Quality Assurance Activities to Support Product Improvement

Dietmar Winkler

Vienna University of Technology
Institute of Software Technology and Interactive Systems

dietmar.winkler@qse.ifs.tuwien.ac.at
http://qse.ifs.tuwien.ac.at

Motivation

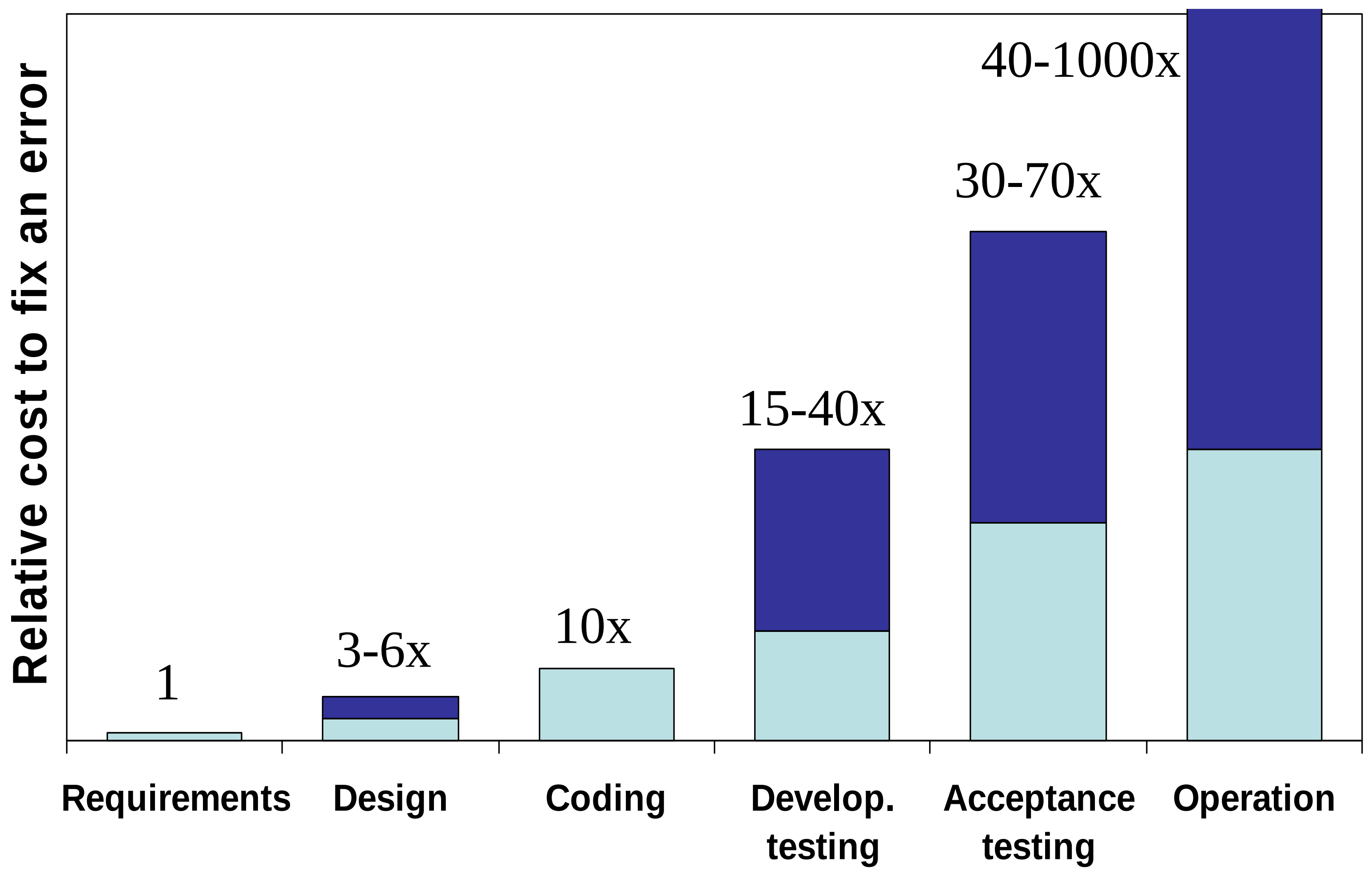
- A major goal in Software Engineering (SE) is the construction of high-quality and testable software products.
- Important key approaches:
  - Software processes (e.g., Life-Cycle Model, V-Modell XT, Scrum) define the sequence of steps, *when* to build *which* products.
  - Constructive methods, e.g., Model-Driven Development, Test-Driven Development, and pair programming support creating new (pieces of) software products (*how to build*).
  - Analytical methods, e.g., inspection and testing help assessing product and process quality (*how to evaluate*).
- Quality Assurance (QA) supports engineers in evaluating products in every stage of software development and must be an integrated part of the development process.

