MASARYKOVA UNIVERZITA

PV213 Enterprise Information Systems in Practice

09 – Testing

MASARYKOVA UNIVERZITA

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE

esf

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE esf

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost 2007-13

UNIVERSITAS MASARYKIANA BRUNENSIS

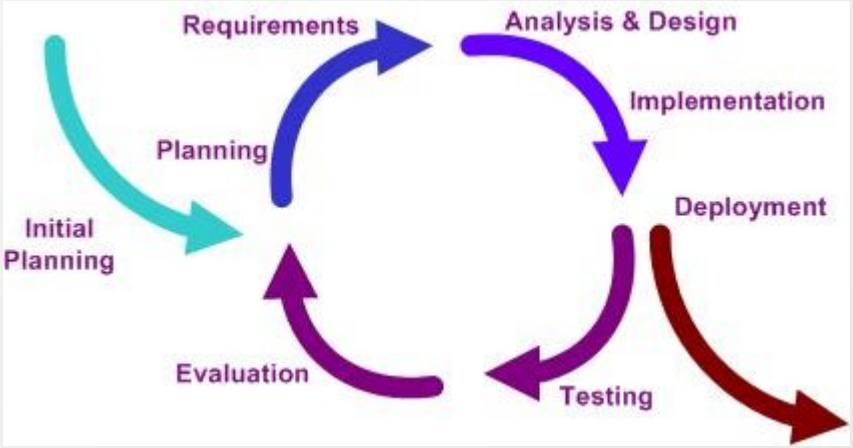
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

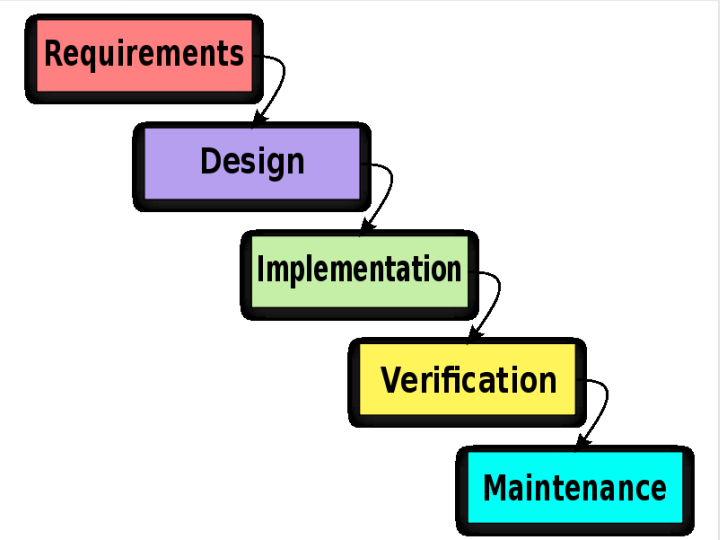
Content of this presentation

- Purpose of testing
- Test process
- Multilevel testing
- Static techniques
- Blackbox and Whitebox testing
- Risk-based testing strategy
- Test-driven development
- Test automation and regression testing

Testing

- Process of validating and verifying that a product fits requirements





Reason for testing

- Human activity → errors
- SW works unexpectly
  - system may fail (or do something it shouldn’t)
  - loosing money
  - time or business reputation
  - injury, even death

- Defects in software, system or document may result in failures, but not all defect do so

AT&T Phone System Crash, 1990

- What happened
  - Mal-function in central server led to chain reaction
  - Service outage of half of the system for 9 hours
  - Loss of 75 million dollars damage for AT&T
- Reasons
  - Wrong usage of break command
  - Software update directly in largest part of the system

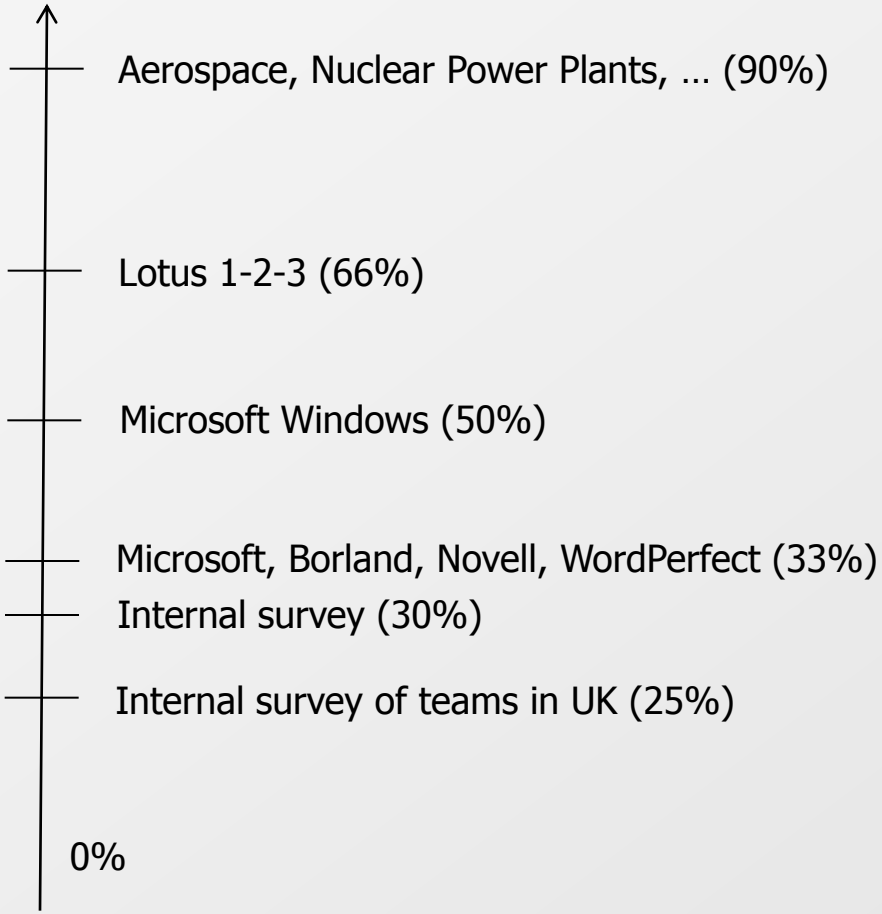
```
switch expression {
  ...
  case (value):
      if (logical) {
          statement;
          break;
      } else {
          statement;
      }
      statement;
    ...
}
```

