Increase Confidence

Confidence – “the feeling or belief that one can rely on someone or something; firm trust.” https://en.oxforddictionaries.com/definition/us/confidence

Confidence dictates how much testing we feel we need to execute before we can sign off on anything we test. Our current confidence in our development team directly impacts how much test time we will take in order to feel our software is ready for sign off. The historical quality coming out of the development team dictates this level of confidence.

High Confidence – Just the right amount of testing is executed ensuring software can be signed off. (Note: This does not apply to mission critical software systems.)

Low Confidence – Based on historically bad code quality testers may over test even when code quality is good.

I believe this confidence level is very impactful to the speed in which we develop software. We might hear “QA is a bottleneck” but this is potentially due to historically low quality code causing testers to over test even when good quality code is being Verified.

To illustrate this point further see the approach below I came up with to test and ultimately verify bug fixes.

Example: A Mobile App Which Requires Users to Login
Imagine we have a mobile app which requires users to login. The fictitious bug we will be verifying is the following:
Title: Login Screen – App crashes after tapping login button.
Preconditions: App is freshly installed.

Steps to Reproduce:
1. Launch the app and then proceed to the login screen.
2. Enter a valid existing email and password.
3. Tap the “Login” button.
Result: App crashes.

Before Verification Begins
Once a bug is marked fixed it’s important we gain more understanding about it before

starting to verify its fix. To do this we ask the following questions of the developer who implemented the fix:
● What was the underlying issue?
● What caused this issue?
● How was the issue fixed?
● What other areas of the software could be impacted with this change?
● What file was changed?
● How confident is the developer in the fix? Do they seem certain? Even this can somewhat impact how we test.

* Special Note: Remember we need to gain context from a developer but as a tester you're not taking direction on exactly what to verify. This is your role as a tester. Of course if a developer suggests testing something in a certain way you can but it's your role as an experienced tester to use your mind to test a fix.

Now that we have gained a full understanding of how the bug was fixed let us start by verifying at the primary fault point (Exact steps listed in the original bug write up). Below are the high level verification/test ideas starting from very specific checks working outwards like layers of an onion. Notice as we execute more tests and move away from the primary fault point our confidence level in the fix is increasing.

Test Pass 1
● Exact Software State : Follow exact "Preconditions". In this case "App is freshly installed".
● Exact Input : Following exact steps listed in bugs "Steps to Reproduce".
● Verify app no longer crashes.
● We could stop here but we would not have full confidence that the bug is fully fixed and that we haven't introduced new knock-on bugs.
Moving another layer away from the fault: Our confidence in the fix is increasing

Test Pass 2
● Varied State : App is not freshly installed but user is logged out.
● Exact Input : Following exact steps listed in bugs "Steps to Reproduce"
● Verify app does not crash
Moving another layer away from the fault: Our confidence in the fix is increasing

Test Pass 3
● Varying State – After logging out/restarting app and clearing app data.
● Varying Input – Missing credentials/Invalid credentials

● Verify no unexpected behavior
Moving another layer away from the fault: Our confidence in the fix is increasing

Test Pass 4
Test features/functions around login such as:
● Forgot Password
● Sign Up
Moving another layer away from the fault: Our confidence in the fix is increasing

Test Pass 5
Moving one final layer away from this fix we enter a phase of testing which includes more outside the box type tests such as: (Note: I love this type of testing as it's very creative)
● Interruption testing – Placing app into the background directly after tapping the login button.
● Network fluctuations – Altering connection while login is taking place.
● Timing issues – Running around interacting with UI elements at an unnatural speed. Example – Rapidly tapping the login button then back button then login button.

At this point our historic confidence plays a role in whether we continue to test or we feel the bug is fixed. If QA's confidence is low we could end up spending too much time testing in this final test pass with little to show for our efforts.

How is Confidence Lowered?
● Initial code quality signed off by development is low. As testers when we begin testing a fix which has been signed off as ready for testing, we will often gauge its quality based on how quickly we discover a bug which will need fixing.
● Repeated low quality deliveries out of development can make testers correctly over test because it's necessary. If bugs are found routinely very quickly in software we test naturally we are skittish in signing off future high quality Work.

This can lead to over testing even when code quality is delivered in a high quality state. This over testing won't provide anything of value. Don't get me wrong you will find bugs but they might end up being more nice to know about then must fix issues. All software releases have bugs. It's our job to identify high value defects which threaten the quality of our solutions.

How Can We Boost Our Confidence?

I believe we can’t perform “just-right” testing unless our confidence in our development teams is reasonably high. We need to make sure baseline quality is established before any “just-right” manual testing can take place. How do we do this?

1. Test automation is a perfect mechanism to establish a quality baseline. “Checking” to ensure all basic functions are working as expected.
2. Shift left into the trench and work with developers as they are implementing a feature so you can ensure initial quality out of development is higher.
3. Measure your testing efforts to ensure you’re not over testing. Learn to know that sweet spot of just enough testing.
4. Expose low quality areas – Retrospectives are ideal places to bring up quality issues with the larger team. Let them know you don’t have confidence and need something to change to boost it back up.
5. Slow down – Oh no we can’t do that right? Yes we can and should slow down if our confidence is low.