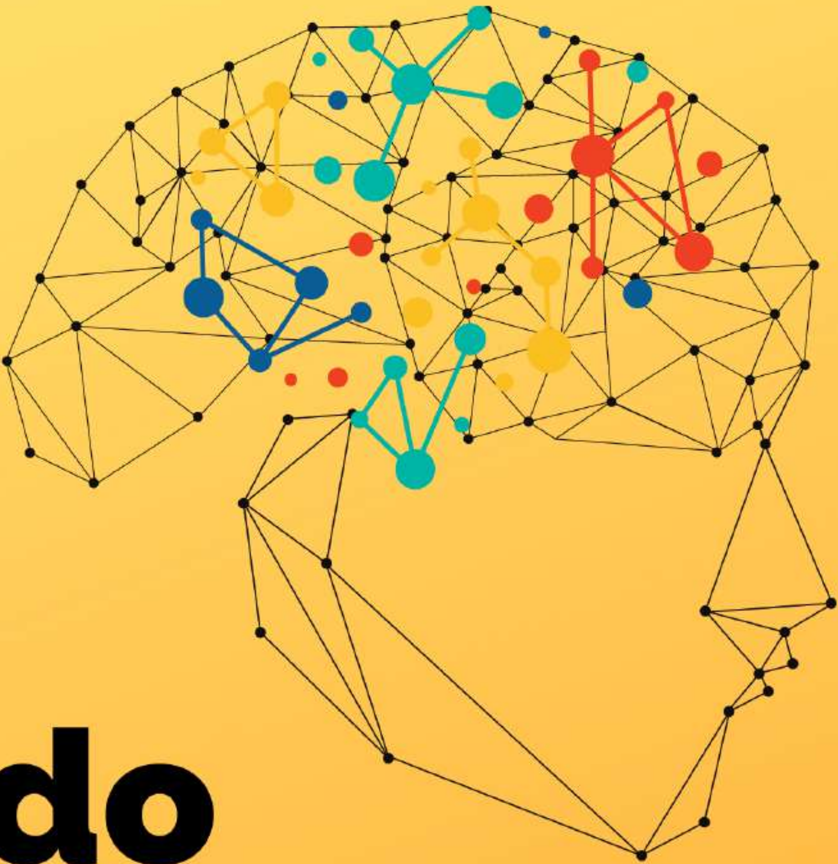
Unshackle capacity.Test deeply. Test rapidly.

A crisp guide for engineering/QA managers to balance cost, quality, time harmoniously.



do SmartQA.

The HyBIST Approach

THIRUVENGADAM ASHOK

TABLE OF CONTENTS

Preface

This crisp book takes a hard look at challenges and problems senior engineering managers face in QA and testing and presents a refreshing and sometimes contrarian approach to solutions. It examines the practice(s), mindset(s), and techniques followed and presents interesting ideas based on problem-solving, culture and mindset, philosophy, and thinking styles, in addition to a scientific approach to test engineering.

THE CURRENT CONTEXT

In these times where systems are complex and machines intelligent, businesses impatient and timelines shortened, customers demanding and work unending, technology expansive and engineering capacity limited, it is only prudent to be smarter to do less and accomplish more.
Software development has significantly evolved and matured, but the QA practice seems archaic, relying solely on automated testing without embracing a comprehensive approach.

The modern development approach is rapid leveraging code via frameworks, components whereas testing seems to be stuck in applying basic principles or relying heavily on experience. The focus is skewed towards validation via automated tests, focussed less on intellectual practice to digging in and questioning, to preempt or detect early, to go beyond bug finding towards suggesting, ideating and delivering higher value.

STATE OF TESTING PRACTICE

Quality seems to be caught up between do-more/test-continuously on one side and finding issues earlier/assure on the other side, without a clue on how to marry them in simple terms. There seems to be extreme reliance on rote continuous testing via automation rather than smart assurance exploiting human intelligence.

Some common observations that stunt the practice are:

- rely on experience for effective testing
- most often the approach is black box
- more of check not as much test (compliance vs bug-seeking)
- unclear partitioning/objectives of dev & QA(system) test
- emphasis on scripted test cases
- template-driven test case design limiting effectiveness
- focus on code validation, not much as questioning/exploring
- effectiveness(speed) usurps effectiveness(test quality)

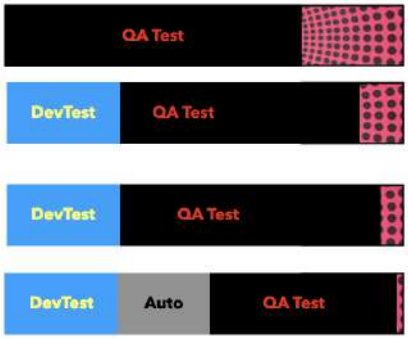
INDUSTRY CHALLENGES IN QA

P1-Grappling with too much testing?

"We are doing a lot of testing, consuming significant effort/time. Yet, we have customer issues that weigh us down. In today's rapid dev model, automation seems to be in catch up mode, another backlog to handle now" a Senior Engineering Director of a global product company said. How can we do better? Is automation the only way forward? What can we do to test smarter?

I have seen QA doing work that Dev Test should have covered, writing detailed test documentation and relying on scripted tests, doing too much regression creating a large automation backlog due to poor test organisation, as key reasons that saps capacity and effectiveness of QA.

Test automation is certainly a way forward, but after enhancing effectiveness via SmartQA. Sharpen QA focus, strengthen scenarios, don't do others work, and then automate is what SmartQA is.

