Testing is not Quality Assurance

Crash course into aerospace-grade QA

Ondrej Tulach

Senior QA, Test Analysis courses lector, 10 years in CA/Broadcom

Who am I – and what has happened to the QA?

- I was system and application admin, developer (C#.NET OOP, PL/I procedural)
- Switched to QA in CA
  - Why would I do that? To get my hands on exotic systems and play with them
- I got SOTA QA training – ISTQB via Rex Black, The Testing House
  - Then, all those skills disappeared completely. Nobody teaches them anymore.
- I was one of CA‘s pioneer test automators
  - ...but found out Automation ≠ quality
- Today: industry idea of „SQA“ is 99% „automation“
  - Everything else is „Terra incognita, hic sunt leones“
- 10 years QA experience, continuous search for QA State-of-the-Art

- Note: no camera, Q&A, polling (5pts most, 1 least)

BROADCOM

Controversy imminent: I will prove that

- „Testing“ is the least important part of QA
- Most QA metrics are misleading or deceptive (Code coverage…)
- Automation cannot be the solution of weak QA
- Most teams massively undertest – not by effort/numbers, but by focus

BROADCOM

Testing may be easy.
Designing good tests is not.

What is and isn‘t easy in „testing“

„No one ever sits in front of a computer and accidentally compiles a working program, so people know—intuitively and correctly—that programming must be hard.

By contrast, almost anyone can sit in front of a computer and stumble over bugs, so people believe—intuitively and incorrectly—that testing must be easy!“

~Michael Bolton, DevelopSense

- It‘s easy to „find *some* bugs“...
- QA purpose is increasing reliability and confidence, trust in the product.
- „*Yeah well, we did find some bugs and fixed them*“ – feeling confident?

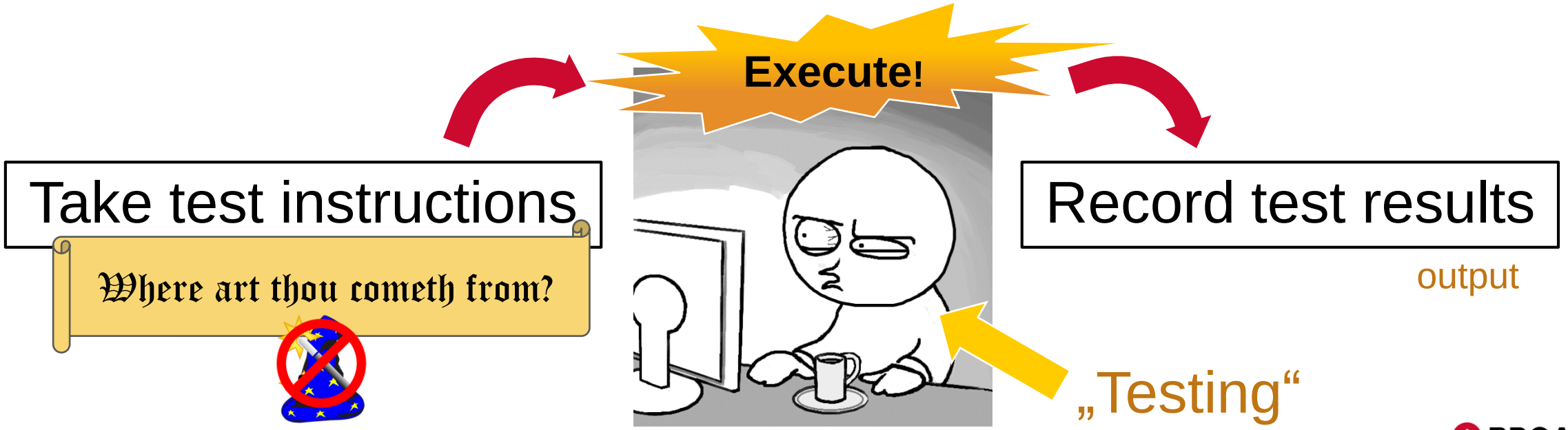
- Not „finding some“ bugs.
  Hunting for *specific* bugs to ensure reliability of the product

BROADCOM

Testing is just the tip of the iceberg

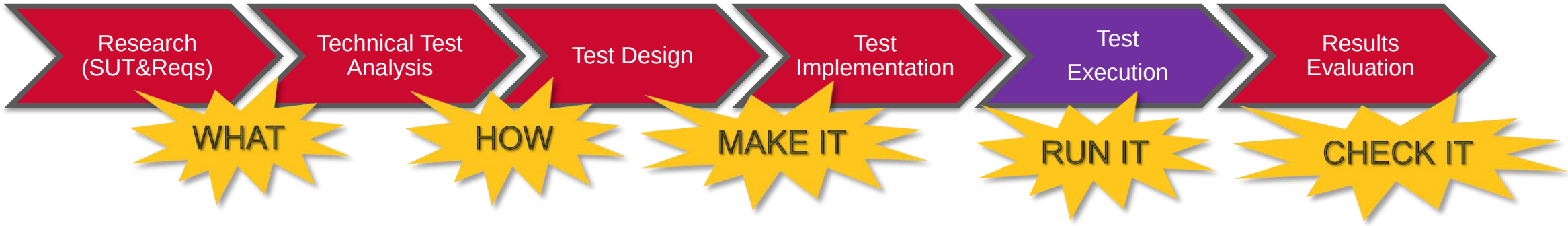
- *„Procedure of submitting a subject of test to such conditions or operations as will lead to its proof or disproof or to its acceptance or rejection”* (Merriam-Webster)
  - Generalization: intent is logically separated from testing as activity. Decompose more...

- *„Subjecting a system to certain activities and recording system response“*
  - This is the core definition of testing. Everything else is connected but logically separated activity



BROADCOM

General QC workflow: „Where does the Test Design come from?“



- „Testing“ (Test Exec) is mechanical, uncreative task. Proof: TestAuto.
- The quality depends on Analysis & Design phases
  - The unobvious and grossly neglected ☹
- If you focus 99% of time and resources on Execution/Implementation...
  - How thorough is your test analysis and design? Trust the coverage?

