Quality Conscious Software Delivery

Lalitkumar Bhamare

Accenture Song

EuroSTAR 2022

BEST PAPER WINNER

EuroSTAR Huddle

eBook

EuroSTAR
Huddle

Welcome

At EuroSTAR, our core purpose is to help software test professionals to achieve their absolute full professional potential and to inspire them through community and collaboration to help each other.

From the EuroSTAR Huddle for testers wishing to learn and improve to the annual EuroSTAR Conference, we have been bringing testing and quality assurance professionals together since 1993.

We are delighted to present this eBook written by Lalitkumar Bhamare, which won the coveted Best Paper Award at the 20222 EuroSTAR Conference - receiving high praise from each of the judges.

Enjoy!

The EuroSTAR Team

Lalitkumar Bhamare

With over 14 years of experience, Lalitkumar (Lalit) is working as a test engineering and s/w quality leader. He has been working in various aspects of test engineering right from the shift left to production telemetry.

Lalit's specialty is to help organizations with their business and digital transfor- mation using testing and quality consciousness as tools for transformation. To achieve that he has created his own delivery framework named Quality Con- scious Software Delivery (QCSD). Lalit is known in the global testing industry for his contribution to the community in various forms such as through his non- profit publication named Tea-time with Testers , as Director at Association for Software Testing - USA, as an international keynote speaker and testing thought leader.

Over the decade, Lalit has been working closely with industry experts and leaders across the globe and has made an active contribution in taking the craft of software testing ahead.

https://www.linkedin.com/in/lalitkumarbhamare

www.talesoftesting.com

Abstract

Major digital and business transformations taking place in organizations warrant rapid development and innovations in the way software is delivered. Project teams do make efforts to deliver value at high speed without sacrificing quality. However, despite technological innovations and modern delivery methodologies, the triple constraint triangle of Speed-Cost-Quality remains. And when it comes to picking any two of them, the trade-off usually happens at the cost of quality. While it helps to monetize the business value of speed, offsetting the cost of poor quality isn't always easy and it can sabotage the benefits brought by speed. The trade-off at the cost of quality happens because organizations lack a proven approach for this balancing act. The steep cost of poor quality becomes very visible if we take a closer look at software failures in the industry. Therefore, quality needs to be seen as an opportunity. Good quality does not necessarily require additional resources if delivered through a proven approach based on quality consciousness.

This eBook offers one such approach to software delivery that puts quality at the center, uses testing education as a tool to facilitate it while promoting modern development methodologies and engineering practices that support faster and cost-effective software deliveries.

Table of Content

Quality Conscious Software Delivery.

The Cost of Quality!

Understanding the dynamics

The motivation for improving quality always starts with a study of the cost of quality.– Philip B Crosby[1]

The cost of quality! Read it again. What does it imply? It implies two things:

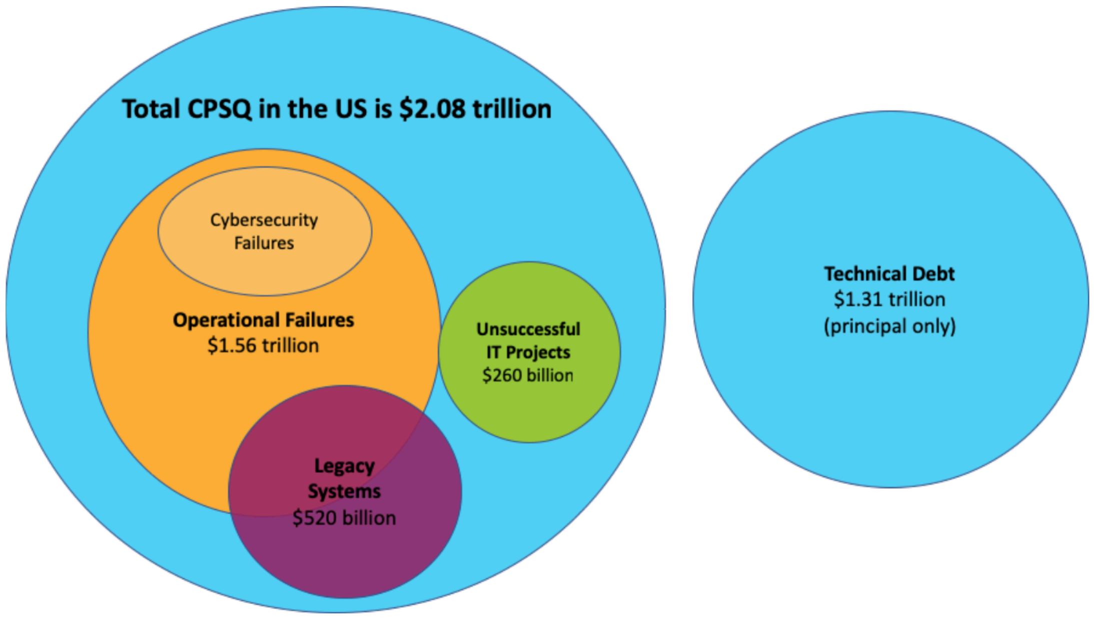
1. The cost organizations must pay if they compromise on quality, and
2. The cost organizations must bear if they are to deliver a quality product.

The organizational stance on quality is often the balancing act between these two cost aspects. Therefore it is important to keep both considerations in mind while discussing the quality and the cost.

Speaking of which, take a look at this interesting paper published by CISQ i.e. The Cost of Poor Software Quality in the US: A 2020 Report. According to this paper, the total *Cost of Poor Software Quality (CPSQ) in the US for the year 2020 is $2.08 trillion (T).* [2]

1. *Quality Is Free – The Art of Making Quality Certain – by Philip B Crosby – A Mentor Book - 1980*
2. *Another aspect worth noting from the paper is that the 2020 US figure for the software technical debt residing in severe defects that need to be corrected would be $1.31 T (minus interest) but did not include technical debt in the total CPSQ since it represents a future cost which is increasing (14% rise since 2018).*

CPSQ in 2020 in the US


Total CPSQ in the US is $2.08 trillion
Cybersecurity Failures
Operational Failures $1.56 trillion
Unsuccessful IT Projects $260 billion
Legacy Systems $520 billion
Technical Debt $1.31 trillion (principal only)

Let us pay special attention to three key findings from the CPSQ report, quoted here as-is:

- The largest contributor to CPSQ is operational software failures. For 2020, the CPSQ report team estimated that it is ~$1.56 T, a 22% growth over 2 years – but that could be underestimated given the meteoric rise in cybersecurity failures, and that many failures go unreported. The underlying cause is primarily unmitigated flaws in the software.
- The next largest contributor to CPSQ is unsuccessful development projects totaling $260 billion (B), which rose by 46% since 2018. The project failure rate has been steady at ~19% for over a decade. The underlying causes are varied, but one consistent theme has been the lack of attention to quality.
- Legacy system problems contributed $520 B to CPSQ (down from $635 B in 2018), mostly still due to non-value-added "waste."

If the estimated cost of poor software quality in the US alone is so significant, what must be the case for the rest of the world? Could it be that these estimates are a symptom of a lack of organizational attention to quality, or there might be something else that is being overlooked?

The answer to that question lies in the second aspect of cost mentioned earlier. The cost organizations must bear if they want to deliver a quality product. But what makes quality a costly affair? Does paying attention to quality mean adding more people to do that job? Does it mean having to invest extra and dedicated time? Does it mean having to invest in costly tools and technology?

The answer to all of those latter questions is mostly yes unless we know how to deliver quality software otherwise. Are there other ways to do so? Definitely yes and this booklet discusses one of those. However, before discussing the approach it is important to reflect upon certain things and discuss those in greater detail.

Quality does not exist in a non-human vaccum. Every statement about quality is a statement about some person(s)- Jerry Weinberg

Considering the business benefit of speed a.k.a. “time to market” and the lower production costs, the only choice organizations are left with is a balancing act trying to deliver value at high speed without sacrificing quality.

