Automation of tests

Course at TU Darmstadt

Dr. Marc Lohmann, 18-May-2015

.consulting .solutions .partnership

Dr. Marc Lohmann

- Study and PhD in computer science (Informatik) at University of Paderborn

- 2001 – 2006: Research associate at University of Paderborn
- 2006 – 2011: software developer and architect at sd&m
  (mainly projects in public sector, Government)

- Since 2011 at msg systems ag
  - Business Consultant
  - Telecommunications and Media

- Actually, I am working as consultant for a manufacturer of an IPTV platform.

Private

- 43 years old, married, 2 Kids
- Hobbies: family, refurbishment of our house, running

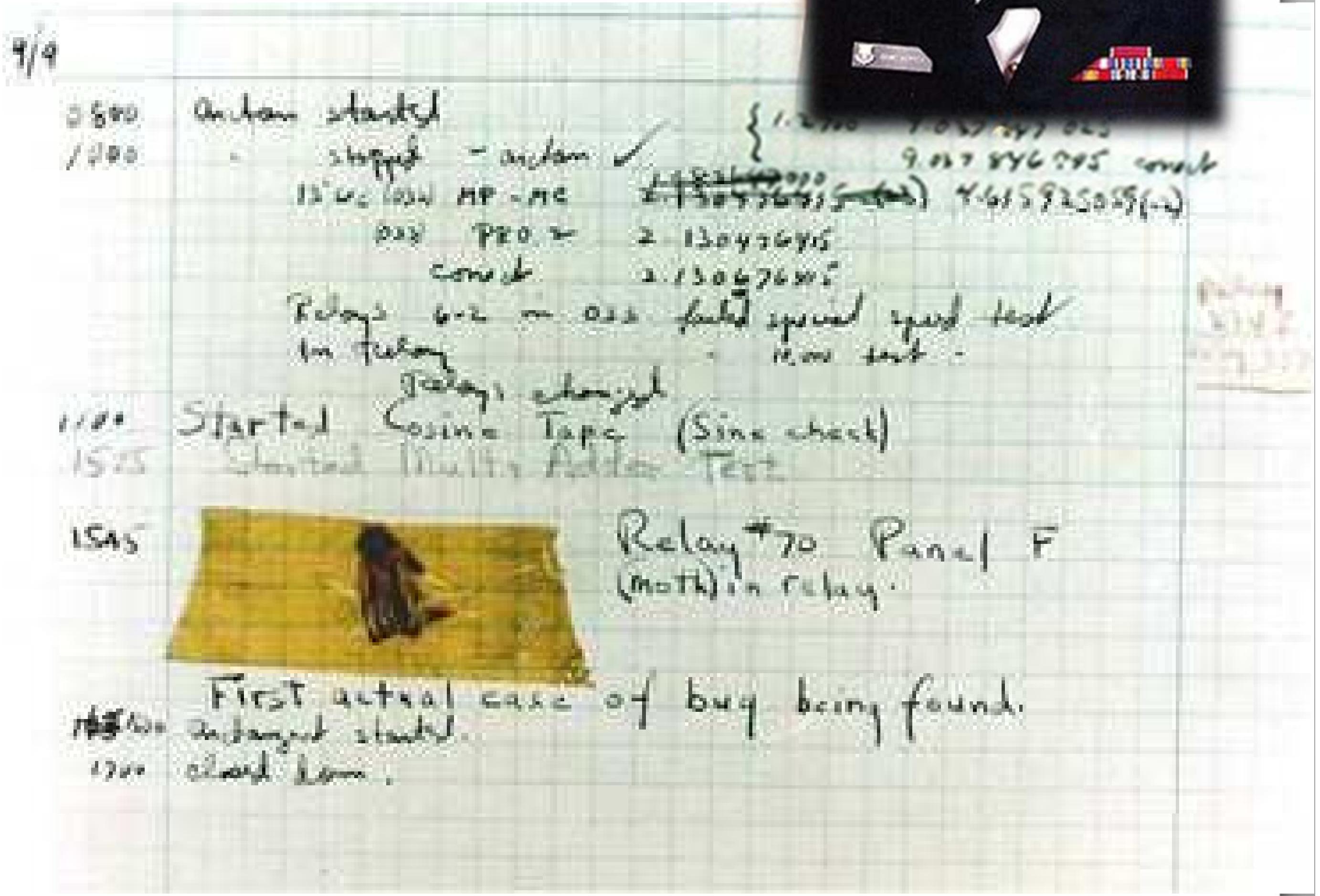
AGENDA

09.09.1945: First Bug

- A moth lead to a downtime of Mark II calculator.
- Mrs. Grace Murray Hopper wrote in log book:

  »First actual case of bug being found.«
- Localization and elimination of bug was easy!

(Wikipedia)


9/9

0800 Antan started

1000 " stopped - antan ✓

13" uc (033) MP - MC [illegible] 9.037 846 995 conect

~~[illegible]~~ 4.615925059(-2)

(033) PRO 2 2.130476415

conect 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multy Adder Test.

1545 Relay #70 Panel F

(moth) in relay.

First actual case of bug being found.

1630 antangent started.

1700 closed down.

Log book of Mark II Aiken Relay Calculator (Wikipedia)

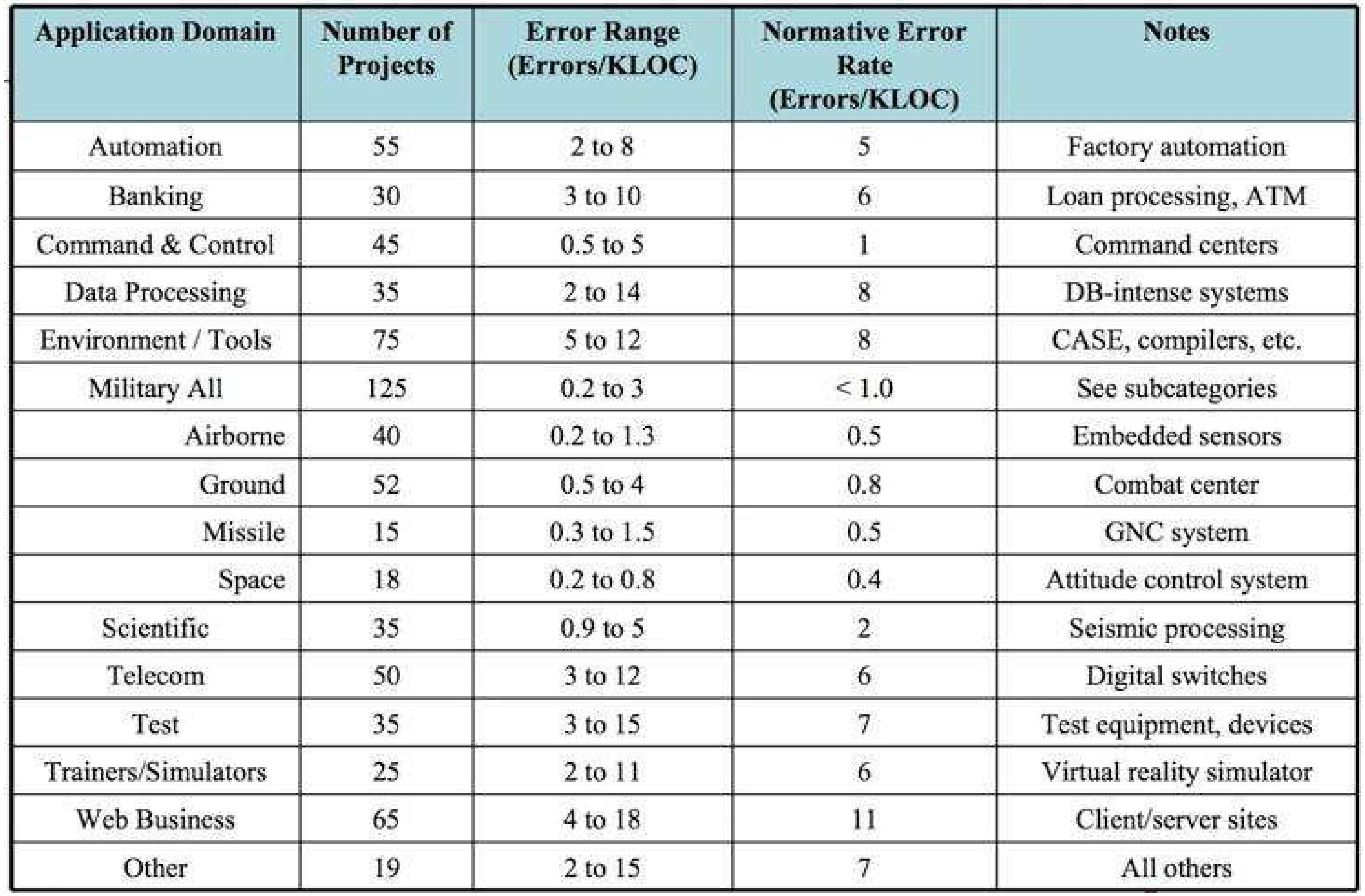
[Spillner, HS Bremen, 2008]

Bug due to reuse of software

- US Air Force developed rocket control program
- Due to storage capacity problems the rocket control program did not recalculate the flight coordinates, if the rocket crossed the equator. The program only reversed the plus/minus sign of the coordinates.
- Thus, the rocket moved around his own axis, if it crossed the equator.
  Not important, only a rocket.
- Rocket control program was reused as part of the autopilot of the F-18.
- What do you think, what happened?

[Spillner, HS Bremen, 2008]

Frequency of errors by application domains in the US



Application Domain	Number of Projects	Error Range (Errors/KLOC)	Normative Error Rate (Errors/KLOC)	Notes
Automation	55	2 to 8	5	Factory automation
Banking	30	3 to 10	6	Loan processing, ATM
Command & Control	45	0.5 to 5	1	Command centers
Data Processing	35	2 to 14	8	DB-intense systems
Environment / Tools	75	5 to 12	8	CASE, compilers, etc.
Military All	125	0.2 to 3	< 1.0	See subcategories
Airborne	40	0.2 to 1.3	0.5	Embedded sensors
Ground	52	0.5 to 4	0.8	Combat center
Missile	15	0.3 to 1.5	0.5	GNC system
Space	18	0.2 to 0.8	0.4	Attitude control system
Scientific	35	0.9 to 5	2	Seismic processing
Telecom	50	3 to 12	6	Digital switches
Test	35	3 to 15	7	Test equipment, devices
Trainers/Simulators	25	2 to 11	6	Virtual reality simulator
Web Business	65	4 to 18	11	Client/server sites
Other	19	2 to 15	7	All others

Source: Harry M. Sneed, Stefan Jungmayr: Mehr Testwirtschatlichkeit durch Value-Driven-Testing, Informatik Spektrum 34, 2011

… and what are the requirements of our customers?

Parameters of one of my projects

- Project with 3 releases
- Each release had a separate go live
- We worked on each release with approximately 12-15 developers for 1,5 years
- We delivered each release 3-5 times to our customer
- Later releases influenced the functionality of earlier releases

Criteria's of acceptance tests:

- **Maximum number of high priority errors found by customer: 5**
- **Maximum number of middle priority errors found by customer: 15**

→ If criteria's are not hold: project delay at our expense

→ Go Live only without high priority defects and low number of medium defects

We were able to reach these conditions with testing and quality assurance.

