Testing Methodology

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

The kind of role for QA in the process is largely determined by the goals set out by the organization for the QA. In my opinion, QA should play a role across the entire software development spectrum, from ideation, to requirements and specifications generation, to architecture and high-level design, to implementation, to post feature completion, and all the way to release preparation. If bandwidth allows, also consider the pertinent aspects in operations and maintenance or DevOps. Even though QA and Dev teams work closely, the QA should maintain independency and avoid being overly influenced by Dev team to avoid group thinking traps. This provide a better opportunity to really be the extra set of eyes on quality.

Right Set of Goals. Obviously, the primary goal of the QA is to ensure both the right solution to the business problem, and the solution is built right. In other words, the software should address the right concerns; at the same time the solution architected, designed and implemented properly by following the contemporary best practices. The solution should provide the right functionality, performance, scalability and security. The overarching goal of the QA is to ensure the strategic objectives of the software solution are accomplished with the proper balance between quality and investment.

QA Engagement from Inception. In order to achieve the goals outlined above, the QA should get involved from the very beginning. This way the QA side has a better chance to ensure the requirements, architecture and designs are right for the business goals. With the adaptation of Behavior-Driven Development (BDD), there is a better opportunity for QA to get involved in defining the right set of requirements and specifications. This should also be the time where QA defines the high-level test plan and test cases, and sizes up the QA effort and develops the key milestones and testing project plan.

QA Involvement in Architecting and Designing. The QA should be reviewing and challenging all the artifacts produced in the early phase of a software project. However, this is one of the prominent challenges faced by QA teams as more often than not, the QA teams do not have the right talent and/or bandwidth to do so. In addition, once the requirements and specifications firm up, the testing plan and the high-level test cases should be updated to include more details or adjustments. In the meantime, either automated and manual execution procedures should be developed and/or specified in collaboration with development team (assuming the architecture is part of the dev team). Input should be sought from product management as well.

QA Automation and CI/CD. If the expertise on the QA side allows, QA should devote a healthy amount of energy and effort toward automating as many test cases as possible. Of course, the automation should make sense in terms of the actual test cases selected, the return of investment is adequate and not to replace mandatory manual test cases. Automated testing should also target those scenarios that have practical CI/CD contributions, and should include the non-functional tests such as performance, reliability, scalability, security and privacy protections.

Running Software QA. The QA team and Dev team should work closely so that the testing effort is in sync with the development and actual testing executions against run-time behaviors are done at as soon as the implementation is completed. This helps with increasing the efficiency of defect eradication as the developer's cost of context switch to fix issues is relatively small at this point in time. Usually, the QA team lags behind the Dev team for several weeks. When the defects are uncovered, the developers have to

spend significant amount of time to refresh their mind on the deep technical details to fix the bugs properly.

Defect Discovery, Reporting and Tracking. When test plans and test cases are executed, defects will be uncovered. The QA should follow a proper process to log the details of the bugs and opt to over reporting rather than under reporting. This will reduce the chances that certain issues would fall through the cracks. The QA should regularly triage the open defects and ensure the right amount attention is paid to them and the correction of the defects occurs at a proper speed. Regular defect reporting and tracking should be part of the routine communication strategy to inform all the stakeholders, from management, to development, to QA, and to DevOps/operations teams. A dashboard of quick summaries of a test cycle should also be made available.

Release Quality Gatekeeping. The QA should be an integral and independent part of the collective decision-making among product management, project management and engineering teams. In this phase, the QA and Dev teams should be treated equality when conflicts arise as to what should be show stoppers or not. The predetermined set release criteria should not be compromised to ensure the right user experiences with adequate performance, scalability, security and privacy parameters.

Support for DevOps and Maintenance. To help with organizational efficiency, the QA should play an important role in discovering and documenting the information for DevOps and maintenance needs. This is because it is well placed in the position to gain useful insights when integrating all the components to build a cohesive solution.

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

The short answer is to aim at identifying and prioritizing the test cases targeting the features to be completed in initial releases, and, at the same time, get ready for a more comprehensive test plan for the latter phase of the development.

This should be accomplished by first understanding and analyzing the available requirements, specifications, architecture and design documents as well as the planned features to be implemented in the early phases of the development process; then the draft test plan with all the key test cases should be developed to cover the early features and reviewed by the Dev team and other stakeholders.

In the meantime, the best test case candidates to be executed automatically should be determined. If expertise and bandwidth are available, the automated test utilities and manual testing scripts should be investigated and matured as the early features are implemented and released to QA.

