

PRACTICAL IMPLEMENTATION OF DEVELOPMENT CODE QUALITY

T Ashok
CEO, STAG Software
[linkedin.com/in/ashokstag](https://www.linkedin.com/in/ashokstag)

© STAG Software Private Limited 2018
stagsoftware.com

TABLE OF CONTENTS

The problem at large	3
Business/economic impact	3
Mistaken notions of WB and BB	3
Unit testing? Dev testing is a better term	4
“Quality Levels” - Defect types matter	4
Dev testing - What is the objective?	5
So, what levels matter to the developer?	5
Is your dev test effective?	5
So, what should be really happening in Agile context?	5
It is not about doing more, it is about doing less	5
Practical implementation of great dev quality	6
Hmm.. do we really need unit (dev) tests?	6
The power of checklists in other disciplines	6
Checklists are not mindless compliance	6
Smart checklists make you think	6
We use checklists too, but are we effective?	7
Smart checklist enables developing clean code	7
The “Test Pyramid”	7
How to make dev tests effective	8
Survey results and my experience	8
“The flow”	9
References	10
Appendix : Smart DevChecklist	11

The problem at large

The quality of early stage code is felt to be a concern by many engineering managers that I talk to. In my numerous consulting assignments, I have noticed that many issues found by QA folks seem to be ones that do not require the expertise of a specialist tester, compromising the effectiveness of 'system test' resulting in avoidable customer issues.

Great quality code is not the result of intense system testing, it is result of well structured filtration of issues from the early stages. A compromised 'unit test' puts unnecessary strain on the QA folks who seem to be compelled to go after these issues at the expense of system test.

Developers on the other hand do not deliberately write bad code, it is just that accidents happen. Accidents seem to happen due to brute force push of unit testing without it being simple and practical, and developers already short of time are unable to adhere to a heavyweight process. The other fallacy seems to be the over dependence on automated unit tests as the saviour without paying attention to test cases. Also the incorrect notion of unit testing as being only white-box oriented with a skew towards code coverage results in ineffective tests that are introverted. Lastly the sheer emphasis of dynamic testing as the only method to uncover defects is possibly overwhelming, when easier static methods of uncovering issues could have been employed.

Business/economic impact

The impact of leakage of issues from early stage is not irritating, but serious enough. Issues reported by customers that are early stage simple issue like poor validation of inputs, results in significant drop of confidence in the customer. The QA folks focus on these issues result in their job of system validation being poor, resulting in field issues related to end-to-end flows and sometimes attributes being compromised.

Also the incorrect focus of a specialist QA results in insufficient time for doing things that can make system test more effective and efficient like automation of end-to-end flows, focus on non-functional requirements, revising the test strategy/approach and sharpening with knowledge gained every cycle.

Mistaken notions of WB and BB

Developers state that they do white box testing while specialist QA folks claim that they do black box testing. Hmm, are they they logical at all? Let us examine the incorrectness of the statement.

Black and white are techniques for examining a system under test for potential issues. The former is about identifying governing conditions of intended behaviour, and then generate these scenarios so as to inject the system with a variety of inputs to assess correctness. On the contrary, the latter is about understanding the internal structure of the system-under-test (SUT) and then perturbing the structure to assess how they can result in deviant behaviour.

So would you do ONLY external or internal examination? or Would you meaningfully adopt both? Also, what is the meaning of structure in the small vs large system-under-test?

Unit testing? Dev testing is a better term

The definition of what an unit is most often unclear and therefore unit testing more misunderstood. Martin Fowler in his blog Unit Test [2] states "It's very ill-defined, and I see confusion can often occur when people think that it's more tightly defined than it actually is".

So let us use the term "Dev test" to state the early validation by developers during the development of code. What is expected of the code quality from developers? What should dev test uncover? To setup a clear objective of what dev test should accomplish (objective), let us take a goal focused approach of as to what types of defects should be uncovered at dev test.

"Quality Levels" - Defect types matter

Purposeful testing is about hypothesising potential type of defects and going after them. Of course, as we one engages in this activity in the scientific manner, we revise and fine tune what to go after and how to go after.

Characterising the system under test as containing a "mixture" of various defect types, and inspired by fractional distillation as an efficient method of separation, Hypothesis Based Testing (HBT) [1] sets up NINE quality levels of which the first FOUR are certain candidates for dev testing.

