Quality Engineering Considered as a Helix of Semi-Precious Quadrants

Vivek Sahai Mathur

mathurvs@outlook.com

Abstract

How has the scope and role of Quality Engineering changed over time, and what does it mean for you?

Software applications have moved a great deal from well controlled, procedural code on green screens supported by a single processor, single database with a limited and trained user base. The present day utilizes multi-device, multi-user, multi-threaded, event-driven workflows, with distributed data, multiple compute instances and fractured business logic.

All this complexity is compounded by increased awareness and implementation of security and privacy regulations and industry mandated best practices.

The increasing complexity has fundamentally and irreversibly changed the definition of "Fitness for use" of software, and non-functional compliance requirements.

The end result has been an evolution in the roles and responsibilities of pod members, and especially in the relationships between the quality team and the developers. This paper casts the trends in software and team dynamics in terms of the software functionality quadrant, and also links it to the classic Hugo & Nebula award winner story by Samuel Delany.

Biography

Vivek S Mathur started his IT career in 1997.

Vivek has 30+ years of varied work experience ranging from the Government to software product and service companies. His focus has always been on setting up robust processes, and seeing their impact on the quality of deliverables and productivity of the workers. Making changes that delivery teams internalise and adopt, and add value to the organization.

Vivek has set up core software development processes for product and service companies like McAfee and Intelligroup, and has set up and led large delivery teams for multiple customers.

Vivek has always tried to give back to the community, and has been associated with STeP-IN Forum (www.stepinforum.org) actively for the past 11 years. He is currently the President, STeP-IN Forum

Experienced in Information Security, Data Analytics/ Big Data projects, and Software Quality, and in merging the three into the Product Lifecycle Management to achieve highest productivity and predictable delivery while leveraging the partner ecosystem.

Copyright Vivek Sahai Mathur 9 Jun, 2023

1 Introduction

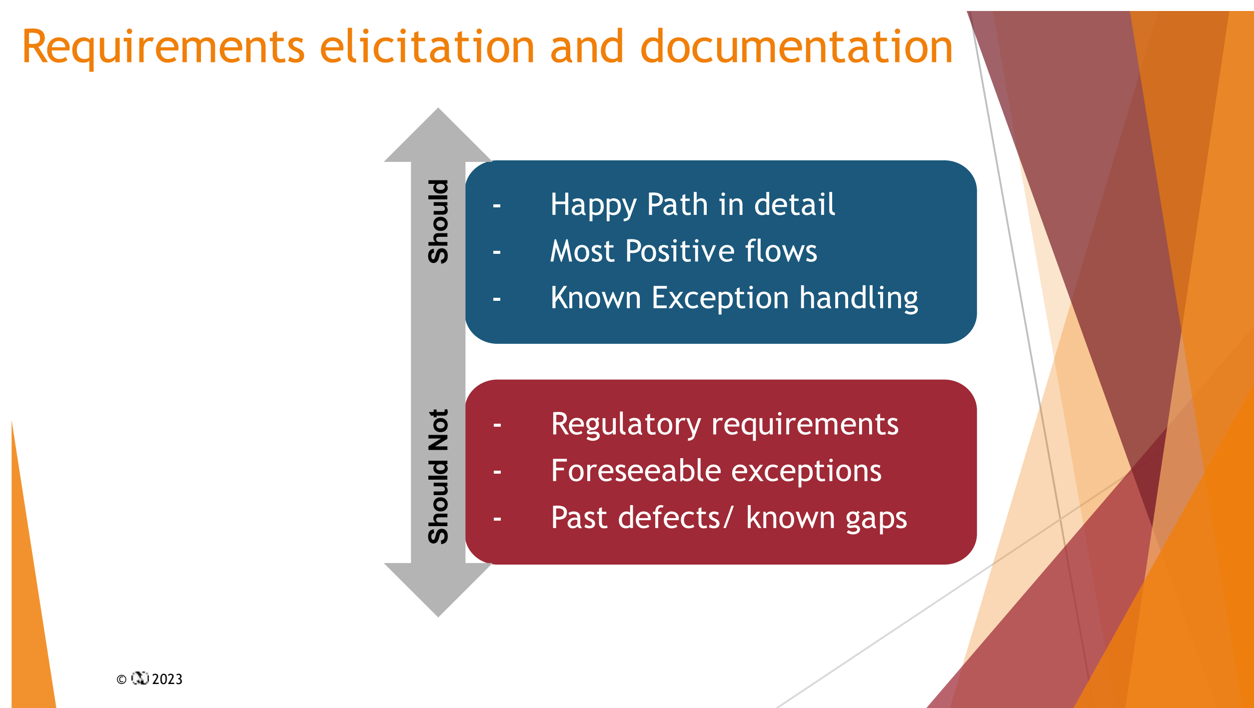
Software applications have been in constant evolution since the first basic automation of work procedures. The impact is apparent in the way we use software, up to the present day, when the application is ubiquitous and central to the business transactions. Every step in the workflow can take place in a different environment. Distributed compute and data storage alters the way applications are built, and the way the transaction integrity is retained.

