Software Design
Example Test Questions

Test questions can be based on the following sources: (1) the textbook(s), (2) readings, (3) lectures, (4) reports, (5) exercises, and (6) on-line notes and slides. They are based on topics from the beginning of the year up to the class before the test.

Consider all concepts and terminology used in the text book, reports, slides and lectures and ask the typical questions - how, why, when, where and what - individually and in combination. In particular, variations are based on "describe", "explain", "define", "what is meant by", etc.

There can be technical questions based on design notations and mathematical concepts. For example on top down design, design by contract (especially with inheritance), BON notation and others. The list is by no means exhaustive.

1. Top Down Design

1. Give a general template for refining an operation into a sequence and state what questions a designer must answer to verify the sequence is a correct refinement.

2. Give a general template for refining an operation into a choice and state what questions a designer must answer to verify the choice is a correct refinement.

3. Give the best precondition, postcondition and loop invariant for the following algorithm to do a binary search on an array $A[1..N]$. Your work should use as much mathematical notation as possible to capture the relevant information.

   ```
   low ← 1
   high ← N
   Result ← 0
   while Result = 0 and low ≤ high do
     mid ← (low + high)/2
     if $A[mid] = k$ then Result ← mid
     elseif $A[mid] < k$ then low ← mid + 1
     else high ← mid - 1
     fi
   end while
   ```

4. What would be the best precondition, postcondition and loop invariant for the following algorithm to find a span of dash characters in an array.

   ```
   charPointer := 1
   loop exit when textLine(charPointer) not equal dashChar
     charPointer := charPointer + 1
   end loop
   ```

5. What would be the best precondition, postcondition and loop invariant for the following correct algorithm to find the maximum integer and its index in the array $A[1..N]$.

   ```
   max := 0 ; maxIndex := 0
   ```
j := 1
while j <= N do
    if max < A[j] then max = A[j] ; maxIndex := j end if
    j := j + 1
end while

6. Prove the following algorithm to sum the odd integers in the range 1 to N inclusive is correct. Clearly show the correspondence with the questions a designer must ask when verifying a loop is the correct refinement of an operation.

Precondition: \( N > 0 \)
Postcondition: \( \sum \{ j : 1..N \mid \text{odd}(j) \times j \} \)
Loop invariant: \( \sum \{ j : 1..p \mid \text{odd}(j) \times j \} \)

\[
\text{sum} := 0 \\
p := 0 \\
\text{while } p < N \\
\quad p := p + 1 \\
\quad \text{if odd}(p) \text{ then sum := sum + p fi}
\]

7. What is a loop invariant? When is it used? How is it used? Why is it used? Where is it used?

8. Given a loop, pre & post conditions and loop invariant prove the loop is correct or is incorrect.

9. Given pre & post conditions and a loop invariant create the corresponding program text in Eiffel.

10. Given an algorithm and description, give the best precondition, postcondition and loop invariants.

11. Give a general template for refining an operation into a sequence (choice or loop) and state what questions a designer must answer to verify the sequence is a correct refinement.

2. **Design by Contract**

1. Explain when one can use design by contract?

2. Explain when one cannot use design by contract?

3. Explain what is meant by strong and weak assertions.

4. State and explain the correctness rule for a class with respect to its assertions for a creation routine.

5. State and explain the formal correctness rule for a class with respect to its assertions for exported routines.

6. State and explain the formal correctness rule for retry inducing rescue clause (does a retry).

7. State and explain the formal correctness rule for failure inducing rescue clauses (doesn’t do a retry).

8. Describe the benefits and obligations of the client and supplier when using design by contract.

9. What is defensive programming?
10. Explain why defensive programming is a poor method.

11. The following figure shows a simple class interface for a class MY_SET that holds only integers. The elements of the set are represented as an ARRAY of integers.

```
MY_SET

is_empty : BOOLEAN
is_full : BOOLEAN
insert (x : INTEGER)
remove (x : INTEGER)
member (x : INTEGER) : BOOLEAN
union (s : MY_SET)

contents : ARRAY[INTEGER]
```

12. Give require and ensure clauses for each of the set features. Write your contracts as formally as possible. You can use any features of ARRAY that you like in your contracts; if you are unsure as to which features ARRAY possesses, clearly state your assumptions. You may use BON assertion language. Answers expressed in mathematics receive significantly higher grades than answers expressed in English.

