TVVS – Software Testing Verification and Validation

Introduction
Acceptance Testing

Ana Paiva

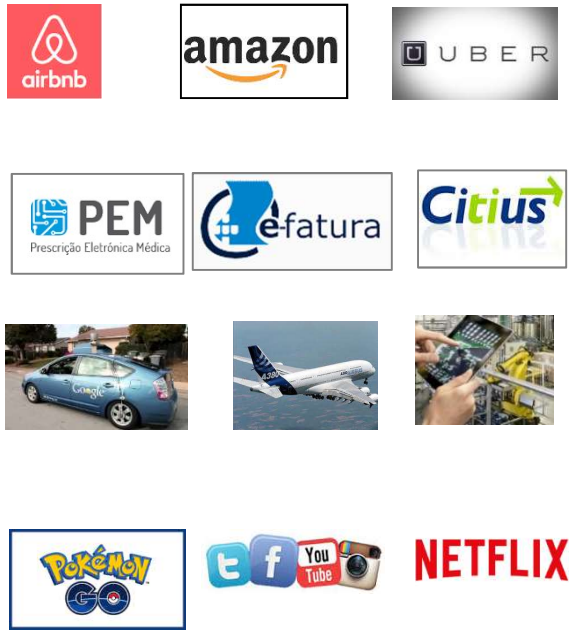
apaiva@fe.up.pt **www.fe.up.pt/~apaiva**

Agenda

- **Current status of software quality**
- Software Testing (introduction)
- Acceptance Testing
- Behavior Driven Development

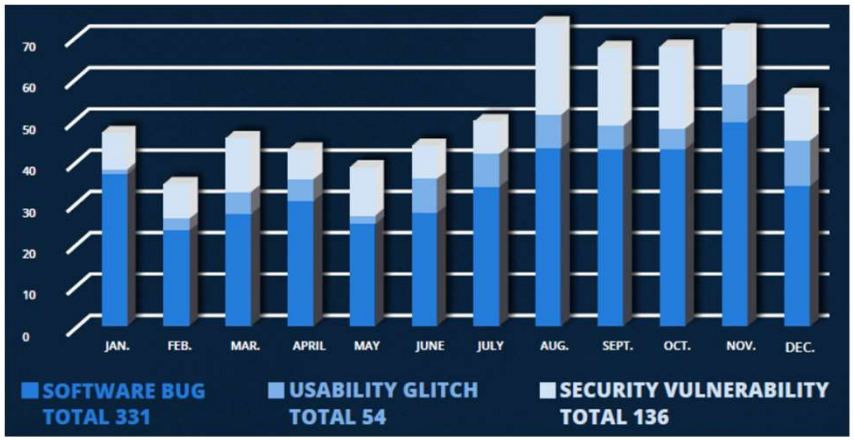
Software is Everywhere

- Enterprise applications & e-commerce
- Public services (healthcare, justice, …)
- Computer controlled systems (transport., I4.0, IoT,…)
- Media and entertainment



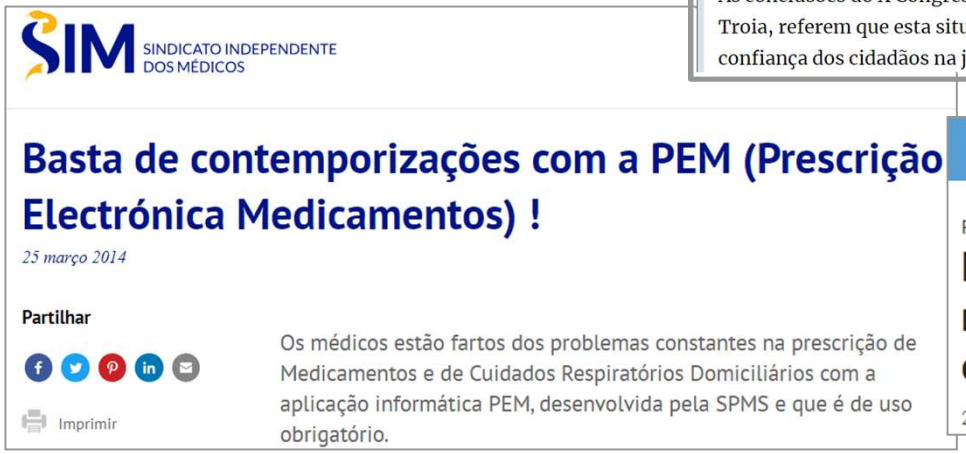
Software failures are everywhere: Software FAIL WATCH 2017

- Software failures are bigger than ever: 304 big cases analyzed in 2017
- Estimated financial losses: 1,7 trillion USD (2% world GDP)
- Estimated persons affected: 3,7 billion (50% world pop.)



https://www.tricentis.com/software-fail-watch/

Software failures are everywhere: examples in Public Services in Portugal



Software failures are everywhere: Famous Examples Worldwide

- Therac-25 (1985-1987)
  - 5+ patients died because of massive overdoses of radiation caused by a software error (race condition)

- Ariane 5 Explosion (1997)
  - $7 billion lost because of a software error (overflow)

- Air traffic control system in UK (2004-2018)
  - Software failures causes chaos in airports

- Blackberry blackouts (2009-2012)
  - Several service blackouts for several hours

More : http://spectrum.ieee.org/computing/software/why-software-fails

More bad exemples…

- Intel spend $475 m correcting a problema with floating point inside Pentium in1994 (Computer Science, Springer Verlag – 1995)
- PrimeCo Personal Communications canceled a contract of $500M with Motorola because of failures (Wall Street Journal – 24/02/98)
- Time Warner Communications spend $1B in information systems to enter the residential business of the telephone network (Computerworld – 05/05/97)
- National Bank of Australia lost $1,75B due to na errors not detected during 2 years (New York Times – Nov/01)
- Ariane 5 (10 years of development $7B) with a charge of $ 500M, exploded 40 seconds after launch. Software module generated untreated event (ESA - 1996)
- Therac-25 provided incorrect doses of X-rays in patients between 1985 and 1987 - 6 deaths (IEEE Computer - 07/07/93)
- More and constant updates at **http://www.risks.org**

The current status of software quality (1)

- Microsoft Windows XP End-User License Agreement:

  11. LIMITED WARRANTY FOR PRODUCT ACQUIRED IN THE US AND CANADA.

  Microsoft warrants that the Product will perform substantially in accordance with the accompanying materials for a period of ninety days from the date of receipt.
  (…)

  YOUR EXCLUSIVE REMEDY. Microsoft's and its suppliers' entire liability and your exclusive remedy shall be, at Microsoft's option from time to time exercised subject to applicable law, (a) **return of the price paid** (if any) for the Product, or (b) **repair or replacement** of the Product, that does not meet this Limited Warranty and that is returned to Microsoft with a copy of your receipt.
  (..)

