Florida Institute of Technology

Department of Computer Sciences

Comprehensive Exam (Fall 2005)

SOFTWARE ENGINEERING

Friday, October 28th, 2005; 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. 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 five 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%).

Note: Use the blank sheet of paper on the next page as 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) The new system needs to be interoperable with our other systems and those of our partners as well. Our competitors too I suppose.

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

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

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. There are several timeless guidelines that have been used by software designers over the years, including:

1. Iterative enhancement
2. Stepwise refinement
3. Information hiding

Problem:

(a) Describe software design by placing it in context of the overall software lifecycle. Your answer should include a discussion of issues such as the different types of design (e.g., high-level versus low-level), different design paradigms (e.g., object-oriented), and different design representations (e.g., UML).

(b) Explain the three timeless software design guidelines shown above.

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

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

3. Construction (25%)

The **uniq** program on UNIX removes duplicate lines from sorted data. Suppose, however, you need to remove duplicate lines from a data file (which might not be sorted), but that you wish to preserve the line ordering. A good example of this might be a shell history file. The history file keeps a copy of all the commands you have entered, and it is not unusual to repeat a command several times in a row, or several times per session. Occasionally you might wish to compact the history file by removing duplicate entries. Yet it is desirable to maintain the order of the original commands.

Problem: Construct an elegant and efficient C++ or Java program that implements this enhanced functionality. Document how your solution works, and the rationale behind your selection of algorithm(s) and data structure(s).

Input: A text file containing the shell commands. Each line is terminated by the newline character (`'\n'`). Assume that the number of characters per line is usually less than 256. Assume that the number of lines in the input is usually less than 1,000. Assume that there are no blank lines in the input.

Output: A printout of the compressed history file in this format:

command # count

where *command* is the shell command, *count* is the number of times the command appeared in history file, and the '#' character separates the two (with a single space on either side of the '#'). Line ordering is important – it must be the same as the input.

Example: Given the following history file data as input:

```
ls
cat foo
date
ls
ls
cd
cd
ls
cd /tmp; ls
cd cs10/as1
g++ -Wall a2q2.cc -o a2q2
echo darn compiler
cd cs10/as1
cd cs10/as1
cd ~/cs10/a1
g++ -Wall a2q2.cc -o a2q2
a2q2
cd
cd
cd
```

Your program will produce exactly the following output:

```
ls # 4
cat foo # 1
date # 1
cd # 5
cd /tmp; ls # 1
cd cs10/as1 # 3
g++ -Wall a2q2.cc -o a2q2 # 2
echo darn compiler # 1
cd ~/cs10/a1 # 1
a2q2 # 1
```

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).
- Provide citations to references you may have used in constructing your solution.
- Input to the program will come from standard input. Output must be to standard output. Do not prompt for input, nor produce spurious output.
- Use the other side of the paper as needed.

Grading: Correctness: 15%; Documentation: 5%; Style: 5%

4. 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. (20%)

(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) 10% for well-written discussion.

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%)

There is a lot of jargon associated with software engineering. This is particularly true for software engineering project management. Nevertheless, understanding such terms is important for properly managing a software project, for instance in terms of scheduling.

Problem: Explain the terms shown below in the context of project management.

Grading: 5% for each clear explanation.

(a) Output

(b) Outcome

(c) Milestone

(d) Deliverable

(e) Critical Path

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.