Florida Institute of Technology

Department of Computer Sciences

Comprehensive Exam (Spring 2009)

SOFTWARE ENGINEERING

Tuesday, March 24th, 2009; 10:00am – 11:30am

Instructions

- Write your student identification number in the space below.
- This exam consists of 17 pages (including this cover).
- **The exam consists of two parts:**
  - **You must answer Questions 1, 2, 3, 4 and 5.**
  - **Answer any one (1) of Questions 6, 7, or 8**

  ➔ Circle the one question that you want graded:

  6 7 8

  (If you leave this blank, Question 6 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	10	
2.	General Knowledge	5	
3.	Requirements	20	
4.	Design	20	
5.	Construction	20	
6.	Testing	25	
7.	Maintenance & Evolution	25	
8.	Process	25	
	Total	**100**	

1. General Knowledge - True/False (10%)

Grading: 1% for each correct answer; total 10 questions.

Note: Circle either **T** (true) or **F** (false) for each question. Circling *both* **T** and **F** will result in 0 points for that question.

(a)	Software engineering is only concerned with the technical activities of the software life cycle, such as requirement, design, construction, testing, and maintenance & evolution.	T	F
(b)	It is generally accepted that one cannot have weak software processes and still create high-quality software products.	T	F
(c)	The use of assertions in source code is an effective means of implementing black-box testing.	T	F
(d)	Functional requirements are more important than non-functional requirements. As long as functional requirements are met, meeting non-functional requirements are optional.	T	F
(e)	One can overload whitespace as an operator in C++.	T	F
(f)	In UML diagrams, use-case actors always refer to people; they can never be hardware devices.	T	F
(g)	Requirements Specification refers to a complete list of what the customer expects the system to do, from the user's point of view.	T	F
(h)	Linked lists can be implemented using arrays.	T	F
(i)	The three most common types of software maintenance are contemplative, adaptive, and perfective.	T	F
(j)	Extreme Programming (XP) is a heavyweight software process model.	T	F

2. General Knowledge - Multiple Choice (5%)

Grading: 1% for each correct answer; total 5 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. 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

2. 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

3. 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

4. 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

5. 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

3. Requirements (20%)

Getting software requirements right is notoriously difficult. One of the main problems is getting everyone to agree to the same thing. In other words, developing a common understanding of the problem, from both a user (requirements definition) and an engineering (requirements specification) perspective.

Problem:

(a) Describe three techniques for representing requirements specifications.

(b) What is requirements *traceability* and why is it important?

Grading: (a) 5% for each clear explanation (15%); (b) 5%.

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

4. 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.

5. Construction (20%)

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

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

```
struct Node {
  int value;
  Node *left, *right;
};
```

(a) 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.

(b) What is the runtime performance of your recursive solution from part (a) in $O(n)$ notation? Be sure to explain your answer. Document all assumptions.

(c) Redo part (a) but make your C/C++ function **iterative**.

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

6. 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.

7. Maintenance & Evolution (25%)

Evolution is an important activity of the software lifecycle. In the 1980s, Lehman and Belady examined the growth and evolution of a number of large software systems. Based on their measurements from these studies, they proposed a set of five hypotheses, known as "Lehman's Laws," concerning system change. They claim that these "laws" are invariant and widely applicable.

Problem: Describe all five of Lehman's Laws. Include commentary on the validity of each of these laws as applied to today's software-intensive systems.

Grading: 5% for each description and commentary.

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

8. 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.