BROADCOM

Test Analysis has „levels“ – what and how do the tests cover

1. **Level 1: positive tests only (QC)**
   - Focus on proving correctness: „*I will demonstrate that the SW works as advertised*“
   - Assumption: only correct, expected inputs are provided: „*demo that the precise way I use it, SW works*“

2. **Level 2: destructive tests (QC)**
   - Focus on finding weaknesses: „what could break it“? → find defects in-dev, not in-the-wild
   - Challenge developer‘s assumptions, discover un-sanitized inputs

3. **Level 3: reduce risks (QC+QA)**
   - System-focused: reduce overall risks even due to external influences (ENV tests); review design
   - QA engages in design, architecture etc., to warn+mitigate possible risks early = cheaply

4. **Level 4: preventive actions accross system lifecycle (QC+QA)**
   - All of the above, plus systematic Reliability Engineering using FHA
   - Models of system/data failures
   - Adopting Analysis↔Design loops across lifecycle

BROADCOM

TTA levels: why good test coverage requires level 2

Debugging	Verification testing	Destructive testing	Risk reduction	Preventive QA
Ad-hoc tests	Prove SUT function with: ▪expected inputs ▪customer use-cases ▪industry standards	All from L1, +	All from L1 + L2, +	All from L1 + L2 + L3, +
No reproducibility		BVT, EPs+DIs	QA of QA: Test Classes	Fault Hazard Analysis, Design↔QA loops
No test analysis				
„Tested“ after single success	Prove negative function: ▪correct error messages ▪no false positives	HOET / UIET	Architect + Test Analyst roles	System modelling: ▪State Transition Testing ▪Fault Tree Analysis ▪FMEA ▪Data-flow testing
		Integration FMT	Informal design feedback	
	Prove client compatibility ▪different clients ▪different locales...	Stress testing	Environmental testing	
		Load testing		Lifecycle focus (SYSENG)
		Challenge assumptions	FIT	Escaped defects RCA
			Regression testing	Scalability
			Integrity levels	Maintenability
				Deployability

Post-2010, majority of SW industry „QA“ is limited to subset of „Verification Testing“ QC and completely fails to cover important testing scenarios *or* do QA work!

BROADCOM

Proof: most QA metrics are wrong

Almost all „QA metrics“ are deceptive and misleading

- Code coverage: misses *all* data bugs, misses >99% of code paths and more
- Path coverage: numerically impossible and still misses all data bugs
- Test cases number: says nothing about actual coverage or quality, contraproductive
- Variations number: great, you‘re randomly trying stuff. Says nothing about targeting the important.
- Automated tests %: what good are 100% automated tests when you have 5 of them?
- Testing cycle time: oh great, push for even less analysis, less design thought, less coverage.

- Goodhart‘s Law: „When a measure becomes a target, it ceases to be a good measure.“
- Paciga‘s Law: „Metrics deform people‘s goals. A counter-metric is required to avert backfire.“

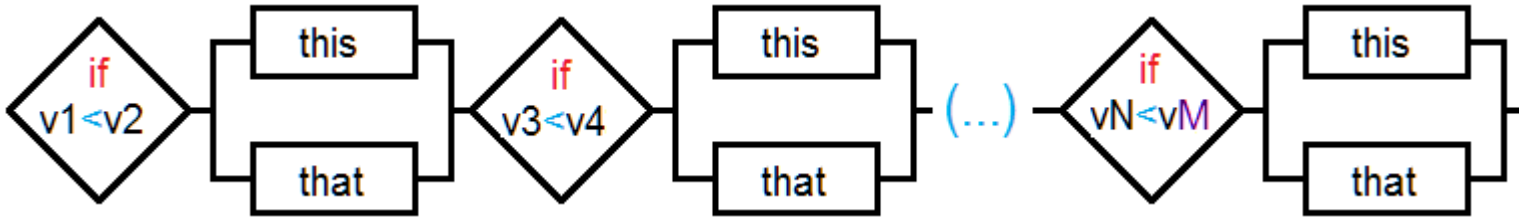
Counting commits? Devs will split to smaller commits. Counting bugs? QCs will find ways to report more bugs. Counting cycle time? People will cut corners for speed…

- What is your real goal? Arbitrary numbers – or better quality?

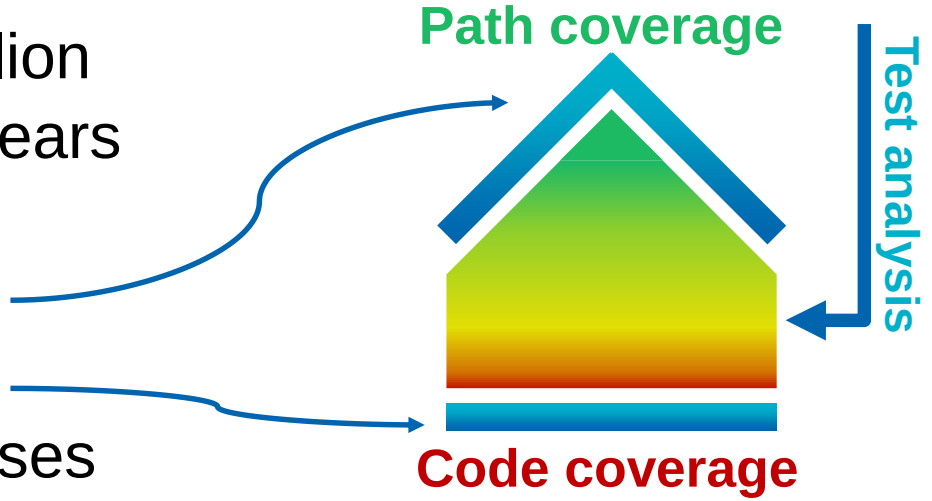
BROADCOM

The mathematical proof of „Code coverage“ fallacy

- 100% **Code coverage**: execute each statement
- 100% **Path coverage**: test all paths thru decisions, loops
- Example: code coverage=2×*m*, path coverage = 2^m tests



