CS 458 - Final Exam Review Suggestions - Fall 2017

last modified: 2017-12-05

- Based on suggestions from Prof. Deb Pires from UCLA: Because of the research-supported learning potential when students study together and explain concepts to one another, you again can receive (a maximum) ***5 POINTS BONUS*** on the Final Exam if you do the following:
  - set up and/or attend an exam study session with at least one other CS 458 student (but it can involve as many other CS 458 students as you would like!)
  - for **YOU** to receive the bonus, **YOU** SEND an e-mail to me, sent on the **same day** as the study session (which needs to take place **before 8:00 am on Thursday, December 14**) in which you:
    - use as the e-mail `Subject: CS 458 - Final Exam bonus`
    - **briefly DESCRIBE (a sentence or two is fine) what** you **covered** at the exam study session
    - INCLUDE a **picture** of everyone involved in the study group
    - (**EACH** person who wants the 5 point bonus **should send an e-mail message to me** containing this `Subject:` line AND these **TWO** parts.)
- You are responsible for material covered in assigned reading, class sessions, and homeworks; but, here's a quick overview of especially important material.
  - (Homework problems are good sources of ideas for exam questions, of course, as are clicker questions, reading quiz questions, and posted notes -- BUT the posted notes are not "complete", and so **you'll need to reference the course textbooks as well**.)
- You are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have **handwritten** whatever you wish on one or both sides. This paper must include your name, it must be handwritten by you, and it will **not** be returned.
  - Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer.
- (Note that final exams are **not** returned, although you can come by my office after they are graded to look at yours, if you would like. I'll keep them on file for at least two years.)
- The Final Exam is CUMULATIVE!
  - If it was fair game for Exam 1, it is fair game for the Final Exam.
  - Thus, using the posted review suggestions for Exam 1 in your studying for the Final Exam would be a good idea. (It is still available from the public course web page, under "Homeworks and Handouts".)
  - studying your Exam itself would be wise. There may be similar styles of questions on the Final Exam.
- This will be a pencil-and-paper exam, primarily involving concepts, also with some reading and writing of code, and some problems related to concepts discussed.

Verification and Validation

- What is verification? What is validation? What is the difference between these? Be able to give an example of each; be able to distinguish between examples of each.
  - Why are both needed? Why isn't either one alone sufficient? What does each help to ensure?

More on Git and GitHub

- You are expected to be more familiar with Git commands, and with the use of GitHub, than at the time of Exam 1.
- How can you push up JUST a branch from a Git local repository to the Git remote repository it was cloned from?
- How can you make a local clone of a local Git repository (that is, make a clone of a Git repository on the same computer)?
- How can you extract an older version of your Git repository and place it in a new branch, such that you do not affect your current master branch?

JUnit and Unit Testing

- Given a Java method's purpose, you should be able to write a testing method that could appear in a JUnit test class that would pass if that method were written correctly, such that that testing method would be run automatically in DrJava if both the class being tested and the JUnit testing class class were open (and saved and compiled) and you click the "Test" button.
- Be able to understand/read/write appropriate `assertEquals` and `assertTrue` method calls.
- What features must a method have within a JUnit test class so that such a method will be run when you click the "Test" button in DrJava? (That is, what annotation should precede a test method? What is the return type for a test method? How many parameters does a test method have?)
  - and, what is the naming convention we used for such a test method?
- What is special about the method `setUp` within a JUnit test class? What annotation should precede it? When will this be called? What might you use it for? You should be able to understand/read/predict the behavior of a JUnit test class including this method.
- Given a simple Java class and a JUnit test class testing that simple Java class, you should be able to reason about what would happen if both were open, saved, and compiled in DrJava and the "Test" button is clicked; this might include describing what would appear in the DrJava "Test Output" window, and what would appear in the DrJava Interactions window.
- If you were doing TDD, *when* would you write a testing method (in a JUnit test class) for a method of a class you are developing? *When* would you write the method of that class?

"Mythical Man-Month" - Chapter 6 - Passing the Word

- What does Brooks consider the manual, or written specification, to be? Of whom is it the "chief product"? What should it include? What are some characteristics that Brooks argues this should have?
  - For consistency to be maintained, how many should be involved in writing it?
- What are some of the advantages and disadvantages of formal definitions? ...of prose definitions?
- What is it essential that the "puzzled implementer" be encouraged to do as implementation proceeds?And what does Brooks suggest should be done with the results of this? (And how might this be done now?)
- What does Brooks describe as "the project manager's best friend" at the end of this chapter? How does this relate to Jalote Chapter 8?

