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

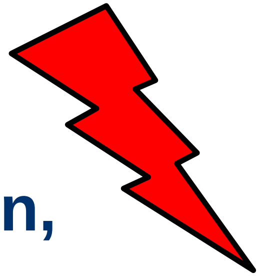
- To determine that they satisfy specified requirements
- To demonstrate that they are fit for purpose
- To detect defects

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



Misleading terminology!

- **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

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-as-quality-accelerant
• Test-last • Independent test team • Separate test phase • Fixed releases	• 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?

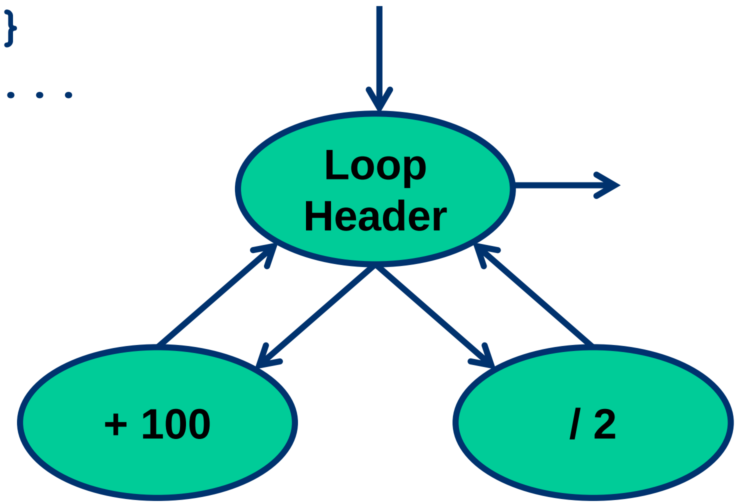
large input/state space

```
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 ($n! = n*(n-1)*(n-2)*\ldots*1$)
    - 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!

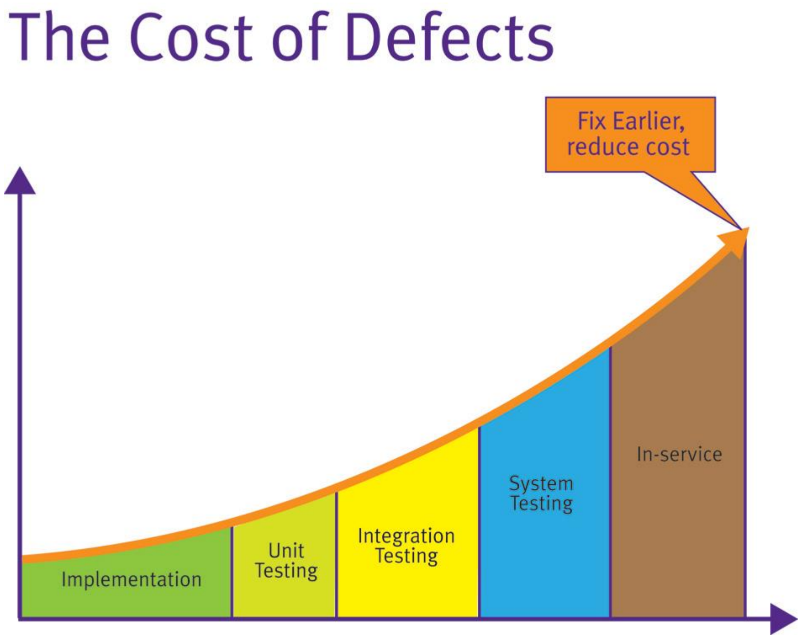
- Consider testing a bridge's ability to sustain a certain weight

- If a bridge can sustain a weight equal to W1, then it will sustain any weight W2 ≤ W1

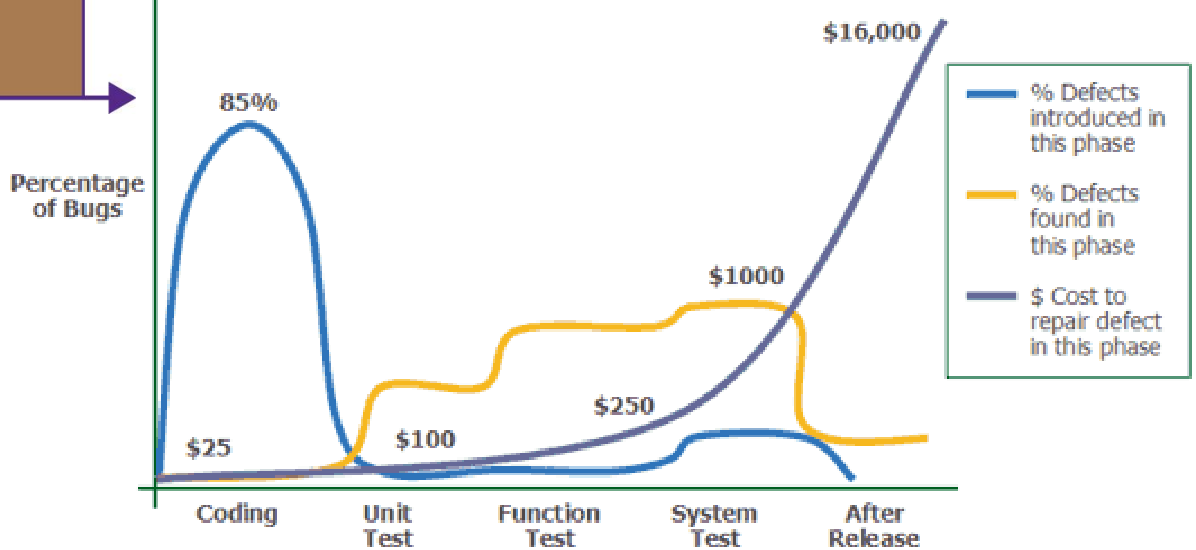
- The same simplifications cannot be applied to software …

P3: Early testing saves time and money

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



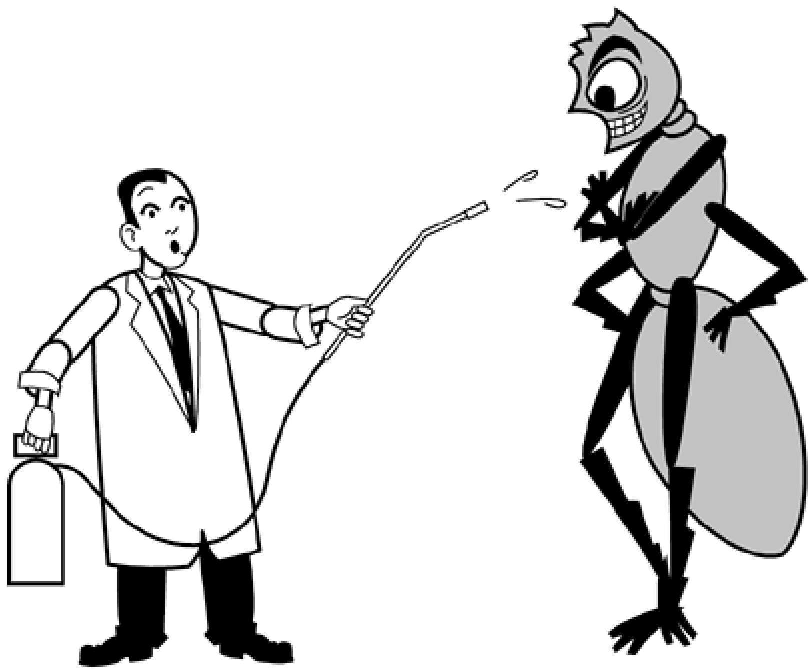
Src: https://forum.keenswh.com/threads/cost-of-bugfixing.7294565/



P4: Defects cluster together

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

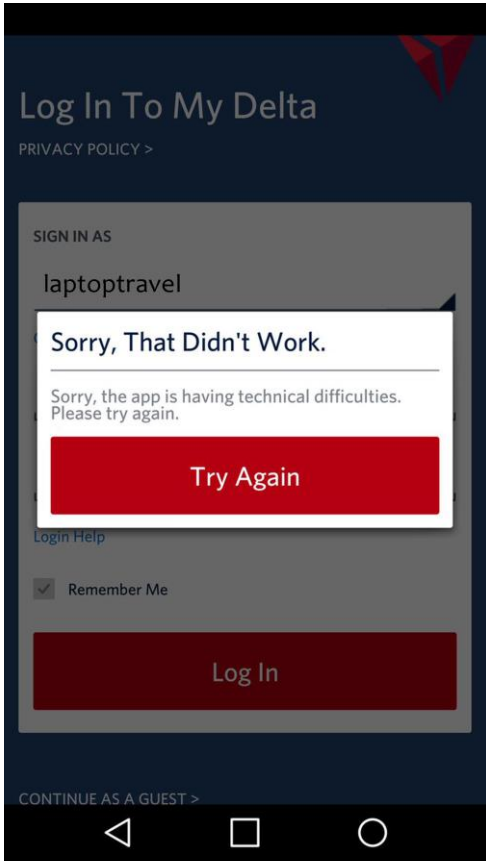
P5: The Pesticide Paradox



- 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

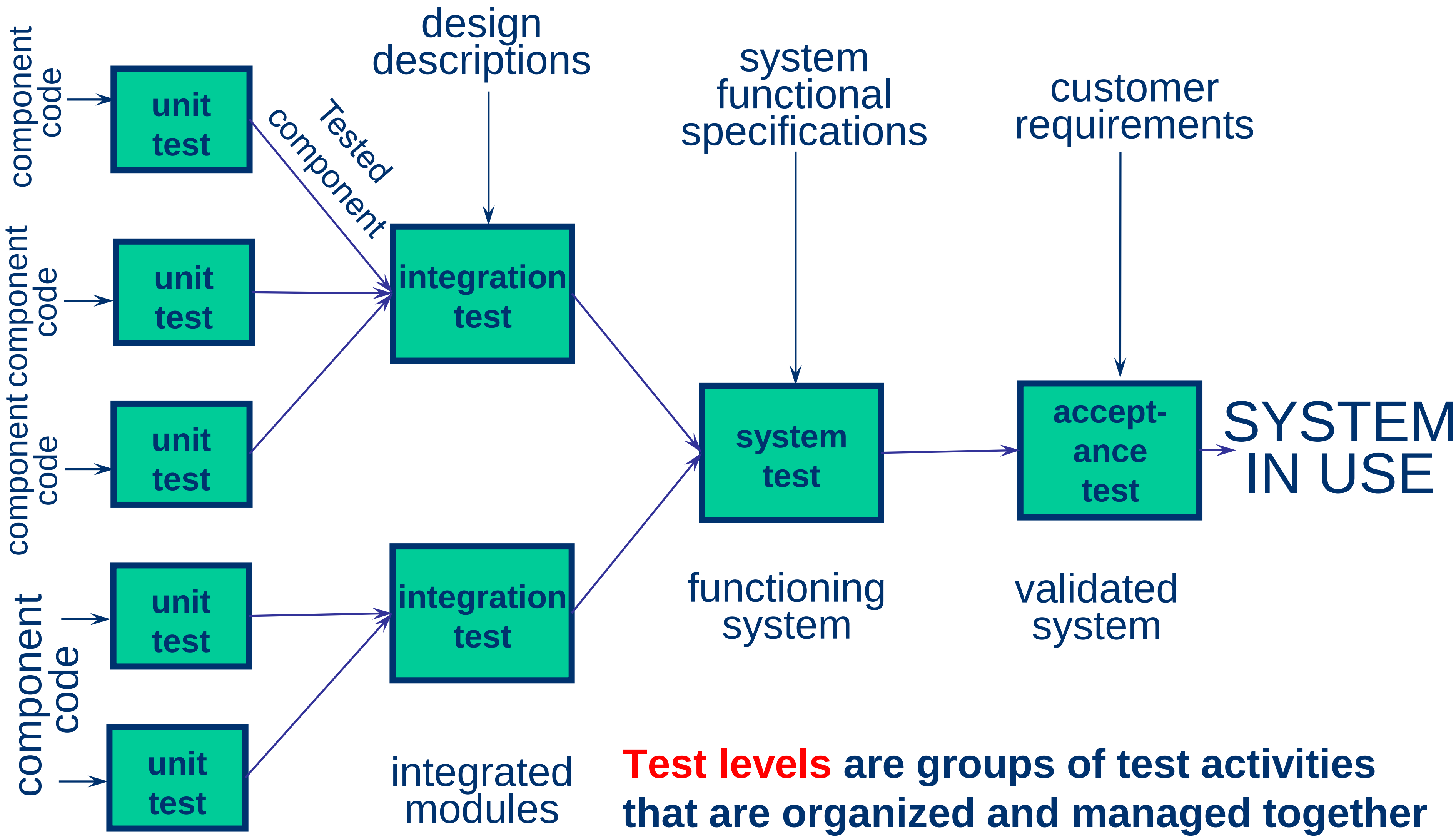
7 Axioms of Testing

- 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

Objectives
- Verify the functional and NF behavior of component
