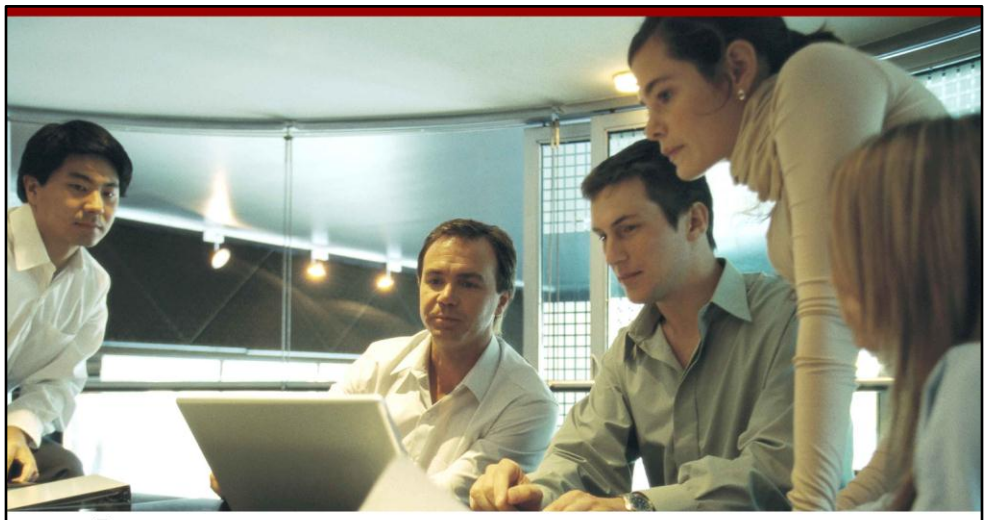
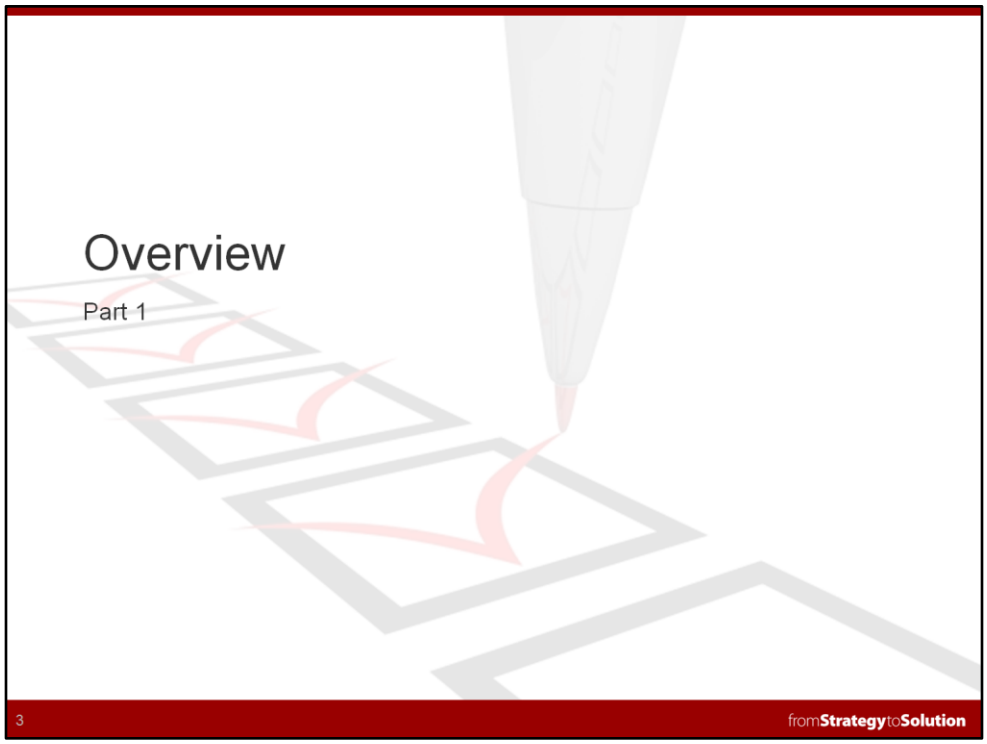


