


SmartQA Wisdom

on

Smart Assurance



Thiruvengadam Ashok
STAG Software

A Transformative Guide to Smarter Quality Assurance

Wisdom on Smart Assurance offers a transformative approach to quality assurance, moving beyond traditional methodologies to deliver smarter, more efficient results. This emphasises the integration of human ingenuity with machine precision, enabling readers to achieve excellence in QA through thoughtful strategies, practical insights, and timeless wisdom.

Key Themes

- **The Thinking Mindset:** Learn to balance logic and creativity, empathy and precision, to cultivate a mindset that fosters clarity and innovation in QA practices.
- **Crafting Brilliant Code:** Explore actionable principles for writing robust, maintainable, and elegant code that minimises errors and maximises quality.
- **Smart Test Lifecycle:** Discover innovative approaches to test design, validation, and organization, focusing on achieving impactful results with minimal effort.
- **Wisdom in Testing:** Adopt a wisdom-driven approach to QA that emphasises prevention, sensitivity to potential issues, and strategic validation.

Core Highlights

- **See the Big Picture:** Evaluate systems from multiple perspectives—users, environments, attributes, and behaviours—for a holistic assurance approach.
- **Minimalism in QA:** Do less but accomplish more by focusing on purposeful test cases and leveraging intelligent tools.
- **Building Resilience:** Design systems that are robust and testable, enabling rapid debugging and reducing dependency on exhaustive testing.
- **Transforming into a Smart Tester:** Gain practical guidance on becoming a modern, tech-savvy QA professional who thrives in Agile environments.

About the author

Thiruvengadam Ashok is the CEO of STAG Software Private Limited & Co-Founder of Pivotrics, based in Bengaluru, India. Ashok has dedicated his career to the pursuit of quality assurance in software, continuously evolving his approaches to match the needs of modern systems. He is the creator of HyBIST, an innovative approach to SmartQA that has revolutionised how teams approach hypothesis-driven testing.

Ashok's professional life is deeply intertwined with his personal philosophy. A passionate ultra-distance runner and long-distance cyclist, he applies the principles of endurance and exploration to his work, constantly seeking out new ways to improve software quality. He is also an avid wordsmith, using his love of language to weave both poetry and technical innovation into his life's work.

He holds an M.S. in Computer Science from the Illinois Institute of Technology, a Bachelor's degree in Electronics and Communication Engineering from the College of Engineering, Guindy, and a Postgraduate Diploma in Environmental Law from the National Law School of India University, Bangalore. His life maxim—"Love what you do & Do only what you love"—is reflected in everything he undertakes, from professional projects to personal passions.

Copyright © 2025, Thiruvengadam Ashok

All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations used in reviews and certain other noncommercial uses permitted by copyright law.

Disclaimer:

The information contained here is for educational and informational purposes only. While every effort has been made to ensure the accuracy of the content, the author and publisher make no representations or warranties regarding the completeness, accuracy, or applicability of the information provided. The strategies and methodologies described are for informational purposes and should be adapted to individual circumstances as necessary.

Trademarks:

All product names, logos, and brands mentioned in this book are the property of their respective owners. Use of these names, logos, and brands does not imply endorsement. HyBIST is the intellectual property of STAG Software Private Limited.

Edition: First edition, 2025.

TABLE OF CONTENTS

The Thinking Mindset for Smart Assurance	6
The Doing Aspects for Smart Assurance	8
Crafting Brilliant Code - Assuring Early	11
Smarts in the Test Lifecycle	13
SmartQA Thinking	13
Smart Understanding	13
Smart Design I	14
SmartQA Design II	15
Smart Validation	16
Smart Plan	18
Team Structure for Smart Assurance	19
Enhance Productivity	20
See the big picture, before you jump in to test	20
Map who uses what, to understand user needs	20
Map attributes to entities, to identify expectations	21
Create environments by combining elements & values	21
Link related entities via associations & compositions	21
Do a simple focused plan for every test session	21
Dive deep to understand an entity well	21
Rapidly design scenarios and refine	21
While you execute tests, observe & refine	22
Organise scenarios well to be effective	22
Analyze test scenarios for effectiveness	22
Analyze execution progress constantly	22
Analyze product quality at end of every session	22
Transforming into a Smart Tester - Tips	23

SMARTQA WISDOM ON SMART ASSURANCE

THE THINKING MINDSET FOR SMART ASSURANCE

(from 5 Thoughts for SmartQA)

It takes a brilliant mindset, intelligent exploration, diligent evaluation, keen observational skills, tech savviness and continual adjustment. It is about being logical yet be creative, it is about being disciplined yet be random, it is about exploring the breadth and depth, it is about understanding deeply and also finding blind spots about being bundled by time but be unlimited/unbounded with the possibilities. Doing SmartQA is about doing mindfully, in a state of brilliant balance.

1 Embrace multiple thinking styles

Inculcate the deductive ability of a mathematician, creativity of an artist, mindset of an engineer, value perception of a businessman, technical savviness, empathy, doggedness and nimbleness.

#2 Analyse well, exploit tools for doing

Humans & Machines : Doctors & diagnosis - In today's medicine, we know that machines play a huge part in diagnostics and treatment. They help us see internals more clearly, enable us to get to the hard-reach parts, perform rapid tests to analyse problems , monitor tirelessly to help us correct our actions. So is the doctor's role redundant? Ouch no! The skill of the doctor in diagnosis and treatment be it via medicine or surgery is far more required now in the complex world of disease, business and law. To assist in this ever increasing complexity, machines are becoming integral for the job.

Much like the skill of a doctor to diagnose with exploiting the machines in the process. "Doing SmartQA"s a brilliant combination of "human powered and machine assisted" . The WHAT to-do is human while HOW-to-do is powered by machine/tools.

#3 Adopt minimalism

Minimalism has always been a guiding principle—doing less work while achieving superior outcomes. There's no preference for an excess of test cases or the resulting need for tools to manage them. Tools, however, are exploited to deliver brilliant work. This philosophy aligns perfectly with the concept of 'doing SmartQA.'

In discussions about left-shifting, TDD, and finding issues earlier, the focus often shifts toward increasing unit tests. But is the objective to create more tests, or to produce cleaner units? The aim should be heightened sensitivity to issues through TDD, writing better code from the start, and using cheaper static methods to catch what might be missed—before resorting to costlier automated unit tests.

SmartQA is fundamentally about 'not doing'—about doing less, but smarter. It's not about doing more and accumulating tools but about sharpening sensitivity, cultivating good coding habits, leveraging mental aids like smart checklists, and using software

SMARTQA WISDOM ON SMART ASSURANCE

tools to handle the heavy lifting of testing. After all, isn't wellness more desirable than expending effort diagnosing potential illness? And wouldn't you prefer a doctor who checks thoroughly before outsourcing diagnosis to machines?

Strive to prevent issues, embed testability, review code carefully, use smart checklists, write minimally, regress intelligently.

4 Have a mindset of brilliant engineering

Is testing merely an act of uncovering bugs? Certainly not. It is a mindset focused on clarifying thought. When developing something—such as code—a smart testing mindset enables stepping outside the role of producer and into the shoes of end users. It fosters empathy, helping to see their perspective, appreciate their environment, and understand what might go wrong, ultimately leading to cleaner code.