Role of testing

- Reduce risk of problems occurring during operation
- Quality assurance activity
- Requirement in contract, industry-specific standards

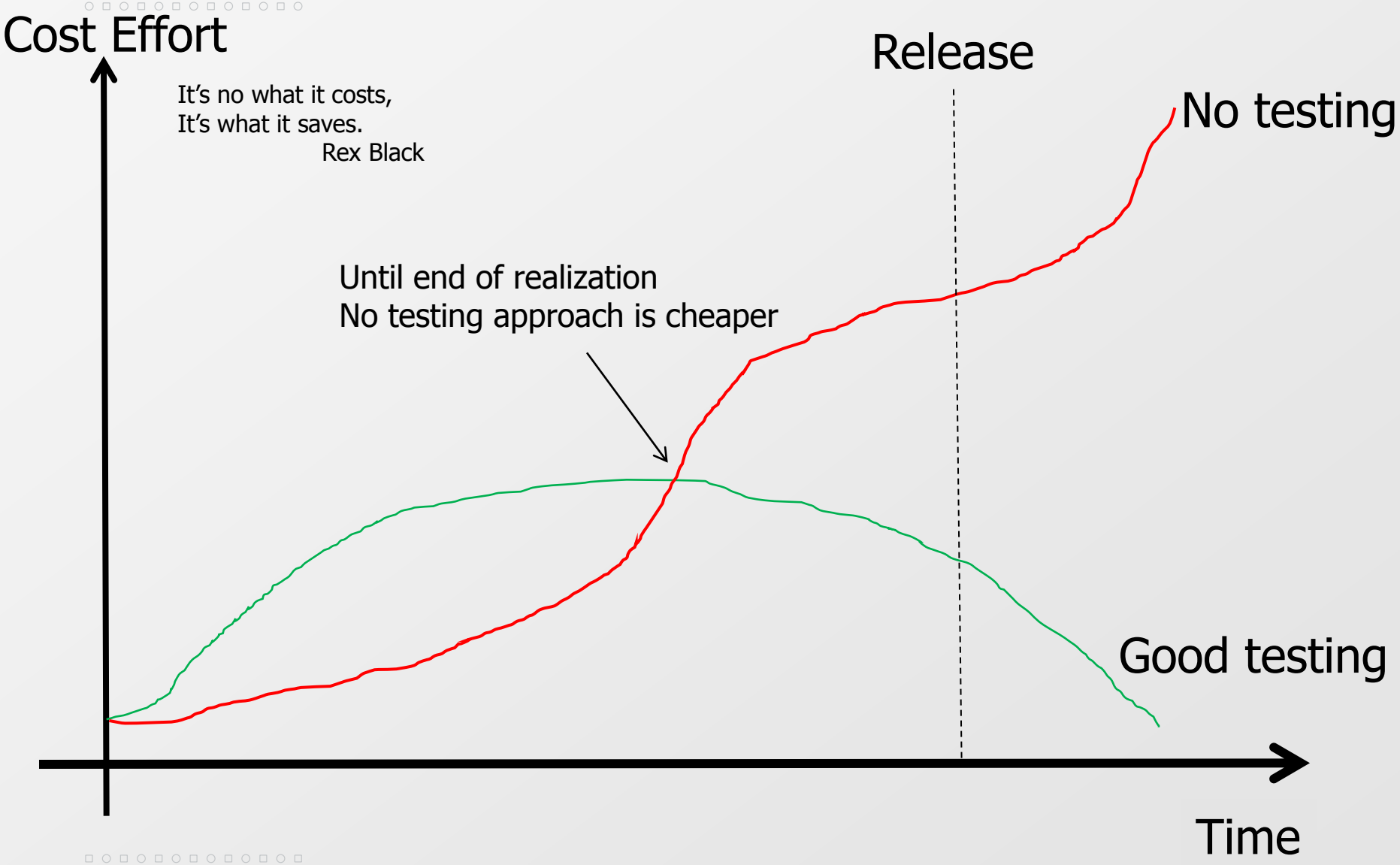
- How much testing needed ?
  - Depends on the level of risk (technical, business, safety) and project constraints (time, budget)
  - Should provide sufficient information to stakeholders to make decision about the release for the next development phase or handover to the customers

