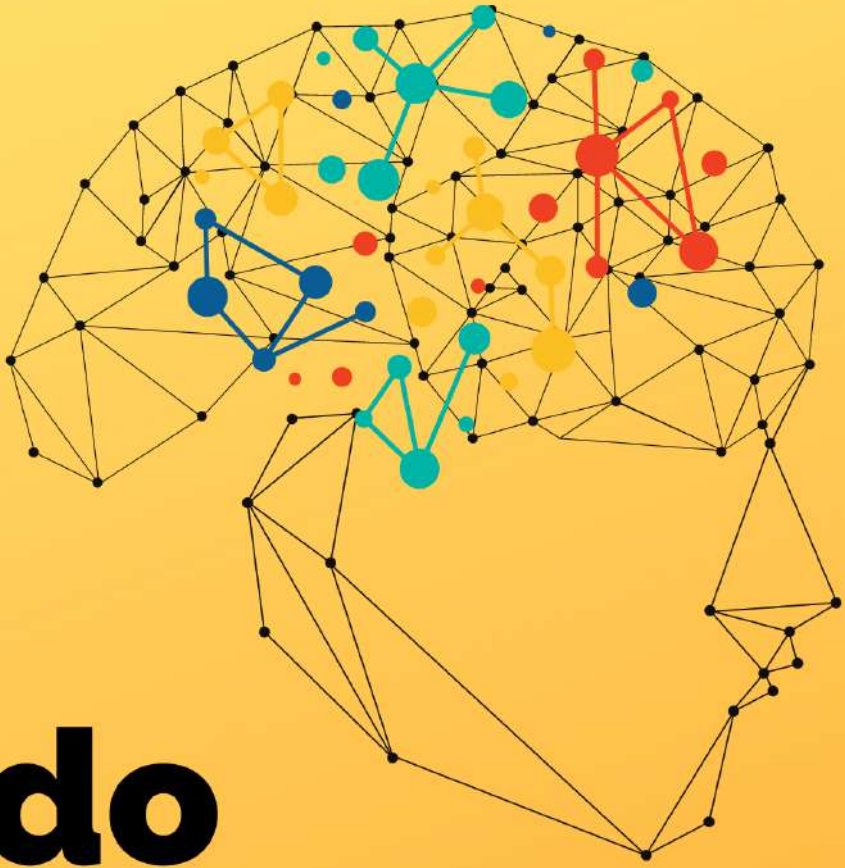
Unshackle capacity.Test deeply. Test rapidly.

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



do SmartQA.

The HyBIST Approach

THIRUVENGADAM ASHOK

PREFACE

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

The ideas outlined here are the outcome of my two decades plus research and application of scientific approaches to bug finding focussed on developing a personal methodology to making software testing effective and efficient.

TABLE OF CONTENTS

Chapter #1: The backdrop “State of affairs”

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)

Chapter #2: Industry Challenges in QA

Here I outline some of the key challenges observed in my numerous interactions with engineering/QA leadership spanning across organisations in different domains, technologies, maturity levels and sizes.

The six challenges outlined here are:

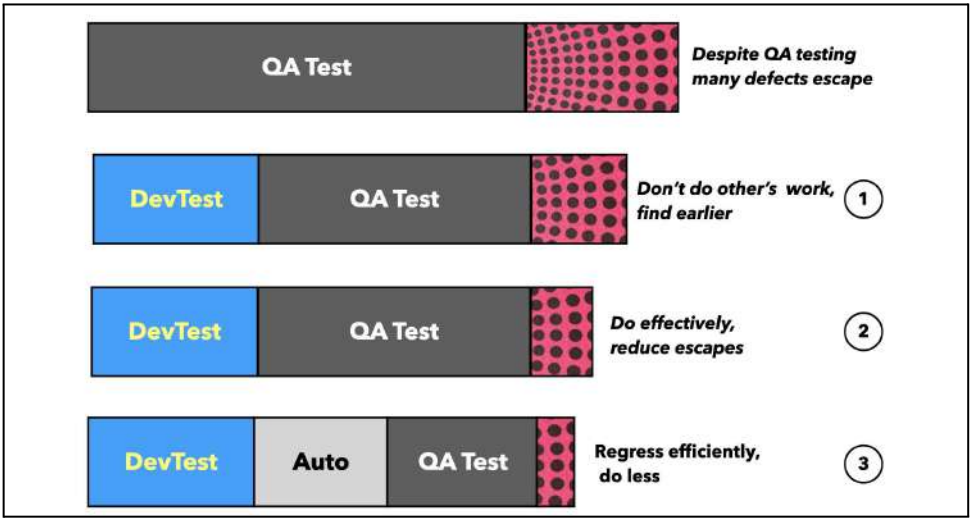
1. Grappling too much testing?
2. Choked by bugs?
3. Challenged by test adequacy?
4. Troubled by development test?
5. Weighed down by automation?
6. Are your quality metrics insightful?