AGENDA

General comments about testing

Objectives of testing:

- Identification of defects
- Determine quality of product
- Increase confidence in product
- Analysis to avoid effects of bugs

→ Increase quality of software
Of course only, if identified defects are fixed ;)

BUT:

- Testing is only a random, not an exhaustive check

→ We cannot ensure with testing that a software system is bug-free

→ No bigger software system is bug-free

In our projects testing is only one part of a comprehensive quality management

Constructive and analytical quality measures

.msg

Constructive quality measures

- Development techniques
  - structured development approach
  - tool-supported development
  - high-level programming languages
- Guidelines are based on
  - experience in numerous projects
  - planning actions in early phases

Restrictions, guidelines and standards during software development (lower variability) ensure that specific (typical) errors do not occur. Thus, we directly get a higher software quality.

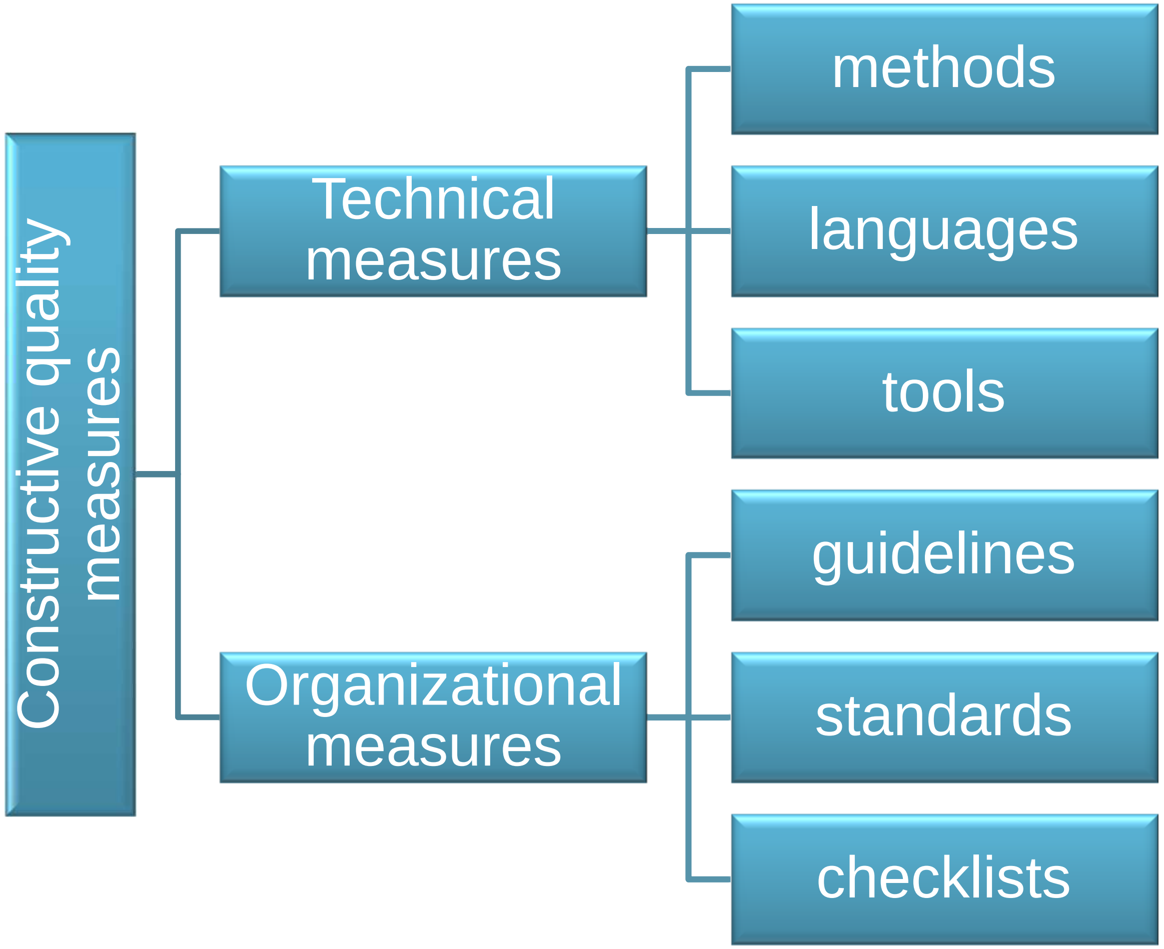
Analytical quality measures

- Diagnostic measures
- Do not directly ensure high quality
- Measurement of the quality of intermediate and final products

Execute data collection to compare the actual state of the software with the desired target state.

Identify quality of software after development.

Constructive quality measures



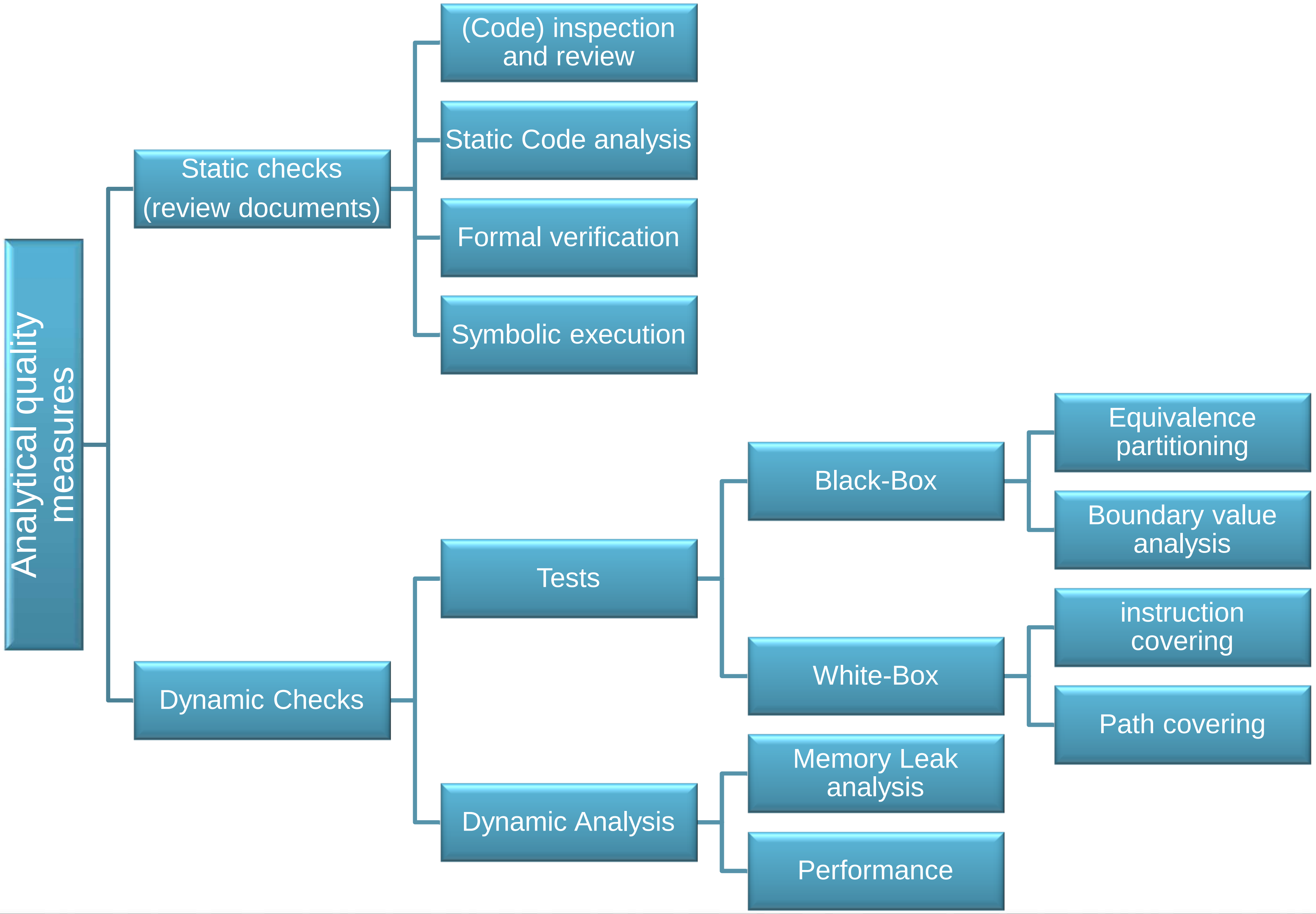
Example of methods:

- target: use a structured approach
- Define intermediate products
  - models
  - graph types
  - document types

Example of guidelines:

- target: define characteristics of product a priori
- Use different forms of guidelines:
  - checklists, stencils
  - review guidelines

Analytical quality measures



Dr. Marc Lohmann

Equivalence partitioning and Boundary value analysis

- Equivalence partitioning
  - divides the input domain of a software unit into different equivalence classes
  - test cases are designed to cover each partition at least once
  - technique tries to define test cases that cover classes of errors, thereby reducing the total number of test cases that must be developed
  - Reduces the time required for testing a software due to lesser number of test cases.

- Boundary value analysis
  - tests are designed to include representatives of boundary values
  - special form of equivalence partitioning
  - Use values that are located on and around the borders of the equivalence classes.

Example of Equivalence partitioning

- System under tests accepts numbers between 1 and 1000 as input.

- Using equivalence partitioning this leads to 3 equivalence classes:

  1. Input data class with valid value from 1 to 1000.

     Use a single value from range 1 to 1000 as a valid test case.
     For other values between 1 and 1000 the result is going to be same.

  2. Input data class with all values below lower limit.

     E.g. use any value below 1, as a invalid input data for a test case.

  3. Input data with any value greater than 1000.

- Equivalence partitioning uses fewest test cases to cover maximum requirements.

Example of Boundary value analysis

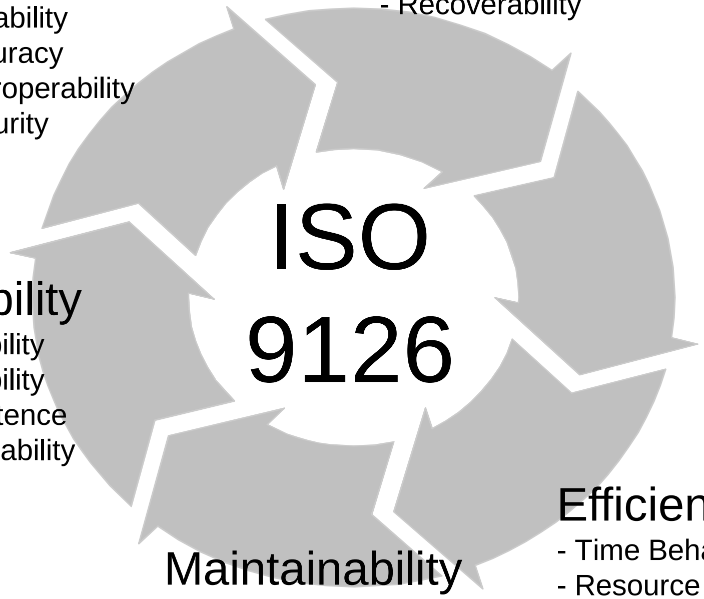
- System under tests accepts numbers between 1 and 1000 as input.