The current status of software quality (2)

	India	Japan	US	Europe & other	Total
Number of projects	24	27	31	22	104
Median output[1]	209	469	270	436	374
Median defect rate[2]	.263	0.020	.400	.225	.150

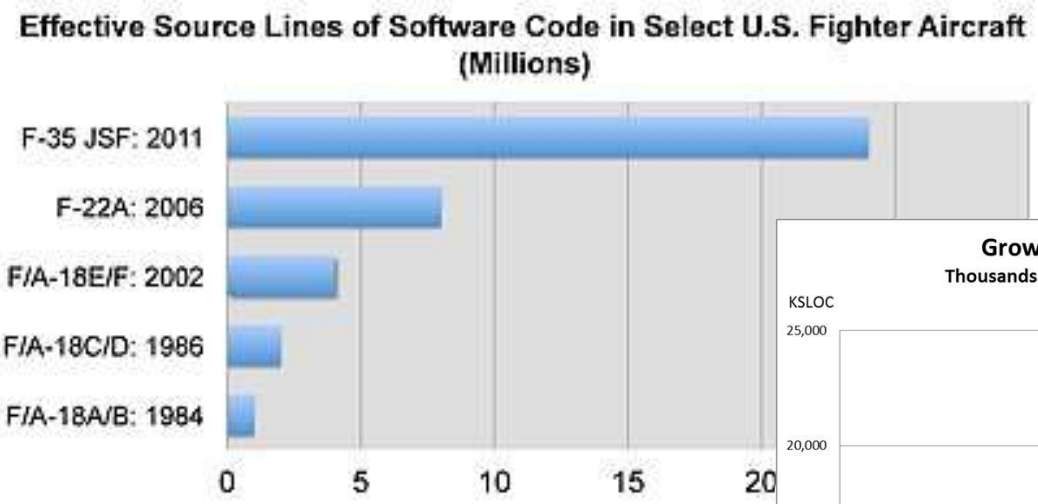
[1] (new?) LOC / programmer-month (considering the whole life cycle)

[2] Number of defects/ KLOC reported by customers in the first year post delivery

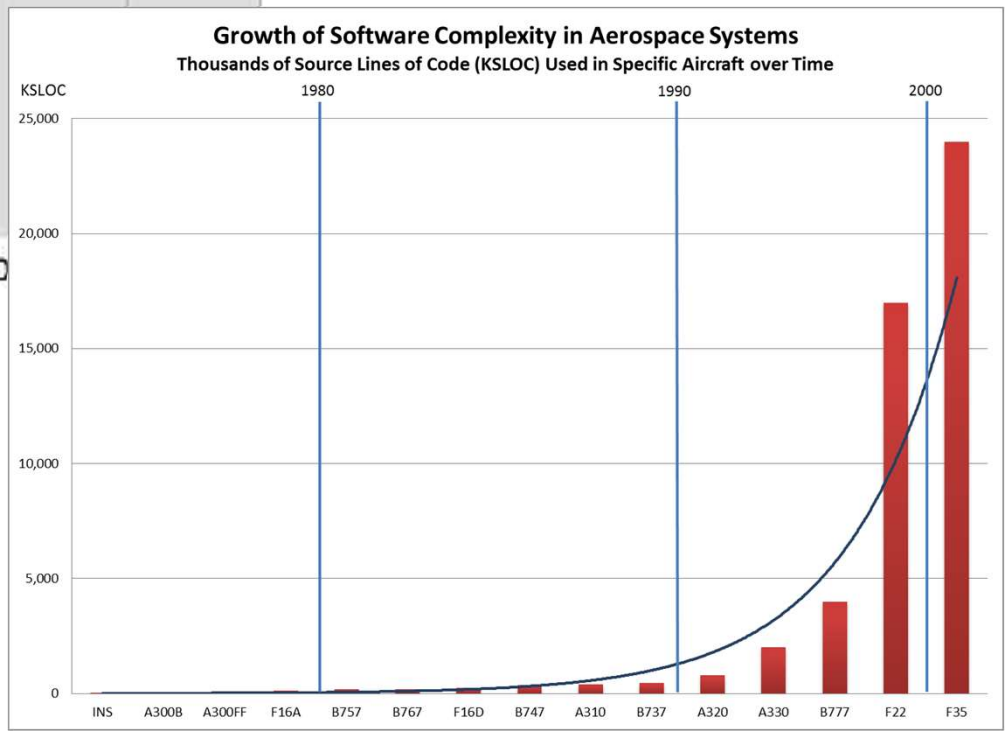
[source: "Software Development Worldwide: The State of the Practice", M. Cusumano (MIT), A. MacCormack (Harvard Univ.), C. F. Kemerer (Pittsburgh Univ.), B. Crandall (HP), IEEE SOFTWARE, 2003]

The importance of software quality (1)

We depend more and more on software …



Effective Source Lines of Software Code in Selected U.S. Fighter Aircraft



Growth of Software Complexity in Aerospace Systems

The importance of software quality (2)

- Software size increases by a factor of 10 every 10 years …
  - 50 KLOC - Word 3.0 for DOS
  - 1 MLOC - Unix, System V, Release 4, 1990
  - 10 MLOC – Linux, 2000
  - 50 MLOC - Windows Vista, 2007

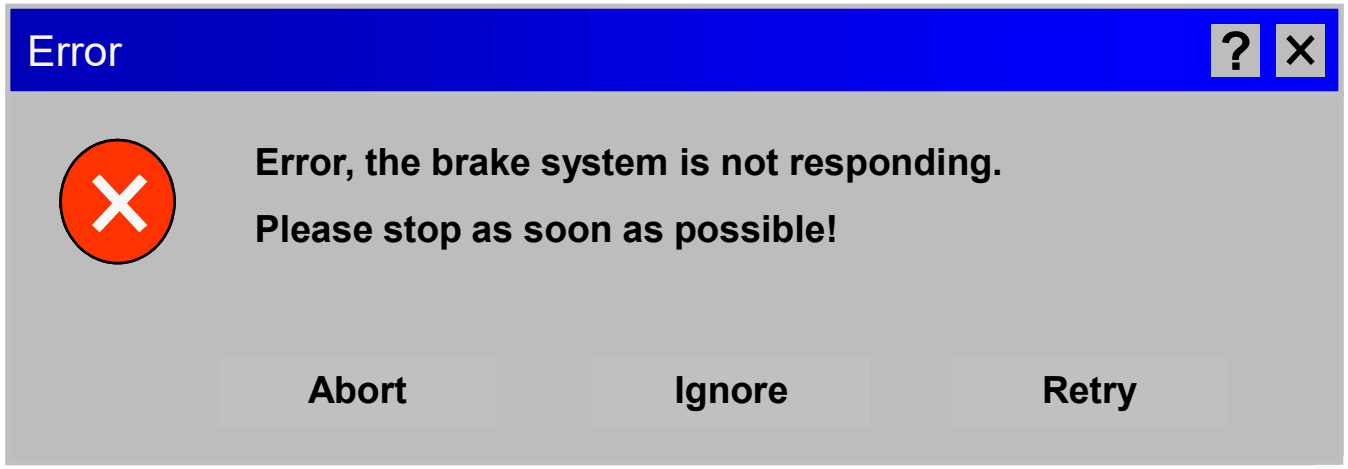
- 50 MLOC * 1 defect/KLOC = 50.000 defects in Windows Vista?

- Ideally: 1 defect / MLOC

KLOC = 1.000 lines of code
MLOC = 1.000.000 lines of code

The importance of software quality (3)

Impact on life and environment …



Tim Davis, Ford Motor Company, 27th International Conference on Software Engineering, 2005

Economic impact …

- According to the National Institute of Standards and Technology (NIST), USA, direct costs of software error represent 0,6 % of GNP (PIB) in the USA

  [source: "The Economic Impacts of Inadequate Infrastructure for Software Testing", NIST, May 2002]

The Need for Testing

- Even experienced developers inject on average one defect per 10 lines of code they write
- Current software systems have millions of lines of code, so thousands of defects are injected during development
- Such large numbers of defects injected have to be removed by applying a combination of “filters”:
  - Static analysis (applied mostly to code, by compilers and specific analysis tools)
  - Reviews and inspections (applied to req. specs, design specs, models & code)
  - Animation and simulation (applied mostly to models)
  - (Dynamic) Testing (at the unit, integration, system & acceptance levels)

Software Testing is Extremely Challenging

- Since exhaustive testing is impossible, how can we generate a set of test cases of manageable size that has a high probability of detecting most of the defects?