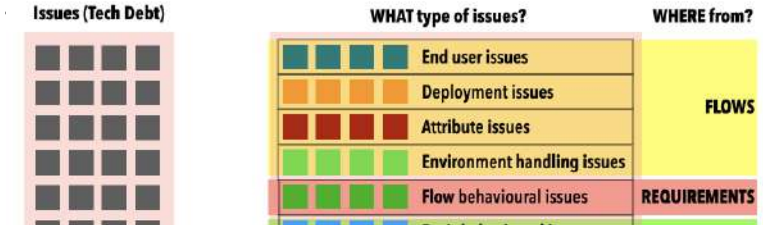


P2-Choked by bugs?

"Despite all the testing we do, field issues do not seem to abate. Sometimes it is a few serious issues that cause us to react intensely, sometimes it is a bunch of simple issues that make us consume bandwidth. Clearly the backlog is building up, with debts to be serviced, straining capacity to deliver new ideas."

This is what I hear from senior engineering managers of product companies. How do you go about fixing this? Well, I have seen a flurry of activity to identify root cause(s) and address them. They help to set focus, but fizzle out.

Analysing '*quality of issues*' to understand types of issues that leak enables practical actions, rather than jumping into the 'reason of why' (root cause). Smart QA it is, to do failure analytics differently, to 'tighten the purse'.



Bugs are indeed a serious drain on engineering capacity, forcing one to fix issues at the expense of building revenue yielding new features. Smart failure analytics visualises problems well, enabling clear actions to strengthen practice and reduce debt significantly.

P3-Challenged by test adequacy?

"Our product is complex and used in myriad interesting ways in different environments. We seem to be discovering a variety of issues continually on the field. Wonder if our tests and test cases are adequate? We do have a lot of test cases, but are we effective?" asked an Engineering Director of a global product company.

How can I tighten the noose? How can I enhance the filter?

Most often I have seen as practice, that test scenarios/cases are designed solely based on one's experience. This, though valuable, poses a challenge - "*How do I logically conclude that it is sufficient, adequate?*"

Well, one cannot surmise that all scenarios can indeed be thought of a-priori; as during the act of execution, we do discover potentially interesting failure cases. The intent to question completeness however is very useful, as it allows one to question deeply, which really is what brilliant testing is all about.

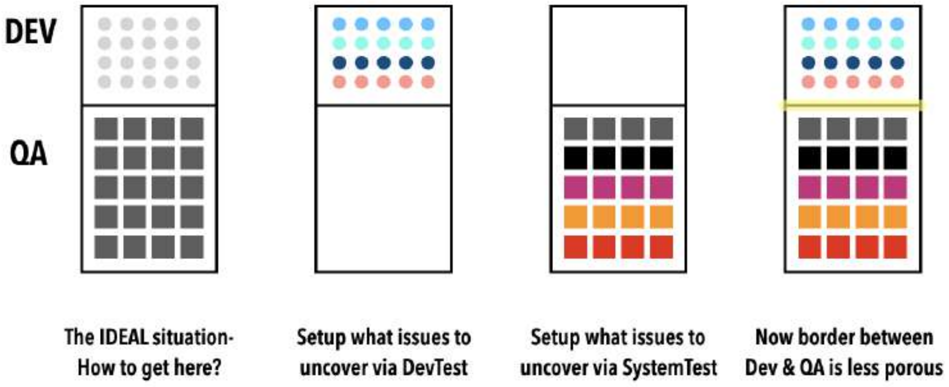
A behaviour driven approach to test design ensures a mindset to extract conditions from requirements, understand perturbations from other parts of the system leading to 'robust test design'. After all, a good filter tightens the noose!

P4-Troubled by development test?

"*Yes, we know that doing early dev test is superior. Despite requisite focus, we don't seem to be effective. QA finds issues that can be found earlier, wasting bandwidth, missing out issues in real life user flows.*"

DevTest is strengthened if specific types of issues were targeted, and these could be detected via DevTest, code review or smart checklists.

A typical problem that I have seen is the lack of clear partitioning of what issues to focus on, in DevTest and SystemTest. The result - a porous gate between early and late stage testing, resulting in high internal defect leakage, strangling effectiveness of QA. Automation of DevTest is powerful, potent if DevTest cases are sharp and focussed.



It is paramount to ensure that the approach does not create more work for developers or upset the development rhythm. Smart Checklist is a brilliant tool for this. (The book “The Checklist Manifesto “by Atul Gawande is an illuminating read).

We have seen remarkable improvement in overall quality via “shift-lifting”, reducing internal leakage by well over 50%. In today’s age of rapid development and frequent releases, strengthening early DevTest has a multi-fold effect on product QA.

P5-Weighed down by automation?

"*As we embrace faster release cycles, testing has become a bottleneck. Yes, we have embraced automation as the way forward. We have a huge regression suite and therefore a big backlog for automation, a tough balance to speed up and yet maintain the fast paced release rhythm. What can I do?*"