- Using boundary value analysis this leads to following tests:

  1. Test cases with test data exactly on the boundaries of the input domain i.e. values 1 and 1000 in our case.
  2. Test cases with test data values just below the edges of the input domain i.e. values 0 and 999.
  3. Test cases with test data values just above the edge of the input domain i.e. values 2 and 1001.

- Boundary value analysis is often used as a part of stress and negative testing.

Software quality encompasses more than the elimination of errors : Characteristics of software quality according to ISO 9126



Reliability
- Maturity
- Fault Tolerance
- Recoverability

Functionality
- Suitability
- Accuracy
- Interoperability
- Security

Usability
- Understandability
- Learnability
- Attractiveness

Portability
- Adaptability
- Installability
- Co-Existence
- Replaceability

Efficiency
- Time Behaviour
- Resource Utilization

Maintainability
- Analyzability
- Changeability
- Stability
- Testability

The quality requirements depend on the application scope of a software system.

Characteristics of software quality according to ISO 9126 (1/2)

Functionality

- Describes the needed functionality of a system
- Often described by pre- and post-conditions
- Tests check the described pre- and post-conditions
- Sub-characteristics:
  - Suitability
  - Accuracy
  - Interoperability
  - Security

Reliability

- Ability of a systems to ensure its performance within a defined period and defined constraints
- Sub-characteristics:
  - maturity
  - fault tolerance
  - recoverability

Usability

- Sub-characteristics:
  - understandability
  - learnability
  - attractiveness
- Important for the acceptance of the system
- Usability depends on the group of users
- Usability is checked in the context of non-functional tests

Characteristics of software quality according to ISO 9126 (1/2)

Efficiency

- Sub-characteristics:
  - response times
  - needed resources
- time and resources needed to execute a specific task
- Characteristic is measured with performance tests (non-functional tests)

Maintainability

- set of attributes that bear on the effort needed to make specified modifications.
- important criteria, because software systems are often used for a long period
- Sub-Characteristics:
  - analyzability
  - modifiability
  - stability

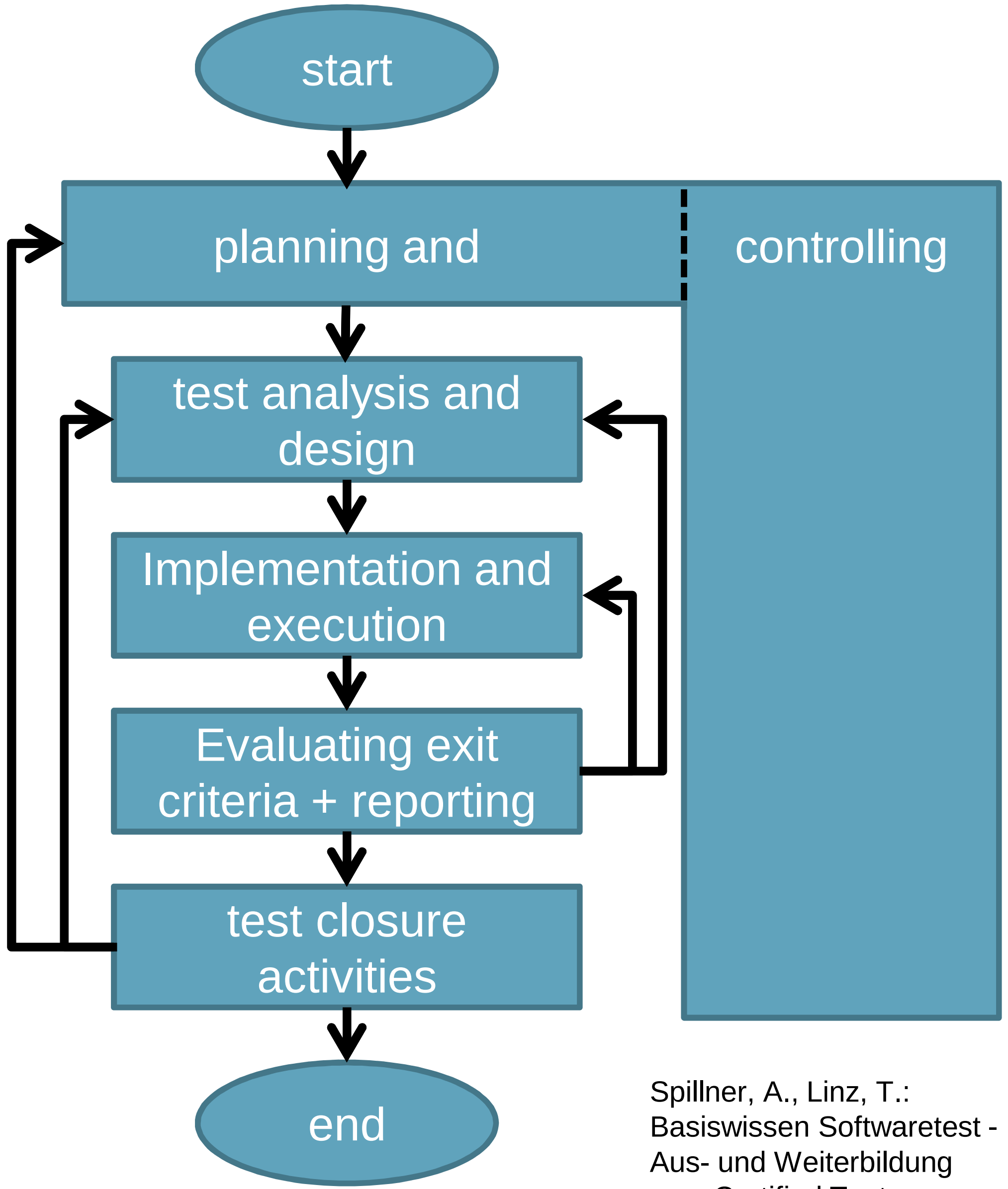
Portability

- set of attributes that bear on the ability of software to be transferred from one environment to another.
- Sub-charactristics:
  - Adaptability
  - Installability
  - Co-Existence
  - Replaceability
- Characteristics are often checked by a static analysis.

Basic test process (according to ISTQB)

- general software development processes consider often only the execution of tests

- basic test process:
  - generic roadmap
  - Process needs to be adjusted for specific projects.
    (E.g. basic test process does not describe roles and documents.)
  - Refines testing process within software development processes.
  - Activities in the process may overlap or take place concurrently.



Spillner, A., Linz, T.: Basiswissen Softwaretest - Aus- und Weiterbildung zum Certified Tester dpunkt, 4. Auflage

Basic test process – Additional information (1)

1) Planning and Control:

- **Test planning** has following major tasks:
  i. To determine the scope and risks and identify the objectives of testing.
  ii. To determine the test approach.
  iii. To implement the test policy and/or the **test strategy**. (Test strategy is an outline that describes the testing portion of the software development cycle. It is created to inform PM, testers and developers about some key issues of the testing process. This includes the testing objectives, method of testing, total time and resources required for the project and the testing environments.).
  iv. To determine the required test resources like people, test environments, PCs, etc.
  v. To schedule test analysis and design tasks, test implementation, execution and evaluation.
  vi. To determine the **Exit criteria** we need to set criteria such as **Coverage criteria.** (Coverage criteria are the percentage of statements in the software that must be executed during testing. This will help us track whether we are completing test activities correctly. They will show us which tasks and checks we must complete for a particular level of testing before we can say that testing is finished.)

- **Test control** has the following major tasks:
  i. To measure and analyze the results of reviews and testing.
  ii. To monitor and document progress, test coverage and exit criteria.
  iii. To provide information on testing.
  iv. To initiate corrective actions.
  v. To make decisions.

2) Analysis and Design:

- **Test analysis and Test Design** has the following major tasks:
  i. To review the **test basis**. (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what the system should do once built.)
  ii. To identify test conditions.
  iii. To design the tests.
  iv. To evaluate testability of the requirements and system.
  v. To design the test environment set-up and identify and required infrastructure and tools.

Basic test process – Additional information (2)

3) Implementation and Execution:

During test implementation and execution, we take the test conditions into **test cases** and procedures and other **testware** such as scripts for automation, the test environment and any other test infrastructure. (Test cases is a set of conditions under which a tester will determine whether an application is working correctly or not.)
(Testware is a term for all utilities that serve in combination for testing a software like scripts, the test environment and any other test infrastructure for later reuse.)

- **Test implementation** has the following major task:
  i. To develop and prioritize our test cases by using techniques and create **test data** for those tests. (In order to test a software application you need to enter some data for testing most of the features. Any such specifically identified data which is used in tests is known as test data.)
  We also write some instructions for carrying out the tests which is known as **test procedures**.
  We may also need to automate some tests using **test harness** and automated tests scripts. (A test harness is a collection of software and test data for testing a program unit by running it under different conditions and monitoring its behavior and outputs.)
  ii. To create test suites from the test cases for efficient test execution.
  (Test suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviours. A test suite often contains detailed instructions and information for each collection of test cases on the system configuration to be used during testing. Test suites are used to group similar test cases together.)
  iii. To implement and verify the environment.

- **Test execution** has the following major task:
  i. To execute test suites and individual test cases following the test procedures.
  ii. To re-execute the tests that previously failed in order to confirm a fix. This is known as **confirmation testing or re-testing.**
  iii. To log the outcome of the test execution and record the identities and versions of the software under tests. The **test log** is used for the audit trial. (A test log is nothing but, what are the test cases that we executed, in what order we executed, who executed that test cases and what is the status of the test case (pass/fail). These descriptions are documented and called as test log.).
  iv. To Compare actual results with expected results.
  v. Where there are differences between actual and expected results, it report discrepancies as Incidents.

Basic test process – Additional information (3)

4) Evaluating Exit criteria and Reporting:

Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the “enough testing”. These criteria vary from project to project and are known as **exit criteria**. Exit criteria come into picture, when:

- Maximum test cases are executed with certain pass percentage.
- Bug rate falls below certain level.
- When achieved the deadlines.

- **Evaluating exit criteria** has the following major tasks:

  i. To check the test logs against the exit criteria specified in test planning.
  ii. To assess if more test are needed or if the exit criteria specified should be changed.
  iii. To write a test summary report for stakeholders.