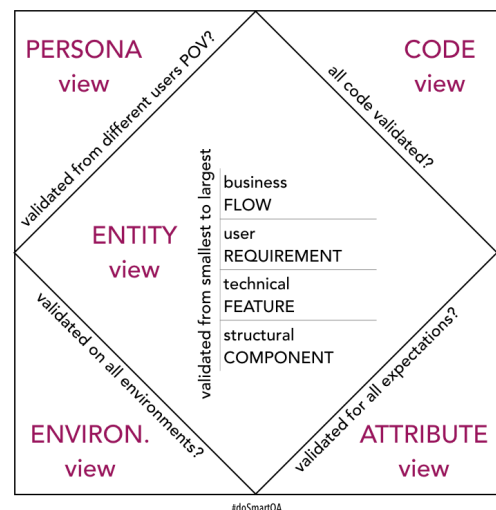
In the worst case, hooks can be placed within the code to provide insights into how it is being affected, allowing for later examination and refinement. SmartQA is not just about finding bugs; it is about adopting a mindset of "brilliant engineering."

Step into end user's shoes, architect/design robustly, inject code to aid testability, strive to test minimally, do test related tasks lightweight.

5 See better, cover more, test less

A well-rounded 'world view' is key to achieving excellent coverage in testing. But is coverage limited to execution alone, or does it enhance our ability to see better?

Coverage is about gaining a clearer perspective from all angles. It goes beyond simply determining whether tests could be or are effective. By adopting viewpoints from USERS, ATTRIBUTES, ENVIRONMENT, CODE, and ENTITIES, we gain multidimensional insights, allowing us to deliver brilliant code with less testing and validation. This perspective also significantly improves our ability to evaluate the quality of test cases and the testing process itself. SmartQA is not merely about execution; it is about empowering better vision, enhancing sensitivity, and achieving excellence.



Continuously see and assess product from multiple views - USERS, ATTRIBUTES, ENVIRONMENT, CODE, ENTITIES

SMARTQA WISDOM ON SMART ASSURANCE

THE DOING ASPECTS FOR SMART ASSURANCE

(from 5 more thoughts for SmartQA)

What does it take to do SmartQA? Thoughtful pause, multidimensional thinking, sensitivity and awareness and designing for robustness & testability. It is about pausing to speed up, it is about thinking multidimensionally, it is about being sensitive and aware, it is about designing for robustness and testability to doing SmartQA.

Doing SmartQA is about visualising the act in one's mind and taking steps to being robust and enabling rapid and easy validation.

1. Pause to speed up

Before testing, pause and ask: what is being tested, for what purpose, and in what environment? If clarity is lacking, explore rapidly to dig deeper and understand from the user's point of view and the system's architecture or construction. Take a moment to gather thoughts on which issues to target and how to approach them, then proceed to uncover them, leveraging tools as needed. After all, SmartQA is about fostering clarity to visualise potential issues before evaluating rapidly.

Pause to accelerate.

2. View system from multiple angles

A recent conversation with a senior engineering manager in a product development organization highlighted an interesting perspective. The manager, a strong advocate of code coverage, explained that he focuses on achieving close to 100% code coverage as the sole measure of quality and a way to implement shift-left. Makes sense, doesn't it? After all, ensuring that all written code is executed at least once and validated seems both logical and necessary.

But what are the fallacies in this approach?

1. It assumes that all the necessary code has already been written.
2. Non-functional aspects of the code cannot be fully validated.
3. It presumes that the implemented code aligns perfectly with what users actually wanted, even if the code works flawlessly.
4. The number of paths to cover at the highest level of user-oriented validation is overwhelmingly large—nearly impossible to cover entirely.

Code coverage is indeed a necessary condition, but it is far from sufficient. Doing SmartQA demands multidimensional thinking, examining the system from various angles. This includes internal perspectives such as code, architecture, and technology, as well as external ones like behavior, end users, environment, and usage. SmartQA

SMARTQA WISDOM ON SMART ASSURANCE

involves making thoughtful choices about what to validate earlier or later and what to prevent or detect statically.

3. Be sensitive and aware

Be sensitive and aware to issues you encounter and potential causes of issues, after all issues creep in due to untold expectation, accidental omission, quiet assumptions, incorrect implementation, inappropriate modifications, interesting side effects, deliberate abuse, and innovative usage.

Untold expectation	Did not know they wanted these, as they did not communicate it.
Accidental omission	They missed stating it clearly.
Quiet assumptions	Filling in the gaps quietly, without confirming.
Incorrect implementation	Mistakes made during transformation to code.
Inappropriate modifications	Making fixes without fully appreciating the larger context.
Interesting side effects	Innocuously affecting others, without realizing they are coupled.
Deliberate abuse	Wantonly using it incorrectly to push it beyond limits and break it.
Innovative usage	Used in a very different context that was never anticipated.

A heightened sensitivity enables us to question, analyse, clarify and validate. Doing SmartQA is not just finding issues, but sharpening one's senses to be able to smell these and spot them from afar or near before they hit us. It is about elevating QA to be far more valuable to business success.

Sharpen your senses to smell better!

4. Design for robustness

Doing SmartQA is not just about evaluation; it is about enabling the code being built to be robust. It ensures resistance to errors, building firewalls into the code, and guaranteeing that everything required is in good condition before consumption. This means data (inputs) and processes are clean, the operating environment is stable, resources are available, and dependencies are functioning properly.

The goal is self-protection. When faced with irritants, there are three choices:

- (a) reject them and refrain from performing the intended task,
- (b) flag them (log) and still refrain from performing the intended task, or
- (c) intelligently scale down and perform a reduced version of the task.

The key focus is on robustness—remaining unaffected by inputs, configurations, resources, or dependent code. Designing for robustness fosters sensitivity to potential issues and ensures avoidance of being cornered by unexpected challenges.

SMARTQA WISDOM ON SMART ASSURANCE

Don't just test, design system robustly

5. Design for testability

Not only is designing the 'how-to-do' essential, but enabling 'how-to-examine-the-doing' is integral to SmartQA. It involves facilitating the ability to test code easily and quickly identify what went wrong to expedite debugging.

Hooking into code to allow input injection for stimulation and verifying the correctness of outcomes is central to SmartQA. Incorporating traces that can be logged and toggled on during testing or debugging and off in production exemplifies SmartQA. Embedding 'self-test code' represents one of the highest forms of testability.