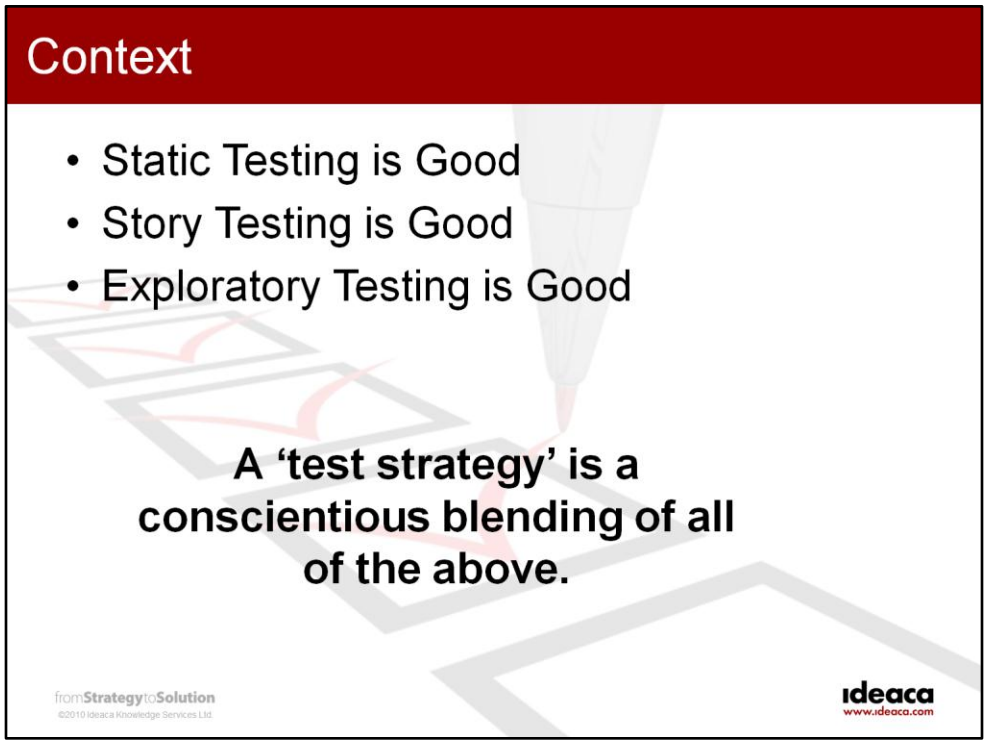
ideaca
www.ideaca.com

Module 2 – Working with Checklists

Methodi Testing Training Series

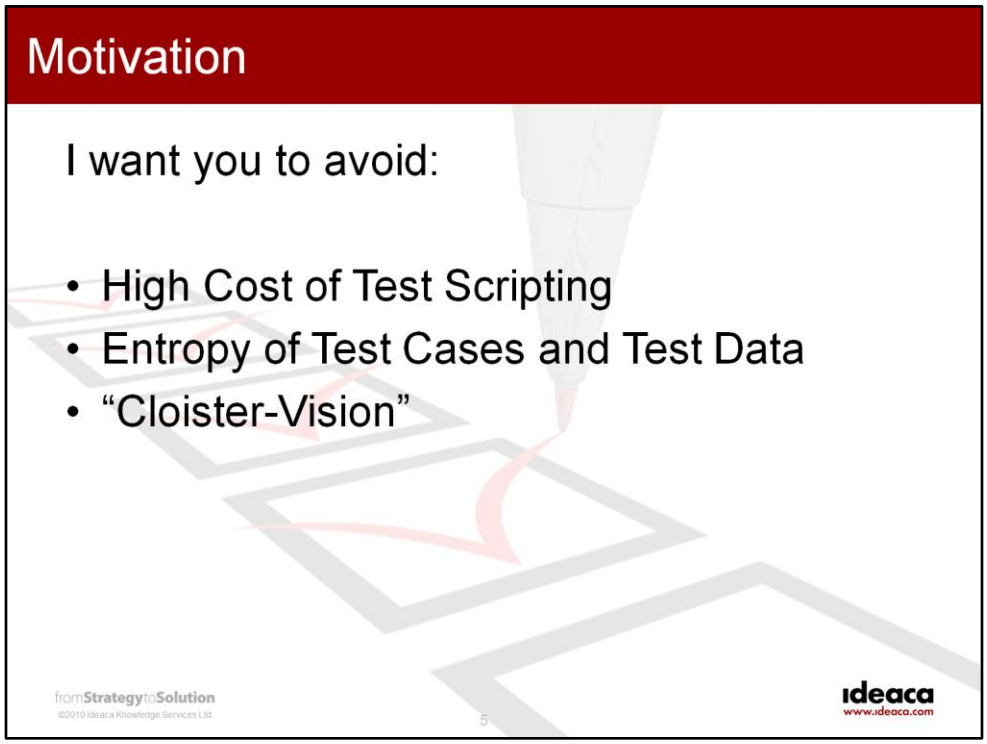
fromStrategytoSolution





My starting position is that a test strategy is a choice; as a test strategist, you choose how thick/thin these test approaches are applied, when they are applied, how they relate to other types of test such as load/performance testing, and so on. You should make this choice as a function of the situation at hand, that is, the context surrounding the need for testing in the first place.

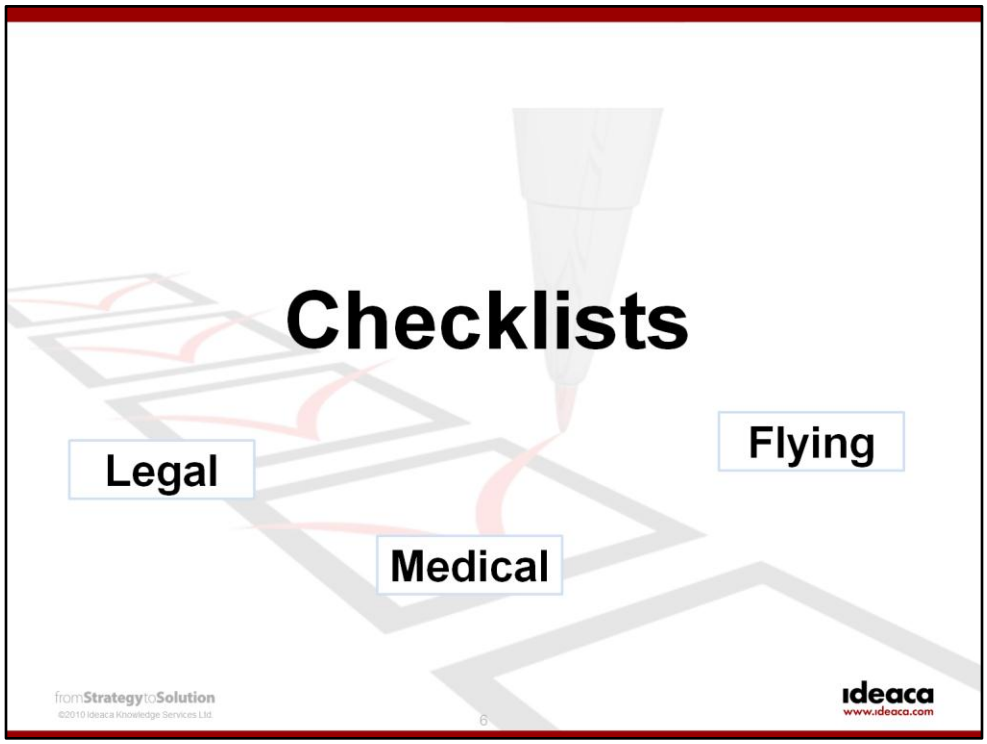
A corollary to my starting position is that static testing (inspections, reviews, and the like) alone is not adequate. Nor is story-testing, whether it is applied test-last, or as part of development in a ATDD/BDD style. Ditto for exploratory testing.



My motivation for giving the talk is the weaknesses of script-centric test strategies:

- Test scripts cost money to build and maintain, especially if they are written in a "how-to" style as opposed to a "what" style
- Test cases, identifiable by the presence of test data, suffer from entropy, that is, their relevance isn't guaranteed to last for all time
- "Cloister-vision", my term for an ailment that teams suffer from when the test mission is separated by either time or cognitive space from the business users that will adopt the solution.

I won't go into detail on the danger of test scripts because Cem Kaner did that in is CAST 2008 Keynote (Toronto). Test cases and test data get out of date, even if the business domain hasn't undergone serious change itself (btw unlikely). Test environments get refreshed from production, and key business relationships change (customer or suppliers referenced in tests), master data changes make some test cases irrelevant, etc.



My proposed solution is not an original proposal – the use of checklists. There are others that have proclaimed the value of checklists for testing and I hope to build on their work and extend in at least one (hopefully) original way.

