**CS673 Software Engineering**

**Review Sheet**

**Yuting Zhang**

**BU MET CS**

# Final Exam Information

1. 3 hours, 100 points. Covering Module 1- 6.
2. 3 types of questions:
   1. 8 multi-answer questions (concept questions): 4 points each, 32 total
   2. 1 project related question: 8 points
   3. 1 case study question (containing 7 subquestions): 60 points

# Basic SE Concepts

1. Explain essential difficulties and accidental difficulties described by Brooks in the paper “No Silver Bullet”.
2. Describe software engineering code of ethics.
3. Compare waterfall model, spiral model, unified process model and agile model. How can these models apply in contemporary development?
4. Describe the software development process used in your group project.
5. Describe what CMMI is.
6. Describe Agile manifesto and principles.
7. Compare XP and Scrum. Explain basic practices and terms in each framework, such as user stories, pair programming, TDD, sprint, scrum master, etc.
8. Describe key features of DevOps.
9. Describe quality metrics (including process and product metrics) and their indications. List quality metrics used in your project and explain how to interpret these metrics and how to improve the quality.
10. Compare validation and verification in the quality assurance process. Explain code review, inspection and testing.
11. Compare distributed and centralized version control systems.
12. Explain basic terms used in git: branch, baseline, clone, commit, push, pull, etc.
13. Describe the risk analysis process. Explain the likelihood, impact and priority. List some top risks.

Requirement Analysis

1. Explain how to write good user stories using the INVEST principle.
2. Explain how to write good acceptance tests for a given user story.
3. Compare user stories and use cases. Explain how to write flow of events in a use case description.
4. Explain the relationships in a use case diagram.
5. Compare functional and nonfunctional requirements. Provide some examples of nonfunctional requirements.
6. Explain what the product backlog is and how to manage the user stories. Explain how to estimate story points and how to calculate and interpret velocity. How to use epics and themes to organize and manage user stories? How to handle requirement changes using user stories?
7. Explain what scope creep is and how to avoid it.
8. Explain what technical debts are and how to mitigate them?
9. Explain how your group manages requirements and handles requirement changes.
10. Explain what ECB (Entity, Control and Boundary) objects are and how to identify them in a given problem.
11. Explain the relationships between classes in a class diagram.

Design

1. Compare coupling and cohesion and explain the general rule of the system design in terms of coupling and cohesion (low or high?).
2. Explain commonly used architecture styles such as client-server, p2p, MVC, layered and tiered. Give an example for each.
3. Compare open and closed layered architecture. Compare layered and tiered architectures.
4. Explain the compound design patterns (Observer, Strategy and Composite) related to the MVC architecture.
5. Compare the strategy pattern and the state pattern.
6. Explain the components in a state transition diagram.
7. Explain the components in a sequence diagram.
8. Compare SOA and microservice.
9. Describe serverless, FaaS and BaaS.
10. Explain when to use specification inheritance, implementation inheritance and composition respectively for resusabilty.
11. Explain the design principles in following acronyms: DRY, KISS, and SOLID.

Implementation

1. Explain refactoring, regression testing and their relationship.
2. Describe some common bad smells (such as duplicated code, long methods, large class, divergent change, shotgun surgery, feature envy, switch statements, message chains, middle man, comments, etc). List bad smells in your group project and possible refactoring techniques. Explain how these bad smells can violate design principles and hinder you from achieving the design goals.

Testing

1. Describe what should be included in a test plan?
2. Compare whitebox testing and blackbox testing.
3. Explain the TDD workflow and the relationships between TDD, refactoring and regression testing.
4. Explain method coverage, statement coverage, branch coverage, condition coverage and path coverage.
5. Explain the equivalence partition and boundary cases used in the domain testing. Explain how to use risk based domain testing to design test cases.
6. Describe the testing techniques and quality metrics used in your group project.

Security

1. Explain CIA and IAAA.
2. Compare Seven Touchpoints, Microsoft SDL and OWASP SAMM. List three common security practices used by all three models and explain how your group uses (or can use) these practices.
3. Explain the STRIDE threat modeling.
4. Compare SAST and DAST.
5. List at least three common vulnerabilities in the web applications or mobile applications. List possible vulnerabilities in your project.

Case Study Questions:

1. Given a short description of the functionalities of a system, identify possible user stories.
2. Given a short description, write acceptance tests for given user stories.
3. Given a short description of the functionalities of a system, identify all actors, possible use cases and draw a use case diagram.
4. Given a use case diagram and some short description, write a use case description for some use cases.
5. Given a short description (or user story or a use case description), identify application domain classes/objects (Entity, Boundary, Control) and draw a class diagram (show class name, major attributes, operations, and associations (including multiplicity)).
6. Given a short description, draw a state diagram.
7. Given a description of requirement or design, draw a sequence diagram.
8. Given a design description, draw a class diagram.
9. Given a short description, identify the possible architecture styles that can be used.
10. Given a problem, identify possible design patterns that can be used. Draw a class diagram and write some sample code. (We will focus on Strategy, observer, composite and state patterns.)
11. Given a problem description, write the unittest code.
12. Given a problem, design test cases using the risk-based domain testing techniques.
13. Given a piece of code, identify the test cases needed for 100% line coverage, branch coverage and basis path coverage.
14. Given a piece of code, identify bad smells and possible refactoring techniques.