Considering the business benefit of speed a.k.a. “time to market” and the lower production costs, the only choice organizations are left with is a balancing act trying to deliver value at high speed without sacrificing quality. Despite existing for over several decades, we as a software industry still have not figured out how to find that balance right. Quality always pays the price.

What makes this balancing act for quality so difficult? Apparently, the answer lies in the very nature of quality itself.

Quality is relative and often difficult to make all-encompassing.

That is because what quality means to some person could be different than what it means to you, and it could mean completely different things for someone else.

“Quality does not exist in a non-human vacuum. Every statement about quality is a statement about some person(s)”.[3]

3.Gerald M. Weinberg. “Quality Software: Volume 1.1: How Software Is Built”.

Quality and The Balancing Act

What makes it so hard to achieve?

In his book Quality Software: Volume 1.1: How Software Is Built, Jerry Weinberg discusses this beautifully with a story about his niece and the book she wrote. Terra's (Jerry's niece) book got published with several gross typographical errors. And it happened because of a bug in the word processing software she was using.

Luckily, Jerry was consulting this company that produced the word processor back then and he discussed the problem with the concerned manager. Jerry learned that the manager knew about the problem but decided not to fix it anytime soon since the number of users affected by that problem was too small to matter. And by fixing it, they might have introduced the worst bug that could affect many users.

In this story, the severity of the bug for Terra was worse. At the same time, it was less concerning for the manager of the company that produced it. And Jerry, for being connected with both, had to constantly fight the dilemma we all can understand.

When making decisions about quality, most of the decision-makers face such a dilemma.

Therefore, when it comes to catering to the needs of many users who understand quality differently, the balancing act of quality can become quite difficult.

If the goal is to deliver value without compromising the quality, it is quite imperative that we need to understand what connects value with quality and vice versa.

Jerry Weinberg's definition of quality does it quite brilliantly:

"Quality is value to some person."

By "value," Jerry means, "What are people willing to pay (do) to have their requirements met."

Quality and The Consciousness

Even though the definition of quality solves the emotional dilemma surrounding it, the political dilemma around it remains.[4]

Whose statement about quality should count? How do we find out who that person is? What if there are multiple people whose opinions count? And how much they count relative to one another.

The political dilemma about quality would always involve a series of decisions like mentioned above. These emotional/political decisions about the quality are mostly hidden from the public eye as Jerry aptly points out.

"What makes our task even more difficult is that most of the time these decisions are hidden even from the conscious minds of the persons who make them. That's why one of the most important actions of a quality manager is to bring such decisions into consciousness, if not always into public awareness."[5]

4. *Despite his powerful definition of quality i.e., "Quality is value to some person", Jerry acknowledges that definition of quality is always emotional and political.*
5. *Gerald M. Weinberg. "Quality Software: Volume 1.1: How Software Is Built".*

Quality, Consciousness, and Software Delivery

Connecting what matters

Even though Jerry recommends for the quality manager to bring all decisions about quality into consciousness, it still raises further interesting questions:

- *Who is the quality manager especially in modern software delivery teams (Agile, SCRUM, DevOps so to speak) where the ownership of quality is meant to be shared?*
- *How exactly do we bring decisions about quality into our consciousness?*

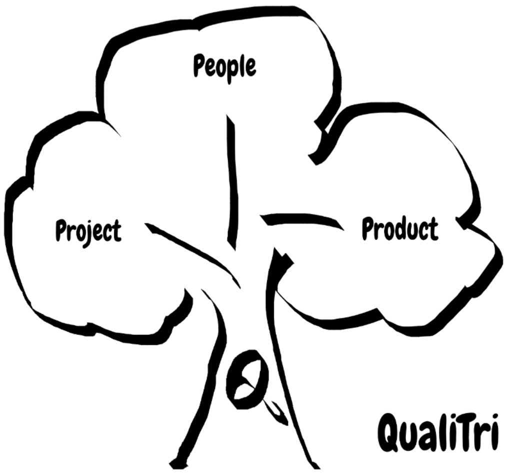
Giving a deeper thought to the questions above, one may realize that focusing on the quality of the product alone is not enough. Because products that we deliver to the customers are not built automatically. There are people who build them by following certain processes and practices. Simply put, the quality that we wish to see in the product can never be the property of that product alone.

Therefore, along with the product, investing in people and project notions of quality is equally important as much as investing in the relationship between the three. We must consider these three notions of quality to make our decisions as conscious as possible.

Why do these notions and the relationship between them matter? It has been explained better with feedback effects, later in this booklet. Let's first understand these three notions in detail.

QualiTri - Three Notions of Quality

Three notions of Quality can be aptly explained by a model named QualiTri.



Source: QualiTri model by Lalitkumar Bhamare

In a nutshell, the Product, People, and Project notions of quality can be described as:

- **Product** - the quality of the product that the team builds together
- **People** - the quality of an individual's work (tester, programmer, or anyone else on the team)
- **Project** - the quality of the project itself

Out of the three above, the project notion of quality is less obvious, and Michael Bolton explains it way better in his words, *"The quality of the project can be observed, assessed, or experienced by things that contribute to great working culture within the team, clarity of roles, depth and development of skills, awareness, availability, and application of appropriate tools and mindset, and, ultimately, with effective and efficient development of the product."*[6]

The Product Notion of Quality

The quality of the product is the most familiar notion of quality that we deal with in day-to-day life as software professionals. However, when we assess the quality of the product, we are generally so more focused on the functional aspects that we hardly get a chance to think about other elements of the product that matter too.

For a wholesome assessment of quality, *broader coverage*[7] of the product during this assessment is important. James Bach, in his Heuristic Test Strategy Model, shows a comprehensive list of *Product Elements* that are worth covering during quality assessment and testing. The checklist is popularly known as SFDIPOT or *San Francisco DIPOT*[8] as a set of heuristic guidewords for the highest levels of a product coverage outline. These elements are:

- **Structure** - Everything that comprises the physical product.
- **Function** - Everything that the product does.
- **Data** - Everything that the product processes.
- **Interfaces** - Every conduit by which the product is accessed or expressed.
- **Platform** - Everything on which the product depends (and that is outside your project).
- **Operations** - How the product will be used.
- **Time** - Any relationship between the product and time.

Similarly, clearly established *Quality Criteria* within the team are equally important to avoid any confusion with making decisions about quality. The criteria at a given time can be some of the various "abilities" of the product that define its quality. For example, *Capability, Reliability, Usability, Charisma, Security, Scalability, Compatibility, Performance, Installability, and Development*. Teams better know at a given time, what aspects of quality are important for the person(s) who matter as they work together on developing the product. Readers are strongly recommended to check out *Quality Criteria* section from Heuristic Test Strategy Model to get the drift. The implementation part of *Product Elements* and *Quality Criteria* shall be discussed when the QCSD framework in explained in detail in this document.

6. An excerpt from *Whole Team Testing for Whole Team Quality - by Lalitkumar Bhamare*
7. *Insightful analysis of what coverage means by Michael Bolton*
8. *See https://www.stickyminds.com/article/how-do-you-spell-testing - James Bach*

The People Notion of Quality

"No matter how it looks at first, it's always a people problem" - Jerry Weinberg

The software that we deliver is built by the people after all, which makes understanding the *people* notion of quality as important as the product notion of it. The people notion of quality stands for the quality of work people in the team do. It is about the way we people collaborate, add value with their skills and the mindset (consciousness) they work with. Typically, when it comes to connecting the people and quality parts in software teams, the responsibility for quality is either given to software testers or programmers are expected to perform testing. Though *quality as a team effort* [9] is not an entirely new idea, the way organizations go about it is still not effective it seems.

To establish a Whole Team Quality culture, organizations usually ask programmers to test, and invest in automating everything. There are several problems with that approach:

1. The key problem is confusing *Testing* with *Quality*. The whole team sharing responsibility for the quality and asking non-testers in the team to perform testing are two different things.
2. Putting most of the efforts to achieve/improve/assess quality focusing only on the product.
3. Caring about quality way later in the process.
4. Reactive strategy i.e., to address quality concerns and risks after they are found in a product (which is too late and costly to fix usually).
5. Over emphasize and reliance mostly on *automated checks*[10] that do not discover hidden risks or find new information. These checks only assert the known information.

