Reflection Report on SFWRENG 4G06

Team #12, Team 12.0
Kanugalawattage, Anton
Subedi, Dipendra
Rizkalla, Youssef
Leung, Tamas
Zhao, Zhiming

1 Changes in Response to Feedback

1.1 SRS and Hazard Analysis

The document content of the SRS was revised in order to address missing sections as advised by the TAs and other teams. Primarily, a naming conventions and terminology table was added to the beginning of the document with additional terms that may be unfamiliar to users, such as names of specific technologies or video game-specific terminology. When appropriate, these terms were cited in the bibliography with the citation present whenever the term is mentioned. Additionally, all sections of the document were overhauled to use paragraphs, with bullet points used more sparingly, when appropriate. This allowed the team to fill in additional details and improve the document's flow, as well as give the document a more professional outlook. Moreover, some sections that were missing were added to the document. In particular, a context diagram was added to better inform the reader of the scope of the product. Similarly, a use case diagram was added in place of the bullet points to more formally and rigorously introduce the product boundary as follow up to feedback from another team. Additionally, a section was added to discuss Off-the-Shelf solutions and to discuss our platform's advantages and disadvantages over them, as well as the new problems that our platform may have to deal with. This includes more "big thinking" related to the user, their needs and how our platform will be used to fulfill those needs. Moreover, a traceability matrix was added comparing all FRs and NFRs rather than a grouped list of requirements. Explanations were also added to the phase-in plan in addition to the table with deadlines in order to justify the functional requirements at each priority level. Finally, an explanation for each state and action in the state machine with added in order to inform the reader of its context.

After completing the design, user testing and implementation, the scope of the project differed in some parts. Consequently, some requirements were changed, removed or added to reflect that. Primarily, requirements that initially over-specified our systems were reworded to avoid influencing the design. For instance, there were requirements which stated the exact number of rounds or players that the system should support. Since these were infeasible to support in a robust system, they were changed from general to an arbitrary number or a boundary was introduced in appropriate cases. Likewise, the initial SRS over-specified the problem of dealing with malicious code, as it specified that it should be detected. As pointed out by course staff, this is infeasible due to many reasons. Thus, those requirements were changed and new security requirements were introduced that ensured that user code would not be able to access the server and that it would be time and memory constrained but did not over-specify how the implementation would accomplish that. Likewise, the constraints of the project were revised to better reflect the end result of the project as well as the course constraints. For example, the budget constraint was changed to $750 to better reflect the budget limit set by the course. Moreover, rationales, fit criteria, and explanations for priority were added to requirements as appropriate to better justify them in cases where it may have been unclear.

The Hazard Analysis was completely rewritten using LaTeX (rather than MarkDown) in order to be more consistent with our other documentation and to provide a PDF rendering. Spellcheck and grammar check software was ran on the document to fix errors, in addition to manual inspection by team members. The scope of the Hazard Analysis was extended to mention that our system will still be responsible and will attempt to minimize or workaround hazards that are not directly controlled by the team, such as external database services. Additional failure causes were identified and addressed in the Failure Modes table, such as additional cases for server outages. The solutions were also revised and improved, such as in the server scaling plan in case of outages. Additionally, requirements were revised following the implementation and design of the system. Since our platform would no longer handle sensitive user data due to using an external auth service, requirements relating to storing user data were removed and hazards that dealt with external services were expanded upon instead. Finally, the scope of the analysis was extended to address the psychological elements of the platform and to consider the well-being of the user-base. Thus, additional failures and actions were added to cover that component as well.

