Quality Begins with Design

Jeyasekar Marimuthu

Jeyasekar_marimuthu@mcafee.com

Abstract

In most software development organizations, QA has been a passive member in the design phase. QA's role has been assumed to begin only when the implementation phase kicks off. Had QA not missed capturing some of the valuable details during design phase, wouldn't their role be much effective than now?

This paper is about involving QA from the design phase. The amount of gain that organizations will benefit by having a QA architect is explained in this paper. There are some key advantages if QA penetrates deeper in design. Some of the noteworthy data points are abstracted in this section.

QA's involvement in the design phase will help the QA team understanding weightage of each user story/ theme/feature-group in terms of complexity, inter dependency, number of functions involved, distributed coverage across various layers of the application and the required number of unit test cases.

QA involvement in the design phase will also help detecting issues in advanced stages of the project and the same will help estimating QA costs with better accuracy.

In design oriented QA planning, test plans will be created in accordance with code flow. Test cases will be executed in a systematic approach. Test cases will be optimized as required by the design. A lot of guess work during QA planning will be avoided. Test ownership can be effectively assigned when the fundamental design is known in advance. Sometimes having a single QA owner for common- code (designed to cover multiple stories across different feature groups) can help avoiding multiple owners.

The efficiency of automation significantly increases when QA knows the design well. Effective automated cases will increase the code coverage. Also QA will gain visibility into the non-functional requirements like fault tolerance, extensibility, reliability, maintainability and availability.

How can QA begin their contributions to design? Let us see in the sections below.

Biography

Jeyasekar Marimuthu is a senior technical lead with 13 years of development experience, currently working with McAfee India Center at Bangalore. Over the past 7 years, he has been on a technical leadership role involved in software development. Out of 10+ product releases, his collaboration with QA has given him wisdom in implementing Quality in much advanced level.

Jeyasekar has worked with a number of security management products for enterprise in McAfee. His development contribution on Middle ware, database technologies over a decade has helped seeding some good thoughts at work. Jeyasekar has a Master of Computer Applications degree from Madurai Kamaraj University of Madurai, India.

Copyright Jeyasekar Marimuthu May-01-2013

1. Introduction

In the past, the primary role of quality engineers was to gate-keep releases and to certify products against a set of quality tests. The products that followed a waterfall model took a hit with a huge number of defects towards the end of release cycles. Since a large amount of use cases were addressed in the end, the load on quality staff used to be high and the same has resulted in creating a huge backlog of unaddressed defects.

Also, it has been realized that defects that were pushed to later project cycles incurred a greater cost than the ones that were caught earlier. As per corporate records, the defects caught in early cycles have reduced customer call volume by 20% to 30%. This metric has been proven in my last two projects that I delivered in my organization.

When companies shifted their development model from waterfall into agile, the defects were caught and fixed within the iteration cycle. The agile model helped the engineering teams to correct flaws and issues within the time window of iterations. This approach has distributed the QA workload more evenly across the project.

Though the situation had improved for QA in terms of distributed load, the following issues continue to bother. In Agile, QA deals with functional use cases by means of stories and tasks. Each story gives them an idea on what is being delivered, not how it is delivered. Implementation details are unknown. The focus is only on results. For example, if the story states “user should be able to add the values on click of ADD button”. QA traditionally tests whether the addition is implemented correctly. QA never had an insight how the addition is done. Had he or she known the “how” details, QA would have steered their testing more effectively.

Let us see the pain points that projects encounter by not involving QA in the design phase.

2. Pain points

SDET (Software Design Engineer in Test) staff work closely with developers, test APIs and automate white box tests. They have a fair idea on what code has been implemented and what functional areas are to be unit tested and automated. Though having a SDET improves the situation, the process is still not fully effective.

1. Their focus is mostly on the code that is being implemented. The pain point is that if the implementation is reworked a number of times by developers, their white box test cases also need to be reworked.
2. The focus is around unit tests. There are no tests to validate the design. For example, if a developer has written the following function,

    public int sum(int a ,int b)

The designated SDET writes unit tests for the above mentioned function with a bunch of input combination. What if the developer has overlooked that this function could perform summing of more than two numbers?

3. When there is a change in the fundamental architecture of the system, the whole bunch of automated and manual test cases are rewritten. The QA plan misses an important element of “QA architecture” document. In the presence of that document, the amount of changes in rewriting test cases could have been as minimal as possible, because a good design will always prevent any large scale changes.
4. Tests are tightly coupled with implementations.

5. QA tests are not given weightage based on the complexity of the task. The high complex tasks should be preempted while testing so that the project's risk is reduced.
6. Nonfunctional aspects like scalability, performance, and concurrency are given less focus while writing a test case. For example, if a SDET engineer is writing tests for the following function, he/she often overlooks having boundary conditions and concurrency checks tested.

   public int sum(int[] intArray)

   Expected boundary conditions: Populate the array with max number of integer values.

   Expected concurrency condition: Create as many number threads and invoke the function with random input samples and assess the behavior.