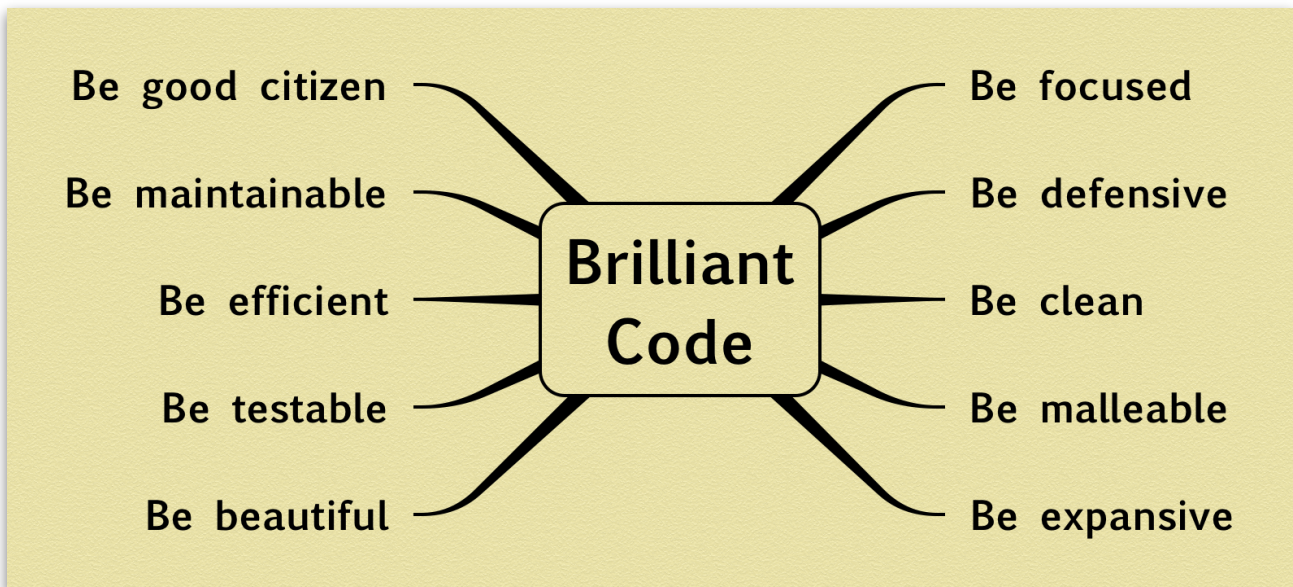
SmartQA is fundamentally about enabling the ability to validate effectively!

Hook in code to aid testability!

CRAFTING BRILLIANT CODE - ASSURING EARLY

(TEN tips for a developer to enable delivery of brilliant code)

Great code is not result of mere unit/dev testing at the early stage. It is really a mindset that is key to producing brilliant code. Here I outline ten things that a developer should be very sensitive to enable the delivery of brilliant code.



1. Be focussed

In each code fragment, do one thing well. Attempting to do many things can become messy! Stay single-minded in what you are solving with this code.

2. Be defensive

Accidents happen, inputs may be tainted. Prepare for eventualities, be defensive in your style of coding. After all, it is your responsibility to stay safe!

3. Be clean

Reduce clutter in code, it is just about somehow getting the code to do work. Organisation, structure matters.

4. Be malleable

Avoid magic numbers, hardcoding. Be soft and pliable so that you can modify, extend easily.

5. Be expansive

Strive to understand the larger context where your code will be used, so that your code delivers value. 'See the forest for the trees' too.

SMARTQA WISDOM ON SMART ASSURANCE

6. Be a good citizen

Respect the environment in which the code runs, consume resources only what you need and release them when you need don't them.

7. Be maintainable

Code begs changes to be done the minute you execute it. Make it maintenance friendly. Also, remember that it may just be you who will maintain the code. Well with time, one also forgets why some things are, the way it is.

8. Be efficient

It is not just about the functionality, it is about all other '-ities', like security, reliability, compatibility, performance etc. Be sensitive to these aspects while you are coding.

9. Be testable

The ability to ascertain if the code is behaving correctly is paramount. Putting it hooks to enable testability enables you to write great code.

10. Be beautiful

Finally great code is not just text that when executed, works correctly. Treat it as a work of art that you produce. Let the choice of names, the structure, the organisation have a sense of aesthetic appeal. After all brilliant engineering becomes art.

SMARTS IN THE TEST LIFECYCLE

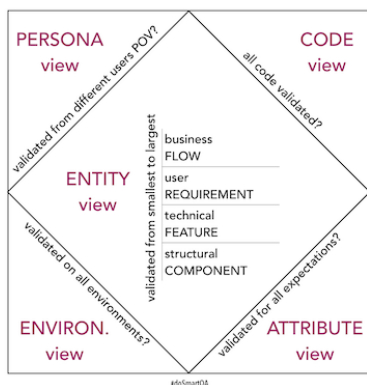
(7 Pictures to 'Doing SmartQA')

SmartQA Thinking

Yesterday a good friend of mine told me about his recent conversation with a senior engineering manager in a product dev org. The Sr Engr Manager, a great believer in code coverage told him that he just focuses on covering close to 100% code as the only measure to ensure quality, and as a means to implement shift-left. Absolutely right, isn't it? After all, ensuring all code written is at least executed once & validated is logical and necessary.

What is/are the fallacy in this? (1) Well you have assumed that all code needed has been written (2) Well, non-functional aspects of code cannot be completely validated (3) Well, it assuming that this is what users really wanted, even if code is working flawlessly (4) Well, the number of paths to cover at the highest level of user oriented validation is just too many to cover, next to impossible! Code coverage is a necessary condition but simply not sufficient.

Doing SmartQA requires multidimensional thinking, of looking of the system from various angles both internal in terms of code, architecture and technology and external in terms of behaviour, end users, environment & usage and then making appropriate choices of what to validate later or earlier and what to prevent or detect statically.



