**CSE 464 - HW 3 Phase 1**
**Bhavya Patel - ASU ID: 1225740997**

1.
i) Balance time: Testers must trade off time spent clarifying and reporting findings versus hunting for new bugs. Because spending too much time on investigation can reduce total discoveries, while spending too little can undermine fix rates and decision-making.
Example: Let's say I find a bug where an app crashes, but it only happens after a very specific 20-step process. I could spend the next four hours trying to find a simpler 3-step way to make it crash, which would make my report better. Or, I could spend those four hours finding five new bugs. I have to make a judgment call on which activity provides more value.
ii) Conflicts of interest among stakeholders: Stakeholders value “quality” differently, so what counts as a must-fix bug varies by perspective.
Example: Our marketing team might want a feature to look flashy, so they ask for a complex animation. But the developer knows this animation is buggy and hard to maintain. When I report a bug in that animation, I'm caught in their conflict.
iii) Presenting under stress: Teams often operate on tight schedules, and late-stage pressure degrades receptivity to reports perceived as minor, complex, or unclear, so communication must be crisp and targeted.
Example: If I walk up to a programmer an hour before a major deadline and report a simple spelling mistake in a help file, they'll probably be (understandably) stressed and annoyed.
iv) Preserving credibility: Credibility drops with exaggeration, weak analysis, or unclear writing; people who write precise, respectful reports and do solid troubleshooting.
Example: If I report excessive "bugs" that aren't actually bugs, but just features I didn't understand, people will stop taking my reports seriously. Then, when I find a critical bug, they might ignore it.
v) Integrity: Preserving my integrity means I have to be honest and report what I found while maintaining a collegial tone and respecting policies.
Example: I should avoid labeling a design issue as a coding error to force priority; instead, classify it accurately and connect it to the stakeholder's impact.

2. a)
The bug workflow is the entire life cycle of a bug report, from the moment it's created to its final resolution. It's the process that defines how a company handles these reports. Steps of bug workflow: report the bug, verify the bug, the engineer evaluates and either fixes or recommends defer/reject, the project manager prioritizes or reassigns, triage decides on expensive or disputed items, engineers mark fixed, and testers verify the fix and retest deferred/rejected items for patterns or appeal
b) i) The bug triage process, defined according to the video as "deciding which bugs to fix and which to leave in the product". It can involve several people. For difficult or deferred bugs, the

decision goes to a formal triage team or Change Control Board (CCB). This team is made up of the project manager, programmers, and testers, i.e., the people with authority and influence over scope, risk, and release criteria.
ii) The main steps for that formal triage team involve reviewing the bugs that are particularly expensive or have already been deferred by a programmer. This team's job is to look at the evidence and make the final decision. They weigh the costs (programmer time, risk of the fix) against the benefits (the value of fixing the bug) and decide whether to defer the bug or reject it.

3. According to the video slide, it states that "A bug report is a tool you use to sell the programmer on the idea of spending their time and energy to fix a bug". I think this means a bug report is a request because just finding a bug doesn't mean it will get fixed. In fact, the video says that "Few (or no) companies fix every bug they find". We are essentially requesting a programmer's time, which is a limited and valuable resource. We are competing for that time against all the other bugs and all the new features they have to build.
It's so important to write an effective and persuasive report because, if my report is effective (clear, accurate, easy to reproduce), I save the programmer time and make it easy for them to fix the bug. If my report is persuasive (explaining why the bug is a significant issue to the user or the business), I motivate the programmer to fix the bug.

4. Here are three usages of a bug tracking system:
i) Primary goal: Getting the right bugs fixed by capturing the right reports, recording relevant details, supporting technical and evaluative dialogue, enabling multiple viewpoints, ensuring accessibility to stakeholders with influence, and preventing loss.
ii) Schedule reality checking: A project manager can look at the number of open bugs, especially critical ones, to get a realistic idea of the product's quality. If there are 100 "showstopper" bugs still open, the manager knows that we're shipping soon schedule isn't realistic.
iii) Archival bug pattern analysis: After a project ends, a manager can analyze the whole database of bugs to find patterns. For instance, if they discover that 40% of all bugs were in a specific feature, it indicates that the component was poorly designed or tested and requires special attention in the next version.

5.
i) Isolate the failure: This is about making the bug report as simple and focused as possible for the programmer. Its significance is that it saves the programmer time and guesswork. This means eliminating unnecessary steps and finding the shortest, simplest set of steps to reproduce the bug. It also means I should only report one failure per report.
ii) Generalize: To show the bug is more serious and widespread than it might first appear. If I find a bug using a weird corner case, then I should also check the failure under mainstream values or across a broader range of systems so the issue appears more credible, widespread, and

worth fixing. This step also includes checking if the bug happens on different configurations, not just my own.
iii) Externalize: Shift focus from program internals to stakeholder consequences by bringing comparative data, historical support costs, competitor context, and who is affected and why, which strengthens the business case.
iv) Clear and dispassionate: It means maintaining credibility and a professional, cooperative relationship with the development team. The report must be easy to understand, but just as importantly, my tone must be neutral and nonantagonistic.

6. Typical Fields in Bug Report:
i) Problem report number: A unique ID for tracking.
ii) Date reported: The day I first wrote the report.
iii) Reported by: The person who found the bug.
iv) Program (or component) name: The product that is broken.
v) Release number/Version (build) identifier: These fields pinpoint the exact version of the software so the programmer knows what to test.
vi) Configuration(s): My hardware and software environment (including all the versions of each library).
vii) Problem summary (problem title): This is the most important part of the report. It's a one-line summary that should describe the failure and when it happens.
viii) Report type: A category of the bug, like "Coding Error", "Design Issue", "Documentation mismatch", or "Suggestion".
ix) Can you reproduce the bug: A simple yes/no/sometimes(under what conditions).
x) Severity: This is the tester's rating of how bad the bug is (e.g., "small/medium/large").
xi) Priority: This is the manager's decision on when the bug should be fixed. A bug can be high severity (such as a server crash) but low priority (if it only occurs on a machine we're no longer supporting).