***Project Questions for Chapter 1***

1. For the project assigned to your student team, discuss the possible functional requirements and non‐functional requirements for the project. Create a draft list of these requirements. Include possible questions that your team will need to answer about the requirements.

Functional Requirements – What a program needs to do

* Display Teams – Show teams as text, illustrations, or both.
* Store Teams – Keep data locally or remotely
* Sort Teams – Winning, losing, tie, match date, ascending vs descending, numerical, alphabetical
* Manage Teams – Up to 10 teams, add, delete, change
* Take Input – Is input entered via CLI, GUI, or neither
* Special cases, boundaries, or error condition – weekly, Saturdays, up to 3 matches (morning, evening, afternoon), 6 weeks long, may be extended by two weeks, each team should compete twice against each other team, shows which teams are playing each other each week, winner – 5 pts, loser – 1 pt, tie – 3 pts, shows scores highest to lowest, administrator and tournament officials have information entry access

Nonfunctional requirements – The manner in which the functional requirements need to be achieved

* Performance – responsiveness, speed of task completion
* Real-time – user feedback, action confirmation, responsiveness
* Security – secure accounts, protected data, tamper resistant

2. For the project assigned to your team, determine a specific, small programming piece from the project. Estimate the effort using section 1.4 of the book.

Developing the GUI

* Estimated Ideal Time: 35 Minutes
* Estimated Calendar Time:
* SceneBuilder project setup - 5 minutes
* Add Buttons – 5 minutes
* Add Labels – 5 minutes
* Set Buttons fx:id – 5 minutes
* Set Labels fx:id – 5 minutes
* Set Controller Class – 5 Minutes
* Save fxml file – 5 Minutes

***Project Questions for Chapter 2***

1. For the project assigned to your team, discuss and document a draft project decomposition. Please consider the divide and conquer technique.

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/Module%20Decomposition%20Diagram.png>

2. For the project assigned to your team, create a list of features for the project. This list of features would serve to “advertise” the project to interested users.

* Manage up to 10 teams
* Scores and rankings are automatically calculated
* Visually see matchups between all teams for the duration of the tournament
* Easy addition and removal of teams
* Flexible scheduling

***Project Questions for Chapter 3***

1. Your student team is a software engineering company. Discuss and develop your company’s home web page. Give a company name; slogan; list of services; etc.

Company Name: MacroSoft / Microsofties

Slogan: Just Code It

Services: Deliver comprehensive, affordable, reliable, and performant software services and solutions

2. Discuss and develop your student team’s code of ethics which will be posted on your company web site. Include in your documentation examples and research.

Code of Ethics:

1. PUBLIC – Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.

5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.

8. SELF – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

***Project Questions for Chapter 4***

1. Discuss and decide on a Software Process Model for your team project. Assign the top two choices advantages and disadvantages for each model. Create a draft of your team’s top choice with the following:

Waterfall Model

Pros

* Document-driven
* Precise status description
* Easy organization

Cons

* Requirements must be specified before anything else can be done
* Sequential flow
* Limited interaction with user/customer

Incremental Model

Pros

* Like Waterfall Model
* Compartmentalization of modules
* Risk containment
* Multiple releases for early feedback on critical components

Cons

* Requires deep understanding of problem, solution, and usage environments
* More difficult to manage

a) phases described with a start date and end date.

b) deliverables for each phase

2. Your student team company is at the initial level in the Capability Maturity Model. Describe what processes your team does with specific forms for the processes.

The CMM is a framework used to help a software organization define its level of maturity in software development. At the initial level there is no process, and any success is usually attributed to a strong and experienced leader. We mostly communicate via video and messaging platforms, e.g., Slack, Discord, or Skype. Communication is informal in the form of chat messages.

***Project Questions for Chapter 5***

1. Your student team software engineering company would like to try test‐driven programming for key functions needed in the project. Identify two key functions from the project and write test cases for these two key functions.

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/TestDatabase.java>

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/testSchedule.java>

2. Divide your student team into pairs of programmers. Each pair will have one week to implement a key function of the project. Each programmer will document their experience with a reflection at the conclusion of the implementation.