Challenge #1 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?

QA doing work that Dev Test should have covered, writing unnecessary detailed test documentation, relying solely on scripted

tests, doing too much regression, creating a large automation backlog due to poor test organisation are some key reasons I have noticed that saps capacity and effectiveness of QA.

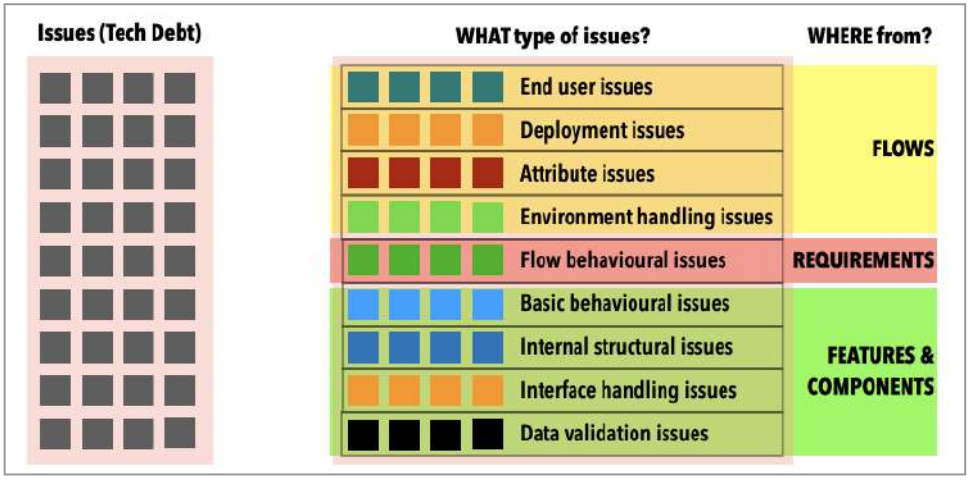


Test automation is certainly a way forward, but after enhancing effectiveness via SmartQA. Sharpen QA focus, strengthen scenarios, don't do others work, and then automate is what SmartQA is.

Challenge #2 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.

Challenge #3 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 is very useful, it allows one to question deeply, which is what brilliant testing is all about.

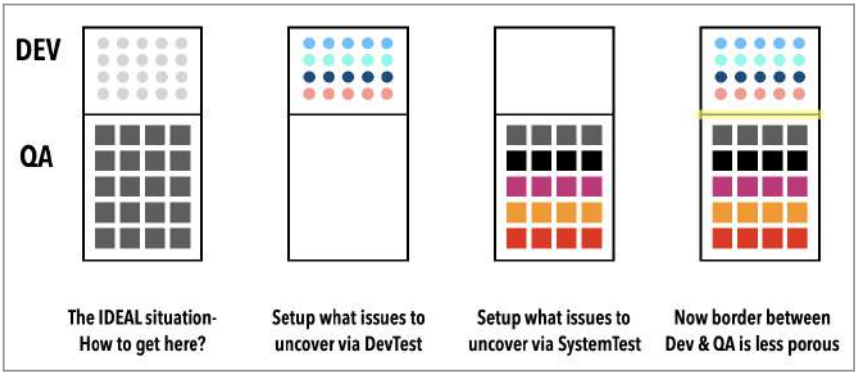
A behaviour driven approach to test design enables a mindset to extract conditions from requirements, understand perturbations from other parts of the system leading to 'robust test design'. After all, a good filter tightens the noose!

