Question 0A: Do not grade. But do look at the assessment submissions of all team members "together", to see if there are any patterns. Bring to my attention anything that looks like a real problem. I'm going to go through them separately, but pointers from you can speed up dealing with any bad cases (I've already heard of several). Hopefully no one will omit explanation this time.

Question 0B: Do not grade. Ignore this unless there's something alarming, I plan to go through all of them.

Graded questions: Please make sure everyone grades consistently.

Part 1 (20 points): You are a member of a team developing the XXX software (assume some arbitrary software project, not your team project for this course). Your team arrives at the scheduled time to give your demo. Your software is hosted on YYYCloudPlatform and cannot run on localhost. You begin to startup your software, but the cloud host displays an odd error message you've never seen before and nothing further happens. You try again, same thing. Everything worked fine last night when your team practiced the demo. Finally, one of your team members admits that they made small changes to a few files after your practice session, committed to your shared repository, and uploaded the new build to your cloud host - overwriting the previous build you practiced with yesterday. This team member has often changed your software right before an important demo, but their previous changes always worked.

Take one of these two positions, A or B:

A. This scenario is, unfortunately, likely to happen and explain what your team would do to fix the problem quickly to proceed with the demo.

B. This scenario is, thankfully, unlikely to happen and explain what your team set up to prevent it.

A trivial answer like "We'd reschedule the demo" or "The overeager team member does not know the cloud host password" will receive 0.

Answer:

The purpose of this question is to check the student's practical understanding of version control and continuous integration processes.

If they choose A:

6 points for discussing the concept that version control supports reverting to a previous version of the source code as it was at some particular point in time (or a particular tag).

7 points for discussing how they figure out which point in time (or tag) they want to revert to, presumably what they had practiced with the night before, This is not necessarily the most immediate previous commit or tag, since the team member might have made a series of commits and possibly tagged them.

7 points for discussing that they have to rebuild and redeploy the software from that previous version, i.e., the version repository does not store builds and the executable does not magically appear on their cloud host (which does not store backups).

Note: Give 0 credit for an answer that evades version control, e.g., discusses debugging.

If they choose B:

6 points for discussing the concept that CI can be configured to run automatically on commit (or when commit is attempted).

7 points for discussing the concept that CI supports rejecting a commit if any analyses, tests, etc. fail (or alternatively it accepts the commit but blocks deployment, which would work if the CI tool also automates deployment).

7 points discussing what CI does wrt running unit tests and analysis tools, which have to be written, configured, etc. in advance.

Note: There are other reasonable approaches that should get full credit, e.g., the VCS or CI is configured to require review and signoff by another team member before allowing commit (or possibly before allowing deployment); the VCS only allows commit to a branch and only deploys from main, after pull/merge has been accepted by a different team member; very fancy CI with a staging environment that prevents deployment to the production environment unless system testing succeeds; probably others. Give 0 credit for 'human resources' approaches like our team member wouldn't do this because then they'd be fired.

Part 2 (40 points – 30 points for 2a and 10 points for 2b): 2a and 2b can refer to the same or different parts of the code.

Part 2a (30 points): Pick a non-trivial class or module from your team project, consisting of at least three non-trivial methods/functions, and state which class/module and methods/functions you're considering. If your program does not have any components with a least three non-trivial methods, pick three non-trivial methods from different components and clarify exactly which ones. By non-trivial, this question means methods including conditional branches and/or loops.

Describe the input equivalence partitions for the three methods and how, ideally, you would test each of those partitions. Recall that functions with multiple parameters usually have separate sets of equivalence partitions for each parameter. Make sure to include invalid as well as valid partitions. Do not modify your team's code.

(If your team project does not include at least three non-trivial methods, contact the instructor asap. She will check your team repository and then provide an alternative question if warranted.)

Answer:

The purpose of this question is to check the student's understanding of equivalence partitions as an approach to choosing test case inputs at the unit testing level.

You will need to check their repository for the specific methods mentioned. Note that teams that already used equivalence partitioning to develop unit tests will generally do better on this question than students from teams that did not; this is intentional.

Up to 10 points per method (for each of the 3, if they do more than 3 just consider the best 3).

Since this is at the method level, where test case code is calling the method code, the compiler or runtime environment will probably prevent calls with parameters of the wrong base type, so 'wrong base type' should not be something they need to write tests for or consider as invalid equivalence partitions (e.g., an integer argument where a string is required, unless the language really allows this). Give 0 for tests that couldn't possibly run anyway, but ask me to look at it if in doubt.

However, if they treat calling the REST API GET and POST functions as unit tests, then wrong base type may indeed be possible, so these should get credit.

In any case, they should be able to devise at least one invalid equivalence partition with inappropriate values, format or size/length.

5 points for covering the main valid equivalence partitions you can think of for each parameter of that method. There might be only one or there might be several.

5 points for covering all main invalid equivalence partitions you can think of for that method. There might be only one or there might be several. If you cannot think of any invalid partitions, ask me to look at it.

For example, if the parameter is the ID of an item where the method is supposed to do something with that item, then the valid partition is an ID for an item that exists and the invalid partition is an ID for an item that does not exist. If IDs are represented as integers, then there might be two invalid partitions, one is a positive integer that does not actually correspond to a real item and another would be any negative integer, assuming they do not use negative integers as IDs. If the method is only supposed to do something for expensive items, then two valid partitions are IDs for items that are considered expensive (above some threshold) vs. IDs for items that are not considered expensive (below some threshold). Notice this implies some lookup on the ID to determine expensive/inexpensive.