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/Pair%20Programming.docx>

***Project Questions for Chapter 6***

1. Find an opportunity to ask either the client or potential users the questions you created in Chapter 1 about the requirements. Document your interview and create a revised requirement list with descriptions.

Functional Requirements – What a program needs to do

* Display Teams – Show teams as text, illustrations, or both.
* Store Teams – Keep data locally or remotely
* Sort Teams – Winning, losing, tie, match date, ascending vs descending, numerical, alphabetical
* Manage Teams – Up to 10 teams, add, delete, change
* Take Input – Is input entered via CLI, GUI, or neither
* Special cases, boundaries, or error condition – weekly, Saturdays, up to 3 matches (morning, evening, afternoon), 6 weeks long, may be extended by two weeks, each team should compete twice against each other team, shows which teams are playing each other each week, winner – 5 pts, loser – 1 pt, tie – 3 pts, shows scores highest to lowest, administrator and tournament officials have information entry access

Nonfunctional requirements – The manner in which the functional requirements need to be achieved

* Performance – responsiveness, speed of task completion
* Real-time – user feedback, action confirmation, responsiveness
* Security – secure accounts, protected data, tamper resistant

2. Rewrite the requirement statements, clarifying and extending anything your team deems necessary, especially in the factors that should be considered for program similarity index. (Also consider how many of the 6‐categories of requirements need to be specified.) This will be the **Software Requirement Specification (SRS) Document** for your project. Refer to **Appendix B** – Essential Software Requirement Specifications (SRS) for examples.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

3. Create four to six drawings that show the screens for your project as a low-level prototype. Interview your client or potential users as a validation test. Document the findings of your validation test.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

***Project Questions for Chapter 7***

1. Create a module decomposition diagram for your student team project.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

2. Create the database design for your student team project.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

3. Create the team project’s software design. Document your design and the similarity index computation algorithm. Refer to **Appendix C** ‐ Essential Software Design for examples.

1. Architectural Design  
   MVC: Model-View-Controller
2. Use – Case Scenarios

Expand the previous documentation of the initial use case diagram into use case scenarios

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

Essential Use Case Documentation

Name: Select View

Preconditions: The selected view exists.

Postconditions: The selected view is shown.

Basic Course of Action

1. A user wants to see the schedule or rankings.

2. The user does not need to log into the system.

3. The user selects the schedule or ranking tab that they want to see.

4. The action event generated from the user’s click on the tab is given to the controller of the current view.

5. The controller engages the methods associated with handling the event generated with the tab in question.

6. The view associated with the tab is loaded into view along with any necessary data.

1. Sequence Diagrams

The use case scenarios developed in Step B would be developed into the sequence diagram with the invention of the many classes needed to support the action in the scenario. In parallel, the class diagram begins to incorporate the classes invented in in the sequence diagram and the methods are added to the class diagram as they are thought out by the team members.

1. Class Diagram

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

1. Collaboration Diagram

When a class needs another class to perform a sub step, those classes are associated in the collaboration.

1. Relational Database Design

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

1. User-Interface Design

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

1. Design Validation

The initial screens drawings and navigational flow are presented to the client or client’s specified user for acceptance, modification, and/or complete revision.

4. Code the system for the team project. This would include a listing of your source code, NOT including the library functions.

<https://github.com/ksturdivantwilson/SoftwareEngProject>

***Project Questions for Chapter 8***

1. Describe two good characteristics of the design in your team project.

Data Coupling – The degree of interaction and interdependence between classes is limited to the passing of data. As a result, the level of interdependence is low, and the level of cohesion is considered loosely coupled. The Teams, Schedule, and Account classes all perform operations independent of each other on the data that is initiated by the user Official. The data is then stored in the Database class to ensure consistency of information displayed across the application. If changes are made, the class responsible for said changes simply instantiates the Database class for access to the necessary information to perform said changes. Once changes are complete, the information is updated in the view and is stored back into the Database for the other classes reference.