Challenge #4 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.*"

Early stage code quality is strengthened if specific types of issues are targeted, 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 “left-shifting”, 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.

Challenge #5 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 the system health especially if the regression tests key flows rather than features or requirements.

I have seen automation embraced as the solution to speed up testing. The challenge is however keeping the 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 focus on finding issues(robustness). A harmonious balance between these two enables clean code to be delivered without being weighed down by automation.

Challenge #6 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 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 is 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 HyBIST (Hypothesis Based Immersive Session Testing) provides a strong foundation for these, enabling one to assess potential test effectiveness, judge product quality objectively and fine tune practice quality .

Chapter #3: Introduction to SmartQA & HyBIST

What is SmartQA?

Testing is not about merely checking for compliance, nor is it just about finding issues with delivered code; it is being curious about what may be intended by producer, what may be expected by consumers, the correctness and incorrectness of what is present, 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 an intellectual practice to preempt issues in early-stage spec, design, and code.

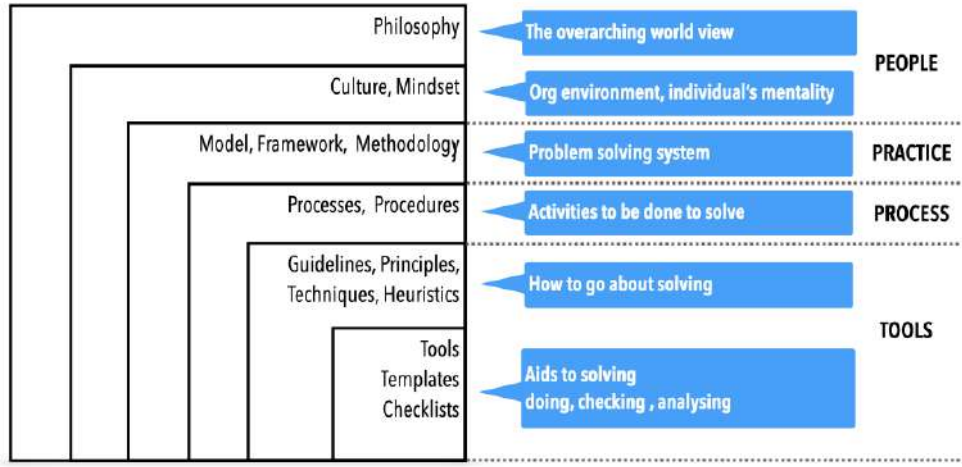
SmartQA is an intellectual practice of going deeper to seek clarity & in the process uncover, preempt issues rapidly, not limited to mere validation of code. It is a brilliant combination of checking for expected, exploring the whole, looking for unexpected, uncovering issues, suggesting needs not yet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

It is about doing less to accomplish more.

HyBIST - An approach to SmartQA

HyBIST(Hypothesis Based Immersive Session Testing) is an intellectual practice driven by a hypothesis based core method to analyse & design, and then done immersively in sessions to test deeply & rapidly.

Brilliant testing happens when there is an overarching philosophy that shapes an individual mindset becoming an organisation culture as a collective that results in good practice assisted by a process & tools to take that forward. HyBIST is precisely this, a personal practice guided by the philosophy of clarity & understanding as central to quality resulting in a curious questioning mindset that is logical yet creative. Smart Checklist is one such tool to enable this practice.



Chapter #4: SmartQA Suggestions

SmartQA is about doing less to accomplish more and here are THIRTEEN suggestions based on “Do less, Do better, Do faster & Don't do”.

1. Focus on direction first, then speed
2. Focus on practice & process, then tools
3. Focus on scenarios, then automated scripts
4. Focus on writing less, to do more
5. Focus on test potency, not quantity
6. Focus on quality of issues, not number
7. Focus on the key measure- “Escapes”
8. Preempt issues, assure not just detect
9. Characterise defects first, then do RCA
10. Unshackle capacity to handle regression
11. Exploit intelligence, artificial and human
12. Expect more out of a professional tester
13. Detect early without disrupting dev rhythm

Suggestion #1 Focus on direction first, then on speed

Given that testing is done frequently in every sprint, there is a definite need for speed and therefore automation of testing. But before you get on to the fast moving lane of validation, pause and be sure that the direction you are headed is right.