- Find defects in a component
- Build confidence in component's quality
- Reduce risks

Test Basis
- Detailed design
- Code
- Data model
- Component specs

Test Objects
- Components, Units
- Classes
- Code + data structures
- Database modules

Typical defects
- Incorrect functionality
- Data flow problems
- Incorrect code and logic

Specific approaches
- Test-driven development
- Many code-level white box tests

- 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

Objectives
- Verify the functional and NF behavior of interfaces
- Find defects in interfaces or in components
- Build confidence in the quality of interfaces
- Prevent bugs escaping to higher levels

Test Basis
- System / Software design
- Sequence diagrams, Workflows
- Protocol and Interface specifications
- Architecture (at component / platform level)

Test Objects
- Subsystems
- Interfaces, APIs
- Databases
- Infrastructure,
- Microservices

Typical defects
- Incorrect / missing data or message structure
- Wrong timing / sequencing of interface calls
- Interface mismatch
- Communication failures between components / subsystems
- Incorrect assumptions about boundaries / units

Specific approaches
- Big bang vs. Incremental integration
- Architecture-based (Bottom-up vs top-down)
- Functional decomposition

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

Objectives
- Verify if the functional and NF behaviors of the system are as specified
- Validate completeness of system
- Build confidence in the quality of the system as a whole
- Prevent bugs escaping to production
- Verify data quality

Test Basis
- System / Software requirement specs