This paper casts the trends in software and team dynamics in terms of the software functionality quadrant, and also links it to the classic Hugo & Nebula award winner story by Samuel Delany.

1. Evolution of software complexity
2. The expected trends and the roles qualiticians can hope to face in the future
3. How to mature the relationships within the pod and improve the output quality.

2 The evolution of software complexity

Traditional application development lifecycle followed the waterfall approach, in which each step had to be completed before the next one could start. So the requirements gathering and analysis preceded the remaining stages.

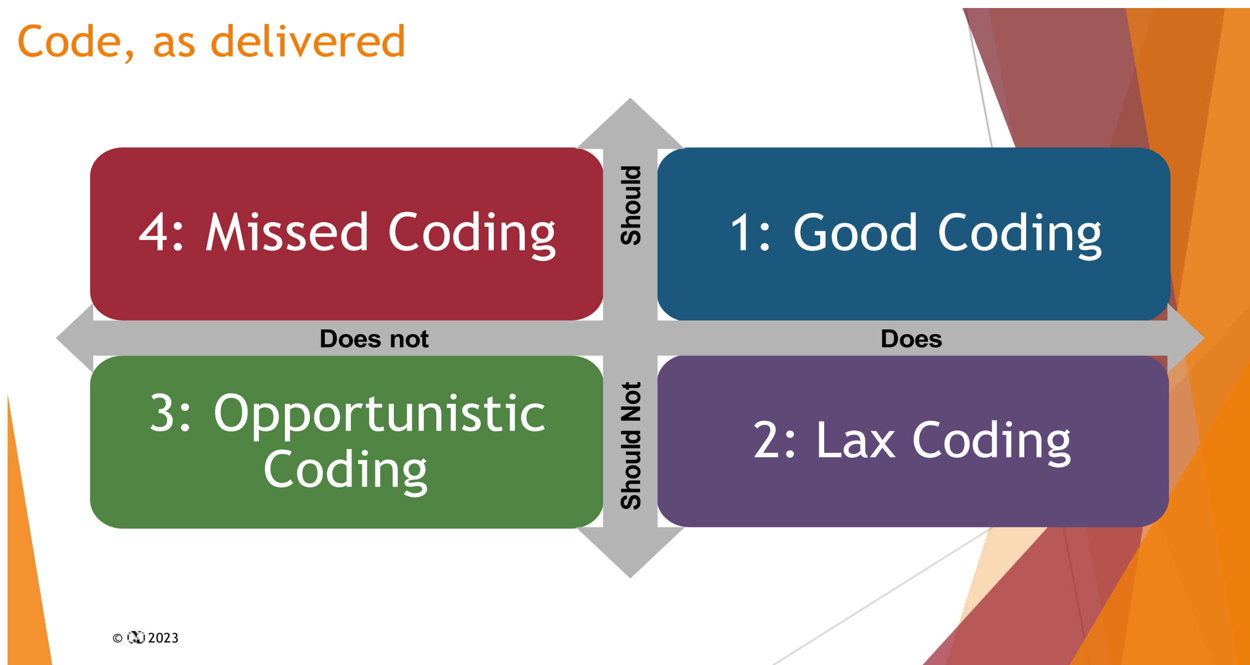


The requirements identify what the software should do and what it should not do, on a single axis. The code that is written should conform to both aspects.