- Simple program with 70 branches: 2^{70} ≅ 1,2 sextillion
- **Automated testing** with 30sec á test case: 10^{15} years



- 100% **Path coverage** = maximum but impossible
- 100% **Code coverage** = possible but not enough
- **Test analysis** = hand-pick fewer important test cases
- (→“QA coverage metrics“ misleading)

BROADCOM

Data flows: why even 100% path coverage guarantees *nothing*

- Imagine such simple code, 2 branches:

```
private static float DivNumbers(float a, float b)
{
    if(a != 0 && b != 0)
        return a / b;
    else
        return 0;
}
```

- And we build 100% automated tests:

```
private static void TestIt(float[,] testTriplets)
{
    float currRes;
    for(int fa=0; fa<testTriplets.GetLength(0); ++fa)
    {
        try
        {
            currRes = DivNumbers(testTriplets[fa,0], testTriplets[fa,1]);
        }
        catch(Exception ex)
        {
                Console.WriteLine("ERROR in Test {0}: {1}",fa, ex.Message);
                currRes = float.NaN;
        }
        Console.WriteLine("Test {4} "+
            (float.Equals(currRes, testTriplets[fa,2])?"PASSED:\t":"FAILED: \t")+
            "{0} / {1} => {2} returned vs. {3} expected",
                testTriplets[fa,0], testTriplets[fa,1], currRes, testTriplets[fa, 2],
                (fa<10?"0"+fa.ToString():fa.ToString())
        );
    }
}
```

- With just these 2 test data sets:

```
float[,] testTriplets = new float[,]
{
    //dividend operand, divisor operand, expected result
    {1.0f, 1.0f, 1.0f}          //code path 1
    ,{0.0f, 0, float.NaN}       //code path 2
};
```

We achieved 100% automation, 100% code coverage, 100% path coverage!

- Enough? Really? What about other tests?

```
,{10.0f, 2.5f, 4}           //larger div smaller
,{10.0f, -5.0f, -2.0f}      //negative divisor
,{1.5e-45f, 1, 1.5e-45f}    //BVT min pass
,{1f, 1.5e-45f, float.PositiveInfinity} //BVT 1:min
,{1, 3.4e38f, 2.941177e-39f}//BVT 1:max
,{0.0f, -0, float.NaN}      //BVT negative zero
,{float.PositiveInfinity, float.PositiveInfinity, float.NaN}
,{1.0f, 3.0f, 0.33333333333333333333f} //periodic
,{4195835.0f, 3145727.0f,
    1.3338204491362410025f} //Pentium FDIV bug
```

- Point: not code flows, but also data flowing

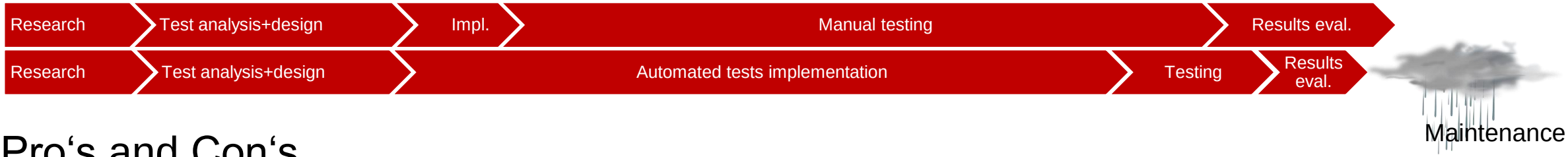
Traditional QA metrics are even more useless

- Number of test cases?
  - Compare 1000 redundant, poorly designed TCs with 50 laser-focused optimized
  - Real world regression testing: too much test cases!
- Number of variations?
  - Ignores mathematical theory behind Equivalence classes, QA stats about dual-mode defects
  - Random can help, but cannot replace skilled selection
- Cycle time
  - Technical Test Analysis and Test Design consume time with „un-demoable“ results.
    Too much focus on Cycle-time-to-demo, without regard for QA analysis, can **kill** quality
- Automated tests % ?
  - That‘s focusing on implementation and execution *away* from the analysis & design!
    - A propos…

Test Automation is not a cure-all

Faith in Test Automation as „magic bullet“ is tragically misplaced

- Yes, Test Automation has *many* benefits, but is *only a tool* with large limitations
- Test Automation cannot replace analysis:
  - Cannot cover defects via brute-force randomized testing „because sextillions Paths to test“
  - Even CASE-generated 100% path testing coverage has poor QC coverage becuase test data
  - Cannot cover test data because brute-force infinite monkey probability math (freeform inputs)
- Test Automation is purely about efficiency of *running* tests - but „It depends“



- Pro‘s and Con‘s
  - Enables frequent repetition of testing + regression testing (biggest advantage)
  - Trades many T.Technicians for fewer T.Automators – but analysis roles remain the same!
  - Autotest maintenance, refactoring grows linearly with time, number of tests!
  - SUT often requires development of in-house TA framework (when common TFs are poor fit)

BROADCOM

Test Automation is way of executing tests, not creating them!