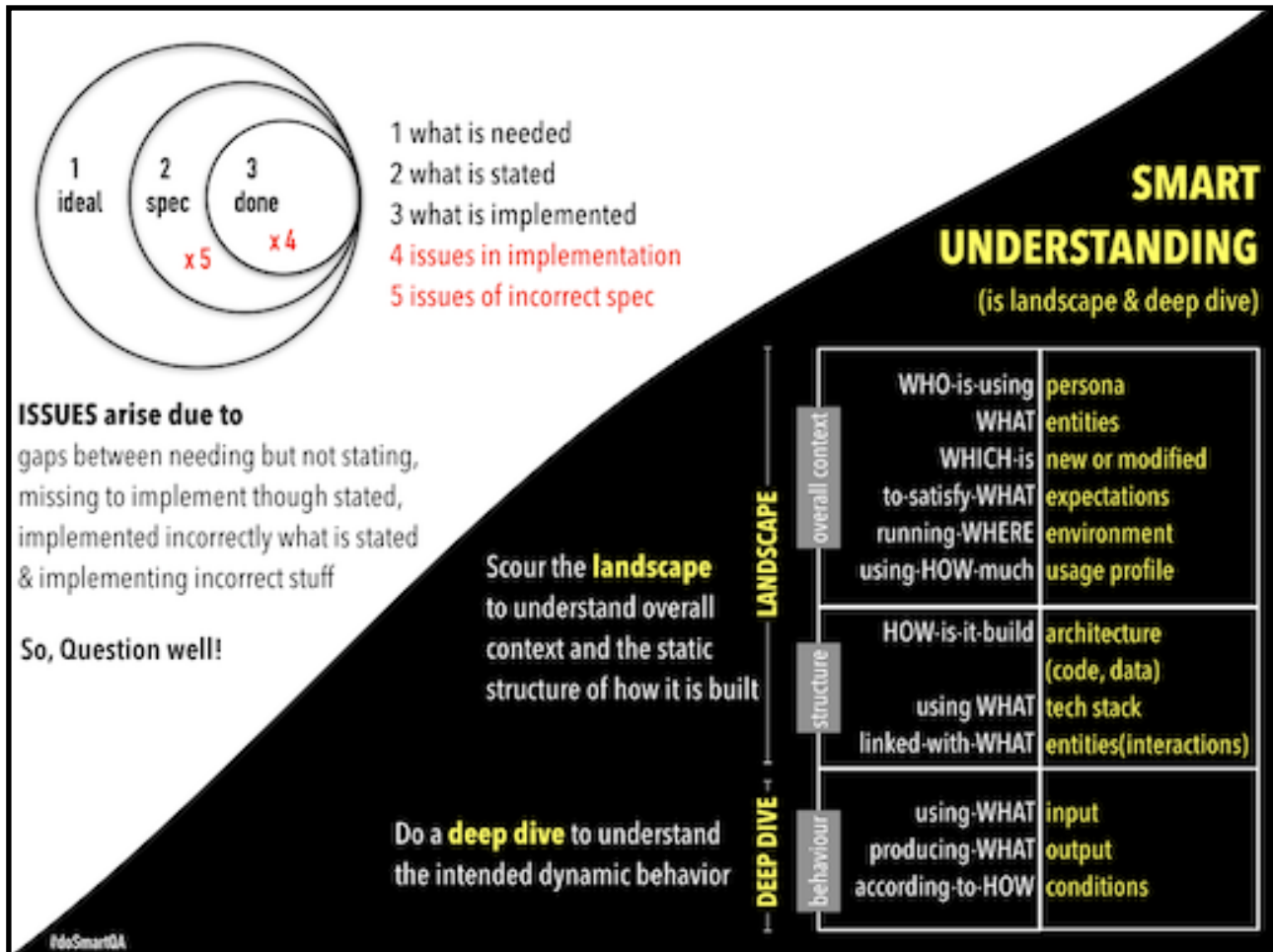
Smart Understanding

Prevention arises from good understanding. Detection, too, depends on good understanding—understanding what is needed, what is stated, and what is implemented. Doing SmartQA demands mental clarity to visualize what is intended, what is present, and what may be missing but could enhance the experience. This clarity drives the ability to question effectively, build better, and both prevent and detect issues. Testing,

SMARTQA WISDOM ON SMART ASSURANCE

at its core, is about discovering what should be there but is not, what is there but is incorrect, and what is present but should not be. Ultimately, it is about understanding the impact of changes, whether within the system or external to it.

Smart understanding involves scouring the "landscape" to grasp the overall context and static structure of how something is built, followed by a "deep dive" to comprehend its intended dynamic behavior. Landscape and deep-dive approaches serve as powerful mental tools for rapidly exploring a system and achieving SmartQA.



Smart Design I

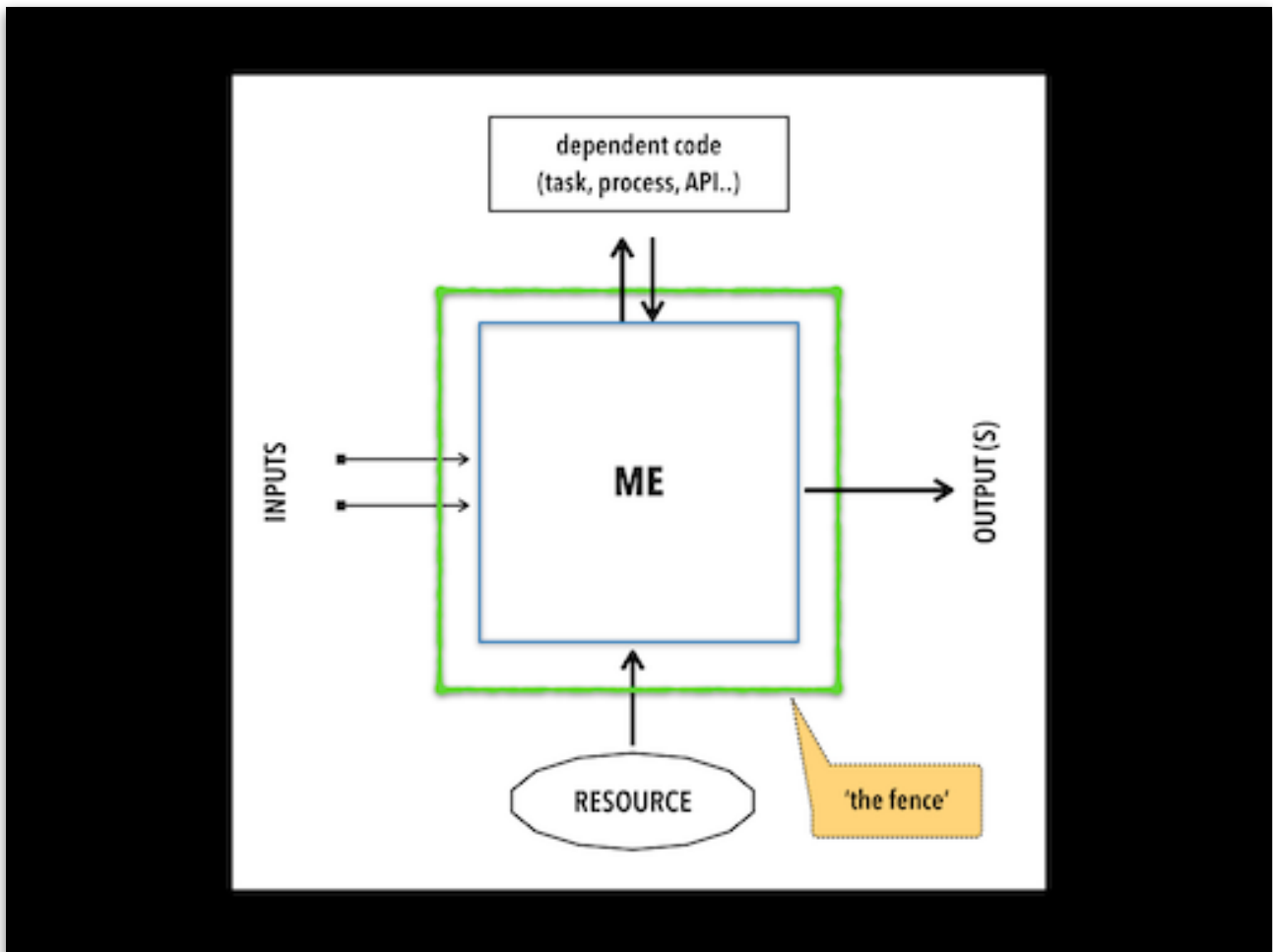
Doing SmartQA is not merely about evaluation; it is about enabling code to be robust. It involves building resilience to prevent errors from creeping in and coding-in safeguards. The aim is to ensure that everything needed is in good condition before it is consumed. This means ensuring clean data (inputs), clean processes, a stable operating environment, available resources, and functioning dependencies.

SMARTQA WISDOM ON SMART ASSURANCE

The focus is on self-protection. How should irritants be handled when they arise? There are three choices:

- (a) Reject them and refrain from performing the intended task.
- (b) Flag them (log) and refrain from performing the intended task.
- (c) Intelligently scale down and perform a reduced version of the task.

The key is to remain robust—unaffected by inputs, configurations, resources, or dependent code. Designing for robustness cultivates sensitivity to potential issues and ensures preparedness to avoid being cornered by unforeseen challenges.