5) Test Closure activities:

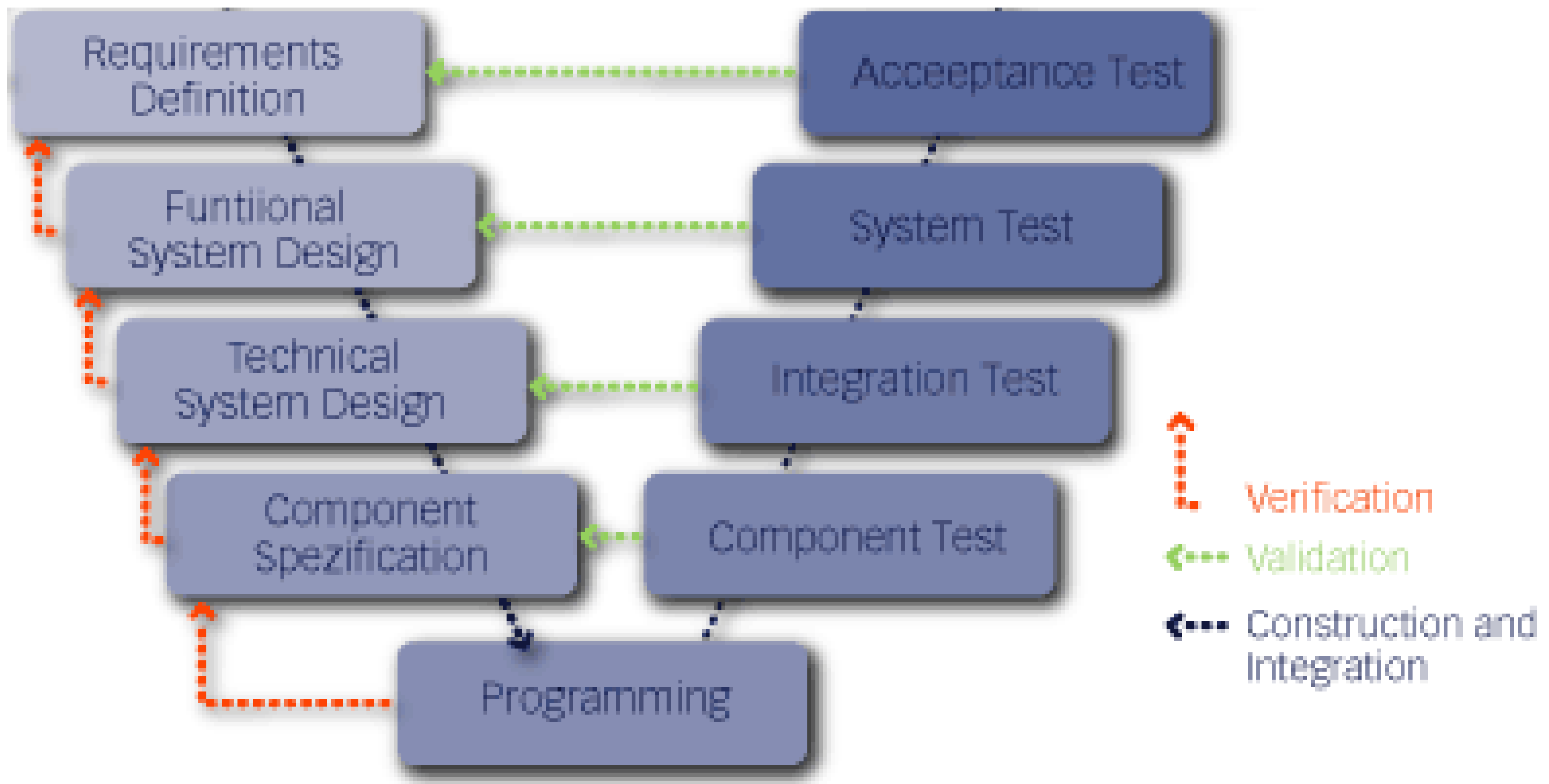
Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

- When all the information has been gathered which are needed for the testing.
- When a project is cancelled.
- When some target is achieved.
- When a maintenance release or update is done.

- **Test closure activities** have the following major tasks:

  i. To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.
  ii. To finalize and archive testware such as scripts, test environments, etc. for later reuse.
  iii. To handover the testware to the maintenance organization. They will give support to the software.
  iv. iv To evaluate how the testing went and learn lessons for future releases and projects.

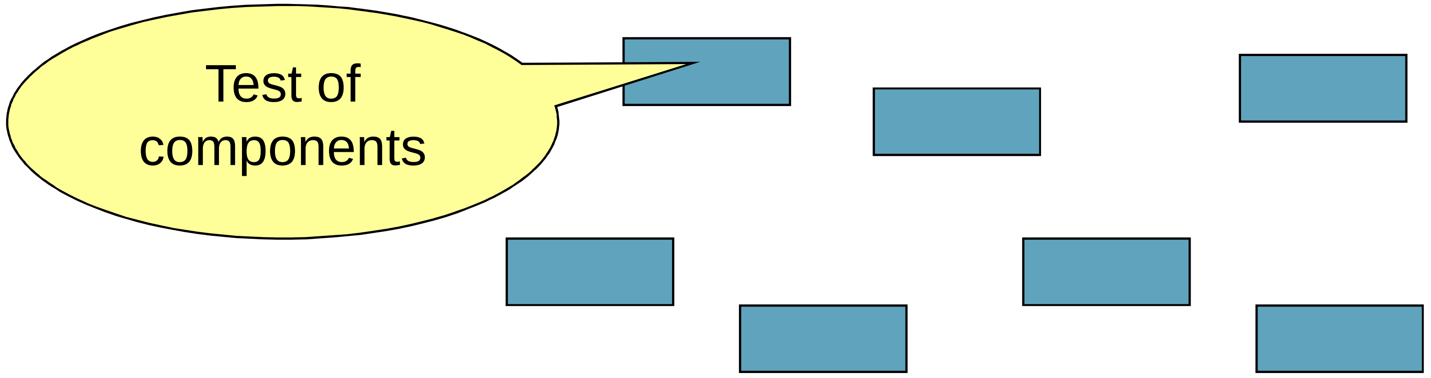
Testing within the software life cycle – Example V-Modell



- **Basic idea of V-Modell**
  Development and test are corresponding actions
- We can transfer this basic idea and the resulting principles to other development processes.

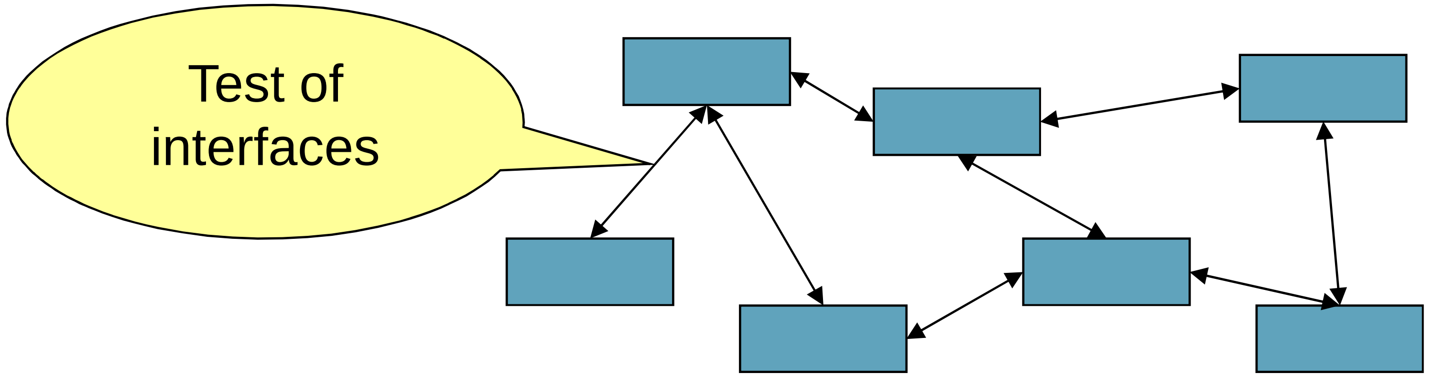
Test level: Component Test

- Component test verify the correct functionality of the individual components of a system.
- Focus is the isolated testing of individual components.
- Component test is usually performed by the developer prior to integration.



Test level: Integration test

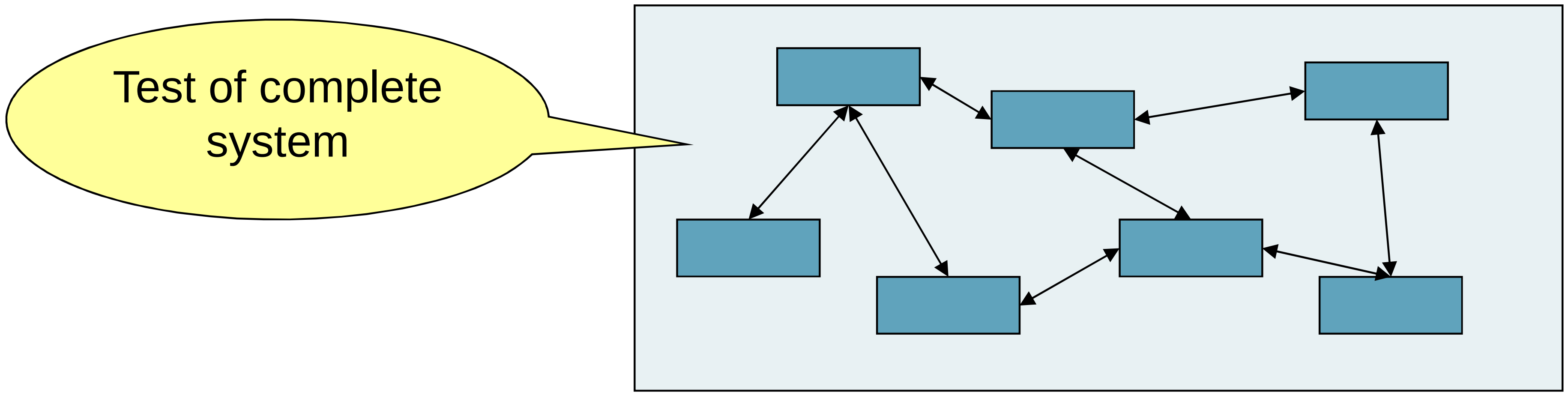
- Integration test is to check the flawless interaction of the system components.
- The individually tested components are gradually integrated and the interaction is tested.
- Focus is the examination of the interaction of the components. Therefore, the interfaces need to be tested in different combinations.



Test level: System test

- System test is the final test in a realistic environment (without customer).
  - Functional testing against the specification
  - Mass testing, performance testing, load testing
  - Usability tests

- Focus of the examination is the behavior of the overall system



Test level: Acceptance test

- Acceptance test is performed by the customer in the operating environment of the customer with real data
  - Generation and installation of the system
  - Test cases for real use cases
  - Random test cases
  - Audit Documentation

- Acceptance testing is the basis for the acceptance by the customer.