- Automation cannot
  - Design tests (analysis)
  - Pick subset of useful testing data (analysis)
  - Discover risks (analysis)
- Automation can only efficiently *execute predefined* test scripts
- Even the best „automated car“ cannot drive you if you don‘t tell it *what* is your destination
- Automation is ***how*** you execute tests, not ***what*** do you test

- **Automation is not QA. It‘s SW development where QA is the customer.**
- **Too much focus on automation reduces analysis → reduces quality**
- Return to the fundamentals: technical test analysis

BROADCOM

Test Classes: mighty cheatsheet *and* metric

Test Classes: the the buried treasure from IEEE 829(2008)

- When I was digging through QA standards…
- Industry standard for QA test plans, IEEE 829-2008, had field „Test classes“:
  - „A designated grouping of test cases“
  - „Summarize the unique nature of particular level of tests“
- IEEE gave a few examples of Test classes:
  - „Positive (or valid) testing of input values that should be processed successfully“
  - „Negative (or invalid) values that should NOT process but do provide an appropriate error processing, such as an error notification message to a user“
  - „All boundary values, including those just above, just below, and just on each limit“
  - „Normal values based on usage profiles“
- **Wait, is that a cheat-sheet?!**
- There is finite number of Test classes → enumerate and magic happens

Test Classes: cheatsheet, checklist and spine for TTA

Why?

- Test Analysis requires „inspiration“ what areas to analyse
- Each TTA level requires considering lot of specific focus areas
- Higher TTA levels require „testing of tests“
- With dozens or hundreds of LLTCs, logical organization/“spine“ is a must

How?

- TCLs serve as „cheat-sheet“ to guide Technical Test Analyst through test design
- TCLs „map“ each test case to a sub-test class
- TCLs are checklist to verify no group of TCs was forgotten…

- **And hence could serve as *qualitative* QC metric!**
- Qualitative: avoid Goodharts and Paciga‘s law via explanation and cheatsheet

BROADCOM

Basic top-level Test Classes Cheat Sheet

1. Positive testing
2. Negative testing (not Destructive)
3. Destructive and advanced testing
4. Equivalence class+BVA testing
5. Input testing
6. Output testing
7. Front-end testing
8. Compliance testing
9. Integration and interoperability testing
10. Performance testing

Download https://broadcom.box.com/v/test-classes

1. **Positive tests**
   a. Smoke tests (bare minimum required for even attempting more tests)
   b. Inputs that should be included in output successfully
   c. Industry-standard values based on usage profiles
   d. Consistency of proccessing
2. **Negative tests**
   a. Data/functional (against false positives, false negatives)
   b. Error and exception handling
   c. Safety tests
   d. Fail-safe
3. **Equivalence class-based tests**
   a. All boundary values (including just above, below, and on each limit)
      i. Freeform input value limits
      ii. Internal thresholds handling
   b. Proccessing pre-defined switches/options
4. **Input tests**
   a. User input sanitization handling
   b. Character conversion sanitization handling
   c. Internal data definitions handling
   d. External inputs handling (object names, object lists)
5. **Output tests**
   a. Actual SW payload (correctness tested in TCI 1/2/3, here focus on formatting, paging etc.)
   b. Messages and codes shown to the user
   c. Correct file/dataset handling (allocation, de-allocation, file/FS consistency, etc.)
   d. Databases interaction (not only payload, but also logging, saving timestamps etc.)
6. **Advanced or atypical circumstances**
   a. Advanced/atypical activity handling tests („expect insane user" tests)
   b. Advanced/atypical objects handling tests
      i. (eg. DB table's associated Archive/Clone/History table, exotic XML or containers)
   c. Advanced/atypical system states handling tests
      i. abrupt or incorrect system termination (forced shutdown, power outages, etc.)
      ii. once-in-a-nevermind scenarios (leap year, „2K", log rotation, DST changes, etc.)
7. **Feature/implementation specific tests**
   a. Compliance to government regulations (EU Directives/Regulations; US FCC, U.S.C.; etc.)
   b. Comformance to relevant industry standards (RFC, IEC, IEEE, ISO, ANSI, W3C, etc.)
   c. Interaction with utilitites or maintenance (handling DB REORGs; FS Defrag; etc.)
8. **User front-end testing**
   a. GUI tests
   b. Help tests (Help panels, help files, guidance messages and pop-ups, tooltips, etc.)
   c. Device/terminal compatibility testing (terminal/screen sizes, font availability, alien OS, etc.)
9. **System integration tests**
   a. Interfaces to internal features and systems
   b. Interfaces to external services and systems
   c. Interference from external services and systems
10. **Performance testing**