What does this mean? It is about starting off with a clear purpose of what, what-for, where and how of test/retest, taking into consideration the various points of view: entities, end users, environments, structure and interactions. It is about clarifying the purpose, setting up direction and then constantly adjusting/refining it during the act of validation.

In HyBIST the purpose is expressed as a Cartesian product: What-to-test/check X What-to-test/check-for X Where-to-test/check X How-to-test/check i.e. {entities X potential issues X environment X how}. Check implies compliance to stated (regress for existing) while test implies looking for issues,the unexpected. Note that How–to-test/check may be also done with Smart Checklists to not only detect but preempt common issues.

Viewing SmartQA as an intellectual practice makes the act purposeful, setting up a clear direction, and then speeding up the validation activities to deliver great outcomes.

It is not speed that matters, it is velocity velocity. Be clear of the specific purpose i.e. direction, before you go full on.

Suggestion #2 Focus on practice & process, then tools

Continuous development, frequent build & validation, the relentless regression to ensure existing(prior) code is not compromised are

certainly interesting demands in today's development regime.

How does one stay on top of this? By automating all tests? Is it about being able to do more by leveraging tools? SmartQA is about doing less to accomplish more, and is not merely about doing more in less time. Let us look at the solution from the aspects of being effective, consistent and efficient.

Firstly sharpen the individual practice via being purposeful, enabling clarity to be effective, and being nimble in the doing. Enhancing individual effectiveness implies less re-work in future and better capacity utilisation.
Secondly tighten and lighten the organisational practice to be meaningful and very consistent. Ensuring consistency implies enhanced bandwidth for doing interesting things.
Finally leverage tools to speed up and lift up mode to be superior and be efficient.

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

Suggestion #3 Focus on scenarios, then automated scripts

The dichotomy of testing into manual and automated should really be intellectual and mechanical of exploiting human smartness and leveraging tools. Is automation key to delivering high quality software?

The need for frequent & continuous validation demands automating the standard health checks to be sure that additions and modifications do not compromise the health of prior working software.

Quality is about what was working well continues to work well plus what is newly implemented does indeed meet the needs and expectations. Note that the latter needs to be done before it turns into the former. This means it is imperative that we have potent scenarios to weed out potentially interesting issues first, and then constantly assess to check the health. This needs to be done not just at a lower level of technical features but also for higher order end flows, with effective scenarios that span across the levels. A subset of these scenarios(at the least) could become health checks, are prime candidates turned into automated scripts. Delineate behaviour validation from the front-end so that scripts may not suffer from frequent updates due to changes done on the human interface.

Be sure that it is done well by exercising potent scenarios and then assess frequently that it is not compromised.

Suggestion #4 Focus on writing less, to do more

Testers spend considerable time on documentation especially in the phase of design often driven by templates. Design truly is intellectual and creative and less conformant to typical low level templates which relegates it to a procedure.

In cases of regulated testing detailed documentation is necessary and often serves as a system of records. The act of writing should not just serve as a record for the future, it should serve as stimulating interesting ideas & thoughts for smart QA. It is about note-taking to jot down interesting observations, potential issues, scenarios to try out, suggestions & ideas while examining spec/product before it is formally distilled into a staid document.

It requires deep intellectual effort 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 that may be useful in the future to note-taking. This helps to immerse and sharpen thinking, use the early test cases as a first cut to evaluate and then refine. The practice of laborious documentation using a template or as part of a test management tool is arcane. The time we gain from not doing detailed documentation can be put to use to doing things that are far more valuable like including smart tooling & automation.

Focus on doing great work first and then protect the future. SmartQA is about lightweight note-taking, lean documentation and nimbleness to be effective & efficient.

Suggestion #5 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. Do we have test cases from a technical, business (end-to-end) view?
2. Do we have functional and non-functional test cases?
3. Do we have specific test cases for specific environments?
4. Do we have test cases that assess correctness & robustness?

5. Do we have feature level and end to end flow test cases?
6. Do we have test cases from different end users' point of view?

HyBIST quality levels sharpens focus on defects to come with a broad and deep net. It helps come up with a good set of initial test cases that are continually enhanced as test sessions progress to enhance coverage. It is not about more test cases(quantity), it is about being potent, minimal yet adequate.

