The following hint often appears on the cover sheets of my exams:

On the essay questions, your answers need not be excessively lengthy, but they should be pithy. Pithy and accurate answers receive more points than those which address the question in a shallow, erroneous, overly-general, or cursory manner.

Helpful Definitions:

Cursory (adj.) - hasty and without attention to detail; not thorough

Pithy (adj.) - having substance and point; tersely cogent.

Cogent (adj.) - having the power to influence or convince.

"*Pithy adds to succinct or terse the implication of richness of meaning or substance*."
(Merriam-Webster)

Essentially, I'm not looking for parroting, but I am looking for you to assimilate core software engineering principles, particularly those covered in class, into your answers, in an intelligent manner. Write as though you have learned something in the class, and have acquired some subject matter expertise.

In the interest of helping students understand my expectations, a few concrete examples are provided on the following pages. (*These are actual student responses to real exam questions*).

Note: These may be questions from a different course than the one you are taking. Accordingly, do not be alarmed if the subject matter seems unfamiliar.

Question:

The Agile Manifesto emphasizes "*Responding to Change*" over "*Following a Plan*."

(a) Why do you think this became so important to agile developers that they named it one of their primary planks?

(b) Do you think that this creed can be abused? How so, or why not?

Partial Credit Answer:	**Full Credit Answer:**
a) In my opinion, agility means dynamism, and responding is a more proactive attitude than simply following the established path. b) I think agile methodologists clearly do not resent having to follow a plan, they just urge managers to be ready to make changes to it. The abuse (or non-abuse) of this is the difference between a good project manager, and a bad one.	a) I think this is largely a response to perceived overkill on documentation and processes, as a high-level CMM organization might have. The actual objective is to produce software that the customer wants, and if the customer changes his mind, or if environmental factors change, following an existing plan would not produce what the customer wants. Instead, the organization needs to respond to the change. b) It can definitely be abused. An organization could have a disdain for plans and processes so much that they don't do any planning. They can state that following plans isn't important, because planning incurs overhead, and due to the volatility of projects, they couldn't even give any benefit if things change which invalidate the plans. This may negate the advantages of being agile by providing no direction for projects.

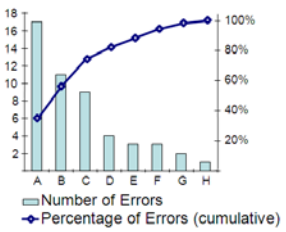
Instructor Comments & Explanation:

The partial-credit answer is vague; it "dances around" the question, rather than answering it directly. The opening clause is weak ("*In my opinion, agility means dynamism...*" So what? The answer doesn't go on to explain why "dynamism" is significant.) The answer to part (a) – a single sentence, by the way – goes on to reference the principle of responding to change in a vague way, but fails to convince me this is a bedrock principle for any particular reason. In fact, I could easily rearrange the words of the answer, and come up with something that is completely opposite, yet sounds equally intelligent: "following a plan is a more proactive attitude than simply responding to change." No justification is provided to solidify the student's argument. In a similar fashion, the statement, "The abuse or non-abuse of this is the difference between a good project manager and a bad one." In that statement, the word "this" could refer to anything! Scheduling, cost estimation, motivation, pizza parties, whatever! The sentence is so vague that is almost meaningless.

On the other hand, the full-credit answer gets to the heart of the matter: responding to change became important because heavyweight methodologies were SO tied to bulky processes that they were unable to adjust quickly enough in a volatile environment.

By the way, I don't even completely agree with the answer provided (I think it would be better to say, "if environmental factors change, it may be difficult to adjust if the existing plan is overly comprehensive or inflexible," rather than, "if environmental factors change, following an existing plan would not produce what the customer wants."). Still, I'm willing to let this slide, because, overall, the answer addresses the question quite strongly, specifically, and completely.

Exam Question:
You are presented with this Pareto chart, which was derived from metrics data collected during the company's last two development projects. Explain how you might use this chart to optimize your process, and help meet the company's stated goal of lowering defect density during development.