Topics

→ Reviews and Inspections

→ Basics in Software Testing

Rework Effort During Project Progress



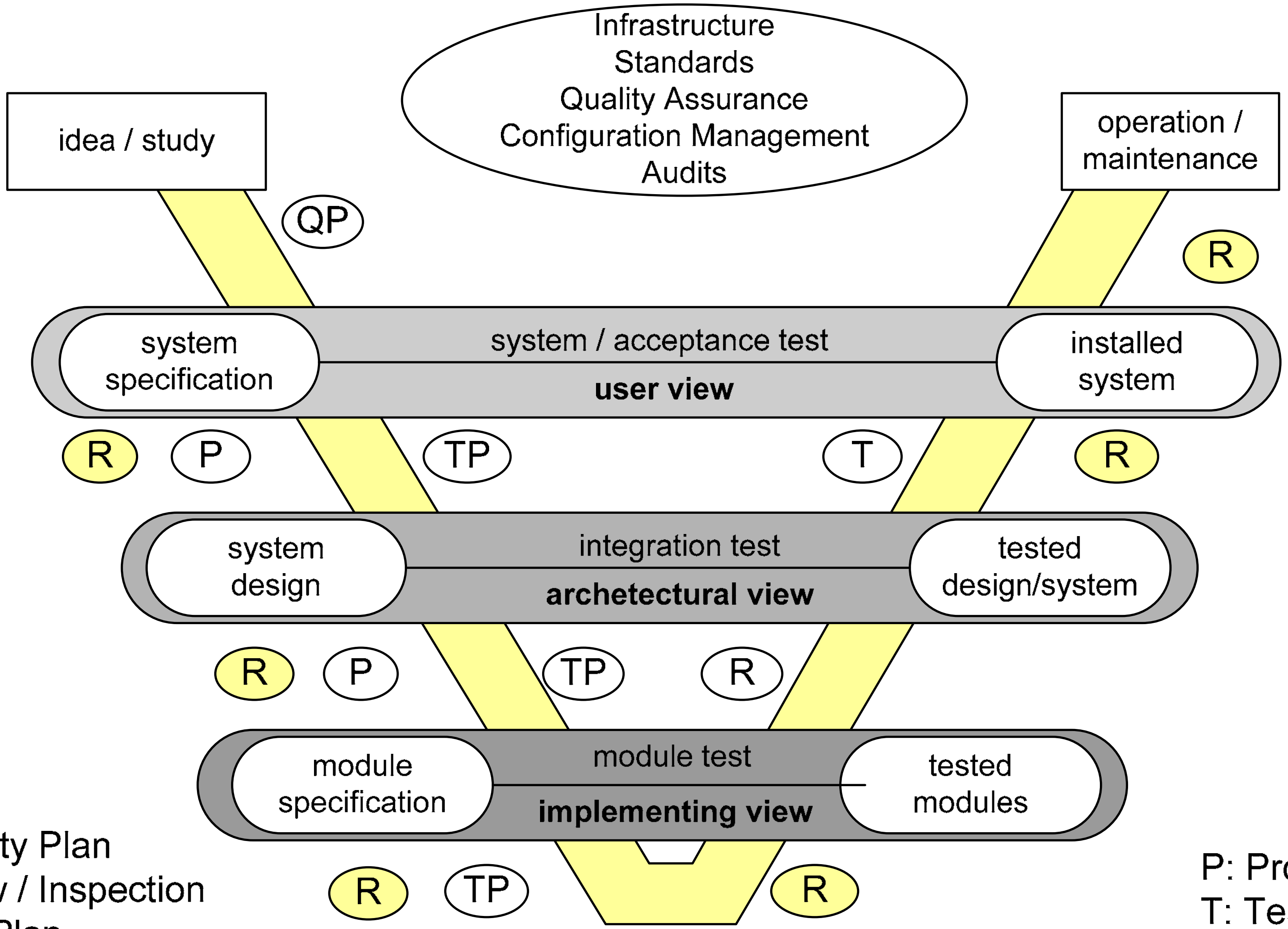
The cost of fixing errors escalates as we move the project towards field use.
From an analysis of sixty-three projects cited in Boehm Barry, „Software Engineering Economics“, Prentice-Hall, 1981

Improve Products Early

- Software defects have a heavy impact on project quality, project duration and project budget.
- Rework effort increases the later a defect is detected within the project course. (in the worst case defects can lead to project termination!)
- Main goal is
  - not to make any defects (seems to be very challenging)
  - or to identify defects as early as possible (and remove them at the point of their appearance)

- Analytical QA approaches must be an integral part of the software development process.
  - Reviews and inspections aim at improving product and process quality in all phases of software development with special focus on early software development phases (no source code required)
  - Testing is applicable to code documents (source code) and is typically located in the implementation phase or later in the development life-cycle.

