Software Testing

Dániel Varró ♦ McGill University
daniel.varro@mcgill.ca

Based on: D. Galin ch.9.1,10.3 and R. Patton ch.3-4,15-16, ISTQB certification
Other sources: Stéphane S. Somé, Lionel Briand, R.V. Binder, Z. Micskei

Table of Contents

BASIC DEFINITIONS AND OBJECTIVES OF TESTING

What is Software Testing?

according to D. Galin:

software testing = formal process carried out by a specialized testing team in which a software unit, several integrated software units, or an entire software package are examined by running the programs on a computer; all the associated tests are performed according to approved test procedures on approved test cases

What is Software Testing?

according to SWEBOK

Source: IEEE, „Software Engineering Body of Knowledge”(SWEBOK) 2004
URL: http://www.computer.org/portal/web/swebok/

Testing is an activity performed for evaluating product quality, and for improving it by identifying defects

Testing is an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component

Source: IEEE, "IEEE Standard for Software and System Test Documentation," *IEEE Std 829-2008*, 2008

according to IEEE

What is Software Testing?

The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products

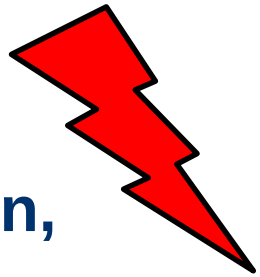
- **To determine that they satisfy specified requirements**
- **To demonstrate that they are fit for purpose**
- **To detect defects**

Source: International Software Testing Qualifications Board (ISTQB), URL: http://istqb.org/

according to ISTQB

Basic Testing Definitions

- **Mistake**: people commit errors
- **Defect (bug)**: a mistake (in the SW documentation, code, etc.) can lead to a defect
- **Failure**: a failure occurs when a defect executes
- **Incident**: consequences of failures – failure occurrence may or may not be apparent to the user
- **Software testing**: exercise the software with test cases to gain (or reduce) confidence in the system (execution based on test cases)
  - Expectation → reveal faults with failures incidences



Misleading terminology!

Types of Defects

- Ambiguities
- Omissions
- Inconsistencies
- Inaccuracies
- Contradictions
- Superfluous statements

Objectives of Testing

For customers and stakeholders

- Support decision making
- Reduce level of risks of inadaquate SW quality

For project manager

- Evaluate work products (reqs, user stories, source code)
- Verify if all requirements are fulfilled
- Build confidence in the level of quality of test object
- Detect and prevent defects

For external authorities

- Comply with legal or regulatory requirements or standards
- Verify the test object's compliance with those standards

Two Testing Schools

Test-as-information-provider

- Test-last
- Independent test team
- Separate test phase
- Fixed releases

Test-as-quality-accelerant

- Test-always
- Testers are quality assistants
- Developers write tests
- Release often / always

Source: https://angryweasel.com/blog/two-new-schools/
Z. Micskei, I. Majzik: Introdution to Testing

Testing vs. Debugging

Debugging

- Find the cause of the bug
- Finds, analyzes and fixes such defects
- Carried out (mostly) by the development team

Testing

- Find the bug
- Shows failures caused by defects
- Carried out (mostly) by the QA team

Sec. 1.3: Foundation Level Syllabus of ISTQB

SEVEN TESTING PRINCIPLES

Principle 1 (P1)

“Program testing can be used to show the presence of bugs, but never to show their absence”

Edsger Dijkstra, 1972

- **Try to find as many defects as possible before they cause a production system to fail**
- **But even if no bugs found → no proof for correctness**
  - Absolute certainty cannot be gained from testing → testing should be integrated with other verification activities

P2: Exhaustive testing is impossible

- **Impossible** to test a program under all operating conditions
  → based on **incomplete testing**, we must gain confidence that the system has the desired behavior

- **Large** input space
- **Large** output space
- **Large** state space
- **Large** number of possible execution paths

- **Subjectivity** of specifications

Why is Testing Difficult?

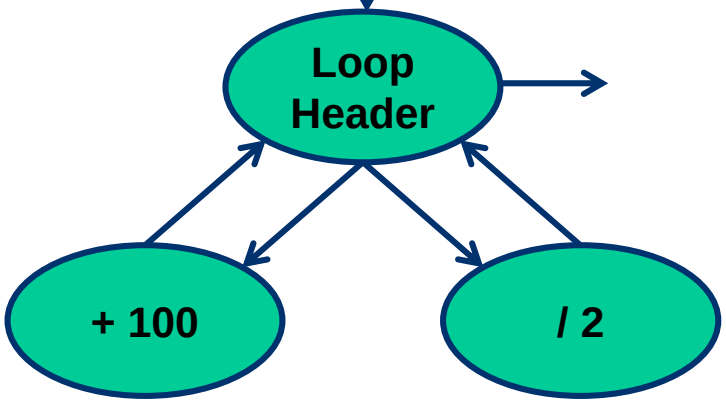
large input/state space

```
int exFunction(int x, int y)
{ ... }
```

- **Exhaustive testing, i.e., testing a software system using all the possible inputs (e.g., trying all possible combination of x and y), is impractical (if not impossible)**
- **Other examples:**
  - A program that computes the factorial function (*n! = n*(n-1)*(n-2)*…*1*)
    - Exhaustive testing = running the program with 0, 1, 2, …, 100, …, 1000, … as an input!
  - A compiler (e.g., javac)
    - Exhaustive testing = running the (Java) compiler with every possible (Java) program

Why is Testing Difficult?

```
...
for (int i = 0; i < n; ++i) {
        if (a.get(i) == b.get(i))
                x[i] = x[i] + 100;
        else
                x[i] = x[i]/2;
}
...
```



number of paths = $2^n + 1$ (for n > 0; including loop header, then exit)

large number of possible execution paths

n	number of paths
1	3
2	5
3	9
10	1025
20	1048577
60	$1.15*10^{18}$

with 10^{-3} seconds per test case → need more time than seconds since big bang for n = 36

Why is Testing Difficult?

upper limit to total number of tests

$$2^n \times (L_1 \times L_2 \times \ldots \times L_X) \times (V_1 \times V_2 \times \ldots \times V_Y)$$

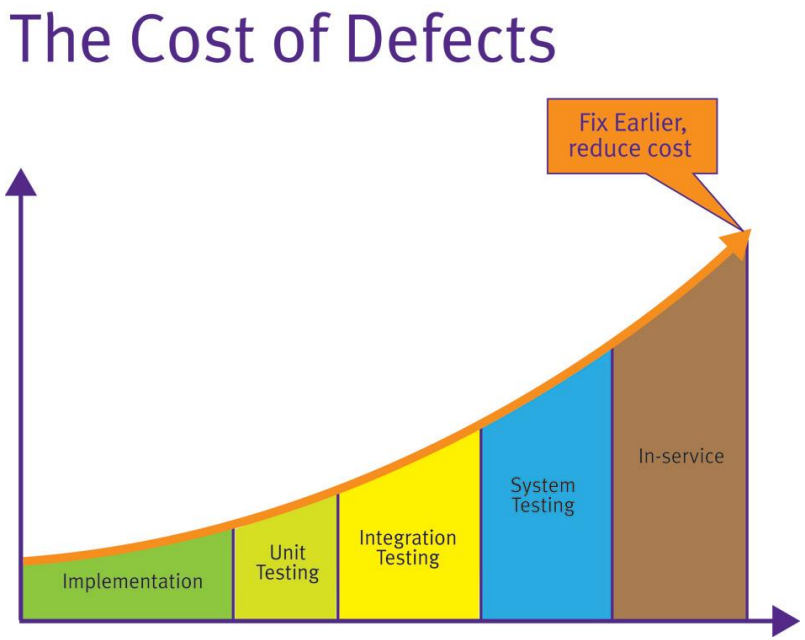
- n: number of decisions
- L_i: number of times a decision can loop
- X: number of decisions that cause loops
- V_i: number of all the possible values each input variable could have
- Y: number of input variables

Why is Testing Difficult?