SmartQA Design II

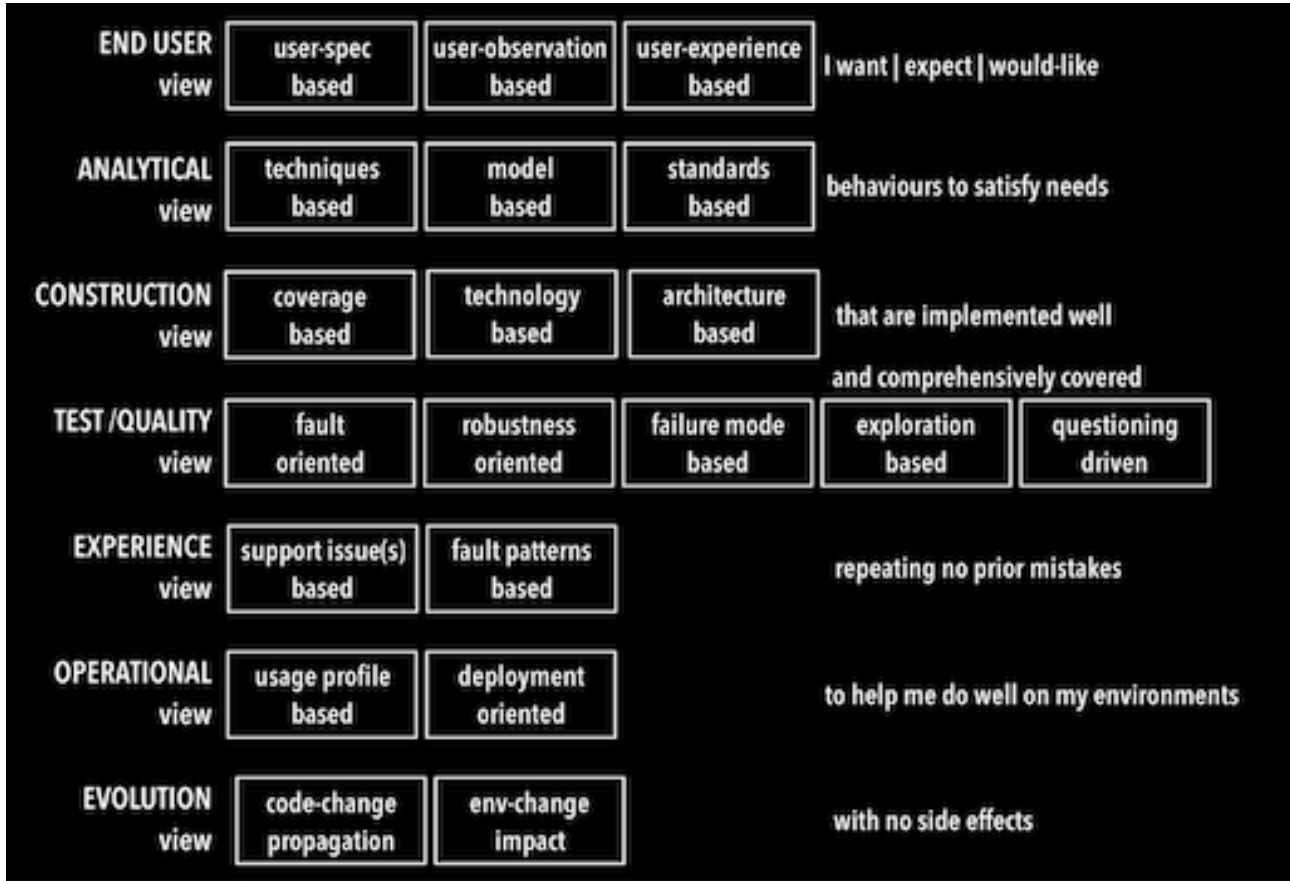
Test design deserves attention. The focus often leans heavily toward execution, emphasizing the ability to cover more. This has shifted priorities toward how frequently tests can be executed, driving an emphasis on automation. It's worth stepping back to revisit the primary objective: delivering clean code. Achieving this requires a deeper sensitivity to the quality of tests, where design plays a critical role.

So, what is a smart approach to designing good scenarios? Ideally, it involves creating a few scenarios that can uncover the most significant issues. Smart design entails examining the system from multiple perspectives to ensure:

SMARTQA WISDOM ON SMART ASSURANCE

"I want | expect | would-like behaviours to satisfy needs that are implemented well, comprehensively covered, and capable of performing effectively in my environment without side effects."

From there, decide what to prevent, detect statically, or test—whether through human effort or machines. Focus on the intent first, then on the activity.



SMART DESIGN *"I want | expect | would-like behaviours to satisfy needs that are implemented well and comprehensively covered to help me do well on my environments with no side effects"*

Smart Validation

The approach to software validation has traditionally been categorized as either 'manual' or 'automated.' However, a better distinction would be 'human-powered' and 'machine-assisted.' When choosing to validate without automation, what are some smart ways of doing it?

There are four key approaches:

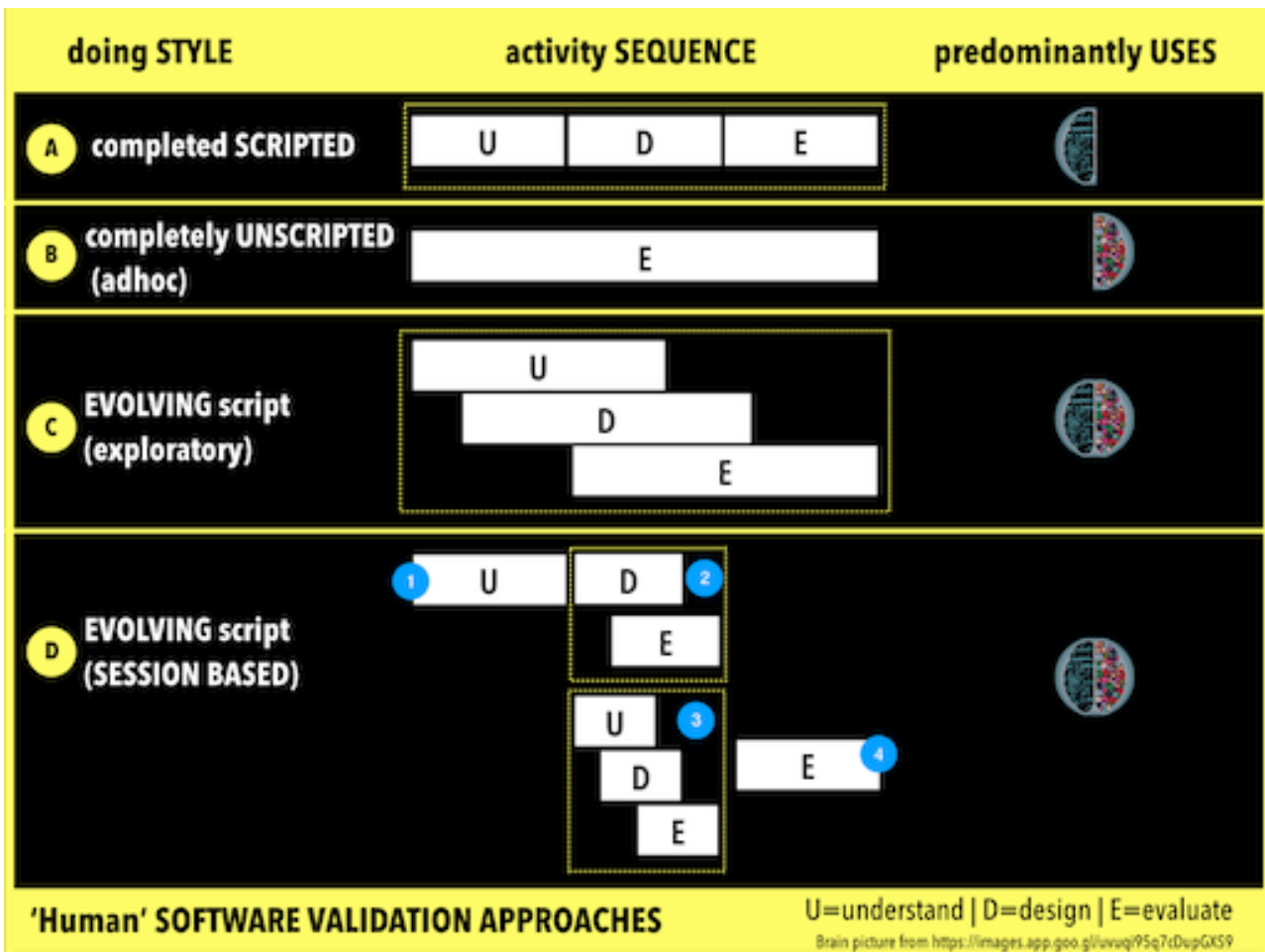
A	Completely scripted	Fully understanding, designing, and then evaluating. Ideal when specifications are clear and complete, relying heavily on logical, left-brain thinking.
B	Completely unscripted	Used in time-crunch scenarios or by casual testers. Involves jumping straight into evaluation, guided by creativity, experience, and sometimes luck.

