Enterprise information systems in practise SW TESTING

Ing. Daniel Mika, Ph.D. (daniel.mika@atos.net)

- Two hours in the course
- 8 years of praxis in IT (ANF Data, SIS, Atos, FEI)
- Area of interest: test and acceptance criteria, quality
- Projects: IMS, WiMAX, ChargingSpot, sLIM, el. microscope
- ISTQB certified tester – foundation level

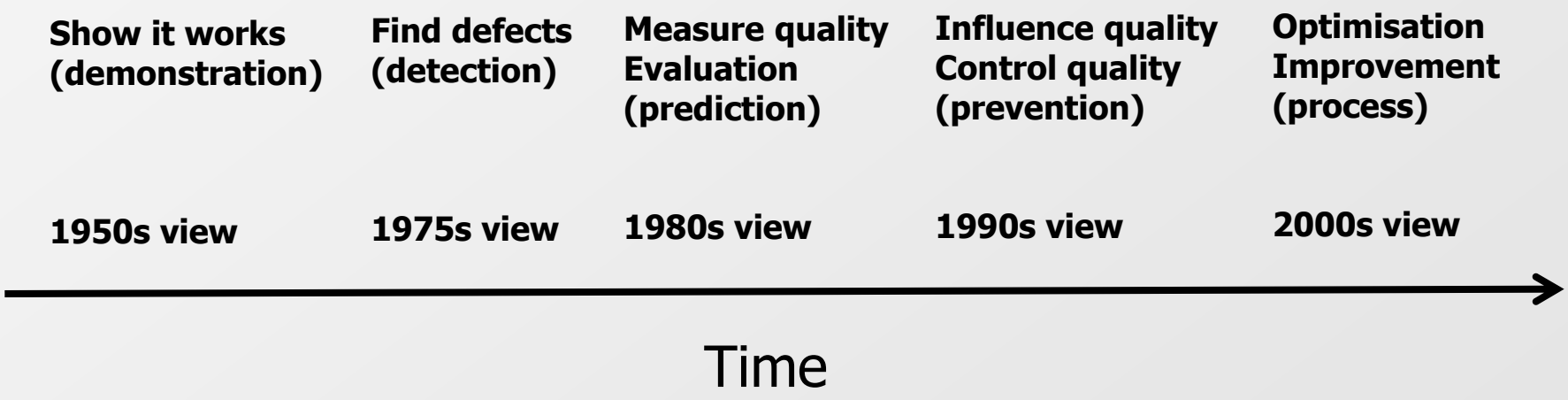
Content (1)