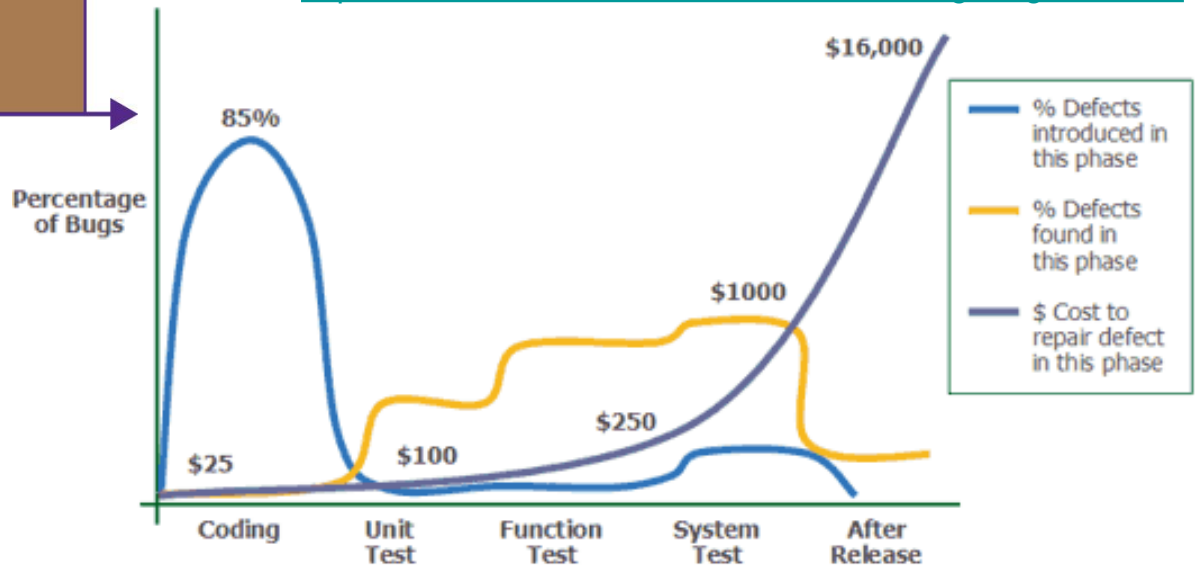
- **Continuity property: small differences in operating conditions will not result in dramatically different behavior → does not apply to software!**
- **Consider testing a bridge's ability to sustain a certain weight**
- **If a bridge can sustain a weight equal to W1, then it will sustain any weight W2 ≤ W1**
- **The same simplifications cannot be applied to software …**

P3: Early testing saves time and money

Src: http://www.embeddedinsights.com/channels/channels/simulation-debugging/



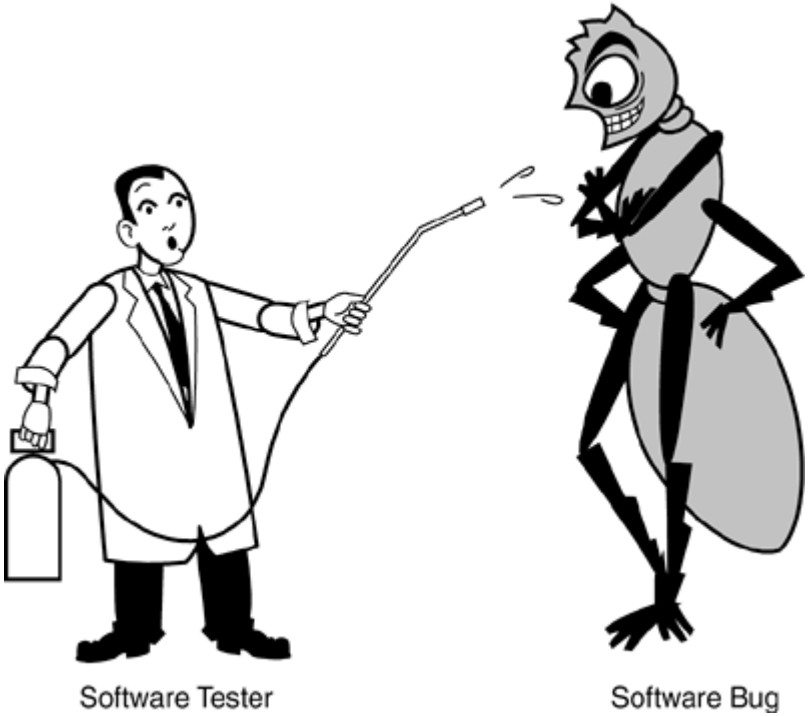
Src: https://forum.keenswh.com/threads/cost-of-bugfixing.7294565/



P4: Defects cluster together

- Testing cannot prove the absence of bugs
  - The more bugs you fix, the more bugs there are

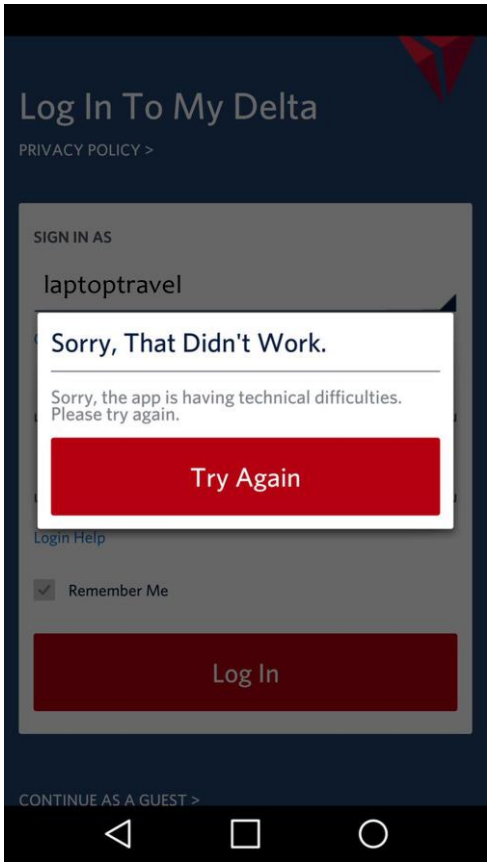
P5: The Pesticide Paradox



- The pesticide paradox (B. Beizer, 1990)
  - A system tends to build resistance to a particular testing technique
  - Executing the same tests will not find new bugs

P6: Testing is context-dependent

- **Different systems are tested differently**
  - Increased level of criticality → increased level of testing



P7: Absence of errors is a fallacy

- The fact that no defects are outstanding is not a good reason to ship the software
- Finding and fixing many bugs does not help if SW does not fulfill user needs

