

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page. The exam has 10 questions with equal weight. Some questions have multiple parts. Point totals are given in the margin.

Student Id: _____ **Answer Key** _____

1. _____ **10**
2. _____ **10**
3. _____ **10**
4. _____ **10**
5. _____ **10**
6. _____ **10**
7. _____ **10**
8. _____ **10**
9. _____ **10**
10. _____ **10**
- Total _____ **100**

No notes or books may be used on the exam. If you have any questions, please raise your hand and I will try to answer them.

- 10 1. List the stages of the waterfall process model in order and briefly describe each stage.

Answer:

1. requirements analysis: determine what is to be built by talking with the customer; develop a specification from which system design can be done; ensure that the requirements are complete and consistent
2. design: devise an elegant, efficient solution to the problem described by the requirements that can be mechanically implemented; ensure that the design satisfies the requirements
3. implementation: implement the product according to the design
4. testing: ensure that the implementation satisfies the design and that the design satisfies the requirements
5. maintenance: fix any defects found in the software; adapt the product to new platforms; add enhancements

2. Consider the following in answering the questions: decision tables, transition diagrams, traceability tables. Note that some, all, or none of the three might apply in each question.

2

- (a) How can we tell if the requirements are incomplete?

Answer: In a decision table, the requirements are incomplete if a column exists that has conditions, but no associated actions. In a transition diagram, the requirements are incomplete if all possible transitions are not shown from some state. Traceability tables are not much use here.

2

- (b) How can we tell if the requirements are contradictory?

Answer: In a decision table, the requirements are contradictory if two columns have the same conditions, but different actions. In a transition diagram, the requirements are contradictory if some state has two transitions on the same condition leading to different states. Again, traceability tables are not much help here.

2

- (c) How can we tell if the requirements are redundant?

Answer: In a decision table, the requirements are redundant if two columns are identical. For transition diagrams, if two transitions on the same condition lead to the same state, then the requirements are redundant. (Although we really need to check for isomorphic subgraphs here.)

2

- (d) How can we tell if the requirements are correct?

Answer: There is no way to ensure that the requirements are correct. All three mechanisms give you extra confidence that the requirements are correct, but none are foolproof.

2

- (e) How can we tell if the design satisfies the requirements?

Answer: We can use a traceability table in which the columns are the requirements and the rows are the design decisions. We need to ensure that each column has at least one mark (i.e., is met by some design decision).

3. Answer the following general questions related to software design:

2

(a) What is the difference between a requirement and a design decision?

Answer: A requirement addresses what the system must do. A design decision addresses how a requirement will be met by the system.

2

(b) Why is low coupling important?

Answer: Low coupling indicates that a proposed change to one module will be less likely to impact the rest of the system.

2

(c) Why is high cohesion important?

Answer: High cohesion helps reuse and understanding since all parts of a task are located in a single location.

2

(d) What is the purpose of architectural styles?

Answer: The purpose of architectural styles is to convey a general impression of the overall form and structure of the system.

2

(e) Why is the design stage typically broken down into system design and program design?

Answer: The design stage is typically broken down into system design and program design for simplicity and manageability. During system design, analysts are usually present since the design is at a high level. During program design, programmers are present since the design is at a low level.

4. Define the following terms related to software design:

2

(a) abstraction

Answer: Abstraction is the rendering of lower-level details temporarily invisible to upper levels in order to facilitate the understanding and design of complex systems.

2

(b) cohesion

Answer: Cohesion is the degree to which a module does a single, well-defined, well-executed task.

2

(c) coupling

Answer: Coupling is a measurement of how much a module is tied to or dependent upon other modules.

2

(d) modularity

Answer: Modularity is the principle or method of decomposing a system into independent components that can be independently managed.

2

(e) stepwise refinement

Answer: Stepwise refinement is the gradual elaboration of details of a system in a top-down manner.

5. Answer the following questions regarding architectural styles (object-oriented, pipe and filter, repository, layered, call-and-return, etc.):

5

(a) Which style or styles would you choose if efficiency was most important? Which would you not choose? Briefly justify your answer.

Answer: Repositories are very efficient because there is typically only one copy of the data. Repositories also allow for parallelism. Object-oriented and event-based architectures can also be efficient. Pipe and filter and layered architectures are typically not very efficient. The former because a complete transformation of the input is required; the latter because crossing layers incurs a cost.

5

(b) Which style or styles would you choose if simplicity was most important? Which would you not choose? Briefly justify your answer.