So what are the quality levels and what is the purpose of tests at those levels?

QUALITY LEVELS	L9 User expectations
	L8 Deployment correctness
	L7 Attribute correctness
	L6 Environment correctness
	L5 Flow correctness
	L4 Behaviour correctness
	L3 Structural correctness
	L2 Interface correctness
	L1 Input correctness

- L1: Ensure bad inputs are rejected, about data types, values, boundaries
- L2: Ensure that input interface is clean, about syntax(format), order, messages
- L3: Ensure that internal structure is robust, about exception, resource handling, timing/sync, loops/iteration
- L4: Ensure behaviour of feature is fine, about the business logic of the base technical feature
- L5: Ensure behaviour of requirement/flow is fine, about the business logic of the requirement/flow that is a collection of technical features
- L6: Ensure system works well is all different external environments and does not affect the environment
- L7: Ensure that key requirement/flow satisfies the expected attributes (e.g performance, volume...)
- L8: Ensure that final system deploys well in the real environment. Installation, configuration, migration are of interest here.

Dev testing - What is the objective?

The objective of dev testing is that the building blocks of a system i.e. structural components are indeed clean enough to deliver the basic feature and worthy of integration to form the larger system. This means that a building block is internally robust (structurally clean) and behaviourally correct (externally clean).

So, what levels matter to the developer?

Quality levels denote a progression of quality as software is built. So as a developer, it is necessary to validate that basic behaviour of the building blocks that we build/modify. This means that a developer performs tests to ensure Levels 1-4 are satisfied, implying that the component is clean enough to integrate into the larger system. Additionally, if the component is critical to certain system attributes, then higher level tests beyond L5 L5 could be useful to do.

Is your dev test effective?

Well, if the defects found by the specialised QA do not belong to the earlier quality levels (L1-L4), then dev tests are have been done well. In the case of non-trivial number of L1-L4 issues found by the QA, it implies that dev test may not be good enough as defects are leaking to the product test stage done by the QA.

So, what should be really happening in Agile context?

Lean thinking is what inspired the Agile movement. Lean is about not producing waste in the first place. It is doing things 'clean' in the first place, so that waste is ideally not there. Waste in software context are bugs. And in the early stages there are 'unit bugs'. Since our focus in Agile is to find these earlier and to ensure that they are never there whenever we modify, we resort to a high degree of automation. Therefore we have a large body of test cases at the lower levels automated to ensure that we can continually execute them. This is great, but should we not focus on adopting a practice that in essence prevents these issues and lessen the need for large number of unit tests to uncover these?

It is not about doing more, it is about doing less

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. Yes, that seems logical. But what a minute, for a developer already busy writing code, is this the right approach? Given that dev test is largely about issues in L1 thru L4, could we not focus on getting this right or statically assess these via smart checklist?

Great quality early stage code is not about doing more testing, it really is doing about doing less test, by enabling sharper focus on 'what-can-go-wrong', 'have-you-considered-this'.

Practical implementation of great dev quality

Segregating the defect finding into levels, it is possible to bring sharp focus into L1~L3 and sensitisation to enable reuse type of defects to be prevent or detected early statically without doing too much work.

Given that structural components that deliver technical features to be reasonably small, it may be possible to validate behavior without resorting to only testing.

Hmm.. do we really need unit (dev) tests?

Yes, dev tests are key to delivering clean structural components, it is just that they can be done much more simply and efficiently. Efficiently, not only via automation but also by resorting to smart checklists that sensitise you to prevent or check rapidly, not mindlessly make you tick and comply.

The power of checklists in other disciplines

In other mature disciplines like medicine, aviation, construction where the impact of simple defects is enormous, it is smart checklists that have come to the rescue and saved millions of dollars and saved many lives.

The Checklist Manifesto [4] by Atul Gawande, a surgeon, extols the power of checklist. He states that any problem can be categorised into simple, complex and complicated, and how the smart checklists can solve these.

'Simple' problems are those that are individualistic in nature with a set of stuff to be done, while 'Complicated' problems implies multiple teams/people coordination/timing issues, and 'Complex' problem is where outcomes are different despite same application.

Checklists are not mindless compliance

Smart checklists are not about mindless compliance, not about ticking boxes, it is really about tickling the brain to think better and ensure fault-proofing rapidly. In our industry, checklists have been seen a cheap brainless activity that is about ticking the checkboxes, and therefore presumed to be useless.

