code.

We also had some constraints when testing the application. Since we were not able to have some actual target users (users with certain conditions searching for studies) test the application, we had to rely on feedback provided by our supervisors and we were forced to make assumptions of what might be best for them or what they might want included in the application.

4 Economic Considerations (LO23)

There is definitely a market for our product. Our project REACH is based on clinical trials and research studies. This means we are targeting users such as the many people that want access to alternate forms of treatment because they do not benefit from conventional or readily available treatment options. REACH is also targeted towards researchers who are unable to find people to take part in their studies. REACH bridges the gap to form the connection between researchers, patients, and healthcare providers in a way that existing alternatives can't.

The existing option in this case would be a website like clinicaltrails.gov; however, as stated, clinicaltrails.gov does not have some of the key functionality that REACH offers which would greatly improve the user experience. REACH is designed to be more user friendly, so that users with all types of abilities are easily able to interact with and understand the application. REACH even uses different phrases or workflow depending on if the user is a clinician or not, so that it is customized to best suit the abilities of the user. We worked to improve the usability and discoverability of the app through rigorous testing and conducting multiple usability tests. As a result, REACH is easier to navigate for a wide range of users, while existing implementations are targeted more towards researchers and less towards clinicians/patients. The main value-added feature of REACH are the advanced condition fields, which allow users to be more accurately matched with studies based on how they fill out the advanced fields. This way users do not have to look through so many unrelated studies. REACH also allows users to create an account so that they can save the information they filled out in the advanced condition fields as a profile. This makes it easier for them to conduct searches as they do not have to fill out all the fields/filters every time they use the site, which is the case for existing implementations. Instead, they can just login to REACH, and their information used for filtering will be saved so they can conduct searches swiftly. Logging in also allows users to bookmark studies to come back to later.

Since REACH is not a traditional product, we aren't able to sell it in units to make money. It is an open source project. We could go about attracting users by marketing our product online through different social media platforms. We could also market our product at relevant locations. In our case, the most relevant locations would be hospitals, clinics, research centres, laboratories, and other medical centres. As stated above, there are countless potential users that currently exists. The potential users are any and all people that want to take part in a clinical trial or research study (for health reasons, volunteering, etc.), as well as all clinicians that want to look for studies on behalf of their patients.

5 Reflection on Project Management (LO24)

5.1 How Does Your Project Management Compare to Your Development Plan

5.1.1 Team Meeting Plan

With respect to the team meeting plan, we consistently met at least once a week, like stated in our development plan, to discuss our status, any issues, and what tasks we need to complete by the next meeting. We also met regularly with our supervisors to share our progress on the project and receive their input. However, we did not meet them on a biweekly basis as stated in the development plan; instead, we met with them whenever they were available, which was about every three weeks.

5.1.2 Team Communication Plan

The team communication plan outlined in the development was followed exactly as stated. We used Git issues to keep track of any formal tasks needed to be completed. Teams was used to host virtual meetings, as well as update the team about minor issues or concerns. We also used Google Calendar to keep track of deliverable deadlines and team/supervisor meeting times.

5.1.3 Team Member Roles

We did not have clearly designated member roles like we stated in the development plan. However, most of the roles were still loosely followed to some degree. Since every team member was flexible and we all had multiple skills, we weren't limited to just the roles that were designated to us in the development plan. We all took on different roles based on the current time constraint and what tasks were available for us to work on.

5.1.4 Workflow Plan

We followed our workflow plan exactly as it was outlined in the development plan. First, we discussed relevant tasks that needed to be completed in the following week and created issues for them on GitHub. Next, we assigned tickets to the team based on each member's skills and availability. We used the GitHub project board to keep track of the status of the issue (whether the issue is currently being worked on, in testing, completed, or not yet started). We used separate feature branches to work on changes related to each issue. After testing, we created a pull request and merged once all automated tests have passed.

5.1.5 Technology

We used all the technology we planned on using in the development plan. The only exception is we ended up not using Jest as a front-end unit testing framework. Due to the time constraint, as well as constant changes to the overall design of the UI and the new inputs/requests from our supervisors, we decided implementing unit tests for the front-end would hinder development with very little payoff. Instead, we tested the front-end manually.

5.2 What Went Well?

In terms of processes and technology, what went well was our consistency with meetings and communication. We discussed tasks thoroughly, did not hesitate to bring up concerns/blockers, and dealt with any issues as soon as possible. We were all flexible and did not hesitate to take on tasks as soon as we were available to. No one on the team refused to do work just because they might have already done "their share". This allowed us to get work done quickly and more efficiently, so even if there was a short deadline, we were able to get our work done. This also allowed us to easily deal with any suggestions or concerns our supervisors had since our consistent and organized workflow made it so that we had time to spare.