Answer: Pipe and filter architectures are probably the most simple since they describe a straightforward, step-by-step approach to solving a problem. Layered architectures can also be simple since they are modeled using abstraction. Event-based architectures are typically not very simple since control is distributed. Other architectures all have aspects that can make them relatively simple.

6. Define the following terms related to object-oriented design:

2

(a) class

Answer: A class is a data type that holds attributes and the operations on those attributes.

2

(b) inheritance

Answer: Inheritance is the mechanism by which a child class can extend the operations and attributes of a parent class. The child class inherits (has access to) all operations and attributes of the parent class, and therefore the child class behaves like the parent class.

2

(c) message

Answer: A message is the mechanism by which objects communicate. A message is a triple specifying the destination object, operation, and its parameters. Messages invoke methods.

2

(d) object

Answer: An object is an instance of a class.

2

(e) polymorphism

Answer: Polymorphism is a mechanism by which an object of a derived class can be used in place of an object of a parent class but with the operations of the derived class invoked rather than those of the parent class.

7. Answer the following questions related to software testing:

2

(a) Why is exhaustive testing of inputs not possible?

Answer: Exhaustive testing is not possible because the number of possible inputs for any nontrivial program is too large to try all of them.

2

(b) What is the goal of software testing?

Answer: The goal of testing is find errors in software.

2

(c) What is the primary advantage of black box testing?

Answer: The primary advantage of black box testing is that the test engineer does not have to know the structure of the code since testing is based solely on correspondence of inputs to outputs.

2

(d) Why is a test engineer unlikely to perform white box testing?

Answer: In white box testing, the structure of the code must be known, and it is unlikely that a test engineer (unless he or she is also the author of the code) would know or understand suitably the code structure.

2

(e) What is the difference between validation and verification?

Answer: Validation determines whether the right product was built (i.e., in accordance with customer requirements). Verification determines whether the product was built right (i.e., in accordance with the design).

8. Answer the following questions related to unit testing:

2

(a) What is the goal of unit testing?

Answer: To test each component separately to ensure that it works correctly.

2

(b) How can we unit test code without actually executing it?

Answer: We can conduct code reviews, formally prove the code correct, or use symbolic execution.

2

(c) Why is it a good idea to have someone else test your code?

Answer: It is a good idea to have someone else test your code so the test cases are not biased.

2

(d) A program expects two points (i.e., pairs of real numbers) as input and computes the distance between them. What are four possible good test cases? Your answers should be in general terms.

Answer:

- the origin and another point
- two random points in the plane
- the same two points
- illegal input

2

(e) Is it better to ensure that all possible paths are executed or that every statement is executed at least once during testing? Explain.

Answer: Ensuring that all possible paths are tested is better in the sense that it provides better test coverage. However, it is generally impractical. Ensuring that every statement is executed at least once is more practical, and still provides adequate coverage for most applications.

9. Answer the following questions related to *The Mythical Man Month*:

5

(a) How does a chief programmer team differ from a conventional democratic team?

Answer: In the chief programmer team, the chief programmer (and copilot) are aware of all of the work that is being done. In the conventional team, each member works more or less blindly on his or her own component after the work is divided. Also, in the chief programmer team, the chief programmer settles also disputes and is clearly in charge, so the team is not completely democratic.

5

(b) Why does Brooks advocate “plan to throw one away; you will anyway”? In particular, what failing in software engineering is he discussing.

Answer: His argument is that it is impossible to get the requirements right the first time you build a system, no matter how hard you try. Therefore, the first system you build will never be exactly what the customer wanted. His question is whether you should deliver that first system to the customer or whether you should instead build a prototype to show to the customer for feedback.

10. A bank might have several branches in different cities and communities, with each branch possibly providing different services such as automated teller machines (ATMs), business services, mortgage services, and traditional customer services. The bank's customers may have different types of accounts at the bank. Example accounts include checking accounts, savings accounts, and money market accounts. Accounts may earn a rate of interest and may have a minimum balance. Customers also may use the bank's ATMs located at branches and at separate locations. Each customer uses his or her personal identification number (PIN) to access an ATM.

Draw a UML structural model class view to represent the bank, its services, its customers, their accounts, and their relationships. You do *not* have to show the attributes, the methods of each class, or the cardinality of the relationships. You need only show the classes and their relationships.

Answer: Example classes and subclasses include:

- bank
- branch
- service
 - ATM
 - mortgage
 - business
 - customer
- account
 - checking
 - saving
 - money market

Example relationships include:

- bank has branches
- branches provide (or have) services
- customers have accounts
- accounts are at a bank (and not a branch)

Note that interest rate and PIN should be attributes, and not classes.