Smart checklists make you think

To solve complex problems, "push the power of decision making from a central authority to the periphery, allowing people to make decision and take responsibility" (Atul Gawande).

The checklist needs to allow for judgement to be used in the tasks rather than enforce compliance, so that actions may be taken responsibly. Hence, a smart checklist is about enabling your thinking process by having:

- Set of checks to ensure stupid but critical stuff is not overlooked
- Set of checks to ensure coordination
- To enable responsible actions to be taken without having to ask for authority.

Good checklists are precise and easy to use in most difficult situations, it does not spell out everything, but provides reminders of critical & important steps that even highly skilled professionals would miss.

Checklists seem to defend everyone, a kind cognitive net designed to catch flaws of memory and attention, a well designed checklist enables one to DO well, SYNC with others well, and take appropriate ACTions as needed. [5]

"The power of checklists is limited, they help experts remember how to manage a complex process or machine, make priorities clearer and prompt people to function well as a team. By themselves, however, checklists cannot make anyone follow them." (Boorman, Boeing) [4]

We use checklists too, but are we effective?

Do we use checklists for early tests like DevTest? Yes, we do. But from what I have seen of numerous checklists like code review/UI checklists , it is more often is used like a compliance assessment, of ticking away a long list of items-to-check for. So this turns to be a mindless job and suffers poor implementation, as it is most ill suited for smart validation.

Smart checklist enables developing clean code

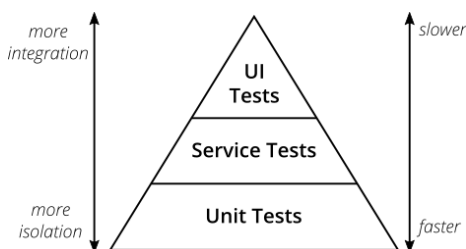
So should we really do dev test? Should we not become sensitive and write cleaner code? Well lean thinking (aka Agile) is of producing less bugs in the first place, not about testing more.

A 'Smart DevChecklist' enables one to precisely accomplish this, to become more sensitive and written code that certainly does L1 through L4 issues. Remember, this is the most cost effective method to good quality. Well, a Smart DevChecklist is an efficient complement to dev test enabling an easy and efficient method to producing great code.

The "Test Pyramid"

'The Test Pyramid[3] is a metaphor that tells us to group software tests into buckets of different granularity. It also gives an idea of how many tests we should have in each of these groups'.

The intention of Test Pyramid is to focus on automation of low level tests that validate individual components, ones that may be done easily via non-UI automation.



As much this is very meaningful, choose the right sized component to make it worth the effort. You should not unnecessarily automate what can be efficiently accomplished by a 'Smart DevChecklist'.

Remember that it is not blind automation that finds defects, it is smart tests (cases really) that do.

How to make dev tests effective

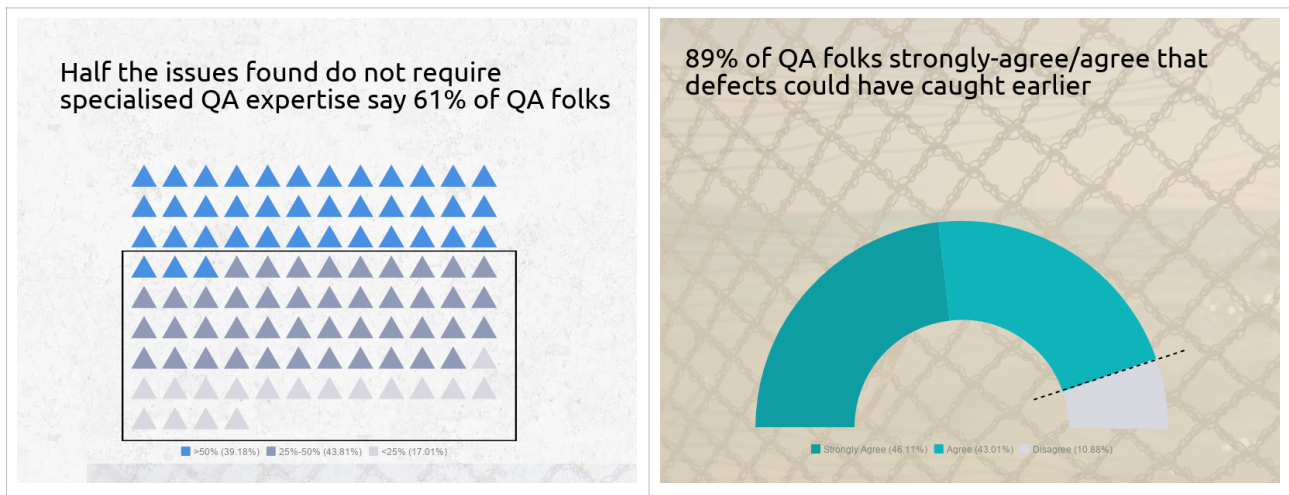
“It is not the number of defects that matter, it is the quality of defects that matter”.

