Ryan Filgas
Chris Gilmore
CS300
Homework 2
11-9-21

1. When describing a system, explain why you may have to design the system architecture before the requirements specification is complete.

   From the slides: The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.

   For this reason, the system architecture may need to be designed in order to get the project from the bidding process. Other than contractual reasons, it might also need to be designed in order to comply with government regulators.

2. **Using examples, explain why configuration management is important when a team of people are developing a software product.**

   From Somerville: Configuration management During the development process, many different versions of each software component are created. If you don't keep track of these versions in a configuration management system, you are liable to include the wrong versions of these components in your system.

   Configuration management comes down to a few points. Development of the different components of a system need to be coordinated. This is done through four activities: version management, system integration, problem tracking, and release management. More broadly, proper configuration of a system needs to be tracked to avoid problems, and there needs to be management in place to ensure that a development team works in unison without causing incompatibilities, losing track of bugs, or inadvertently overwriting someone elses work.

   Scenario: Bobs business just built a piece of software to track and predict global weather patterns for a national government owned

weather agency. Unfortunately the section of the software being developed for predictive weather patterns was built for version 3 of the system, and the configuration had changed in versions 4 and 5. This important piece of the weather software has different arguments and method names than called by version 5 due to miscommunication, and further doesn't connect to the servers properly. It was also reliant on another piece of the weather system in version 3 for intermediary calculations, and this is no longer compatible as another developer thought it was a good idea to overwrite someone else's work without notifying the team. To complicate things further the predictive module had ignored a few bugs in previous versions which after pieces were updated caused the predictive module to work properly 50% of the time.

Version management would coordinate development by several programmers so they didn't overwrite each others code, and so that they didn't change configuration between interconnected pieces without communicating to necessary parties.

System integration could define which version of the predictive section's dependent software is used so that the predictive software used version 3 for the overwritten component it relied on instead of the incompatible version 5.

Problem tracking could notify the developer that a bug was reported in the predictive module early on, perhaps version 1 or 2 so that the problem didn't compound in such a way the code was too difficult to fix later due to reliance on faulty components.

Release management could have had the software release of version three delayed until this crucial component was completed and fixed.

This is a sad and unrealistic story, but it got worse when they fixed the compatibility issues and launched version 5. The result of bad configuration management practices unfortunately caused a publicity scandal for a national weather station providing data that spanned several news networks. The weather news was rarely

accurate, and viewership dropped on each of those networks. Many news stations pulled out of the agreement, and the agency lost much needed funding. Bob lost this client and several others the week they flipped the switch on the new build. Because they had no system to track and resolve problems, it wasn't long before the damage had been done.

3. **Explain why it is not necessary for a program to be completely free of defects before it is delivered to its customers.**

   A program is usable when the client deems it usable. A certain level of low to medium severity bugs may be acceptable, and if it's a video game even severe bugs might sneak through. That is, it may be better to get a product to market first and fix minor bugs later, than fix the bugs and have no business because the development took too long. This pattern is very prevalent in the start-up world. Code might be cobbled together in a rush to stay afloat, and the team will have to refine their development processes and codebase later on.

   Another reason might be that money could be better spent adding features rather than bug fixing finished ones. A certain amount of code coverage may be acceptable to management.

4. **Explain why testing can only detect the presence of errors, not their absence.**

   It's not always possible to test or anticipate every input. Even if we could test every input and output, this may be unfeasible due to time or budget constraints. If I've tested five inputs, I can only verify bugs or no bugs for those five inputs, and there could be millions I haven't thought of. This is what can make formal verification of code an extremely expensive proposition, and even then there might be input that wasn't anticipated.

Ryan Filgas
Chris Gilmore
CS300
Homework 2
11-9-21

5. **A common approach to system testing is to test the system until the testing budget is exhausted and then deliver the system to customers. Discuss the ethics of this approach for systems that are delivered to external customers.**

   Depending on the budget the delivered system might be extremely bug prone. The severity and frequency of these bugs could cause a potential problem, but the impact on the user of the software depends on the type of software delivered. For example, in a pace maker there shouldn't be any bugs as this is a critical system. For a video game, a bug here and there isn't a problem, and a program crash is just a nuisance. That nuisance should be avoided, but it's not going to kill anyone as might happen with a pace maker or other critical system.

   If testing the code fully means running a test suite for a few months, that might not be acceptable for a client who wants the product out the door in two weeks under budget. If the system has 70% testing coverage and all normal inputs are covered, the client may not want to spend budget on potentially unlikely or minor errors. Fixing bugs costs money, and minor or medium severity bugs might be acceptable.

   When deciding how ethical shorting the projects testing budget is, the short answer is that it depends on how the end user is affected, and whether or not the result is acceptable. Software bugs shouldn't harm, kill, or cause material loss to users. Minor inconvenience on the other hand might be acceptable if the utility of the software outweighs the annoyance of a program bug.