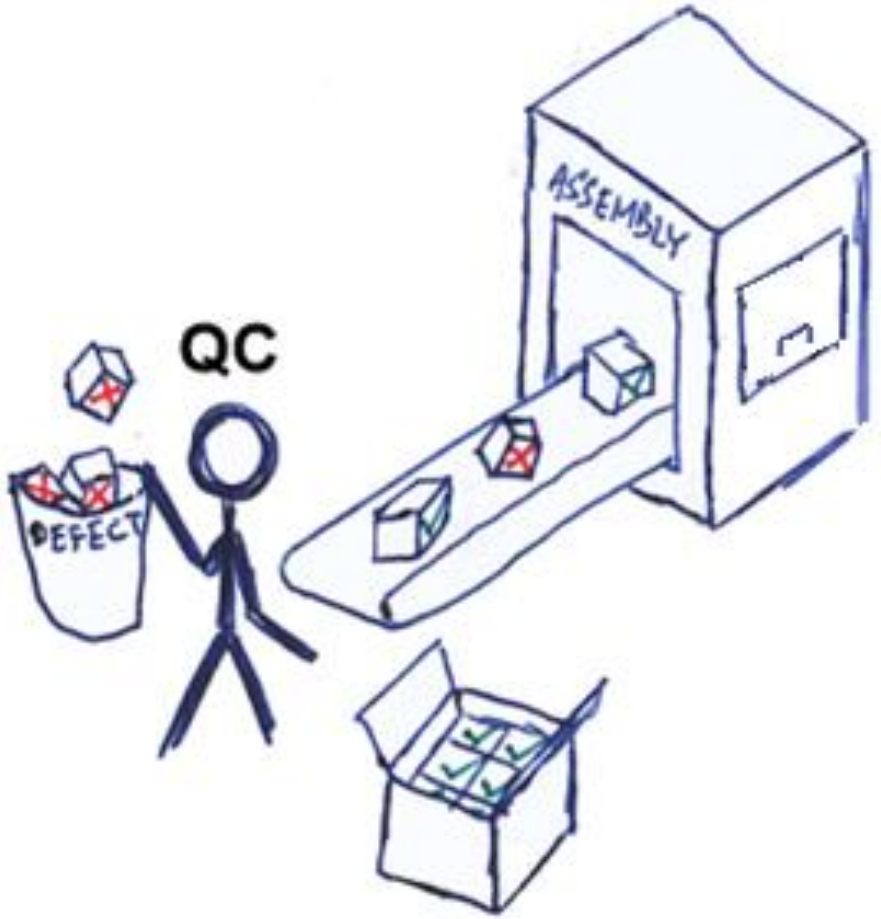
BROADCOM

QC does not equal QA (and why it matters)

What is Quality *Control*

- Checking finished product before allowing it down release chain
- Goal: stop defective products from being released
- Reactive process
- It doesn‘t matter *when* the QC activities begin (TDD), it matters *what* they do

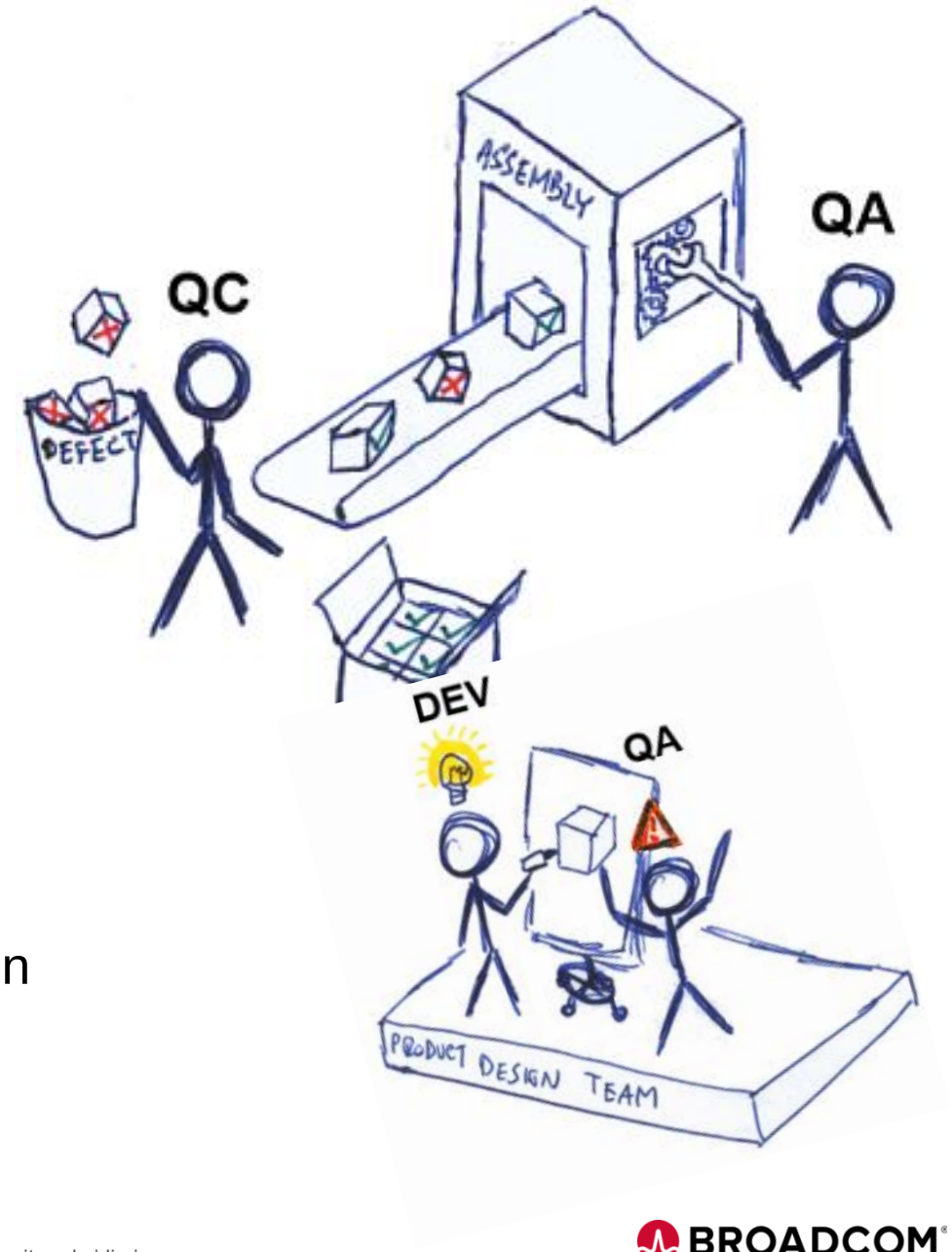
- Tools:
  - **testing procedures**: measuring properties or behavior of the product (syntax, compliance to specs, I/O)
  - **analysis**: identify what could fail, how, why



BROADCOM

What is Quality *Assurance*

- Intervention <u>across product lifecycle</u> to assure quality
- Goal: stop defective products from ever <u>being made</u>
- Preventive process
- Includes QC
- Tools:
  - **analysis**: identify what could fail, how, why
  - **processes**: help humans make less errors
  - **modelling**: simulate, deduce errors causes & propagation
  - **architecture**: robust design resilient against failures, scalable, maintenable, secure
  - **improved QC coverage**: detect more defects