Tests as % of overall development



Testing activities

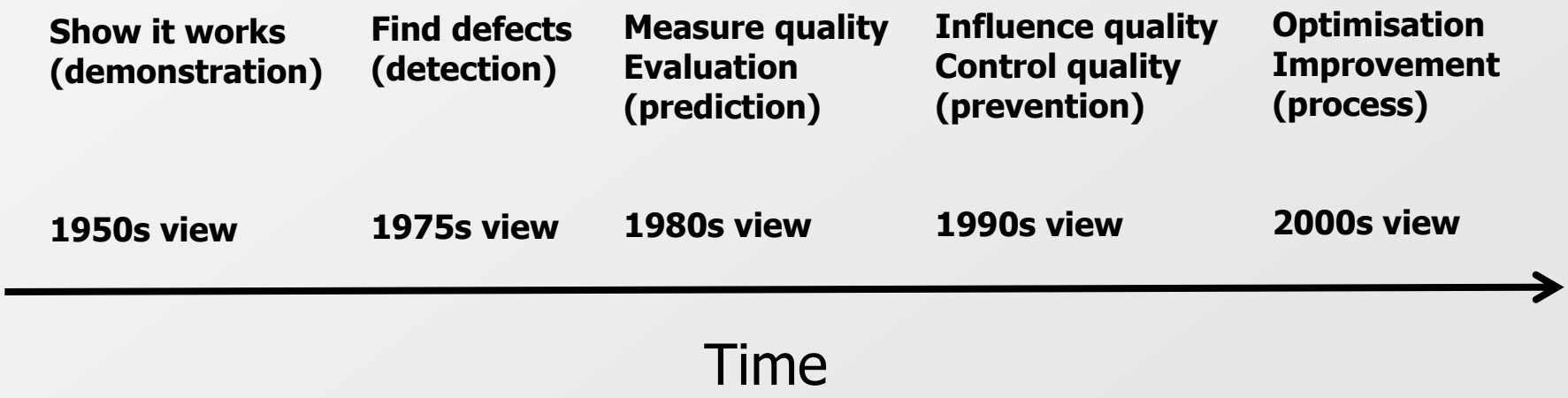
- Planning and control
- Choosing test approach
- Design and test case execution
- Checking results
- Evaluating exit criteria
- Reporting results
- Tracking bugs
- Review of documents, source code, ...
- Conducting static/dynamic analysis


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

Basic Testing Principles

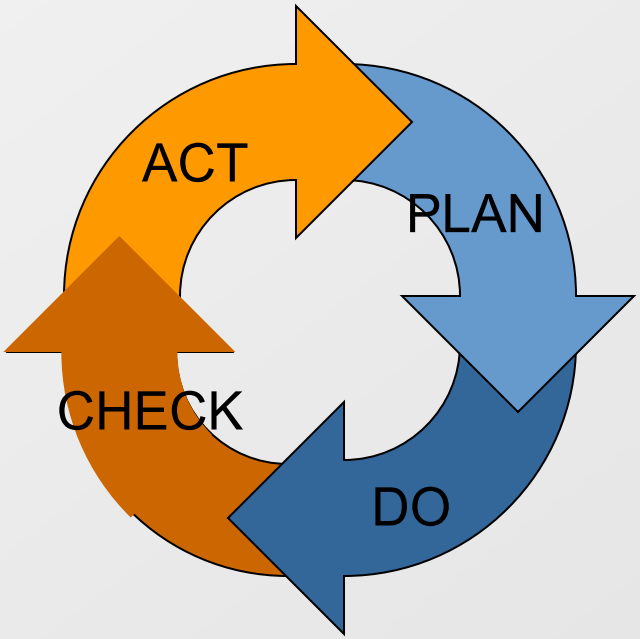
- Principle 1 – Testing shows presence of defects
- Principle 2 – Exhaustive testing is impossible
- Principle 3 – Early testing
- Principle 4 – Defect clustering
- Principle 5 – Pesticide paradox
- Principle 6 – Testing is context dependent
- Principle 7 – Absence-of-errors fallacy

