Florida Institute of Technology

Department of Computer Sciences

Comprehensive Exam (Spring 2008)

SOFTWARE ENGINEERING

Friday, March 14th, 2008; 10:00am – 11:30am

Instructions

- Write the last four digits of your student identification number in the space below.
- This exam consists of 18 pages (including this cover).
- **The exam consists of two parts:**
  - **You must answer Questions 1, 2, 3, and 4.**
  - **Answer any one (1) of Questions 5, 6, 7**

  ➔ **Circle the one question that you want graded:**

  5 6 7

  (If you leave this blank, Question 5 will be graded.)
- Use a pen to write your answers in the space provided.
- When a question asks you to "describe," "discuss," or "explain" something, it means you must provide a convincing, clear, and reasonable answer; simply stating a fact without any supporting argument is insufficient.
- No study aids (notes, books, etc.) are permitted during the exam.

Good luck!

ID Number:

For Grading Use Only

	Question	Worth	Grade
1.	General Knowledge	15	
2.	Requirements	20	
3.	Design	20	
4.	Construction	20	
5.	Testing	25	
6.	Maintenance & Evolution	25	
7.	Process	25	
	Total	**100**	

1. General Knowledge - Multiple Choice (15%)

Grading: 1% for each correct answer; total 15 questions.

Note: For each question, please select only *one* answer that best suits the question. Selecting no answer or more than one answer will result in 0 points for that question.

1. The aim of software engineering is to produce software that is:
   (a) Fault-free
   (b) Delivered on time
   (c) Delivered within budget
   (d) Satisfies users' needs
   (e) All of the above

2. Software deteriorates rather than wears out because:
   (a) Software suffers from exposure to hostile natural environments
   (b) Defects are more likely to arise after software has been used often
   (c) Multiple change requests introduce errors in component interactions
   (d) Software spare parts become harder to order
   (e) All of the above

3. The waterfall model of software development is:
   (a) A reasonable approach when requirements are well-defined
   (b) A good approach when a working program is required quickly
   (c) The best approach to use for project with large development team
   (d) An old fashioned model that cannot be used in a modern context
   (e) None of the above

4. If a specification statement is to be testable then one of the properties it must have is:
   (a) Tenacity
   (b) Probability
   (c) Usability
   (d) Traceability
   (e) None of the above

5. Which of the following is not a software life-cycle model?
   (a) Waterfall
   (b) Rapid prototyping
   (c) Synchronize and stabilize
   (d) Evolutionary
   (e) All are software life-cycle models

6. In which of the following circumstances might the waterfall model be an appropriate software life-cycle model to use?
   (a) Large-scale, new products
   (b) For products utilizing an open architecture with a complex user interface
   (c) Real-time systems
   (d) Mission-critical systems
   (e) Follow-on releases on mature systems

7. Which of the following software process models specifically address risk analysis and risk resolution?
   (a) Waterfall
   (b) Rapid prototyping
   (c) Synchronize and stabilize
   (d) Revolutionary model
   (e) Spiral model

8. A common solution used in real-world situations by knowledgeable programmers to deal with a component that encounters an error during processing is to:
   (a) Return an error value to the callee
   (b) Throw an exception
   (c) Set a status flag
   (d) All of the above
   (e) None of the above

9. A common metric for estimating the effort to develop a software product is:
   (a) Function Points (FP)
   (b) Lines of Comments (LOC)
   (c) COCOMO
   (d) Bytes
   (e) None of the abov

10. The aim of the synchronization stage of the synchronize-and-stabilize software life-cycle model is to:
    (a) Repair faults found in any earlier releases
    (b) Freeze change requests so that the build can be stabilized
    (c) Draw up the specification document
    (d) Release the current version(s) of the product to the clients based on the versions and releases installed at their various sites
    (e) Put the partially completed components together and test and debug the resulting product

11. The use of traceability in requirements tools helps to:
    (a) Debug programs following the detection of run-time errors
    (b) Determine the performance of algorithm implementations
    (c) Identify, control, and track requirements changes
    (d) All of the above
    (e) None of the above

12. Methods of requirements gathering include:
    (a) Interviews using structured techniques and close-ended questions
    (b) Interviews using structured techniques and open-ended questions
    (c) Questionnaires
    (d) An analysis of forms used by clients
    (e) All of the above

13. Use-case diagrams:
    - (a) Describe what the system should do
    - (b) Display object interactions arranged in a time sequence
    - (c) Are collections of objects with the same characteristics
    - (d) All of the above
    - (e) None of the above