During my consulting, I have analysed defects found by the specialised QA team by ‘bucketising’ them into HBT quality L1 ~L8 and have found it interesting that a non-trivial number falls into the earlier levels L1~L4. What this implies are : (1) Dev tests are not as leaking basic issues to later stages (2) That QA is not focused on higher levels, pulled down by lower level issues and is challenged in terms of meaningful QA.

Survey results and my experience

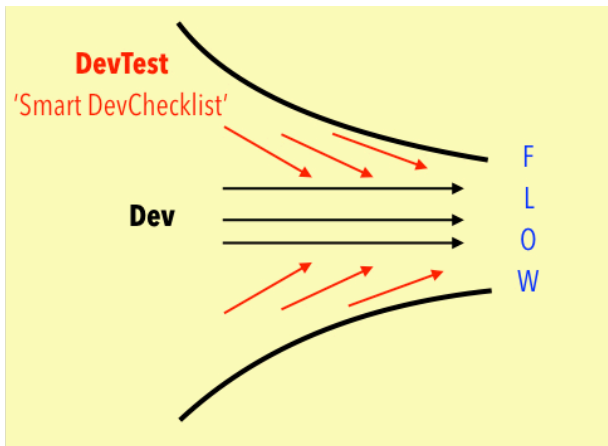
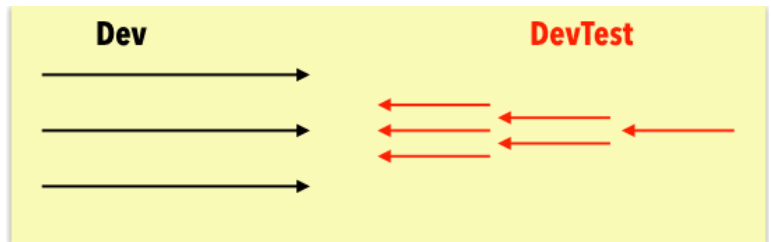
In many companies that I have consulted with, I have noticed that the DevTest (Unit test as they called) is woefully leaky. I have noticed that at last half the issues found by QA team could have been found earlier. This simply means that QA is probably doing others’ job compromising their effectiveness at a higher level.

A few months I conducted a simple survey asking QA folks on their perceived effectiveness of DevTest. The inference from this two question survey is shown below.



“The flow”

When we code, it seems natural to test to ensure it is clean. Here it is, pictured as two sets of arrows facing each other with the dynamic test as a primary means to ferret out the ‘bugs’.



In the world of Agile, with extreme focus on agility and speed, would it make more sense to align in the same direction to be more nimble and speedy?

What if ‘Smart DevChecklist’, is a proactive DevTest is aligned with Dev, where we do not ferret out bugs, rather we sensitise better to constantly de-weed while we code. That is a “Smart DevChecklist” at Dev will enable, to do less and accomplish more.



“The flow” - a state when we harmonise what we do and deliver the best. Flow, the concept expounded beautifully by Mihaly Csikszentmihalyi in classic book “Flow- The psychology of optimal experience” [4] states that the state of consciousness called flow is makes an experience genuinely satisfying. During flow, people typically experience deep enjoyment, creativity, and a total involvement with life.

So when the act of DevTest is well aligned with Dev and not just seen as a task to be completed, clean code can happen joyfully. Smart DevChecklist is one of the enablers of the “Flow”.

A draft of the Smart DevChecklist is listed in the Appendix at the end.

