Gherkin

**1. Write Gherkin tests for the program you wrote above. Use any Gherkin features or practices you want. Don’t write step definitions (i.e., the tests don’t have to be executable).**

Please see string\_sort.feature

**2. Explain in detail why these tests might be helpful in the future.**

-The first scenario verifies that the script produces any output. By definition, null sets are sorted, but should not count as a passing result for these input sets.

-The second scenario verifies that the number of elements in the input set is equal to the number of elements in the output set.

-The third scenario verifies that each element present in the input set is present in the output set, ensuring no elements were dropped during sorting. (This combined with the previous scenario also ensures that no elements were duplicated.)

-The final test is the classic descending sorting test, which verifies that each element is “less than or equal to” the previous element. Note that this test may not be meaningful if the prior corner cases are not covered.

Tools

**1. In your opinion, what’s helpful about version control systems? What’s annoying about them?**

Version control systems provide an organized means of collaborating on code changes, a history of all significant changes, and a means to rollback to a solution to a prior state.

Locking version control systems like SVN and PTC-Integrity are prone to “breaking the build” since their branching features are unfriendly, and require offline coordination when multiple feature changes are worked at ones.

Non-locking version control systems like Git are prone to merge conflicts, and have a single messy commit history. Gits diff system requires downloading a history of all changes that have ever been committed to clone a repository.

**2. What are some pros and cons of using Docker to develop, test, and deploy software?**

This is my first experience with Docker. For comparison, I’ve worked with Vangrant, manually set up local VMs and AWS EC2 Instances.

Given a Dockerfile, the setup time of a Docker build is by far the fastest compared to any of the other approaches. Dockerfiles are similarly lightweight compare to Vagrant files (which are far smaller than VM snapshots), but the environment setup is much faster since an entire VM does not need to be spun up. The use of OS-level virtualization reduces the overhead required by guest system hypervisors, allowing more resources for both virtualization and the host environment.

Docker containers do not achieve the same level of separation from the host environment or other containers as full virtualization. This is a concern when virtualized applications are mutually distrusting, or the virtualized application could be compromised.

**3. How do you choose which language to use for a given task? How did you choose the language 1 for the programming exercise above?**

I choose based on the size and complexity of the task, including the number of engineers required. For smaller assignments like this one, that require a single individual, a scripting language like Python feels ideal. For very large projects that require a number of engineers over time a language like Java is more suitable due to its extensive object orientation, modularity, and community of enterprise users.

Testing Methodology

**1. What’s the right role for QA in the software development process?**

QA engineers should be partner to developers throughout the entire development process, from requirements definition through final acceptance. Generally, developers should write their own unit tests, but QA engineers should develop integration and feature-level tests as a quasi-independent review. Software testing should occur throughout development: Once a final release candidate is readied, very little additional test development should be required.

**2.** **As a QA person, you have 2 weeks to prepare before your team starts writing software. What do you do?**

-Work with the development team to understand the application requirements, and ensure that they are testable.

-Write a couple of minimal smoke tests, even though application development has not begun. Start with a “Hello, World!” test placed in the appropriate fixture of your test infrastructure. If necessary, create a minimal mock using the technology used to create the target application and write some simple tests against it. This reduces lag time to get the first tests running.

**3. When is it appropriate to use automated testing? When is it appropriate to use manual testing?**

-Automated testing should be used for most feature acceptance testing, and later reused for regression testing.

-Manual testing should be limited to:

-Instances where automated testing is impractical

-Investigation of bug reports

-Exploratory testing

**4. Your dev team has just modified an existing product by adding new features and refactoring the code for old features. The devs claim to have written unit tests; you’re in charge of integration testing. Dedicated teams are handling performance and security testing, so you don’t have to. As is always the case in the real world, you don’t have time to test everything. What factors do you think about as you decide where to focus your testing efforts? How do you decide what not to test?**

-Most of the testing effort will be focused on verifying the functionality of the new features. Not only does this add the most value, but new features are the area most likely to be impacted by a misunderstanding of the requirements.

-Regression testing (ideally automated) is appropriate for the remainder of the features. Automated integration tests should not need change to track refactoring or changes to unrelated features. If a lapse in coverage must occur, it should be for the features that are topically unrelated to the changed components.