**Initial Software Orchestration Prompt**

This thread will be used to complete one of 9 experiments for a Software Orchestration paper. The experiment will evaluate chatGPT’s response to the initial prompt designed to initiate a (human, AI) collaborative process to generate an application. The 9 experiments will form a matrix of application size (class A, B, C) (Y Axis) to Detail or complexity of the initial prompt (low, medium, and high detail) on the (X axis). The assessment will be the response to the initial response during the reflection period. Please respond OK if the instructions are clear before the first experiment begins. We will be doing medium application with high detail in this thread Note the Software orchestration process and abstract are provided below:

The Software Orchestration process is delineated as follows:

1. **Initial Prompt**: The software architect engages in "prompt engineering," crafting prompts that accurately convey the design specifications to the AI collaborator. These prompts encapsulate the overarching design, UI/UX stipulations, database schema, and other pertinent specifications. A detailed and meticulous prompt at this juncture invariably augments the quality of the outcomes. An example initial prompt is presented in section 2.3.
2. **Reflection Period**: This phase fosters a dialogue between the software architect and the AI. The AI provides feedback, seeks clarifications, and identifies areas necessitating further refinement.
3. **Infrastructure Implementation**: This phase is pivotal in establishing the application development environment, typically comprising on-premises or cloud-based virtual machines, primed for development and testing.
4. **Application Implementation**: The AI embarks on the development journey, methodically crafting each class, commencing with utility classes like database managers. This phase includes the concurrent generation of code, UML diagrams, and ancillary documentation.
5. **Debugging**: Each developed class undergoes immediate testing, ensuring incremental and iterative development. The cyclical nature of the implementation and debugging phases ensures a robust and error-free software product.
6. **Post-Development Review**: A thorough review is conducted, focusing on security, performance, and scalability considerations.
7. **Deployment**: AI-crafted prompts facilitate the seamless deployment process, ensuring a smooth transition of the software to its intended environment.
8. **Maintenance**: The dynamism of the Software Orchestration process extends to the maintenance phase, enabling swift and efficient system modifications, as evidenced in our case study.

Software orchestration represents the natural evolution beyond traditional Software Engineering and Development, encompassing the seamless integration of AI tools such as ChatGPT into the software development process. This experiment evaluates the concept of Software Orchestration, a term derived from a fusion of two fundamental notions: the symphony of a musical concert, where a conductor guides the orchestra to perform a harmonious composition, and the computer-related definition of orchestration, which involves automated processes for configuring, coordinating, or managing computer systems, data, or software.

In this context, the "orchestra" comprises AI Neural Networks, specifically large language models, while the "conductor" takes the form of a Software Engineer or "Conductor," orchestrating the AI to craft, refine, and execute software compositions. Software Orchestration transcends mere automation; it represents a collaborative approach to designing, developing, validating, and documenting software compositions, merging human expertise with AI capabilities for enhanced software craftsmanship. This experiment studies the Software Orchestration approach comparing it to a traditional Software Engineering approach, exploring its principles and applications in the domain of application software development.

Additionally, it evaluates the efficacy of this approach, particularly in an academic context. By shedding light on this innovative paradigm, we aim to foster a deeper understanding of the transformative potential it holds for the software development landscape.

# Experiment 3 (Medium Application, High Detail initial Prompt) Class B Application (5-10 classes): Class B Application – Club Membership Planner