At this point, the information and data for a more comprehensive test plan should be collected and documented and the time allows attention should be switched to work on that part of the work in parallel as much as possible.

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

The ultimate yard stick is the return of investment (ROI) of the automation efforts and the associated quality impact of the automated testing. At the same time, most aspects of the solution that interact with a human user should be the primary focus of manual testing, in addition to those infrequent or exploratory test cases and those have very low ROI for automation.

More detailed set of criteria for automated and manual testing are listed as follows.

Automated Testing

- Readiness: The test cases involved should be adequately understood and the testing procedures should be relative stable and are ripe for automated testing.
- Available expertise: Proper level of expertise for automation on the team should be present. Otherwise, the team might have a hard time to fulfill its routine test execution obligations.
- Accessible tools or framework: The tools and/or framework suitable for automating the target test cases should be easily obtained and leveraged. Unless explicitly set as the team's goals, internal testing framework development should not be picked up without proper justifications. Otherwise, the QA of the QA tool may be another distraction.
- Execution frequency: The regularly, routinely and error-prone executed test cases should be prioritized for automated execution.
- Effort level: The work involved to automate the candidate test cases should be reasonable when compared to the value or impact the test cases. If a lot of efforts are required, the automation priority should be lowered or delayed.
- Portability and reusability: If there is a high potential to reuse part of the automated code for other test cases, it would be a good candidate. This can be considered even when the effort level is otherwise high.
- High manual execution error rate: If the manual execution is error prone in cases where a lot of manual execution of interactive commands against the APIs. The automation should be quite effective and a morale booster to the team.
- Verifiability of automated test results: Test results can be properly and independently verified so that confidence in the automated test results are high.

Manual Testing

- Human facing features: User interfaces that are very user interactive and those that are routinely used by the human users.
- Human overall experiences: The overall usability, accessibility and visual effects of the static part of the application is another set of great candidates for manual testing. The automated testing will not be able to discern the human perceptions of the usability and esthetics.
- Exploratory testing: Those test cases that are either in the pre-planned and yet in the early stages of development, or those to be conducted by experienced QA engineers to identify additional test cases and scenarios. The key element here is the need of human intelligence and interventions.
- Automation challenged test cases. The test cases that are very difficult to automate should be in the bucket of manual testing effort.
- Other improper automation candidates. This includes those cases that can be automated, but the team is unable to automate in time; those whose execution is very infrequent; or those whose automation ROI is quite low or ad hoc in nature.

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?***

The overarching strategy is to maximize the ROI of the testing effort in this highly time constrained situation while minimizing the risk or taking educated risk. This can be achieved by conducting quick and yet adequate fact-finding, analysis and determination of areas of focus. The time-limited testing effort should concentrate on the carefully selected high-risk areas, features and the potential regression inducing integration issues.

Factors for focused effort:

- Note: Efforts should made to solicit and receive feedback from the Dev, DevOps and other stakeholders on the quick, documented prioritized plan developed with consideration of the following factors.
- Areas that could be impacted and are highly visible to the end users, high stability and security implications.
- If the features have not been tested or features that had been tested in the past but has undergone changes in the current cycle.
- The existing features that are not directly impacted but may be affected based on the past experiences, or newly identified based on the careful analysis, research and the feedback from the Dev team.
- Areas not adequately covered by unit tests. Review and analyze the relevant unit test plan, test cases and test results. Then try to identify the opportunities for QA to focus.
- Any broad impact from the changes in architecture, interconnected components or integration points are also fair game.
- Other potential side-effects from both the new features and refactored features on other features should be analyzed and risk-mitigated.
- Vulnerable areas based on the historical information where things have gone wrong or could easily go wrong.

Areas not to test:

- Hopefully, there is already a process in place to prioritize what not to test in this type of scenarios. If not, we probably need one so that the team can draw on with consistency to mitigate the risks of passing up the set of test cases not be included in the testing cycle;
- Areas that can be confidently identified as no or very low impact. For instance, if the changes are mostly on the front end with data model or data service changes, such as cosmetic changes to the layout and positioning of web elements, then backend or datastore functionalities may be safely excluded from the current testing cycle;
- Survey the source code change history for the current development cycle, and use the results and knowledge about the code module, components to determine what might be some other areas that can be safely ignored;
- Not likely impacted based on the analyses or well-educated guesses from the Dev, QA and/or DevOps teams.