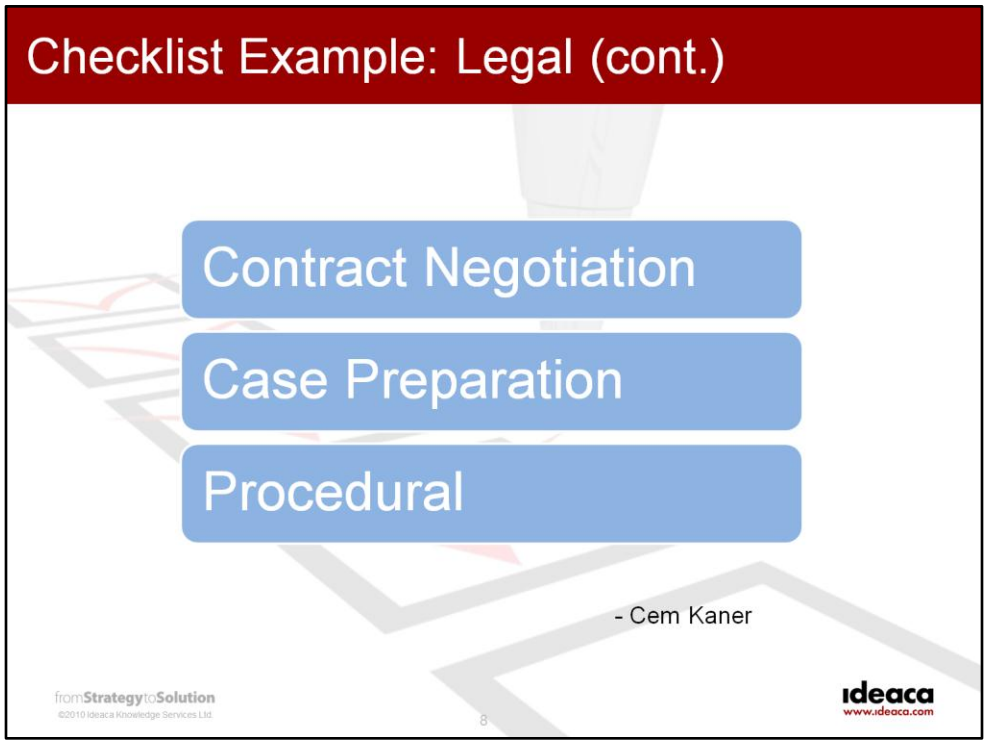
There is precedent for using checklist in the legal, medical, and flying industries that I will introduce/review briefly before describing checklists for testing in more detail.

Checklist Example: Legal

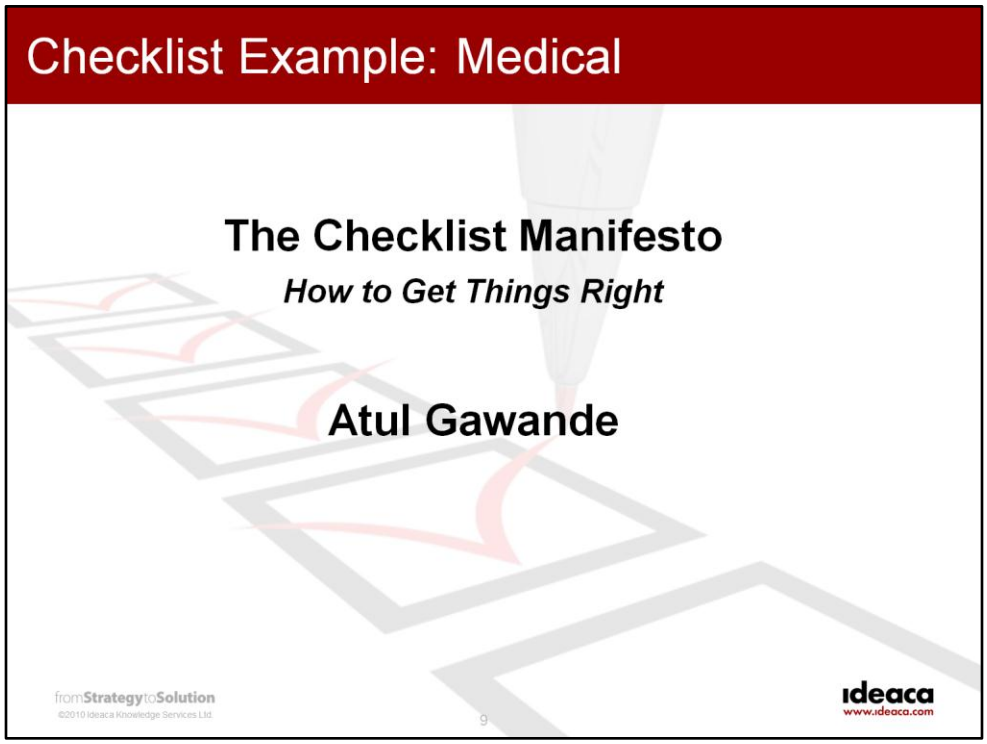
"I am a lawyer, and I can say with confidence that in THAT field:

- Checklists are tremendously valuable and lawyers would often be fools not to use them
- But checklists are NOT scripts"

Cem Kaner (CAST 2008)

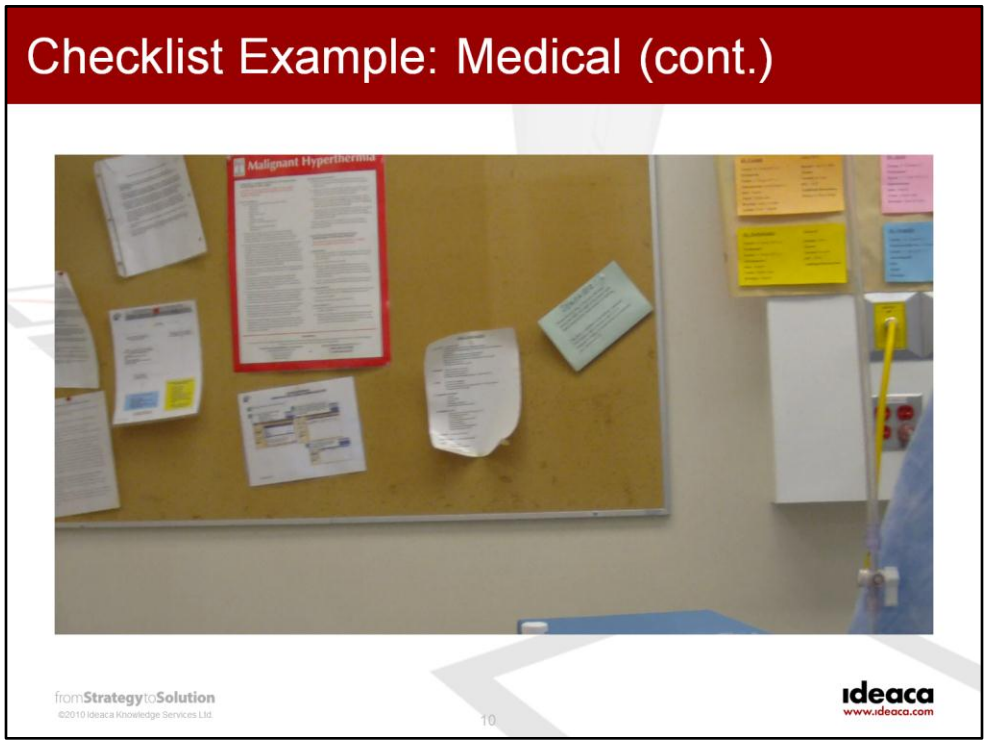


Kaner goes on to describe that in his experience, there are checklists for contract negotiation, case preparation and for adhering to certain procedures. He states that the procedural checklists might tend to look like scripts, however, they still leave enough room for a practitioner's expertise/experience to be applied, making them distinct from a script.