9. *Quality is a team effort - Succeeding with Agile - page 323 - Mike Cohen*

10. *See Testing vs Checking- Michael Bolton*

People and The Whole Team Quality

Picture credit: ChurchOfJesusChrist.org

Do you know the ancient parable around the six blind men and the elephant? If not so, then read this wonderful poem by John Saxe:

The Blind Man And The Elephant

It was six men of Indostan, to learning much inclined,
who went to see the elephant (Though all of them were blind),
that each by observation, might satisfy his mind.

The first approached the elephant, and, happening to fall,
against his broad and sturdy side, at once began to bawl:
"God bless me! but the elephant, is nothing but a wall!"

The second feeling of the tusk, cried: "Ho! what have we here,
so very round and smooth and sharp? To me tis mighty clear,
this wonder of an elephant, is very like a spear!"

The third approached the animal, and, happening to take,
the squirming trunk within his hands, "I see," quoth he,
the elephant is very like a snake!"

The fourth reached out his eager hand, and felt about the knee:
"What most this wondrous beast is like, is mighty plain," quoth he;
"Tis clear enough the elephant is very like a tree."

The fifth, who chanced to touch the ear, Said; "E'en the blindest man
can tell what this resembles most; Deny the fact who can,
This marvel of an elephant, is very like a fan!"

The sixth no sooner had begun, about the beast to grope,
than, seizing on the swinging tail, that fell within his scope,
"I see," quothe he, "the elephant is very like a rope!"

And so these men of Indostan, disputed loud and long,
each in his own opinion, exceeding stiff and strong,
Though each was partly in the right, and all were in the wrong!

So, oft in theologic wars, the disputants, I ween,
tread on in utter ignorance, of what each other mean,
and prate about the elephant, not one of them has seen!

Each man in the poem above is describing the truth. His truth. And because his truth comes from personal experience, each insists that he knows what he knows. After combining their individual truth, they come to know the reality of Elephant.

True decisions about quality depend on the correctness of the information they are based on. How we perceive some information to be true depends on lots of factors such as our personal experience, how that knowledge is obtained, our consciousness, and the five senses through which it is evaluated and experienced. And therefore, to develop a true understanding of the quality of the product, we need to create opportunities where people in different roles exchange their information, combine their skills and individual truth to arrive at a common broader conclusion about quality.

If saving time and cost matters, then enabling programmers to do their own work with higher quality consciousness is way more effective than asking them to do skilled testing (which is not their primary job). And same goes with other roles in the software team. The Whole Team Quality approach is at the core of the QCSD framework.

Testing is not the goal, quality is. But testing education for the whole team can certainly lay the necessary foundation. [11]

Lisa Crispin and Janet Gregory have done quite some work on the topic of the whole team approach for quality. Recommend checking it out.

The Project Notion of Quality

Though not visible in plain sight, the project notion of quality plays a critical role in how people in that project deliver a quality product. The quality of the product is ultimately influenced by the quality of the work done by people who build it, which gets affected by the quality of the project they are part of.

Are the team members supported by the management in making decisions about quality? Do they get the funding for tools and training if required for up-skilling? How is the mindset/quality culture of the project? Is the environment of the project supportive and motivated to improve quality? Are people encouraged to collaborate and be quality conscious?

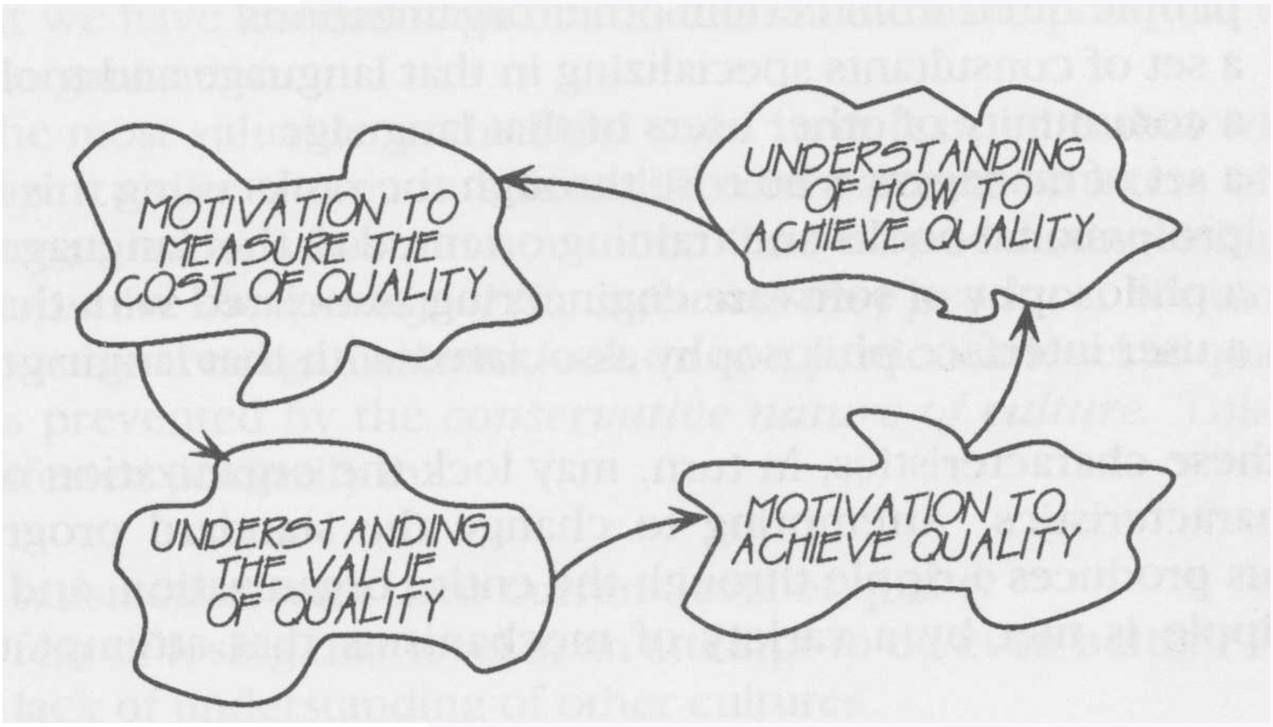
Answers to all the questions above tell us about the quality of the project. The better it is, the more it enables and empowers people to do quality work and ultimately the quality of the product improves.

11. See https://www.stickyminds.com/article/whole-team-testing-whole-team-quality

Though it appears so straightforward, unfortunately, it is not so easy to go about. To deliver quality, a project's *motivation* to invest in quality is essential and this motivation often gets influenced by the quality consciousness and culture of the organization itself. [12]

Organization's understanding of the value of quality is directly proportional to its quality consciousness.

With his popular "diagram of effects" or "feedback effects", Jerry Weinberg explains this vicious cycle that prevents projects from starting to improve quality.


MOTIVATION TO MEASURE THE COST OF QUALITY
UNDERSTANDING OF HOW TO ACHIEVE QUALITY
UNDERSTANDING THE VALUE OF QUALITY
MOTIVATION TO ACHIEVE QUALITY

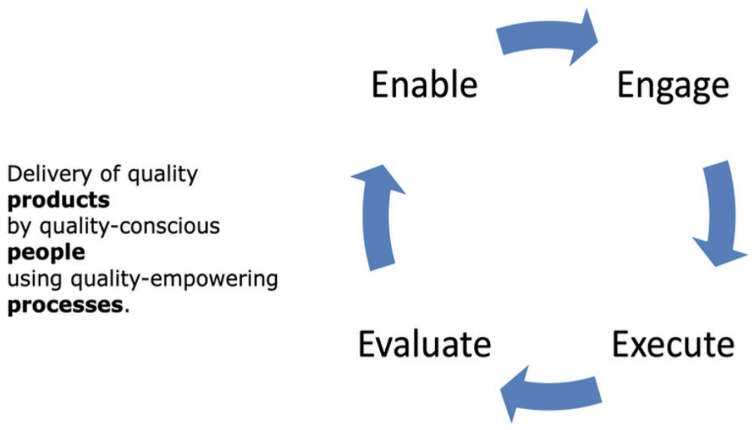
"Quality Software Volume 1.1: How Software Is Built"- Chapter 6 - Feedback effects

If there is no understanding of the value of quality (read - quality consciousness), then there is no motivation to achieve quality, and thus no improvement in the understanding of how to achieve quality. And without knowing how to achieve quality, why would anyone try to measure its value?[13]

Quality is essential because it is about creating and providing value. Our ability to control software quality comes with our ability to control the value of our software efforts. And to control the value of our software efforts (for quality), we need to do everything we do with a higher quality consciousness.