In terms of technology, we all had experience using Git and GitHub in the past. We utilized our knowledge of branching, GitHub issues, GitHub project board, pull requests, and more for this project. This allowed us to keep things organized, keep track of the work we are doing, and minimize conflicts, making the workflow smoother.

Each member of our team had experience using different technologies, which meant we were able to assist and teach each other. Combining our skills allowed us to be a well-rounded team and successfully carry out this project.

5.3 What Went Wrong?

In some cases, our team members had differing opinions or ideas about what the expected outcome of a task was supposed to be. This is particularly true for some of the front end UI work. This is because we did not create and finalize designs for every part of the user interface (using rough sketches or Figma). For the parts of the UI where we did not finalize a design, we all visualized what it would look like differently. This would lead to some confusion about what was the expected outcome from the completion of that task. This consequently hindered development since we had to take some extra time to ask questions and clear up confusions. In the worst case, we would even waste time implementing something that was not agreed upon, and would need to refactor it after discussing and coming to an agreement.

5.4 What Would you Do Differently Next Time?

For a future project, we would ensure that our GitHub issues were more detailed. As explained in section 5.3, sometimes there was some confusion around what we needed to do or complete for certain tasks. Writing more detailed issues which outline the expected outcome necessary in order to close the issue and complete the task would essentially solve that problem. This would ensure that we are all on the page, leading to less confusion and less wasted time.

6 Reflection on Capstone

6.1 Which Courses Were Relevant

The following course provided beneficial experience towards the undertaking of our capstone project:

- SFWRENG 3XA3 Software Project Management - Provided experience in coding development, but more importantly in managing the planning and documentation of a project from start to finish.
- SFWRENG 3S03 Software Testing - Provided knowledge of software testing strategies, such as static testing, unit testing, and code reviews.
- SFWRENG 3A04 Software Design III - Provided project development experience, specifically in module design and documentation.
- SFWRENG 3DB3 Databases - Provided knowledge on the operation and interaction with databases.
- SFWRENG 2XB3 Software Engineering Practice and Principles - Provided experience in software design, as well as providing knowledge in measuring system metrics.
- SFWRENG 2AA4 Intro to Software Development - Provided our first experience towards software system development, including data structures, design principles, and coding practices.

6.2 Knowledge/Skills Outside of Courses

David - I learned several things during this capstone that I had not learned through previous courses (or even during my coops). First, I had little to no experience with React, and during the development of REACH, I gained a significant amount of experience with it. Additionally, I had never previously used typescript, and so this project allowed me to learn a new programming language. Finally, none of my previous courses touched upon the cloud, and how one can deploy applications to the cloud - and this project enabled me to learn and gain experience in this field, by deploying and managing the app in google cloud.

Deep - Both of the web frameworks that we used in developing REACH were new to me. I had never learned React or Django, and so this project was a great learning experience, not only because of learning the web frameworks themselves, but also by improving my skills in backend and frontend web development in general, which was not touched upon in previous courses. I also had never used typescript before, so this project helped me learn a new programming language.

Alan - The capstone course allowed me to learn many new things, including a new backend web framework, a useful and commonly used data analysis/manipulation python library, and api development/interaction. While I had some experience with the backend web framework Flask, I had never used Django. Furthermore, in the work that I did on the trial fetching and filtering modules, I was able to learn, and develop my skills in Pandas, a very powerful python library for data manipulation. Finally, I improved my skills and understanding of API's and API development in general, by both working on our internal API's, and by interacting with the external clinicaltrials.gov API.

Aamina - The main thing that was new to me was the backend web framework that we used, Django. Additionally, while I did have some experience with Python through previous courses, I had never used it in a real-world setting, on a project of this scale. This was important since Python is one of the most commonly used languages, and I learned a lot of subtle complexities that I did not previously know about.

Anika - There were two main things that this capstone course allowed me to learn which I did not previously learn in any of my past courses or coops. First, I had never learned/used typescript or React and considering this was one of the most integral parts of our app, I had to learn it fairly well to be successful in developing the frontend. Second, although I did have some experience working with the cloud (from previous coops), I had never used google cloud platform, and there was a surprising amount of differences between GCP and the platforms I have previously used, so I am glad I was able to learn about it.