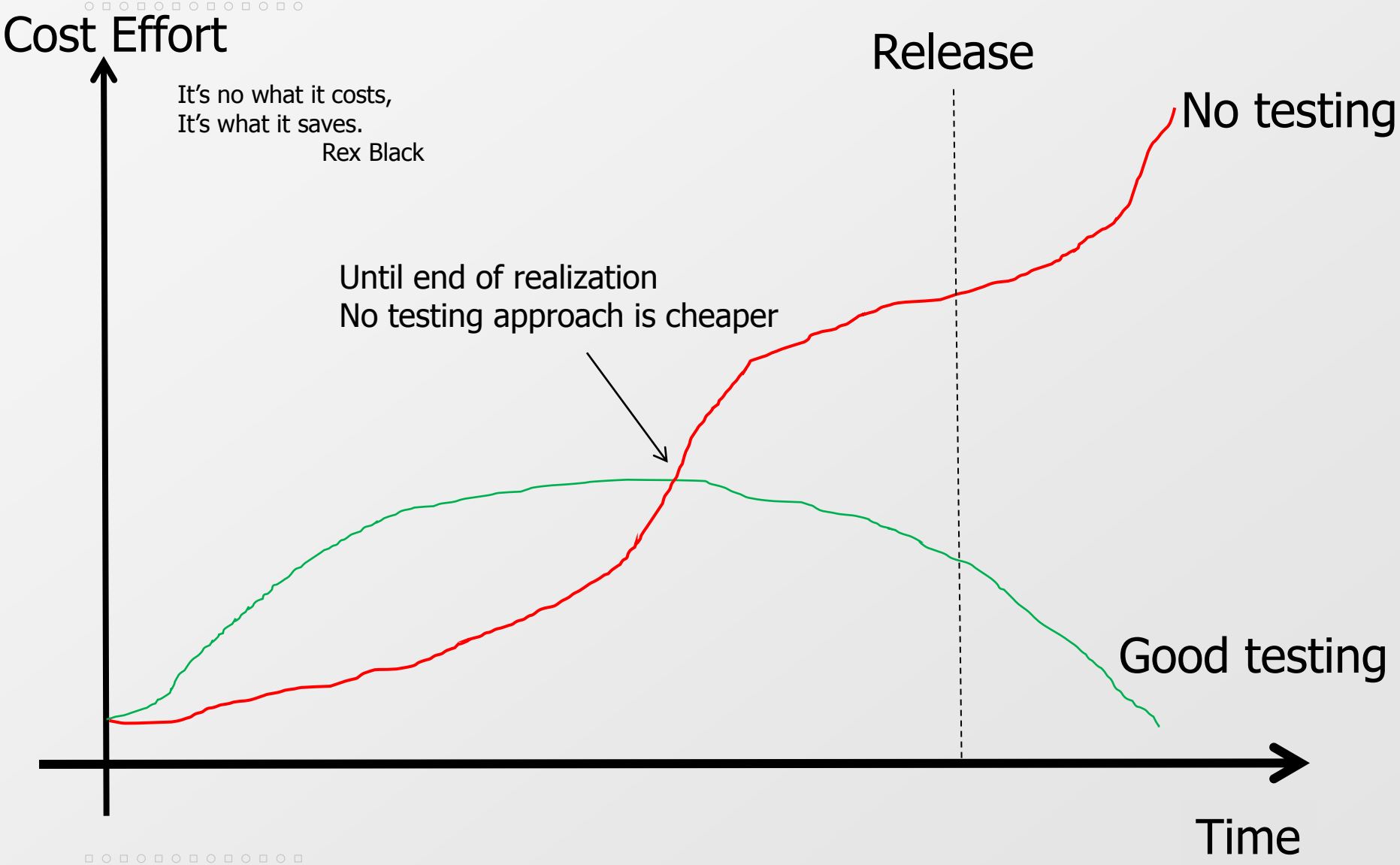
- Purpose of testing
- Basic test principles
- Test process
- Multilevel testing
- Static techniques
- Blackbox vs. Whitebox testing

Content (2)

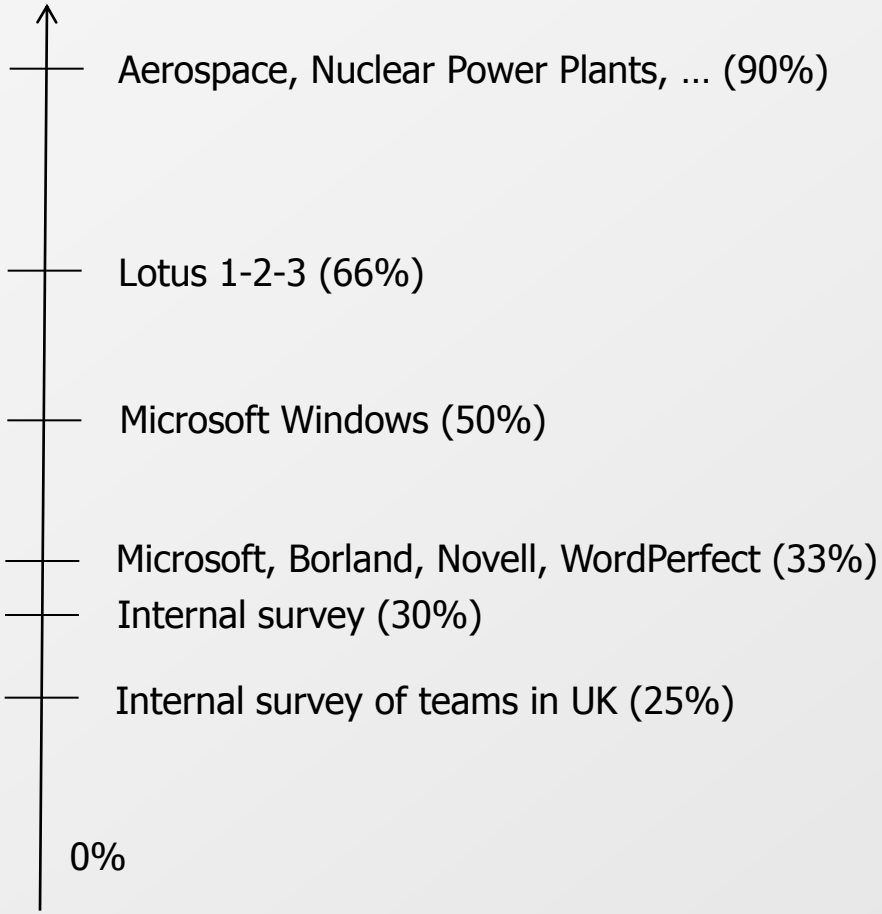
- Test plan document
- Risk-based testing strategy
- Test exit criteria
- Test-driven development
- Combinatorial testing
- Test automation and regression testing
- Test tools in praxis