Raymond Gillespie, TCUK15 19

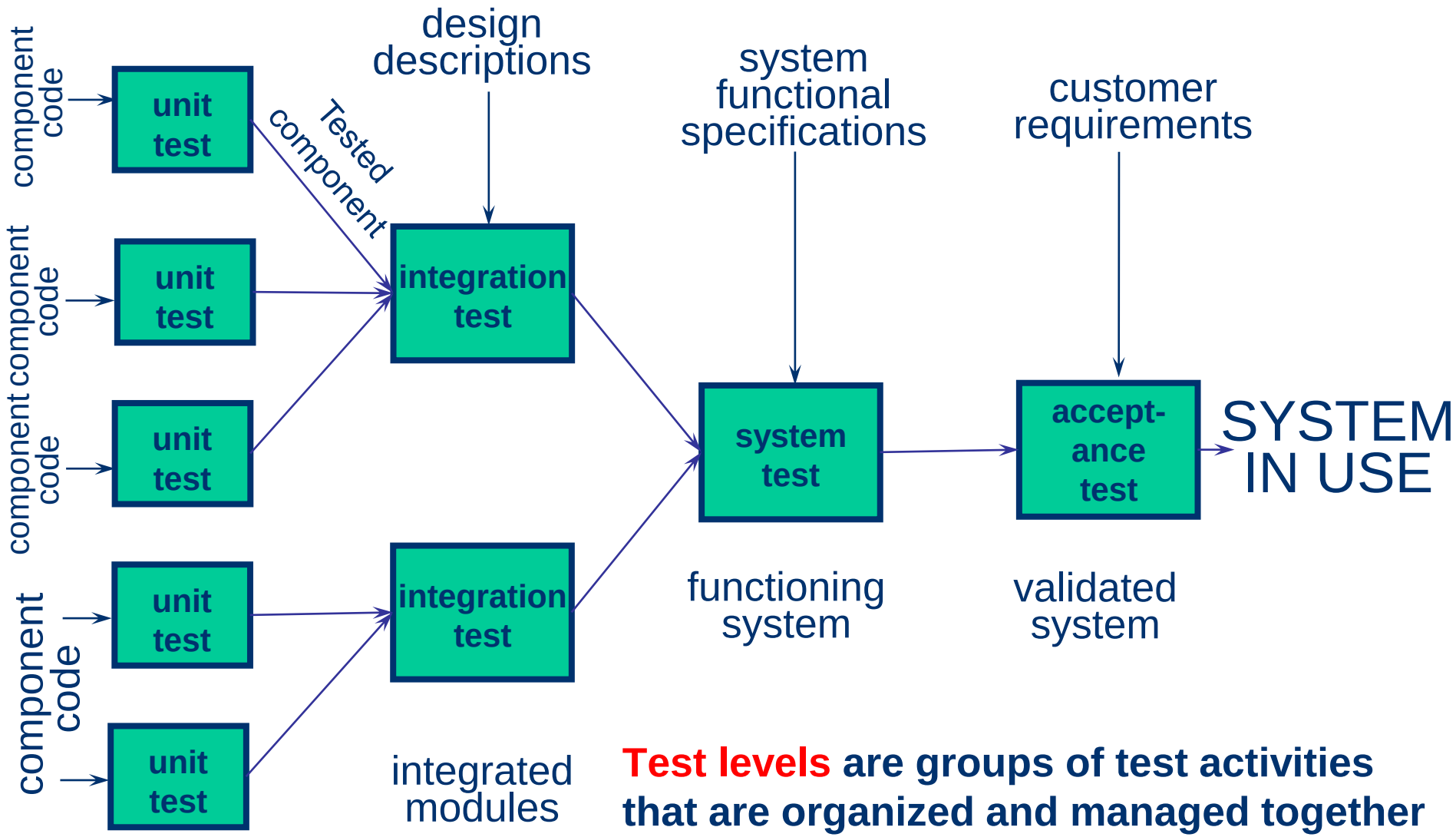
7 Axioms of Testing

- Testing shows the presence of bugs, not their absence
- Exhaustive testing is impossible
- Early testing saves time and money
- Defects cluster together
- The pesticide paradox
- Testing is context dependent
- Absence of errors is a fallacy

Sec. 2.2: Foundation Level Syllabus of ISTQB

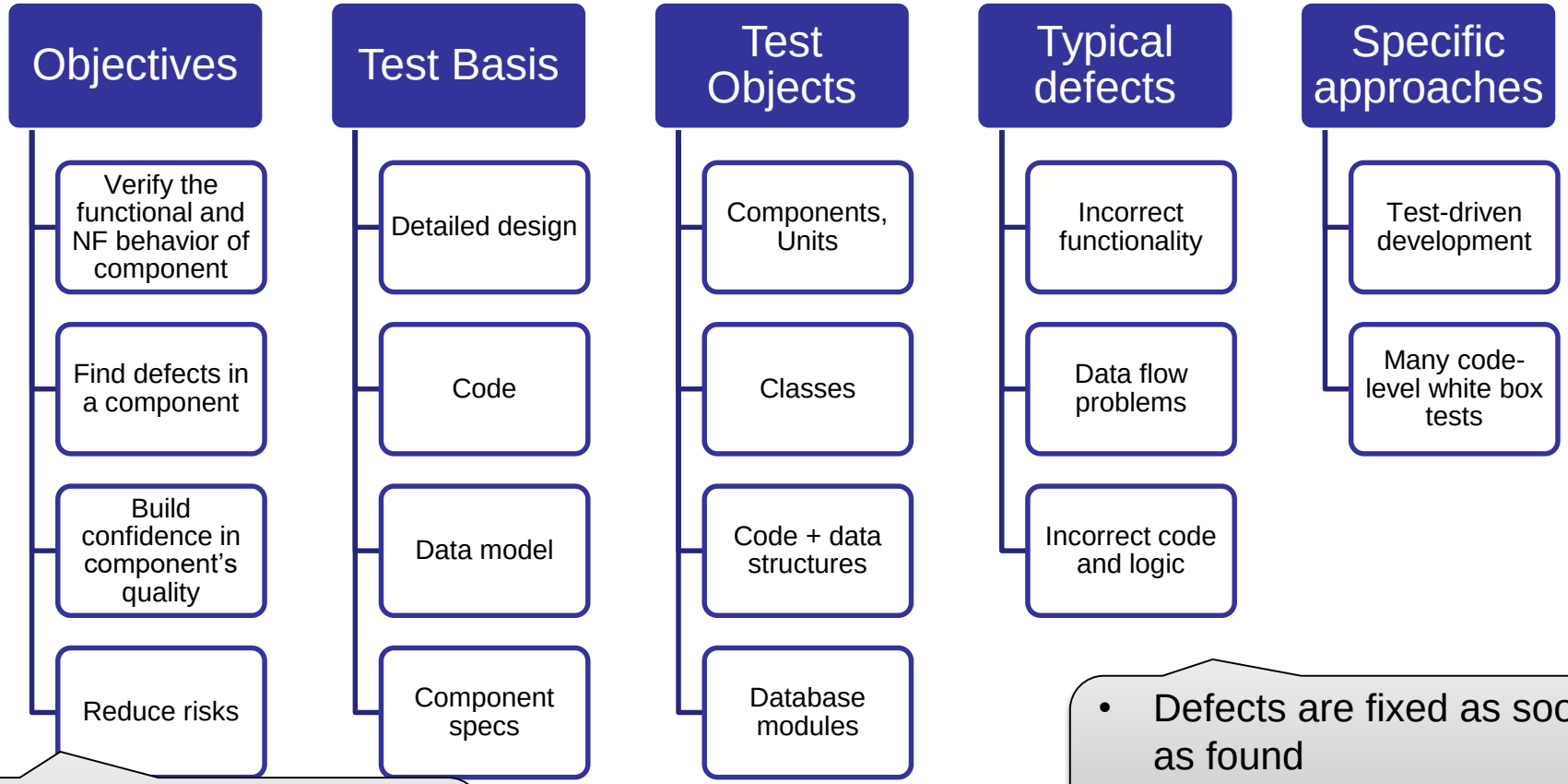
TEST LEVELS

Test Levels



Test levels are groups of test activities that are organized and managed together

Component / Unit Testing



- Done in isolation from rest of the system
- May cover functional and non-functional aspects