12. *In "Quality Software Volume 1.1: How Software Is Built", Jerry Weinberg discusses software subcultures and how they affect quality management throughout. He discusses the problem with erroneous assumptions about economics of quality, correctness of requirements in greater detail. Readers of this booklet are recommended to give it a read.*
13. *Excerpt From: Gerald M. Weinberg. "Quality Software: Volume 1.1: How Software Is Built"*

Quality Conscious Software Delivery



Enable
Engage
Execute
Evaluate
Delivery of quality **products** by quality-conscious **people** using quality-empowering **processes**.

Combining the three notions of quality that we discussed, consciousness around quality that is required, and software delivery, what we ultimately get is a goal to deliver which is:

> *Delivery of quality products by quality-conscious people, using quality empowering processes.*

That is what the *Quality Conscious Software Delivery (QCSD)* approach is all about. And now we shall discuss how to achieve that goal.

The proven approach of QCSD can be implemented with the cycle of 4E framework i.e.

1. **Enable** - people and project for quality (mindset, skills, culture)
2. **Engage** - for quality through pairing, collaboration, risk storming, continuous testing etc.
3. **Execute** - for quality with clean and testable code, improving testability, skilled testing with effective strategy, automating checks, continuous discovery and continuous feedback through monitoring and alerts etc.
4. **Evaluate** - the evidence gathered through 1-3 to assess quality (what was done, what was found as information, what it tells about the business risk, what needs to change)

and repeat. This 4E framework needs to be applied on the software efforts around Project, People, and Product notions of quality. Let's discuss that in detail.

1. Enable

This is the first and crucial step. It is about enabling people and the project for intellectual culture of quality, enabling the quality mindset, and giving people the required skills to do that well.

Introducing and succeeding with the change in organizations is an art. In my experience, introducing the change in subtle way, by including people and slowly making progress with it usually helps.

Ideas that involve changing mindset and introducing people with something they are not familiar with, are often tricky to deal with. Understanding people and their reasons for doing things the way they do (or not do) is essential aspect of introducing desired change.

Act Early, Act Small

"Act Early, Act Small" is the golden rule from Weinberg's work that can be found useful in many situations.

We discussed that Whole Team Quality culture is at the core of succeeding with QCSD.

To enable people with quality mindset and embrace Whole Team Quality culture, it is useful to understand what blocks them or disables them from accepting it or contributing to it in meaningful way. That is why *understanding their no,* early on becomes a useful point to start with.

Particularly in case of the programmers, the cognitive dissonance[15] they deal with when asked to assess quality of their own work often prevents them from testing their own code effectively. Programmers can strive to write the code without obvious bugs in it but that is not sufficient to uncover all hidden risks. Expecting programmers to test their own code like professional testers, in the name of Whole Team Quality, does not seem to be an effective approach at large. [16]

The emerging trend of relying only on programmers to test their code, is deeply rooted in mistaken notion of testing in our industry and the damage it has caused to the craft of testing[17] but that is beyond the scope of this booklet.

Instead, the pairing sessions between programmers, testers, and other members of the team to get familiar with skilled testing and using that knowledge to do their individual work with more quality consciousness, is highly recommended.

Under QCSD framework, below are some of the session-types teams can pair on, that are found to be immensely effective in projects the approach was followed.

15. *See https://en.wikipedia.org/wiki/Cognitive_dissonance*
16. *See Mind, Matter, Testing and The Cargo Cult – by Lalitkumar Bhamare*
17. *Read We need to talk about testing by Dan North, the creator of BDD.*

Quality Experience Session

The common problem observed with many engineering teams is that they often assume/expect product requirements/acceptance criteria coming from the product owners to be true and complete. In fact, there seems to be an unwritten rule that says, “product teams will care for product quality” and “engineering teams will look after engineering quality”.

A 100% code coverage, 100% up time, or zero defects in production metrics are useless if you do not provide value that customers want. Therefore, when we think of the requirements for engineering implementation, it is important to have them understood in terms of value they will create for users. Product requirements often try to address that but engineering requirements derived from those cannot always establish that connection.

There is an interesting story about Mike Jones and uSwitch in the book Leading Quality[18]. Mike made his engineering teams prioritize every build, feature, release, and issue against the question, "What will make more users switch?". This change in engineering teams’ mindset helped their business grow from $1.3 million to a $200 million in five years.

Now imagine what would happen if ***design thinking*** met ***system thinking***?

A regular and close exchange, typically during the design phase, between product design team and engineering team (the test engineers on the team to be precise) has tremendous potential to create a better user experience with enhanced product quality.

This exchange between Testers/Quality Advocates and User Experience professionals can co-create Quality Experience for the stakeholders.

QA + UX = Quality Experience

Sure, the design team can get some product ideas based on user tests and interviews they perform, but there is a lot more that matters when it comes to building software that a large customer base would like to use. The challenges faced by testers and UX professionals are usually similar in nature when it comes to ensuring better quality and better user experience.

If a UX designer has a solution for a product problem, a skilled tester with quality insights, product knowledge, and awareness around cross-functional dependencies can point out a variety of ways in which the solution may fail, detecting risks earlier, helping to avoid lots of unnecessary research and rework. Testers can also borrow realistic information from UX's research that can help them design tests that matter and avoid straying into unwanted territory. Testers can get statistical data or interaction-based information from the UX team that can shape a better scope for their tests.

James Christie, long-time test consultant with experience in IT/Security audits builds a compelling case for such exchange between testers and designers when he writes about resilience engineering. He says,

“Testers should be ready to explore and try to explain the holes, the gap between the designers’ limited knowledge and the reality that the users will deal with. We have to try to think about what the system as found will be like. We must not restrict ourselves to the system as imagined.”

---

18. Leading Quality a book by Ronald Cummings-John & Owais Peer.

19. Resilience Requires People – James Christie

Another highly recommended resource to make such pairing more useful, is Explore It! book by Elisabeth Hendrickson. The *Never and Always*[20] tips Elisabeth shares in her book can also be used for effective QX pairing and discussion.

A rather elaborative article has been written on how exactly this exchange can be done in systematic way. Give it a read.

