Debugging Quiz Rubric

Robert Underwood

1 Rubric

category	excellent	great	good	fair	poor
question	the question is *succinct* and (1) **describes an unsolved problem** and (2) is **possible to answer**	the question is not succinct, but is all of the above	the question is 1 of the above	the question is none of the above	There is no question
hypothesis	The hypothesis is *complete*, and (1) **systematic**, (2) uses a **sound model**, and (3) is **based upon to evidence**	the hypthosis is not complete, but describes all of the above	the hypthosis is not complete, but describes 2 of the above	the hypthosis is not complete, but describes 1 of the above	hypthosis is none of the above or unresponsive
prediction	the prediction is *succinctly* stated as a (1) **logical argument** with (2) **testable premises**	the prediction is not succinct but is all of the above	the prediction is 1 of the above	the prediction is neither of the above	the prediction is nonexistent
testing	the testing plan is *efficient* (1) **sound**, (2) **complete and specific**, (3) **controls for extraneous variables**	the testing plan is inefficient but is all of the above	the testing plan is 2 of the above	the testing plan is 1 of the above	the testing plan is none of the above
analysis	the analysis is *succinctly* stated is **complete** and **valid**	the analysis is not succinct but is all of the above	the analysis is 1 of the above	the analysis is none of the above	the analysis is non-existent

2 Definition of Terms

2.1 Question Selection

Solved questions seem like an obvious thing to avoid when asking a question, but a surprising amount of time, people design their experiments without considering what type of questions have designed in the past. This is especially prevalent in academia: "What has been is what will be, and what has been done is what will be done, and there is nothing new under the sun." Often times problems were solved 20+ years ago, but the answer was either lost, forgotten, ignored. Consider this carefully when designing new systems; people think about designs of the systems they build. To do otherwise is a form of chronological snobbery.

Also seemly trivial, It is also important to choose a question that can be answered. Common mistakes include:

- The answer may require more time or resources than you possibly have to answer. For example the problem may be N.P. Hard and extremely large.
- The answer may be unaccessible to strictly scientific methods (i.e. matters of personal preference, morality, or ascetics). For example which computer system promotes the best societal good? See also Section 2.5.
- The answer may require data that you can't access, does, or cannot exist. For example you probably can't get data on all of the device level, detailed root cases of network outages on campus. Trust me, I've tried.

2.2 Valid vs Sound

What is the difference between **valid** and **sound**? In this document, **valid** refers to when conclusions follows from valid logical inference. While **sound** arguments refers to when an argument is **valid** and its premises are true. Consider the following examples:

1. if memory is overfull, the computer will go slowly
2. the computer is going slowly
3. the memory is overfull

This is an **invalid** argument, specifically, it commits a formal logical fallacy called "affirming the concequent". Even if the premises were true, we could not validly conclude claim 3 is true We could try to get out of this by saying:

1. **if and only if** memory is overfull, the computer will go slowly
2. the computer is going slowly
3. the memory is overfull

This makes the argument **valid** because the first claim implies its converse via the text, "if and only if". However, is this argument plausibly true? Of course not! There are a myriad of ways computers could be slow that have nothing to do with memory being slow. For example, we may be spending all of our time in a tight CPU bound loop. Therefore while this argument is **valid** it is not **sound**.

An argument that is both valid and sound may be stated like this:

1. if memory is overfull, the computer will go more slowly than if it was empty
2. memory is overfull
3. the computer will go slowly than if it was empty

2.3 Systematic vs Complete?

For the purposes of this class, A hypothesis is **systematic** if it is works from a valid ontology of the system. An ontology is an enumeration of all possible states of a given system.

For example consider the classical Von Neumann model: a computer consists of inputs devices, memory, a single processing element, and output devices. Using this model consider how we address the problem of a computer being slow.

If we are not being systematic, we may consider only the processing element. In doing so, we ignore the possible impacts of the input device not providing input sufficiently fast, memory not serving data quickly, or the output devices not providing the output when they are available.

If we are being systematic, we consider all the aspects of the machine. Now the Von Neumann model is a bit primitive, most modern machines are NUMA meaning they that memory access don't take uniform time. Does that mean we are not being systematic? It depends, if we are running a single threaded program that will likely be bound to a single core, probably not. If we are running a program with thousands of CPU cores distributed over a super computing cluster, probably yes.