Description: Club Membership Management Application

1. **Background:** The application will serve as a centralized platform for a club membership chairman to manage a list of members. Specifically, to add, update, delete and report on membership.
2. **Basic Application Design (high detail):** 
   1. The application will follow basic Java CRUD design with the following class structure. The AddMember, UpdateMember,DeleteMember, and Report classes must be classes that can be tested independently**.**
   2. MainMenu Class: Upon launch, users are presented with a menu offering options to add, update, delete members, and generate membership reports.
   3. AddMember Class
      1. Upon launch displays a screen window to allow entry of membership data.
      2. Use the Java border layout configuration.
      3. Screen title in the north section.
      4. In the center section, the screen contains textboxes for member id (6 digit numeric) , member first-name, last-name, email address, dues paid, dues paid date, renewal date. Dropdown boxes for membership level (Level1, Level2, Level3), status (active, inactive)
      5. In the south section, Buttons: 'Submit' to save data, 'Cancel' to return to the main menu.
      6. Process
         1. Upon launch an empty screen is displayed.
         2. User enters membership data and indicate in iv. above.
         3. User clicks Submit to save the membership information. Note the member id is assigned as follows: scan the members table for the last member id in the members table add. if members table is empty start with member id 000001.
         4. The application calls the addMember method in the DatabaseUtil class to insert the member into the table.
         5. If user clicks the 'Cancel' button, the application returns without saving changes.
   4. UpdateMember Class:
      1. Upon launch displays a screen window to allow a search for a member id and the entry of membership data.
      2. Use the Java border layout configuration.
      3. Screen title in the north section.
      4. In the center section, the screen contains textboxes for member id, member first-name, last-name, email address, dues paid, dues paid date, renewal date. Dropdown boxes for membership level (Level1, Level2, Level3), status (active, inactive)
      5. In the south section, Buttons: ‘Search’ to search for the member id, ‘Submit' to save data, 'Cancel' to return to the main menu.
      6. Process:
         1. Users input a member ID (integer) and click the Search button. The application calls the getMember method in the DatabaseUtil class to obtain the data for the member.
         2. If found, the program displays the member's data, allowing edits.
         3. If not found, a "member not found" message appears.
         4. If user clicks the 'Cancel' button, the application returns without saving changes.
         5. If the user clicks Submit, the application calls the updateMember method in the DatabaseUtil class with the updated information.
   5. DeleteMember Class:
      1. Upon launch displays a screen window to allow a search for a member id
      2. Use the Java border layout configuration.
      3. Screen title in the north section.
      4. In the center, a textbox for the member id.
      5. In the south section, Buttons: ‘Delete’ to search for the member id, 'Cancel' to return to the main menu
      6. Process:
         1. User inputs a member ID.
         2. User clicks Delete button. The application calls the getMember method in the DatabaseUtil class to confirm the member id exists.
         3. If found, a confirmation prompt appears before deletion. If not, a "member not found" message is displayed.
         4. If the deletion is confirmed, the application calls the deleteMember method in the DatabaseUtil class to delete the member.
         5. If user clicks the 'Cancel' button, the application returns without saving changes.
   6. Report Class:
      1. Upon launch displays a screen window to allow selection of report type.
      2. Use the Java border layout configuration.
      3. Screen title in the north section.
      4. In the center, a Dropdown box for report type (all members, active members, inactive members, level1, level2, or level3).
      5. In the south section, Buttons: ‘Submit’ to generate the report, 'Cancel' to return to the main menu.
      6. Process:
         1. Upon launch the Report options scree is displayed.
         2. User selects report type based via dropdown selection.
         3. If the user clicks ‘Submit’. The application calls the appropriate method in the DatabaseUtil class and displays the list of members meeting the criteria.
         4. If user clicks the 'Cancel' button, the application returns without saving changes.
   7. DatabaseUtil Class:
      1. There must be a Database Utility class to manage access to the database.
      2. Methods
         1. addMember accepts membership information and inserts it into the database. If an error occurs, a message should be issued.
         2. getMember accepts member id information and retrieves it from the database returning the data to the caller. If an error occurs, a message should be issued.
         3. updateMember accepts membership information and updates the database. If an error occurs, a message should be issued.
         4. deleteMember accepts member id and deletes the member. If an error occurs, a message should be issued.
         5. statusReport accepts status selection criteria (active or inactive) information and retrieves it from the database returning the data to the caller. If an error occurs, a message should be issued.
         6. levelReport accepts level selection criteria (level1, level2, level3) information and retrieves it from the database returning the data to the caller. If an error occurs, a message should be issued.
         7. allMemberReport retrieves all members from the database returning the data to the caller. If an error occurs, a message should be issued.
3. **Modular Program Design:** Each function (add, update, delete, report) is developed as a separate class, ensuring maintainability and scalability. The classes should be designed to be tested independently.
4. **Database Structure:** 
   1. A MySQL database with a table for members.
   2. No other tables are needed for this application.
5. **GUI Framework**: Utilizes Java's Swing framework, ensuring a consistent and responsive user experience. No specific branding is needed.
6. **Error Handling**: Basic error messages for common issues like "member Not Found" or "Invalid Input". No other guidance is needed.
7. **Authentication**: Given the internal nature of the application, no authentication layer is required.
8. **Concurrency**: Designed for individual use, ensuring data integrity without the need for concurrent access handling.
9. **External Libraries**: While the core functionality relies on Java's standard libraries, external libraries can be integrated for enhanced database connectivity or specialized GUI components. NO external 3rd party libraries should be used stick to standard Java libraries.
10. **Deployment**: Packaged as a standalone application, it's deployable on both local VMWare Virtual Machines or cloud platforms for broader accessibility.
11. **User Roles and Permissions:** Single user application.
12. **Data Validation and Constraints:** Validate the status (inactive, active) and level (level1, level2, level3) fields. The date fields should be numeric.
13. **Backup and Recovery:** Not applicable
14. **Performance Requirements:** No specific requirements.
15. **Integration with Other Systems:** Not applicable.
16. **User Experience (UX) and User Interface (UI) Design:** No specific design guidelines, color schemes, or branding elements are provide – use your judgement.
17. **Logging and Auditing:** Not applicable.
18. **Feedback Mechanism:** Not applicable.
19. **Training and Documentation:** Consideration for user training materials or documentation to help faculty and staff get acquainted with the application. Develop after application is generated.
20. **Scalability and Future Expansion:** Not applicable.

Do you have any reflection questions or need clarifications?