Testing in Software Development Life Cycle Quality Management

You may remember this line from the movie Forrest Gump: “Life is like a box of chocolates…., you never know what you’re gonna get”. When the quality of an application build becomes like what you get in a box of chocolates, it is time to start thinking about Software Development Life Cycle (SDLC) Quality. No matter how hard the testers try, there is not much that you can do to guarantee quality when the code quality is unpredictable.

I started my career as a hardware engineer, before switching to the field of financial applications. In custom application development space, testers usually compare testing to an art as opposed to a science; mainly because they rarely know what they’re gonna get. However, in hardware engineering it is different; testing has always been a consideration in your system design. Design for testability is a well known concept in hardware engineering. You have to design and build components which are independently testable. This makes it possible to find defects where they happen. The sooner you find a defect, the cheaper the fix would be. Hardware engineers have tried to build quality into their development process, and they never think of testing as an art, rather a science which influences the design of the final product.

Software development has come a long way with the introduction of Test Driven Development (TDD) and Agile methodologies. TDD has moved thinking about test design to the left in the project cycle. With TDD, unit test cases are written before the code is developed and the code is considered complete when all unit test cases pass. This at a minimum tells the testers what they’re gonna get. As opposed to the non-TDD methodologies, which define the scope of unit testing to testing every line of code written, which might not be the same as testing every requirement to be delivered.

Not every interpretation of TDD takes it to the level of Test Driven Design though, i.e. testability considerations influencing the design of the application. Depending on the complexity of your application architecture, testable design considerations might be crucial. For example, some of you might have experienced environment level defects which are sporadic and sometimes very hard to reproduce. A good example can be environment level cache session bleeding happening to concurrent transactions which might be hard to reproduce, but can be disastrous in production. Testability considerations in system design can help testers with identifying root causes of such defects more easily.

Negative and defensive code testing must also be some of the considerations for design for testability. If most of your application defects reported in production is due to negative cases, then code design for easier negative testing should be a consideration in your application design. Using hardware design best practices make sure your system components are individually testable to a point that testing and finding component level defects do not require a fully integrated

environment. This might be a stretch for many complex legacy applications but it is an achievable goal and certainly the right direction to set for your application design. A more complete component level testing will also reduce the time you spend in end to end environments, besides eliminating the waste in the process due to many iterations of build and redeploy.

So how should we go about improving SDLC quality? There is no silver bullet or one size fit all solution to this problem. Organizational and product maturity are two of the major factors defining your shortest path to SDLC quality. Going from current state to the future state may require changes to roles and responsibilities and most likely to the company culture as well. For example Agile methodology success depends on very close IT partnership with the business. TDD chances of success increases when team members can play multiple roles of systems analyst, tester, and/or developer depending on the phase of the project. For example, when business requirements have to be captured in the form of business test cases, your ordinary tester or systems analyst may not have the right skills to play the new role needed for TDD.

To meet design for testability goals, systems architects and senior developers have to put the tester hat on and work very closely with testers. Testability and production readiness requirements have to be considered in system design requirements. Also during test design phase, application architects must play a key role in ensuring test coverage completeness as test design reviewers at a minimum.

Once you start going down the SDLC quality path, you have to be able to measure key process indicators and iteratively improve the process. There are many steps you can take to improve your software development life cycle quality. Depending on the size of your organization and product, you can right size the changes. Some of the low hanging fruits are: 1) think of testing before you code, 2) think of testability, production readiness and availability when you design, 3) implement an automated unit testing solution, 4) rethink and redefine the roles and responsibilities of the testers, systems analysts and developers in your team, 5) make sure you measure the right metrics for constant process monitoring and improvement, 6) work towards Continuous Integration (CI), or at a minimum make your build process as lean as possible, if CI is not an easy goal to achieve.

As mentioned before, the level of emphasis you should put on every one of the above steps depends on the maturity of your product and organization, for example the waste in your build and deploy process may define how much emphasize you should put on unit test coverage of data scenarios and permutations. Data scenario testing may be pushed off to system integration testing without too much waste in the process, if you have a CI in place. Once you start moving in the right direction, key progress indicators (KPIs) such as first time defect ratio, or defect leakage can indicate where in the process you can apply fine tuning.

Hopefully, with following the above recipe the quality of your code would be better than what you would get in a box of chocolates.

About the author:

Amir Hashemi: Amir started his career as a hardware engineers. He worked on VHDL compiler based simulators and testing during his Masters program. He holds a Ph.D. in computer science with a minor in digital signal processing. For the past 10 years Amir has worked as an information technology and application development architect in the finance industry.