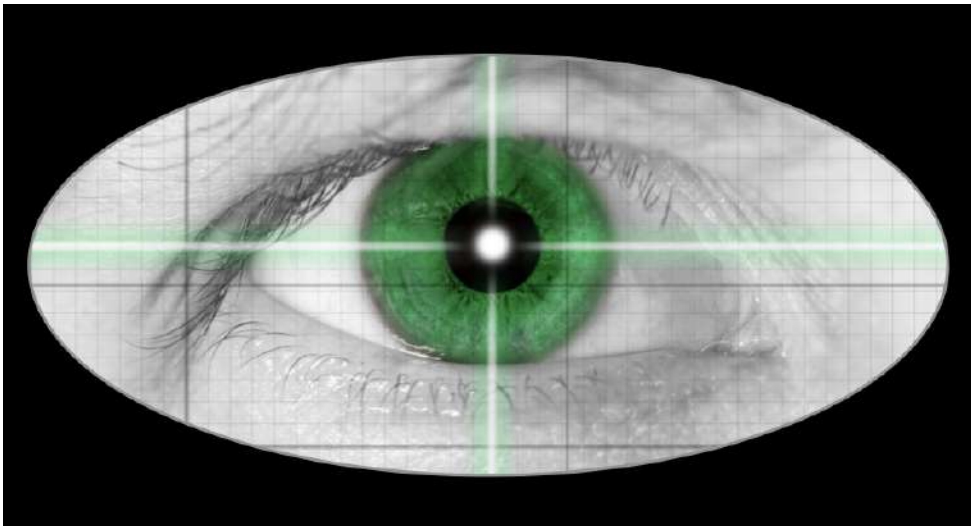
Automated tests are great to monitor a system's health. Rather than just use regression as the candidate for automation, key flows that signify the pulse of a system's health are superior, don't you think? And, this won't create a huge backlog for automation, right?

Most often I have seen automation embraced as the solution to speed up testing. Conceptually correct it is, the problem is - what makes it worth the while to automate? Automated tests have to be in sync with the product and are therefore not a one time effort.

Choosing the right ones implies, it needs to be at the level of user flow, and be a clear indicator of health. Unless test scenarios are well structured and organised, choosing the right ones will turn out to be difficult, and ultimately weigh you down. It then becomes a pursuit of catching up with automation rather than making it work for you.

The goal is not 100% automation, it really is no leakage of defects. Automated tests are really 'checks' that assess key paths for good health (correctness) while intelligent human tests are focused on finding issues(robustness). A harmonious balance between these two enables clean code to be delivered without being weighed down by automation.

P6-Are your quality metrics insightful?



"We track a lot of metrics related to progress of development and quality every

sprint, like backlogs, technical debt, velocity, task status etc. What is not very evident is the 'quality of movement' i.e. how well done, so that we create less debt as we move." How can I get a better insight of the quality of tests done and a more objective measure of product quality?

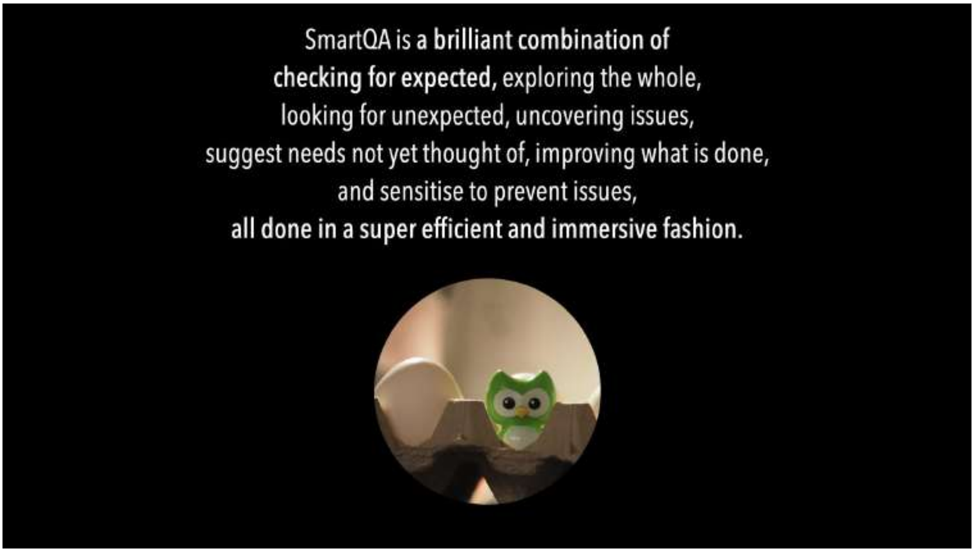
Extrinsic metrics are easier to measure and give visibility of direction, progress, speed and external feel of product quality. Intrinsic metrics are deeper, harder to measure but can give greater insight into the quality of work. Measuring this requires a good structure and organisation of test artefacts. The benefit - a greater insight into effectiveness of outcome and therefore lower technical debt & greater acceleration, don't you think?

Metrics can be classified as measuring work progress, work quality, product quality and practice quality. Except for the first one on work progress where we have a lot of measures facilitated by project and test management tools, the others depend on test organisation and clarity of types of issues to uncover. 'Quality Levels' based on HBT (Hypothesis Based Testing) provides a strong foundation for these, enabling you to assess potential test effectiveness, judge product quality objectively and fine tune practice quality .

INTRODUCING SMARTQA

Testing is not about merely checking for compliance, nor is it just about finding issues with delivered code; it is to be curious about what may be intended by the producer, what may be expected by consumers, the correctness and incorrectness of what is present, the identification of what is probably needed but missing, and the plain inquisitiveness resulting in interesting questions that enable deepening the understanding, ultimately delivering value in a larger business context by this act. It is no longer just a physical act of validation of code, but a mental analysis and corroboration of early-stage spec, design, and code so that one may preempt issues rather than detect them later or painfully miss them.

SmartQA is about "doing less and accomplishing more".



SmartQA is a **brilliant combination of checking for expected,** exploring the whole, looking for unexpected, uncovering issues, suggest needs not yet thought of, improving what is done, and sensitise to prevent issues, **all done in a super efficient and immersive fashion.**

SMARTQA SUGGESTIONS

(Suggestions to - do less, do better, do faster and don't do.)