1.2 Design and Design Documentation

The main system design document originally had a section on communication protocols but did not fully explain why the specified protocols were chosen. Consequently, explanations were added to justify the usage of HTTPS and WebSockets and to define the cases in which each was appropriate. In particular, the differences between HTTP and HTTPS were discussed and an explanation was added for why HTTPS was preferable. Moreover, all figures in the documents were updated to use PDF renderings instead of PNG, which allows for higher-quality figures at any resolution size. To accompany the figures, paragraphs were added as appropriate to describe the relevance of the figure to the overall design. Additionally, the team ensured to link to all relevant documentation at the start of every design document. In the Module Guide, the unlikely changes were rewritten to address TA feedback. In particular, the original UC2 and UC3 were removed as they were likely to change or did not add much information to our design. They were replaced with new changes that better reflected fixed design changes in our system and were more integral to our system. Moreover, the formatting for all modules in the Module Interface Specification was revised. In particular, the team fixed the formatting for longer and capitalized variable names in which the LaTeX rendering sometimes resulted in strange kerning that is difficult to read. Moreover, the quotation marks were fixed to result in the correct rendering for opening and closing quotations. Finally, modules that differed too much from the resulting implementation were slightly revised in order to create a clearer mapping between the code and modules. This usually happened in cases where the implementation changed due to user testing (i.e. copying the code versus copying the link and adding Python as a language).

Side Note: No other team created GitHub issues for our Design Documentation, so there was less to address in comparison to other documents.

1.3 VnV Plan and Report

Primarily, the formatting and presentation of the VnV plan were revised. All sections in the VnV plan were spelling and grammar checked by the team members. Moreover, Company branding and styling were also corrected when appropriate (i.e. GitHub). Finally, all relevant documentation was linked at the beginning of the document. In regard to the document's content, the plan section was extended with a justification for the roles of each team member, as well as the addition of backup plans in case a testing area is more (or less) labour-intensive than expected. Additionally, all tests were changed to be made consistent with the VnV Report. These changes were largely based on the reflection done in the VnV Report which addresses the initial changes between the VnV Plan and the VnV Report. Moreover, tests which relied on a specific waiting time were changed to wait until a page was loaded to reduce flakiness. To better demonstrate dependency among tests, a new section (5.3) was added alongside a dependency graph to describe such dependencies. Likewise, in an effort to make the document easier to follow, a traceability matrix was added in addition to the current tabular format for describing the traceability between tests and requirements. After the completion of the design documentation and the implementation, a unit test summary was added to the VnV plan. The section describes our philosophy regarding unit testing, the modules which were to be covered and points to the unit testing code. Finally, a section was added to describe the user acceptance testing effort that was initially missing from the VnV plan. To accompany it, the Usability Survey Questions were added to the document's appendix.

Like the VnV plan, the spelling and grammar in the VnV plan was checked and corrected. Additionally, the figure captions were updated to be more succinct. A new section (9) was added to describe possible next steps in the testing effort for the overall system. This section discussed potential longer term studies regarding the effectiveness of the system to the demographic's learning process as well as tests that could be performed with a higher budget and more time. As suggested by other teams, the document linked to parts of the VnV plan which may be relevant (such as the description for the test cases) to make the document easier to follow. Additionally, the document elaborated on features that could be added and tested resulting from user feedback that were initially not implemented due to time constraints.

Someone added this here but I think it is not relevant to this section, might use it for another section - Youssef

After performing the usability test, we received information that resulted in changes made to the product. The first change we made was adding support for the Python language. This was because we saw a great amount of interest in Python when comparing additional languages we could add for support on CodeChamp. The second change we made was adding shareable links when inviting friends to join a user's lobby. With some A/B testing, we found that users preferred to copy and paste links that take users to the lobby directly, versus the entry point of typing in a lobby code.

2 Design Iteration (LO11)

The team started out with designing features that would be integral to a proof of concept. Initially, this involved implementing high priority functional requirements which were unlikely to change, as identified in the SRS. This helped identify the risks associated with the development of the project as well as ensure that time would not be wasted in the early stages, as the features developed in this stage could be re-used or evolved for future iterations. This involved implementing aspects of the platform such as the code compilation capabilities and the storage / retrieval of core data such as problem data. After this stage, it became more clear to the team which aspects of the project were important to focus on, especially in relation to non-functional qualities, based on the feedback from the instructors and other teams. Thus, the second iteration of the design moved towards a minimum viable product, in which the team developed qualities which made the platform unique and appealing to potential users. At this stage, the product was essentially considered complete with features such as lobbies, profiles, match histories and leader-boards, which were all features inspired from the gaming industry aimed to provide a fresh experience for users. Additionally, at this stage, the design was adapted to accommodate for non-functional requirements that were not accounted for in the proof of concept stage due to time-constraints. This included things like containerizing user code for security considerations as well as improving the user interface. Finally, having a minimum viable product meant that the team could perform user acceptance testing to consider user feedback and integrate that into the final design. Based on the user feedback, as documented in the VnV report, some features were adjusted or added. This included things such as adding support for more popular languages and making UI/UX decisions based on user data rather than intuition or assumptions.