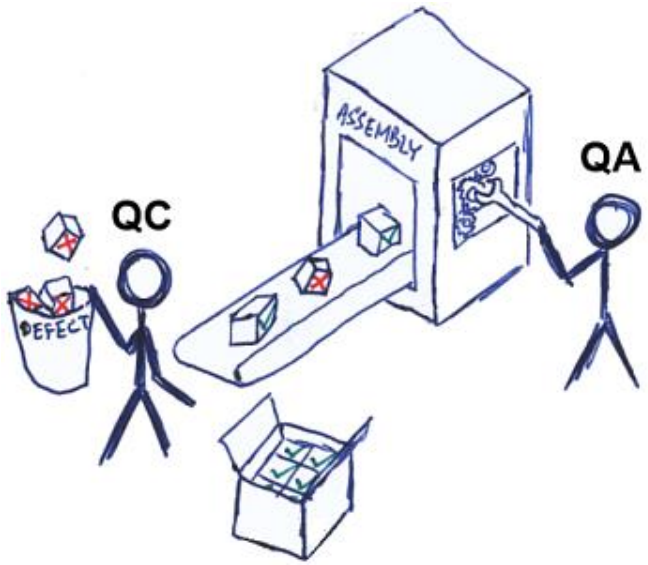
BROADCOM

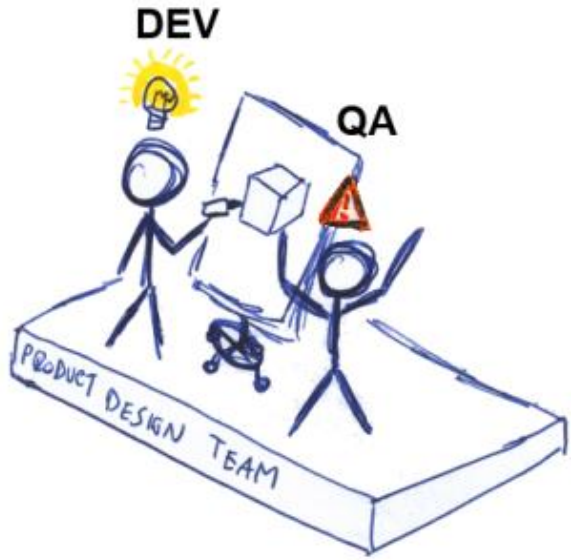
Quality Control vs. Quality Assurance

1. Targets built results
2. Analysis to identify what to test
3. Takes Dev assumptions for granted
4. Focused on verifying overt functions and obviously possible faults

5. Ends when subject passes tests
6. Balancing feedback loop („fix this until it‘s fixed“)

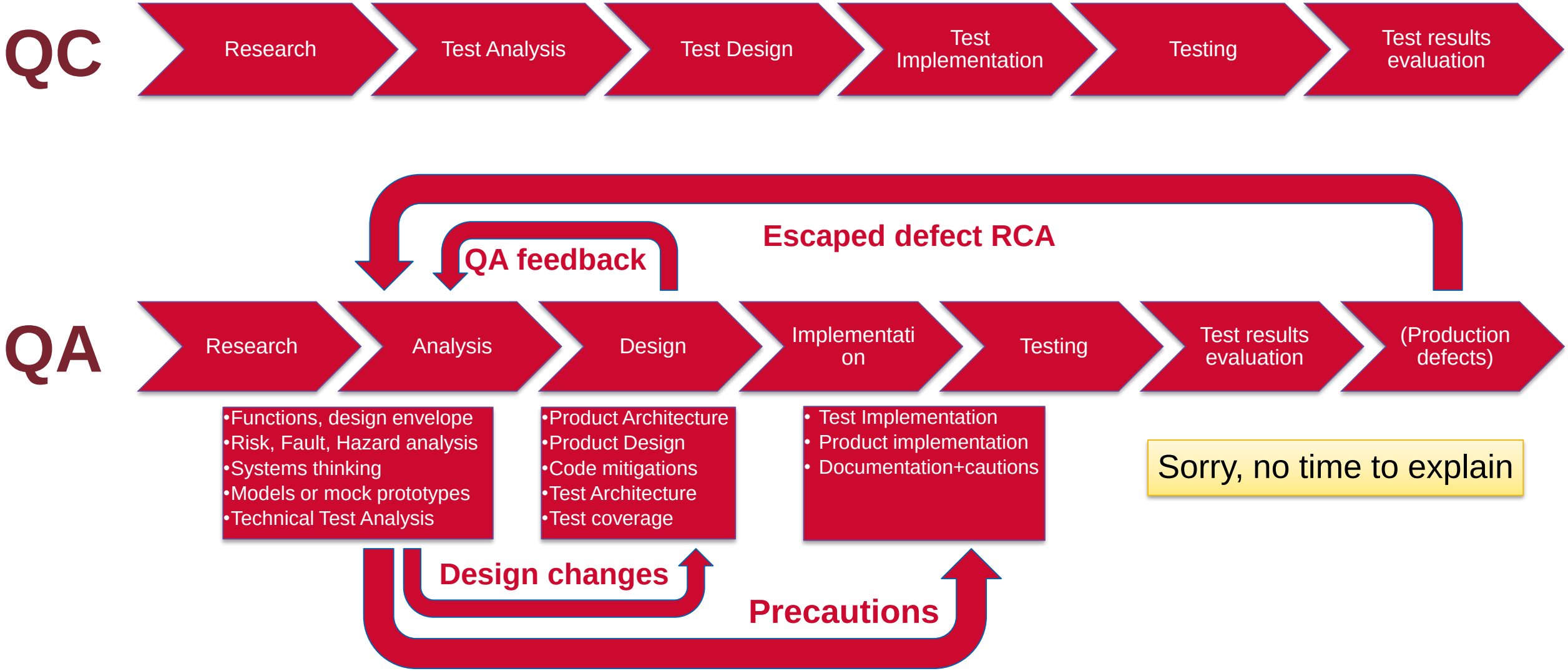
1. Targets design, architecture *and* built results
2. Analysis to identify what to redesign, improve
3. Challenges all assumptions
4. Focused also on inconspicuous functions, conditions, external influences, LPHI faults, near-boundary operations
5. Never ends, constrained only by resources
6. Reinforcing feedback loop („take the system and add/change X to increase quality“)





BROADCOM

QA is the feedback loops and design adjustments



BROADCOM

How to start with proper QC?