Historical view



Test process

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

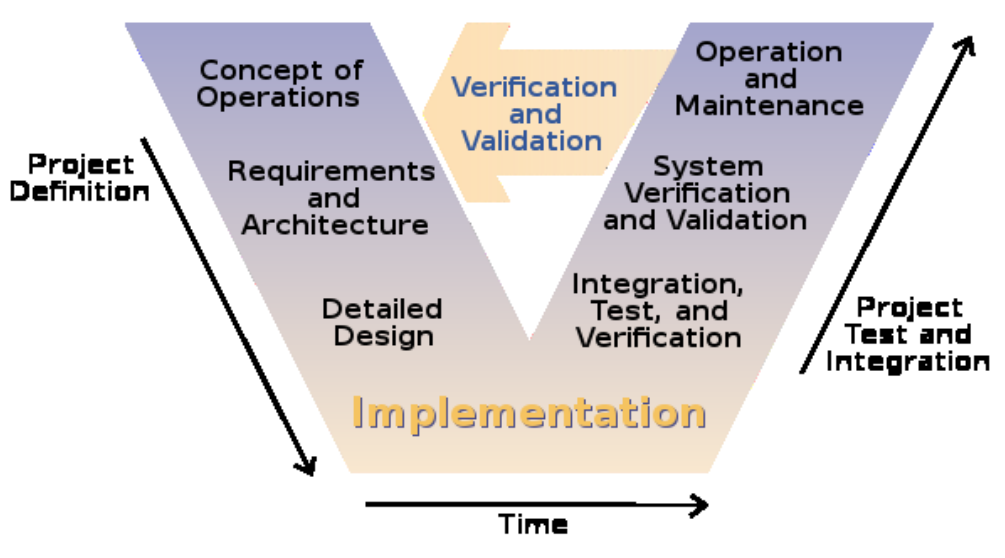


Although logically sequential, the activities in the process may overlap or take place concurrently.

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

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 – 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.

Test exit criteria – unit and integration tests examples

- All unit and integration tests and results are documented
- There can be no High severity bugs
- There must be 100% statement coverage
- There must be 100% coverage of all programming specifications
- The results of a code walkthrough and they are documented and acceptable

Test exit criteria – system tests examples

- All test cases must be documented and run
- 90% of all test cases must pass
- All test cases high risk must pass
- All medium and high defects must be fixed
- Code coverage must be at least 90% (including unit and integration tests)

Test exit criteria – acceptance tests examples

- There can be no medium or high severity bugs
- There can be no more than 2 bugs in any one feature or 50 bugs total
- At least one test case must pass for every requirement
- Test cases 23, 25 and 38-72 must all pass
- 8 out of 10 experienced bank clerks must be able to process a loan document in 1 hour or less using only the on-line help system
- The system must be able to process 1000 loan applications/hour
- The system must be able to provide an average response time of under 1 second per screen shift with up to 100 users on the system
- The users must sign off on the results

Test exit criteria – also possible

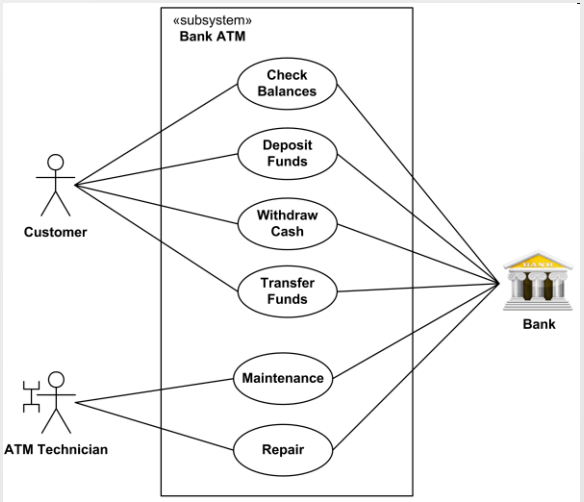
- Time is over
- Budget is used up
- The boss says “ship it!”
- Testing is never finished, it’s stopped!
- Software products are never released, they escape!

Static techniques: Static code analysis

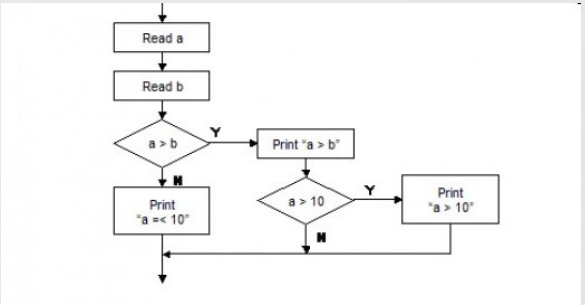
- Typical defects discovered by static analysis tools include:
  - Referencing a variable with an undefined value
  - Inconsistent interfaces between modules and components
  - Variables that are not used or are improperly declared
  - Unreachable (dead) code
  - Missing and erroneous logic (potentially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

Black box and white box testing

- **Black box testing**
  - No knowledge of internal implementation (user view)