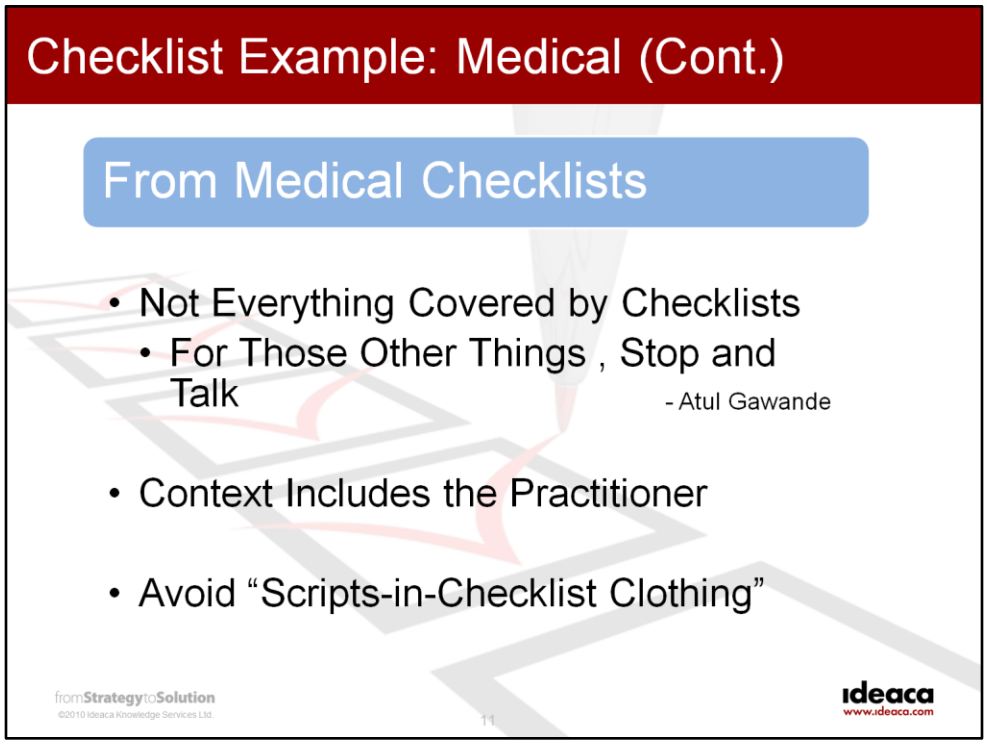
From the medical field, there is a book entitled, “The Checklist Manifesto” where Dr. Atul Gawande (a medical doctor) cites the beginning of checklists in the medical field and the vast difference they made in situations such as operating rooms. It is a wonderful read (if not, ironically, text-heavy).

In the book, Dr. Gawande describes how surgery has four big killers: infection, bleeding, unsafe anesthesia, and the “unexpected”. The first three are excellent candidates for checklists, but the fourth one is not. The classic checklist by itself does not help, instead the researchers identified that stopping and talking about the case together was the best way to approach it. An important lesson for testing using checklists: it isn’t going to catch everything, you can’t simply walk through the checklist and declare being done. The checklist plus the context is what is important, and that warrants a conversation.



I have my own experience with medical checklists in an operating room – that's a photo that I took while our daughter was being born. The different-coloured pages on the wall in the upper right-hand side are checklists for the same procedure for each of the different surgeons. Even more context.

So it seems that checklists don't have to be the same for everyone, we can each have different checklists for the same thing depending on our experience, background, and preferences.



Apply this to testing using a checklist: we cannot assume that the checklist is a list of the only things that we need to check, we must assume that there is something "unexpected" that might cause a failure and we should stop and talk about the solution and how we're testing in order to maximize our chances of preventing that failure.

And maybe it's OK if we don't all have an identical checklist, that is, a reminder that the checklist should be devoid of "how" and ideally focus on "what". It should not be a script-in-checklist-clothing!