As we see in the previous example of a super computer, we can account for all aspects of our ontology, but still not accurately describe the system. When the ontology is a good approximation for the actual system, we say that it is **complete**. Notice the use of the word "approximation" in this explanation. All models are approximations, otherwise they are not models but rather the system itself. Models must leave things out to allow us to wrap our minds around them. The question comes "what is plausible to leave out the model?". In science, we leave things out of model we can prove do not impact our outcome. For example in the example in the previous paragraph, the Von Neumann model works well for the single threaded application, because we can look at the source code of the OS or at the kernel's runtime meta data and determine it was pinned to a single CPU. Since the task was pinned to one CPU, we can show that CPU memory access times are approximately uniform form a specific core.

2.4 Evidence Based

What do I mean by "Evidence Based", and how is it different than Completeness or Systematicness? By a model being Evidence based, I mean the assumptions of the model are identified, and evidence is provided that those assumptions are true. Consider again the example of a slow computer:

The following hypothesis would be systematic and based on a sound model, but not evidence based on a single application:

> The input device, output device, processing elements are not the bottleneck. The computer is slower than it would be if because the memory is empty because swapping is occurring.

Notice how this explanation provides no justification for eliminating the input, output devices or processing element as bottlenecks. An evidence based hypothesis would provide justifications such as:

> After conducting experiments to measure the effective bandwidth of the system, the input device, output device, processing elements preform much faster than the memory device and are therefore not the bottleneck. The computer is slower than it would be if because the memory is empty because swapping is occurring.

Notice you could be evidence based, but not be complete or based on a sound model. For example, you may think that the color of light in the server is part of the model of machine execution. This is not a sound model, but you may have provided evidence that it had no effect. Likewise you may have provided evidence that the input and output device are not at fault, but not considered the processing element.

2.5 Logical Arguments and Testable Premise

For this class, I would like to see the logical argument called out in a explicit numbered list. Examples of this can be found in Section 2.2. You may then refer to steps of your argument in the remainder of the argument by claim/premise number (claim 1, claim 2, etc.)

Each claim in your argument must be testable. For example, consider the claim: "Windows is better at File IO and Linux". Not only is this claim dubiously true, there is no way to evaluate if Windows is "better" at File IO than Linux. You could be arguing that Windows, Window's provides drivers with lower IOPS, have greater bandwidth, have better disk schedulers, or something else entirely. There is no way to discern which of claim is actually be asserted. Even if you did know which claim was being asserted, it is possible that Linux does better at some of these areas than Windows and Windows does better than Linux at some. In that case, how do you make a comparison between the two? There is no objective way to judge this claim. Therefore, consider phrasing your claims either in terms of a aspect of the model (if memory is overfull, swapping slows down the computer), or some measurement that you can perform (there are this many virtual pages allocated).

A related aspect of this is the **specificness** of the claim. Prefer more specific claims to more general ones. General claims are harder to prove and are less likely to be testable. By making your claims more specific, it makes writing and constructing your test cases easier.

2.6 Test Efficiency

A test is efficient if it uses an approximately minimal amount of time and resources to verify or reject the claim.

Consider for a moment the question of using print statements. Yes, they take practically no time to insert into the code. However, if you have a put them in a tight inner loop, you may have more prints than you have time to analyze. Additionally, you then have to recompile the code which for large projects is time consuming. Finally, you may perturb the results of your experiments because often the time it takes to print something is vastly larger than the actual message itself.

Consider the case of using a debugger. Using step instructions inconsiderately is a waste of time. Putting breakpoints on every line is also a waste of time. If you are going to use a debugger, consider what you are going to inspect and what you hope to learn from it.

2.7 Controls For Extraneous Variables

Often times there are multiple factors that are not currently under test which can impact the results of your experiments. You should identify these elements and describe how you are going to mitigate these effects. Sometimes you won't be able to control for these effects, in this case you should employ randomness to mitigate the effects of extreme cases.

2.8 Be Succinct

While there isn't a hard guideline, after you've proved your point move on. Every word you write has to read in order to be graded or to be useful. Most people stop reading emails carefully after the first few lines. Organize your information to account for this fact and choose your verbosity carefully.