Comprehensive Exam (Spring 2005)

SOFTWARE ENGINEERING

Friday, March 18th, 2005; 10:00am – 11:30am

Instructions

- Write the last four digits of your student identification number in the space below.
- This exam consists of 12 pages (including this cover).
- Answer any four (4) of the following six (6) questions. Each question is of equal value (25%). Circle the questions that you want graded:

  1 2 3 4 5 6

  (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, lucid, 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.	The Software Lifecycle	25	
2.	Software Requirements Analysis	25	
3.	Software Testing	25	
4.	Design & Implementation of Software	25	
5.	Software Project Estimation & Planning	25	
6.	Software Reviews	25	
	Total	**100**	

1. The Software Lifecycle (25%)

A basic software engineering lifecycle model consists of the phases Requirements, Design, Construction (Implementation), Testing, and Maintenance.

Problem: Which of these phases is the most EXPENSIVE phase – justify your answer.

Grading: 5% for identifying correct phase; 20% for justification.

2. Software Requirements Analysis (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. In other words, we want to develop requirements in such a way that, upon completion of the system, both the developer and the client can agree easily on whether or not a specific requirement has been met.

Problem: Examine the following 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: 3% for identifying the problem (if any) (15%); 2% for each example (if needed) (10%).

A. The system shall provide an easy-to-understand user interface.

B. When the pressure exceeds 115, the system shall alert the pilot.

C. The processing time shall be fast enough to satisfy all users of the system.

D. During emergency conditions, the system shall suspend all non-critical functions.

E. The system shall always be available.

3. Software Testing (25%)

Software is often made from independent modules of code that have to be integrated together to form a complete application.

<u>Problem</u>: Describe "Top-Down Software Integration and Testing" and "Bottom-Up Software Integration and Testing." Outline the principals of each approach (including any additional requirements to assist with testing) as ~~well some strengths and weakness of each approach~~in which circumstances each method may be used.

<u>Grading</u>: 10% for Top-Down description; 10% for Bottom-Up description; 5% for ~~strengths and weaknesses~~ when method may be used.

4. Design & Implementation of Software (25%)

Getting the design of a software system is an important consideration when it comes to performance, testability, and maintenance. There are several factors to be considered with regard to a design, two of these are cohesion and coupling.

Problem: Define the concept of module coupling and cohesion. Why are they considered an important software attribute. Within each concept there is a spectrum of ways modules can be related. Identify some different examples (in terms of levels or complexity in this spectrum) of both coupling and cohesion

Grading: 10% for defining concepts; 10% discussion of why they are important; 5% for examples.

5. Software Project Estimation & Planning (25%)

A typical technique for estimating the cost and schedule for a software project is to begin with an estimate of the size of the project expressed (in most cases) in numbers of lines of code. This estimate is then used as input to an empirically developed equation (COCOMO or SLIM, for example) to produce estimates of labor and schedule for the project.

Problem: A criticism of using estimates of the number of lines of code as a basis for project estimation is the inability, early in the project, to develop an accurate estimate of the size. Identify and discuss 3 reasons that these early size estimates may be wrong and, for each reason, identify a strategy to correct the problem.

Grading: 2% for each reason (6%); 3% for each discussion (9%); 3% for each strategy (9%); 1% for writing style.

6. Software Reviews (25%)

Because software processes are error-prone, the software development cycle is characterized by a sequence of technical reviews designed to verify/validate both the intermediate and final products of the processes.

Problem: Describe two objectives in holding a Software Technical Review, e.g., code review, inspection? What preparations, if any, are needed prior to a review? What are some guidelines for conducting a review? What information should be recorded during and at the conclusion of a review?

Grading: Objective Descriptions (5% each) (10%); Preparations (5%); Guidelines (5%); Information Recorded (5%)