CS2103 Review Questions

You need to be able to answer these questions confidently!

Requirements Gathering / Specification

1. Who are typical stakeholders of a software project?

users, customer, developers?, dev company? govt? law enforcement? special interest groups?, public at large? …

2. Why asking the user is not enough?

The user is not the only stakeholder of the project. They can communicate the problem but rarely the solution.

3. How do *models* help to combat complexity?

Models can provide a simpler view of a complex entity by capturing only a selected aspect.

Models can be used to analyse complex entities related to software development. Models of the problem domain can be built to aid the understanding of the problem to be solved.

4. Why use UML?

UML is general purpose and known to all.

5. When should we prefer *use case descriptions* over *user stories*?

User Stories: User needs and justifications.

Use Case Description: User-system interactions for a usage scenario

There is no need to prefer one over the other, both serve a different purpose and are equally useful for requirement specification.

6. Comment on this step that appeared in a use case description: “user clicks the open button”

This step contains UI-specific details such as “Open Button”. Instead, it should communicate the intention of the actor. For example: “User chooses to open a folder”.

7. What does an *OODM* capture?

- **An OODM captures how objects in the problem domain interact with each other in the real world before we emulate them in the solution.**
- **OODMs represent the class structure of the problem domain and not their behaviour.**

Note: OODMs do not contain solution specific classes

8. What does an *object diagram* capture?

Object diagrams are used to capture the object structure at a given point of time that results from a design represented by a given class diagram.

9. How to use *object diagrams* in analysis phase?

During the analysis phase of a project, you might create a class diagram to describe the structure of a system and then create a set of object diagrams as test cases to verify the accuracy and completeness of the class diagram.

10. What are *non-functional requirements*?

- **Non-functional requirements specify the constraints under which the system is to be developed and operated.**
- **Examples of non-functional requirements include: Data requirements, Environment requirements, efficiency, fault tolerance, portability, reliability, response time, scalability, security, testability,…**

11. What's the difference between *requirements specification* and *system specification*?

Requirement specifications are written in terms of the problems that need to be solved, while system specification is written in terms of how the system solve those problems.

- **Requirements specification (user requirements) define the problem / need (a.k.a what the user wants the solution to do but it is NOT the solution)**
- **System specification is more about the solution, it specifies (using precise and contrasting statements) how the system will meet the user requirements. It is still what the solution will do, not how will the solution work, but it is how the system will meet the requirements.**

12. For what purpose do you use *sequence diagrams*?

To model interactions between various entities in system in a specific scenario. It is useful to verify that the design of the internal interactions is able to provide the expected outcomes.

- **High Level: Model how components of a system interact with each other to respond to a user action.**
    - **(Example: How to UI interacts with the Controller and Model components)**
- **Low Level: Model how objects inside a component interact with each other to respond to a method call that it received from another component.**
    - **(Example: How the Shopping Cart component in an online store responds to a method call it received from the User Interface)**

Design

13. Complete this sentence: A is coupled to B if ….

- **A has access to the internal structure of B**
- **A and B depends on the same global variable**
- **A calls B**
- **A receives an object of B as a parameter or return value**
- **A inherits from B**
- **A and B are required to follow the same data format or communication protocol.**

14. What is a *pattern*?

An elegant reusable solution to a commonly recurring problem within a given context of software design. (In this case, pattern refers to a recurring problem)

15. What's the difference between *separation of concerns* and *single responsibility principle*?

Separation of concerns: Separate the code into distinct sections such that each section addresses a separate concern.

Single responsibility principle: A class should only have and only have one reason to change. This means that a class should only have one responsibility.

Remarks
• Interestingly SOC and SRP are closely related. • Separation of Concerns is a broader concept that emphasizes breaking down a system into distinct loosely coupled components, each addressing a specific concern. The goal is to make the system more modular and easier to understand and maintain. • Single Responsibility Principle on the other hand is a more specific guideline focusing on individual classes. It advises that a class should have a single responsibility, meaning that it should encapsulate one reason to change. This promotes code that is easier to understand, maintain and modify, as changes to one responsibility does not affect other classes.

Project Management

16. Is Unified process *iterative* or *sequential*?

Unified process is iterative.

17. What's the difference between *buffers* and *padding*?

A buffer is time set aside to absorb any unforeseen delays because estimations for software development are notoriously hard.

Padding would refer to inflating task estimates (extra time added to a schedule) just to make you feel more confident in the estimate.

18. What's the benefit of identifying the *critical path* of a project?

Critical paths identify the sequence of project tasks that determine a project's duration and outline important deadlines to meet to deliver a project on time. In short, the critical path is the longest distance, or duration of time, between the start of a project and its completion.

Implementation

19. How is refactoring different from bug fixing?

Bug fixing alters the external behaviour of a component while refactoring improves a program's internal structure in small steps without modifying its external behaviour.

20. How are assertions different from exceptions?

Assertions are used for verifying assumptions about the program state while exceptions are used to deal with unusual but not entirely unexpected situations.

Tools

21. Is Git centralized or distributed RCS?

Git is a distributed Revision Control System (RCS).

22. What distinguishes a platform from a framework or a library?

- **A platform provides a runtime environment for applications.**
- **Frameworks are a reusable implementation of a software providing generic functionality that can be selectively customized to produce a specific application.**
- **Libraries are meant to be used as it is as they are a collection of modular code that is general and can be used by other programs.**

23. Describe CI (Continuous Integration).

- **The precursor to CI is build automation, which automates the steps of the build process which can include steps such as pulling code from the RCS, compiling, running automated tests, updating of release documents, packaging into a redistributable and so on.**
- **CI is an extreme application of build automation in which integration, building, and testing happens immediately after each code change.**

Quality Assurance / Testing

24. Is acceptance testing validation or verification?

Acceptance testing is more towards validation than verification as it is to check if the system was built to do what it was intended to do based on the use case specification defined at the beginning of the project.