S1-Focus on direction first, then on speed

S2-Focus on practice & process, then tools

Shrinking timelines is a challenge to testing in today's rapid development cycles, and the typical approach to speeding them up is to automate them. It helps to do more in less time, but it is to regress rather than uncover newer issues and requires effort to build and keep in sync.

Is speeding up via automation the only approach to doing more? No.
Is it about doing more or less in a given amount of time? Both, right? The former is about being efficient, while the latter is about effectiveness. Efficiency is about speeding up the act of doing something using technology or automation, and being effective is about improving outcomes by enhancing personal practice via superior problem solving. What about consistency? Well, it is about being reliable in the process of doing it. Summarising, effectiveness is related to personal practice, while efficiency is related to both.

So, in addition to embracing automation, it is necessary to focus on the core, the personal practice. Make it smarter by equipping testers with problem-solving methods to enhance their effectiveness so that (1) issues are found earlier and (2) issues are not missed out to cause test debt. The first (1) is always on the speed of doing. Given that we are building products much faster, it is natural to expect testing to be faster, too. First examine what it takes to do things quickly. Speed is about doing things quickly, with minimal rework. After all, to do things fast, we must only focus on what needs to be done and not be slowed down by the baggage of rework. So what does it take to do it quickly? It is about doing less to be able to do more.

What does minimal rework mean? It is about doing whatever we can early to prevent or detect an issue and doing what we're doing now. So efficiency is not just about accelerating what we are doing now. It also requires effectiveness. Effectiveness implies doing less and doing whatever we do very well. You become efficient in doing things through process or practise, have better skills, or use technology to accelerate the work i.e. automation.

Now let us look at consistency. Consistency is about delivering the same or similar results with similar inputs; it can be considered a function of well-established processes and practices.

Now, does consistency play a role in efficiency and effectiveness? Yes. If we can be consistent by doing certain things that are fundamentally rote, then we will not spend unnecessary time thinking about it. If doing things is mechanical, then technology helps to automate these to reduce the work.

When effectiveness, efficiency and consistency are in harmony, it is a brilliant balance, SmartQA.

S3-Focus on scenarios, then automated scripts

In today's world, we use the words manual and automated. There is serious emphasis on automated testing, as the general belief is that automation is key to delivering great-quality software. Is it true? It enables frequent validation software & systems to be sure that the health of the current state of code is not compromised, which is useful. It appears that evaluating health every day rather frequently is a good quality. But is that true? Remember, we are building products much faster today, which means we are incrementally adding a lot of code to build new features, modify or enhance existing features, or fix some issues found by us or customers.

So every time we touch the code to add, modify, or delete something, there is an innate worry that it could affect the existing code. That the functioning of the current features may be compromised, and therefore we need to evaluate whether it is not compromised. As we speed up the development cycles, this becomes frequent, and hence we tend to move towards automated tests so that we can validate this frequently as and when we build new code and ensure the health is not compromised.

Remember, it is about the health of the existing code before we did this job, so it is like taking the same test cases and rerunning them. So humanly regressing this does not seem very smart, and therefore we should resort to automation testing, which is perfectly correct. But remember, we are only talking about regression most of the time. Automation is not limited to functional regression but also to validating non-functional aspects and many other tasks like test pipelines, data creation, static analysis, etc.

It is important to note that we are evaluating the current health and the changes as soon as we deliver a new set of improved setup features in a product. Unless we have automation along with development, there will always be a backlog of automated regressions, making it harder to fix. What is very important to understand in this entire context is not only automation but also the quality and focus of what is being automated.

1. Are we sure that test cases are adequate, complete, and potent enough to find issues that matter?
2. Do these validate high-level entities like end-user requirement and end-to-end flow to give a clearer picture of the impact of change?

3. How well implemented it is, is important so that changes do not require significant changes in the automated script. UI-based automation is more brittle than non-UI-based automation, i.e., using APIs, and therefore requires less maintenance effort. Note that these require a system tester to be adept with programming and tech-savvy.

Do remember that testing is not just about checking past health; it is also about current health, so there is a need for significant effort in creating newer scenarios and cases, enhancing the existing ones to be powerful and sharper, and factoring these into the act of testing in addition to what we do as automation or automated testing.

In a nutshell, solid tests enable high quality, not the reverse, as it is not the automated tests that always ensure exceptional quality.

S4-Focus on writing less, to do more

Testers spend considerable time in documenting the test cases based on their organisational test case template that outlines out how to execute outlining the detailed steps/procedure. What is wrong with it? It is just that an unnecessary amount of effort and time is spent doing this mundane documentation.

So when I ask them, why do you do this? The typical answer is. Well, the company has given us a template and we need to use it because it could be useful in the future. When a new engineer joins the team this would be a good starting point. I think it's important to understand that especially in this modern world where everything is seemingly Agile, the focus is more on the act of delivering great outcomes. Therefore the focus as far as a test person is concerned, should be on thinking, on what the test is, how should it work, who is going to use it, what could go wrong, what are the various interesting what-ifs, requiring exploration, digging in, deep diving, walking across the entire breadth, understanding the context and then designing, to come up with a first cut of test scenario/cases/data and finally evaluation.

It requires deep intellectual effort in constrained time to think, explore, question, design, and write as minimally as possible. It is necessary in these times to move away from thinking of test documentation as a system of records which may be useful in the future to note-taking to immerse and sharpen thinking, and use the early test cases arrived at as a first cut to evaluate and then refine. The practice of laborious documentation using a document template or as part of a test management tool is arcane.