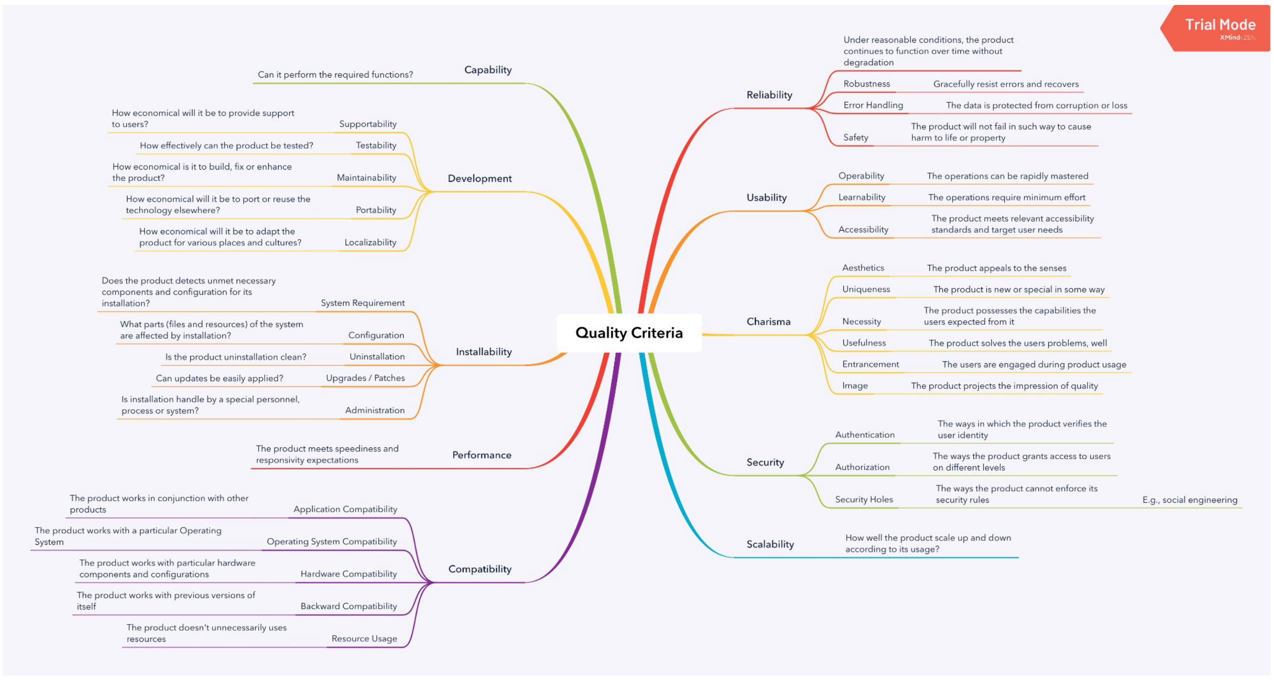
Quality Criteria Session

Processes if used property can act as a catalyst for enabling quality consciousness.

If you are working in a typical SCRUM setup, then consider starting with this session. The team would perform this session at the time of sprint planning or discussing the sprint goal and epics. Or you could do it each quarter (so to speak) when teams define their OKRs, KPIs or milestones to achieve in near future.

Ideally, everyone should participate, including the product owner and other relevant stakeholders.

A recommendation would be to creating a *risk checklist* for your team context based on the quality criteria heuristic from the Heuristic Test Strategy Model. For each item below in the mind map determine if it is important to your project for given sprint goals or milestones you want to achieve, then think about how you would recognize if the product worked well or poorly in that regard.


Trial Mode
XMind·ZEN
Quality Criteria
Capability
Can it perform the required functions?
Development
Supportability
How economical will it be to provide support to users?
Testability
How effectively can the product be tested?
Maintainability
How economical is it to build, fix or enhance the product?
Portability
How economical will it be to port or reuse the technology elsewhere?
Localizability
How economical will it be to adapt the product for various places and cultures?
Installability
System Requirement
Does the product detects unmet necessary components and configuration for its installation?
Configuration
What parts (files and resources) of the system are affected by installation?
Uninstallation
Is the product uninstallation clean?
Upgrades / Patches
Can updates be easily applied?
Administration
Is installation handle by a special personnel, process or system?
Performance
The product meets speediness and responsivity expectations
Compatibility
Application Compatibility
The product works in conjunction with other products
Operating System Compatibility
The product works with a particular Operating System
Hardware Compatibility
The product works with particular hardware components and configurations
Backward Compatibility
The product works with previous versions of itself
Resource Usage
The product doesn't unnecessarily uses resources
Reliability
Under reasonable conditions, the product continues to function over time without degradation
Robustness
Gracefully resist errors and recovers
Error Handling
The data is protected from corruption or loss
Safety
The product will not fail in such way to cause harm to life or property
Usability
Operability
The operations can be rapidly mastered
Learnability
The operations require minimum effort
Accessibility
The product meets relevant accessibility standards and target user needs
Charisma
Aesthetics
The product appeals to the senses
Uniqueness
The product is new or special in some way
Necessity
The product possesses the capabilities the users expected from it
Usefulness
The product solves the users problems, well
Entrancement
The users are engaged during product usage
Image
The product projects the impression of quality
Security
Authentication
The ways in which the product verifies the user identity
Authorization
The ways the product grants access to users on different levels
Security Holes
The ways the product cannot enforce its security rules
E.g., social engineering
Scalability
How well the product scale up and down according to its usage?

Mind map created by João Farias based on Heuristic Test Strategy Model

As Bach/Bolton point out, *by thinking about different kinds of criteria, you will be better able to plan tests/ activities that discover important problems fast. Each of the items on your list can be thought of as a potential risk area. Do not get overwhelmed by the volume of the ideas; not everything will be applicable for every sprint. More important is that the team has a discussion and makes it part of the process.*

20. *See https://pragprog.com/titles/ehxta/explore-it- Evaluate Results - by Elisabeth Hendrickson*

When it becomes part of the team culture to regularly assess your sprint goals or user stories for risks and quality criteria, it will become part of an individual's habit to think of those factors consciously, putting further efforts toward achieving them. By involving everyone in this pairing activity, you are making sure that you address important problems before it is too late.

Pairing for Testability Session

The importance of good testability and the role it plays in enabling teams to identify risks earlier and faster, cannot be emphasized enough. In their book on testability, Ash Winter and Rob Meaney make a remark that is closely related to the idea of Quality Consciousness.

> *A testable system provides information about its own limits, giving your business and technical stakeholders the ability to make crucial decisions at peak times when reputations can be won or lost.*[21]

One of the key characteristics of a better testable software is its un-bugginess. It means the software is free from obvious bugs so that testing efforts can find hidden risks faster. It naturally warrants a programmer to be more conscious of the quality of their work when they develop the code and ensure it has no obvious bugs before integrating it with the bigger application code.

Our ability to observe the software behavior more transparently (observability) and the ability to control software to perform experiments to find hidden risks (controllability) are equally important to become more conscious of the factors that affect quality. *Enabling project teams to build better-testable systems is one sure way to unlock the faster discovery and faster feedback about hidden risks.* Therefore, Pairing for Testability sessions in the team on a regular basis, is strongly recommend.

Testers and programmers can pair to improve testability in several ways, such as:

- Creating an alert system for error logs from production
- Creating solutions to easily create test data
- Improving error-logging and handling for the product
- Creating solutions for better observability
- Improving the controllability to make deep explorations easier and possible
- Writing meaningful logs, especially error logs that make debugging easier
- Writing code with dedicated tags/ids so that element identification for automation becomes less cumbersome.

Testers can perform the role of advocate here while also learning from programmers about technical details and useful practices.

Teams can create user stories or technical debt tasks and do these activities at regular intervals. It will greatly benefit to make *intrinsic testability*[22] as part of your checklist for this pairing session and to use it for all major tickets, the earlier the better.

21. *Team Guide to Software Testability- Ash Winter & Rob Meaney*

22. See https://www.satisfice.com/download/heuristics-of-software-testability

2. Engage

Engage step is about doing a consciousness exchange. About making conscious efforts and staying engaged in doing activities that ensure a strong focus on quality. Interestingly, it is not about doing extra activities but about doing regular activities in a collaborative and quality-conscious manner.

Collaboration is the key here. If we are to think of the story of blind men again, this is the step where active exchange of individual information, skills, and perspectives happens. And that's where people in the team influence each other with feedback and ideas which they can take back and do their work with more consciousness.

Product Coverage session or Requirement Engineering session is one powerful way to go about it.

Product Coverage Session/Requirement Engineering session

> *Requirements are not an end in themselves, but a means to an end—the end of providing value to some person(s)[23]. - Jerry Weinberg*

Expecting requirements to be clear, complete, and relying on those works better in the manufacturing industry with assembly lines. But building software for human use, based on defined product requirements is often not enough. We must think critically of those requirements and derive the "true requirements" for engineering implementation. And that warrants us to have methods to do it well.