Historical view




Cost Effort
It's no what it costs,
It's what it saves.
Rex Black
Release
No testing
Until end of realization
No testing approach is cheaper
Good testing
Time

Tests as % of overall development



Testing @ Microsoft

- ✓ Nearly 10 000 Testers
- ✓ Tester to Developer ration – roughly 1:1
- ✓ Flagship projects 10's of millions of lines code
- ✓ Millions of tests
  - ✓ 6.7 million automated tests in Vista
- ✓ Nearly 15 million entries into bug and project management tool every year

Purpose of testing

- Why testing ?
  - SW works unexpectly => many problems (loosing money, time or bussines reputation
  - Cause injury or death !
- Human activity -> error (mistake), which produces a defect (fault,bug) int the program code or documentation
- If defect is executed : system may fail (or do something it shouldn’t)
- Defects in software, system or document may result in failures, but not all defect do so

Purpose of testing

- Defect origin
  - Time pressure
  - Complex code
  - Complexity of infrastructure
  - Changing technologies
- Failure origin
  - Environmental conditions (radiation, magnetism, electronic fields, pollution …)
  - Change of HW conditions

Purpose of testing

- Role of testing in Software Development, Maintenance and Operations
  - Reduce risk of problems occuring during operation
  - Contribute to the quality of SW system
  - Requirement in contract, industry-specific standards
  - Quality assurance activity
- How much testing needed ?
  - Depends on the level of risk (technical, business, safety) and project constraints (time, budget)
  - Should provide sufficient information to stakeholders to make decision about the release for the next development phase or handover to the customers

Purpose of testing

- What is TESTING (SW product) ???
  - Common understanding : execution of SW
  - Testing activities
    - Planning and control
    - Choosing test conditions
    - Designing and executin test cases
    - Checking results
    - Evaluating exit criteria
    - Reporting results
    - Tracking bugs
    - Review of documents, source code, ...
    - Conducting static/dynamic analysis

Basic Testing Principles

- Principle 1 – Testing shows presence of defects
- Principle 2 – Exhaustive testing is impossible
- Principle 3 – Early testing
- Principle 4 – Defect clustering
- Principle 5 – Pesticide paradox
- Principle 6 – Testing is context dependent
- Principle 7 – Absence-of-errors fallacy

Test process

- Test planning and control
- Test analysis and design
- Test implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

Although logically sequential, the activities in the process may overlap or take place concurrently.

Tailoring these main activities within the context of the system and the project is usually required.

Test process – Test planning and control

- Planning
  - activity of defining the objectives of testing and the specification of test activities in order to meet the objectives and mission
- Controlling
  - ongoing activity of comparing actual progress against the plan
  - reporting the status, including deviations from the plan
  - involves taking actions necessary to meet the mission and objectives of the project
  - monitoring the testing activities throughout the project
- Note: Test planning takes into account the feedback from monitoring and control activities.