"Mythical Man-Month" - Chapter 7 - Why Did the Tower of Babel Fall?

- Why does Brooks argue that the Tower of Babel project failed (even before technological limitations)?
- How does Brooks suggest that teams communicate in this chapter?
- What is the project workbook, according to Brooks? What should be part of this? Why?
- How was the project workbook handled in the OS/360 project? What are some of the means by which this would/could be handled today?
- What does Brooks describe as the distinction between a project's producer and its technical director? What are the three relationships he discusses between these?

"Mythical Man-Month" - Chapter 8 - Calling the Shot

- According to a study mentioned in this chapter, how did full-time programmers tend to mis-estimate the time it would take them to complete work? Why?
- In the different studies cited in this chapter, what were some of the factors that affected productivity on different projects?
- What does Brooks mean when he says "Productivity seems constant in terms of elementary statements"? What are some possible implications of this?

"Mythical Man-Month" - Chapter 9 - Ten Pounds in a Five-Pound Sack

- According to Brooks in this chapter, "Like any cost, size itself is not bad, but ..." ... what?
- Brooks argues that every project needs a notebook full of good subroutines or macros for what four classic categories of tasks? And for what two varieties of each?
- What does Brooks argue that "lean, spare, fast programs" are almost always the result of?
- What does Brooks describe as "the essence of programming" in this chapter?

"Mythical Man-Month" - Chapter 10 - The Documentary Hypothesis

- What is Brooks' hypothesis in this chapter?
- What does Brooks believe are the critical documents for a software project?
- Why does Brooks argue that having formal documents -- writing down decisions -- is essential?
- What does Brooks believe is the manager's major task?

"Mythical Man-Month" - Chapter 11 - Plan to Throw One Away

- What is Brooks' principal recommendation in this chapter? Why? How does this relate to the software development process models discussed in Jalote Chapter 2?
- What are some ways that Brooks suggests for planning a system for change? ...for planning the organization for change?
- What percentage (at least!) of the cost of project development is the cost of maintaining a widely-used program?
- Be familiar with the bug occurrence cycle/curve discussed in this chapter, and its implications.
- What does Brooks cite as the fundamental problem with program maintenance? (Or: what is the chance that

fixing a defect introduces another?) What, then, is Brooks' cheery conclusion of this chapter?

"Mythical Man-Month" - Chapter 12 - Sharp Tools

- What are some of the "sharp tools" needed for a programming project?
- When does Brooks describe system debugging as having always been/traditionally been done? Why does he believe this has been the case?
- Be familiar with Brooks description and concept of playpen/system integration sub-library/current version sub library. Why does he think this worked so well? What can be some parallels of this using software and services such as Git and GitHub?
- What are some of the ways that use of a (suitable) high-level language benefits a project?

"Mythical Man-Month" - Chapter 13 - The Whole and the Parts

- What are some of the "most pernicious and subtle bugs", according to Brooks? What can help with preventing these?
- What does Brooks consider to be the most important new programming formalization of the [1965-1975] decade? Why?
- What is scaffolding? How much scaffolding is a "not unreasonable" amount, according to Brooks?
- what is top-down design? what is structured programming? What is the relationship between these? How can each help to avoid bugs (as well as make those bugs that still occur easier to find)?
- What does Brooks recommend with regard to system debugging? (When should one begin system debugging?)
- What is regression testing?

"Mythical Man-Month" - Chapter 14 - Hatching a Catastrophe

- According to Brooks, how does a project get to be a year late? In a related vein, what does he argue is "harder to recognize, harder to prevent, and harder to make up" than major calamities?
- In this chapter, what does Brooks recommend as the first step to controlling a big project on a tight schedule? Why?
- Why are milestones that are concrete, specific, measurable, and defined with knife-edge sharpness preferred over those that are not?
- Why does Brooks believe that hustle is essential for a successful programming team?
- How can a critical path/PERT chart help in keeping a project on track?
- What does Brooks suggest bosses do to make sure they get the "true" picture of how a project is going?