SMARTQA WISDOM ON SMART ASSURANCE

C	Evolving script	An exploratory approach where understanding, design, and execution happen concurrently, using a balanced mix of left and right brain thinking.
D	Evolving script in sessions	A clear objective is set for each session, focusing on specific aspects like 'only understanding,' 'understanding and designing,' or the full process. This approach leverages both left and right brain thinking for maximum efficiency.

The choice of approach depends on the context. Approach A might be suitable when specifications are clear, while C or D work better for evolving systems or when specifications are high-level. Approach B, though less structured, complements A, C, and D as quick checks or to harness the power of randomness.

HyBIST (Hypothesis Based Immersive Session Testing) builds upon these approaches by encouraging immersion in the act of evaluation, fostering a state of 'flow.' This session-based method employs a suite of thinking tools to understand, design, and evaluate with the primary objective of 'doing less and accomplishing more.'



SMARTQA WISDOM ON SMART ASSURANCE

Smart Plan

A smart plan is simple and objective-based: identify what entity to test, what issue to test for (by conducting which test), in what environment, by whom (in case of a team), from whose point of view, and how to test (human-powered or automated). A concise plan encompassing these aspects enables sharp focus and rapid evaluation, whether through human efforts or by building nimble automated suites.

At a higher level, determining what kind of issues to uncover at different levels of entities in specific environments constitutes a strategy. Meanwhile, identifying specific issues for specific entities forms a plan. This approach facilitates designing two types of scenarios:

1. Objective Scenario - Focused on outputs, these are simple one-liners outlining what issues to uncover for which entity in a specific environment.
2. Executable Scenario - Focused on inputs, these are detailed sets of inputs used to stimulate an entity to uncover specific issues in a given environment.

SMART PLAN is being objective based: *what-to-test, for-what, where, by-whom, from whose-POV and how-to-go-about-it*

WHAT (EUT)	WHAT-FOR	WHERE	WHO	WHOSE POV	HOW
business FLOW user REQUIREMENT technical FEATURE structural COMPONENT	what issues to look for to satisfy attributes, collection of similar issues form a test type	on environment a combination HW,SW a mix of Frontend, Backend, Middleware	to be validated by whom, especially useful in large team setup	from which end user point of view, user may be a human or another system	how to evaluate? human powered or machine assisted?

Testing certain EUTs for given ATTRIBUTE(s) on ENVIRONMENT(s)
=> STRATEGY

Testing an EUT(X) performing TEST(Y) on ENVIRONMENT(Z) by PERSON(N)
=> PLAN

Testing an EUT(X) for ISSUE(Y) on ENVIRONMENT(Z) by PERSON(N) from POV(P)
=> an objectiveSCENARIO

Testing an EUT(X1) performing TEST(Y) with INPUTs(I1,I2) on ENVIRONMENT(Z1)
=> an executableSCENARIO

Testing an EUT(X1) for ISSUE(Y1) on ENVIRONMENT(Z1) by PERSON(N1) from POV(P1)
Testing an EUT(X2) for ISSUE(Y1) on ENVIRONMENT(Z1) by PERSON(N1) from POV(P1) ..
=> a TESTSUITE

Testing an EUT(X1) performing TEST(Y) with INPUTs(I1,I2) on ENVIRONMENT(Z1)
Testing an EUT(X2) performing TEST(Y) with INPUTs(I3,I4) on ENVIRONMENT(Z1)
=> an automatedTESTSUITE

SMARTQA WISDOM ON SMART ASSURANCE

TEAM STRUCTURE FOR SMART ASSURANCE

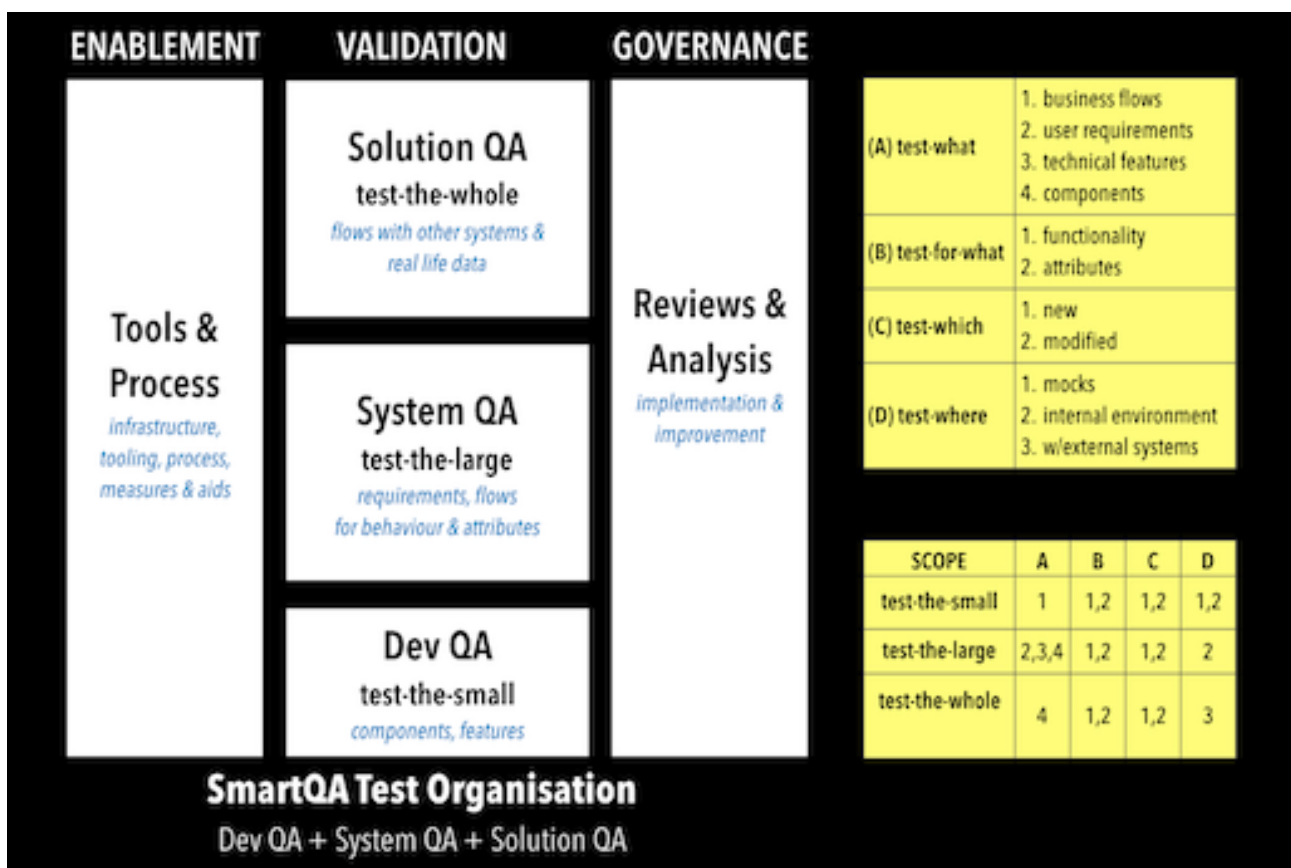
Software engineers once developed code and tested it themselves. Over time, dedicated QA teams became the norm, with testing being "owned" by these teams. In modern times, driven by Agile and rapid development, this is merging back into development, with dedicated QA teams becoming thinner.