What is needed is a lot more brevity, a lot more non-textual aspects in terms of pictures and a non-linear way of writing to stimulate thinking to do better.

After all, it is smarter to spend time and effort on doing something that has a direct bearing on good outcomes. This would give us better time & effort utilisation and value. The time we gain from not doing detailed documentation can be put to use fordoing things that are far more valuable like including smart tooling & automation.

Smart testing is about doing lean documentation, note-taking, staying nimble, and being effective & efficient.

S5-Focus on test potency, not quantity

As an engineering manager, you're always keen on understanding or knowing how good the tests are. Is it sufficient? Is it adequate? Does it have good coverage? Merely knowing the number of test cases and their traceability does not add significant value. What is needed is something that gives confidence in the quality of tests before commencing testing.

Smart test analytics helps in building trust & confidence in the coverage of tests. What would this entail? It is about knowing if we have covered all points of view— users & usage, environment, attributes, early-stage entities to end to end flows, and impact of change.

1. Testing is not about evaluating a product from a technical viewpoint; it is about evaluating it from the users' and customers' business

viewpoints to fulfill their needs and expectations. Needs are validated by functional tests whilst attributes are by specialised tests like performance, security, migration, compatibility, load and others.

2. Do we have test cases that go beyond finding issues in our code to those that may surface due to external environments and vagaries in other systems which we interact with?
3. Looking at the types of issues targeted by our test cases enables us to sharpen our goal of seeking defects and therefore enhance coverage.
4. Do we have test cases that ensure correctness and also assess robustness?
5. Do we have test cases that span across different-sized entities like features to end-to-end flow?

HyBIST (Hypothesis Based Immersive Session Testing) sharpens the focus on defects to come with first cut of purposeful test cases that are continually enhanced as test sessions progress while staying vigilant of coverage.

Great coverage implies being adequate, implying lowered defect escapes, lesser rework, and superior utilisation of capacity to deliver faster. This is what SmartQA is all about.

S6-Focus on quality of issues, not quantity

As engineering managers, we look at defect analytics via charts—defect rates, defect distribution, and so on. They give us a good bird's-eye view of bugs with respect to time, status (i.e., closed or open) and distribution by entities (features, flows, requirements).

Is that good enough? It is smarter to view types of bugs, not just arrival and closure rates, in a sense the quality of bugs. I am sure you would agree that quality of tests/test cases is paramount to quality of product. To look at "quality of bugs' is to get a better insight into test quality and therefore product or application quality. Well this is also a reflection of the quality of test practice.

How can we view this? What can we look at?

1. Look at issues from newly built entities versus those that are enhanced or fixed. This helps us understand if we are doing a good job in building or a bad job in enhancements or fixes.
2. Look at the distribution of issues in normal flows vis-à-vis exceptional cases, i.e., bugs due to positive test cases vs. negative test cases. This helps us to understand that if the working system is robust, whether it can withstand abuse. (Negative is about abuse, while positive is about use.)
3. Look at issues by product attributes, i.e., how many are related to functional aspects vis-à-vis non-functional aspects, such as usability, security,

and so on. This reveals if we are validating end-user expectations and not functional needs alone.

4. Look at distribution of issues from user’s or persona’s perspective, View issues as to how may affect end users (persona viewpoint) rather than just product view point. Note we can also analyse from an ‘environment viewpoint’ too.

5. Finally, look at issues by entity granularity, i.e., issues at lower or elemental level, in terms of a feature or component and higher-level issues related to a business flow or an end user requirement.

Thus, we get a better feel for what kinds of issues have been found.

As for issues not yet found, maybe we won’t find them because the product is good or because we didn’t look for them, the latter probably more important. Mere analysis in terms of numbers, distributions, and rates is not sufficient, delve into quality of bugs via a smarter stratification to enhance effectiveness & efficiency so we may accomplish more with less.

S7-Focus on the key measure- “Escapes”

Do you focus on escapes? i.e. defects that have escaped testing. They are like a mirror, they reflect the capability of tests and test teams. Keeping a tab on defect escapes, both internal and external, helps engineering managers to understand leakage in process or practice. The simple analogy is to look at testing as multi-stage filtration, defined as quality levels in Hypothesis Based Immersive Session Testing or HyBIST inspired by fractional distillation. We apply different filters at different levels to catch different kinds of issues. However filters are porous for they cannot be without holes; it needs to have a certain amount of porosity, of varying fineness across levels.

At earlier levels, porosity is about leakage of issues from developers, i.e. dev test, putting pressure on test folks to uncover them. At the later stage(system test) porosity implies that defects escape into the field causing customer dissatisfaction, requires rework and delivered as patches (some urgent) later.

Remember, in addition to the construction phase, the earliest phase of analysis/requirements, there can also be early-stage requirements in construction. If these are not "questioned" well, they form holes during construction as incorrect code/code that should not be there. So these three categories of leakages would be interesting to know on a periodic basis. I.e., requirements to construction, development to test, and test to release.

Escapes are really a mirror, they reflect the test capability, and keeping a tab on this gives us a better sense of how good our filters are, to enhance effectiveness, sharpen the way we do things so we do tests speedily, reduce rework to exploit capacity well. What does this mean? It is about smart defect typing and associating them into SDLC stages so that we can contain them at the right stage. The key concept of defect types, quality levels and entity granularity are central to HyBIST to accomplish just this to do SmartQA.

It is about smart containment, enabled by smart escape analysis to change behaviour from finding to assuring.