Allow and assign time for *conscious* QC activities

- Introduce QA research, TTA, Test Design and Test implementation stories/tasks

- Time allocation could be small. Point: *consciously* go through each phase
- Use basic TTA methods (BVA) Test-classes Cheatsheet to systematically discover *all* things which *need* to be covered by tests
- Also recognize there is a *difference* between test implementation (automation), execution and evaluation; estimate complexity of each phase during planning
- Introduce „*testing of tests*“
  - Use test classes checklists to verify no test area was forgotten = qualitative metric
  - Review how automated pass/fail criteria are evaluated, false negatives, false positives
  - Escaped defect: do RCA why didn‘t existing regr.tests catch it and how to improve them

BROADCOM

How to upgrade QC to aerospace-grade QA?

QA workflow – activities and artifacts

Function list:

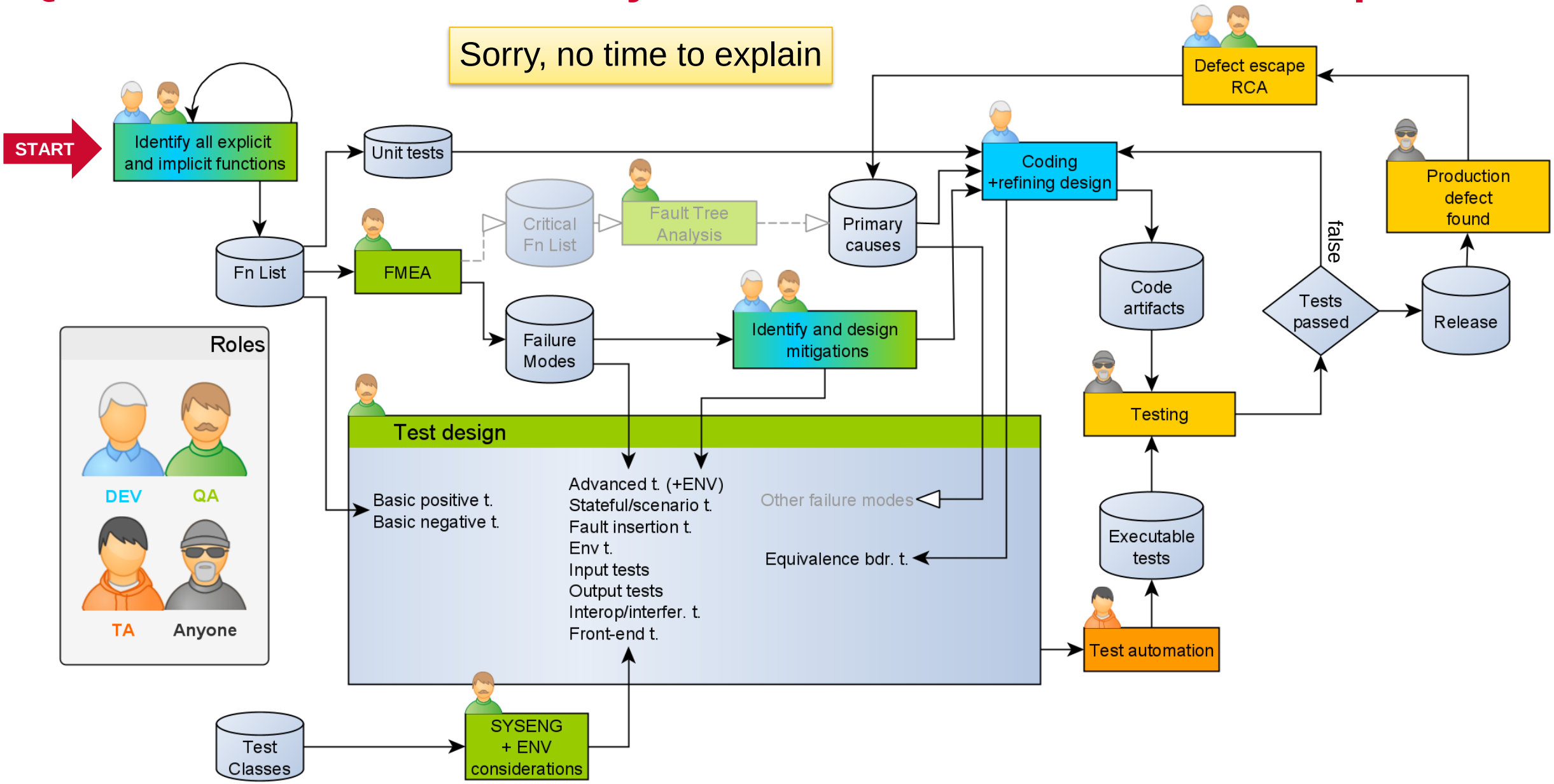
1. Explicit functions
   A. Authenticate users
2. Implicit functions
   A. Create users
   B. Encrypt password safely
   C. Create permissions system to admin users
   D. Create password reset option
3. To think thru
   A. Should we log auth violations?
4. Sanitizations/mitigations proposed
   A. Protect against roque brute force - add login attempt timeout?

FMEA:.