Test process – Test implementation and execution

- Test implementation and execution has the following major tasks:
  - Finalizing, implementing and prioritizing test cases (including the identification of test data)
  - Developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts
  - Creating test suites from the test procedures for efficient test execution
  - Verifying that the test environment has been set up correctly
  - Verifying and updating bi-directional traceability between the test basis and test cases
  - Executing test procedures either manually or by using test execution tools, according to the planned sequence

Test process – Test implementation and execution

- Test implementation and execution has the following major tasks:
  - Logging the outcome of test execution and recording the identities and versions of the software under test, test tools and testware
  - Comparing actual results with expected results
  - Reporting discrepancies as incidents and analyzing them in order to establish their cause (e.g. a defect in the code, in specified test data, in the test document, or a mistake in the way the test was executed)
  - Repeating test activities as a result of action taken for each discrepancy, for example, re-execution of a test that previously failed in order to confirm a fix (confirmation testing), execution of a corrected test and/or execution of tests in order to ensure that defects have not been introduced in unchanged are as of the software or that defect fixing did not uncover other defects (regression testing)

Test process – Evaluating exit criteria and reporting

- Evaluating exit criteria has the following major tasks:
  - Checking test logs against the exit criteria specified in test planning
  - Assessing if more tests are needed or if the exit criteria specified should be changed
  - Writing a test summary report for stakeholders

Test process – Test closure activities

- Test closure activities include the following major tasks:
  - Checking which planned deliverables have been delivered
  - Closing incident reports or raising change records for any that remain open
  - Documenting the acceptance of the system
  - Finalizing and archiving testware, the test environment and the test infrastructure for later reuse
  - Handing over the testware to the maintenance organization
  - Analyzing lessons learned to determine changes needed for future releases and projects
  - Using the information gathered to improve test maturity

Multilevel testing

- A common type off V-model uses four test levels, corresponding to the four development levels.
  - Component (unit) testing
  - Integration testing
  - System testing
  - Acceptance testing

Multilevel testing – testing within a life cycle model

- In any life cycle model, there are several characteristics of good testing:
  - For every development activity there is a corresponding testing activity
  - Each test level has test objectives specific to that level
  - The analysis and design of tests for a given test level should begin during the corresponding development activity
  - Testers should be involved inn reviewing documents as soon as drafts are available in the development life cycle

Multilevel testing – Component Testing

- Test basis:
  - Component requirements
  - Detailed design
  - Code
- Typical test objects:
  - Components
  - Programs
  - Data conversion / migration programs
  - Database modules

Multilevel testing – Integration Testing

- Test basis:
  - Software and system design
  - Architecture
  - Workflows
  - Use cases

- Typical test objects:
  - Subsystems
  - Database implementation
  - Infrastructure
  - Interfaces
  - System configuration and configuration data

Multilevel testing – System Testing

- Test basis:
  - System and software requirement specification
  - Use cases
  - Functional specificaton
  - Risk analysis reports
- Typical test objects:
  - System, user and operation manuals
  - System configuration and configuration data

Multilevel testing – Acceptance Testing

- Test basis:
  - User requirements
  - System requirements
  - Use cases
  - Business processes
  - Risk analysis reports
- Typical test objects:
  - Business processes on fully integrated system
  - Operational and maintenance processes
  - User procedures
  - Forms
  - Reports
  - Configuration data

Multilevel testing – Alpha and Beta testing

Developers of market, software often want to get feedback from potential or existing customers in their market before the software product is put up for sale commercially.

Alpha testing is performed at the developing organization's site but not by the developing team.

Beta testing, or field-testing, is performed by customers or potential customers at their own locations.

Statique techniques

Unlike dynamic testing, which requires the execution of software, static testing techniques rely on the manual examination (reviews) and automated analysis (static analysis) of the code or other project documentation without the execution of the code.

Reviews are a way of testing software work products (including code)) and can be performed well before dynamic test execution. Defects detected during reviews early in the life cycle (e.g., defects found in requirements) are often much cheaper to remove than those detected by running tests on the executing code.

A review could be done entirely as a manual activity, but there is also tool support. The main manual activity is to examine a work product and make comments about it. Any software work product can be reviewed, including requirements specifications, design specifications, code, test plans, test specifications, test cases, test scripts, user guides or web pages.

Statique techniques

Benefits of reviews include early defect detection and correction, development productivity improvements, reduced development timescales, reduced testing cost and time, lifetime cost reductions, fewer defects and improved communication. Reviews can find omissions, for example, in requirements, which are unlikely to be found in dynamic testing.

Reviews, static analysis and dynamic testing have the same objective – identifying defects. They are complementary; the different techniques can find different types of defects effectively and efficiently. Compared to dynamic testing, static techniques find causes of failures (defects) rather than the failures themselves.