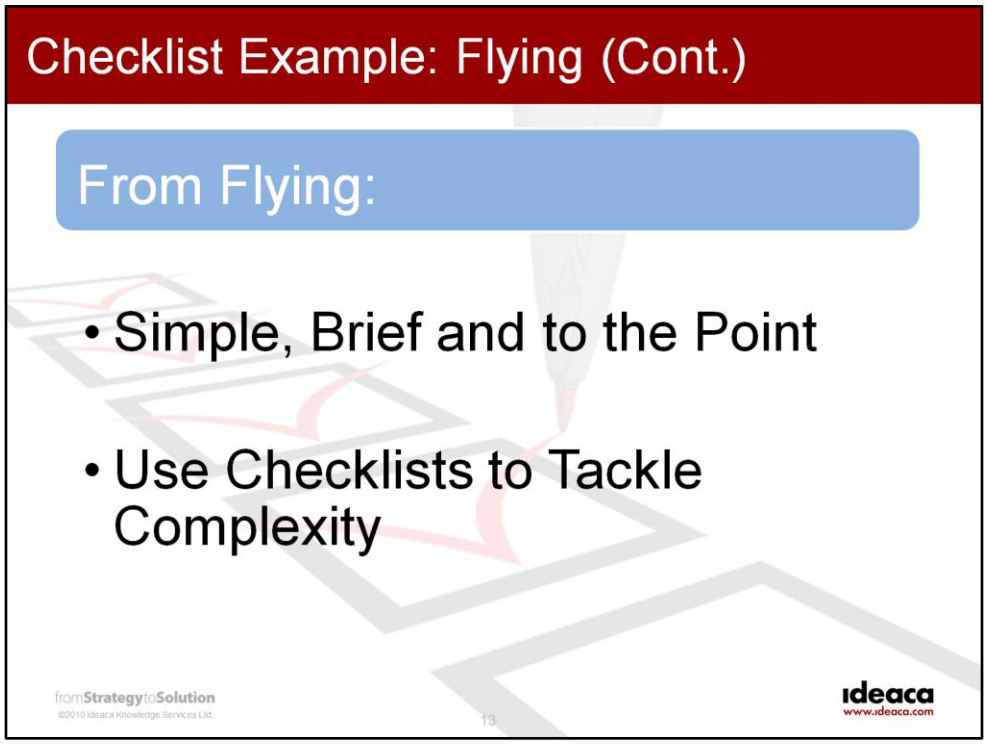
Checklist Example: Flying

Thirty October 1935: The Model 299 burns at Wright Field after taking off with the controls locked. Major Ployer Hill was in the left seat, Boeing test pilot Leslie Tower in the right. Rescuers managed to pull both men from the shattered cockpit but the two pilots died shortly after. Three other crew in the rear fuselage managed to scramble to safety even though they were burned. Lt. Robert Giovannoli rescued the pilots, receiving burns in the process, but he was killed in an air crash before he could be given an award for heroism. (USAF Museum)

Via http://www.nationalmuseum.af.mil/factsheets/factsheet.asp?id=2478

12

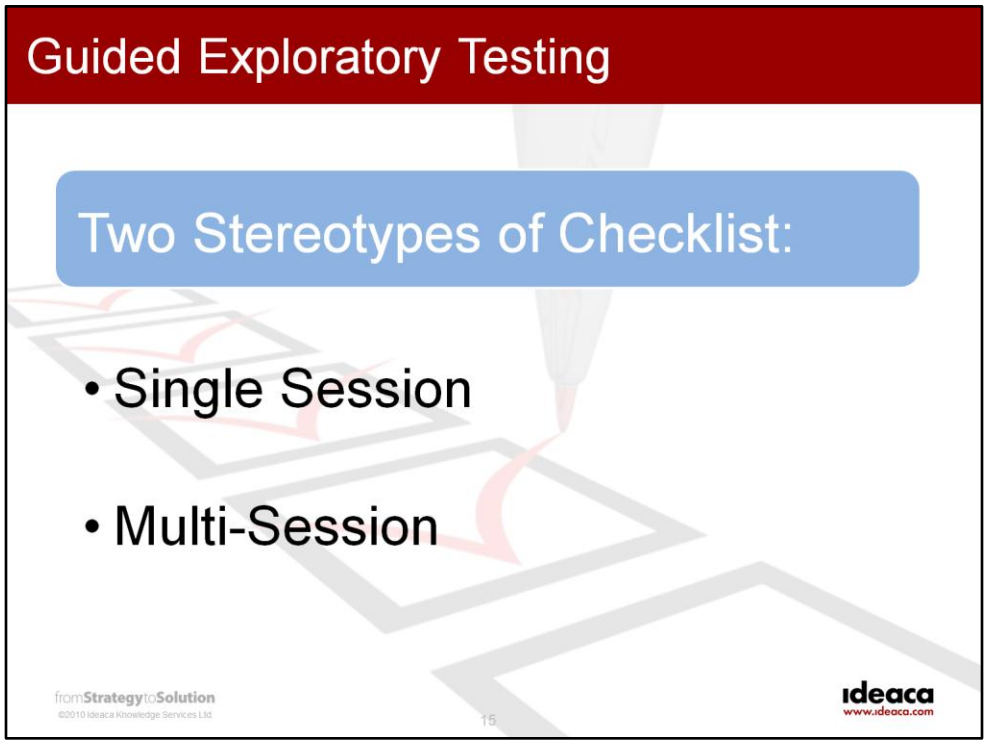
The 'Flying Fortress' was supposed to be one of those breakthroughs in aviation – and it was, but for an unfortunate reason. It crashed on one of the early flights simply because (after analysis) that there were locks left 'on' and even the experienced pilots couldn't control the aircraft as a result. The plane was deemed too complex to fly. Put another way, this was the first time that a plane was built that was too complex for even a trained pilot to fly safely; there was just too much to remember. The resolution came from the pilots involved in the analysis: create checklists for takeoff, after takeoff, before landing, after landing (paraphrased from checklists.com).



The counter-intuitive part of this is that checklists are there to tackle the complexity. We might be tempted to believe that because something is complex, we need a script. The experience reported by Gawande is the opposite: the pilots created the checklist in response to the complexity.

We can apply this lesson directly to testing. Avoid the temptation to create a script, even in the face of complexity. Frequently-asked question: what about tests that aren't executed often, such as a year-end process or a test that involves a number of different systems? My contention would be to stick to separating concerns; avoid mixing the "how to" information that people need with the checks on the checklist.

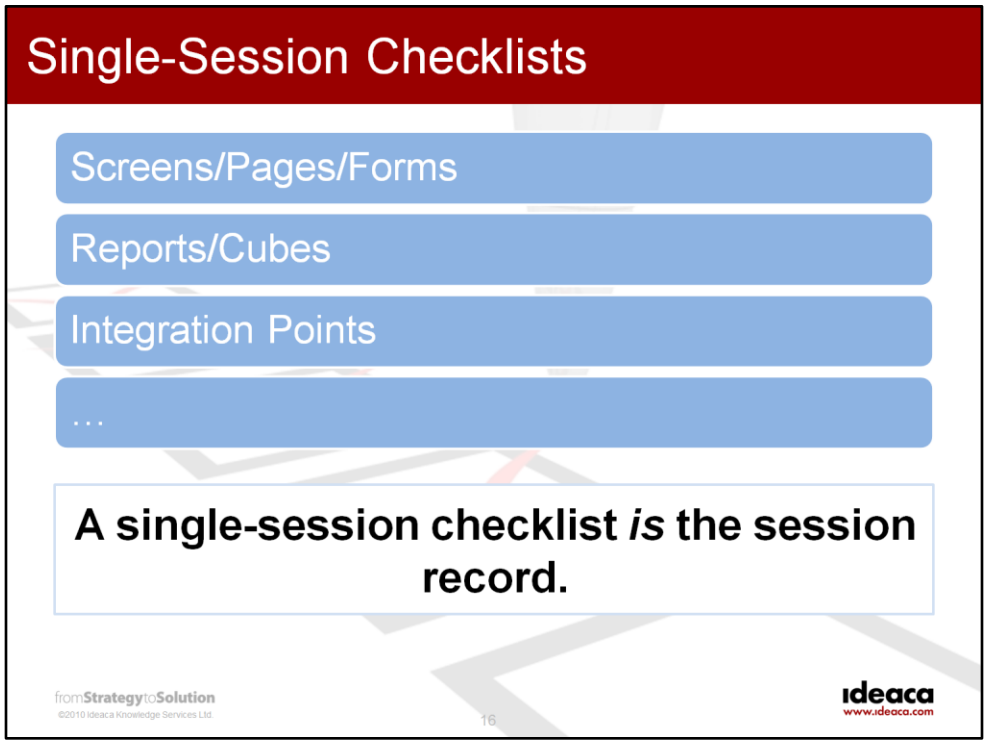
Like others before me (Kaner, J. Bach, J. Bach) I contend that checklists and exploratory testing are synergistic. I use the phrase “guided exploratory” testing to emphasize the idea that explorers still have a context that they need to operate in – they generally can’t explore away until their time runs out. Progress and status updates need to be made, at least in most enterprise testing contexts. Checklists are excellent means of providing both guidance to those testing and to those needing to understand the testing.