Test potency is about good defect finding power, less escapes, lesser rework, and superior utilisation of capacity to deliver faster. This is what SmartQA is all about.

Suggestion #6 Focus on quality of issues, not number

As engineering managers, we look at defect analytics via charts—defect rates, 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? Seeing defect distribution by types i.e. in which quality level as in HyBIST gives a better insight into product quality, a reflection of test effectiveness. How can we view this? What can we look at?

1. Look at issues from new entities versus those enhanced or fixed. This helps discern if we are doing a good job in building or a bad job in enhancements or fixes.
2. Look at distribution of issues in normal flows vis-à-vis exceptional cases, This helps us to understand that if the working system could be robust i.e can withstand abuse.
3. Look at issues by functionality & attributes too, that we are validating end user expectations and not functional needs alone.
4. Look at distribution of issues from user's or persona's perspective, how may it 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. lower level features vs. end to end flows.

Mere analysis of issues in terms of numbers, distributions, and rates is not sufficient, diving into quality of bugs via smarter stratification helps to enhance effectiveness & efficiency, to accomplish more with less.

It is not just the number of issues, it is the quality of issues found that matter.

Suggestion #7 Focus on the key measure- "Escapes"

Metrics help in understanding activities and outcomes and help us improve by changing behaviour. One of the most powerful metric that acts like a mirror is "defect escapes" i.e. defects that have been missed. HyBIST views testing as a fractional distillation with filters of varying porosity to catch different types of bugs at different levels. Given that filters are porous, defects will leak and escape, measuring this helps in fine tuning the act of filtration. Early stage filters are pre-code & early stage code, defect leakages are reflection of requirements validation and development testing while later stage filters are a reflection of professional QA validating the whole system.

It would be useful to analyse three categories of leakages periodically- requirements to construction, development to test, and test to release so that we constantly tune the filter, to reduce rework and unshackle capacity.

Measures or metrics help in understanding facts objectively to course correct, to improve and most importantly change behaviour which is a serious transformative act. Defect escapes are the latter, very powerful, in a transformative sense to become smarter.

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

Suggestion #8 Preempt issues, assure not just detect

Testing is more often seen as an intense activity of evaluation. Is it just about executing or running? Not really. Testing requires good preparatory work before the act of evaluation, it requires the tester to understand the larger context, then figure out scenarios to evaluate and discover some 'interesting' cases during evaluation.
Prep work is understanding the big picture, asking interesting questions about who, where, what, why, what-if, and so on. In the process find missing information, 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 and preempt issues.

It also helps us understand the impact of change, of feature addition or modification, in the process of exploration, meander into rabbit holes to understand interconnections and impacts to validate deeply. Validation is not a physical act of execution; it is intellectual, 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 to preempt, to assure. Just like for good health, activity or exercise is good for the body, but rest and sleep are equally important.

Testing is not always an intense activity, it is silent thinking, quietly exploring in the mind's eye too, to visualise correct flows and see issues too, to preempt.

Suggestion #9 Characterise defects first, then do RCA

Often we analyse issues from the perspective of improvement of how we can do better by trying to identify root causes. 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. These can be discerned without detailed RCA, right?

When we analyse, it may be smart to look at it from these dimensions:
1.From actions to do rather than figure out reasons
2.From SDLC stages to figure out who needs to act
3. From new implementation or enhancement of where to tighten

A smarter defect analysis would therefore be:
1. How many missed out due to carelessness?
2. How many missed for lack of scenarios/cases?
3. How many due to non-availability of environment/customer data sets?
4. How many due to lack of clarity/ambiguity/misinterpretation of spec?

5. How many 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?

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 enabling one 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.

Suggestion #10 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.

Suggestion #11 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 assessing whether something is correct or incorrect. It is an exploration into the unknown, of discovering interesting stuff that demands an intelligent human.

Suggestion #12 Expect more out of a professional tester

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

When managers expect from QA folks a deep ownership mindset, then the way QA folks 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. Their 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.

When 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, to my organisation and its business?”

Expect from QA how to enhance experience in addition to finding issues, so that they may have a deep ownership mindset.