In QCSD framework, it is recommended to do Product Coverage Sessions or Requirement Engineering Sessions on regular basis. It can be done on an epic level or a complex feature requests/user stories.

Experienced/dedicated testers in the team can analyze the epic or feature story using SFDIPOT (i.e., Product Elements) checklist we discussed earlier and come up with test ideas, questions about risks, missing information, dependencies not considered, risks identified and so on.

A guided discussion on this SFDIPOT analysis of user story/epic during grooming meetings or refinement meetings or planning meetings (if this discussion happens that is) can help the team identify hidden risks, determine the completeness of the requirement, create a plan for development, identify gaps and dependencies, estimating with more information at hand, and most importantly to avoid doing rework by accidentally finding things halfway through the programming is started.

If we are to save costs and time, and still deliver quality software then, it is always cheaper to do things right the first time.

The approach we discussed above helps teams do exactly that.

23. *Gerald M. Weinberg. "Quality Software: Volume 1.1: How Software Is Built".*

The Power of Pairing

There is one more way to go about Product Coverage session which is through programmer and testers pairing for the test design.

For this purpose, start with analyzing the impact of a given change on your product, parts of the product, downstream, or other interfaces your product must interact with.

Pairing with the programmer will better help the tester to identify these areas of impact from technical aspects, and the programmer will benefit from the tester's input to understand functional dependencies, end-to-end business flows, blockers of any kind, or even test data and environment dependencies. Of course, the edge cases and other tricky situations are added benefits.

In the absence of a clear idea, this is easier said than done. If you need help getting started, consider referring to the same product elements checklist as a basis for this pairing session. Having a quick discussion about all those elements of your product should help everyone think of aspects for better all-round test coverage.

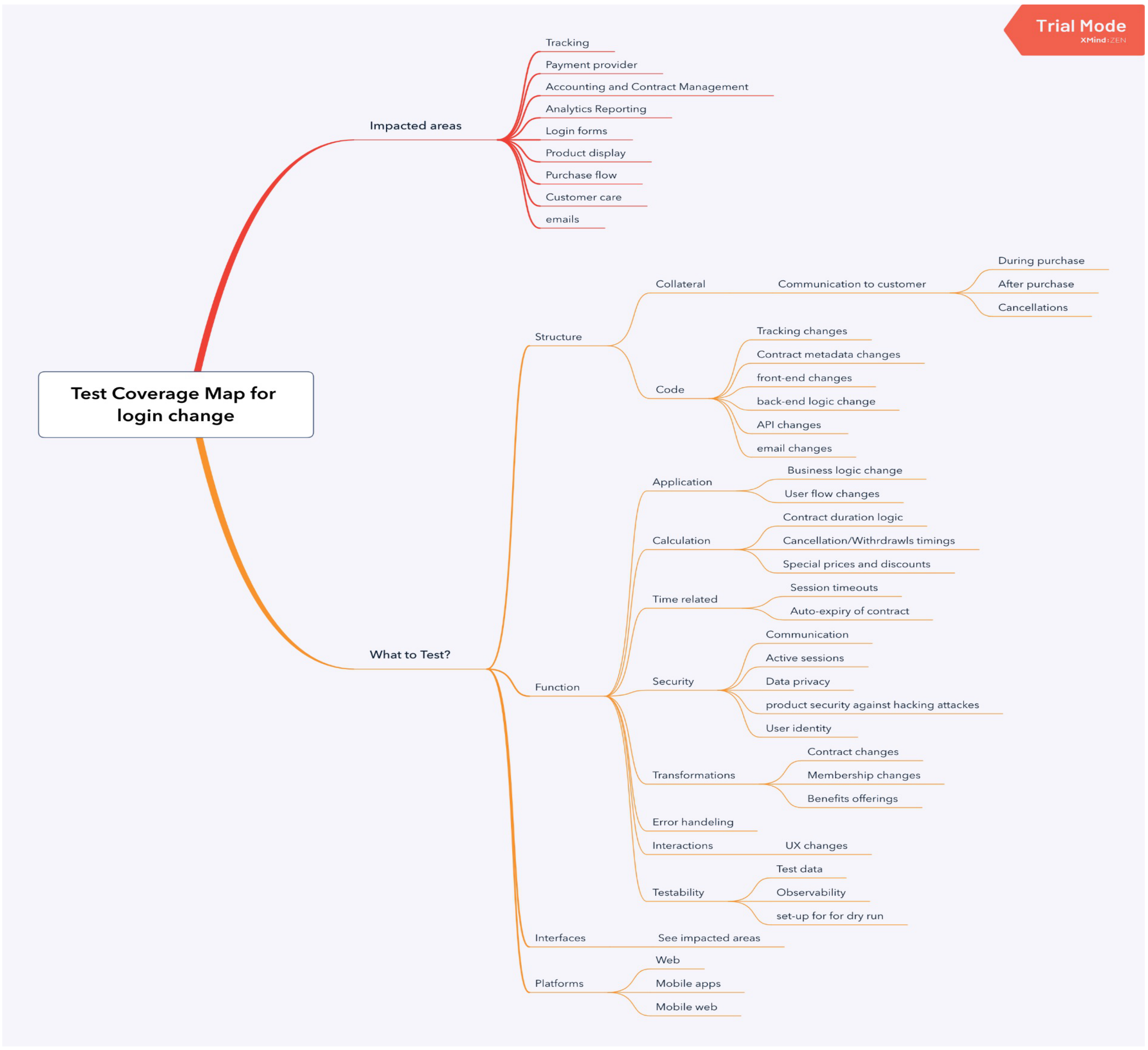
Testers can also create a risk list that is applicable for each product or project and pair with programmers for developing it further. Whether to document these ideas in pull requests, user stories, or some other way is up to what your team feels more comfortable with. It is less about the format or process and more about the value it is creating.

This session would usually be done before coding starts on high-complexity tickets by pairing with a programmer, or after the coding has been done to review and refine the test coverage map that the tester has already created.

For example, let's say that the user story warrants you to change how the purchase of your product is being made, while that change depends on the payment solution being worked on by other teams. This change is complex and beyond just changing the business flow—it involves changes on many levels and interfaces.

There are also business-analytics reasons for tracking this change. It is unlikely for an individual to come up with a comprehensive test strategy in a short time, so this would be a good situation for a Pairing for Coverage session about the requirements.

A high-level outcome for the above scenario could look something like this (link to the high-resolution picture here)



Suggestion would be to start practicing on larger, more complex tickets first, so that the pairing experience is not overwhelming for non-testers. Testers can use this pairing opportunity to onboard others with given heuristics and prepare them to have a checklist in mind while they work through their tasks. Once they get familiar with these lists, they will find it easier to apply them to smaller items.

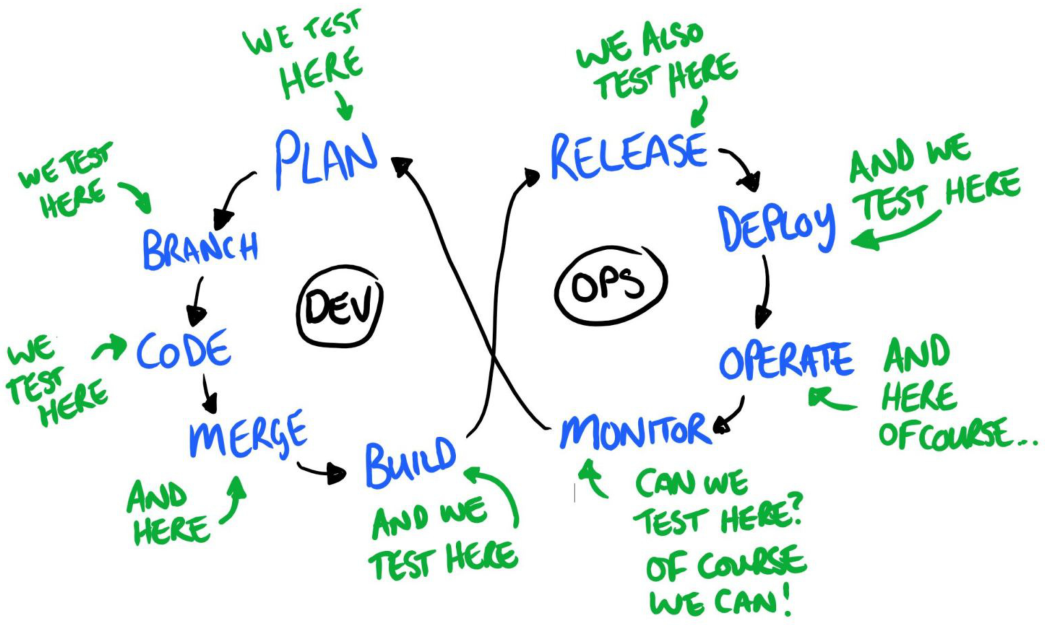
This pairing can also be extended to identifying what to automate and creating process flows and agreements around ownership of automation efforts. It would be great if testers could write automation and make sure that other members in the team are familiar with their checks so that others can take over to add new tests or fix broken ones if needed. Writing an automated check before deploying the change to production can also be an effective way to encourage sharing of responsibility for automation.

