Comprehensive Exam (Fall 2004)

SOFTWARE ENGINEERING

Friday, October 28th, 2004; 10:00am – 11:30am

Instructions

- Write the last four digits of your student identification number in the space below.
- This exam consists of 14 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.	Process	25	
7.	Management	25	
	Total	**100**	

1. Requirements (25%)

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: Describe three techniques for representing requirements specifications. Give an example of two of the three techniques for the same hypothetical system.

Grading: 5% for each clear explanation (15%); 5% for each example (10%).

2. Design (25%)

The Unified Modeling Language (UML) is the *de facto* standard for modeling modern software applications. It is particularly well-suited to object-oriented systems.

Problem: A software product is to be installed to control *n* elevators in a building with *m* floors. Your task is to design the system that controls the movement of the elevators between floors according to the following constraints:

- Each elevator has a set of *m* buttons, one for each floor. The buttons illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited.
- Each floor, except for the first floor and the top floor, has two buttons, one to request the up elevator and one to request the down elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor and then moves in the desired direction.
- When the elevator has no requests, it remains at its current floor with its doors closed.

As a first step in your design, consider the following scenario:

1. User A presses the Up floor button at the 3rd floor to request the elevator, wishing to go to 7th floor.
2. The Up floor button is turned on.
3. An elevator arrives at 3rd floor. It contains User B who entered the elevator at the 1st floor and pressed the elevator button for the 9th floor.
4. The Up floor button is turned off.
5. Elevator doors open. User A enters.
6. User A presses button for 7th floor.
7. 7th floor elevator button is turned on.
8. The elevator doors close.
9. The elevator travels to 7th floor.
10. 7th floor elevator button is turned off.
11. The elevator doors open to allow User A to exit the elevator.
12. The timer starts. User A exits.
13. Elevator doors close after timeout.
14. Elevator goes to the 9th floor with User B.

Problem: Provide UML use case, sequence, and class diagrams for this scenario.

Grading: Use case diagram (5%); Sequence diagram (12%); Class diagram (8%).

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

3. Construction (25%)

Imagine that for some reason that the **<stack>** container adapter was not available in the Standard Template Library (STL). You has been tasked with creating a simplified version of **<stack>** matching the following C++ interface:

```
template <class Item>
class Stack {

  private:
    // implementation-dependent code

  public:
    Stack(int);
    int empty() const; // 1 = empty, 0 otherwise
    void push(Item);
    void pop();
    Item top() const;
};
```

Problem: Implement a version of **<stack>** matching this interface in C++ using *singly-linked lists* as the underlying data structure.

Notes:

- *Your code must not rely on any aspect of the STL.*
- The argument to the **Stack** constructor specifies the maximum size of the stack.
- If an error is detected, call the predefined routine **error(char *)**. This routine will print the error message to standard error and terminate the program.
- Make sure your solution is constructed clearly and idiomatically, so that it adheres to the commonly accepted definition of good coding style.
- Use the other side of the paper as needed.

Grading: Correctness: 20%; Style: 5%

4. Testing (25%)

Certification is an outside source's endorsement of a system's correctness. It is often granted by comparing the system to a predefined standard of performance. For example, the U.S. Department of Defense may certify an compiler after testing it against a long list of functional specifications.

Problem: In the terminology of system testing, is such a test (a) a function test? (b) a performance test? (c) an acceptance test? (d) an installation test? For each question, explain why or why not. Comment on the validity of such an outsourced system test.

Grading: 5% for a clear answer to each question: 5% for your commentary.

5. 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. Comment on the validity 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.

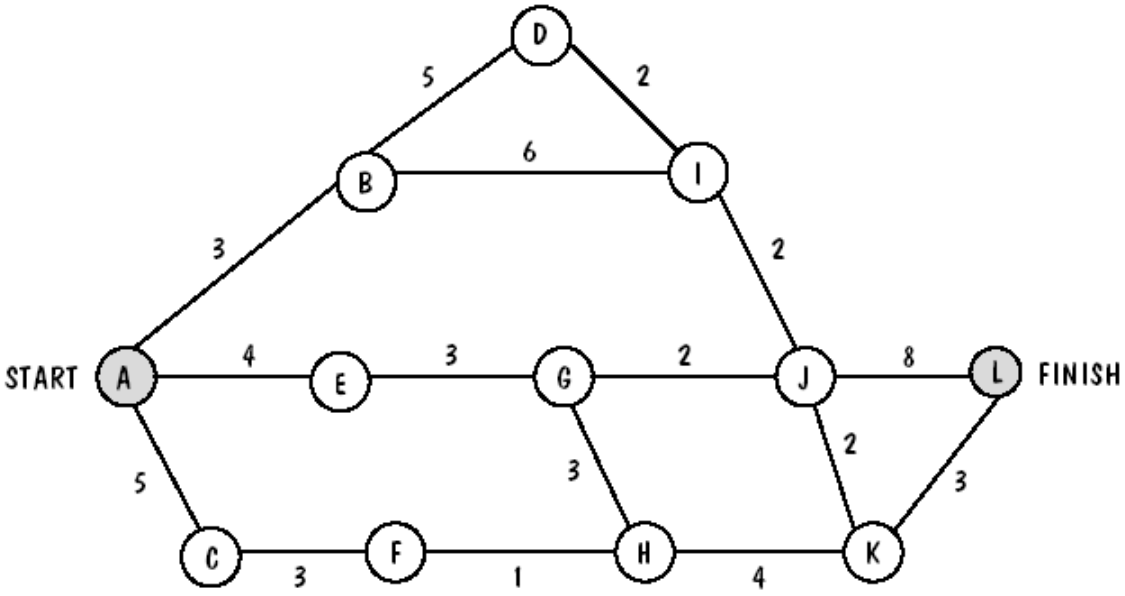
6. Management (25%)

Project scheduling is an important part of delivering software in a timely manner. The figure below is an activity graph for a software engineering project. The number corresponding to each edge of the graph indicates the number of days required to complete the activity represented by that branch. For example, it will take 4 days to complete the activity that ends in milestone E.

Problem: For each activity, list its precursors, and compute the earliest start time, the latest start time, and the slack. Then, identify the critical path in the schedule.

Grading: Precursors (7%), Start/End/Slack Time (16%), Critical Path (2%).

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: Describe the SW-CMM. Explain each level. Discuss the advantages and disadvantages of this software process improvement model.

Grading: Description (15%); Advantages & Disadvantages (10%).

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