- Use cases, State diagrams, other models
- User stories, Epics
- Risk analysis reports
- System and User manuals

Test Objects
- Applications
- HW/SW systems
- System under test
- System configuration and configuration data

Typical defects
- Incorrect calculations
  Failure in end-to-end functional tasks
- Incorrect / unexpected system behavior
- Incorrect control and/or data flows within system
- Failure to operate in production environment
- Discrepancy with user manual

Specific approaches
- End-to-end testing
- Exploratory testing
- Behavior-driven development

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

Acceptance Testing

Objectives	Test Basis	Test Objects	Typical defects	Specific approaches
Verify if the functional and NF behaviors of the system are as specified	User and system requirements	Business processes of full system	System workflows do not meet business / user requirements	User acceptance testing (UAT)
Validate completeness of system	Use cases, Business processes	Forms, reports, Existing production data	Business rules implemented incorrectly	Operational acceptance testing (OAT)
Build confidence in the quality of the system as a whole	Risk analysis reports, Regulations, legal contracts, standards	System under test	System does not satisfy regulatory requirements (e.g. safety)	Contractual & Regulatory acceptance testing
Assess the system's readiness for deployment and use by customer	System or user documentation, Installation procedures	System configuration and configuration data	NF failures (e.g. Performance)	Alpha and beta 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

System vs. Acceptance Testing

- **System testing**
  - The software is compared with the requirements specifications (verification)
  - Usually performed by the development team that knows the system

- **Acceptance testing**
  - The software is compared with the end-user requirements (validation)
  - Usually performed by the customer (buyer) who knows the environment where the system is to be used
  - Sometimes, split into alpha & beta-testing for general purpose products

Example: System Requirements

- **Errors at this stage will have devastating effects as every other activity is dependent on it**
- **Natural language: flexible but ambiguous, low testability (guidelines & templates help)**
- **Is it possible to devise a test to check whether the requirements have been met?**
  - e.g., not testable: the system should be user-friendly, the response time should be reasonable
  - e.g., testable: the response time is less than 1.5 seconds for 95% of the time under average system loading

Example: System Requirements

- **Identify early low testability specifications or design!**
  - e.g., devising acceptance tests from requirements early on allows us to assess whether they are testable
  - May even accelerate development!

- **Many life-cycle development artifacts provide a rich source of test data**

- **The lowest level of requirements testing is to generate test data for every requirement at least once**

- **But we want to use techniques that are a bit more demanding: limits and interactions of requirements**

An example decision table

Conditions	R1	R2	R3
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

http://reqtest.com/requirements-blog/a-guide-to-using-decision-tables/