S8-Preempt issues, assure not just detect

Sadly, 'testing' as a word is seen as an activity of actual validation, i.e., running the tests. Step back and ask “Is it just about executing or running?” Not really. Testing requires a good set of preparatory work before we get into the act of evaluation, and that demands that the tester get a good, detailed understanding of what he or she is going to test along with the larger context, then figure out scenarios to evaluate. Also, during the evaluation, discover some 'interesting' cases. This is what 'good testing' is all about.'

A lot of meaningful preparatory work is done upfront, and then the preparatory work outputs are used in execution. Often, we don't give much importance to the preparation done before execution. This preparation needs the ability to review a given specification or requirement, to dig in and ask interesting questions about who, where, what, why, what-if, and so on. In the process, we might find some holes, need clarifications, discover potential issues, jot down observations, and come up with suggestions too. We are not just viewing testing as an act of evaluation, but pulling it forward. This process helps clarify things much better.

It also helps us understand the impact of change. It helps us understand the impact of the feature addition or modification, explore it in detail, find the various rabbit holes that a tester can get lost in, and therefore be able to validate much better. Hence, validation is not just about the act of execution; it's about the mental aspect rather than physical validation. Physically validating is about what we do, what we easily understand, and what we typically automate.

Mental validation is an invisible process that goes on in one's brain to figure out possibilities and probabilities of what may go wrong, what may happen that is not yet spec'd out, and come up with interesting questions to answer. Testing is not to be seen as an act of just executing tests frequently.

Let us use an interesting analogy. We know that for good health, activity or exercise is good for the body, but we also know rest and sleep are equally important. It is not just intense activity, i.e., test execution, but quiet thinking, exploration, and mental activity that are equally important. If the latter is not good enough, then the former will be less effective and show up much later.

S9-Detect early without disrupting rhythm (text TBD)

S10-Characterise defects first, then do RCA

Often we want to analyse issues from the perspective of improvement of how we can do better. So we analyse defects to identify root causes of what we can do better. What I have seen in RCA or Root Cause Analysis is an extraordinary amount of fine-grain detail bordering analysis-paralysis in coming up with the self-evident root cause(s) i.e. need more time/effort/capability. Well, this can be discerned without detailed RCA. Isn't it ?

So when we analyse, it may be smart to look at it from these dimensions:

a. From actions to do than infer reasons
b. From SDLC stages so that we know who needs to act
c. From new implementation or enhancement of where to tighten

A smarter defect analysis would therefore be:

1. How many were missed out due to carelessness?
2. How many were missed for lack of scenarios/cases?
3. How many were due to non-availability of environment/customer data sets?
4. How many were due to lack of clarity/ambiguity/misinterpretation of needs & expectations(requirements)?

5. How many were due to “well, never thought about this situation”?
6. If it was due to input spec, how can we preempt this?
7. If it was due to poor construction, how can we make it robust?
8. If it was due to poor test quality, how can we strengthen it?
9. Was it due to enhancement or due to new implementation?
10. Was it due to incorrect judgement of correctness?

A smarter analysis would go a long way to get a better insight into the type of defect and why it may have been missed out, thus enabling us to learn, improve, and become smarter.

Purposeful defect analysis is like a perfect mirror that enables one to continually adjust and discover the optimal path, the hallmark of smartness.

S11-Unshackle capacity to handle regression

We want to build rapidly but have a problem with capacity. What can we do? As we speed up the product development cycle by embracing Agile, it is a challenge to ensure that the quality of each sprint is not compromised. What is the challenge? The challenge is to avoid compromising health by frequently adding, updating, and modifying.

In a sense, we end up doing far too much regression testing. Secondly, as we speed up, the challenge seems to come from a lack of time due to the time crunch to test. Well, let's dwell on what is being tested. Is it execution?' No, because it involves understanding what was intended by reading, reviewing, discussing, exploring, and questioning, and all these take time. Then, come up with interesting scenarios to evaluate, and then automate as appropriate to convert them into automated scripts. Then, of course, evaluate using automated tests using humans, also doing disciplined vs. ad-hoc tests. As time gets crunched, we seem to be running into a capacity problem, and to resolve this issue, fix one component regression with automation.

That turns out to be challenging because automation also requires meaningful effort and further crunching capacity. What do we do? Here is why I think it's necessary to figure out if it requires evaluation at a later stage: Could we have done something earlier? Could we have looked at possibly finding some of these issues earlier, maybe by pre-empting, interrogating, or reviewing? Let's say the requirements given to us are user stories, no interrogating stuff to figure out what-if scenarios, or at the implementation stage by developers for certain kinds of issues.

So we can't just add more capacity to solve this capacity problem because it requires more money, which is always a scarce commodity. So it is necessary to get smarter to be able to accomplish this. Smarter means doing less, doing earlier, exploiting technology, not doing it, preventing it, etc.'

Solving the capacity problem is not about adding more. It is about figuring out (a) how to do less, (b) how to do it earlier, (c) what not to do, (d) what to do manually, and (e) what can be automated using tools or technology. Considering these things together and addressing them can help solve the capacity problem. The capacity issue is a challenge that cannot be compromised to result in defect escapes.

S12 (#5) Exploit intelligence, artificial and human

In today's world, artificial intelligence has usurped human intelligence. Is there a role for human intelligence in testing? We have dichotomized testing into manual and automated. Sadly, the word manual is highly incorrect. It should be human testing, i.e., using human intelligence to test or validate, while automated testing is about using technology, tools, and machines to validate software.

Now we have the new cat out of the bag: AI, which has recently made waves with generative AI. We think that this will replace human tests. So, let's get back to the subject of what is exploiting intelligence. There are certain things that a machine (technology) can do very well, and one needs to be smart enough to exploit them.