First, I assume that you are already familiar with session-based testing and how it relates to exploratory testing (ref: Jon Bach). At higher test levels such as UAT, we've used session records in a formal sense, that is, they are audit trails of testing activity. We expected to find evidence that testing was done, and that it was according to the process outlined in the test strategy. The strategy simply stated that testers would generate session records that mapped what they did to what they expected to what they observed.

A single-session checklist contains the heuristics that we wanted the testers to use in their exploration, and we found that we could tailor these checklists to specific types of test target. More to come.

A multi-session checklist is more oriented to test management, that is, understanding what will be tested and what has been tested. They support the coordination and collaboration across a number of testers oriented to a number of test targets.

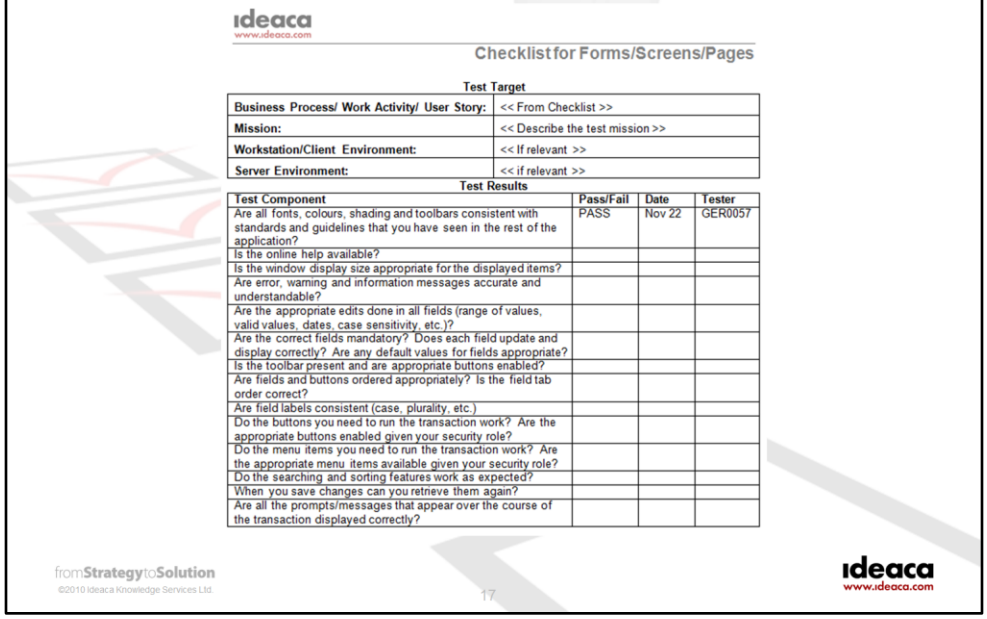


The test logistics would be something like the following.

- The unit tests are all passing.
- The story tests associated with a given use case are all passing.
- The tester opens the screen/page/form checklist and runs through the main success scenario, testing items from the checklist along the way. This means entering expectations, recording observations on the checklist.
- The tester continues to explore variations of both success and failure scenarios to seek mis-alignments between expectations and observations.
- When the tester runs out of smells to investigate, the completed checklist is submitted into the test repository.

Note: a story test is (typically) an automated functional test that encodes the minimal acceptance criteria (MAC) for a given user story. A use case scenario might be comprised of one or more user stories, so there might be many story tests for any given use case scenario.

Single-Session Checklists (Cont.)



ideaca
www.ideaca.com

Checklist for Forms/Screens/Pages

Test Target

Business Process/ Work Activity/ User Story:	<< From Checklist >>
Mission:	<< Describe the test mission >>
Workstation/Client Environment:	<< If relevant >>
Server Environment:	<< if relevant >>