25. Give one pro and one con of white box test case design.

White box test case design is more thorough and extensive than black box testing due to the ability to understand the program's internal workings. White box test case design is ineffective in a code base that changes quickly.

26. How is integration testing different from unit testing?

Integration testing aims to discover bugs in the glue code by testing whether different parts of the software work together as intended. Additional test cases are written to focus on the interaction between parts.

27. How does equivalence partitioning help in increasing E&E of testing?

Equivalence partitioning can help to increase efficiency of testing by reducing redundant test cases and by ensuring that all partitions are tested, it increases the effectiveness of testing by increasing the chances of finding bugs.

28. Give an advantage of TDD.

High test coverage because a test is written for each feature.

29. Which is stronger: statement coverage or path coverage?

Path coverage is stronger. Path coverage measures the number of possible paths given a part of the code executed. Often it is the case that the number of possible paths is more extensive that covering the number of lines of code executed.

30. What is the purpose of dependency injection?

The purpose of dependency injection is to inject stubs to isolate the software under testing from its dependencies so that it can be tested in isolation.

Less known facts

1. **Two unidirectional associations do not always equate to a single bidirectional association.**
2. **User stories can also capture non-functional requirements.**
3. **Some facts about use cases:**
    a. **A use case can involve multiple actors.**
    b. **An actor can be involved in many use cases.**
    c. **A single person / system can play many roles.**
    d. **Many persons / systems can play a single role.**
4. **Use cases only describe externally visible behavior, not internal details of a system.**
5. **If class A cannot compile without class B, class A is dependent on class B.**
6. **Code alone may not be enough to draw the matching UML diagram.**
7. **OODMs do not show methods or navigabilities**

8. Here are some less obvious dependencies that you should list down in your class diagram.

Example 1

```
class Activity{
  private Watcher[] watchers = new Watcher[ProgressWatcher.MAX];
  //...

  void watch(Watcher w){
  //add w to watchers
  }

  Activity getInstance(){
  //...
  }
}
```

No need to draw this dependency since Activity is already associated to Watcher.

package level access

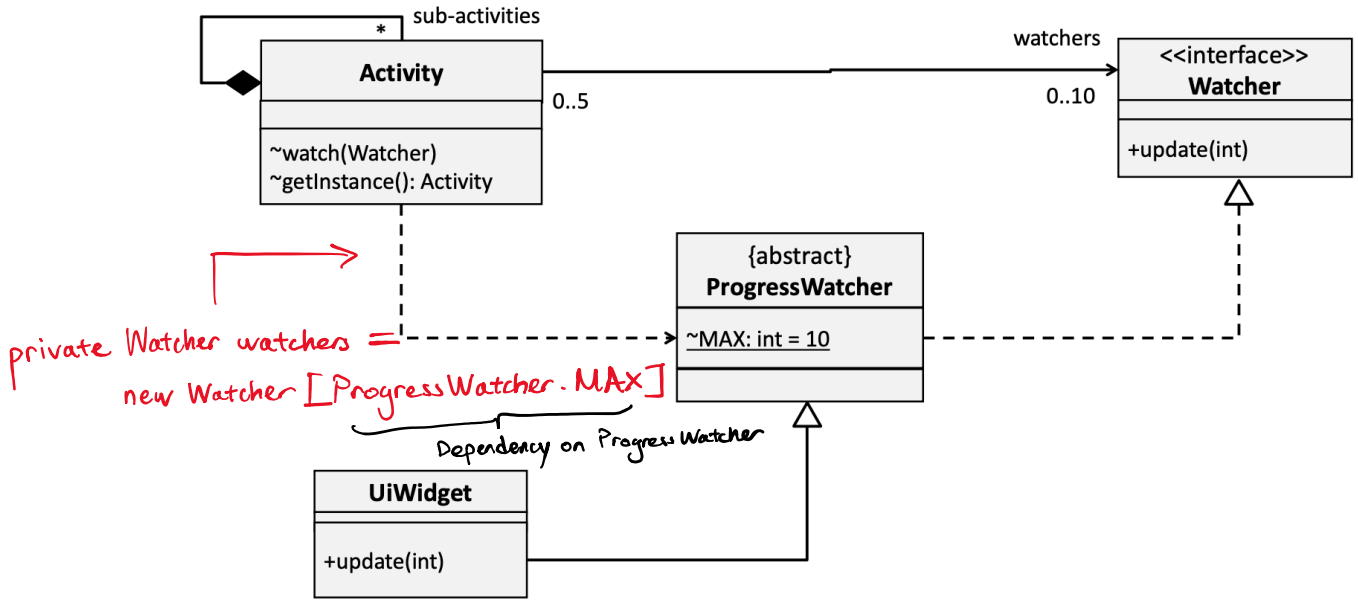
```
interface Watcher {
  void update(int value);
}
```

```
abstract class ProgressWatcher
implements Watcher {
  static final int MAX = 10;
}
```

Additional Information

- **An Activity object can be composed of other Activity objects i.e., sub-activities.**
- **A Watcher object may not be associated with more than 5 Activity objects.**
- **UiWidget class inherits the ProgressWatcher.**

Suggested Answer



Class Diagrams: Okay to omit methods, visibilities, navigability.

Object Diagrams: Okay to omit variables and variable names.

Sequence Diagrams: Okay to omit activation bars and return arrows if omitting it does not lead to ambiguities.

Activity Diagrams: Okay to omit the diamond at the end of the alternative path if it doesn't introduce ambiguities.