*This document does not need to be updated with wording changes for the questions, but does for major substantive changes*

**The Three Levels of Software**

**Intent:** This is the topic of the lecture. It adds a few pieces not present: better emphasis on the concept of an error of modular reasoning which is not also a code bug, and the logics each of the levels corresponds to.

**Q:** Give another example of a defect of modular reasoning ("level 3") which does not result in a code ("level 2") or runtime ("level 1\*\*")\*\* failure. Make sure your answer contains (1) a program or description of a program, (2) an explanation of why the program cannot be shown correct by modular reasoning(e.g.: function precondition not being met, a subroutine's guarantees are not strong enough to ensure desired behavior, etc), and (3) an explanation of why there is no concrete input or scenario that would exhibit a defect, but how a small future change could cause a sudden failure.

**Intent:** This question forces them to practice the lessons of the article, which is also the biggest idea of the unit. The extra instructions were added in August 2022 in an effort to reduce the number of incorrect answers by forcing the students to take the necessary steps.

**The Design of Software is a Thing Apart**

**Intent:** Application of a lesson from this unit, replacing some common citrus advice with something deeper.

**Q:** Give another example of when two different (or partially different) design intentions result in the same code, so that, if one changes, the other should likely not be changed. I.e.: a case where duplicated code is actually a good thing. Make sure your answer contains (1) a code snippet or a precise description of code, and (2) two alternative specifications for this code snippet, informally stated.

**Answer:** This question forces them to practice the lessons of the article, and to think about code in terms of its spec. The extra instructions were added in August 2022 in an effort to reduce the number of incorrect answers by forcing the students to take the necessary steps.

**Painless Functional Specifications Part 1**

**Intent**: In significant part, this is here so that not every required reading in this unit is by Jimmy. This also functions as an intro to Joel Spolsky for anyone who’s not already read him. Pedagogically, its role is less clear. It does show a famous figure telling them they should have specs, which helps counter pushback we may get from the course seeming to advocate specs.

**Q:** A software artifact is anything that comes out of the software process: code, diagrams, installers, etc. What artifacts depend on the design spec as advocated by Spolsky.

**Intent:** Students approximately never get this wrong, and it doesn’t require thinking hard about the article. It checks the box for trying to attach a question to most readings, but its existence is rather questionable.

**You are a Program Synthesizer**

**Intent:** This one is actually what we ask students to watch before the course. It is genuinely good content that we want students to know, and contains content that overlaps with many parts of the course — and Dojo. It helps set the tone for the course: you will be taught by someone who is good enough to speak at Strange Loop, the most prestigious industry conference; you will be taught things inspired by some very cool research; the things you have already learned are citrus advice. It also contains some content we directly use in this unit: conditionals being logical rather than syntactic, and multiple specs/intentions for the same code.