Component 1	Failure 1.1	Effect 1.1
	Failure 1.2	Effect 1.2
	Failure 1.M	Effect 1.M
Component 2	Failure 2.1	Effect 2.1
	Failure 2.2	Effect 2.2
	Failure 2.M	Effect 2.M
Component N	Failure N.1	Effect N.1
	Failure N.2	Effect N.2
	Failure N.M	Effect N.M

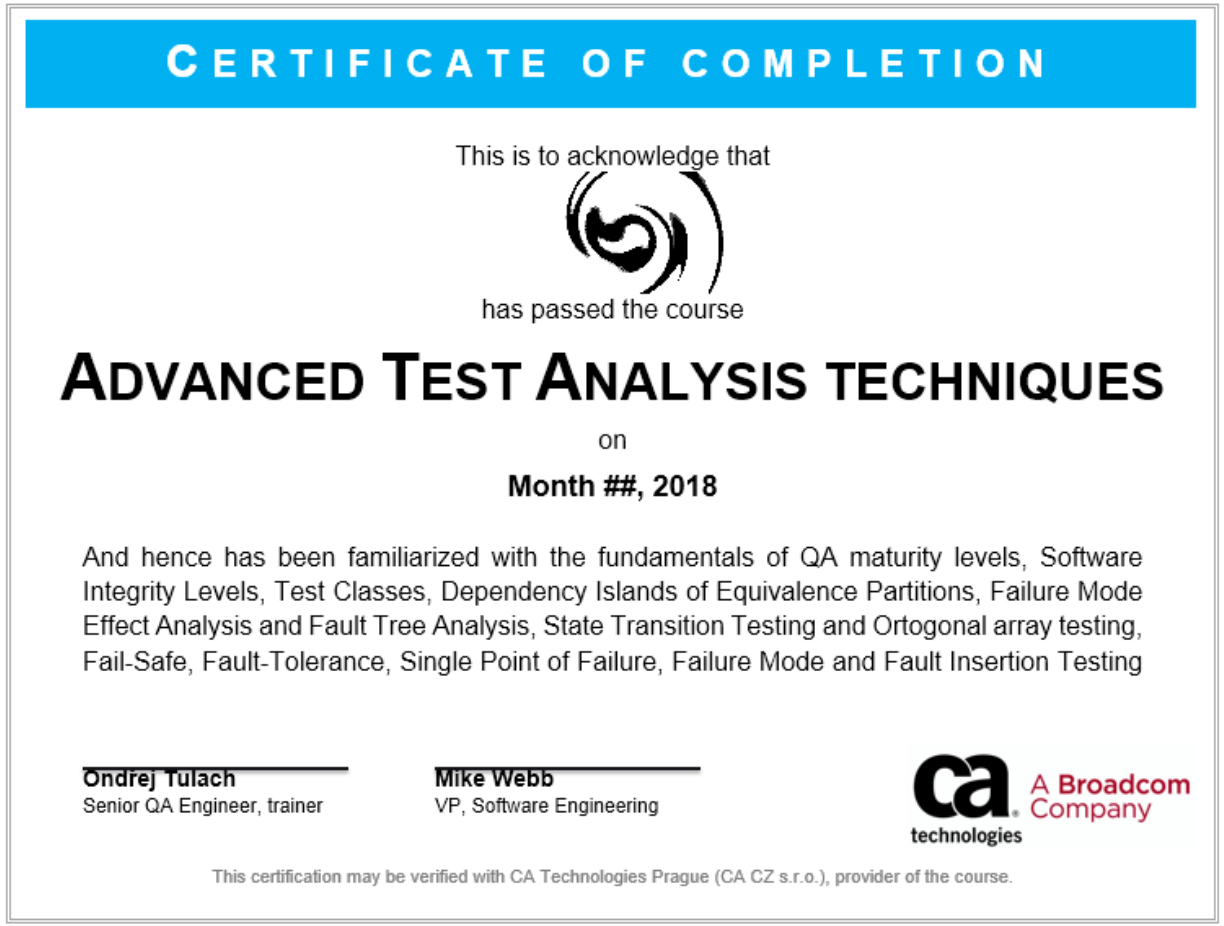
BROADCOM

QA workflow – across lifecycle and team with feedback loops



Expert TTA, RENG techniques: ask Zuzka Pruchova for course

- Failure modelling & risk analysis
  - FTA
  - FMEA
  - Error propagation modelling
  - SPoF, Fail-safe, Fault-tolerance
- Systems modelling
  - State Transition Testing
  - Data Flow Testing
- Lifecycle tools
  - Stress limits identification and sanitization
  - Installation, configuration, maintenance tests
- Equivalence classes refinement
  - EC Dependency islands
  - All-pairs testing with ortogonal arrays
- Specialized mindeset training
  - Divergent thinking
  - Systems thinking



CERTIFICATE OF COMPLETION

This is to acknowledge that

has passed the course

ADVANCED TEST ANALYSIS TECHNIQUES

on

Month ##, 2018

And hence has been familiarized with the fundamentals of QA maturity levels, Software Integrity Levels, Test Classes, Dependency Islands of Equivalence Partitions, Failure Mode Effect Analysis and Fault Tree Analysis, State Transition Testing and Ortogonal array testing, Fail-Safe, Fault-Tolerance, Single Point of Failure, Failure Mode and Fault Insertion Testing

Ondrej Tulach
Senior QA Engineer, trainer

Mike Webb
VP, Software Engineering

ca technologies A Broadcom Company

This certification may be verified with CA Technologies Prague (CA CZ s.r.o.), provider of the course.

BROADCOM

Summary

- Testing as „executing defined test instructions “ is tip of iceberg, mechanical job
- Test quality depends on Technical Test Analysis quality
  - **Ad hoc or improvised analysis = bad**
  - **Automation cannot replace analysis**, analyze risks, or define testing data
- Too much focus on test automation = focus on Execution away from Analysis
- Quality Assurance requires preventive risk analytics: FHA, FMEA or FTA
- Code coverage, path coverage, test № - all very misleading metrics
  - Use qualitative, not quantitative metrics – like Test Classes
- QC on L1:“just positive testing“ is dangerously insufficient
  - full L2:“destructive testing“ is QC minimum,
  - real quality starts on Level 3:“preventive QA“

BROADCOM

Q&A (&C)