Moreover, there are many different ways a pairing exercise can be explored for good. Check out this interesting work done around mob-testing by Lisi Hocke.

3. Execute

This step is about putting all the learnings from previous two steps into action and keeping at it till you are "done" shipping it to the end user, at the same time simultaneously exploring for information about risk all phases from ideation to delivery and even after.

Dan Ashby brilliantly explains the similar idea through his Continuous Testing in DevOps model.[24]



24. *Dan Ashby's version of Continuous Testing in DevOps beautifully compliments the core idea of QCSD.*

In a nutshell, Execute phase is about "continuous discovery and continuous feedback" for information about risk. And to ensure we consistently provide quality that people value and do not produce what they don't value.

Which means:

- Programmers to write their code by keeping all information about risks and quality in mind, making sure there are no obvious bugs, integrating early and often i.e., continuous integration
- Testers to plan, design and execute tests keeping the information about risk in mind, at all phases in applicable manner i.e., continuous testing.
- Writing automated checks at various levels to cover known risks and exploring further to decide what could be automated more (if needed)
- Investing in and investigating production logs and alerts to understand/ identify risks and user behaviors. Using that information to design more tests or re-thinking the original feature/idea
- Aligning all engineering actions, at all phases of SDLC keeping the quality aspects in mind and doing so consciously.

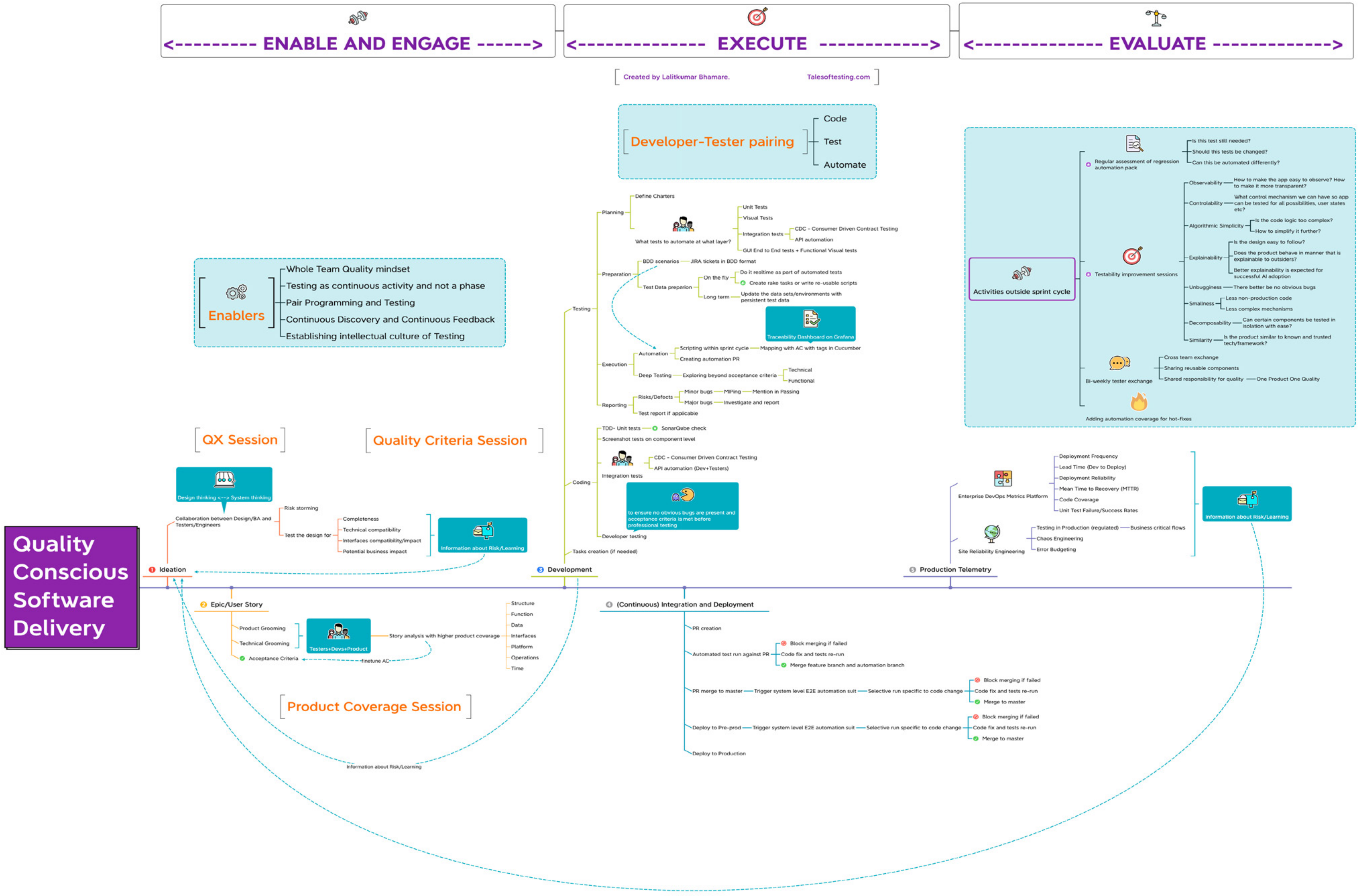
4. Evaluate

This is the last but important step in QCSD framework. All the work done from Enable to Execute, can help teams gather information which can serve as evidence for assessing the quality[25] that was delivered throughout. *Teams can use engineering dashboards and metrics that are carefully designed to gather data points about quality, which can serve as evidence for assessment.*

Evaluate step is about critically assessing what was done, how it was done, what could have been done differently, what was missed, how things helped uncover/address busines risk and what must change to yield better results in future.

For a typical DevOps set-up, the QCSD in action can be summarized in diagram shown on next page. You can access the high resolution version from here. Please note that the tools and technologies mentioned are not endorsements but purely for representational purpose. Teams can always choose what works best in their context.

