Name: ______Prof. Penny______________________ Student #:________________

CSC444F'09 Midterm Test

50 minutes – No Aids Allowed – 100 points total - 10 points per question

Answer all questions in the spaces provided. Use the backs if you run out of space.
Write your name and student number on each sheet.

1. To conform to the ISO 9000 quality standard for software, what software development process must a company follow? How do auditors determine if the company is ISO compliant?

   A company may follow any reasonable process. Auditors determine if the company is ISO compliant by reviewing the process documentation, and then checking to see if the steps of the processes documented have actually been carried out. They do this by auditing "quality records", which are records that document that process steps have been carried out.

2. In Moore's technology adoption lifecycle model, describe "the chasm" and explain why it is dangerous.

   The chasm exists between two psychographic market segments of buyers of new technology: early adopters and early majority. Early adopters see a benefit to new technology and wish to use it as soon as possible, despite any deficiencies. Early majority, by contrast, need a more polished product to buy.

   A software company may have processes that are successful for dealing with the early adopters, but these processes are not the same that are required to satisfy early majority buyers. If the company fails to change the way they do things, they risk losing their lead at this point, and may "cross the chasm" late or not at all to successfully address the needs of the larger early majority and subsequent segments.

3. What is an ECD? Why is it a useful concept?

   ECD stands for "effective coder day". It represents an idealized day of coding: 8 hours of completely uninterrupted time coding new features into the next release of the software.

   It is useful as by using this unit to size features, we can objectively separate out an estimate or actual measure of how long a feature will take or took to code, versus how many hours per day our coders can focus on working on getting new features into the release. If the hours are low, that is one kind of problem that needs to be addressed in a certain way. If the estimates in ECDs differ from the actual ECDs, that is a different problem entirely.

4. In the formula for converting hours to ECD's a factor of 8 is used. Why?

   The 8 is entirely arbitrary. The real unit is "the hour". 8 is used as a convenient scaling factor so we can talk about days, which may be easier to grasp, rather than hours (e.g., 7 ECDs is easier to understand than 56 hrs.)

5. If developer A can code the same feature twice as quickly as developer B, and developer B's work factor is 0.4, then what is developer A's work factor?

   Based on only this information, we cannot say. The work factor is a measure of how many hours a coder can spend on average per workday, it is not a measure of how productive they are during those hours.

6. If the nominal workday is 9am-6pm with one hour for lunch, and a coder works for 6 hours each workday coding new features into the next release of the software, and works an additional 5 hours (total) each weekend, what is their work factor, *w*?

   Over an average one week period = 5 workdays, the coder spends 5*6 +5 = 35 hours.

   w = 35/8*5 = 35/40 = 0.875

7. Say the coding phase of a release is planned to take 20 workdays; the ratio of the coding phase to all test phases is 2:1; and there are 5 coders working on this release each with a work factor of 0.6. If one of the coders leaves the team just before fork, how many workdays would you say the entire release should now take, from fork to GA?

   N = 5 coders * 0.6 ECDs/workday/coder = 3 ECDs/workday
   T = 20 workdays
   Therefore F=NT=60 ECDs

   If a coder leaves, then N=4*0.6 = 2.4 ECDs/workday
   Solving for the new T' we get T'=F/N=60/2.4= 25 workdays

   Given that coding to testing is 2:1, testing will now take 12.5 workdays.

   Therefore workdays from fork to GA is 37.5 (or 38, to round up).

Name: ______Prof. Penny______________________ Student #:_________________

8. If the previous release had poor quality as reported from the field, will adding more testers to the next release always improve the situation? Explain your answer.

   In general, to improve the quality we would need to add more coders and more testers in the appropriate ratio. More testers to find the bugs, and more coders to fix the bugs thereby found. Or one can increase the length of time spent testing versus coding, and that is in effect doing the same thing.

   If the coders tend to be idle during the testing phases (not enough bugs to fix), or are acting as testers themselves, then adding testers will find more bugs for the coders to work on, and hence will improve the shipped quality.

   If, on the other hand, the coders are going flat out fixing defects during the testing phases, then adding more testers will not mean we can fix the bugs any faster, and hence the delivered quality will appear to be more or less the same. Mind, more testers would find more higher priority bugs, the coders will prioritize those, and hence the quality would improve without increasing the number of coders. But this is a second order effect.

9. Show how releases can be overlapped. Why is it a good idea to overlap releases?

   In the beta testing phases the coders tend to wind down their efforts, as there are fewer and fewer bugs to fix.

   In the specification and design phase, there is more and more work for the coders to do, as more features get more worked out.

   Therefore attempt to schedule fork of the next release to coincide with GA of the previous release. That way beta test can be overlapped with the next release's spec & design phase in order to keep the coders 100% utilized.

10. Why do some software companies say that support will end on a feature release the day the feature release after the next is released?

    This is done so that the company can limit the number of active maintenance streams to 3: the ongoing development, the previous release, and the one before that. As soon as the ongoing release is shipped, the one two before goes off maintenance, and the company must again work on only 3 maintenance streams at once.

    Each maintenance stream supported means that coders must fix any defects found in any of the streams in all of them: they must see if the defect is still there in each maintenance stream, and if so then fix it there, and the fix may be quite different each stream if the code has changed. The more of these maintenance streams must simultaneously be supported, the less time coders can spend on coding new features into the next release, and the more future dollars are potentially lost to the software company (opportunity cost).