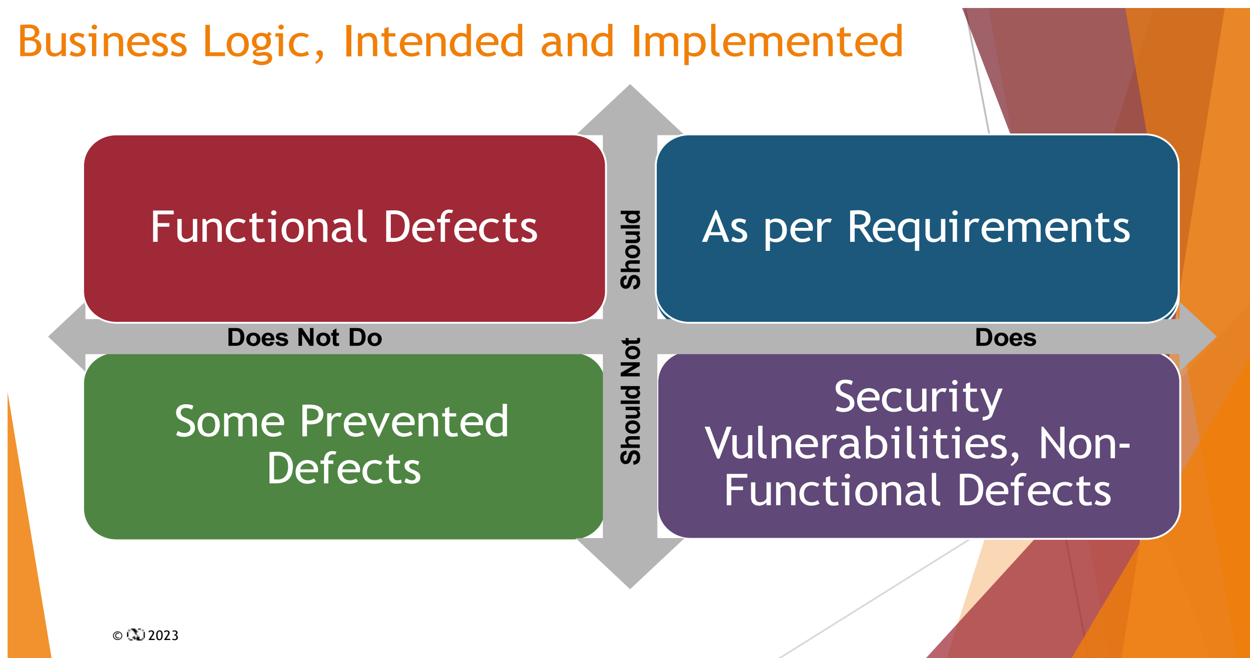
Due to paucity of time and effort available, and the need to ensure that the MVP (Minimal Viable Product) can be delivered at the earliest, complete delineation of the design and code is not possible, and gaps emerge. Taking the code developed as the second axis, we get the 2 aspects of whether the code actually implements the requirements.