- How can we automate the test process (test generation & test execution) in order to increase the efficiency (effort per defect found) and effectiveness (percentage of defects found) of the test process?

Why quality pays?

- Poor-quality software can be life-threatening
  - Or mission/business/environment/economy-threatening …
- Quality work saves time and money
  - Defect density decreases 10 while productivity increases
- Quality work is more predictable
  - The testing and repair effort of a bad quality product is unpredictable

[Source: Watts Humphrey, “Winning with Software”, 2002]

Quality costs

- Costs of **conformance**
  - All costs associated with planning and running tests (and revisions) just one time
- Costs of **nonconformance**
  - Costs due to **internal** failures (before release)
    - Cost of isolating, reporting and regression testing bugs (found before the product is released) to assure that they're fixed (left-hand side of fig. 1.2)
  - Costs due to **external** failures (after release)
    - If bugs are missed and make it through to the customers, the result will be costly product support calls, possibly fixing, retesting, and releasing the software, and – in a worst case-scenario – a product recall or lawsuits (right-hand side of fig. 1.2)

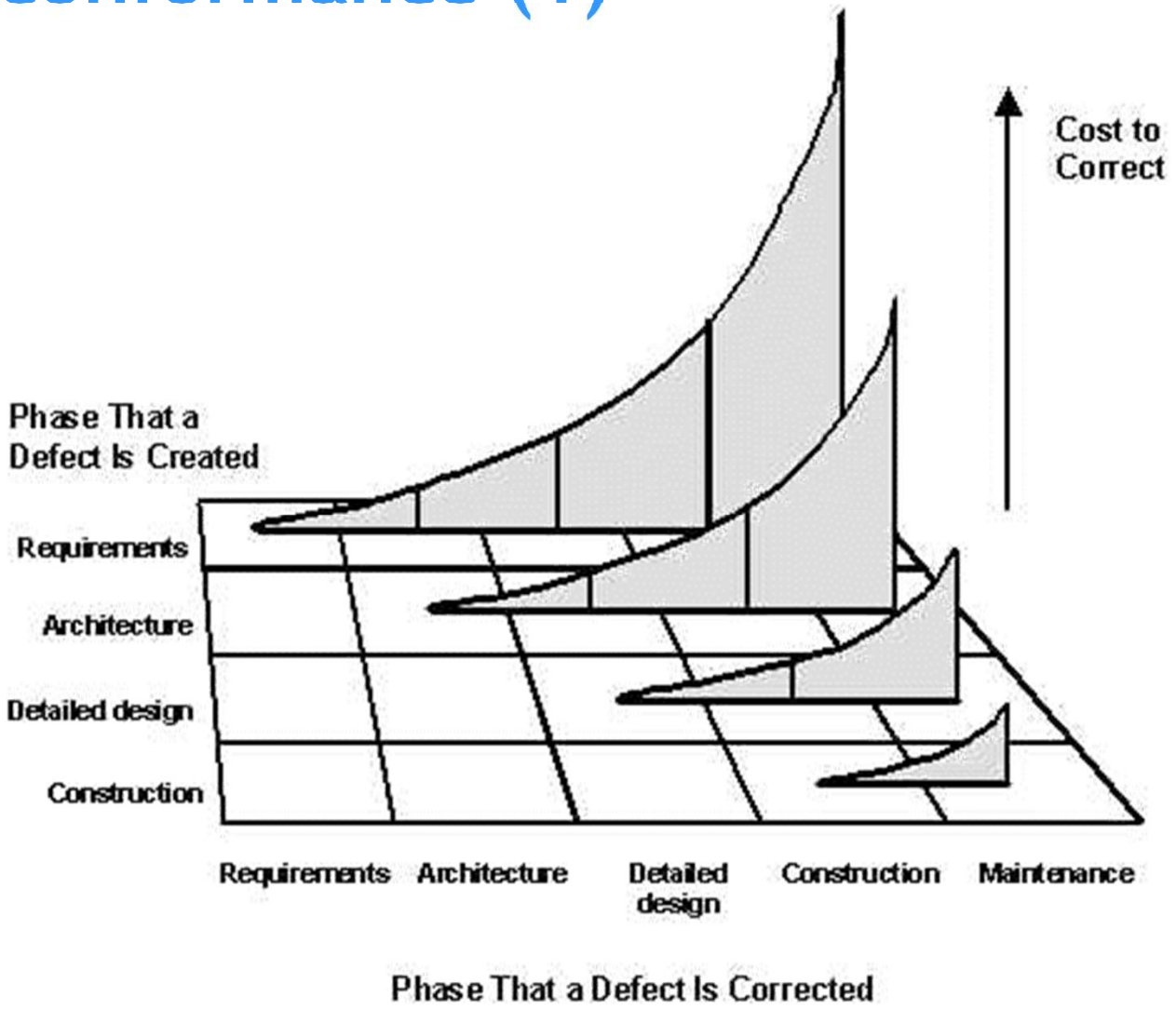
[source: "Software Testing", Ron Patton]

Quality costs
Costs of nonconformance (1)

If a defect is created in the early stages of the development life cycle and corrected in the later stages, it will be very expensive to correct.

If a defect is created and removed in the same phase, it won't be too expensive to fix.

The longer the bug remains, the more expensive it will be to fix.



Quality costs
Quality is free!?

- In his book "Quality is Free: The Art of Making Quality Certain", Philip Crosby argues that the costs of conformance plus the costs of nonconformance due to internal failures is (usually) less than the costs of nonconformance due to external failures

costs of conformance
+
costs of nonconformance due to internal failures

<

costs of nonconformance due to external failures

[source: Ron Patton, "Software testing“]

Which defects are more frequent?

IBM defect data:

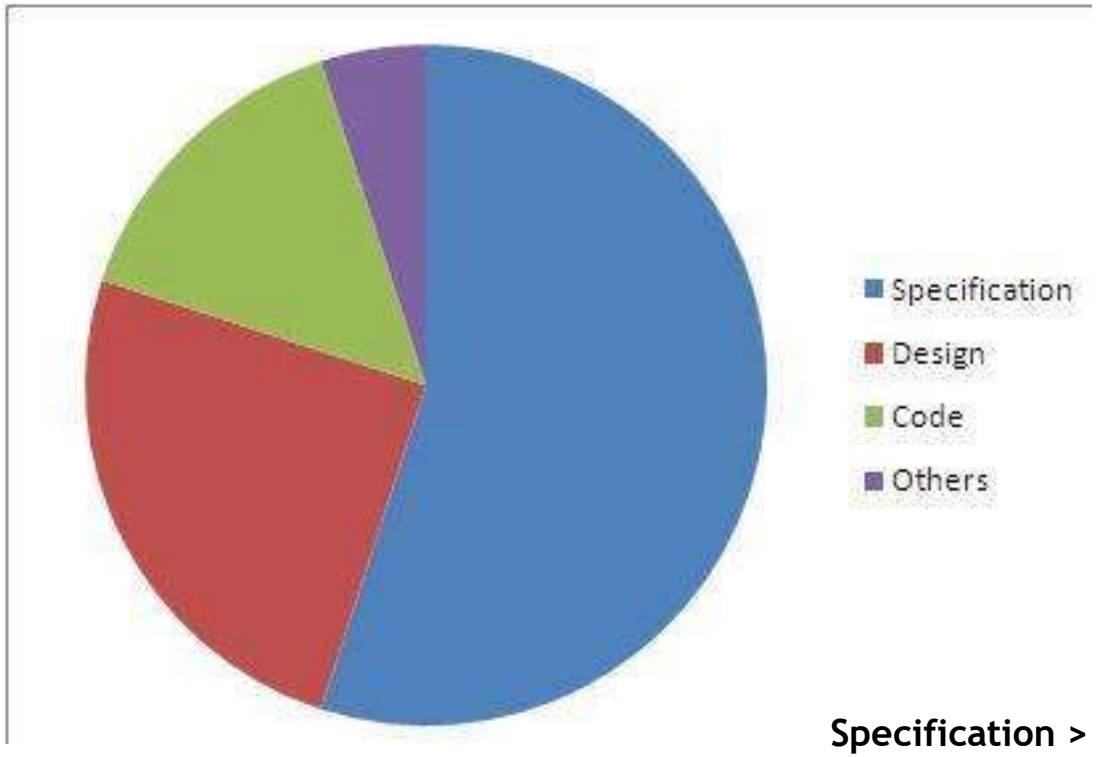
Classificação ODC (*)	Description	Frequency
Algorithm	"execução incorrecta ou em falta que pode ser corrigida sem ser necessário introduzir alterações arquitecturais no software"	43.4 %
Assignment	"valores incorrectamente atribuídos ou não atribuídos"	22.0 %
Checking	"validação de dados incorrecta ou expressões condicionais incorrectas"	17.5 %
Function	"falha que afecta uma quantidade considerável de código e refere-se a uma capacidade do software que está em falta ou construída incorrectamente"	8.7 %
Interface	"interacção incorrecta entre módulos/componentes"	8.2 %

(*) Orthogonal Defect Classification (ODC)

Useful for review check-lists and fault based testing!

[source:Henrique Madeira, Universidade de Coimbra]

Main sources of defects



Specification > Design > Code > Others

So,
take more time to get and understand specifications!

Agenda

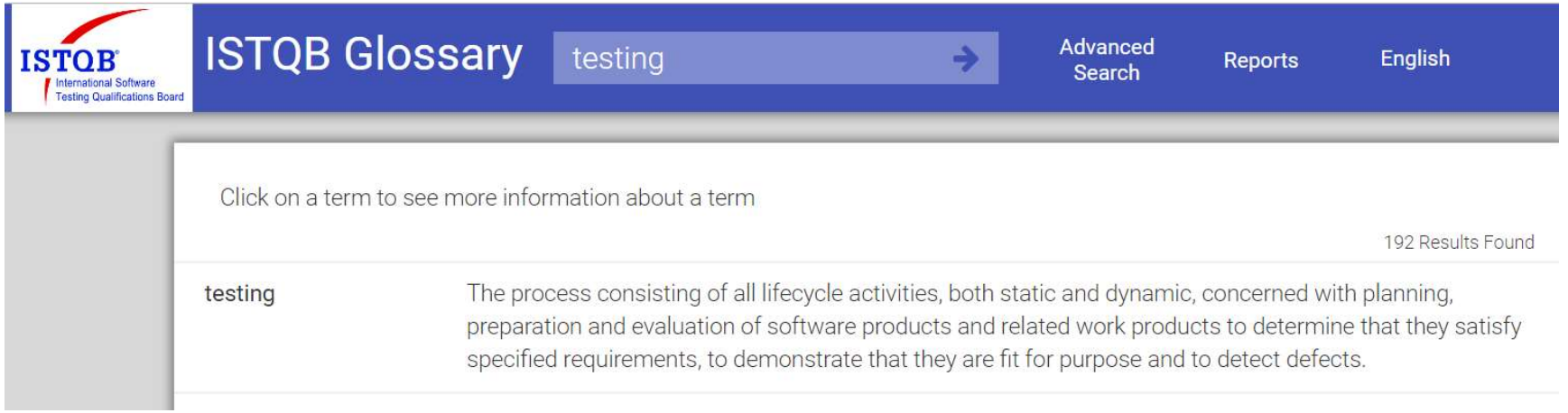
- Current status of software quality
- **Software Testing (introduction)**
- Acceptance Testing
- Behavior Driven Development

Static vs dynamic

- **static testing:** Testing of a software development artifact, e.g., requirements, design or code, without execution of these artifacts, e.g., reviews or static analysis.
- **dynamic testing:** Testing that involves the execution of the software of a component or system.

[http://glossary.istqb.org/]

Software Testing - ISTQB



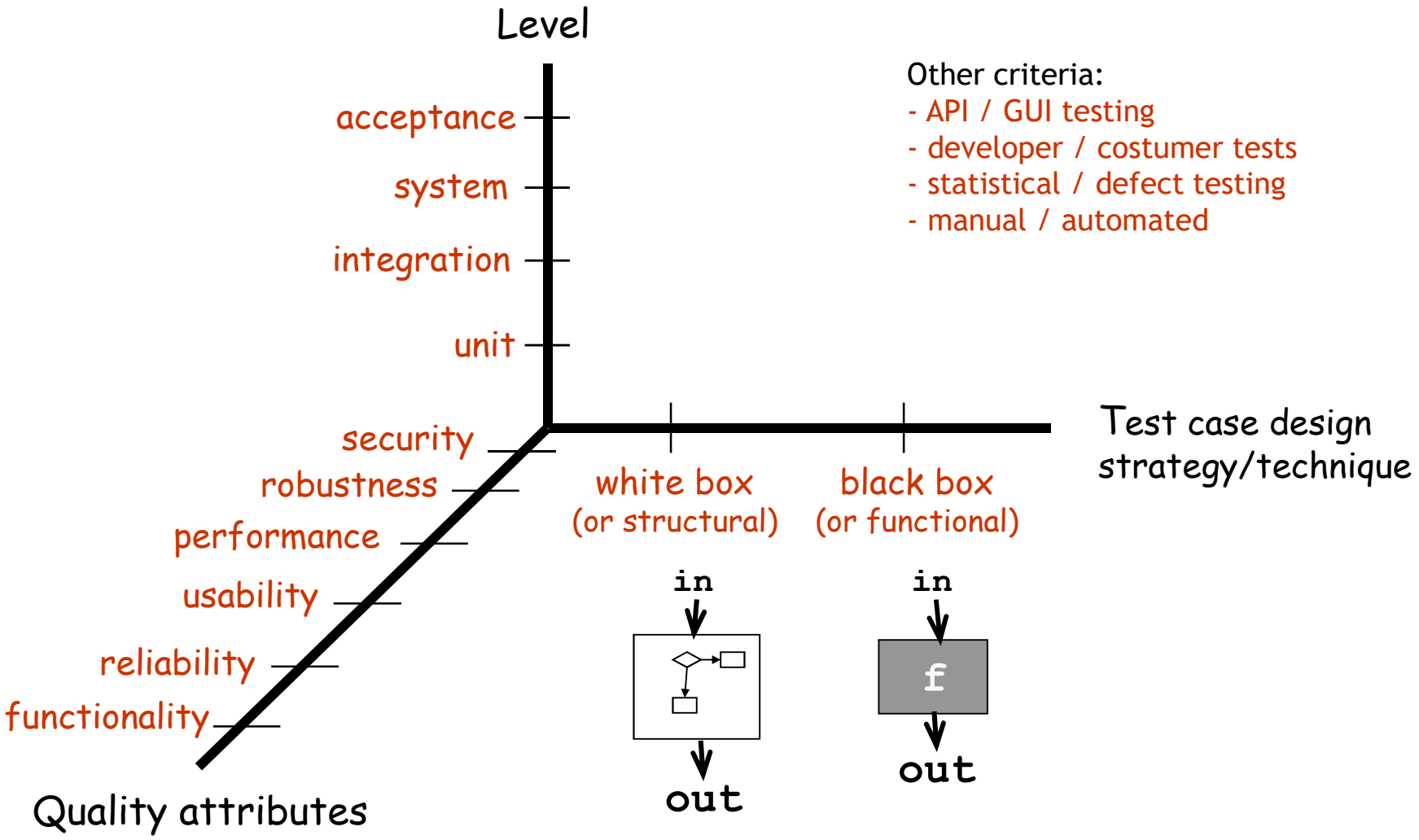
- The process consisting of all lifecycle activities, **both static and dynamic**, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