Overview of different test levels

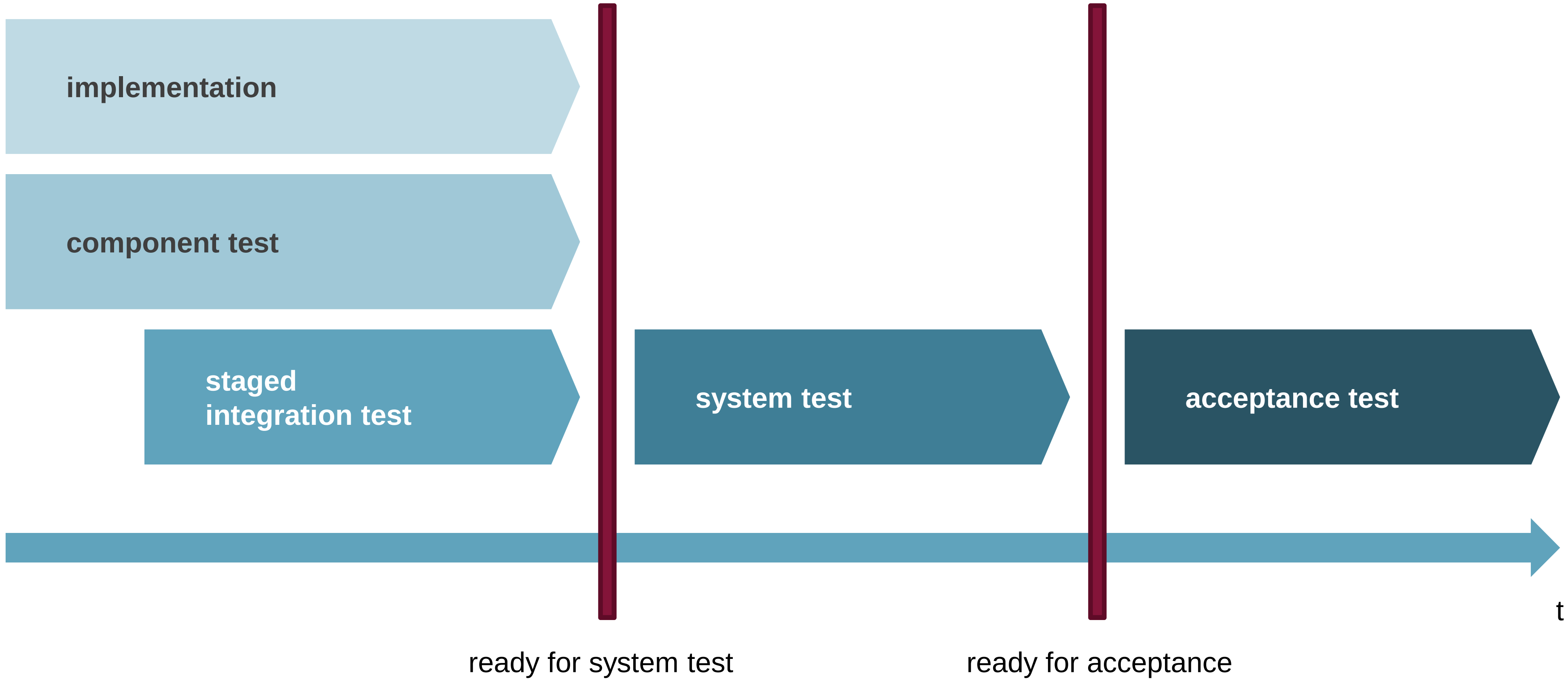
	Component test	Integration test	System test	Acceptance test
test basis	• Specification of components. • Software design	• System design • Use cases • Workflow descriptions	• documents on system level: specifications, requirements ...	• Documents that describe the system from a user perspective
Tester	• developer	• tester	• tester	• customer
Test strategy	• White-Box	• Black-Box	• Black-Box	• Black-Box
Test goals	e.g. • functionality • robustness • efficiency	e.g. • errors on interface level • Interactions between components	e.g.. • test if system fulfills functional / non-functional requirements	e.g. • objectives of contract • acceptance of users and system operators

Component tests and integration tests are normally not executed sequentially.

System and acceptance tests are normally executed in the last weeks of a project.

An early test execution is important for a successful project

- Early tests reduce project risks.
- Early tests show the quality of the product during development.



Basic principles of software testing (according to ISTQB)

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

Spillner, A., Linz, T.:
Basiswissen Softwaretest -
Aus- und Weiterbildung
zum Certified Tester
dpunkt, 4. Auflage

Basic principles of software testing - Additional information

1. Testing shows presence of defects
   Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.
2. Exhaustive testing is impossible
   Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, we use risk and priorities to focus testing efforts.
3. Early testing
   Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives.
4. Defect clustering
   A small number of modules contain most of the defects discovered during pre-release testing, or show the most operational failures.
5. Pesticide paradox
   If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new bugs. To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.
6. Testing is context dependent
   Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.
7. Absence-of-errors fallacy
   Finding and fixing defects does not help if the system built is unusable and does not fulfill the users’ needs and expectations.

AGENDA

Test tool classification (1/2)

Tool Support for management of testing and tests

- storage, administration and monitoring of test cases
- administration of incidents
- administration of requirements
- configuration management

Tool support for test case specification

- administration of pre-/post-conditions of test cases
- test data preparation tool
  (e.g. based on database or specification)

Tool support for static tests

- support of review processes
- static analysis (e.g. cyclomatic complexity)
- model checker

Test tool classification (2/2)

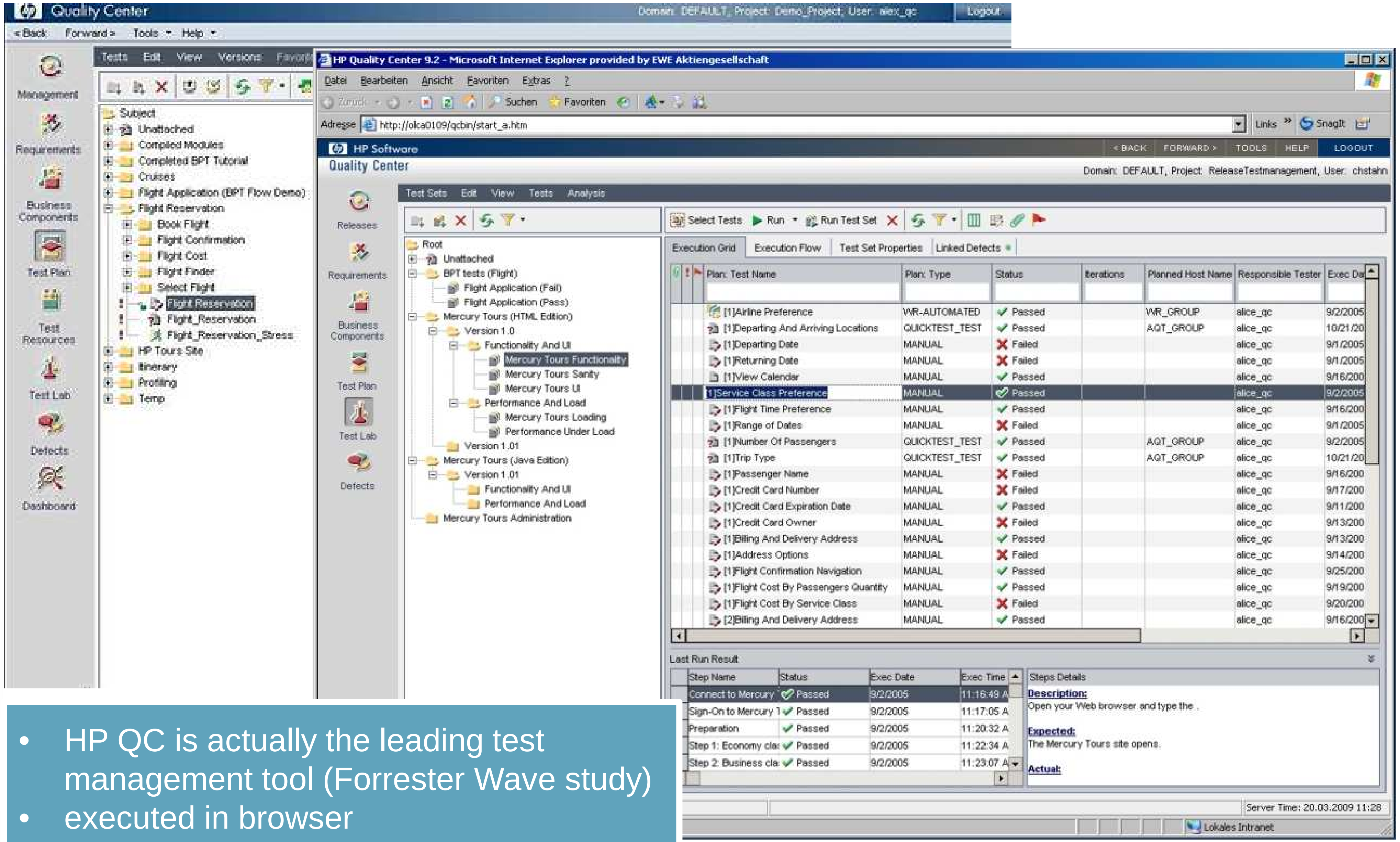
Tool support for execution and logging

- test harness/unit framework tools
- test comparators
- coverage management tools
- security testing tools
- simulation,
- capture-and-replay-tools

Tool support for non-functional tests

- performance testing, load testing, stress testing tools
- monitoring tools

Example test management: HP Quality Center



Plan: Test Name	Plan: Type	Status	Iterations	Planned Host Name	Responsible Tester	Exec Da
[1]Airline Preference	WR-AUTOMATED	✓ Passed		WR_GROUP	alice_qc	9/2/2005
[1]Departing And Arriving Locations	QUICKTEST_TEST	✓ Passed		AQT_GROUP	alice_qc	10/21/20
[1]Departing Date	MANUAL	✗ Failed			alice_qc	9/1/2005
[1]Returning Date	MANUAL	✗ Failed			alice_qc	9/1/2005
[1]View Calendar	MANUAL	✓ Passed			alice_qc	9/16/200
[1]Service Class Preference	MANUAL	✓ Passed			alice_qc	9/2/2005
[1]Flight Time Preference	MANUAL	✓ Passed			alice_qc	9/16/200
[1]Range of Dates	MANUAL	✗ Failed			alice_qc	9/1/2005
[1]Number Of Passengers	QUICKTEST_TEST	✓ Passed		AQT_GROUP	alice_qc	9/2/2005
[1]Trip Type	QUICKTEST_TEST	✓ Passed		AQT_GROUP	alice_qc	10/21/20
[1]Passenger Name	MANUAL	✗ Failed			alice_qc	9/16/200
[1]Credit Card Number	MANUAL	✗ Failed			alice_qc	9/17/200
[1]Credit Card Expiration Date	MANUAL	✓ Passed			alice_qc	9/11/200
[1]Credit Card Owner	MANUAL	✗ Failed			alice_qc	9/13/200
[1]Billing And Delivery Address	MANUAL	✓ Passed			alice_qc	9/13/200
[1]Address Options	MANUAL	✗ Failed			alice_qc	9/14/200
[1]Flight Confirmation Navigation	MANUAL	✓ Passed			alice_qc	9/25/200
[1]Flight Cost By Passengers Quantity	MANUAL	✓ Passed			alice_qc	9/19/200
[1]Flight Cost By Service Class	MANUAL	✗ Failed			alice_qc	9/20/200
[2]Billing And Delivery Address	MANUAL	✓ Passed			alice_qc	9/16/200

Last Run Result

Step Name	Status	Exec Date	Exec Time
Connect to Mercury	✓ Passed	9/2/2005	11:16:49 A
Sign-On to Mercury T	✓ Passed	9/2/2005	11:17:05 A
Preparation	✓ Passed	9/2/2005	11:20:32 A
Step 1: Economy cla:	✓ Passed	9/2/2005	11:22:34 A
Step 2: Business cla:	✓ Passed	9/2/2005	11:23:07 A

Steps Details

Description:
Open your Web browser and type the .

Expected:
The Mercury Tours site opens.