25. *Quality can't really be measured or quantified, it can only be assessed. See Assess Quality, Don't Measure It by James Bach*

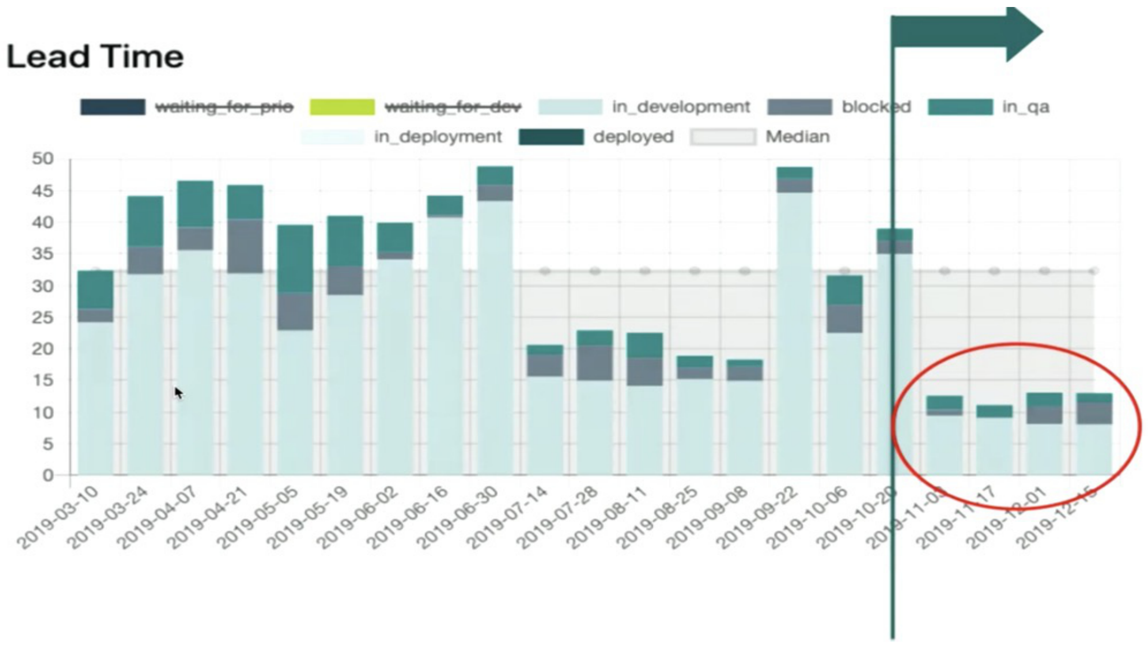


<---------- ENABLE AND ENGAGE ------>
<--------------- EXECUTE ------------>
<--------------- EVALUATE -------------->
Created by Lalitkumar Bhamare.
Talesoftesting.com
Developer-Tester pairing
Code
Test
Automate
Enablers
Whole Team Quality mindset
Testing as continuous activity and not a phase
Pair Programming and Testing
Continuous Discovery and Continuous Feedback
Establishing intellectual culture of Testing
Testing
Planning
Define Charters
What tests to automate at what layer?
Unit Tests
Visual Tests
Integration tests
CDC - Consumer Driven Contract Testing
API automation
GUI End to End tests + Functional Visual tests
Preparation
BDD scenarios
JIRA tickets in BDD format
Test Data preperion
On the fly
Do it realtime as part of automated tests
Create rake tasks or write re-usable scripts
Long term
Update the data sets/environments with persistent test data
Traceability Dashboard on Grafana
Execution
Automation
Scripting within sprint cycle
Mapping with AC with tags in Cucumber
Creating automation PR
Deep Testing
Exploring beyond acceptance criteria
Technical
Functional
Reporting
Risks/Defects
Minor bugs
MIPing
Mention in Passing
Major bugs
Investigate and report
Test report if applicable
Coding
TDD- Unit tests
SonarQube check
Screenshot tests on component level
Integration tests
CDC - Consumer Driven Contract Testing
API automation (Dev+Testers)
to ensure no obvious bugs are present and acceptance criteria is met before professional testing
Developer testing
Tasks creation (if needed)
Activities outside sprint cycle
Regular assessment of regression automation pack
Is this test still needed?
Should this tests be changed?
Can this be automated differently?
Testability improvement sessions
Observability
How to make the app easy to observe? How to make it more transparent?
Controlability
What control mechanism we can have so app can be tested for all possibilities, user states etc?
Algorithmic Simplicity
Is the code logic too complex?
How to simplify it further?
Explainability
Is the design easy to follow?
Does the product behave in manner that is explainable to outsiders?
Better explainability is expected for successful AI adoption
Unbugginess
There better be no obvious bugs
Smallness
Less non-production code
Less complex mechanisms
Decomposability
Can certain components be tested in isolation with ease?
Similarity
Is the product similar to known and trusted tech/framework?
Bi-weekly tester exchange
Cross team exchange
Sharing reusable components
Shared responsibility for quality
One Product One Quality
Adding automation coverage for hot-fixes
QX Session
Quality Criteria Session
Design thinking <--> System thinking
Collaboration between Design/BA and Testers/Engineers
Risk storming
Test the design for
Completeness
Technical compatibility
Interfaces compatibility/impact
Potential business impact
Information about Risk/Learning
Quality Conscious Software Delivery
1 Ideation
2 Epic/User Story
Product Grooming
Technical Grooming
Acceptance Criteria
Testers+Devs+Product
Story analysis with higher product coverage
finetune AC
Structure
Function
Data
Interfaces
Platform
Operations
Time
Product Coverage Session
Information about Risk/Learning
3 Development
4 (Continuous) Integration and Deployment
PR creation
Automated test run against PR
Block merging if failed
Code fix and tests re-run
Merge feature branch and automation branch
PR merge to master
Trigger system level E2E automation suit
Selective run specific to code change
Block merging if failed
Code fix and tests re-run
Merge to master
Deploy to Pre-prod
Trigger system level E2E automation suit
Selective run specific to code change
Block merging if failed
Code fix and tests re-run
Merge to master
Deploy to Production
5 Production Telemetry
Enterprise DevOps Metrics Platform
Deployment Frequency
Lead Time (Dev to Deploy)
Deployment Reliability
Mean Time to Recovery (MTTR)
Code Coverage
Unit Test Failure/Success Rates
Site Reliability Engineering
Testing in Production (regulated)
Business critical flows
Chaos Engineering
Error Budgeting
Information about Risk/Learning

It is also a great idea to gather more information about delivered quality through user reviews and feedback, analyzing the change in user behavior through monitoring and analytics dashboards.

To evaluate the speed of delivery, teams can measure and compare the "Lead Time" before and after experimenting with the framework.

Below is such comparison done for one of the project teams that experimented with QCSD.



The Lead Time *after QCSD* period is highlighted with the eclipse in red. It was the first sprint in a long time, where we as a team finished all the tickets and pulled more, the so-called testing bottleneck was minimal, and the bugs reported that would make into the backlog or warrant some critical rework post-production were negligible.

Conclusion

Teams change, and business contexts change too which affect the overall delivery and end quality of the products we ship. Implementing QCSD helps you identify the patterns that work best in your context. Figure out what patterns and practices help you with the speed-quality-cost you need and make sure to keep working within those patterns consistently. Consistency is the key and Quality-consciousness is the path.

You have the key; you know the path. Now all you need to do is start walking and unlock the secret door of Speed-Cost-Quality.

“The best time to start was yesterday. The next best time is now.”[26]

26. - unknown

EuroSTAR Huddle

EuroSTAR Huddle

Europe's Largest Selection of Software Testing Content.

1,250+ Blogs | 100+ eBooks | 200+ Webinars

The EuroSTAR team invite leading testing experts to share their knowledge with the community on Huddle. When you join Huddle you can access an unrivalled selection of resources across all the latest topics in software testing.

Expand your testing knowledge and join us for regular live webinars from prominent speakers and top contributors to the world of testing. Ask for help in the Huddle Forum and avail of our Huddle blog for the latest articles and trending topics in testing. Being part of EuroSTAR Huddle is an investment in your ongoing professional development and will give you added skills to help you achieve the very best in your career.

www.eurostarhuddle.com

Join The Community

Our Events

When you have enjoyed the online resources on Huddle, the training continues at our annual software testing events.

Experience the welcoming EuroSTAR Community. Be inspired by exceptional speakers sharing real life testing experiences. Try out the latest tools in the Expo and take advantage of the nonstop networking to connect with leading testing experts and upcoming innovators all in one place.

EuroSTAR Software Testing & Quality Conference

Celebrating 30 years of the EuroSTAR Community in 2022 - the longest running and largest software testing conference in Europe welcomes over 1,000 delegates every year.

Visit EuroSTAR Website

EuroSTAR Huddle on Tour

We also host a number of smaller conferences and events in different locations and bring the EuroSTAR Huddle live experience to partner events around Europe.