Typical defects that are easier to find in reviews than in dynamic testing include: deviations from standards, requirement defects, design defects, insufficient maintainability and incorrect interface specifications.

Statique techniques - Review

- Roles and Responsibilities
  - Manager: decides on the execution of reviews, allocates time in project schedules and determines if thee review objectives have been met.
  - Moderator: the person who leads the review of the document or set of documents, including planning the review, running the meeting, and following-up after the meeting. If necessary, the moderator may mediate between the various points of view and is often the person upon whom the success of the review rests.
  - Author: the writer or person with chief responsibility for the document(s) to be reviewed.
  - Reviewers: individuals with a specific technical or business background (also called checkers or inspectors) who, after the necessary preparation, identify and describe findings (e.g., defects) in the product under review. Reviewers should be chosen to represent different perspectives and roles in the review process, and should take part in any review meetings.
  - Scribe (or recorder): documents all the issues, problems and open points that were identified during the meeting.

Statique techniques – Types of Review

- Informal Review
  - No formal process
  - May take the form of pair programming or a technical lead reviewing designs and code
  - Results may be documented
  - Varies in usefulness depending on the reviewers
  - Main purpose: inexpensive way to get some benefit

Statique techniques – Types of Review

- **Walkthrough**
  - Meeting led by author
  - May take the form of scenarios, dry runs, peer group participation
  - Open-ended sessions
    - Optional pre-meeting preparation of reviewers
    - Optional preparation of a review report including list of findings
  - Optional scribe (who is not the author)
  - May vary in practice from quite informal to very formal
  - Main purposes: learning, gaining understanding, finding defects

Statique techniques – Types of Review

- Technical Review
  - Documented, defined defect--detection process that includes peers and technical experts with optional management participation
  - May be performed as a peer review without management participation
  - Ideally led by trained moderator (not the author)
  - Pre-meeting preparation by reviewers
  - Optional use of checklists
  - Preparation of a review report which includes the list of findings, the verdict whether the software product meets its requirements and, where appropriate, recommendations related to findings
  - May vary in practice from quite informal to very formal
  - Main purposes: discussing, making decisions, evaluating alternatives, finding defects, solving technical problems and checking conformance to specifications, plans, regulations, and standards

Statique techniques – Types of Review

- Inspection
  - Led by trained moderator (not the author)
  - Usually conducted as a peer examination
  - Defined roles
  - Includes metrics gathering
  - Formal process based on rules and checklists
  - Specified entry and exit criteria for acceptance of the software product
  - Pre-meeting preparation
  - Inspection report including list of findings
  - Formal follow-up process (with optional process improvement components)
  - Main purpose: finding defects

Statique techniques – Static code analysis

- The value of static analysis
  - Early detection of defects prior to test execution
  - Early warning about suspicious aspects of the code or design by the calculation of metrics, such as a high complexity measure
  - Identification of defects not easily found by dynamic testing
  - Detecting dependencies and inconsistencies in software models such as links
  - Improved maintainability of code and design
  - Prevention of defects, if lessons are learned in development

Statique techniques – Static code analysis

- Typical defects discovered by static analysis tools include:
  - Referencing a variable with an undefined value
  - Inconsistent interfaces between modules and components
  - Variables that are not used or are improperly declared
  - Unreachable (dead) code
  - Missing and erroneous logic (potentially infinite loops)
  - Overly complicated constructs
  - Programming standards violations
  - Security vulnerabilities
  - Syntax violations of code and software models