3 Design Decisions (LO12)

The design of the authentication scheme was a major design decision in the design of CodeChamp. The first approach we considered was implementing our own system. The main advantage of this is that there is no reliance on a third party service, which can improve the performance of the application as well as its reliability in case of downtime of the third party service. We opted to use an external service, Google Auth, as it is a widely accepted security solution, with essentially zero historical downtime. Furthermore, the vast majority of users will already have an account, reducing the sign-up process on CodeChamp to a single click. Finally, this prevents possible errors in our security integration which can result in compromised accounts and passwords, as with this approach we never process or store the user's password using our own service. This was also to work around the time constraints, as we had a limited timeframe for implementing the system. Thus, Google authentication provided a convenience to the developers and the users alike. Due to the ubiquitous nature of Google with anything related to the web, it is a safe assumption that users would have Google accounts, and even if they did not, they could create one easily through the CodeChamp interface.

Another design decision was to use Docker containers for the purpose of isolating code from the main server. As we are a platform that processes user submission of code, there is a chance that the code could be malicious, so we utilized containerization to avoid catastrophic results. Since we also needed to introduce limits on the code such as memory and time limits, containerization also provided a framework for implementing these features and constraints. Docker was chosen as it provided performance and security. Since caching the images for each language on the server is possible, the server could execute the code quickly enough by re-using the image every time code was submitted. Moreover, we could utilize Docker's features to limit the memory used by the user's code as well as to disable network access, which are both essential to securing our system.

We also considered different ways of structuring the whole system. The current design consists of a completely separate back-end and front-end, implemented using Node.js and Angular respectively. An alternative design could use a monolith application which serves the HTML from the server. For instance, many frameworks such as Razor Pages use the MVC pattern to achieve this design. The main advantage this presents is an easier implementation, as we can re-use types across the system and not spend time implementing abstraction layers for communication protocols. Another advantage is that a multi-page application will have better search engine optimization. However, we opted for this design as the usage of a Single Page Application framework for the front-end allows for better performance, as all rendering can be done on the client. Furthermore, it allows for a separation of concerns by hiding the implementation logic on the server-side from the display and layout logic on the client-side. This also reaps other benefits, as it allows developers outside of the CodeChamp team to develop tools which interact with our backend APIs, as they are not reliant on a specific client-side implementation and can be communicated with using HTTP and Web

Socket protocols. Finally, it is safe to assume that our application will not benefit much from better search engine optimization, as most pages of the client-side pertain to temporary game information and interactions, rather than information typically picked up by web crawlers.

Regarding the game itself, different lobby systems were considered when designing CodeChamp. Initially, the idea was to hide the concept of a lobby from the players, and instead match-make them and place them in a game. We did not choose this system as it relies on the existence of a skill-based matchmaking system. However, skilled based matchmaking requires a large player-base in order to have meaningful results, as wait time for a game will be too large with a small player-base. Thus, we opted for this system as it allows users to invite friends and start a lobby whenever they wanted, reducing the friction and wait-time in the early stages of CodeChamp. The upside to implementing a skill based matchmaking system is that it would increase the competitiveness as well as lower the barrier to entry, which are two traits of a successful game. The system would have to track statistics such as win rate, types of problems solved and the difficulty of the problems solved for each user in order to evaluate a skill rating for them. Furthermore, the developers would have to design a match-making algorithm to group similarly skilled players in a lobby. With a growing player-base and with more time, this would be part of the next steps for the CodeChamp platform.