Suggestion #13 Detect early without disrupting dev rhythm

A compromised ‘unit test’ puts unnecessary strain on QA folks who seem to be compelled to go after these issues at the expense of system test.When we find issues in the product/app especially those that can be caught earlier, we focus on more rigorous dev test with extreme focus on automation. Seems logical, but given that a developer is already under intense time pressure, is it the right approach?

The typical approach of “more early testing” is harder to implement. Testing building blocks should be easy and quick, in, not so well done in reality.. A smart lightweight approach that does not create more

work for developers or upset the development rhythm is what is paramount to success. Great quality early stage code is not about doing more testing, it is about doing less testing, by enabling sharper focus on 'what-can-go-wrong', 'have-you-considered-this'.

What if we changed our views?
1. That testing need not be limited to dynamic evaluation, it could also be done via static proving. That ascertaining correctness is not only via dynamic execution of test cases but by thinking through what can happen with various data sets.
2. Instead of starting with conformance test cases, start in the reverse with non-conforming data sets first. Prove that the system rejects bad inputs before we evaluate for conformance.
3. Instead of designing test cases for every entity, use a potential defect type (PDT) catalogue as a base to check for non-conformances first (HyBIST Smart Checklist).

Do better developer testing by being friction-less, not upsetting the development rhythm, by doing less smartly. Given that dev test is largely about detecting issues in quality levels L1-L4 (of HyBIST), use Smart Checklists (HyBIST Aids) to sensitise & prevent or detect easily.

We don't need a jackhammer to hit a nail!

Chapter #5: do SmartQA. The way forward.

The SmartQA Promise

The SmartQA promise to engineering managers are:

- Unshackle capacity to accomplish more with same/less
- Superior 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	**.unshackle 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,immersive		.enhance coverage
product DEVELOPER			.superior dev testing .low friction preemptive			.contain issues

SmartQA Outcomes

Left shifting : Improve DevTest

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

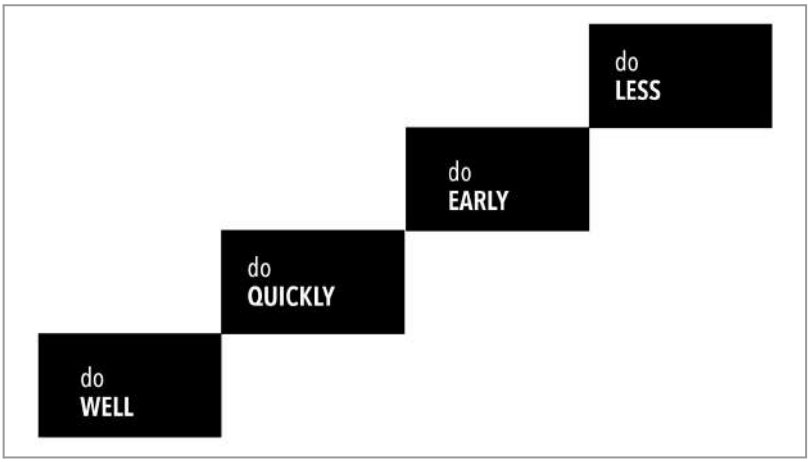
Enhance quality of user stories

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

QA Practice Improvement

- 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



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, immersive 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 enables organisations to embrace SmartQA via:

1. HyBIST Courses for Product owner/Manager, Dev & QA
2. SmartQA Consulting & Advisory to help implement practice
3. doSmartQA - HyBIST test analysis & design tool
4. SmartQA Coaching for Test Leads

About the author

Thiruvengadam Ashok is the Founder & CEO of STAG Software (stagsoftware.com), a pure play test boutique. Passionate about problem solving, he has been focused on methodologies to scientifically test software and is the architect of HyBIST. A believer of "Simple is complex' , he revels in taming complexity and enjoys the learning and discovery it offers. A strong believer in opposites, the Yin and Yang, he strives to marry the western system of scientific thinking with the eastern system of belief and mindfulness.

He is an alumnus of Illinois Institute of Technology, Chicago and College of Engineering, Guindy. An avid long distance cyclist and ultra marathoner he loves writing poems on various topics including testing!

He can be reached on LinkedIn at linkedin.com/in/ashokstag.