If we can do faster using a machine, then do so. If some things can be observed, perturbed, or exercised by using a tool or technology, then it is 'smart' to exploit it to do so. But understanding, figuring out questions, connecting various parts to see the big picture and using that to evaluate requires human intelligence. Today, this can be augmented with AI systems and software, it is like getting help from an intelligent assistant. But figuring out the test approach, understanding behaviour and identifying the conditions, understanding the human psyche, the environment in which they use it, and the constraints, expectations, and potential mistakes that one might make are still things that require human intelligence.

It is necessary to exploit human intelligence, as testing is not merely a notion of evaluation at the end. It demands a systems-thinking approach to "look at individual trees but also understand the larger context of the forest". Human intelligence plays a definitive role, and today it is aided by the intelligent tools that are more supplemental than replacement.
Repetition of tests and probing that is hard to be done by a human are those that demand tools/technology i.e., machine intelligence, that we state as automated testing. A harmonious mixture of machine intelligence and human intelligence helps deliver clean code. Testing is not always postcode-based; it is about doing earlier to detect, prevent, or to enhance clarity.

Testing is more than just determining whether something is correct or incorrect. It is about walking into the sphere of the unknown, not necessarily doing what is right or wrong; it is a process of discovery that demands an intelligent human.

S13 (#11) Expect more out of a professional tester

Do you view testers as one who executes tests and reports issues? Good managers expect more, impact/risks, suggestions, ideas, interesting observations from multiple viewpoints of users, product, environment etc.

When managers expect of them a deep ownership mindset, then they expect ownership mindset, they may turn out to be merely executors.. If they see themselves as “I am going to buy this product and consume it for my business,” then the way they look at the product will be quite different, with an eagle eye for issues that may inhibit work, and ideas/suggestions to enhance quality of work. The mindset changes, now sharply focused on great user experience and higher business value.

With an ownership mindset, it is no longer just about executing test cases or writing test scripts and finding issues. It is a mindset that becomes sharply critical to identifying aspects that could be done better; suggestive, observing every minute thing, getting into the skin of end users, and therefore not just evaluating, validating, or executing.

If an engineering manager expects ownership mindset in them, then testers will see their job as more challenging and value-adding, moving from postcode assessment to earlier state critiquing, ideation and suggestion forcing them to think: “What is the business value that I have added to the test, to the product, to the customer and their business, and finally to the company and its business?”

So setting those expectations with the team and viewing them as more than mere executors enables the team to have a smarter, sharper mindset and certainly much deeper ownership that will transform them into smarter QA folks.

Expect smarter outcomes from QA in addition to execution and issue rather than mere evaluation activities. Ensure that the team has a smarter, sharper mindset and certainly much deeper ownership that will transform them into smarter QA folks.

Introducing SmartQA

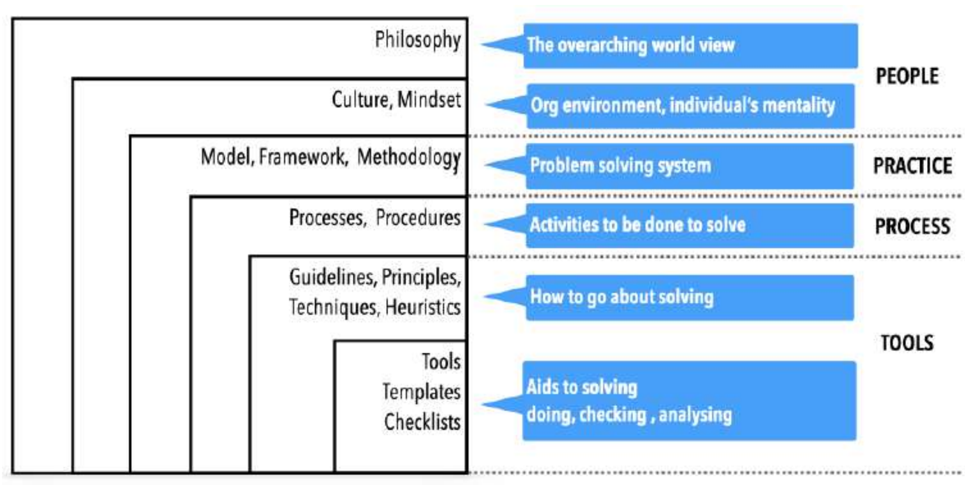
SmartQA is an intellectual practice of going deeper to seek clarity & in the process uncover, preempt issues rapidly, not limited to mere validation of code. It is a brilliant combination of checking for expected, exploring the whole, looking for unexpected, uncovering issues, suggesting needs not yet thought of, improving what is done, and sensitising to prevent issues.
It is about doing less to accomplish more.

Introducing HyBIST - a method to do SmartQA

HyBIST is an intellectual practice driven by hypothesis based core method to analyse & design, and do immersively in sessions to test deeply & rapidly.