4 Economic Considerations (LO23)

There exists a market for this product as it combines gaming and interview preparation. The age distribution for LeetCode users and gamers overlap with majority of the people in the young adult age range (18 - 34). Looking at existing platforms such as LeetCode, our research shows that there are about 18 million visits to the site per month. This can be a rough estimate of the potential userbase for our application and showcases that a large demand in this space is present. As the genre of gaming we aim for is Battle Royale, we believe that a lot of interest can be garnered as it is a very popular game mode. One potential way to market for our product would involve promoting the platform through word of mouth in Computer Science forums such as Reddit and job searching platforms like LinkedIn. We can also host competitions on the platform and can partner with local companies to have contestants compete for job opportunities such as internships. This is a viable strategy as this would effectively be a live coding interview, where users compete against each other for a job.

When considering costs for the product, the team will only have to pay for server hosting which is variable on the number of users that we support, as well as any costs for marketing. If we project about 1000 monthly active users to start, the server hosting costs would be about $40 a month using our current infrastructure. We can also market CodeChamp as a free-to-play game, meaning that the base product itself would be free. This means allowing users to use the basic features of the platform at no cost, which will allow us to attract a larger user-base to the platform at an earlier stage. To monetize the product, we could run advertisements and have a subscription based model for extra features, similar to how free-to-play games monetize in the gaming industry. Some examples of extra features could be pair programming, voice call, practising tailored problem sets, and coding environment customisation. The subscription itself could be billed monthly for a small amount such as $4.99. Additionally, the platform could also have varying subscription tiers which offer different features at each tier, allowing players of different commitment levels and needs options as well.

5 Reflection on Project Management (LO24)

5.1 How Does Your Project Management Compare to Your Development Plan?

Throughout the duration of the project, we followed our development plan very effectively. Our team had meetings every week on Discord and we also had additional meetings when deadlines approached, to ensure we finished work on time. To manage the project, we utilized GitHub to keep track of issues and features we were working on. The GitHub project board allowed us to easily keep track of work to be done, work in progress, and work that was finished. We also utilized the merge request service provided by GitHub to perform code reviews on each merge request before merging the code into the main branch. We all fulfilled our roles with respect to what was laid out in the development plan, but we also helped each other out when needed. The focus areas we presented in the development plan stayed the same throughout the duration of the project. One area in which the project differed from the Development Plan is the technologies used. In particular, the external services for components such as authorization and containerization. The original plan was under the assumption that CodeChamp would have its own authorization system, so it was going to use Auth0 as an authorization service. However, the actual system ended up using Google Auth, for the reasons mentioned in Section 3. As for containerization, the system ended up using Docker while the original development plan failed to account for this in the first place. This is because the team was more focused on functional qualities when drafting the development plan rather than non-functional qualities such as security.

5.2 What Went Well?

We divided the work amongst the team very well following the team member roles, while also having team members work on issues and feature outside of their focus area to learn new technologies and increase productivity. The team tracked the progress of issues and features with GitHub project board, which displayed the visibility of everyone's status to the rest of the team. This also allowed us to see what we finished and what needs to be worked on. It was useful to have a weekly meeting even during "down-time" in which there were no course deadlines, since it allowed us to space out the work more effectively and reduce crunch-time.

5.3 What Went Wrong?

While we followed the development plan for our meetings, sometimes these meetings would last much longer than scheduled, and we would not be as productive as we could have been. This was likely because some meetings lacked an agenda, so it was easy for the discussion to go off-topic. In terms of technology, we also did not use GitHub's code review functionality as rigorously as we should have. Some code reviews were rushed when submitted near a project deadline, leading to technical debt and issues arising later, in which someone would have to clean up the code produced in an earlier code submission.

5.4 What Would you Do Differently Next Time?

Something that we could do differently would be to perform code reviews more thoroughly. This could be done through code reviews as a group for large features. This would help with maintainability and visibility of our code, to ensure we are pushing the best code to the main branch. We can also improve meetings by having an agenda beforehand, to ensure a productive meeting. This agenda would be reviewed by each team member before the meeting, so that everyone would be on the same page. Hosting meetings that are smaller but more frequent would also enable focusing on smaller features. While our development plan was a good start, it did not account much for non-functional qualities, which can completely re-shape the design of the entire application. In the future, it would be useful to consider the constraints and limitations of our development team and project in order to identify the tooling we need at an earlier stage.