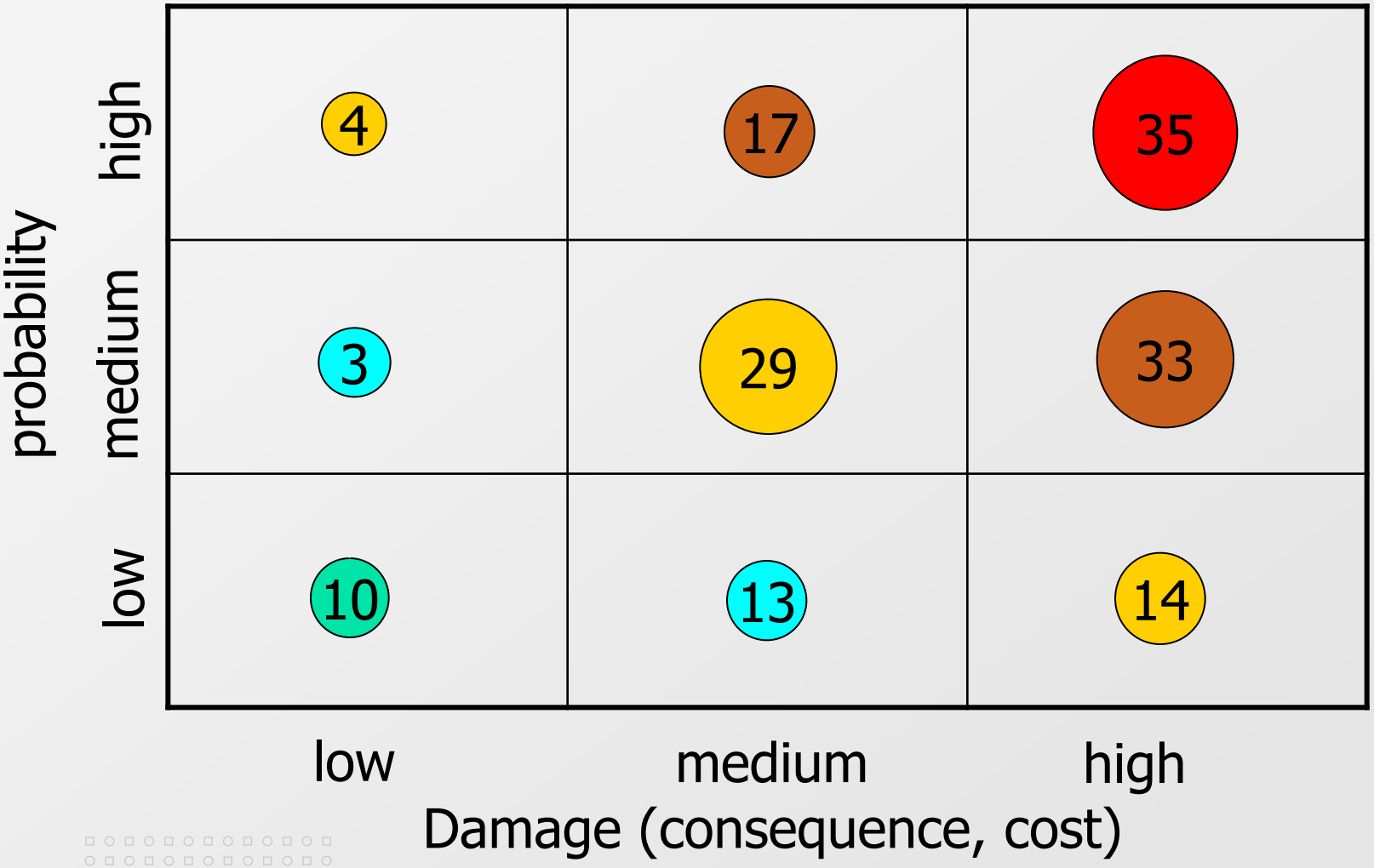
- **White box testing**
  - Internal implementation is know and usually also tested