A = Interface Error
B = Overflow/Out-of-Bounds Index
C = Computation/Math Error
D = Typographical/Clerical Error
E = Initialization Error
F = Security Vulnerability
G = Control Flow Error
H = Timing Error

Partial Credit Answer:
More and better test cases are needed for the areas with the highest number of errors (A, B, C). If possible, problem sections of code should be developed first, so that testing can be done on these areas earlier.

Full Credit Answer:
There are many steps I would take to lower the number. Since a lot of errors are related to interface errors, I would focus on setting up peer reviews that emphasize evaluating the interface and providing feedback to fix problems that are found. There could even be a problem with our requirements analysis process, so I would optimize that process to make sure everyone understands the required functionality. Perhaps the biggest source of defects is in intergroup communication, and putting individual components of code together. I'd improve the process in which groups working on separate parts of code put pieces together, by requiring more documentation, or giving them extra time to meet, and educate each other on their code. Furthermore, I would improve the testing process so that it is integrated into the whole project process – it sounds like it is just performed at the end, where we would naturally find more defects than if we tested more thoroughly along the way.

Instructor Comments & Explanation:
The partial-credit answer is overly brief, and overly vague. The statement "More and better test cases are needed for the areas with the highest number of errors" states the inherently obvious, and offers little concrete advice on how the problem could be solved. The strategy of "producing problematic sections of code first" is not necessarily a bad one, but the student does not explain how this strategy relates to the data provided. How would one know which segments of code are most likely to be problematic? Overall, it seems like the student is hoping for lenient grading, more so than intelligently addressing the issue head-on.

Conversely, the full-credit answer is thorough, and contains some creative ingenuity. Although the answer only talks about the #1 issue (Interface Errors), the treatment of that problem very detailed. The suggested steps are clearly related to the problem at hand. "Interface problems are often communication problems" is an astute hypothesis, and the strategy outlined is clearly aimed at mitigating communication difficulties in the development environment.

Exam Question:

An organizational manager collects and analyzes metrics in order to determine the average cost of producing new, modified, and reused lines of code. She hypothesizes that new LOC will cost the most to develop, and reused LOC will cost the least. However, after collecting the data, she finds that changed code costs over three times more to modify than new lines of code cost to generate. Her first thought is to dismiss the methodology as flawed. Offer an argument that instead supports the methodology by rationally explaining the result.

Partial Credit Answer:

Changing code can be more costly than new code because of the number of bugs that can be produced can be greater than that of new code.

Code already in place can be referenced by many other lines of code, and if her software engineers just change the code without altering other lines of code that reference that code and don't test for compatibility, there will be many defects.

New code, however, references other code that is in place already and doesn't use the new code at all (yet). References to new code can be placed precisely and not conflict as much with reused code. Her hypothesis is what is flawed.

Full Credit Answer:

With a well-laid-out design, writing new code can be fairly simple; if you've designed down to pseudocode it is just a matter of translating into a programming language. However, modifying code can take more time because it involves reworking an existing program. This means taking tome to understand the old code, and finding a new way to rewrite it that will not affect its functionality in concert with the unchanged and new code. Therefore, modifying code takes the time to develop the new code, along with the time to understand the old.

Instructor Comments & Explanation:

The partial-credit answer is written in a confusing way. It is lengthy, but seems to be saying the same thing a couple of times in a different way. More significantly, it contains some shaky assumptions. The notion that the number of bugs in modified code "can be greater than that of new code" may be true; but then again, the number of bugs could be less as well! The accompanying argument fails to convince me that modified code will *always* be buggier code. Without that key point supported, her explanation of the finding is rather tenuous.

On the other hand, the full-credit answer has a more logical and plausible explanation: that it takes longer to modify code than to write it from scratch because of the additional step needed – that is, the necessity of first understanding the code that is being changed. The caveat of "with a well-laid-out design..." is important as well. Without that stipulation, one could argue that writing new code can be more time consuming than modifying code, because, in program maintenance, at least you have a working system to begin with! But the student's opening remark alludes to an assumption that the manager's team is devoutly adhering to some strong design methodology, which is why, in that particular situation, they have reached the finding that code generation may be less time-consuming than code maintenance.