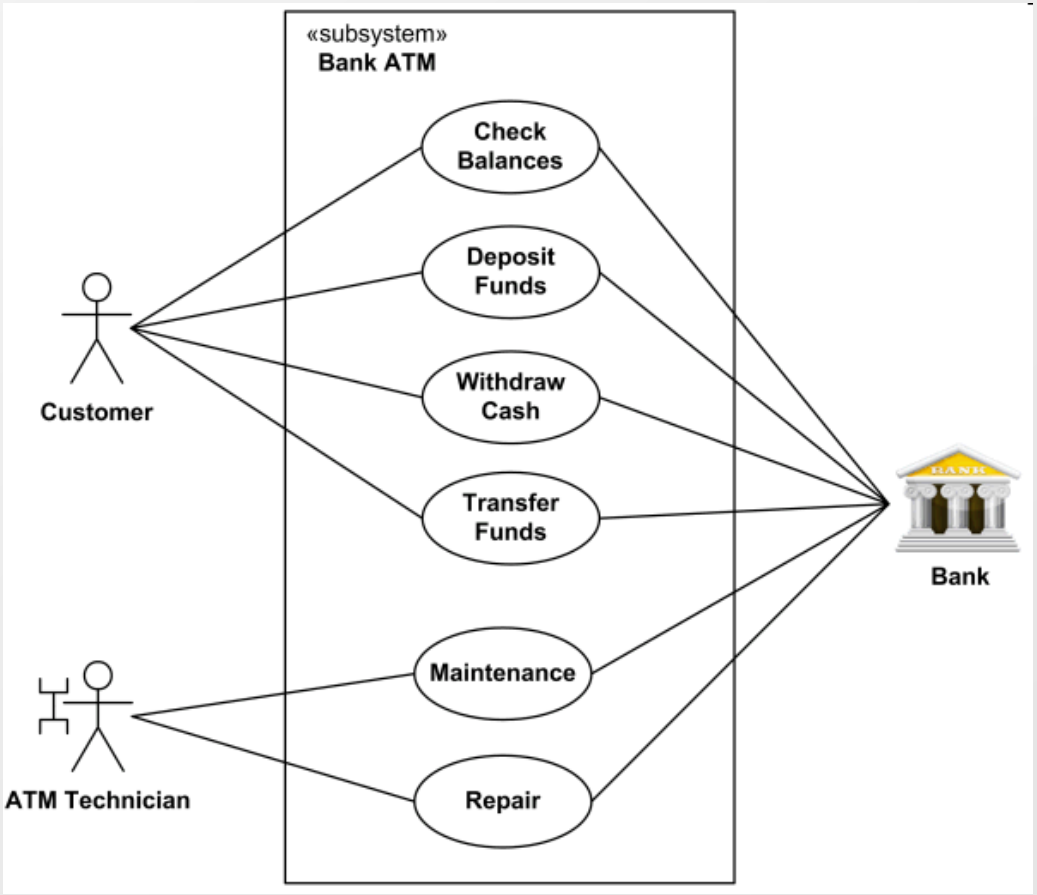
Blackbox testing

- Requirements-based

Objectives / Inventories	TestCase1	TestCase2	TestCase3	TestCase4	…
Req. 1	X		X		
Req. 2		X			
Req. 3				X	
Req. 4			X		
…					