"Mythical Man-Month" - Chapter 15 - The Other Face

- What does Brooks consider to be "the other face" of a written computer program in this chapter?
- How does Brooks argue that most documentation fails? What are some of the aspects that he suggests should be included?
- How do Brooks' suggested test cases compare to those described in Jalote Chapter 8?

- Why does Brooks argue that "The flow chart is a most thoroughly oversold piece of program documentation"? What kind of flow chart does he believe is very handy?
- What are some of the programming practices that Brooks argues can make a program "self-documenting" (at least to some degree)?

"Mythical Man-Month" - Chapter 19 - *The Mythical Man-Month* after 20 Years

- What does Brooks still believe is central to product quality? And what does he still believe is the most important single step toward that?
- What is featuritis? Why can it be a problem?
- What is the WIMP interface? How does Brooks view this?
- Why does Brooks now perceive his advice to "Plan to throw one away; you will, anyhow," to be wrong?
- What is the relationship between incremental-build and rapid-prototyping?
- What does Brooks say that Parnas was right about, and he (Brooks) was wrong about? Why?
- How true is Brooks' Law?
- What does Boehm's COCOMO model find to be the largest factor in a team's success?
- What, to Brooks, was the "biggest new surprise" in the decades between the publishing of MMM and this essay, written 20 years later?
- What does Brooks see as the state and future of software engineering?

Jalote - Chapter 4 - Planning a Software Project (from Quality Control onward)

- (part of this chapter was considered Exam 1 material -- now the whole chapter is Final Exam material)
- What is a quality plan for a project?
- What is meant by a Quality Control (QC) activity? Be able to give examples of such activities.
  - what are some of the differences between **reviews** and **testing**?
- What is meant in this chapter as a risk? What is included in a risk mitigation plan?
- According to Boehm, what are the top ten risks items likely to compromise the success of a software project? What are some risk management techniques for each of these?
  - Consider a given project scenario -- you should be able to reason about what kinds of risks might be high priority risks for such a situation, and you should be able to brainstorm about some possible risk management techniques for each in the context of that particular scenario.
- What are some of the software engineering lessons that can be taken from the Therac-25 accidents?
- What does the term defect injection mean? When are defects said to be injected into a system?

Jalote - Chapter 5 - Software Architecture

- What is the software architecture of a system?
- Consider a given software system. How many architectural views are reasonable for that system?
  - (the point: there CAN be multiple views for a system's architecture -- to specify different **structural aspects** of the system being built)

- What are some of important uses/potential benefits that software architecture descriptions can provide?
- Most proposed architectural views belong to one of which three types? Which of these three types of architectural view has become the de facto primary view type? (the Component and Connector view)
- Component and Connector architectural view
  - What is a component? What are some common component types?
  - What is a connector? What are some of the characteristics of and possibilities for connectors?
  - How are these often depicted?
- What are some common architectural styles for the Component-and-Connector architectural view?
  - pipe-and-filter
  - shared-data
  - client-server
  - publish-subscribe
  - peer-to-peer
  - communicating processes
  - you should be familiar with the basics of each of these styles; you should be able to recognize them, distinguish them; should have an idea of the kinds of systems well-suited to each of these styles.

Jalote - Chapter 6 - Design

- Design - or, those parts of design after determining the software architecture (VERY high level design)
- When can the design of a system be said to be correct? (2 criteria)
- What are 4 criteria we discussed for assessing the quality/"goodness" of a design?
- What does it mean for a system to be modular? What are some potential benefits of a modular system?
- What does it mean for two modules to be independent? Why might this be considered to be desirable?
- What does the concept of coupling attempt to capture? Is a low or high degree of coupling between modules more desirable? Why?
  - What are some module characteristics that tend to affect the degree of coupling between modules? What are the three major factors that influence coupling? Why does each affect coupling?
  - What are three types of coupling particular to object-oriented systems? Be familiar with these, what they mean, and why they affect coupling;
- What does the concept of cohesion attempt to capture? Is low or high cohesion within a module more desirable? Why?
  - What is the distinction between coupling and cohesion?
  - What are some module characteristics that tend to affect the cohesion within that module?
  - What are the seven levels of cohesion we discussed? Should be familiar with these, and what they mean. How do they differ? Which is considered to be the highest level of cohesion?
  - As we discussed, there isn't a hard-and-fast formula for determining a module's cohesion -- BUT what are some rules-of-thumb we discussed for determining that level? And which of these rules-of-thumb suggests that a module has functional cohesion?

