Comprehensive Exam (Spring 2006)

SOFTWARE ENGINEERING

Friday, March 17th, 2006; 10:00am – 11:30am

Instructions

- Write the last four digits of your student identification number in the space below.
- This exam consists of 16 pages (including this cover).
- Answer any four (4) of the following seven (7) questions. Each question is of equal value (25%). Circle the questions that you want graded:

  1 2 3 4 5 6 7

  (If you leave this blank, questions 1 through 4 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.	Requirements	25	
2.	Design	25	
3.	Construction	25	
4.	Testing	25	
5.	Maintenance & Evolution	25	
6.	Management	25	
7.	Process	25	
	Total	**100**	

1. Requirements (25%)

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. A very common way of expressing functional requirements is to use prose, with each requirement expressed as a simple sentence describing an action or behavior expected of the system. Such sentences are typically referred to as the "shalls," as in "The system shall process Web server log files."

To improve the chances that a requirement will not be misinterpreted, good practice states that each requirements statement should be accompanied by:

- The rationale for the requirement;
- The source of the requirement;
- The acceptance criteria for ensuring the final product satisfies the requirement;
- Potential conflicts with other requirements; and
- Traceability back to a higher level specification (e.g., Use Case, Business Event).

Problem: For each of these 5 accompanying components:

(a) Explain how each component aids understanding of the requirements statement.

(b) Provide a specific example of how a problem could occur if the component were omitted from the requirements statement (5 problems in all).

Grading: (a) 3% for each clear explanation (15%); (b) 2% for each example (10%).

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

2. Design (25%)

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.

(c) Discuss the applicability of the Unified Modeling Language (UML) as a design aid for each of these two sub-system decomposition techniques.

Grading: (a) 10%; (b) 10%; (c) 5%.

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

3. Construction (25%)

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 even.

Notes:

- Make sure your solution is constructed clearly and idiomatically, so that it adheres to the commonly accepted definition of good coding style.
- 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: 15%; Documentation: 5%; Style: 5%

4. Testing (25%)

Software testing plays a key role in ensuring overall product quality and effectiveness. One of the difficulties in testing is knowing when to stop. While one would like as complete a testing procedure as possible, like all software engineering activities, testing must take place within a realistic context of limited resources.

Problem:

(a) Explain the definition of "coverage" in software testing.

(b) Describe three different types of coverage that can be measured. For each one, describe a type of bug that you would be certain to find with this type of coverage and describe a type of bug that you might miss even if you achieved 100% of this type of coverage.

Grading: (a) 7%; (b) 6% for each description (18%).

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

5. 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 evolution.

Problem:

(a) Describe the three 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 development.

Grading: (a) 5% for each description; (b) 10% for a clear discussion of the main issues.

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

6. Management (25%)

Estimating the cost and effort required for a particular task in a software project is an important management activity.

Problem:

(a) Describe two *classes* of productivity measures that are commonly used to aid cost and effort estimation. Note that in this context, a *class* refers to a generic category of productivity measure, not to a specific type.

(b) Give specific examples of each class of productivity measure that are commonly used in practice.

(c) Comment on the advantages and disadvantages of each example.

Grading: (a) 5% for each class description (15%); (b) 5% for each example (10%); (c) 5% for a discussion of the relative advantages and disadvantages of each example.

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

7. Process (25%)

Process models are useful instruments to help software engineers manage large-scale projects. For example, the models can provide guidance in the context of improving software quality. Three of the software engineering process models commonly discussed are the waterfall model, the evolutionary model, and the spiral model.

Problem:

(a) Clearly explain each of these three process models. Draw diagrams of their phases. Describe the relative advantages and disadvantages of each model.

(b) Under which circumstances would one choose the waterfall model over the other models? Clearly explain why. Provide a realistic example to support your answer.

Grading: (a) 5% for each process model (including diagrams and discussion); (b) 10%

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