Another example is if the parameter is a node in some data structure, then an invalid partition might consist of the null node. If that data structure is a tree, then there are could be multiple valid partitions: root node, interior node, leaf node.

Some parameters will have range values, so there can be values below, within, and above the range.

For methods with multiple parameters, if any parameters are not considered then prorate the points. For example, if there are two parameters and they only consider one, then they can get at most 2.5 points for valid and 2.5 points for invalid. They should state something indicating they are aware of the interactions across multiple parameters, but they do not need to detail the cross-product.

The students do not need to consider "inputs" received as return values from system calls, library calls, etc. unless that's the only way to get inputs. The arguments to HTTP GET and POST are considered inputs.

Part 2b (10 points): Did your team actually employ input equivalence partitioning when designing your unit tests and/or system tests? Do not modify your team's code.

If so, please tell us which specific tests in your codebase address which specific partitions (you only need to cover three units and/or API entry points for this question).

If not, describe what approach your team did use to invent test case inputs (again consider three units and/or API entry points).

Answer:

The purpose of this question is to check the student's general understanding of testing and how it applies to their team project. The students on the same team should be graded "together". Inconsistent answers are not necessarily wrong, but might give a clue that some students participated more fully in the project than others (e.g., who apparently have no idea where the team's test case inputs came from).

If the student answers yes, check whatever code and tests they refer to and give full credit if it seems like those tests really did intend to cover equivalence partitions. These might or might not be the same as in 2a, either way is ok.

If the student answers no, again check the code/tests they refer to and see if whatever they say seems to match their stated 'approach'. One good approach is choosing inputs to force branches, improving coverage. Choosing inputs planned for their upcoming demo is also a reasonable approach. Choosing inputs at random is not a good approach, but still give full credit if the student says they chose at random and it indeed looks like that's what they did. Pretty much any answer that sounds like they understand the question should get full credit.

Part 3 (20 points): Did your team use any design patterns in writing your code? This might include design patterns listed in any design patterns catalog whether or not they were discussed in class.

If your team used at least two design patterns, pick any two distinct design patterns, point to the specific code that reflects each of them, and describe in each case how it implements that design pattern.

If you used only one design pattern, point to the specific code that reflects it and describe how it implements that design pattern. In addition, pick a second design pattern and describe how your team could have used it to improve the quality of your code.

If your team did not use any design patterns, pick two distinct design patterns and describe how your team could have used each of them to improve the quality of your code.

Do not consider MVC as a design pattern (you will receive 0 for this question if you do) and do not modify your team's code.

Answer:

The purpose of this question is to check the student's knowledge and understanding of design patterns. This is intended to be an easy question for students who understand design patterns, and hard/impossible for those who don't know what a design pattern is and are unable to use google to find out.

Up to 10 points for each of two design patterns. If they discuss more than two design patterns, consider only the best two. If the student's answer points to any real code claimed to implement a design pattern, check it. I suspect most teams can find some use for the singleton, adaptor and/or facade patterns, but its fine if they use others instead.

5 points for explaining the gist of the design pattern.

5 points for describing some plausible way the student's team did use the design pattern or could have used the design pattern if they didn't already. Students do not need to describe details of how they'd refactor the code to introduce a new pattern.

Give 0 credit if they describe design principles (which were covered in the first assessment) rather than design patterns.

Part 4 (20 points): Imagine that your team project was required to develop a *library* instead of a *service*, with the intent that the library could be used by other developers as part of their own application or service.

What are the most significant changes that would be needed in the software architecture, API and persistent storage of your system and why?

Ignore whether your system's functionality might not make sense for a library, this question is concerned with the structure of your software, not what it does. Do not modify your team's code.

Answer:

The purpose of this question is to check the student's understanding of the distinction between a library and a service. And, hopefully, help those students who are still confused realize that a service is not an app.

Up to 8 points for Architecture changes: The gist is a service runs as a separate program whereas a library runs as part of someone else's program. The service program runs on a platform the student's team configures and generally controls, whereas a library has to run on any platform configured to execute the program in which the library is included. (A library can still place some restrictions, e.g., it will only run on Windows, but I don't expect them to address that.) The answer should be tailored to the student's team project but don't be picky about details.

Up to 8 points for API changes: A service receives inputs over the network using a protocol like REST or RPC. In contrast, a library receives inputs as parameters to function calls. If the student's team built their service using some framework that hardwires how inputs are handled, it probably has to be removed. The answer should be tailored to the student's team project but don't be picky about details.

Up to 4 points for persistent storage changes: Accept anything that sounds plausible. There are real-world libraries that connect to a remote database shared by all instances of the library, there are libraries that insist on being configured with a local instance of some specific database, there are libraries with multiple interfaces to work with whatever data store is available locally, and so on.

The documentation ought to be presently differently, but the students weren't asked to say anything about documentation. They do not need to consider operating systems kinds of issues like shared libraries, memory mapping, multi-threading, re-entrancy, etc. or anything else outside the scope of this course.

This is the end of the assessment.