[http://glossary.istqb.org/]

Purpose of software testing

- **“Program testing can be used to show the presence of bugs, but never to show their absence!”** [Dijkstra, 1972]
  - … because exhaustive testing is usually impossible
- **The primary goal** of software testing is **to find failures**
  - "The goal of a software tester is to find “bugs”, find them as early as possible, and make sure that they get fixed“ [Ron Patton]
- **A secondarily goal** is to **increase the confidence** on the software correctness and to assess software quality
  - **statistical testing**: A test design technique in which a model of the statistical distribution of the input is used to construct representative test cases.

Test types



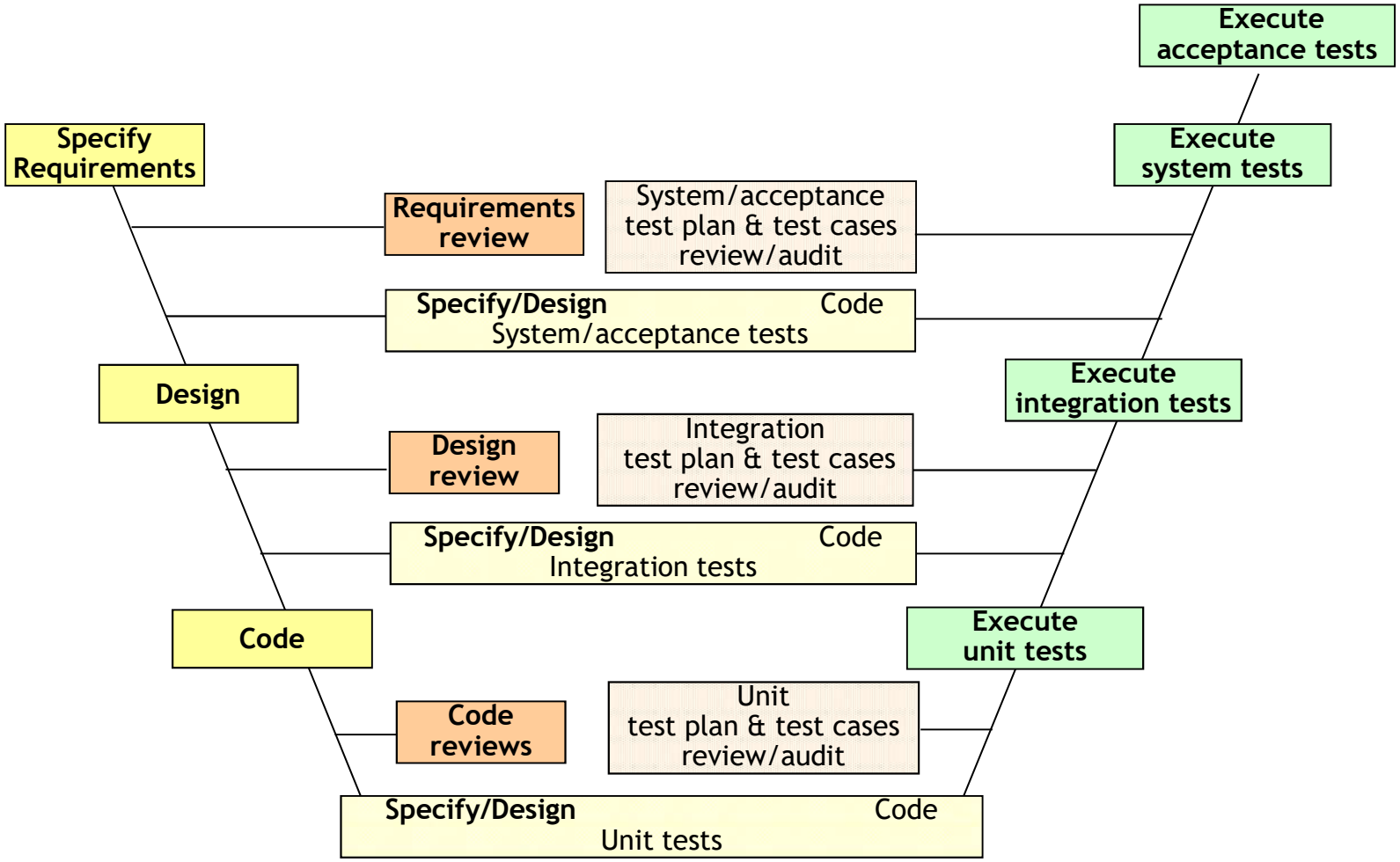
Other criteria:
- API / GUI testing
- developer / costumer tests
- statistical / defect testing
- manual / automated

Test Types

- Classified according to:
  - Level: acceptance, system, integration, unit
  - Quality attributes: functionality, reliability, usability, performance. Robustness, security,…
  - Test case design strategy: white box (or structural) and back box (or functional)
- Other criteria
  - API / GUI
  - Developer /customer tests
  - Statistical / defect testing
  - Manual /automated

V-model

Testing of the product is planned in parallel with a corresponding phase of development in V-model.



The extended V-model of software development [I.Burnstein]

Test levels – Unit Testing

- Testing of individual (hardware or) software units or groups of related units [IEEE Standard Glossary of Software Engineering Terminology 610.12-1990]
- Component testing: the testing of individual software components [http://glossary.istqb.org/]
- Usually API testing
- Usually the responsibility of the developer -> developer tests
- Tests are usually based on experience, specifications and code
- A principal goal is to detect functional and structural defects in the unit

Test levels – Integration testing

- Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them.
  - [IEEE Standard Glossary of Software Engineering Terminology 610.12-1990]
- Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.
  - [http://glossary.istqb.org/]

Test levels – Integration testing

- Usually the responsibility of an independent test team (except sometimes in small projects)
- Tests are usually based on a system specification (technical specifications, designs)
- A principal goal is to detect defects that occur on the interfaces of units (interaction testing)

Test levels – System testing

- Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
  - [IEEE Standard Glossary of Software Engineering Terminology 610.12-1990]
- Testing an integrated system to verify that it meets specified requirements.
  - [http://glossary.istqb.org/]

Test levels – System testing

- Usually GUI testing
- Usually the responsibility of an independent test team
- Tests are usually based on a requirements document
- The goal is to ensure that the system performs according to its requirements, by evaluating both functional behavior and quality requirements such as reliability, usability, performance and security