An article in SD Times discussed Who owns QA? Is it the development organization or a dedicated QA organization? What is the right fit?

What "dedicated QA" really means is that there is a focus on QA and testing. In a software organization, specialists are necessary—whether they are analysts, architects, developers, or testers. Dedicated QA doesn't mean solely reporting into a QA leadership role; it simply means having QA specialists with deep expertise in systems validation.

So, what might be the right fit for building a Smart QA organization? QA can be seen as a blend of Dev QA, System QA, and Solution QA, each with a unique objective in validation.

Beyond validation, a dedicated QA (System/Solution QA) is well-suited to provide enablement and governance. This includes tasks like setting up test infrastructure, creating tooling frameworks, establishing processes, defining metrics, publishing aids, conducting reviews, and improving the overall system.



ENHANCE PRODUCTIVITY

(39 tips to being productive - Do SmartQA)

Software tools help in increasing productivity by allowing us to do faster, cheaper and better. But the most powerful tool “the human intellect” can help us do lesser and coupled with tools of speed, productivity scales geometrically. In these times of AI, it is necessary to exploit HI (Human Intelligence) to do stuff beyond intelligent machines to deliver a higher value. This article outlines 39 tips in thirteen sections to “Do SmartQA”.

“Productivity is a measure of your output divided by your input. Output is measured by the importance of the accomplishment to your goals” says Scott H. Young in his article What’s the Point of Productivity?. He also adds that people obsessed with productivity often neglect the hard-to-quantify-but-ultimately-essential work that goes into achievement, being driven by measurements on short-term output over things that matter, but are harder to measure.

“Working longer hours is not a surefire way to get more stuff done. In fact, consistently working too much invariably leads to a drop in productivity” says Karen Baner in 6 Habits of Insanely Productive People. Similarly, on a different note “Constantly re-running automated tests helps us know that we are still good, but in the same place”.

“Productivity doesn’t mean doing the most, but getting the most from what you have done” says Scott H. Young in another article 7 Habits that Seem Lazy (But Actually Let You Get More Done).

In an era where we are obsessed with productivity, it is not about doing more, of being busy that is deemed as high productivity. In fact it is the converse, of being smart, of doing less and accomplishing more. Software tools help in increasing productivity by allowing us to do faster, better and cheaper. But the most powerful tool “the human intellect” can help us do lesser, coupled with tools of speed, productivity scales geometrically.

In these times of AI, it is necessary to exploit HI (Human Intelligence) to do stuff beyond the intelligent machines and deliver higher value. Here are THIRTY NINE tips arranged in THIRTEEN sections to “Do SmartQA”.

See the big picture, before you jump in to test

Tip #1	Start with the WHO first, it may be a person or machine.
Tip #2	See features from a product angle, requirements from a user angle, and flows from a business angle.
Tip #3	Guess attributes that may be expected and then refine.

Map who uses what, to understand user needs

Tip #4	Put yourself in the user’s shoes and ask what you need.
--------	---

SMARTQA WISDOM ON SMART ASSURANCE

Tip #5	As you map requirements to a user, ask what attributes you expect.
Tip #6	In an enterprise system, there may be many end users; in a personal system, a person may play multiple roles.

Map attributes to entities, to identify expectations

Tip #7	If they are clearly spelled out, map them.
Tip #8	Put yourself in the user's shoes and think of attributes they might expect.
Tip #9	Look at similar/competitive products to identify attributes.

Create environments by combining elements & values

Tip #10	Elements could be OS, browsers, devices, network, DB, etc.
Tip #11	Values of elements could be versions of the same or different similar types.
Tip #12	Create environments that are largely representative of real-life deployments.

Link related entities via associations & compositions

Tip #13	Think of association as "both affecting the same data."
Tip #14	Think of composition as "output of one is the input to another."
Tip #15	Being part of a sequence of activities is also composition.

Do a simple focused plan for every test session

Tip #16	A plan includes 'what-to-test,' 'what-to-test-for,' and possibly 'where-to-test-on.'
Tip #17	If deferring the environment to test on later, do so.
Tip #18	Let the list of tasks flow and assign them a session later.

Dive deep to understand an entity well

Tip #19	Describe the entities tersely to understand better.
Tip #20	Identify conditions that define functional behavior.
Tip #21	Appreciate who uses this, what they expect, and how it interacts with other entities.

Rapidly design scenarios and refine

Tip #22	Set objectives in terms of acceptance criteria and potential issues.
---------	--

SMARTQA WISDOM ON SMART ASSURANCE

Tip #23	Acceptance criteria are what you expect; potential issues are what you don't expect.
Tip #24	Combine conditions judiciously to create a good mix of positive and negative scenarios.

While you execute tests, observe & refine

Tip #25	While executing, look for interesting issues to enhance scenarios.
Tip #26	Empathize—put yourself in the shoes of end users to enhance scenarios.
Tip #27	Look for issues, but also identify opportunities to improve the user experience and make suggestions.

Organise scenarios well to be effective

Tip #28	Ensure a good distribution of scenarios across all quality levels & test types.
Tip #29	Ensure scenarios cover various users (personas).
Tip #30	Create suites tailored to specific business objectives.

Analyze test scenarios for effectiveness

Tip #31	Ensure scenarios test small features and big flows.
Tip #32	Include scenarios across all quality levels and test types.
Tip #33	Maintain a good mix of positive and negative scenarios.

Analyze execution progress constantly

Tip #34	Check if progress is being made, evaluating features to flows.
Tip #35	Ensure attributes are being tested in addition to functionality.
Tip #36	Confirm all entities are being tested as part of the progress.