- Defects are fixed as soon as found
- No formal defect management

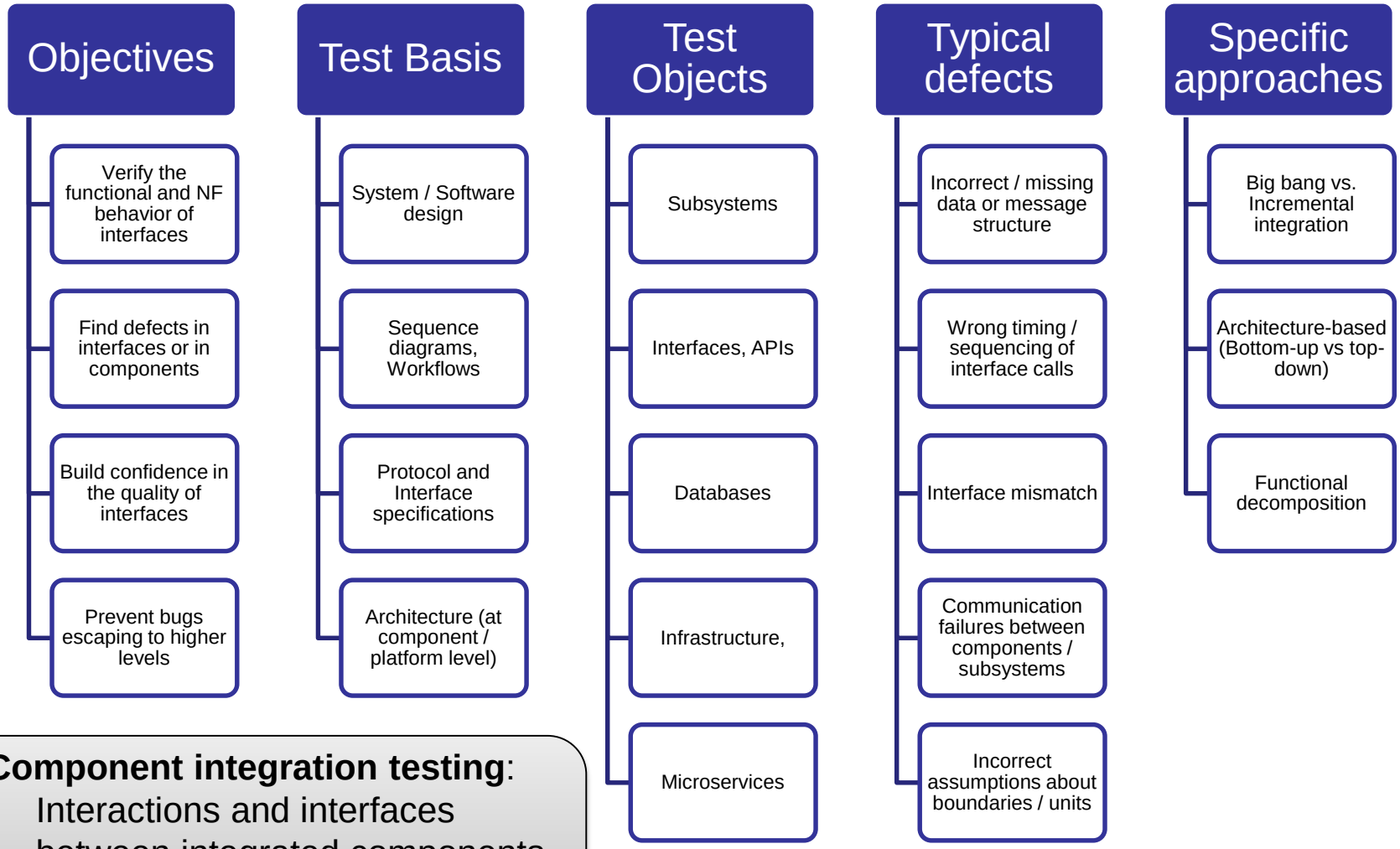
Integration Testing

- Well-tested modules may still fail integration tests…

Integration Testing

- Well-tested modules may still fail integration tests…

Integration Testing



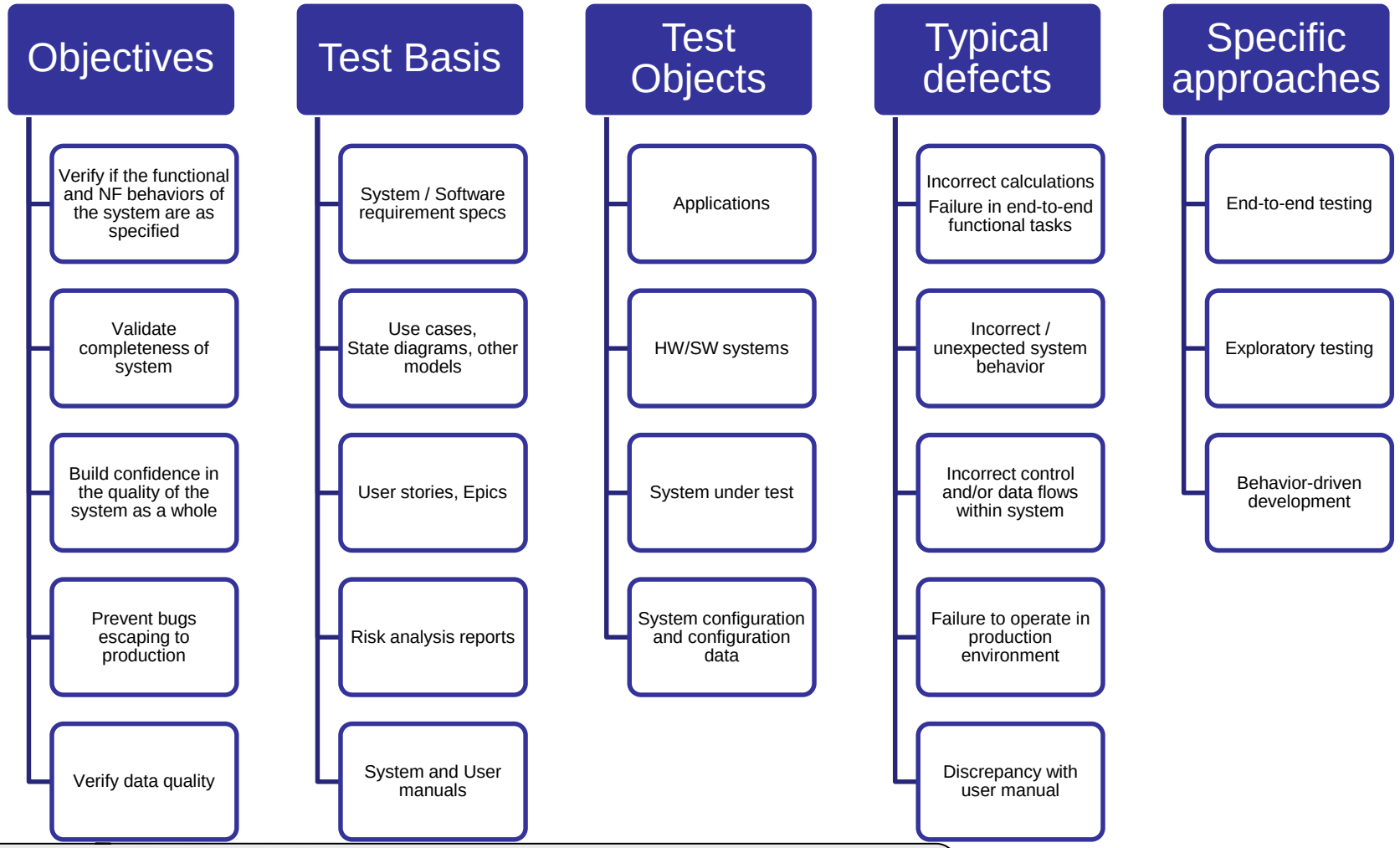
Component integration testing:

- Interactions and interfaces between integrated components
- Right after component testing
- Typically automated

System integration testing:

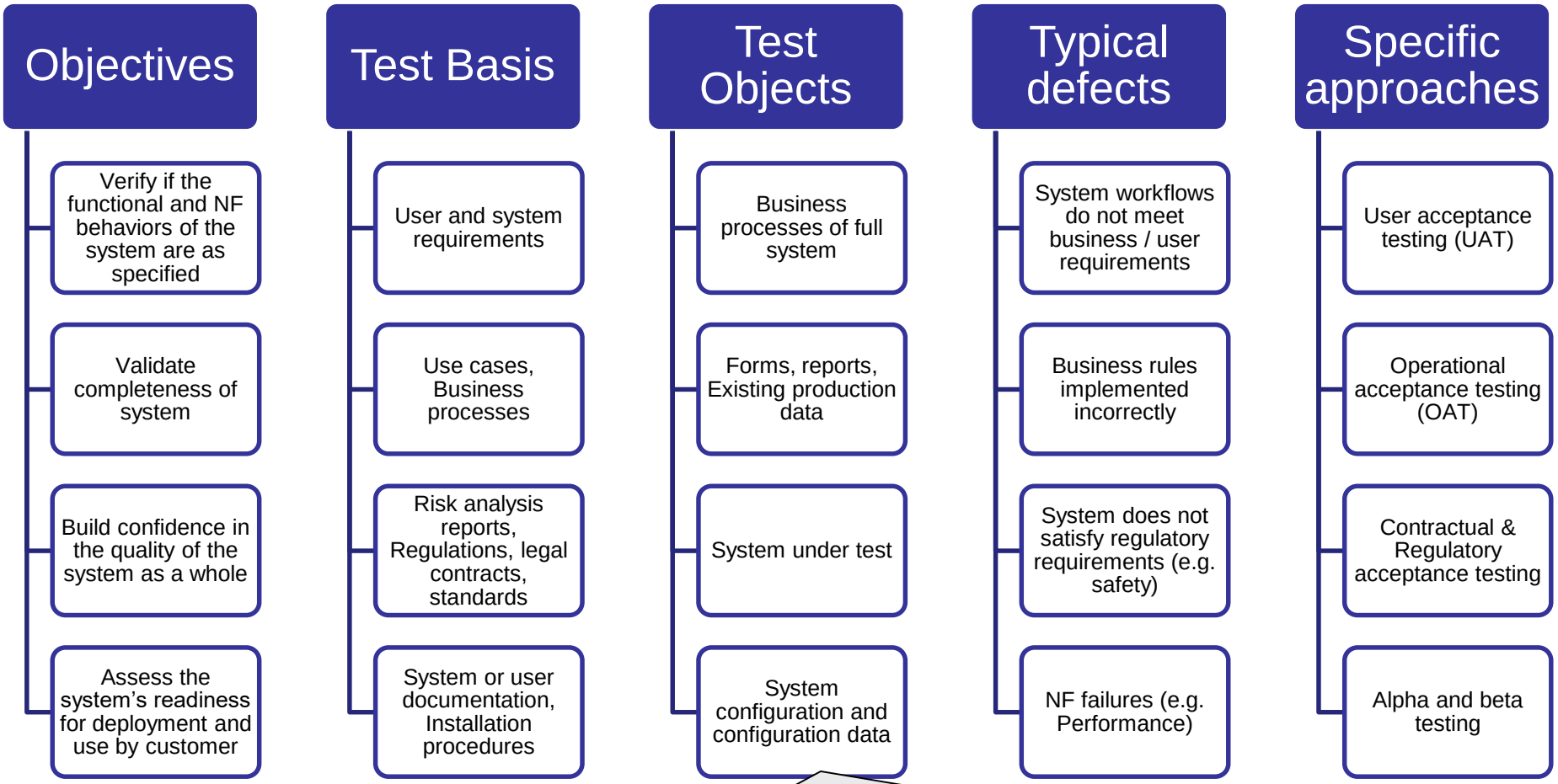
- interactions and interfaces between packages, subsystems, microservices, external services
- After / in parallel with system testing

System Testing



- Typically carried out by independent testers
- Best practice: involve testers early in defining user stories

Acceptance Testing



Other work products:
Disaster recovery procedures,
NF requirements, Performance targets, Safety/security standards, etc.

Acceptence testing approaches

User acceptance testing

- Fitness for use by intended users
- In real or simulated environment
- Build confidence in that users get what they need
- Business processes are performed correctly

Operational acceptance testing

- SysAdmins perform in a simulated production env.
- Test backup and restore
- Install / uninstall
- Disaster recovery
- User management
- Data load & migration
- Performance testing
- Vulnerability checks

Contractual and regulatory acc. testing

- **Contractual**:
  - Check wrt contract's acceptance criteria
- **Regulatory**:
  - Check adherence to regulations (goverment, legal, safety)
  - Performed by independent users or authorities

Alpha and Beta testing

- Build confidence among potential or existing custmers and operators that they can use the system under regular conditions
- Reveal defects of heterogeneous environments
- **Alpha testing**:
  - Performed at the developer organization site
  - by existing customers / users
- **Beta testing**:
  - Performed by existing users/customers
  - At their location

Sec. 2.3: Foundation Level Syllabus of ISTQB

TEST TYPES

Functional vs. Non-functional tests

Functional testing

Goal:

- Evaluate functions that the system should perform wrt. requirements / specification (user stories, use cases, etc.)
- What the system should do?

Quality characteristics:

- Completeness, correctness, appropriateness

Scope:

- Should be performed at all levels

Thoroughness:

- **Functional coverage**: to what extent a functional element has been exercised by tests (%)

Non-functional testing

Goal:

- Evaluate characteristics of a system as a whole
- How well the system behaves?

Quality characteristics:

- Reliability, performance, security, usability, etc.

Scope:

- Should be performed at all test levels (not only at system-level!)
- Sample technique: boundary value analysis for extreme conditions

Thoroughness:

- Non-functional coverage: the extent to which some type of NF element has been exercised by tests (%)

This is rather naive! Instead: Confidence intervals:
Throughput is 1500±5 with 95% probability
Response time is < 2s with 95% probability

Black-Box vs. White-Box Testing

Black-box

Goal:

- Evaluate functions that the system should perform wrt. requirements / specification (user stories, use cases, etc.)
- What the system should do?

Quality characteristics:

- Completeness, correctness, appropriateness

Scope:

- Should be performed at all levels
- It cannot reveal unexpected functionality (not part of the spec but implemented)

Thoroughness:

- **Functional coverage**: to what extent a functional element has been exercised by tests (%)

White-box

Goal

- Derives tests based on the system's internal structure e.g. Control or data flow
- What the system should do?

Quality characteristics:

- Completeness, correctness, appropriateness

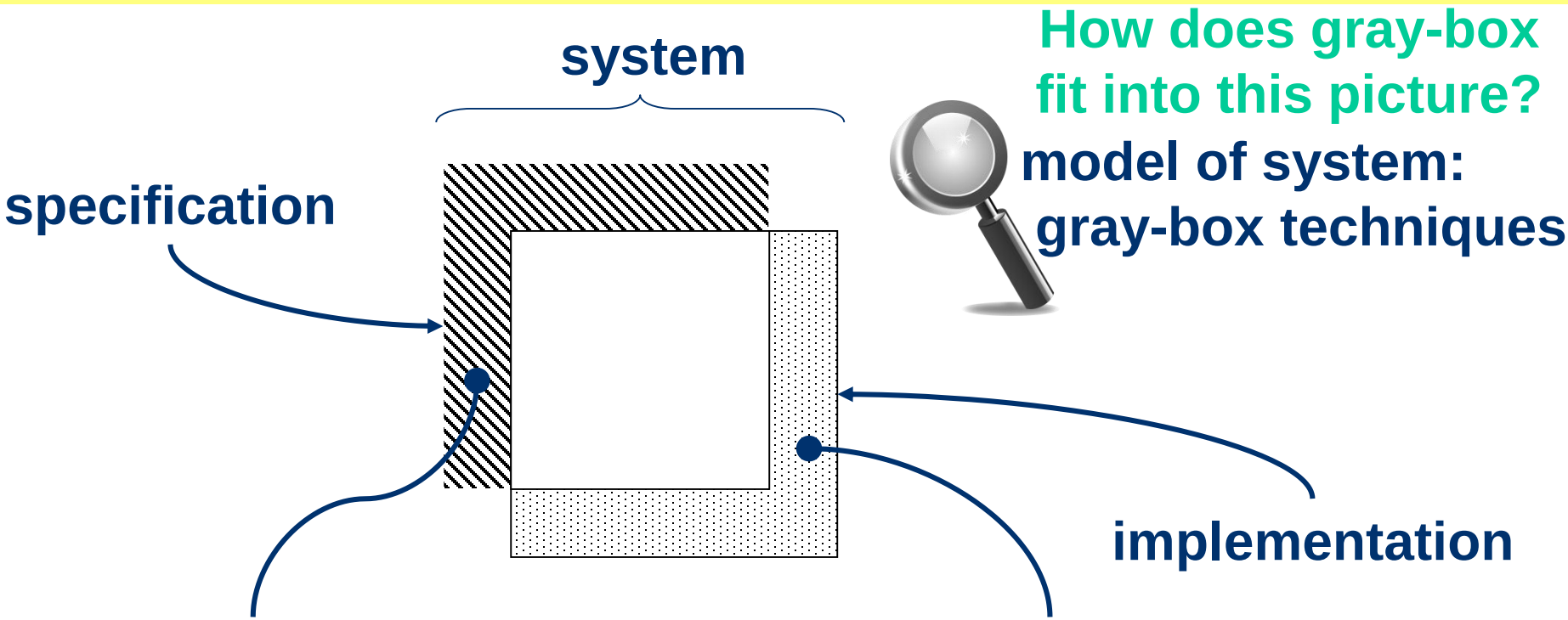
Scope:

- May be defined at all levels (but typically at unit or integration-level)
- It cannot reveal missing functionalities (part of the specification that is not implemented)

Thoroughness:

- **Code coverage**: e.g. Statement, branch investigated by tests (%)

Black-Box vs. White-Box vs. Model-based (Gray-Box) Testing

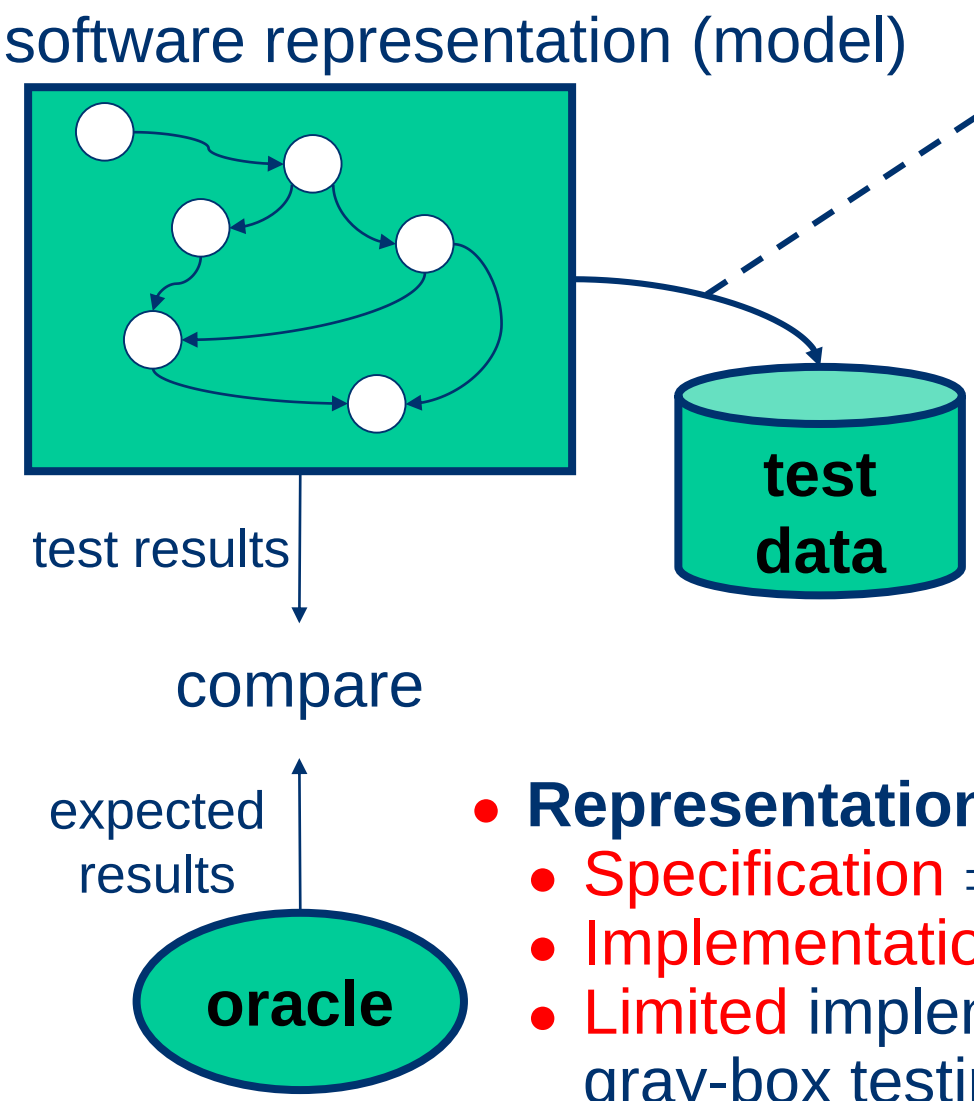


How does gray-box fit into this picture?

model of system: gray-box techniques

- Missing functionality cannot be revealed by white-box techniques
- Unexpected functionality cannot be revealed by black-box techniques

Oracles and Test Coverage



- **Associated criteria**
  - Test cases must cover all the … in the model
  - Technique used to reduce the # of inputs
  - Testing criteria group input elements into (equivalence) classes
  - A small number of inputs are selected in each class (maximize test coverage)
- **Representation of**
  - Specification ⇒ black-box testing
  - Implementation ⇒ white-box testing
  - Limited implementation knowledge ⇒ gray-box testing

Complete Coverage: White-Box

```
...
if x > y then
        Max := x;
else
        Max := x;  // fault!
end if;
...
```

See details later in the course

- **{x=3, y=2; x=2, y=3}**
  - Can detect the error, more "coverage"
- **{x=3, y=2; x=4, y=3; x=5, y=1}**
  - Larger test set but cannot detect the error
- **Testing criteria group input domain elements into (equivalence) classes (control flow paths here)**
- **Complete coverage attempts to run test cases from each class**

Complete Coverage: Black-Box

- **Specification of Compute Factorial Number:**
  - If the input value n is < 0, then an appropriate error message must be printed. If 0 <= n < 20, then the exact value of n! must be printed. If 20 <= n < 200, then an approximate value of n! must be printed in floating point format, e.g., using some approximate method of numerical calculus. The admissible error is 0.1% of the exact value. Finally, if n>=200, the input can be rejected by printing an appropriate error message.

- Because of expected variations in behavior, it is quite natural to divide the input domain into the classes {n<0}, {0<= n <20}, {20 <= n < 200}, {n >= 200}. We can use one or more test cases from each class in each test set. Correct results from one such test set support the assertion that the program will behave correctly for any other class value, but there is no guarantee!

See details later in the course

Complete Coverage: Model-based

- **Coverage criteria depend on the type of model used to represent the system**
  - E.g., all execution paths of a labeled transition system
  - E.g., all transitions in a state machine

See details later in the course

Change-related Testing

Confirmation testing

- Confirm that original defect is successfully fixed
- At least failed test cases need to be re-executed

Regression testing

- Change in one part of the code may accidentally affect other parts
- Detect unintended side- effects

ISTQB

These tests are
- **Performed at all levels**
- **Evolve slowly**
- **Run many times**

→ **Highly automated**

IEEE

regression testing = selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements

Regression Testing

- **Different possible scopes of regression testing:**
  - Local: direct testing of the code changed or added
  - Surrounding: testing of functions supported or directly impacted by the change
  - Confidence: predefined suite of tests routinely run after any change is made to the product/system
- **Automation is essential**

- **Regression test suite includes more effective test cases (e.g., boundary tests, tests that revealed bugs, tests for customer-reported bugs)**

- **Size of regression test suite must be kept reasonable**

What kind of test is this?