The process of translating requirements into design and code leads to the first set of quadrants:

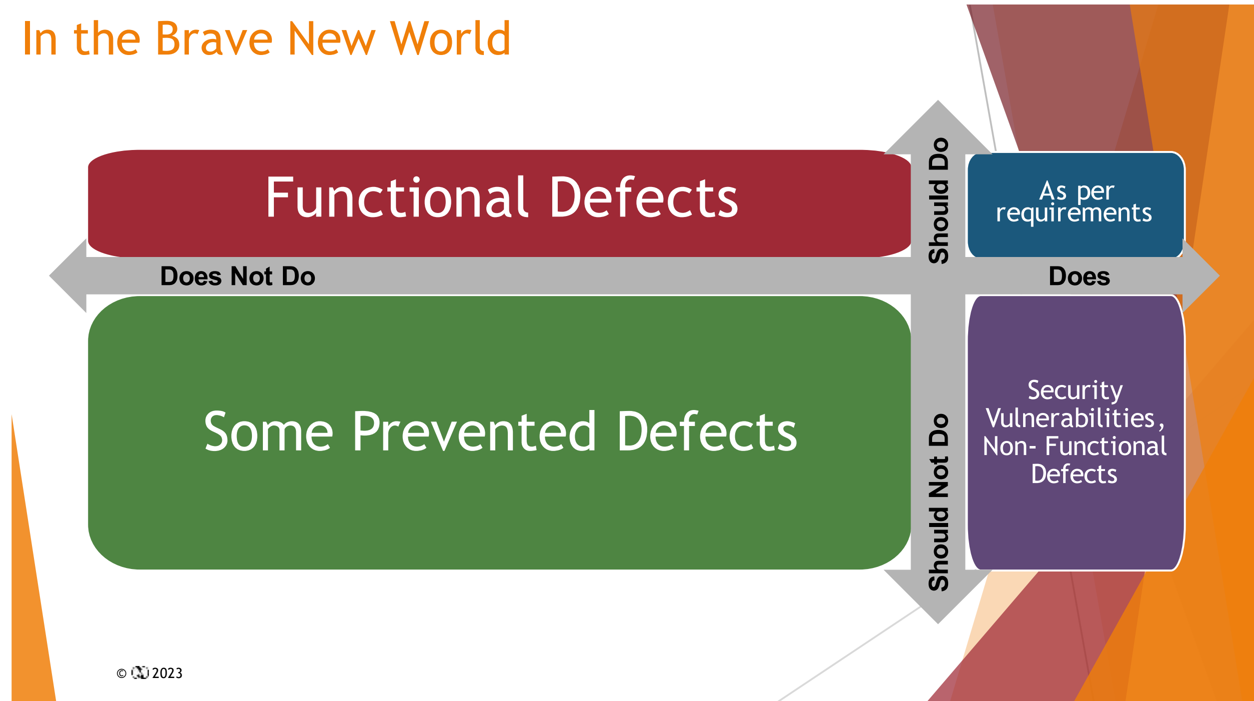
1. Code that does what is should do: This is the correct code, and covers the most common and straightforward functionality.
2. Code that does what it should not do: This is defective code, and may not be easy to identify, since the hidden functionality will not be visible during normal use of the application.
3. Code that does not do, what it should not do: This is the code built to address the 'negative requirements'.
4. Code that does not do what it should do: These are functional defects and should be identified through tests, if the coverage is adequate. Defects in corner cases may reach Production deployment and be hidden for relatively longer periods of time.