Though having a SDET improves the situation further, most of the white box cases written by SDET are around the implemented code. They only help developers to add additional unit tests or discover automatable functional tests.

Still, there is huge potential for advancing the quality process by introducing a Quality Architect or SDET right from the design phase.

Shouldn't the approach of a SDET be more design oriented and systematic rather than just being a unit test engineer?

3. Need for Quality Architect in the design phase

During the design phase, high level and low level designs are created against a set of requirements. In agile, stories are picked from a backlog and designs are done at modular levels. This makes easier for QA staff to understand the design and to derive QA specific designs. The design phase also allows architects to come up with detailed system diagrams that capture the following:

1. Logical breakdown of functionalities against functional and nonfunctional requirements.
2. Separation of concerns - An established way of reducing complexity.
3. Interface designs for internal and external facing APIs. Internal facing APIs are meant for SUD (System Under Development) whereas the external facing APIs are meant for external systems that want to integrate with SUD.
4. Defining nonfunctional system attributes such as Fault tolerance, specific set of tradeoffs, extensibility, reliability, maintainability and availability.

In order to integrate quality with the design phase, the QA architect should prepare test plans for each of the item mentioned above.

Organizations seldom have a designated owner for creating quality designs for Systems Under Development. In the absence of such role, it is important for organizations to designate a Quality architect or a senior SDET to take control of quality designs. The designated owner will work closely with product architects and other design owners and will create a quality design document for the System Under Development. The owner of the quality design document is expected to have wealthy knowledge about the system both functionally and technically.

4. Responsibilities of a Quality Architect

The proposed approach is to have a quality architect with the following responsibilities:

1. **Quality document creation:** The Quality design document needs to be created by deriving it from architecture documents and development- design documents. The number of QA architecture/design documents should be equal to the number of development documents. There should be a one to one map between the number of Development and QA design documents.

2. **Focus:** The focus of quality design documents should be on possible automation areas, required Build Verification Test (BVT) cases, Integration tests, Performance tests and User Interface tests, at a high level. Basically the QA design document will capture all sorts of test plans with granular details. The test plan will include coverage on system testing, black box testing, white box testing and BVT. When the document is complete, QA will have a fair idea about the length and breadth of tests the project will need to have.

3. **System testing:** The quality design document will give a fair idea about how tests are spread across multiple sub-systems. This knowledge is helpful to perform system testing effectively.

4. **Education:** The Quality architect needs to educate the team on how does development designs translate into QA focus areas in terms of Complexity, Inter dependency, Number of lines of code, Distributed coverage across front end (User Interface), Middle tier (Java, .NET etc.) and backend (database).

5. **Organized test plans:** The QA architect will help the QA team to execute their test plans in an organized way and in a timely manner.

6. **Synchronized design:** The QA architect should ensure that the QA plan is well in sync with development designs.

7. **Capturing actions items:** The QA architect should map all non-functional requirements to QA centric action items.

8. **Guidelines to Test engineers:** The QA architect should give pointers to QA engineers who will estimate individual user stories. The pointers can be on complexity, number of function points, number of interfaces etc.

5. System Design Document versus Quality Design Document

The system design document defines the architecture, component, modules, interfaces and data for a system to satisfy specific requirements. The proposed QA design document will be an extended version of the system design document with QA perspective added to each section. Some of the key highlights of the quality design document are below.

1. **Call out dependencies:** All use cases have a dependency section if applicable. Those dependency sections will denote how the module is to be approached.

2. **Define complexities:** The complexity of each component in the document is assigned with weightage number and the stories are categorized according to their assigned weightage. The other way to categorize the stories is to categorize them with complexity levels such as low, medium and high. Based on the weightage or complexity levels, QA planning needs to be done accordingly. If the weightage is already assigned by the development team, then QA can follow the same weightage numbers. It is always recommended that QA revalidates the complexities before rolling out their test plans.

3. **Test optimization:** Test cases (both manual and automation) are optimized to a greater extent. When there is reusable code that is serving multiple modules, automating one of them would pass all the tests. There is no need to write multiple automation tests for each user story. Optimization of test resources results in significant cost savings.

4. **Technical knowledge:** QA gains insight into deeper system design by working against low level design. Since each user story design connects with a distributed set of components (Front end, Middle tier and Backend), test ownership gets assigned breadthwise. The quality design document captures the details that are not captured by the system design document. For example, if a system design document that describes the functionality of use case “Registration”, it will capture how the request is initiated, how the business component processes the request and how the response is obtained. The quality design document will have the same details with additional test information. The additional points captured include the quality impact in terms of possible UI tests, business component tests, response tests, non-functional tests, automation opportunities, integrations tests etc.

5. **Cost savings:** When SDETs/ test engineers log defects around design, the benefit that the defect brings is much higher than the ones caught in later cycles. One uncaught design defect may have the potential of introducing multiple functional defects across the system during later cycles of the project.

   For example, if a developer has designed the following function to sum two numbers,

   public int sum (int a,int b)

   QA can file a defect asking “how can this function support more than two numbers”.