- **Options: Functional (F), Non-functional (NF), White-box (W), Change-related (CH)**

1. For component testing, performance tests are designed to evaluate the number of CPU cycles required to perform a complex total interest calculation. NF

2. For system testing, tests are designed based on how account holders can apply for a line of credit on their checking accounts. F

3. For system testing, all tests for a given workflow are re-executed if any screen on that workflow changes. CH

4. For component integration testing, tests are designed to exercise how each screen in the browser interface passes data to the next screen and to the business logic. W

5. For component integration testing, tests are designed based on how account information captured at the user interface is passed to the business logic. F

6. For acceptance testing, tests are designed to cover all supported financial data file structures and value ranges for bank-to-bank transfers. W

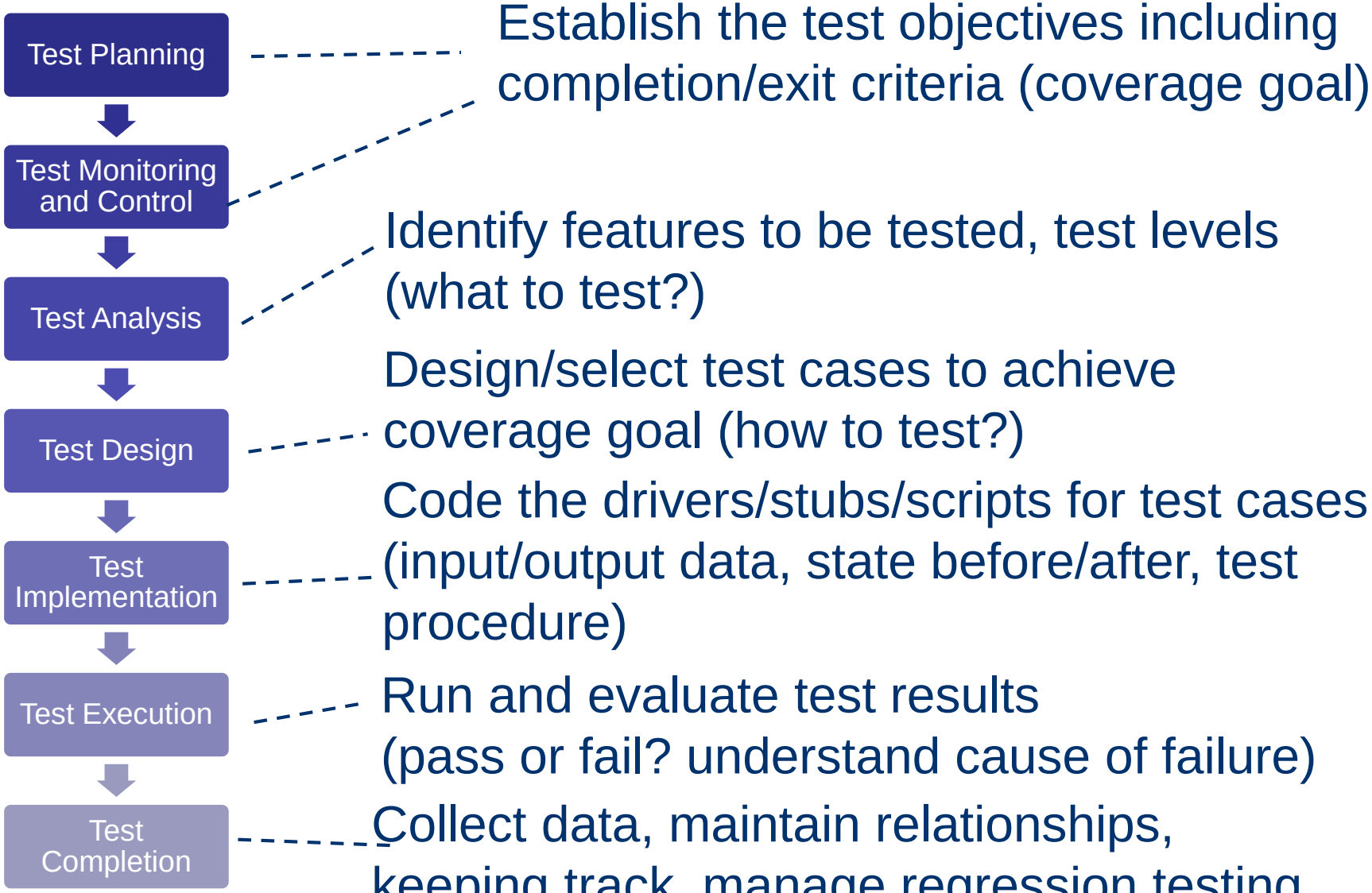
Maintenance Testing

- **Goal:**
  - Carried after system is already in production env.
  - Changes made for maintenance → testing is needed
  - Evaluate success + Ensure the lack of side effects
- **Triggers for maintenance:**
  - Modification (e.g. planned enhancement)
  - Migration (e.g. from one platform to another)
  - Retirement (of a functionality / subsystem)
- **Impact analysis for maintenance**
  - Evaluate the planned / executed changes
  - Identify consequences + possible side effects
- **Why is impact analysis difficult?**

Sec. 1.4: Foundation Level Syllabus of ISTQB

TEST ACTIVITIES AND PROCESSES

Testing is a Process



Test Activities

Test Planning

- Define the objectives of testing
- Define the approach how to meet test objectives within constraints imposed by the context

Test Monitoring and Control

- Compare (continuously) actual progress against the test plan using test monitoring metrics (defined in the test plan)
- Evaluate exit criteria (e.g. check test results against coverage criteria)

Test Analysis

- TA determines “what to test” in terms of measurable coverage criteria
  - Identify testable features
  - Define and prioritize associated test conditions
  - Capture traceability (between test basis and test condition)
- Analyze test basis to identify testable features
  - Example test basis: Requirements, design model, implementation, etc.

Test Activities

Test Design

- Elaborate test conditions into (sets of) high-level test cases
- Design and prioritize test cases
- Identify necessary test data to support test conditions and test cases
- Design test environment and identify required infrastructure and tools
- Capture traceability between test basis, test conditions, test cases, and test procedures
- Informally: How to test?

Test Implementation

- Informally: Do we now have everything in place to run the tests?”
- Create test software for test execution
  - Develop and prioritize test procedures
  - Create test suites and automated test scripts from test procedures
  - Arrange test suites into an efficient test execution schedule
  - Build the test environment (incl. test harnesses, service virtualization, simulators, etc.)
- Prepare test data and load it into the test environment
- Verify and update traceability between the test basis, test conditions, test cases, test procedures, and test suites

Test Activities

Test Execution

- Run test suites in accordance with test execution schedule
  - Execute tests (manually or automatically)
  - Compare actual results with expected results
  - Analyze anomalies to identify their causes
  - Report defects based on the observed failures
  - Log outcome of test execution
  - Verify and update traceability

Test Completion

- Collect data from completed test activities to consolidate experience, testware, etc.
- occurs at project milestones (e.g. Software release, completion of iteration)
  - Check if all defect reports are closed
  - Create a test summary report for stakeholders
  - Analyze lessons learned from the completed test activities to determine changes needed for future iterations