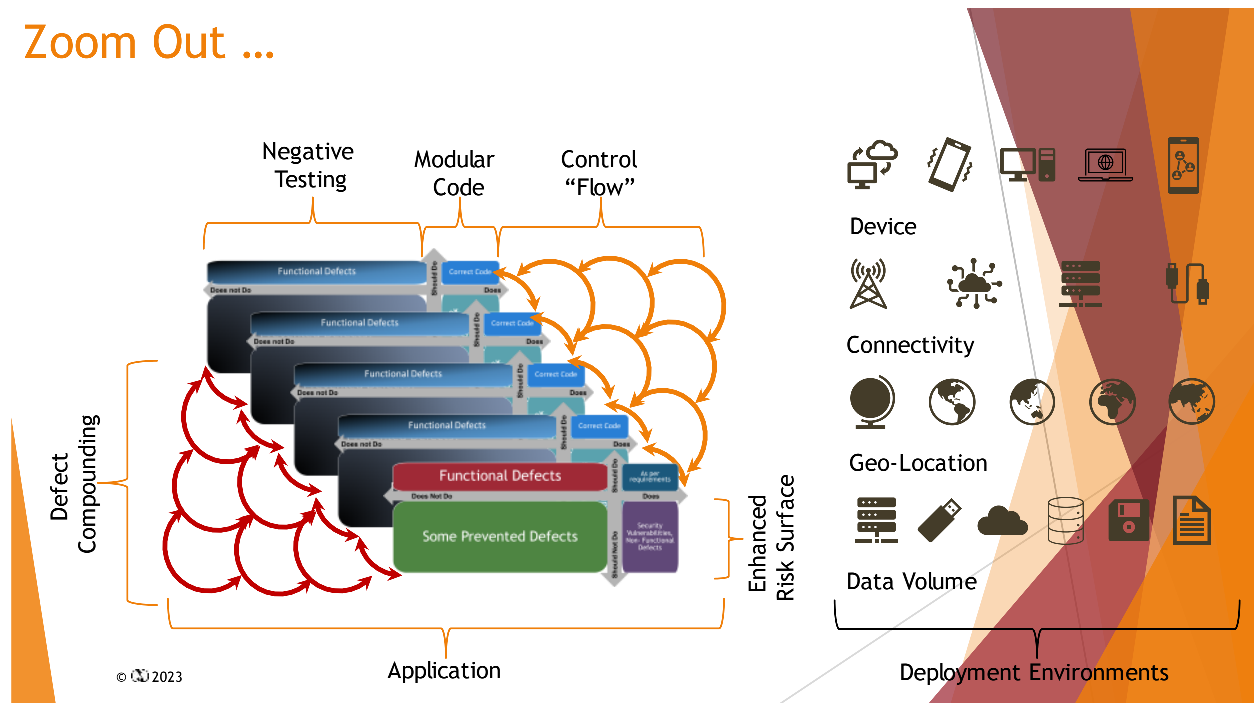
From the aspect of delivered functionality, the results are as follows:



While the above depiction covers the standard monolithic application, recent trends have reduced the functionality in each code module. This makes it difficult to ensure that negative functionality defects are completely covered, and the scope for both nonfunctional and functional defects that need to be avoided increases.



Since a lot of single function modules (e.g. Microservices) need to work together, the business transaction itself needs to jump between multiple modules to reach completion:



Taking each of the aspects pointed out in the figure in sequence:

2.1 Application

1. Modular Code: Expected to complete a single function in a robust and repeatable fashion based on the input parameters but without the transaction context. Needs to be able to handle all inputs correctly.
2. Control Flow: The transaction itself may bounce from one code module to another based on the events that are triggered. While there is a "Happy Path" that can be assumed to complete the workflow in the simplest way, there is no force to follow that path, and the deployment environments add another level variability.
3. Enhanced Risk Surface: The expected ability to handle the "Should Not" aspects increases.
4. Negative Testing: The requirement for Negative testing and assuring that incorrect logic cannot occur increases
5. Defect Compounding: As the event flow is unpredictable, the errors in the workflow can impact downstream errors and make the debugging and data correction difficult.

2.2 Deployment Environments

The transactions and workflows have to handle the changes in the following:

1. Device – desktop to mobile devices and back. Databases on-site to on-cloud.
2. Connectivity – switching between telecom network cellular towers to Wi-Fi or wired network, while retaining the transaction context and data.
3. Geo-Location – Regulatory requirements address data processing, data flow, and data storage of non-public data based on the geo location of the data subject, controller processor, or sub[-processor.
4. Data Volume – data processing can occur in multiple location and on multiple devices, necessitating the transfer of transaction data to follow the location (and network, device) where the transaction is being processed.

