Reflection Report (1-2 pages)

Our group worked under the Agile Process, which has several strengths and weaknesses.

One clear strength of Agile was the use of user stories. User stories forced us to break large features (such as creating and joining study sessions) into manageable tasks. This made the workload less overwhelming for developers and gave each sprint a clear goal. Another major strength was the flexibility of the Agile process. In theory, Agile is designed to cater to changing requirements and shifting user needs. In practice, we found this valuable, since our project timeline was tight and our design choices often changed as we encountered different scenarios. Iterative sprints helped us improve and reflect on our early decisions, which helped the product take shape better. Finally, Agile emphasized teamwork and transparency. Daily standups and retrospectives provided dedicated time slots to share progress, blockers, and reflections. Standups encouraged each developer to be honest about what we were struggling with, which helped avoid delays, and retrospectives let us recognize our successes and failures. Even though we often worked independently on our tasks, we had checkpoints to ensure we were on the same page.

A key weakness we experienced in Agile was the demand for frequent meetings. Typically, Agile recommends daily standups, sprint planning, and retrospective meetings across 1 to 4-week sprints. However, our short project timeline made these meetings pile up and sometimes feel overwhelming, forcing us to fit a sprint into less than a day. Another low point we found is how Agile's flexibility can become a liability. The constant shifting of requirements between all of us sometimes resulted in neglected areas like testing or documentation. In theory, this process balances these across sprints, but our constraints forced us to prioritize development over maintenance. This spilled over into our communication at times, and some developers found themselves experiencing scope creep with some stories.

When we started this project, we expected that the use of AI tools would make this project very easy and that we'll be able to finish it in a day. But the reality couldn't be any more different, as it took us more than 3 days to fully implement and debug everything. We were a little lost at the beginning and weren't sure where to start since we never fully gathered our requirements and project architecture. It was one step at a time. We expected that it'd be very easy to split all the work among ourselves. But since the project itself wasn't that lengthy, a lot of the user stories that we split among ourselves overlapped with each other, which required one person to finish their task before the other person could start working, resulting in delays.

Moving on to the actual feature implementation, we wanted to implement a simple username and password feature for students. The idea was that when a student creates a profile, their login info would be saved in a JSON file. On execution, the program would check whether the person already has an account. If yes, they could log in and access their profile, if not, they'd register. This would have allowed people to log out and log back in with different accounts, so we could test accepting or declining invitations from multiple students more realistically. But right after our Sprint 1, we realized that this outcome would take much more time than we'd

hoped and would make our program very complex. After thorough evaluation, we decided not to go forward with it. Other features, like confirming and suggesting sessions, were much easier to implement than we'd thought.

The use of AI tools influenced just about every aspect of our program. We started off by using ChatGPT to write our user stories. This influenced what we deemed was a requirement for our program. That led us to design our code as we felt was most appropriate to fit these requirements. In order to fulfill the user stories but still maintain our code's readability, we followed an MVC design pattern to organize our source files. When it came to producing code, ChatGPT was best for small and specific sections of code. A specific example of this is when I asked it to produce my starter menu options, which were just a couple of print statements. This was helpful since it saved me a bit of time. The downside was when we asked for more logic-based code. We either had to be really specific or heavily correct the code. In other words, ChatGPT was good at generating starter code and small, non-logic-based code. The use of AI also came in handy when we needed to test our program. We used it to generate student data for our tests, but also to store it in the program as our default students. The AI tools saved us a lot of time by reducing the amount of time we would've spent generating the data by hand. While the tool was a great help, it still required a lot of revising, so the code wasn't overly complicated, but also so we understood what the code did and how to modify it later if needed.