Actual:

Server Time: 20.03.2009 11:28

- HP QC is actually the leading test management tool (Forrester Wave study)
- executed in browser
- offers automation of tests with HP Quick Test

Static analysis by tools Examples

Checkstyle :

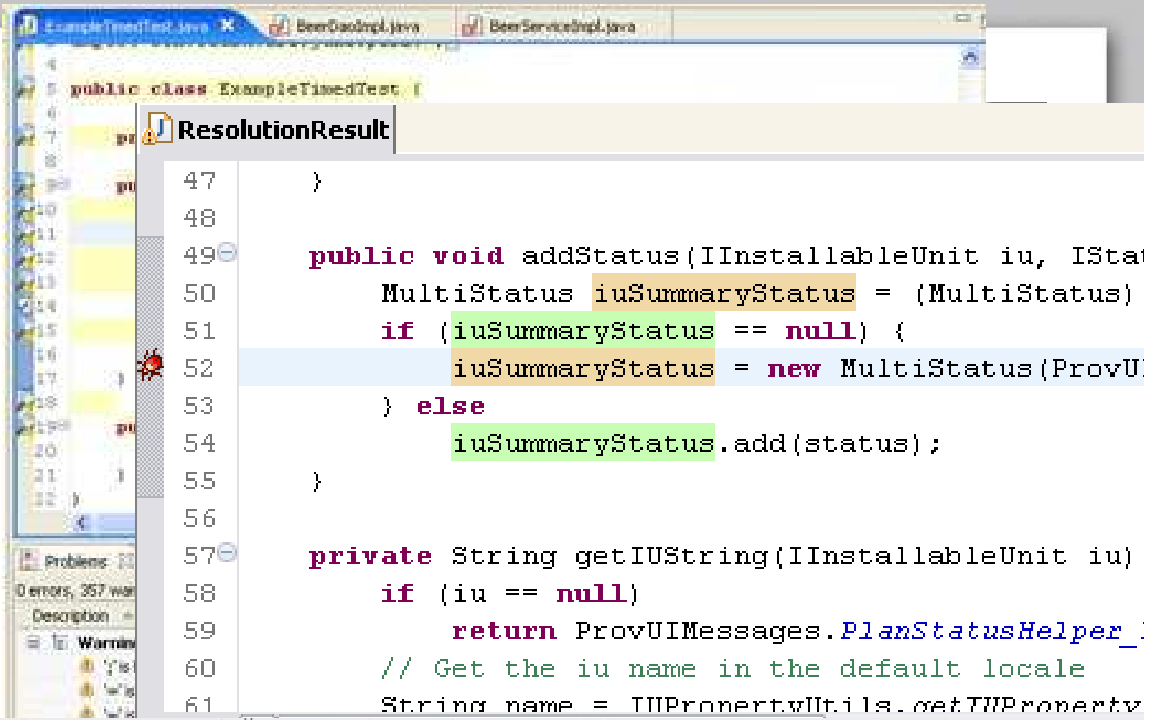
- check of coding conventions in Java
- project configuration necessary
- integration in Eclipse possible

FindBugs:

- Finds typical error patterns in Java programs
- Error patterns often indicate real defects.
- In larger projects you need to filter the warnings (number of warnings is very high).

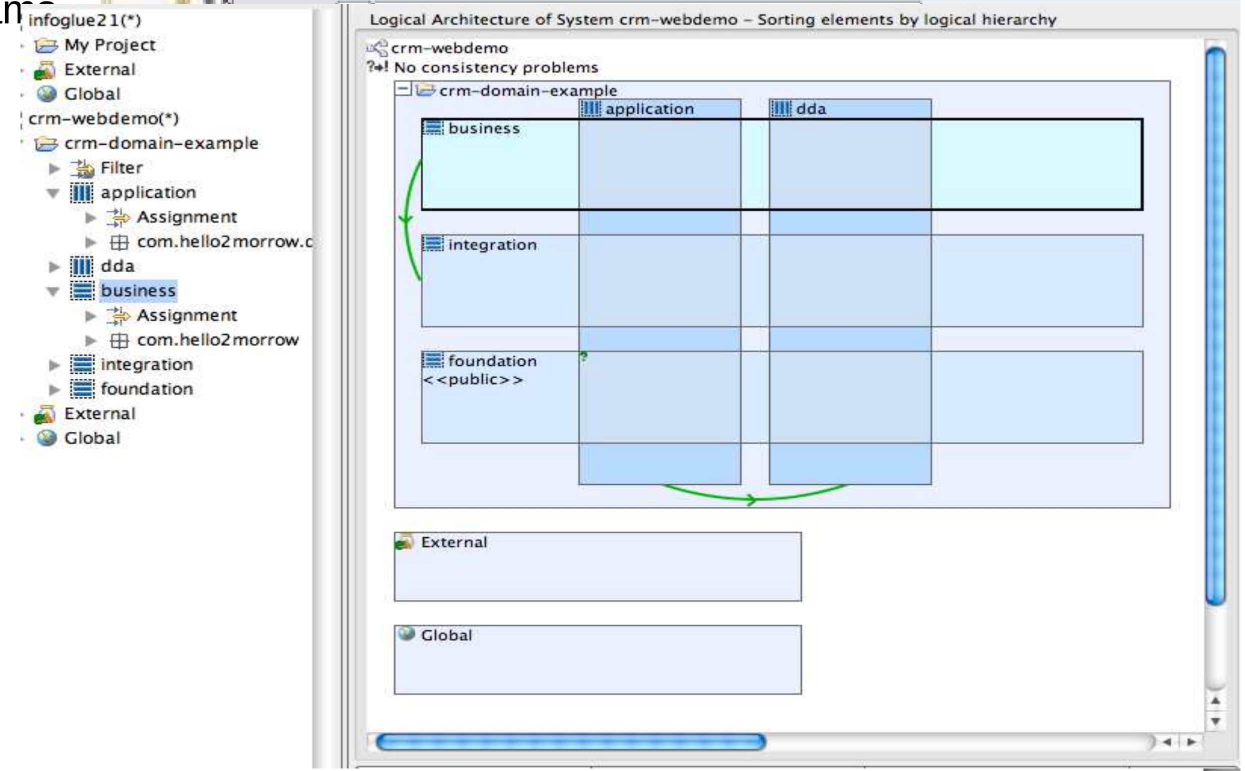
Sonograph for Java:

- Sonograph needs as input an architectural model of the project
- Identification of architectural problems and wrong dependencies according to architectural model
- Identification of cyclic dependencies



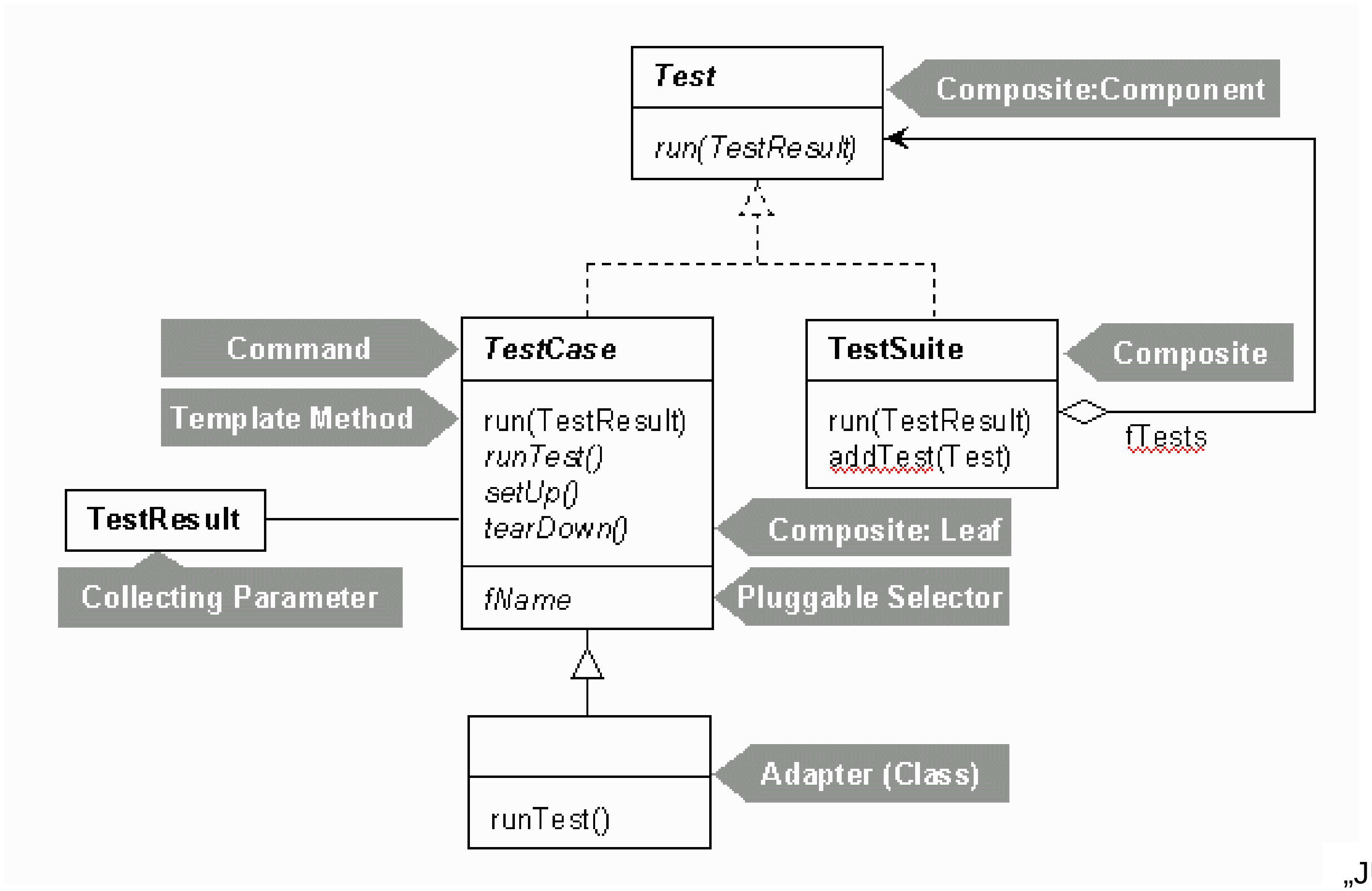
```
public class ExampleTimedTest {
```

```
    }

    public void addStatus(IInstallableUnit iu, IStat
        MultiStatus iuSummaryStatus = (MultiStatus)
        if (iuSummaryStatus == null) {
            iuSummaryStatus = new MultiStatus(ProvUI
        } else
            iuSummaryStatus.add(status);
    }

    private String getIUString(IInstallableUnit iu)
        if (iu == null)
            return ProvUIMessages.PlanStatusHelper_
        // Get the iu name in the default locale
        String name = IUPropertyUtils.getIUProperty
```



Tool support for automatic test execution
Example JUnit

- Framework for testing Java programs
- Especially used for the automation of unit tests (test of methods and classes)



„JUnit A Cook's Tour“

A simple JUnit test case

tests inherit from Junit class TestCase

```
public class StringTest extends TestCase {

          protected void setUp(…)

          protected void tearDown(…)

          public void testSimpleAdd() {

                    String s1 = new String(“abcd”);
                    String s2 = new String(“abcd”);

                    assertTrue(“Strings not equal”, s1.equals(s2));
          }
}
```