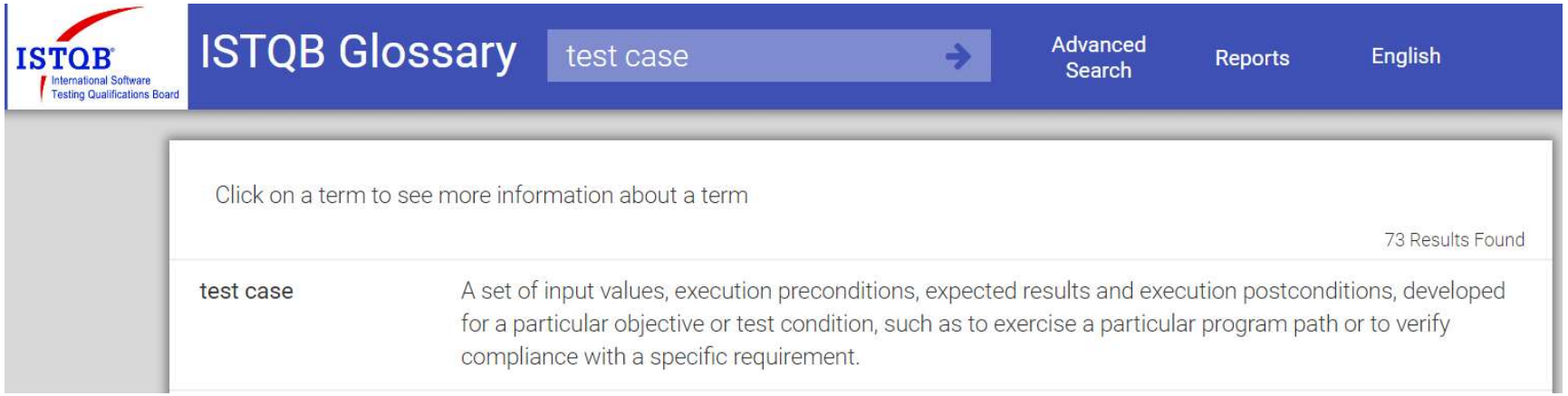
Test levels – Acceptance testing

- (1) Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. (2) Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component.
  - [IEEE Standard Glossary of Software Engineering Terminology 610.12-1990]
- Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.
  - [http://glossary.istqb.org/]

Test levels – Acceptance testing

- Usually the responsibility of the customer -> customer tests
- Tests are usually based on a requirements document or a user manual
- A principal goal is to check if customer requirements and expectations are met

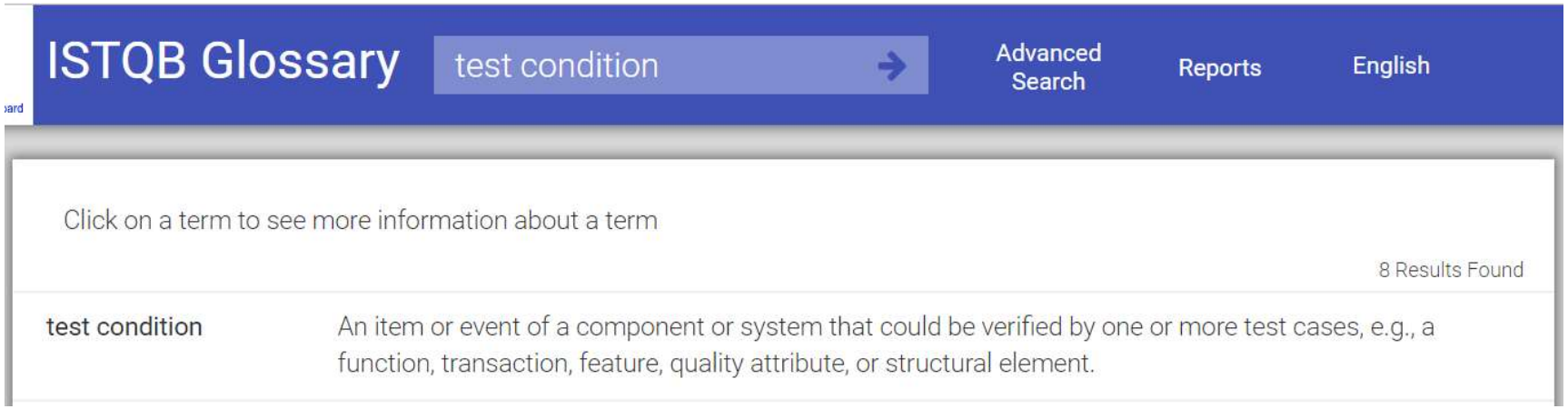
Test case



- A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

[http://glossary.istqb.org/]

Test Condition



- An item or event of a component or system that could be verified by one or more test cases, e.g., a function, transaction, feature, quality attribute, or structural element.

[http://glossary.istqb.org/]

What is a good test case?

- Capability to find defects
  - Particularly defects with higher risk
  - Risk = frequency of failure * impact of failure ≈ cost (of post-release failure)
- Capability to exercise multiple aspects of the system under test
  - Reduces the number of test cases required and the overall cost
- Low cost
  - Development: specify, design, code
  - Execution (fast)
  - Result analysis: pass/fail analysis, defect localization
- Easy to maintain
  - Reduce whole life-cycle cost
  - Maintenance cost ≈ size of test artefacts

(See also: “What Is a Good Test Case?”, Cem Kaner, Florida Institute of Technology, 2003)

Test cases

- The **best test cases** are the ones that
  - have higher probability of revealing defects,
  - particularly defects with higher impact
  - particularly defects that occur in the most frequently used features of the system

Test adequacy/coverage criteria

- **Adequacy criteria** - Criteria to decide if a given test suite is adequate, i.e., to give us "enough" confidence that "most" of the defects are revealed
  - Used in the evaluation and in the design/selection of test cases
  - In practice, reduced to coverage criteria
- **Coverage criteria**
  - Requirements/specification coverage (black-box)
    - At least one test case for each requirement/specification statement
  - Code coverage (white-box)
    - Control flow coverage (statement, decision, MC/DC coverage …)
    - Data flow coverage
  - Model coverage
    - State-transition coverage
    - Use case and scenario coverage
  - Fault coverage

See also: "Software Unit Test Coverage and Adequacy", Hong Zhu et al, ACM Computing Surveys, December 1997

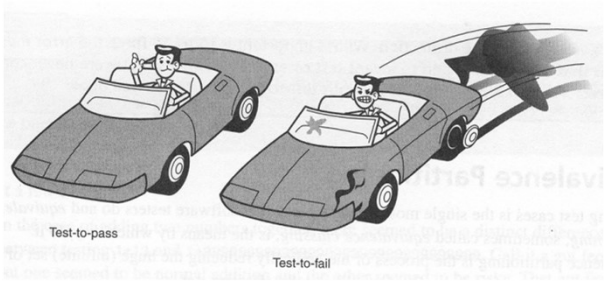
Testing best practices (1)

- Test as early as possible
- Automate test case execution $\Rightarrow$ JUnit/NUnit, etc.
  - because of the frequent need for regression testing (repetition of testing each time the software is modified)
- Write the test cases before the software to be tested $\Rightarrow$ TDD; FIT
  - applies to any level: unit, integration or acceptance/system
  - helps getting insight into the requirements
  - test cases are verifiable partial specifications
- The more critical the system the more independent should be the tester
  - peer, other department, other company (*)

