SRS Prompt Usage:

You are taking on the role of Senior Software Engineer for a small project. I need a formal SRS documentation sheet for a small software project. The project is a “Study Buddy” student study session scheduling app for Clemson students: To be coded in C++, front-end to back-end.

- This needs to be a simple implementation, as it is just for a class. Don’t use anything exotic to enhance performance or sum.

Key requirements:
1. Students can create a profile and enter classes they are enrolled in for that term, as well as enter their weekly availability for study sessions.
2. They can search for and schedule study sessions with classmates. The search should be based on the exact course number they are trying to set up a study session for, as well as their availability and the potential other student’s availability.
- Features Add/Remove availability(part of profile), suggest study group matches(based on course they are studying for), and confirm/cancel meetings with other users.
4. Implementation: Useable in the command line
No changes allowed after approval so we need to make it good before start implementing
Please consider:
- Error handling
- Edge cases
- Best practices for C++
- Please follow common SRS documentations sheet format that is commonly accepted

Output: SRS generation in Artifacts

PSEUDOCODE Prompting:

Objective: Write Pseudocode for Study Buddy app based off of SRS documentation generated

It is crucial to maintain utmost accuracy. Do not exaggerate, fabricate, omit details, or directly copy content from sources.

Verify the accuracy of any factual information you provide. Avoid spreading misinformation.

Write pseudocode for the Study Buddy app based off of the SRS Documentation that was generated earlier.

Please respond in English

I want you to write in a technical writing style. This style will be precise, clear, and objective, focused on conveying technical or specialized information in an accessible way. It involves the use of clear language, structured format, and logical organization to explain complex concepts, procedures, or instructions. The writing should be devoid of any personal opinion or unnecessary jargon, aiming for maximum clarity and efficiency in communication.

Output: Pseudocode in Artifacts

Code Prompting:

You are taking the role of a Senior Software Engineer. Based on the Software Requirements Specification (SRS) document and pseudocode previously written for the “Study Buddy” project (a C++17 CLI app for Clemson students to create profiles, record courses and availability, match classmates, and schedule study sessions), generate the full working project in standard C++17.
Requirements:

1. Implement the project according to the SRS and pseudocode.
2. Use only the C++ standard library (no third-party dependencies).
3. Organize the project with a clear directory structure:

- src/ → all .cpp source files
- include/ → all .h headers
- data/ → sample input/output files (e.g., profiles, schedules)
- Makefile at root with build, run, and clean targets
- README.md with instructions to compile and run
- Ensure that all of the functionalities work, so if I want to add a course, I am able to successfully do that, and make sure you put directions for each command on a successful attempt in the read me, so it's user-friendly.

4. Ensure the project builds successfully with make.
5. After generating the code, compress everything into a .zip file and provide it for download.

Deliverable:

- A .zip file containing the entire Study Buddy project with code, sample data, Makefile, and documentation.

Testing Prompting:

You are a QA/Test Engineer. We have completed a small C++17 CLI project called Study Buddy (student profiles, courses, availability, scheduling, confirming study sessions). The development followed the Waterfall model, and I am now in the Testing Phase.

Your task:

1. Generate a Test Plan that covers the core functional requirements only:

- Create a profile
- Add/list courses
- Add/remove availability
- Suggest matches (based on course + overlapping availability)
- Schedule sessions
- Confirm sessions
- Handle invalid/duplicate inputs

2. Generate 15-20 unit tests in the code file so that it is testable. Output the result after the unit tests finish to a CSV file, saying if passed or failed.

Constraints:

- Keep it concise and professional (like a QA submission document).

- Results should show PASSED for all working cases and clearly indicate errors for invalid input cases.
- Assume this is a final Waterfall deliverable (no requirement changes)

Outputs are in our Artifacts folder as SRS, Pseudocode, code prompting and test plans.

Through utilizing ChatGPT to help write a formal Software Requirements Specification (SRS), we found that in order to obtain the desired results with the SRS, we needed to be specific in our prompts. For instance, we began with a well defined prompt that outlined our desired app with key requirements that were necessary for the project. In doing so, ChatGPT outlined a SRS that was mostly correct. We found that we needed to specify certain aspects of the requirements more clearly in order to obtain a more refined SRS. After doing so, we finalized our SRS, and could begin prompting ChatGPT to assist us with our pseudocode and our implementation.

Within the same chat, we then asked ChatGPT to assist us with our pseudocode which would allow us to get good code in the long run. One of the major parts of our pseudocode prompt was to ensure that ChatGPT maintained accuracy and did so in a way that aligned with the SRS to avoid deviating from our requirements. Following our prompt to maintain accuracy and avoid misinformation, we made sure to include that the pseudocode should be clear and concise in its representation of our code in order to ensure that it is easy to understand while maintaining the ability to convey technical information. Additionally, we continued to be specific in our prompts and ensured that the pseudocode would be written in English by including that requirement in the prompt. We found that the best way to ensure that what ChatGPT responded with was correct and usable, that we needed to be clear and specific in the way we prompted it to assist us. We then moved to code generation and testing plans we aimed to again be very clear in our prompts to the system and I believe that contributed to our success. We ensured to leave no gaps in our prompts and focused on the idea that ChatGPT should be clear in its responses and avoid using any bias or opinion in its responses which is a large setback with LLM's. This allowed us to gain good responses and effectively use AI in the code aspect of our project by taking more time on the front end with requirements documentation and pseudocode. Overall, being clear with our prompts led ChatGPT to provide us with the best assistance, because if we left our prompts too open-ended, we would be left with responses that were too lengthy or even incorrect from what we needed. This project allowed us to gain good knowledge and experience in the powerful tool of AI while also understanding its limitations.