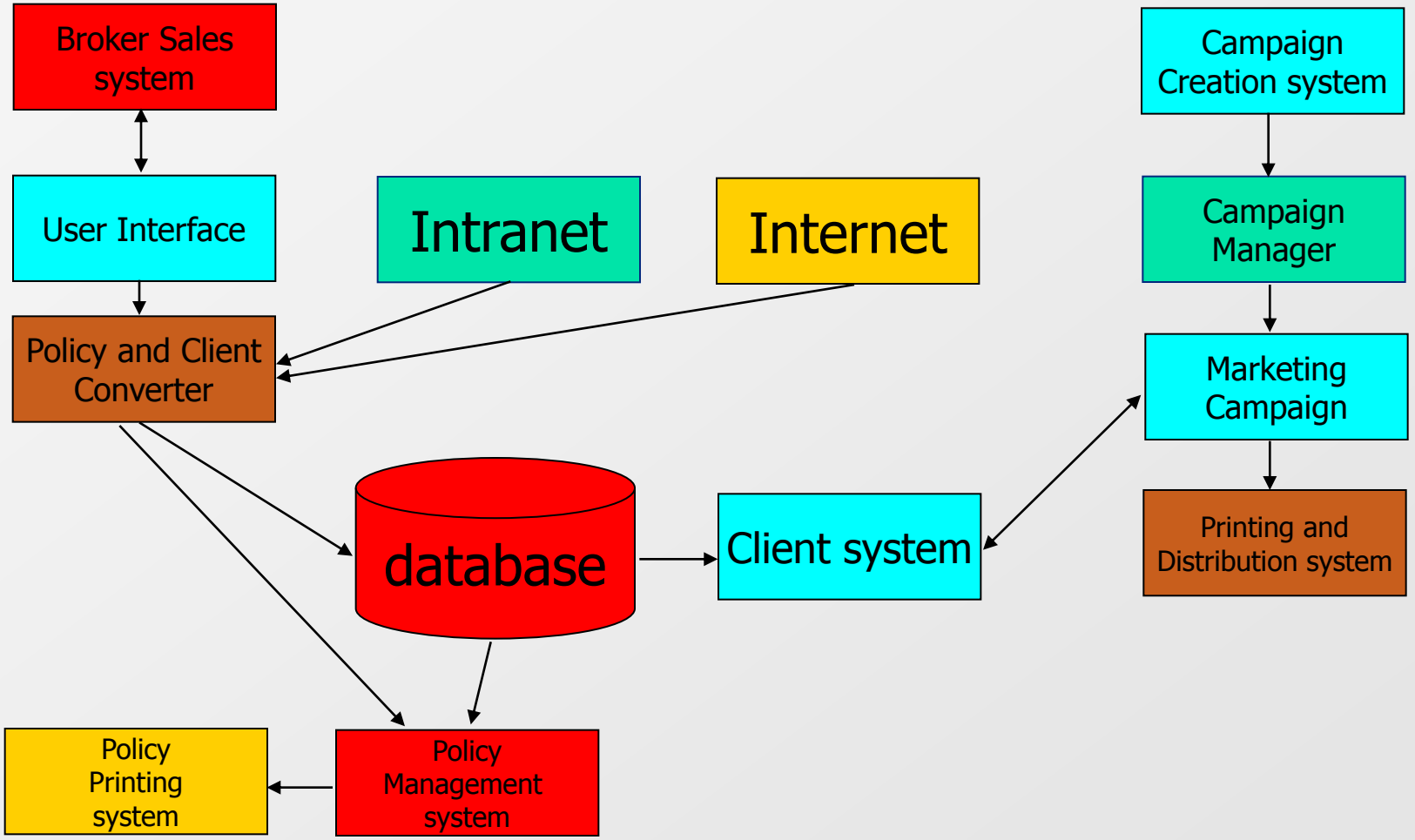


Risk-based testing strategy

- Base the testing strategy on business goals and priorities => Risk-based testing strategy

- No risk = No test
- Risk = P x D
  - P … probability of failure
  - D … damage (consequnce & cost for business & test & usage)


probability
high
medium
low
4
17
35
3
29
33
10
13
14
low
medium
high
Damage (consequence, cost)

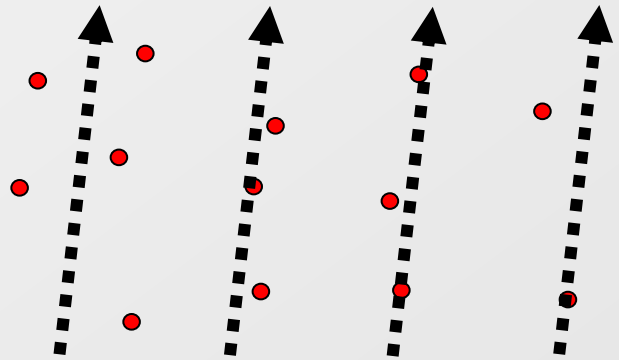

Broker Sales system
User Interface
Intranet
Internet
Policy and Client Converter
database
Client system
Campaign Creation system
Campaign Manager
Marketing Campaign
Printing and Distribution system
Policy Printing system
Policy Management system