Suppose you are given the following classes.

```plaintext
class A
feature
    x : INTEGER ; y : INTEGER
do_work (a : INTEGER) is
        require a > 0
        ensure x >= a + 10
            and y < old y + 5
        end
end

class B
inherit A redefine do_work end
feature
    do_work (a : INTEGER) is
        require else
            a >= 0
        ensure then
            x >= 14 + a - 3
            and y - old y = 3
        end
end
```

Is the redefinition of feature do_work valid according to Eiffel’s refinement rule? Carefully prove that your answer is correct.

13. Design an appropriate contract for an abstract data type.

14. Defensive programming suggests the following programming style.

```plaintext
remove (object) is
    require size > 0
    if size < 1 the throw exception
    else ... rest of procedure ...
end remove
```

Using example pseudocode from the client side show why defensive programming is futile, if the client is a good programmer.
15. Suppose a class has an invariant that includes the clause \((x \geq 0 \text{ and } y = 0)\). I want to inherit from this class and in doing so, add a new clause \((x \geq 0 \implies y \geq 0)\). Is this acceptable? Why or why not?

3. **General Design Questions**

1. Explain what the Explicit Interfaces Rule is and the context in which it is used.

2. What is the self-documentation principle? How does Eiffel support this principle?

3. Distinguish between programming and design.

4. Explain what is meant by referential transparency. Explain why referential transparency is desirable.

5. Critic a design -- what is good about it, what is bad, what are fatal flaws.

6. Critic a design with respect to good OO design techniques and all the other "motherhood" concepts in first chapters of the textbook and the slides on design context, modularity and reuse.

7. Explain what the Explicit Interfaces Rule is and the context in which it is used.

8. Describe the waterfall model of software development. What are its limitations.

9. Describe each of the following object oriented design principles: abstraction, encapsulation, modularity.

10. Modular design issues deal with information hiding and independence. Explain what is meant by these terms.

11. Explain what is meant by the Open-Closed principle.

12. Describe goals of Resuability.

13. Explain the relationship between garbage collection and dynamic memory management.

14. Give and explain the type rule without genericity.

15. Give definitions of the following with respect to software design: Error, Defect, Fault.

16. Explain what is meant by seamlessness in the context of software design.

17. What is the direct mapping rule?

18. What is the single choice principle?

19. What are the pros and cons for functions not having side effects?

20. Explain what is meant by referential transparency.

21. What are the issues on the number of arguments for a feature?

22. Is class size and issue when designing a class? Explain why or why not?

23. What are active data structures? When would they be used? Why are they used?
24. What is the rule of change in the context of deciding between use and inheritance?

25. What is the distinction between design and program?

26. Where does design fit in the life cycle of a program?

27. What is the life cycle of a program? Describe each stage.

28. Why is design necessary? What is its purpose?

29. What is a program design?

30. Why is feedback necessary in program development?

31. What arguments support the intertwining of specification and implementation? Have you experienced or can you imagine other examples.

32. What is the accident in design and programming? What is the essence?

33. How do the methods described in the course address the accident and essence in design and programming?

34. How do the methods described in the course fit in with the "past breakthroughs" that Brooks describes?

35. What is structured design as proposed by Stevens, Meyers and Constantine?

36. How do abstract data types fit into a design method?

37. What are the classical design methods?

38. Describe each of the following design methods: top down design; bottom up design; structured design?

39. Explain the four principles of program design which Douglas Comer described in his paper.

40. Describe how the following design methods are used in combination to create a design? Top Down, Bottom Up, Structured.

41. What are Dijkstra’s guarded commands? Show the syntax and explain the semantics.

42. Why are Dijkstra’s guarded commands important? What are the benefits of using guarded commands?

4. **Eiffel**

1. Java distinguishes between the primitive types -- int, char, real, etc. -- and real objects. Eiffel does not make this distinction. Explain how Eiffel can treat the primitive types as first class objects.

2. Eiffel has four mechanisms for adaptation. Describe and give an example of each one.

5. **ADTs and Classes**

1. Describe the relationship between an abstract data type and a class.
2. Describe the basic steps in getting a class from an abstract data type.