References

1. HBT Central, hbtcentral.org
2. Martin Fowler, Blog on "UnitTest", <https://www.martinfowler.com/bliki/UnitTest.html>, 2009
3. Atul Gawande, "The checklist manifesto- How to get things right", Penguin books, 2009.
4. Ham Vocke, Blog on "The Practical Test Pyramid", <https://martinfowler.com/articles/practical-test-pyramid.html>, 2018
5. Thiruvengadam Ashok, "Design checklists to 'Do, Sync & Act'", <https://www.linkedin.com/pulse/design-checklists-do-sync-act-ashok-t/>

 <p>STAG Software is a specialist test boutique that focuses on methods and tools for smart assurance of systems. Powered by HBT, its scientific test methodology, STAG offers third party product validation services, advisory and consulting solutions to enhance organisation test practices and conducts specialist masterclasses on HBT to enable smarter testing by software folks. www.stagsoftware.com</p>	 <p>T Ashok is the Founder & CEO of STAG Software. Passionate about quality, he is the architect of HBT, a personal scientific test methodology . A strong believer in opposites, 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 & College of Engineering, Guindy. linkedin.com/in/ashokstag</p>
--	---

Appendix : Smart DevChecklist

SMART DEV CHECKLIST

Am I OK ?	Bad INPUTS rejected? SCREEN/UI well done?	DISPLAYS well? All TEXT fine?	All OUTPUTS checked? Are ACTIONS to do fine?	"I am fine"
------------------	--	--	---	--------------------

- | | | | |
|---|--|---|---|
| Inputs ok?
<input type="checkbox"/> Limits
<input type="checkbox"/> Type
<input type="checkbox"/> Defaults | Screen ok?
<input type="checkbox"/> Alignment
<input type="checkbox"/> Consistency
<input type="checkbox"/> Dependencies
<input type="checkbox"/> Colours fine | Display ok?
<input type="checkbox"/> Responsive
<input type="checkbox"/> Orientation
<input type="checkbox"/> Resolution | Text ok?
<input type="checkbox"/> Spelling
<input type="checkbox"/> Grammar
<input type="checkbox"/> Meaningful
<input type="checkbox"/> Actionable |
| Outputs ok?
<input type="checkbox"/> Stored fine
<input type="checkbox"/> No duplicates
<input type="checkbox"/> Appropriate msg | Actions ok?
<input type="checkbox"/> Navigation
<input type="checkbox"/> Default
<input type="checkbox"/> Confirmation | | |

Am I ROBUST?	Handled errors/EXCEPTIONS well? Work on different ENVIRONMENTS?	Not affected by DEPENDENCIES? Will system attributes be met?	"I am resilient"
---------------------	--	---	-------------------------

- | | | | |
|---|--|---|--|
| Errors handled?
<input type="checkbox"/> Connection loss
<input type="checkbox"/> Low resources
<input type="checkbox"/> Services down | Env friendly?
<input type="checkbox"/> Diff browsers
<input type="checkbox"/> Diff resolutions
<input type="checkbox"/> Diff devices
<input type="checkbox"/> Diff version OS/SW.. | Dependencies ok?
<input type="checkbox"/> In common lib
<input type="checkbox"/> In shared data
<input type="checkbox"/> memory
<input type="checkbox"/> files/DB
<input type="checkbox"/> Ext settings/config | Attributes ok?
<input type="checkbox"/> Security
<input type="checkbox"/> Large volume
<input type="checkbox"/> Basic performance |
|---|--|---|--|

Are you OK?	Used/Released RESOURCES? SETTINGS/CONFIG change side effects?	DATA change side effects? INTERFACE changes side effects?	"I am a good citizen"
--------------------	--	--	------------------------------

- | | | | |
|--|--|---|---|
| I am not messing up the environment | I am not messing up other's code via side effects due to changes that I may have made in my SETTINGS/ CONFIG/DATA/ INTERFACE | | |
| Resource usage ok?
<input type="checkbox"/> No leaks
<input type="checkbox"/> memory
<input type="checkbox"/> handles
<input type="checkbox"/> OS resources
<input type="checkbox"/> No tmp files | SETT./CFG side effects?
<input type="checkbox"/> App settings
<input type="checkbox"/> Env. settings
<input type="checkbox"/> Permissions | DATA side effects?
<input type="checkbox"/> Formats
<input type="checkbox"/> Types
<input type="checkbox"/> Defaults
<input type="checkbox"/> Width | I/F side effects?
<input type="checkbox"/> API parameters
<input type="checkbox"/> Msg parameters
<input type="checkbox"/> DB tables
<input type="checkbox"/> File contents |