Test data selection

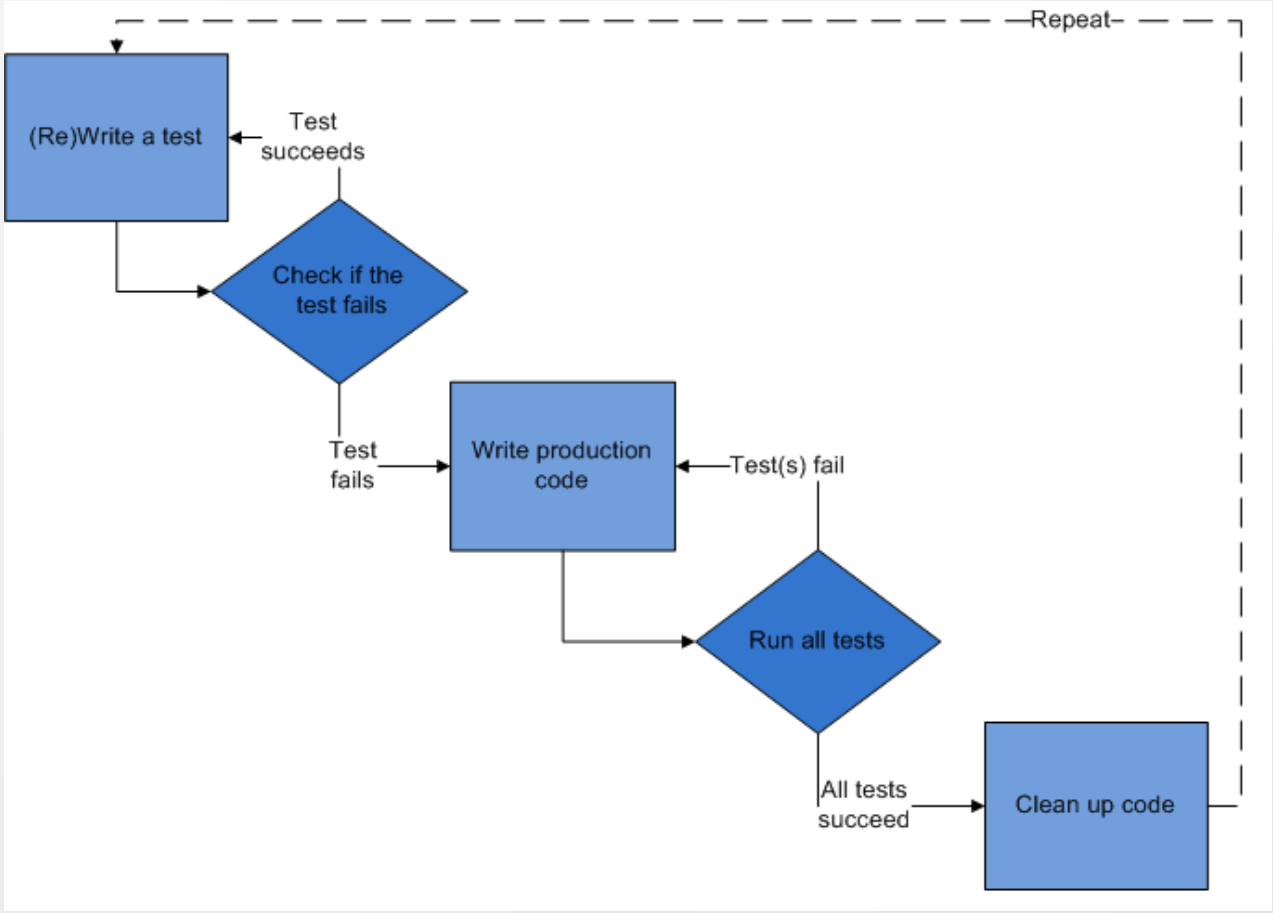
- Best guess
  - Intuition and hope and luck
- Random choice
  - Expert know-how
- All combinations
  - Every combination used in test cases
  - Suitable for trivial cases
- Each choice
  - Each value of each parameter to be included in at least one test case

Test Automation

- SW to control test execution
  - Code driven testing
  - Graphical user interface testing
- Limitations (Minefield metaphor)



Test-Driven Development (TDD)



TDD - calculator example

x	y	add()	subtract()	multiply()	divide()
0	0	0	0	0	error
1	1	2	0	1	1
4	2	6	2	8	2
9	3	12	6	27	3
35	5	40	30	175	7

```
public class CalculatorFixture extends ColumnFixture {
  public int x;
  public int y;
  public int add() { return 0; }
  public int subtract() { return 0; }
  public int multiply() { return 0; }
  public int divide() { return 0; }
}
```

TDD - calculator example

x	y	add()	subtract()	multiply()	divide()
0	0	0	0	0	expected: error
1	1	expected: 2 actual: 0	0	expected: 1 actual: 0	expected: 1 actual: 0
4	2	expected: 6 actual: 0	expected: 2 actual: 0	expected: 8 actual: 0	expected: 2 actual: 0
9	3	expected: 12 actual: 0	expected: 6 actual: 0	expected: 27 actual: 0	expected: 3 actual: 0
35	5	expected: 40 actual: 0	expected: 30 actual: 0	expected: 175 actual: 0	expected: 7 actual: 0

```
public class CalculatorFixture extends ColumnFixture {
  public int x;
  public int y;
  public int add() { return 0; }
  public int subtract() { return 0; }
  public int multiply() { return 0; }
  public int divide() { return 0; }
}
```

TDD - calculator example

x	y	add()	subtract()	multiply()	divide()
0	0	0	0	0	expected: error
1	1	2	0	expected: 1 actual: 0	expected: 1 actual: 0
4	2	6	2	expected: 8 actual: 0	expected: 2 actual: 0
9	3	12	6	expected: 27 actual: 0	expected: 3 actual: 0
35	5	40	30	expected: 175 actual: 0	expected: 7 actual: 0

```
public class CalculatorFixture extends ColumnFixture {
  public int x;
  public int y;
  public int add() { return x+y; }
  public int subtract() { return x-y; }
  public int multiply() { return 0; }
  public int divide() { return 0; }
}
```

Result of the refactoring

```
public class Calculator {
public int plus(x, y) { return x + y; }
public int minus(x, y) { return x - y; }
public int times(x, y) { return x * y; }
public int divide(x, y) { return x / y; }
}
public class CalculatorFixture extends ColumnFixture {
  public int x;
  public int y;
  private Calculator calc;
  public CalculatorFixture() { calc = new Calculator(); }
  public int add() { return calc.plus(x,y); }
  public int subtract() { return calc.minus(x,y); }
  public int multiply() { return calc.times(x,y); }
  public int divide() { return calc.divide(x,y); }
}
```

Regression Testing

- The fundamental problem with software maintenance is that fixing a defect has a substantial (20-50 %) chance of introducing another.