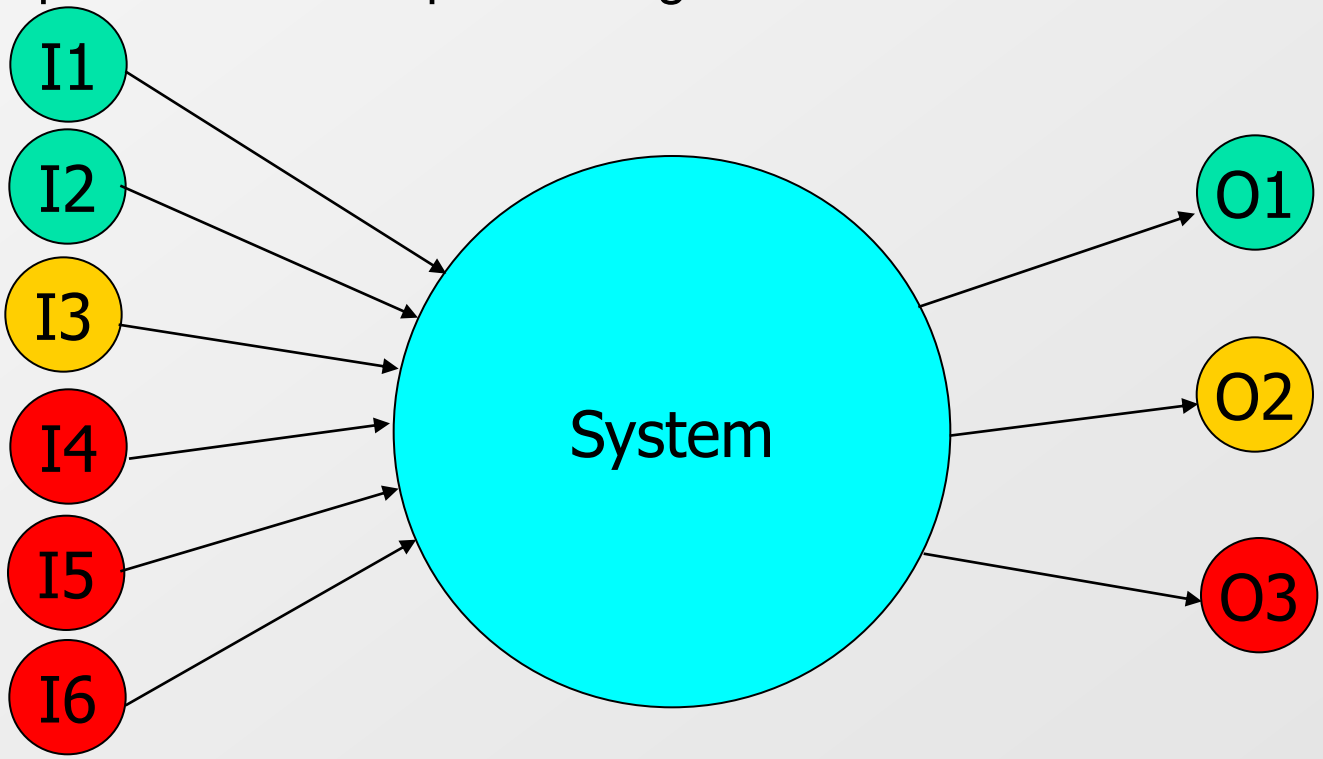
Blackbox testing

- UseCase-based



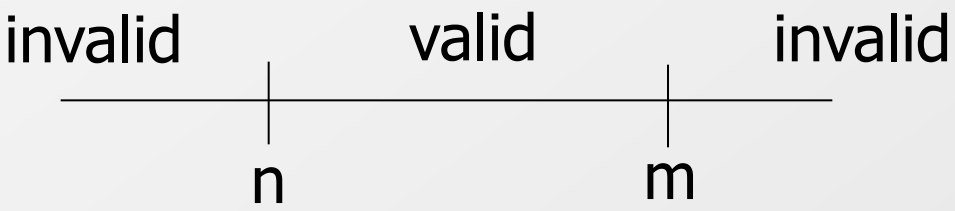
Blackbox testing

- Equivalence class partitioning



Blackbox testing

- Boundary value analysis

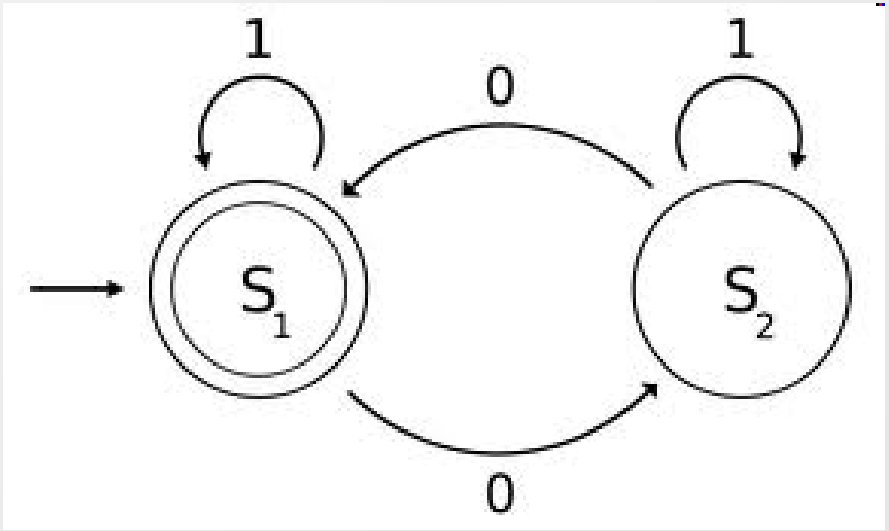


Boundary values: n,m
Test: n-1,n+1 & m-1,m+1

Blackbox testing

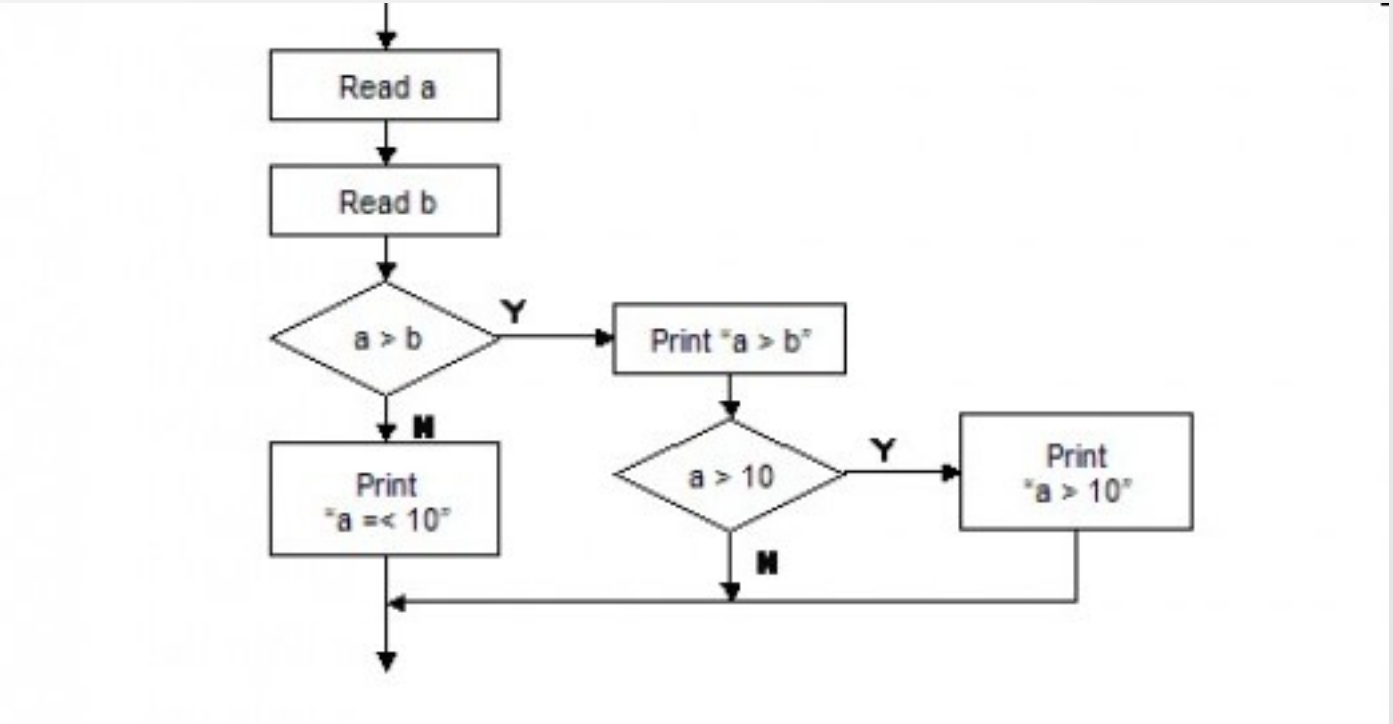
- State transition based

Input/state	0	1
S1	S?	S?
S2	S?	S?



Whitebox testing

- Statement / Decision testing



Whitebox testing

Test 1

First, we shall run the code with the values: 30 and 20. We expect the output to look like:

a > b

a > 10

Using this input data we execute the following statements:

1. Read a // a = 30
2. Read b // a = 30, b = 20
3. IF a > b THEN // True
4. Print "a > b" // "a > b“
5. IF a > 10 THEN // True
6. Print "a > 10" // "a > 10“

This test has executed all executable statements except the Print statement “Print a<=10”. This has exercised 6 of the 7 executable statements of the program, that is 6/7 or 86% statement coverage.

Whitebox testing

Test 2

To execute the final statement, we shall run the code with the values: 20 and 30. We expect the output to look like:

a <= b

Using this input data we execute the following statements:

```
Read a                  // a = 30
Read b                  // a = 30, b = 20
IF a > b THEN           // False
Print "a =< b"          // "a > b“
```

By performing both two tests (Tests 1 and 2) we can execute all statements of the code, achieving 100% statement coverage.