  - what are three aspects of cohesion particular to object-oriented systems? Be familiar with these, what they mean, and why they affect cohesion;
- What are some aspects of object-oriented design that are not aspects of function-oriented design?
- What is the open-closed principle? Why is it desirable? How can it be satisfied in object-oriented programming?
  - if an inheritance hierarchy is built trying to support the open-closed principle, it likely satisfies Liskov's Substitution Principle -- what is this principle? Why is following this principle considered desirable?
- What is UML? What are some of the types of UML diagrams? What are some of the things that UML diagrams are intended to model?
  - what does a UML class diagram model?
  - how does a UML object diagram differ from a UML class diagram?

Jalote - Chapter 7 - Coding and Unit Testing

- Remember that coding, testing, and maintenance are all inter-related; remember, too, that code is read a lot more, and by more people, than just by its writer when it is written;
  - Hence, code should be written so it is easy to understand and read, not easy to write;
- What was Weinberg's experiment, described near the beginning of this chapter? What were its results? What are some interesting/important software engineering implications of these results?
- What is unit testing?
- What is TDD (test-driven development)? What is the relationship between test cases and code in TDD? When are test cases developed compared to when code is developed in TDD?
- What are "bad smells"? Be able to give some examples of these. What might be advised if any of these are observed in one's code?
- What is a stub?
- What is code inspection? What are some of the characteristics of a code inspection (as described in this chapter? What is the main goal of a code inspection? What are some of the hoped-for benefits?
  - know that this is recognized as an industry best practice;
  - what are the phases described for a code inspection? What are the roles, and the role's responsibilities?
- What are some code metrics described in this chapter? ...metrics for size? ...metrics for complexity?
  - What is the cyclomatic complexity of a module?
  - What are some of Halstead's measures?
  - What is meant by live variables? ...by span? Why might these be considered complexity metrics?

Jalote - Chapter 8 - Testing

- What is a failure? What is a fault? What is the relationship between these?
- What is the role of testing?
- What is SUT?

- What is a test case? What can a test case show? What can a collection of test cases NOT be said to show?
  - What are the expected components of a test case?
- What is a test suite? What is a test harness/testing framework?
- What is the preferred intent/attitude that one should start testing with? Why?
- What are the different levels of testing we discussed? (unit testing, integration testing, system testing, acceptance testing)
  - What is the focus of each of these levels of testing? Who typically performs testing at that level?
  - What are some other, additional forms of testing discussed briefly in this chapter?
- what are the three high-level tasks discussed as making up the testing process for a project? (test planning, test case design, test execution)
  - what is a test plan? what does it typically contain?
  - what is meant by test case design? what is the typical scope intended by "test case design"? what are some of the desired characteristics for a set of test cases?
- How should defects be handled as they are discovered? What is the defect life cycle?
- What is black-box testing? What is considered in black-box testing? How are test cases chosen in this?
  - what are equivalence classes? What is their role in black-box testing? Given a module, you should be able to suggest some equivalence classes for that module.
  - what is meant by boundary values/conditions in black-box testing? Why are they considered to be important? Given a suitable module, you should be able to suggest some boundary value/condition test cases for that module.
  - what is meant by pairwise testing?
  - in the context of black-box testing, what is meant by "special cases"? Why are these considered to be important? What is meant by error guessing?
- What is state-based testing? What are some state-based-testing criteria? Why is this sometimes considered to be "grey box testing"?
- What is white-box testing? What is considered in white-box testing? How are test cases chosen in this?
  - what are some of the types of white-box testing discussed? Which of these types is most commonly used? (control-flow based)
  - what is statement coverage? what is branch coverage? What is the relationship between these? Does one imply the other? If so, which? Why or why not?
    - What is a control flow graph? What is full statement coverage, in terms of a control flow graph? What is full branch coverage, in terms of a control flow graph?
  - given code or a control flow graph, and test cases, should be able to reason about the statement coverage and/or branch coverage of that set of test cases.
- What is regression testing? Why is it important?
- What are some of the ways of specifying the reliability of a system?
- What are some of the ways of specifying the efficacy of a project's approach to testing?