  (*) ISVV - Independent Software V&V - assures technical, managerial and financial independence
- Be conscious about cost

Testing best practices (2)

- Use test cases to objectively measure project progress
- Combine tests with reviews
- Start to design test cases based on the specification (black-box) and subsequently refine to cover the code (white box)
- Test-to-pass (1) in the first test iterations, and test-to-fail (2) in subsequent test iterations
  - (1) check if the software fundamentally works, with valid input, without stressing the system
  - (2) try to "break" the system, with valid inputs but at the operational limits or with invalid inputs

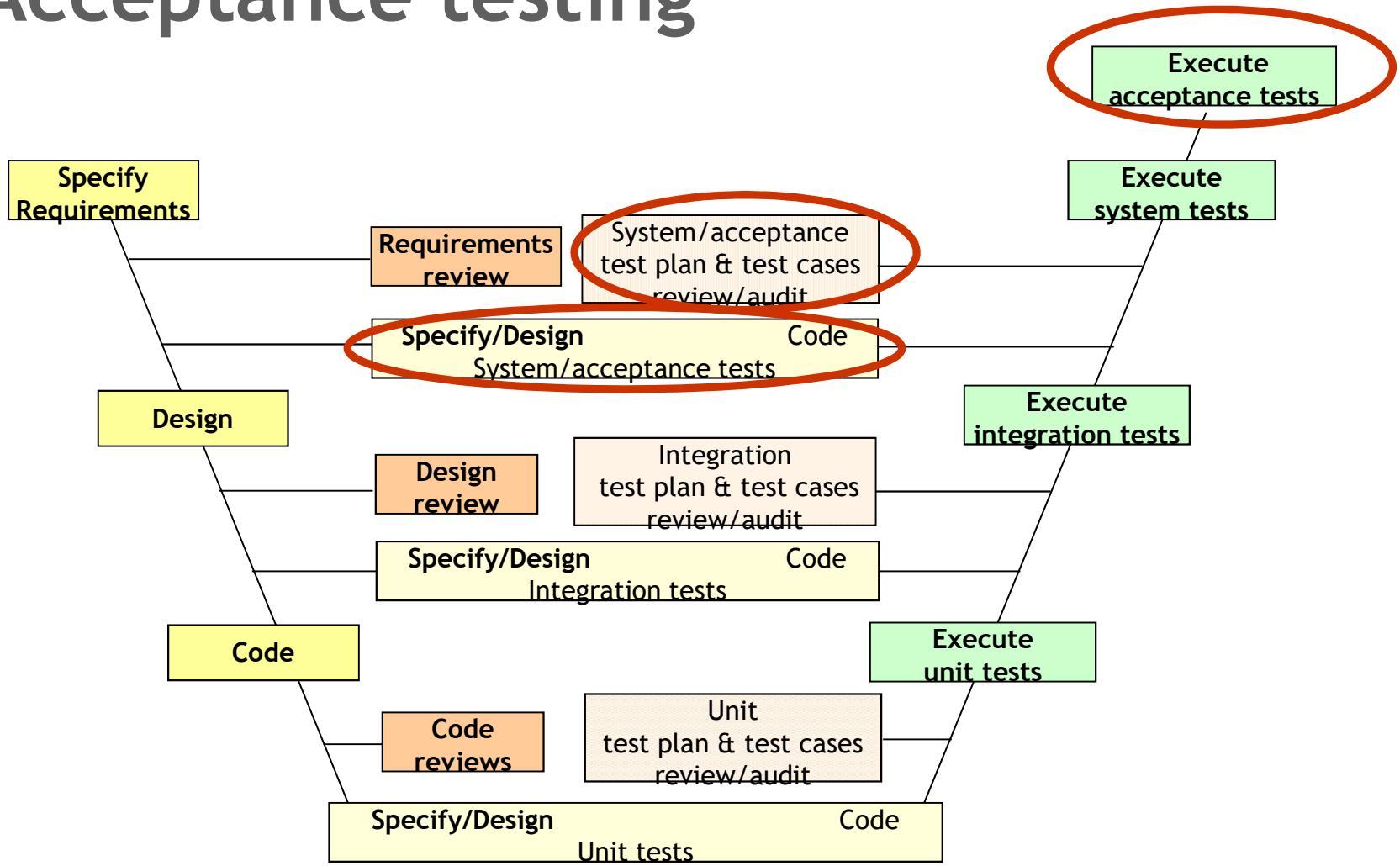


[source: Ron Patton]

Agenda

- Current status of software quality
- Software Testing (introduction)
- **Acceptance Testing**
- Behavior Driven Development

Acceptance testing



The extended V-model of software development [I.Burnstein]

Acceptance testing

- **acceptance testing**: Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system. [After IEEE 610]

[ISTQB]

Acceptance testing

- Different types of acceptance testing:
  - User Acceptance Testing (UAT)
  - Alpha testing
  - Beta testing
  - Site acceptance testing
  - Operational acceptance testing
  - Production acceptance testing

Acceptance testing

- **Operational acceptance testing** Operational testing in the acceptance test phase, typically **performed in a (simulated) operational environment** by operations and/or systems administration staff focusing on operational aspects, e.g., recoverability, resource-behavior, installability and technical compliance. *See also operational testing*.
- **Production acceptance testing** *See operational acceptance testing*.

[ISTQB]

Acceptance testing

- **Alpha testing**: Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site, but outside the development organization. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing.
- **Beta testing**: Operational testing by potential and/or existing users/customers at an external site not otherwise involved with the developers, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes. Beta testing is often employed as a form of external acceptance testing for off-the-shelf software in order to acquire feedback from the market.

[ISTQB]

Acceptance testing

- **Site acceptance testing**: Acceptance testing by users/customers at their site, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes, normally including hardware as well as software.

[ISTQB]

Smoke test

- A subset of all defined/planned test cases that **cover the main functionality** of a component or system, to ascertaining that the most crucial functions of a program work, but not bothering with finer details.

[ISTQB]

Acceptance testing

- **acceptance criteria**: The exit criteria that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity. [IEEE 610]

[ISTQB]

Acceptance criteria

The acceptance criteria are defined on the basis of the following attributes:

- **Functional Correctness and Completeness**
- **Accuracy**
- **Data Integrity**
- **Data Conversion**
- **Backup and Recovery**
- **Competitive Edge**
- **Usability**
- **Performance**
- **Start-up Time**
- **Stress**
- **Reliability and Availability**
- **Maintainability and Serviceability**
- **Robustness**
- **Timeliness**
- **Confidentiality and Availability**
- **Compliance**
- **Installability and Upgradability**
- **Scalability**
- **Documentation**

Acceptance Testing in eXtreme Programming

- In XP framework the user stories may be used as acceptance criteria
- The user stories are written by the customer as things that the system needs to do for them
- Several acceptance tests may be created to verify the user story has been correctly implemented
- The customer is responsible for verifying the correctness of the acceptance tests and reviewing the test results
- A story is incomplete until it passes its associated acceptance tests
- Ideally, acceptance tests should be automated, either using the unit testing framework, before coding
- The acceptance tests take on the role of regression tests

Acceptance testing tools

- Framework for Integrated Test (*Fit*)
- FitNesse
- Behat
- Lettuce
- Cucumber
- ...