14. A finite state machine consists of five parts: a set of states J, a set of inputs K, a transition function T that specifies the next state given the current state and the current input, the initial state S, and the set of final states F. What is required to determine the next state?
    - (a) The initial state and the transition function T
    - (b) The final state and the transition function T
    - (c) The current state, the current input and the transition function T
    - (d) The current state, the transition function T and the final state.
    - (e) Sets J, K and T

15. Which of the following is not the task for requirements validation and verification?
    - (a) Check for completeness
    - (b) Check for consistency
    - (c) Check for feasibility
    - (d) Check for traceability
    - (e) Check for adaptability

2. Requirements (20%)

Stating software requirements precisely is notoriously difficult. One of the main problems is getting everyone to agree on the exact meaning of the requirements statements. In other words, we want to develop requirements in such a way that, upon completion of the system, both the software engineer and the client can agree on whether or not a specific requirement has been met.

Problem: Examine the following four requirements. If there are problems with the requirement, identify the problems and give an example of how the requirement should be stated. Analyze each requirement independently—not as a group.

Grading: 5% for each requirement: 3% for identifying the problem (if any); 2% for each restatement example (if needed).

(a) The system should be secure. We don't want it hacked!

(b) The user interface has to be easy to use; we've been getting a lot of complaints from our beta testers on the new version. But we can't remove any of the new functionality either.

(c) During emergency conditions, the system shall suspend all non-critical functions.

(d) Our developers tell us the program needs to be maintainable.

3. Design (20%)

Design is a key component of the overall software lifecycle. Good design contributes to the construction of elegant and bug-free software. During high-level design, once the overall system organization has been chosen, one needs to make a decision on the approach to be used in decomposing sub-systems into modules. Sub-systems are composed of modules and have defined interfaces, which are used for communication with other subs-systems. A module is a lower-level artifact than a sub-system that is composed from a number of other simpler system components.

Problem:

(a) Describe *object-oriented decomposition*. Clearly identify the advantages and disadvantages of this approach to sub-system decomposition. Given an example of object-oriented decomposition for a hypothetical system's design.

(b) Describe *function-oriented pipelining*. Clearly identify the advantages and disadvantages of this approach to sub-system decomposition. Given an example of function-oriented pipelining for a hypothetical system's design.

Grading: (a) 10%; (b) 10%;

Note: Use the blank sheet of paper on the next page as needed.

4. Construction (20%)

Trees are one of the essential data structures used in software construction.

Problem: Consider a binary tree with integer values in its nodes.

(a) Write the C/C++ class/struct(s) that implement(s) such a binary tree.

(b) Construct an elegant and efficient **recursive** C/C++ function that takes such a binary tree and returns the number of nodes with integer values that are odd.

Notes:

- Make sure your solution is constructed clearly and idiomatically, so that it adheres to the commonly accepted definition of good coding style.
- Your solution cannot make use of the Standard Template Library (STL).
- Be sure to properly comment your program; explain how the solution works and why you selected particular algorithm(s) and data structure(s).
- Include defensive programming techniques in your solution.
- Use the other side of the paper as needed.

Grading: Correctness: 10%; Documentation: 5%; Style: 5%

5. Testing (25%)

There are many different types of strategies commonly used for software testing. However, the strategies can be broadly classified into two distinct categories: "Black Box" testing and "White Box" (aka "Glass Box") testing.

Problem:

(a) Describe "Black Box" testing and "White Box" testing. Include in your description an explanation of the difference(s) between the two categories, and the advantages and disadvantages of each.

(b) Comment on who should do "Black Box" testing: the developer or someone else.

Grading: (a) Description of each software testing category: 10% each; (b) 5%.

Note: Use the blank sheet of paper on the next page as needed.

6. Maintenance & Evolution (25%)

Maintenance is the act of modifying a program after system deployment. Following good software practice doesn't negate the need for maintenance—just its severity. In fact, maintenance is the most common form of software evolution.

Problem:

(a) Describe the three (3) most common types of software maintenance. Be sure to provide examples of each type of maintenance using realistic scenarios.

(b) Discuss how software maintenance differs from software construction.

Grading: (a) 5% for each description (15%); (b) 10%.

Note: Use the blank sheet of paper on the next page as needed.

7. Process (25%)

Of the various software process models that have appeared in the literature, it can be argued that the Software Engineering Institute's "Capability Maturity Model for Software" (SEI SW-CMM®) has had the most impact for large organizations.

Problem:

(a) Describe the SW-CMM. Explain each level. Provide a diagram.

(b) Discuss the advantages and disadvantages of this process improvement model.

Grading: (a) 15%; (b) 10%.

Note: Use the blank sheet of paper on the next page as needed.