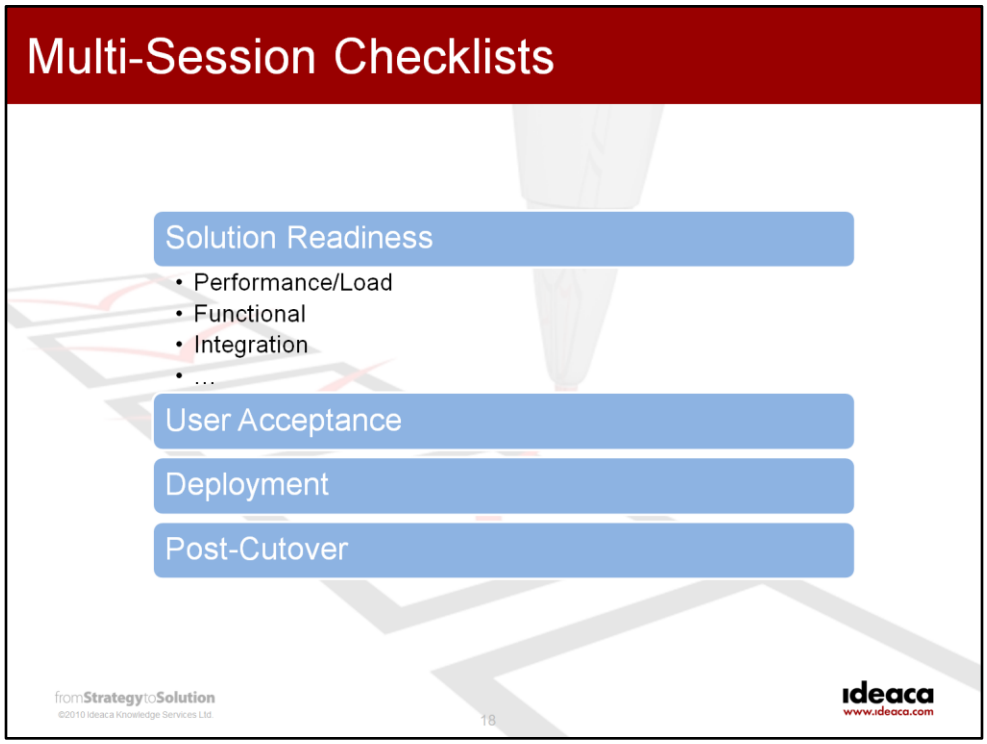
Test Results

Test Component	Pass/Fail	Date	Tester
Are all fonts, colours, shading and toolbars consistent with standards and guidelines that you have seen in the rest of the application?	PASS	Nov 22	GER0057
Is the online help available?			
Is the window display size appropriate for the displayed items?			
Are error, warning and information messages accurate and understandable?			
Are the appropriate edits done in all fields (range of values, valid values, dates, case sensitivity, etc.)?			
Are the correct fields mandatory? Does each field update and display correctly? Are any default values for fields appropriate?			
Is the toolbar present and are appropriate buttons enabled?			
Are fields and buttons ordered appropriately? Is the field tab order correct?			
Are field labels consistent (case, plurality, etc.)			
Do the buttons you need to run the transaction work? Are the appropriate buttons enabled given your security role?			
Do the menu items you need to run the transaction work? Are the appropriate menu items available given your security role?			
Do the searching and sorting features work as expected?			
When you save changes can you retrieve them again?			
Are all the prompts/messages that appear over the course of the transaction displayed correctly?			

fromStrategytoSolution
©2010 Ideaca Knowledge Services Ltd.
17
ideaca
www.ideaca.com

A couple of considerations:

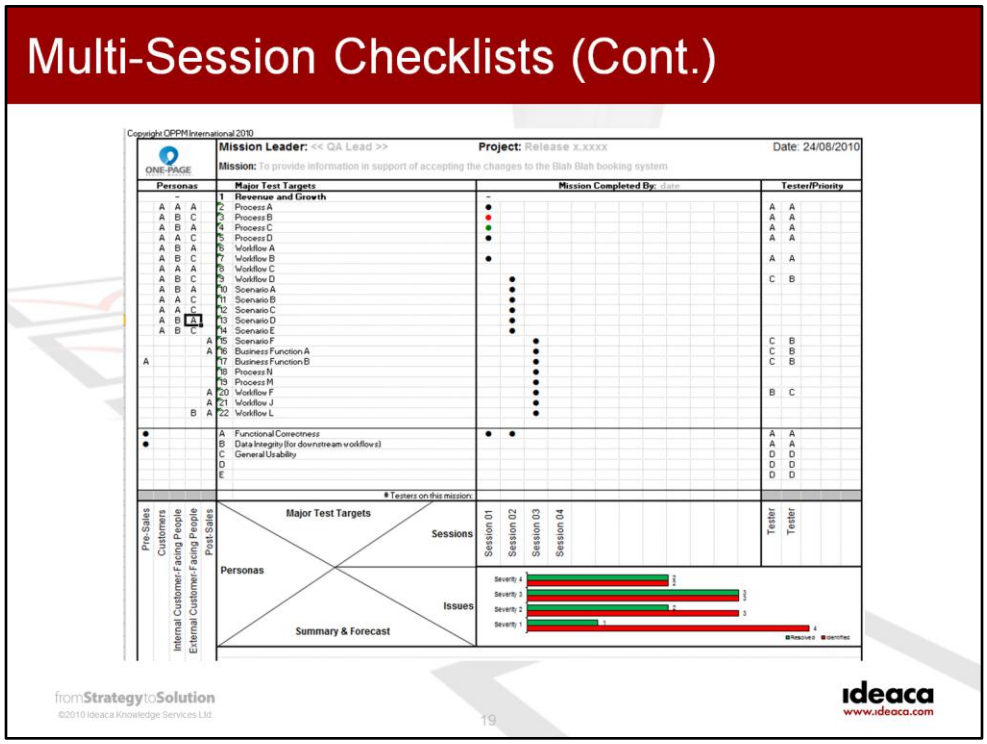
- A professional tester might bring their own version of this to the table – might not even be recorded as such.
- A business tester seconded to the project for a specific length of time might not have such distinctions to have their own – this is where a single-session checklist becomes useful. These business testers are generally not professional testers but they are being asked to test in support of either solution readiness testing or user acceptance testing.
- The items on the checklist aren't in any particular order.
- The items on the checklist aren't complete – additional heuristics such as those found on @testobsessed's heuristics cheat sheet support the items on this checklist.



Multi-session checklists support a number of investigators working over a variety of test sessions. We have integrated the use of checklists into the overall test strategy by focusing key release management decisions on checklists:

- Solution readiness: is the release of adequate quality that we should ask the users to adopt it?
- User acceptance: are there any barriers to the users adopting the release, now?
- Deployment: is the organization ready to start using the release, now?
- Post-Cutover: did we install/configure it correctly, can users start using it now?

In some organizations, the last two decisions might be part of the 'release plan' and in the hands of the release team as opposed to the test team.