High level of Cohesion - Communication to Functional. The activities of all classes are targeted on the same data that is stored in the Database Class. The classes focus on and achieve one to two main activities/functions. The classes are single/double goal oriented in some cases. The classes perform logic on the data and then pass the data to the next class. The Teams, Schedule, and Account classes are cohesive due to only performing functions related to the class itself. The classes never interact with data stored in other classes and manipulate said data. The functions pertain to relevant requirements associated with each class.

2. Describe two characteristics of your student team project’s user interface and demonstrate how these two characteristics positively impact the user.

Reduces the user’s memory load – The user interface is simple and intuitive where the overall design pattern is the use of tabs that allow the user to see all the possible paths immediately upon opening the application. The user does not need to worry about navigation between windows or screens as a result. All possible information is 1 – 3 mouse clicks away, assuming the worst-case scenario is a user who is a tournament official logging in. The user does not need to remember locations of major functions such as viewing the teams, rankings, and scores.

Design consistent user interface – When the user performs an action, the feedback from the system is informative and understandable. The user interface leaves little to no room for error. The only possible area prone to user mistake is the view representing access to the Official’s administrative functions. The login page’s text fields are try-catch blocked to intercept any unapproved formats or characters that may be entered by a user. The messages generated provide clear feedback should the user need any clarification on what they entered and what the login page is needed for.

***Project Questions for Chapter 9***

1. Document your team’s programming style and coding guidelines into your company’s standard. Discuss how this exercise makes your company stronger in CMM.

Coding Guidelines

The names for all classes, methods, variables, and files are aptly named. Standard Java naming convention is applied. Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Names are simple and descriptive. Whole words avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized, e.g., run (), runFast(), and getBackground(). Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore \_ or dollar sign $ characters. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.

Blank lines improve readability by setting off sections of code that are logically related.

Two blank lines should always be used in the following circumstances:

Between sections of a source file

Between class and interface definitions

One blank line should always be used in the following circumstances:

Between methods

Between the local variables in a method and its first statement

Before a block or single-line comment

Between logical sections inside a method to improve readability

Blank spaces should be used in the following circumstances:

A keyword followed by a parenthesis should be separated by a space. Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls. A blank space should appear after commas in argument lists. All binary operators should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. The expressions in a for statement should be separated by blank spaces. Casts should be followed by a blank space.

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Note: Examples for use in documentation should have a shorter line length-generally no more than 70 characters.

When an expression will not fit on a single line, break it according to these general principles:

Break after a comma.

Break before an operator.

Prefer higher-level breaks to lower-level breaks.

Align the new line with the beginning of the expression at the same level on the previous line.

If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

These coding guidelines lead to the development of a CMM of Level-3: Defined

At this level, documentation of the standard guidelines and procedures takes place.

It is a well-defined integrated set of project-specific software engineering and management processes.

2. Describe two characteristics of the good implementation in your student team project.

Readability – The software code can be easily read and understood by other programmers. Standard java convention and the coding guidelines stated above contribute to readability. Readability also improves Maintainability by proxy. As a result, both lead to the essential characteristics of correctness and completeness. Standards of indentation and formatting are followed, so that the code and its structure are clearly visible. Variables are named meaningfully, so that they communicate intent. Comments, which are present only where needed, are concise and adhere to standard formats. Facilities of the language are used skillfully, leveraging iteration and recursion rather than copy and paste coding. Functions are short and to the point and do one thing. Indirection is minimized as much as possible, while still maintaining flexibility.

Correctness – The software operates as intended as defined during the design and hence requirements. Adheres to the specifications that determine how users can interact with the software and how the software should behave when it is used correctly.

***Project Questions for Chapter 10***

1. Divide your team project so that each student has a section of the project. Prepare and document the testing intention, the test plan, and the test cases with expected outcome. Conduct the test and record the results with comments. Prepare and fill out a detailed time recording log and a complete defect log of the experience.

