CS422 Software Engineering Principles **Final Exam Study Guide**

The final exam will be comprehensive covering all the materials discussed in class up to the last lecture. This study guide is intended to help focus your study time. There is no guarantee that this study guide is complete.

You are responsible for all of the material included in the slides. In some (most) cases this material can be found in the Sixth Edition of the Somerville textbook. The Brooks book *Mythical Man Month* (Anniversary Issue) is also fair game. For example, refresh your memory on conceptual intregrity and review the study/exam questions on this topic. The lecture slide chapters (which correspond to the fifth edition) that were covered include the following (1-25 excluding 10 and 11):

1. *Introduction*
2. *Computer Based System Engineering*
3. *Project Management*
4. *Requirements Engineering*
5. *Requirements Analysis*
6. *System Models*
7. *Requirements Definition and Specification*
8. *Software Prototyping*
9. *Formal Specification*
10. *Software Design*
11. *Architecture Design*
12. *Object Oriented Design*
13. *Function Oriented Design*
14. *Real-Time Systems Design*
15. *User Interface Design*
16. *Software Reliability*
17. *Programming for Reliability*
18. *Software Reuse*
19. *Safety Critical Software*
20. *Verification and Validation*
21. *Defect Testing*
22. *Static Verification*

You should study the following materials:

Exams one and two keys (numerous questions ~15% were taken from the past exams)

All the study questions and keys

In regards to the lecture material from Somerville, listed below are some key issues and topics you should review:

1. Describe *hard* versus *soft* real-time systems and degraded operation. Name three major components of a real-time executive. Describe periodic versus aperiodic stimuli.What are three embedded systems characteristics?
2. Describe (compare and contrast) the different behaviorial models from the chapter on systems modeling.
3. How does prototyping support requirements engineering (explain)?
4. What constitutes well-formed modules (you should discuss coupling, cohesiveness, cohesion and binding)?
5. How is conceptual integrity achieved (give the three basic ideas)?
6. What are configuration and fault managers? Know the fundamental actions in fault tolerance.
7. How would you characterize monitoring and control systems?
8. Describe fault tolerance versus fault (error) avoidance.
9. Know the difference between software development for and with reuse (also recall how application portability is a form of reuse). Know the different portability dependency types and how the are encapsulated into their respective APIs.
10. What are the advantages of software development with reuse?
11. Know what is information handling and how it avoids faults.
12. Describe defensive programming (three principle things [what are they and how do they work]).
13. What are the principles of information hiding and encapsulation?
14. Why are information hiding and encapsulation used in programming for reliability?
15. How does structured programming help error avoidance?
16. How does data typing help error avoidance and give two examples of languages that support strong typing (i.e., the ones given in our text)?
17. Know what is safety critical software.
18. Know the difference between defect testing and debugging.
19. Give the slightly different meanings of the terms *failure* and *defect* and *fault* and *operational profile* as I was telling you in class. Try to put your definitions in the context of software and hardware
20. What does testing do and what does is not do (in the context of Chapters 22 - 24)?
21. You should be able to differentiate all the different types of testing (i.e., recognize what and how different testing strategies are conducted). What is the basic idea behind equivalence partitioning (i.e., look at the figure I showed you).
22. You need to remember how to compute cyclomatic complexity and the idea behind measures of test coverage.
23. Know the definitions of the different interfaces (parameters, shared memory, procedural, message passing).
24. Remember: A successful defect test is a test, which causes a program to behave in an anomalous way, and tests show the presence of defects not the absence of defects.
25. Know about fault trees and hazard analysis (what is the difference and how they are applied in practice). How is risk (and cost) involved in the process?
26. Know your process models (including extreme programming www.extremeprogramming.org) and how to differentiate among them.

Make sure you know the meaning of these terms:

1. Extreme Programming
2. Forward engineering
3. Pre/post conditions
4. Instantiation
5. Implementation profile
6. Legacy system
7. Software project management
8. Target machine
9. Exception handling
10. Domain specific architectural models
11. Syntax
12. Formal method
13. Verification
14. Fault
15. Expert systems
16. Extensible
17. Objects (in Object Oriented Design)
18. Reliability
19. Real-time system
20. Structure chart
21. COTS
22. Process metrics
23. Simulation
24. Integration testing
25. Regression testing
26. Blackbox testing
27. Fault tolerance (or tolerant)
28. Fault detection
29. Recovery block
30. Waterfall model
31. Software prototyping
32. Safety critical systems
33. Systems engineering
34. Software quality assurance
35. Automatic programming
36. Reverse engineering
37. Predicate
38. Inheritance
39. Operational profile
40. Software maintenance
41. Software configuration management
42. Vehicle machine

43. Interrupt handling
44. Walk-through
45. Semantics
46. Software life cycle
47. Validation
48. Failure
49. Bespoken systems
50. Adaptable
51. Functions (in Function Oriented Design)
52. Availability
53. Transaction system
54. P-Spec (or Pseudo code)
55. Portability
56. Product metrics
57. Reusability
58. Partition testing
59. Mutation testing
60. White box testing
61. Fault avoidance
62. Specification errors
63. N-version programming
64. Boehms spiral model
65. Application system portability
66. Requirements analysis
67. Software engineering
68. GUI design principles