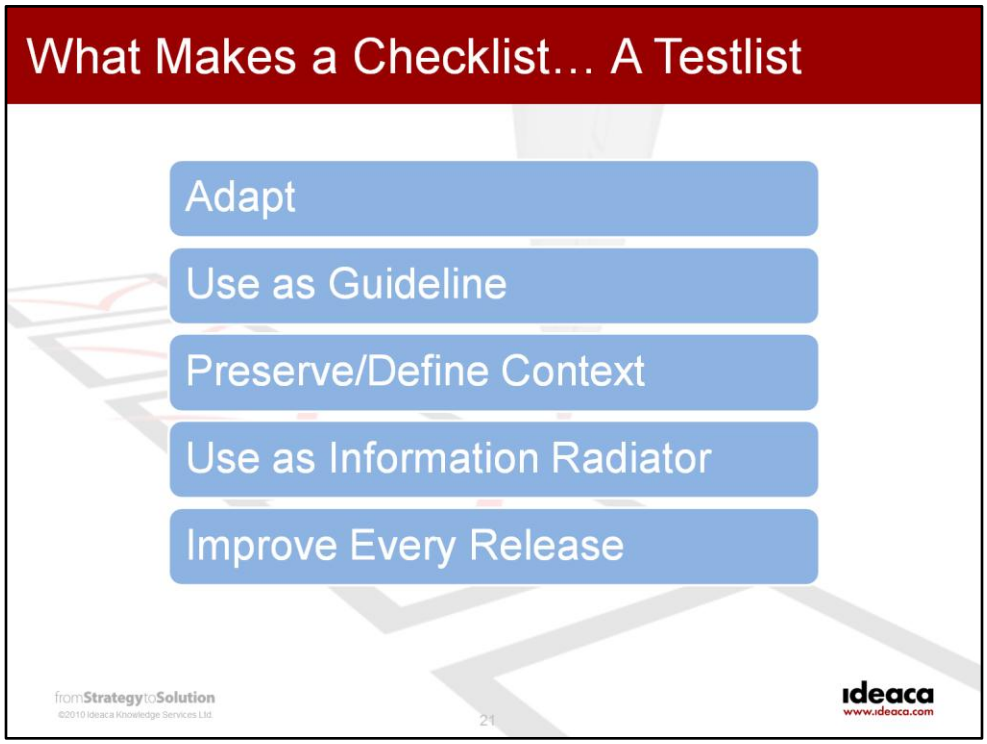


The above chart is an extension of the One Page Project Manager (http://oppmi.com/) for testing involving a number of testers over a number of sessions. The checklist content is in the section labeled 'Major Test Targets'. Sessions might be one-week, one-day, one-hour long sessions, depending on the level of abstraction of the checklist. Note that our experience using this in one very large effort was that if the sessions on this chart are one-week duration, then there needs to be a lower-level checklist for each day within that session. Different audiences.

- On one project the daily checklists were further broken out by tester so that we could generate single-session checklists for each one of them. In that case, the single-session checklists became more akin to call sheets used in making movies to ensure that the right actor was in the right place at the right time, and that they were prepared to complete a specific scene. (Aside: my sister worked in the movie industry and we adapted her call sheet template for our purposes). More study needed to confirm that this is useful, but it seems to be, especially when the testers are not dedicated 100% to the project.

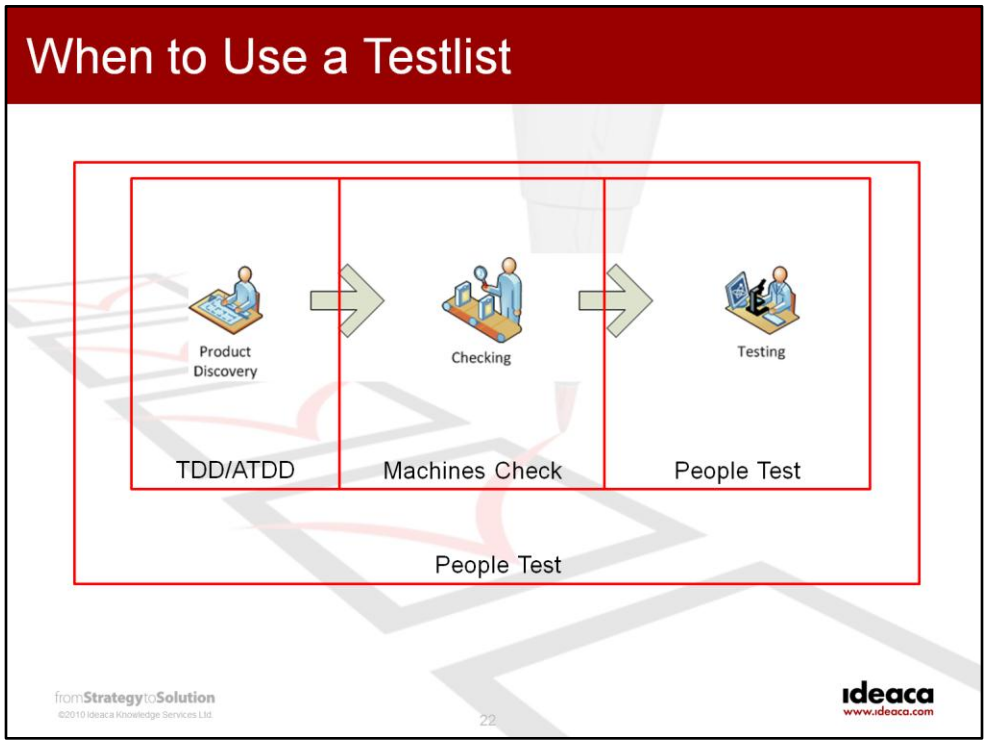
The problem with the word “checklist” is the first part of it – “check”. As per Michael Bolton, “checking” seems to imply a non-sentient sort of verification step, something that a machine could do, or something that a person could do without thinking. That’s not what we want.

Adopting Bolton’s use of the word “test” to indicate thinking was involved, I’ve adopted the word “testlist” to promote a way of working with a checklist that requires thought and analysis. Exploratory testing provides the “thought and analysis” and the checklist provides the guidance from one release to the next, or from one test run to the next, as the linchpin for both guiding and reporting on the work.



Here are the ways to convert a checklist into a testlist:

- Evolve the testlist as the team learns. Avoid finalizing the testlist ahead of time and then fight changes to it. Adapt and evolve.
- Encourage/demand people use it as a guide but not a list of the only things they test.
- Make it hierarchical, or evolve it into a mindmap so that big things and little things have a place (a testmap).
- Add attributes or map markers so that testers know when to go fast and when to go slow. Candidates: risk, areas of intense change, personas …
- The end game isn't working software, it's solution adoption. Observe what happens after go-live.



Automated test and deployment is only going to increase and we should be viewing that as a good thing provided the effectiveness of the automated checked is continually improving. In a typical agile project, for example, the TDD/ATDD process yields a set of automated tests, that is, the set of tests that were used to help discover the right product to build, and to help verify that it was built right. Later, those same tests, perhaps supplemented by a formal test group perhaps not, are run as regression tests. THEN people test using testlists.

Certainly the checks are input to the exploration/investigation, but mostly those checks enable the people testing to focus on barriers to successful adoption like usability, mobility, security, organizational readiness, etc.

There are opportunities to explore/investigate using testlists throughout the entire process, even if that feels counter-intuitive. The developer writing unit tests, for example, is thinking first, automating second. In effect, that is turning a testlist item into an automated check – a natural process that we should encourage. This can occur at all levels of abstraction.

Aligning levels of detail to fit the test mission. If your test mission is UAT, then you will have business processes, business scenarios, business workflows, and business functions on your testlist. If you are performing functional testing on a CRUD screen, then you will have usability expectations on your testlist. And there are levels of abstraction in between these two examples.

Writing with strong nouns and strong verbs is the way to go for testlist items. Thinking about business objects, states, and state transitions is an effective source of strong verb-noun combinations. Strong verbs: approve, reject, calculate, generate, define, register, (and there are domain-specific strong verbs such as to check-in (for a flight), to book (customer service), to cauterize (surgery), etc. Strong nouns are from the business domain: invoice, customer, account, etc.

JAT == Joint Application Testing. Create testlists collaboratively and with everyone involved early and often (principle from lean).

Thank You

- To those that thought of things like this before:
  - Cem Kaner
  - Jon Bach
  - James Bach
  - Michael Bolton