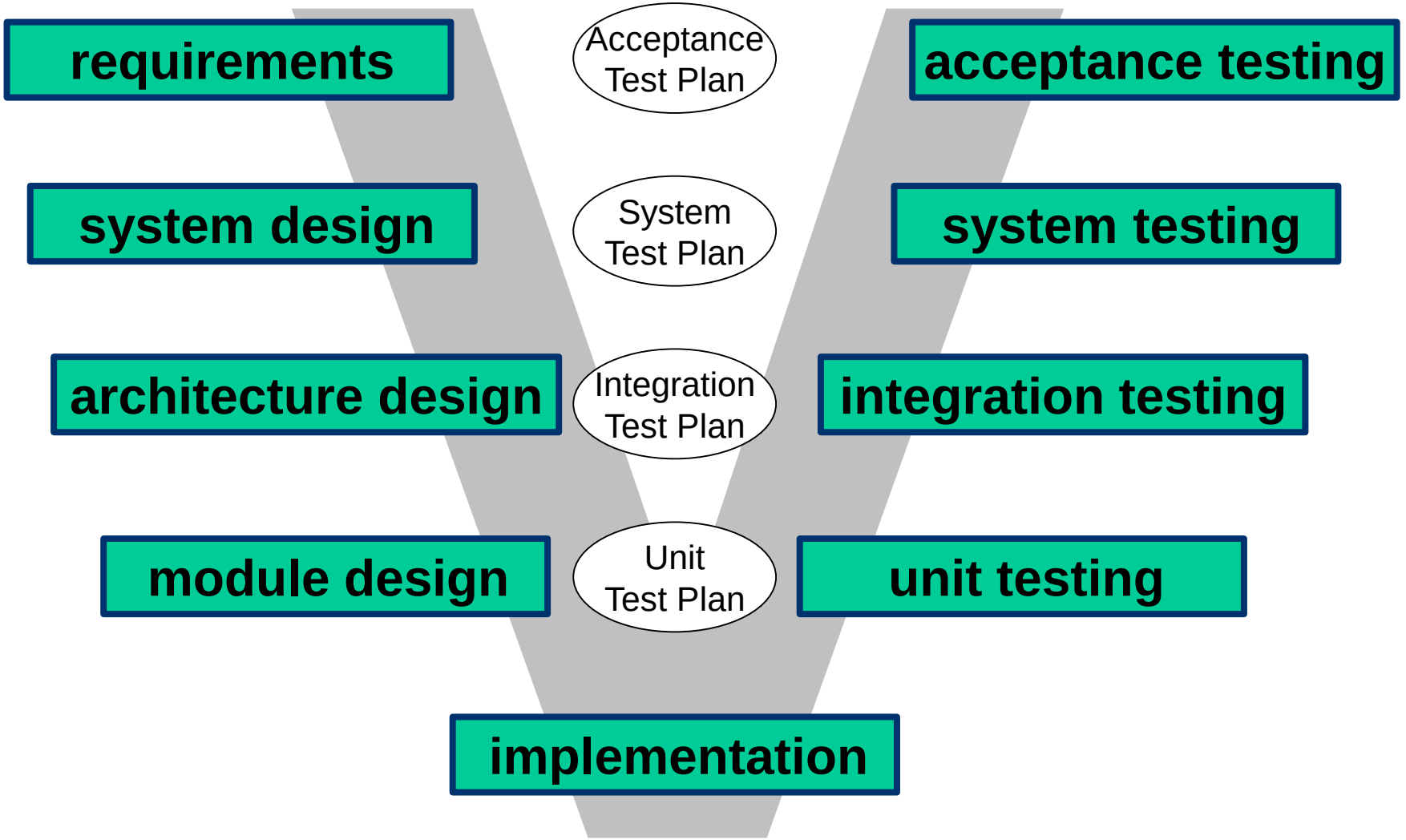
Importance of Traceability

- **Bi-directional traceability between test basis and test work product**
  - Analyze the impact of changes
  - Auditing and certification (safety, security, IT governance)
  - Improve understandability of various test reports
- **Example (Mobile app):**
  - The **test basis** may include a list of requirements and supported mobile devices.
    - Each requirement is an element of the test basis.
    - Each supported device is an element of the test basis
  - **Sample Coverage criteria**:
    - at least one test case for each element of the test basis
  - Test results tell stakeholders if requirements are fulfilled or if failures were observed on supported devices

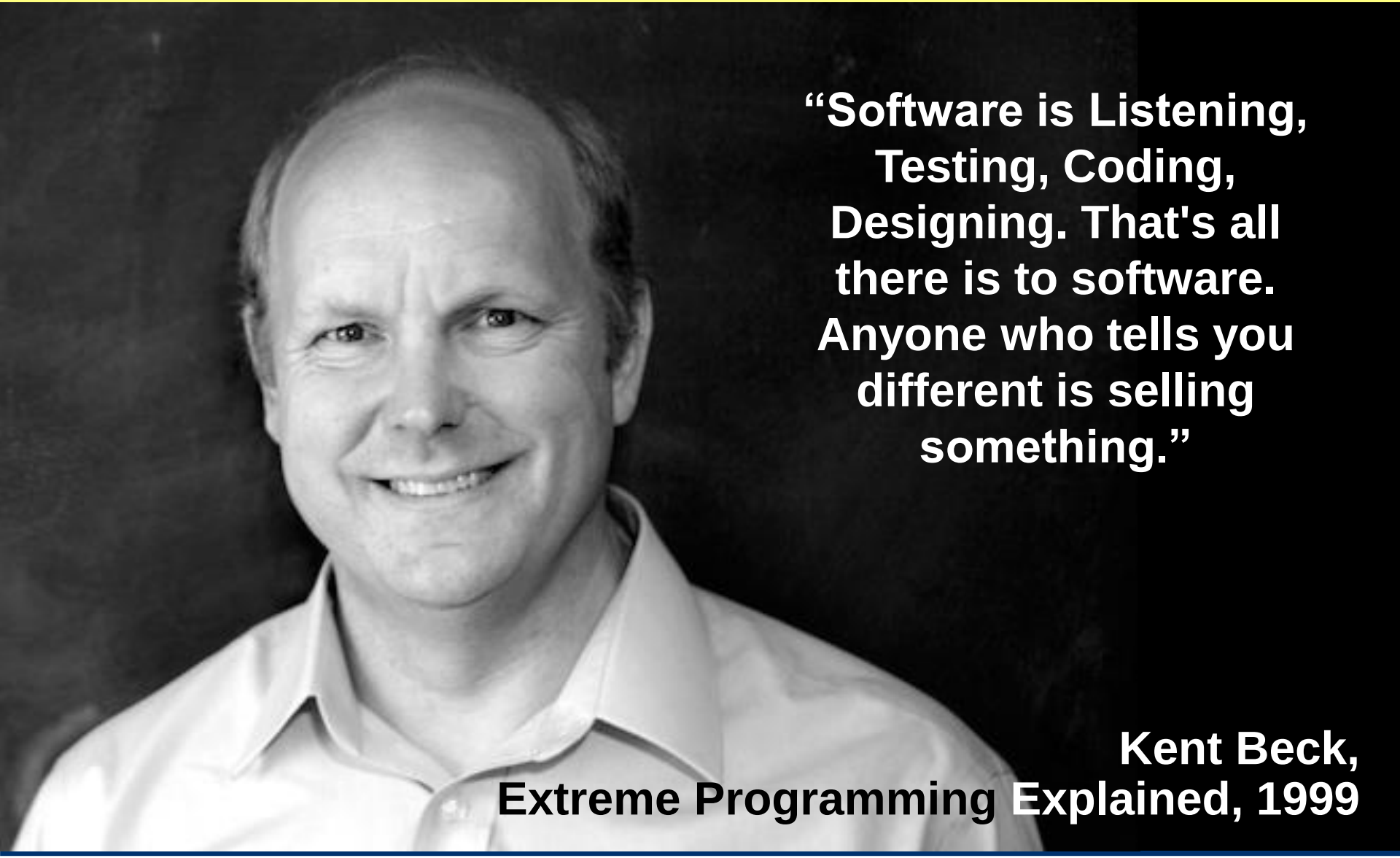
Test work products:
- Requirements, User story
- Test plans, Test reports,
- Source code ...

Examples in CTFL Syllabus

Testing in the V-Model



Test-Driven Development



Test-Driven Development

Listen – Test ~~Design~~ – Code – ~~Test~~ Design

- **Test-Driven Development (TDD)**
- **Listen to customers while gathering requirements, develop test cases, code the program, (re-)design / refactor / clean up as more code is added to the system**

Testing Takes Creativity

- **Testing often viewed as dirty work (though less and less)**

- **To develop an effective test, one must have:**
  - Detailed understanding of the system
  - Knowledge of the testing techniques
  - Skill to apply these techniques effectively and efficiently
- **Testing is done best by independent testers**
  - Programmer often stick to the data set that makes the program work
  - A program often does not work when tried by somebody else
- **Or, write the tests before the code (→ TDD)**

TEST AUTOMATION

Test Automation

- **Use of automated tools is essential:**
  - Provide speed, efficiency, accuracy, precision, resource reduction, simulation & emulation, relentlessness
  - Allow repeatability (regression testing)
- **Types of tools:**
  - Drivers and stubs
  - Stress and load tools
  - Analysis tools (e.g., file comparison, screen capture and comparison)
  - Viewers and monitors (e.g., code coverage tool, debugger)
  - Random testing tools (monkeys)
  - Continuous integration tools
  - Defect tracking

Testing in Continuous Delivery