6. **Well-defined boundaries:** System boundaries are well-defined. Since QA's design document covers all parts of the system, QA knows what tests make the system 100% covered. System tests are derived from integration-interface designs. Hidden parameters like “network throughput”, “connectivity”, ”system load” are very well known in advance.

7. **Collaboration:** Tight collaboration between development and QA: Both development and QA units become interlocked as both know the system design equally well. QA can also push for critical functions to get prioritized so that there are fewer surprises at later cycles.

8. **Automation Plan:** The quality design document gives a clear picture, in the design phase, of what can (and can’t) be automated. A number of redundant automation cases can be eliminated when QA knows the design well. In time-crunched projects, QA can automate only the necessary or important cases rather trying to automate the whole number of test cases.

6. QA design diagram: Illustrated from a use case

QA design diagrams are part of the quality design document. Development teams may use different tools and techniques to prepare their design. It is the QA architect’s responsibility to understand each representation (like block diagram, use case diagram, sequential diagram, and class diagram) and to come up with a QA equivalent design.

For example, if developer uses a class diagram, QA should understand the following to extract QA designs.

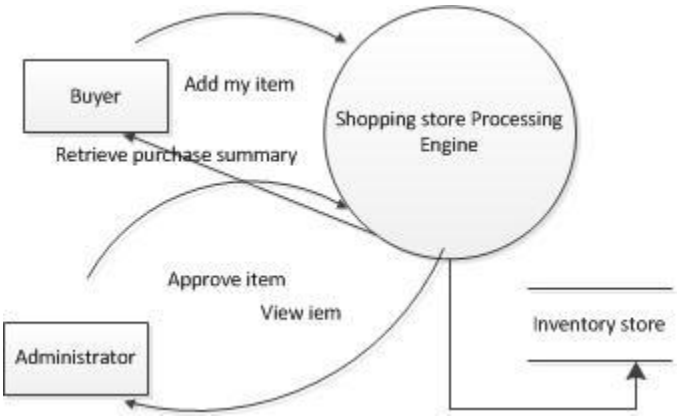
1. Number of public members
2. Number of public APIs
3. External dependencies for the class
4. Relationship with other classes (one to many, many to one, many to many)

5. Possible number of automatable APIs
6. Possible number of critical APIs that has many to one relationship

As an example, let us consider a shopping cart application.

At a high level, the activity diagram looks as below.

1. Buyer adds his item.
2. Buyer retrieves his purchase summary.
3. Administrator views the item ordered and approves.
4. The shopping processing engine or business component services the above 3 activities by persisting the user(s) data into inventory store.



Development design

1. Buyer object sends the shopping information to shopping cart.
2. The shopping store business component identifies the buyer by his session id and adds buyer's shopping data to inventory store.
3. Admin object requests for shopping info added by buyers.
4. The shopping store business component validates the admin's id and retrieves purchase requests from inventory store.
5. The shopping store Business component constructs purchase object and sends it back to admin.
6. Admin object views the info in the front end (view JSP) and clicks on approve
7. The approve request is submitted to shopping store business component.
8. The shopping store Business component commits the order.
9. End.

QA design mapped from the development design

1. Make sure unit tests are written for buyer object in order to check whether the object contains valid values like session id and shopping parameters. The unit tests can cover with positive values, negative values, lower boundary value, upper boundary value, empty value etc.
2. Write a test API to check whether the business component commits the data into data store successfully.
3. Check the maximum number of requests that the business component can handle (nonfunctional Performance test).
4. Check the correctness of admin id and its corresponding request object.
5. Validate the validation logic by exercising with a number of unit test values.

6. Write unit tests for checking the completeness of the purchase object.
7. Ensure all fields in the UI are validated.
8. Write a test API to check whether data submission was successful.
9. Check relevant tables in the inventory store and make sure the commit operation happened
   End.

Deriving non functional tests from QA design

1. Interoperability tests covered between Buyer object, Business component and admin object
2. System availability, durability and maintainability test cases.
3. Load test, performance test coverage.
4. System test boundaries: API test, serviceability of the component in standalone and distributed environments.

7. Conclusion

It has been discussed what is lacking in the traditional QA process. The pain points were analyzed in detail. The merits of QA involvement in the design phase were discussed in detail.

In order to overcome the existing challenges, it is important that QA should address the items that have been lacking in the design phase. Organizations should consider having a QA architect to handle the design aspects of the quality process.

Some of the key learnings from this paper are:

1. QA should approach each story based on its criticality, complexity, and interdependencies.
2. QA should prepare an architecture document capturing all important design decisions that can influence their testing.
3. The designated QA architect should guide his team to derive tests from designs.
4. Involving QA in the design phase results in well controlled QA plans with significant cost reduction. It also allows creating extensible QA designs.
5. QA gains adequate system knowledge when they become co-owners of designs.

8. References

1. Wikipedia, “Systems Design”, https://en.wikipedia.org/wiki/Systems_design (accessed August 01, 2013)