Agenda

- Current status of software quality
- Software Testing (introduction)
- Acceptance Testing
- **Behavior Driven Development**

BDD – Behavior Driven Development

- Behavior Driven testing is an extension of TDD. Like in TDD, in BDD we also write tests first and then add application code. The major difference that we get to see here are
  - *Tests are written in plain descriptive English type grammar*
  - *Tests are explained as behavior of application and are more user focused*
  - *Using examples to clarify requirements*
- This difference brings in the need to have a language which can define behavior in an understandable format.

BDD – Behavior Driven Development

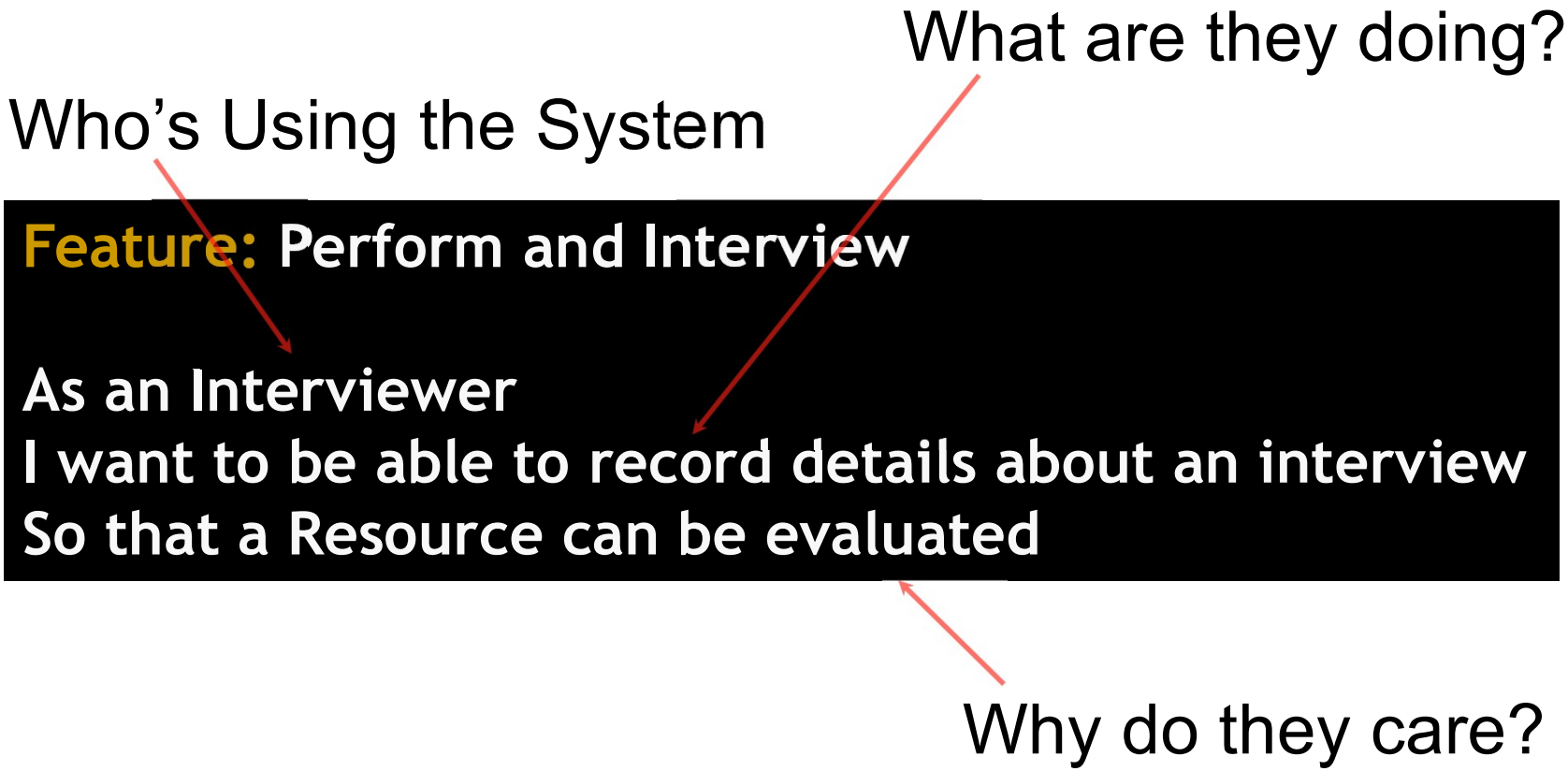
- Shifting from thinking in “tests” to thinking in “behavior”
- Collaboration between Business stakeholders, Business Analysts, QA Team and developers
- Ubiquitous language, it is easy to describe
- Driven by Business Value
- Extends Test Driven Development (TDD) by utilizing natural language that non technical stakeholders can understand
- BDD frameworks such as Cucumber or JBehave are an enabler, acting a “bridge” between Business & Technical Language

BDD – Behavior Driven Development

- Features
  - Who's using the system?
  - What are they doing?
  - Why do they care?
    - As a <role>
    - I want <feature>
    - So that <business value>
- Features are defined by one or more scenarios
  - Sequence of steps thru the feature that exercises on path
- Use BDD style – given-when-then
    - Scenario: <description>
    - <step 1>
    - …
    - <step 2>

BDD – Behavior Driven Development

FEATURES



BDD – Behavior Driven Development

SCENARIOS

- **Given** - Sets up preconditions, or context, for the scenario
- **When** - The action, or behavior, that we're focused on
- **Then** - Checks post-conditions and verifies that the right thing happened in the When stage

```
Scenario: Assign a candidate to an opportunity
  Given a user is logged in as "joe"
  And seeded users are created
  And seeded opportunities are created
  And I am on NewOpportunity1's opportunity edit page
  When I select "smith5" from "opportunity_resources_"
  Then I should see "success"
```

BDD (example: CUCUMBER - GHERKIN)

Feature: Google Searching

As a web surfer, I want to search Google, so that I can learn new things.

Scenario Outline: Simple Google searches

Given a web browser is on the Google page

When the search phrase "<phrase>" is entered

Then results for "<phrase>" are shown

And the related results include "<related>"

Examples: Animals

phrase	related
panda	Panda Express
elephant	Elephant Man

References and further reading

- http://glossary.istqb.org/
- www.junit.org
- www.nunit.org
- http://testdriven.net/
- [IEEE 90] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
- "The art of unit testing – With Examples in .NET", by Roy Osherove, ISBN: 1933988274
- xUnit Test Patterns, Gerard Meszaros, Addison-Wesley, 2007
- http://www.onestoptesting.com
- Test Driven Development, Kent Beck, Addison Wesley, 2002
- Test-Driven Development in Microsoft.Net, James W. Newkirk, Alexei A. Vorontsov, Microsoft Professional, 2004
- IEEE Software, May/June 2007 (Vol. 24, N. 3), Special issue on TDD
- http://msdn.microsoft.com/en-us/library/aa730844.aspx#guidelinesfortdd_topic3
- "Extreme Programming in the Real World - Java Extreme Programming Cookbook", by Eric M. Burke & Brian M. Coyner, O'Reilly
- http://www.umsl.edu/~sauterv/analysis/f06Papers/Hutagalung/#xp
- http://www.geocities.com/xtremetesting/TestingDictionary.html
- Lean, Agile, and Extreme Programm, By Mark Windholtz, ObjectWind Software Ltd, in the RailsStudio.com
- "Extreme Programming Explained: Embrace Change", By Kent Beck, Published by Addison-Wesley, 2000
- http://christopher-steiner.com/html/?p=37