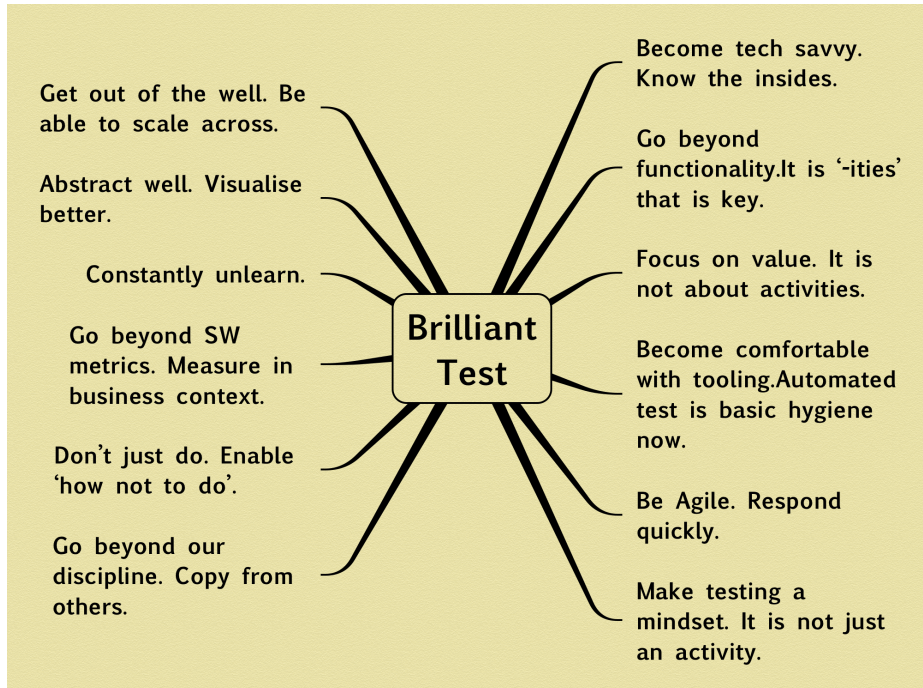
Analyze product quality at end of every session

Tip #37	Check for functional stability, covering features, requirements, and flows.
Tip #38	Assess robustness—ensure negative scenarios run clean.
Tip #39	Verify the product meets expected attributes, in addition to functionality.

TRANSFORMING INTO A SMART TESTER - TIPS

(TWELVE tips to become a modern smart tester)

The way we build systems has changed, both in terms of technology and the process. The expectations of end users/businesses have changed in terms of speed of delivery and in terms of expectations.



1. Become tech savvy. Know the insides.

Understand what happens behind the hood. Know what your system is composed of. Learn to think of issues resulting from integration of various technologies, of different systems that make your solution.

2. It is '-ities' that is key. Go beyond functionality.

Yes, correctness of functionality is important. But in these times, it is '-ities' that are key to success. Well we know for sure how usability has become mainstream. We also know 'compatibility' is critical especially device compatibility of mobiles/tablets. Performance, security, error recovery is now a given. So it is necessary to become adept in evaluating '-ities' too.

3. Focus on value. It is not about activities.

What matters now is not how-many, it is really how-valuable. End users are keen on the value-offering i.e. how does it help me do better, how does it ease my life..?

4. Automated test is basic hygiene now. Become comfortable with tooling.

Well it is expected that you exploit technology/tools to accelerate what you do and replace what you do. So being comfortable with tools and rapidly able to exploit other

SMARTQA WISDOM ON SMART ASSURANCE

tools/languages to getting things done is expected. Tooling is no more an esoteric skill. Remember it is not about 'big' tools, it is about also having a nice 'SwissKnife' tool set to enable you to do faster/better/smarter.

5. Be Agile. Respond quickly.

It is no more about days, it is about hours. Change your mental model to test in short sessions, change your mental model to re-test far more efficiently, change your mental model to focus on impact far sharper.

6. Test is no more just an activity. Make it a mindset.

As we morph to deliver clear code faster, it may not always be an explicit activity. It is about having a 'test/perfection mindset' so that we built/craft code quicker and cleaner.

7. Go beyond our discipline. Copy from others.

Stay sharp and wide open to see how great quality/perfection happens in other disciplines. Unabashedly copy and adapt. It is necessary to be non-linear. Be inspired from lateral disciplines and humanities/social, nature, arts etc. to evaluate, to prevent, to build better.

8. Don't just do. Enable 'how not to do'.

It is not just about evaluation anymore, it is about how we can prevent evaluation. Enable building robust code. Enable better sensitisation of issues early. Do more 'what-if' to build better code.

9. Go beyond software metrics. Measure in business context.

It is great to use measures of testing to guide the act of testing. Given that we are in the age of speed and instant gratification, it is very necessary to relate the software measures to business & end user context to ensure success. For example (1) it is no more just a performance metric, it is about how (say) response time affects the business(end-users) positively (2) it is not about overall coverage alone, but about what it means to the risk of the immediate releases.

10. Constantly unlearn.

Unlearning is a skill. The ability to constantly question if what we know is relevant and drop it to make way for newer skills is paramount.

11. Abstract well. Visualise better.

Today the act of building systems is brilliant with excellent abstractions facilitated by frameworks. The focus is on great clarity and the ability to reassemble/morph quickly, much like 'Lego' bricks. The same is applicable for us test folks too. Abstract well (1) the

SMARTQA WISDOM ON SMART ASSURANCE

system and how it is composed (2) the issues you are going after and therefore the strategy (3) test assets to facilitate continual adjustment (4) automated suites so that you can flex it to suit the changing needs (5) test data so that it can be relevant for a longer time.

12. Get out of the well. Be able to scale across.

What we do now is no more a silo related to evaluation. It is imperative to build/tweak code, setup environments, deploy, assist in debug, help ideate features to improve value, in addition to testing. Be able to do 'everything' to scale across.



"We are SmartQA evangelists. For over two decades we have transformed how individuals, teams and organisations have practised testing. We espouse methodology to test intelligently. Our mission - Elevate to high performance via SmartQA."
www.stagsoftware.com



The HyBIST Approach to SmartQA - MASTERCLASS

Testing is deep probing to seek clarity and in the process uncover, preempt issues rapidly. The HyBIST approach enables designing smart probes and probing the system smartly.

<https://smartqa.academy/courses/smartqa-using-hybist>



doSmartQA - AI based Smart Probing Assistant to interrogate, hypothesise issues, design & evaluate user story or a set of stories in a sprint rapidly. An assistant for smart session-based testing based on HyBIST.

Download personal edition from [here](#)



SmartQA Musings - A gentle flurry of interesting thoughts on smart assurance as a weekly webcast. A refreshing view of assurance to broaden & deepen thoughts/actions.

www.stagsoftware.com/subscribe



SmartQA Biweekly - Ignite your curiosity with fresh insights, thought-provoking ideas, and inspiring content delivered straight to your inbox every fortnight.

www.stagsoftware.com/subscribe

A rich collection of original content on smart assurance

SmartQA Wisdom - Profound nuggets of wisdom to think deeply, do rapidly & smartly for Test Practitioners.



on Personal Growth
on Test Design
on Smart Understanding
on Mindset & Habits
on Problem Solving

Download from www.stagsoftware.com/smartqa-wisdom

SmartQA eBooks - for Sr Engg/QA managers, Sr Test Practitioner & Young Test Practitioners too.

HyBIST at a glance

do SmartQA -The HyBIST Approach

High Performance QA

50 Tips for SmartQA

Communicate Clearly



Download from www.stagsoftware.com/smartqa-ebooks