**Dataset**

<https://github.com/FrankLeeeee/CZ3003-SSAD/blob/master/backend/mazerunner/utility/questions.csv>

**World 1 Requirement Engineering**

Section 1 Requirement Engineering

Section 2 Requirement Analysis

Section 3 Software Requirement Specification

**World 2 Software Design and Quality**

Section 1 Software Design Principles

Section 2 Software Design and Quality

Section 2 Software Quality Attributes

**World 3 Software Architecture**

Section 1 Software Architecture

Section 2 Software Architectural Thinking

Section 3 Software Architectural Styles

**World 1 - Requirement Engineering**

Q1. Requirements engineering is concerned with identifying the purpose of a software system and the contexts in which it will be used.

Q2. 4 steps of requirements development: elicitation, analysis, specification, and validation.

Q3. Requirement elicitation is to classify the voice of the customer, which includes 9 aspects: business requirement, use case, business rule, external interface requirement, quality attribute, constraint, data definition, functional requirement, and solution idea.

Q4. 4 objectives of requirement analysis are to: decompose high-level requirements into details; identify the priorities; create alternative views of requirements; and look for incomplete/ambiguous/conflicted requirements.

Q5. 3 types of frequently used diagram for requirement analysis are: context diagram, use case diagram, and structured analysis models.

Q6. 5 types of structured analysis models are: data flow diagram, entity-relationship diagram, state-transition diagram (dialog map), decision tree, and CRUDL matrix.

Q7. Entity-relationship diagram is a representation of the logical relationships of data, which is useful for requirement engineering and database design.

Q8. Dialog map is a concise, complete and unambiguous representation of a finite-state machine for systems or objects.

Q9. CRUDL matrix includes actions, Create, Read, Update, Delete, or List, performed on entities by use cases/events/processes.

Q10. Software requirements specification (SRS) is a set of precisely stated properties and constraints that a software system must satisfy.

Q11. 2 objectives of software requirement specification are to: achieve agreement regarding the requirements among developers, customers, and other stakeholders; and provide the basis for design and system testing.

Q12. 5 teams that are audiences of software requirements specification are: end users, marketing, product management; project management; software, hardware, and systems engineers; testing group; maintenance and support team.

**World 2 - Software Design and Quality**

Q1. 4 software design principles are: modularity, single responsibility principle, principle of least knowledge and open close principle.

Q2. Modularity requires that the design should be composed of replaceable, self-contained assemblies of elementary parts, thereby aiding both the initial development and the later maintenance.

Q3. Single responsibility principle requires that each component or module should be responsible for only a specific feature or functionality, or aggregation of cohesive functionality.

Q4. Principle of least knowledge requires that a component should not know about internal details of other components.

Q5. Open close principle requires that software entities like classes, modules and functions should be open for extension but closed for modifications.

Q6. Software performance is the responsiveness of a system.

Q7. Portability is the easiness of a software system to be transported to multiple hardware/software platforms.

Q8. Reusability requires that the components of one software system can be easily reused in the development of other software systems.

Q9. Software design process is in nature iterative and model-based and has multiple solutions.

**World 3 - Software Architecture**

Q1. Software architecture is considered as a description of the high-level structure of a software system in terms of architectural elements and the interactions between them.

Q2. An architectural element, also known as component, is a unit of software that performs some function at runtime.

Q3. As with building blocks, software components must be decomposable to a point when they are no longer components. At this point, the design level changes to low-level classes and their operations and properties.

Q4. The interactions between components, also known as connectors, are the communication, coordination, and cooperation among components.

Q5. Some commonly used architectural styles include client & server, implicit invocation, batch sequential, pipe-and-filter, main program and sub routine, and layered system.

Q6. Different architectural styles could be combined as hierarchically heterogenous style, simultaneously heterogenous style, or locationally heterogenous style to design the system.

Q7. There are two fundamental categories of software: product and system.

Q8. Product is a set of dependent components joined immutably.

Q9. System is a bag of self-sufficient, substitutable components joined dynamically.

Q10. 5 key concepts to keep in view for building satisfactory software architecture are: essential elements, system perspective, structure, behavior, stakeholders.

Q11. 2 widely used software architectural process are the jumpstart and the software architecting canonical process (SACP).

**World 4 – UML Modelling**

Q1. Software components are high-level design, each is a set of classes, files, or scripts. The common atomic unit of software is a class.

Q2. Entity-Control-Boundary (ECB) pattern is used to create business-oriented components that are easier to unit-test and have a higher separation of concerns.

Q3. Entity is a persistent, passive class holding system data.

Q4. Control manages the boundary and entity classes and their interactions.

Q5. Boundary interacts with external class, and internal control classes.

Q6. Controller is the one-to-one “software interface” to a hardware device that typically is installed directly into the device.

Q7. Driver is the component in the operating system/system software that allows the OS to connect and communicate with a device.

Q8. Scheduler is the form of control effected when the current time and date reach some preset values.

Q9. Handler is the form of control effected when certain steps/actions are triggered.

Q10. Manager is the “brain” of all or a large part of some software, which often manages a group of schedulers or handlers.

Q11. Database Manager is the “front-end” that manages a database and handles queries to the database.

Q12. 4 recommended categories of software components are: thematic, standard, integrative and reused.

Q13. A thematic component provides a service embodying the intended theme of a software.

Q14. A standard component provides a service that is seen in many other software of its kind.

Q15. An integrative component connects other components together, at the same time “hiding” them from design changes.

Q16. A reused component is a clone/copy of design and implementation of some other component.

Q17. Façade class is a class through which external client classes can access a set of functions in supplier classes. It delegates the client class’ calls to the appropriate supplier class.

Q18. Façade design pattern decouples the supply classes from direct access by the client.

Q19. Model-View-Controller (MCV) architecture style is commonly used for an application’s first-level decomposition.

Q20. Model takes current state of the application and environmental conditions as inputs, applies rules and formulations to analyses and selects the most appropriate responses.

Q21. View outputs information in a unique perspective.

Q22. Controller controls operations of the MVC such as sending input parameters to model, launching views, and synchronizing views and model.

**World 5 - Software Reusability**

Q1. Common framework is a guide or support for building the complete structure and can be reused in many design scenarios and implementation instances.

Q2. Library is a collection of individual class compatible with a particular programming language that each provide reliable implantations for a variety of standard computer functions.

Q3. Toolkit is an application that provides a standard computer service.

Q4. Components and classes are software element main reusables.

Q5. 4 steps to reusing software components: identity opportunities to reuse; qualify fitness of candidate components; isolate any unwanted service; and make architectural style.

Q6. 2 effective and trustworthy sources of reusable software components: software frameworks and component-based software engineering (CBSE) repositories.

Q7. Inversion of control (IoC) describes a design where the reusable software dictates the overall flow of control, and not the custom-written portions.

Q8. With IoC, the reusable code and problem-specific are developed independently, even though they operate together in the program.