  Frederick P. Brooks, Jr., 1995

- When you fix one bug, you introduce several newer bugs.
- ISTQB Glossary (2007)
  - Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

Regression Testing – test selection strategy

- Retest all
- Retest by risk – priority, severity, criticality
- Retest by profile (frequency of usage)
- Retest changed parts
- Retest parts that are influenced by changes

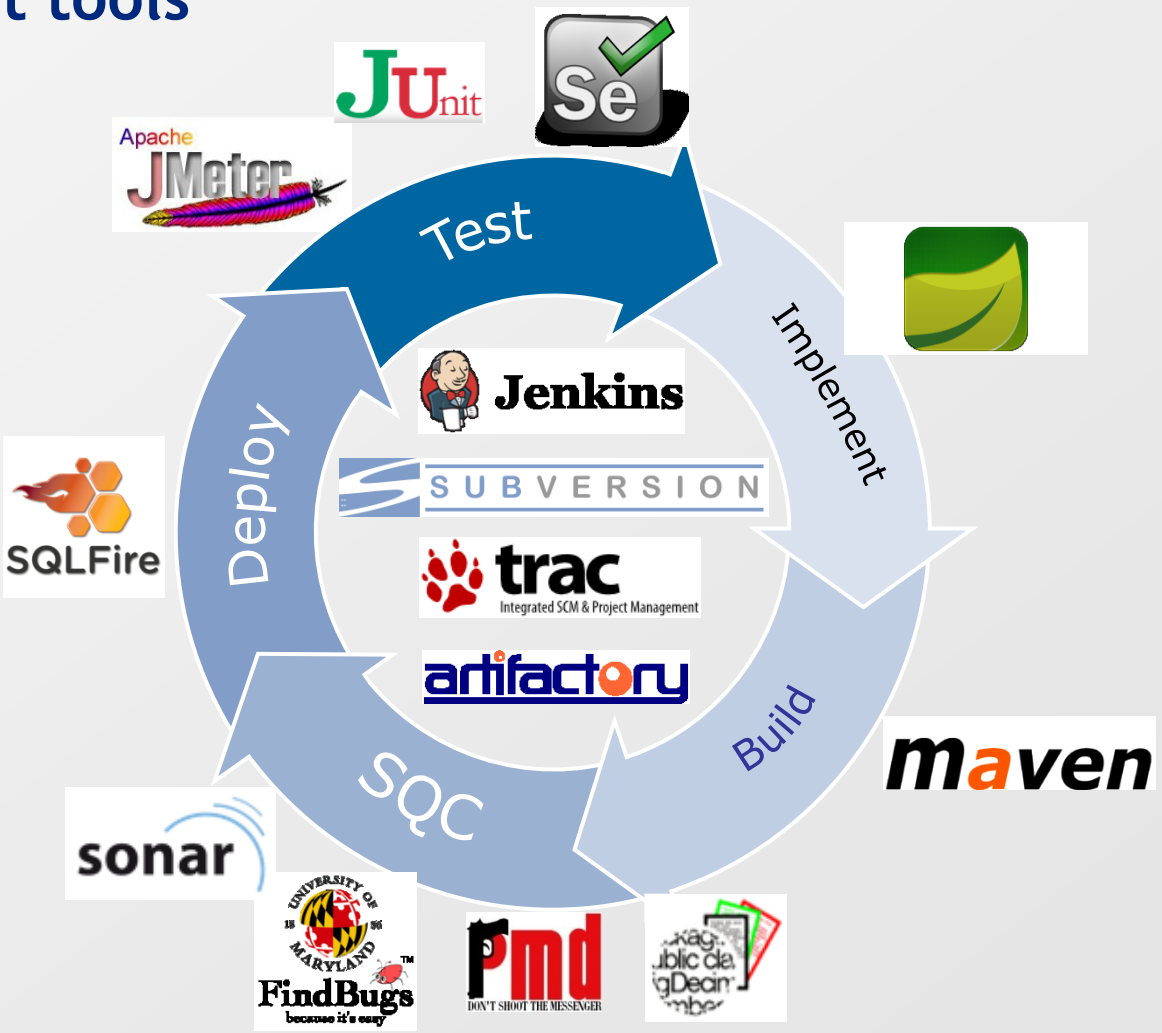
Some characteristics of good testing

- 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 in reviewing documents as soon as drafts are available in the development life cycle

Continuous Integration example

- Production
- Module Tests, run in "common mode"
  - 4,000
  - Note: PureCov & Purify on a weekly base
- Automated Installation
- Pre-Integration Tests
  - 100 (as far as they fit into the 24h-frame)
  - Sanity Tests (on target-machine)
- Bring-Up & Sanity Tests
  - Plus Integration Regression Tests that fit into a 24-hours run
- Integration tests
  - Cca 1,800 fully automated regression and feature tests

Development tools example



Certifications

- ISTQB (International Software Testing Qualifications Board)

Next lessons

- 21.4. Introduction to Safety Critical Software Development in Aerospace (Radek)
- 28.4. Retrospective, final game (Ales)

Děkuji za pozornost.

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE esf

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost 2007-13

UNIVERSITAS MASARYKIANA BRUNENSIS

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