3. Consider a priority queue \( PQ \) as a sequence consisting of items, each called \( \text{item} \), each containing the following fields.

\[
< \text{priority} : \text{INTEGER} , \text{time} : \text{INTEGER} , \text{data} : \text{ANY} >
\]

Larger integers indicate higher priority and later arrival time. Time increases indefinitely

A. Mathematically, give the simplest class invariant that describes such a priority queue.

B. Mathematically, give a class invariant that describes such a priority queue and also captures the notion of a priority queue as consisting of a sequence of sub-sequences of the same priority.

4. Describe the relationship between an abstract data type and a class.

5. Given the signatures and a brief description of the methods for an ADT give a set of axioms for the methods. For an example see slides on Abstract Data Types. Typically data types could be stacks, queues, priority queues, and other data structures from the second year course.

6. Describe the basic steps in getting a class from an abstract data type.

7. Describe various ways which are used to find classes.

8. Describe various issues pertaining to designing classes.

6. **Inheritance**

1. Give and explain the type rule with genericity using an example method call.

2. When you redefine a feature, there are two major forms you redefine. Explain what they are, why you need redefinition of that form and what are the constraints on redefinition?

3. Consider the following inheritance diagram. What problems can arise and how can they be resolved? You are not required to solve all problems simultaneously, just describe the various problems and their resolution.
4. What is the rule of change in the context of deciding between use and inheritance? Give three natural language definitions and their corresponding Eiffel class definitions that illustrate the choices in deciding between use and inheritance?

5. Give examples of the modelling variations between use and inheritance.

7. BON

1. For each of the following requirements, identify candidate classes (with appropriate names) and define their relationships using BON. Do not provide any class interface details (use the short ellipse notation for classes). Briefly explain each diagram in English.

   A. A father or a mother may have a son or daughter. A son or daughter may have a brother or a sister.

   B. A vehicle may be rented on a certain date. The rental will be at a specific rate and will be for a specific driver. Insurance is optional.

   C. A conference session may be a tutorial session, paper session or demo session. Each session involves one or more presentations. Paper presentations are reviewed, while tutorial presentations are not.

2. Describe, in development order, the nine development steps for the BON process to analyze and design a system — include static and dynamic aspects. Include, in the earliest step that an artifact is produced, a brief description of the artifact. Take time to organize your answer. It must fit within the allotted pages. Note instructions 4 and 6.

3. Give a BON diagram description of a design.

4. Explain what a BON diagram describes.

5. Given a description of a simple system. Give a BON static diagram for the system.

6. Given a BON static diagram. Critique the design.

7. Draw a diagram and describe the components of the BON static model.

8. Draw a diagram and describe the components of the BON dynamic model.

9. Describe the development steps and the artifacts produced or modified at each step of the BON process to analyze and design a system.

10. Briefly and informally clarify the difference between an association relationship and an aggregate relationship. When might you use aggregation but not association?

8. Patterns

1. In a few sentences, explain the Visitor pattern. Give an example of its use, written in Bon. Briefly describe your example in English.

2. What are design patterns? When are they used? When can they not be used?

3. Be prepared to describe design patterns discussed in class.

4. In a few sentences, explain the Decorator pattern. Give an example of its use, written in Bon.
9. **JSP**

1. Describe and give examples for the basis behind JSP.

2. How are sequence, choice and iteration diagramed by the Jackson method?

3. Given a small problem, give an example design.

4. What is meant by structure correspondence and by structure clashes?

5. What are the major types of structure clashes?

6. Describe and give examples for each of the following: boundary clash; structure clash; order clash; multi-threading clash.

7. How are structure clashes resolved using JSP?

8. How is JSP used to solve problems where data structures may contain errors? Give an example.

9. Describe and give examples for the basis behind JSP; that is, what program design problems does using JSP solve.

10. Describe how one designs a program using JSP. Do not consider implementation, only design.

11. What are the similarities and differences in overview and perspective on design among the following design methods? top down, structured design, JSP, programming calculus.

12. Describe and give examples of the following communication methods: asynchronous communication; synchronous communication.

13. What is program inversion? If inversion is chosen as a programming technique, how does one choose which programs to invert? Why? When is process inversion used?

14. What is forward chaining, backward chaining? How do these notions relate to schedulers?

15. What are coroutines? How would they be implemented? What purpose do they have in program design and or implementation?

16. Outline the steps that need to be done to invert a process. Illustrate with pseudo-code statements and/or segments of code where appropriate.