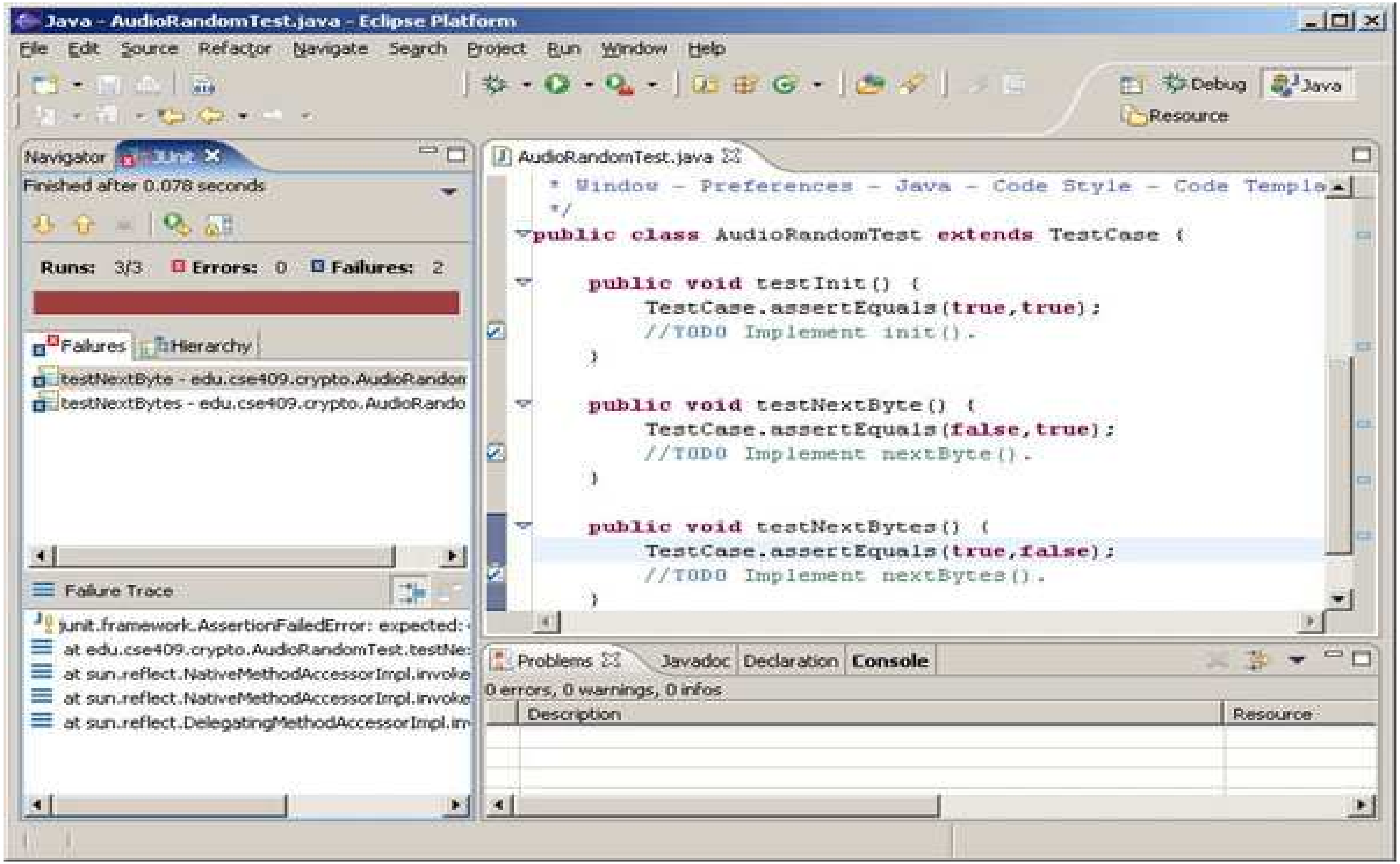
initialization of test object

resetting of test object after test execution

test object is checked with assert-methods of JUnit

- Projects generally develop a test framework, which specializes the JUnit class TestCase.
- The test framework initializes all components, the database etc. (uniform for all tests)
  - The tests itself only contain a few specific initializations for the concrete test.

JUnit: Eclipse integration



Finished after 0.078 seconds

Runs: 3/3 Errors: 0 Failures: 2

```
testNextByte - edu.cse409.crypto.AudioRandom
testNextBytes - edu.cse409.crypto.AudioRando
```

```
junit.framework.AssertionFailedError: expected:
at edu.cse409.crypto.AudioRandomTest.testNe
at sun.reflect.NativeMethodAccessorImpl.invoke
at sun.reflect.NativeMethodAccessorImpl.invoke
at sun.reflect.DelegatingMethodAccessorImpl.in
```

```
 * Window - Preferences - Java - Code Style - Code Templa
 */
public class AudioRandomTest extends TestCase {

    public void testInit() {
        TestCase.assertEquals(true,true);
        //TODO Implement init().
    }

    public void testNextByte() {
        TestCase.assertEquals(false,true);
        //TODO Implement nextByte().
    }

    public void testNextBytes() {
        TestCase.assertEquals(true,false);
        //TODO Implement nextBytes().
    }
```

0 errors, 0 warnings, 0 infos

Benefits, Disadvantages of JUnit

Benefits

- Simple creation of test drivers.
- Easy automation of tests.
- Integration in IDEs
- Periodical execution of tests is possible (nightly builds)
- Projects specific adaptions are possible
- Refactoring of code affect test drivers

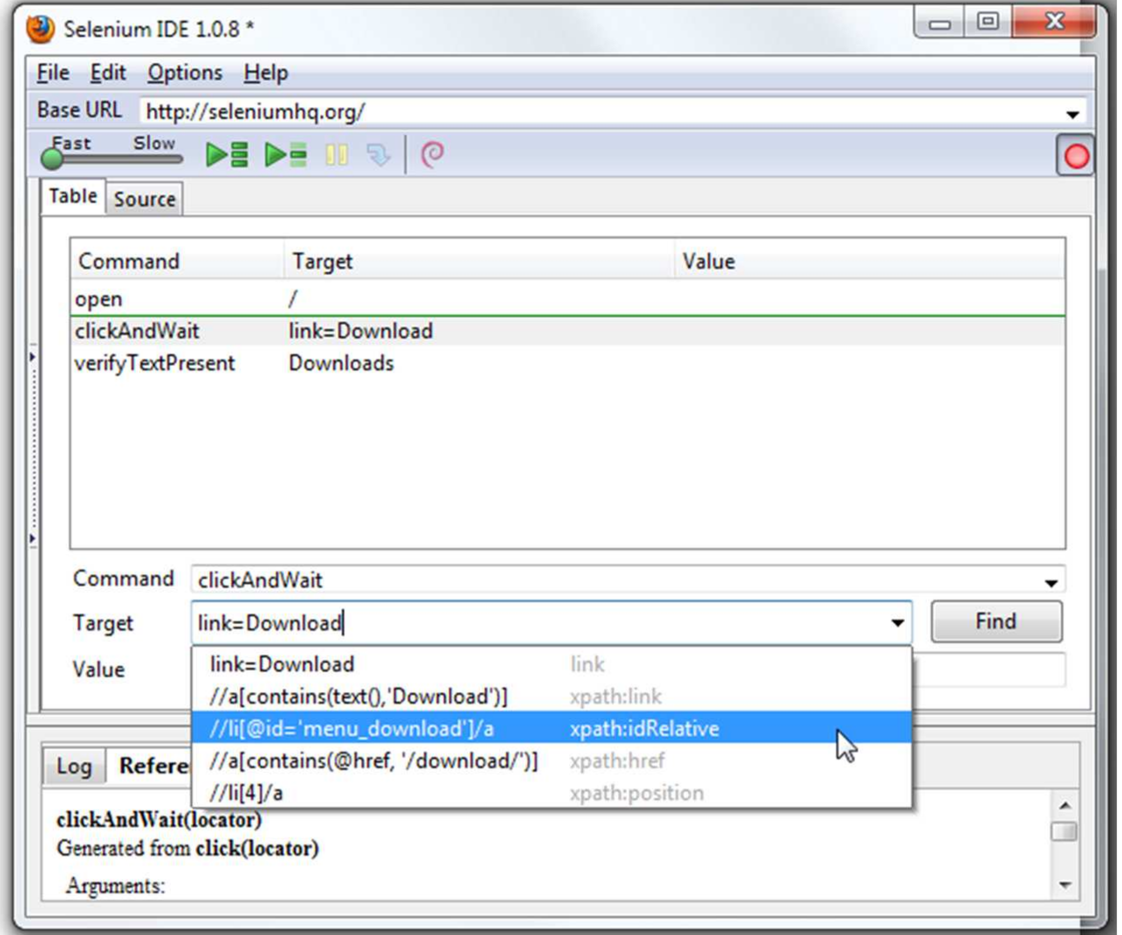
Disadvantages

- No direct separation of test code and test data
- Dependent components must be initialized before test execution
- No GUI-Tests possible

Tool support for automatic test execution
Example Selenium

- Test framework for web applications
- Capture manual executed operating steps.
- Operating steps are stored as test scripts.
- You can repeat the stored test scripts.
- You can store test scripts as HTML-tables.
- Selenium is based on HTML und JavaScript.
- Selenium-IDE available as Firefox-Addon
- http://seleniumhq.org/

Capture-and-Replay-Tool



Base URL http://seleniumhq.org/

Command	Target	Value
open	/	
clickAndWait	link=Download	
verifyTextPresent	Downloads	

Command clickAndWait
Target link=Download
Value

link=Download	link
//a[contains(text(),'Download')]	xpath:link
//li[@id='menu_download']/a	xpath:idRelative
//a[contains(@href, '/download/')]	xpath:href
//li[4]/a	xpath:position

clickAndWait(locator)
Generated from click(locator)
Arguments:

Benefits, Disadvantages of Selenium

Benefits

- Simple creation of repeatable GUI-test scripts
- Suitable for regression tests
- Contains a reasonable reporting of executed test cases.
- Generation of test scripts possible.