2.3 What future trends do we see?

The approach to software development will continue to evolve, and some of the trneds that can be identified are listed below:

- Supply Chain/ ecosystem – Outsourced components and reuse of the Open Source
- Constant upgrade churn – OSes/ Browsers/ Dev Frameworks will be upgraded, and the applications need to be built to accommodate these constant changes
- "My Name is Vuca" (Volatility, uncertainty, complexity and ambiguity)-changes in the real world scenarios that the software has to address will only increase with new technology and user interfaces (VR, Speech, brain implants?)
- "Would you like some more PII?" – The increasing regulatory controls on the use of personally identifiable data, covering the rights and duties of the data subjects, controllers, processors and sub-processors, and impact of the locations of data origin, processing, and storage will require more real-time monitoring to retain compliance with regulations.
- Built to sell – The software is not built cover a complete business transaction, but as a part of one.
- Edge Computing – technology that distributes the processing and data manipulation, along with the UX management necessarily slices the business process.
- Gen AI – still need to understand the impact this will have on IP, and the backdoors it can open up in the codebase.

3 The evolution of testing expectations

3.1 The evolution so far

Testing and code development have always evolved, and the relationship between the functions has also been changing through the years.

- Purpose built applications were tested by the developer as part of the coding process, and there was no separate testing phase in effect.
  Relationship – non-existent

- With the approach of structured coding practices, the focus for the tester was to achieve Verification and Validation (V&V, V-model, W-model, etc.) success. Test cases were created, with the objective of being MECE (Mutually Exclusive and Collectively Exhaustive) and leaving no requirement un-covered. The testing effort was clearly defined, and had temporal and sequential integrity.
  Relationship – unaware/ ignorant

- Test Automation for faster regression testing, Model based testing (MBT) approaches for faster test case development happened when the factory model of code development transformed with Agile approaches. Scrum Pods included a testing resource,
  Relationship – Condescending

- There were brief periods of adoption of Pair Programming, TDD, xTDD, etc. with a much closer scrutiny of coders by testers, and vice-versa. Dedicated testing positions in the scrum pod were eliminated.
  Relationship - Competitive

- Finally in the present moment, the tester role has split into in-sprint testing in the dev phase, and confirmatory testing at the end of the SDLC. The focus is on business domain experience and exploratory testing to trap corner cases.

Relationship - Cooperative

With the increased adoption of CI/ CD and more automated complex error finding, the tester needs to govern the code development process while validating the business functionality.

4 How the Helix explains the changing relationships

1. “Later HCE finds Arty the Hawk on his doorstep. Arty explains that he sought him out because their relationship was about to undergo a change. HCE is puzzled, but eventually realizes that Arty and he are about to become rivals. Arty will try to buy him out, then to kill him, because that is the way the world works. If he survives and prospers, he and Arty will eventually become friends because there will be more profit in cooperating than in competing. He tells Arty this, and Arty wholeheartedly agrees. He responds that HCE is starting to think holographically, just like Maud and Special Services. He departs, leaving HCE to contemplate his future.” – Wikipedia

   Alternatively:

2. “First they ignore you”
   “Then they laugh at you”
   “Then they fight you”
   “Then you win” – attributed to MK Gandhi

Copies may not be made or distributed for commercial use

References

Internet Hyperlinks:

- Hugo and Nebula Award Winners, 1968: https://nebulas.sfwa.org/nominated-work/time-considered-helix-semi-precious-stones/
- Plot Summary (last paragraph): https://en.wikipedia.org/wiki/Time_Considered_as_a_Helix_of_Semi-Precious_Stones
- Helix linkage to relationship evolution: https://www.lyricsondemand.com/tvthemes/thepartridgefamilylyrics.html
- Theory of Emergent Evolution: https://en.wikipedia.org/wiki/Emergent_evolution
- Quote attributed to Mahatma Gandhi: https://www.snopes.com/fact-check/first-they-ignore-you/
- My Name is Luka: https://en.wikipedia.org/wiki/Solitude_Standing -> Track listing