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/TestAdmin.java>

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/TestDatabase.java>

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/TestNewAccount.java>

<https://github.com/ksturdivantwilson/SoftwareEngProject/blob/main/TestTeam.java>

2. Create the **Test Plan & Test Result Document** for the entire project. Design and execute the test cases and record the testing activities while showing the success or failure of each test case. (e.g. Show sample screen snapshots of test results. Create a summary‐table with each row of the table depicting a test case, it’s expected result, its actual result, its pass/fail status, and failure severity, if applicable.) Refer to **Appendix D** – Essential Test Plan.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

A. Goals and Exit Criteria

1. Quality goals that need to be met for test phase to exit.

2. Robustness goals of the product.

3. Schedule goals of the project.

4. Performance and efficiency goals of the product.

B. Items to be Tested/Inspected

1. Executables such as modules and components.

2. Non executables such as requirements specification or design specification.

C. Test Process/Methodologies

1. Unit Test/Function Test/Acceptance Test/Regression Test/ and so on, methodologies.

2. Inspections/ reviews methodologies.

3. Black box testing (e.g., Input domain test, boundary value testing)

4. White box testing (e.g., control path testing, data flow testing)

5. Test metrics (e.g., code coverage, branch coverage, number of problems by severity)

6. Test – bug report – fix – retest process

D. Resources

1. People (number of skills, etc)

2. Tools (for measurement, defect management, etc.)

3. Systems (test execution platform, test case development, etc.)

E. Schedule

1. Test case development

2. Test execution

3. Problem reporting and fixing

F. Risks

1. Missing goals

2. Back-up resources needed

G. Major Test Scenarios and Test Cases

1. Boundary value and input domain test cases

2. Control path and dataflow test cases

3. Integration and intermodular test cases

***Project Questions for Chapter 11***

1. Trace your team project’s artifacts to uncover the defects injected for one of the test cases where you detected a fault.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

2. Your student company would like to have standards for future projects. Create a naming model for the requirement, design, and implementation artifacts. Show the traceability of your model with an example.

Part 0 – PP – Product

Part 1 – PHA – Artifact Origin Phase

Part 2 – CC – Country Code

Part 3 – RRR - Release

Part 4 – VVV - Versioning

Part 5 – TTT – Artifact Type

Part 6 – FF – Format

PP. CC. RRR. VVV. TTT. FF.

CM. US. 001. 001. DOC. IMG.

CM = ChessMaster | DES = Design Phase | US = United States | 001 = Version 1 |

DOC = Document | IMG = Image

***Project Questions for Chapter 12***

1. The client of your student project would like a training session for 10 new users. Create an outline and handouts for the training session.

2. Discuss and create a tailored maintenance change request form for your project. Document how this form is tailored to the project’s needs.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>

The Change Form Request artifact allows the developers to track popular customer/user requests for the chess management application. The form assists developers in keeping track of feature requests and establishes an official form of communication with users. Rather than having to rely on face-to-face communication with a representative or rely on email, the form provides a more official means of submitting requests for the clientele. The forms also assist developers in scheduling and dedicating the appropriate amount of personnel to the user’s issues, issues, or feature requests. The form tracks origin, approval, rejection, status, and completion of the submission.

***Project Questions for Chapter 13***

1. Discuss and document the monitoring activities your student company would like to establish as standard future operations.

1. Collection of project information

a. regular and formal project review meetings

b. management walk arounds

c. morning stand-ups

d. end of day stand-ups

2. Analysis and evaluation of the collected data

a. Data trend analysis and control charts

b. Data correlation and regression analysis

c. Moving averages and data smoothing

d. General model building for both interpolating and extrapolating purposes

3. Presentation and communication of the information

a. Pie chart, Histogram, Pareto diagram, Time chart, Control chart, Kiviat diagram

2. Create the **Team Document ‐** A description of the team members’ roles, initial project schedule, initial estimate of the project effort, and actual effort of the project. Include an analysis of the positive and negative outcomes with this team project.

<https://github.com/ksturdivantwilson/SoftwareEngProject/tree/main/ProjectDocumentation>