Disadvantages

- No direct separation of test code and test data.
- Very sensitive according to GUI-changes.
- Test cases are not influenced by refactoring.
- Manual post-processing of scripts is necessary.
- For a long term use of Selenium you need a good test (framework) architecture (modularization of test cases, …)

Opportunities and risks of automated testing

Opportunities

- Time pressure and risks are reduced in later project phases
- It is possible to test the application quickly after a change, especially important after the go-live
- Protection of side effects
- Cost saving
- Less stupid test executions

Risks

- Team is too inexperienced regarding testing
- wrong tools
- Different tools are not compatible
- Test case not maintainable or it is not possible to automate test cases
- High costs for maintenance of test cases

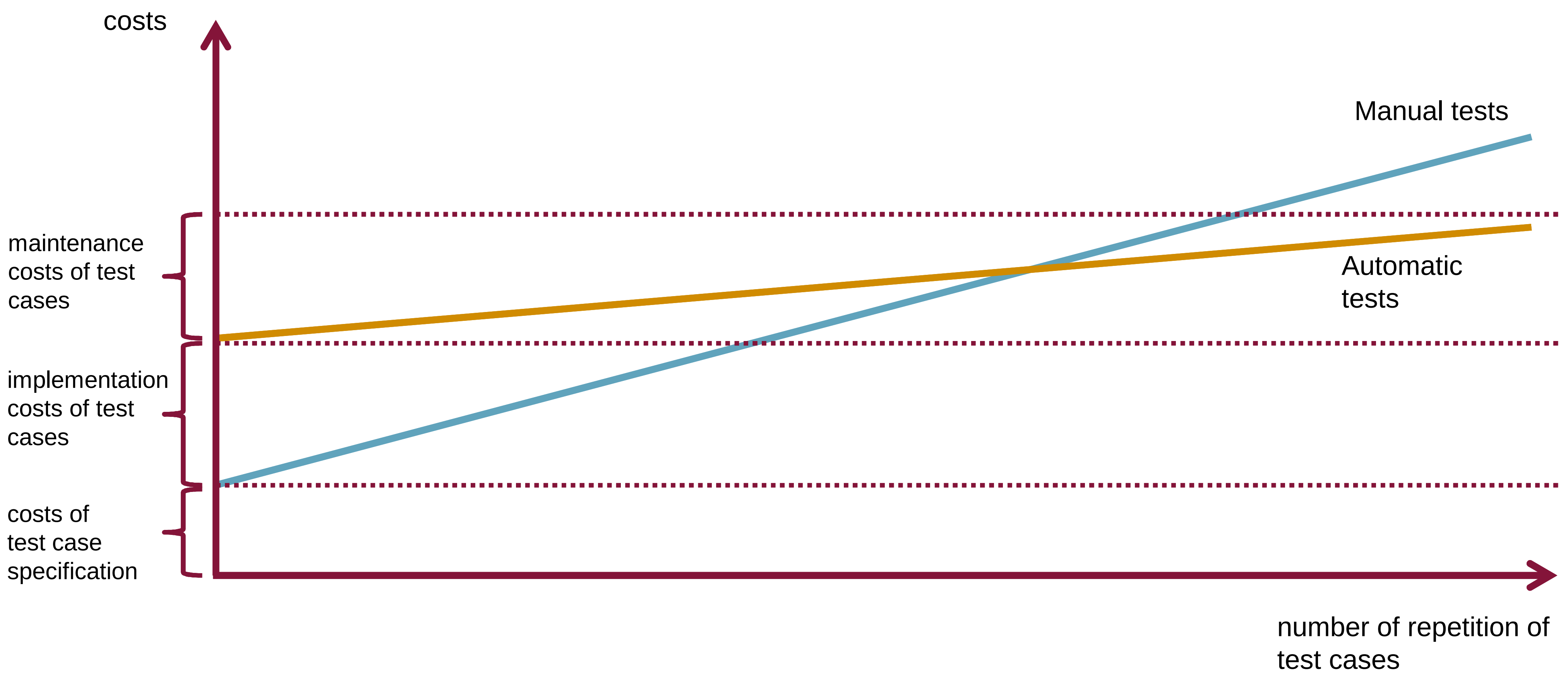
Automating chaos gives faster chaos*

- Do not start a project with the introduction of tool support for automatic test execution.
- You can only increase the productivity with tools for automatic test execution, if you have a defined and use a systematic test process.

- Order to introduce tool support for testing
  1. defect management
  2. configuration management
  3. test planning
  4. test execution
  5. test specification

*Fewster, M., Graham, D.:
Software Automation,
Effective use of test execution tools

Payback of test automation depends on number of repetition of test cases



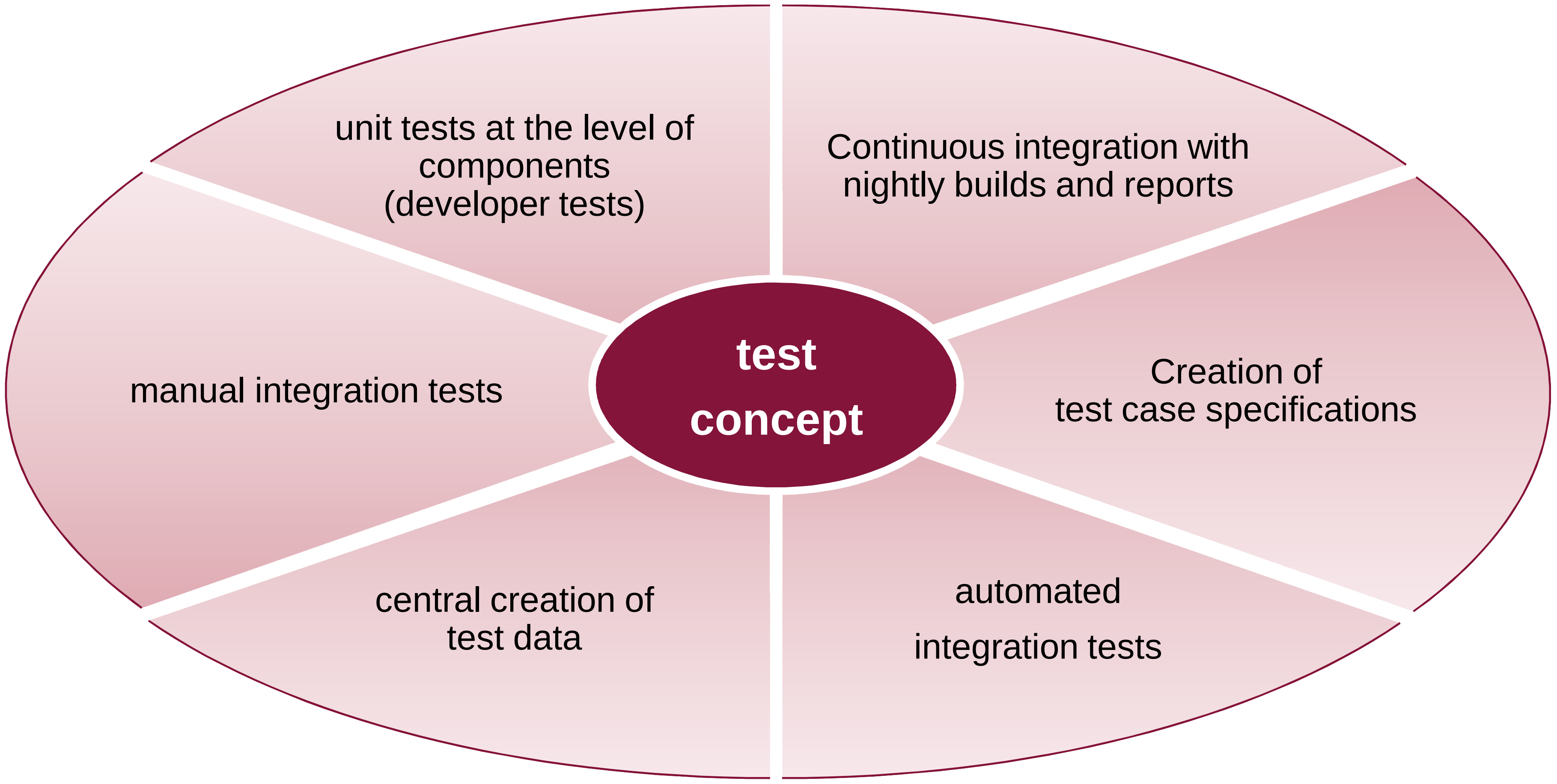
- Test automation pays of after round about **5** repetitions of test cases
- Large projects take several years with several releases

→ Test automation is normally meaningful in large projects.

AGENDA

1. Why should we test?
2. Fundamentals of software testing
3. Tool support for testing
4. **Automatic testing – experience report**

Elements of our test concept


test concept
unit tests at the level of components (developer tests)
Continuous integration with nightly builds and reports
Creation of test case specifications
automated integration tests
central creation of test data
manual integration tests

Developer tests and Continuous Integration

- Developer tests
  - Manual developer tests for dialogs
  - JUnit tests
  - Developer was only allowed to check-in code changes, if all automated tests of the corresponding component have been successful.
  - Concrete guidelines about kind and coverage of tests
  - Refactoring of Junit tests has been working well

- Continuous Integration
  - Continuous Integration with Hudson
  - After a developer checked-in new code, Hudson has built the application and executed all Junit tests
  - Detailed reports and automatic delivery of reports to responsible persons

- Nightly-Build
  - Nighly-Build was executed daily
  - Build and deployment of the application for manual tests on the next day.
  - Execution of automated developer and integration tests during nightly-build
  - History of reports
    (Sometimes we needed to identify at which time an error occurred the first time)

Testcase specification

- Concrete guidelines how to extract test goals from the system specifications
  - Guidelines ensured test coverage
  - Guidelines ensured that all topics are tested similarly (with similar test coverage)
  - Examples:
    - Test pre-/post-conditions
    - Do not test simple validations (number of allowed characters, special characters, …)
  - We aligned the guidelines to the structure of the system specification.
- Test strategy on the level of test case specification
  - Simple behavior of dialogs is tested *manually*, test are not repeated
    - mandatory fields
    - correct generation of documents
    - …
  - Use case descriptions are tested *automatically*
    - Pre-/Post-conditions
    - External interfaces
    - ….
- Decision about used tool (Proven! or JUnit)
- Review of test case specification
  - Early quality assurance
  - Check of completeness
- We shared our test case specifications with our customer
  - We identified with the help of our test case specifications problems or errors on the level of our specifications

Guidelines for implementing automatic tests

- Problems in our project
  - Proven! test cases not robust with respect to application changes
  - Proven! Test cases not robust with respect to test data changes
  - → High costs to adjust the test cases often

- Identification of anti-pattern:
  - SQL-Antipattern
    Testers used SQL statements in test cases to adjust test data
    → If we adjusted the data model, the SQL statements did not work correctly.
  - Wrong assumptions regarding long-term constancy of test data
    Testers assumed result sets (e.g. after a search) as “static”
    → After a change of the test data the result sets changed. We could only ensure constant test data for defined central aspects.
  - Programming in test cases
    Tester tried to implement smaller business checks directly in the test case
    → usage of central business connectors, that can be called from all test cases

Experiences with the automation of dialog tests

- Automation makes sense in a project over several years
- A complete automation of all tests is not economical
  - Effort of automation is high
  - Maintenance support of automated tests can be high
  - Depending on the project, it has to checked which test cases have to be executed how often
- Modularization of tests
  - But: Modularization worsens the readability of tests
- Frameworks for the automation of dialog tests allow an easy implementation of test cases
  - Guidelines for the implementation are absolutely necessary. Otherwise, maintenance is not possible.
- Automated test cases are often implemented in a very naive way
  - Test cases are very sensible regarding application and test data changes.
- Normally, frameworks for the automation of dialog tests do not support refactoring of test cases
- Continuous monitoring and maintenance of automated test cases is necessary
  - Otherwise, test cases and application are not on the same level.

Vielen Dank für Ihre Aufmerksamkeit

msg systems ag

Mergenthalerallee 73-75
65760 Eschborn
www.msg-systems.com

.consulting .solutions .partnership