- Connect HyBIST to Philosophy, Mindset, Methodology, IST as Process (Highlight that we do not touch org process), Smart Checklists, highlight that

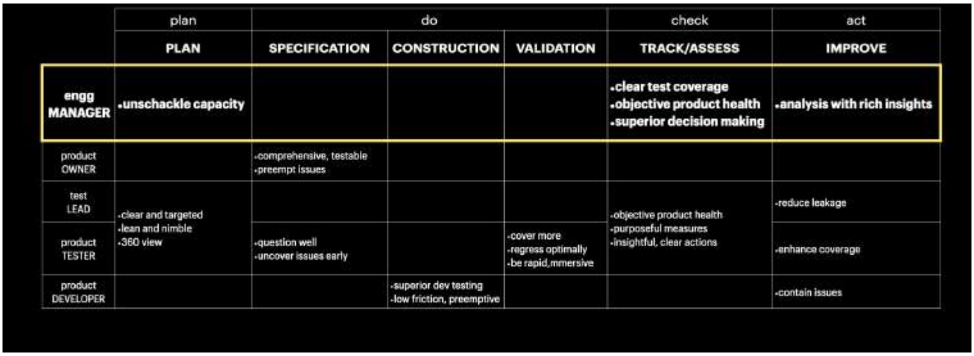
HyBIST can make an organisation do SmartQA(enterprise-wide), not just transform an individual.


Philosophy
Culture, Mindset
Model, Framework, Methodology
Processes, Procedures
Guidelines, Principles, Techniques, Heuristics
Tools Templates Checklists
The overarching world view
Org environment, individual's mentality
Problem solving system
Activities to be done to solve
How to go about solving
Aids to solving doing, checking , analysing
PEOPLE
PRACTICE
PROCESS
TOOLS

The SmartQA Promise

The SmartQA promise to engineering managers are:

- Unshackle capacity to accomplish more with same/less
- Superior test coverage to enhance test effectiveness, lower rework
- Objective health assessment to aid better decision making
- Provide rich insights to foster practice improvement



	plan	do			check	act
	PLAN	SPECIFICATION	CONSTRUCTION	VALIDATION	TRACK/ASSESS	IMPROVE
engg MANAGER	•unschackle capacity				•clear test coverage •objective product health •superior decision making	•analysis with rich insights
product OWNER		•comprehensive, testable •preempt issues				
test LEAD	•clear and targeted •lean and nimble •360 view				•objective product health •purposeful measures •insightful, clear actions	•reduce leakage
product TESTER		•question well •uncover issues early		•cover more •regress optimally •be rapid,mmersive		•enhance coverage
product DEVELOPER			•superior dev testing •low friction, preemptive			•contain issues

SmartQA Outcomes

<u>Left shifting : Improve DevTest</u>

Reduce defect escapes from Dev to QA by 60% & 25% in two software product companies.

<u>Enhance quality of user stories</u>

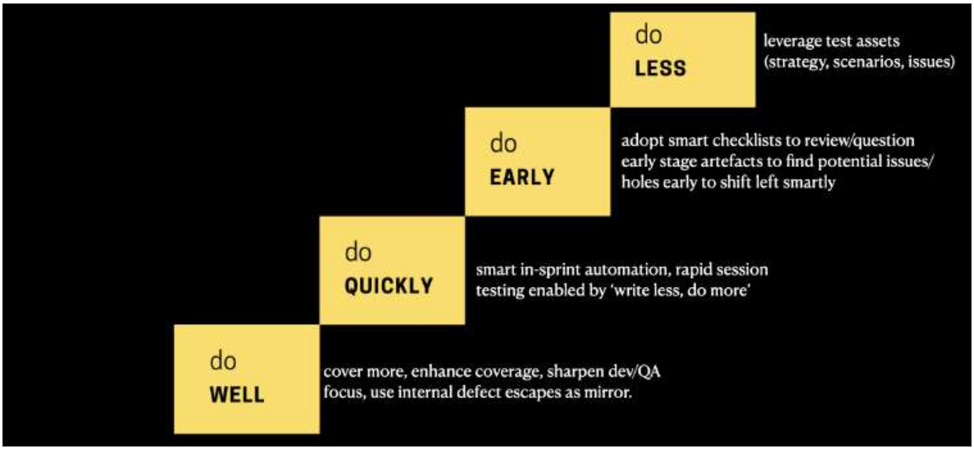
Sensitising & preventing issues on user stories i.e. testing requirements in a German product engineering major. Sharpen sensitivity to issues in Agile sprints.

<u>QA Practice Improvement</u>

- Enabled a Chipcard major to find dormant issues that would have derailed certification & improved coverage by 10+% on their already well tested code.

- Enabled large Japanese SI to rollout large applications (instant cutover) with no hitch.
- Vastly improved test assets and practice of QA to instil high confidence in senior management of a complex product
- Significantly enhanced test practice in a large product major (300 people world wide engg team) making their Agile journey sharpen their focus on tech debt. Changed mindsets, re-jigged practice, helped in automation strategy and sharpened focus on measurement and actions.

Embracing SmartQA


do LESS
leverage test assets (strategy, scenarios, issues)
do EARLY
adopt smart checklists to review/question early stage artefacts to find potential issues/ holes early to shift left smartly
do QUICKLY
smart in-sprint automation, rapid session testing enabled by 'write less, do more'
do WELL
cover more, enhance coverage, sharpen dev/QA focus, use internal defect escapes as mirror.

First “do WELL” - cover more, enhance coverage, sharpen dev/qa focus, use internal defect escapes as mirror.
Secondly “do QUICKLY” via smart in-sprint automation, rapid session testing enabled by ‘write less, do more’
Thirdly “do EARLY” - adopt smart checklists to review/question early stage artefacts to find potential issues/holes early to shift left smartly
Lastly “do LESS” - leverage test assets (strategy, scenarios, issues)

STAG Software focuses on enabling organisations to embrace SmartQA via:

1. HyBIST Courses for Product owner/Mgr, Dev & QA to transform mindsets
2. SmartQA Consulting & Advisory to assist in practice implementation
3. doSmartQA - HyBIST test analysis & design tool