Software Processes and Quality Assurance

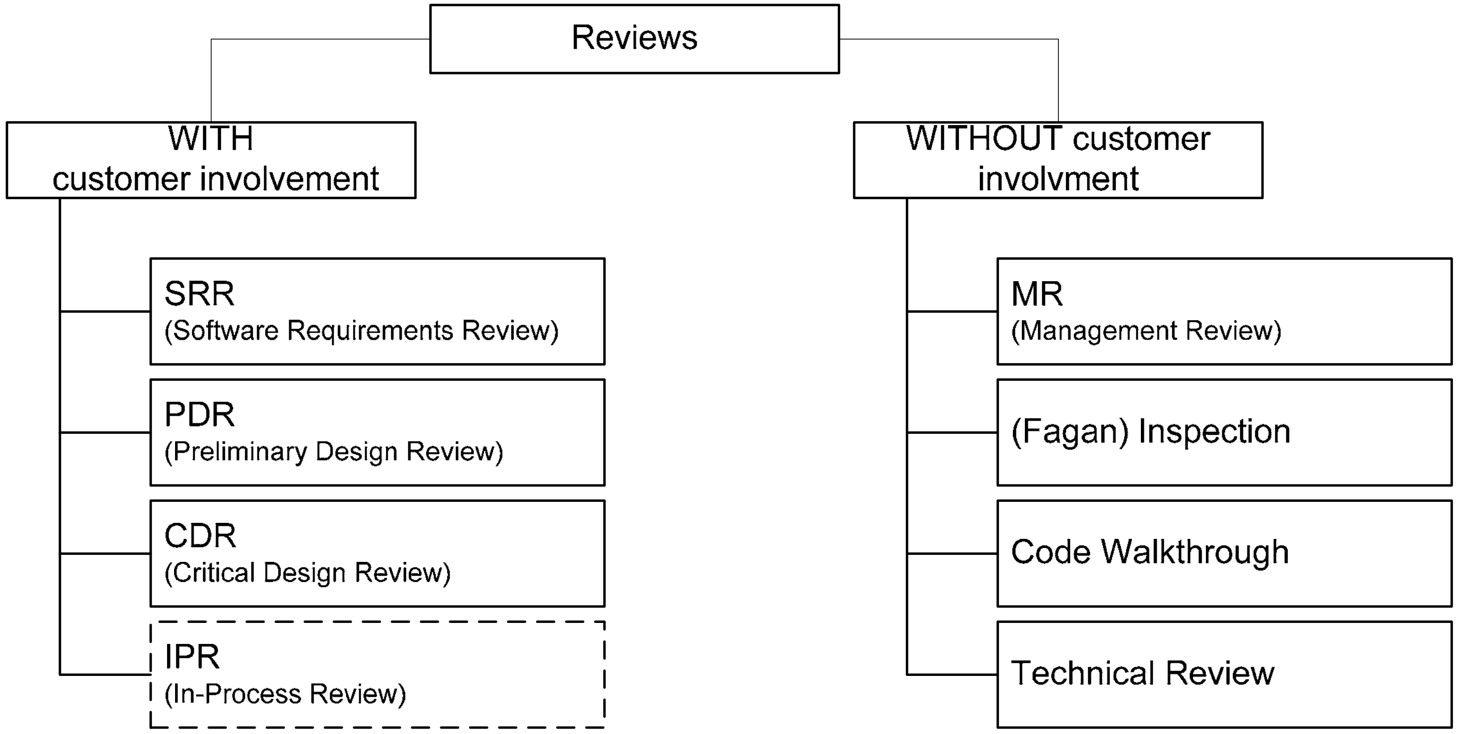


QP: Quality Plan
R: Review / Inspection
TP: Test Plan

P: Prototype
T: Testing

Reviews

- A review is a formal and systematic analysis and assessment process to comment and release any software product by a team of assessors.
[acc. to IEEE Std. 610]
- Reviews focus on assessing products and processes qualitatively, e.g., reviewing models and documents.



Reviews
WITH customer involvement
WITHOUT customer involvment
SRR (Software Requirements Review)
PDR (Preliminary Design Review)
CDR (Critical Design Review)
IPR (In-Process Review)
MR (Management Review)
(Fagan) Inspection
Code Walkthrough
Technical Review

- Different review types focus on different goals.
- Clearly defined roles and assigned tasks.

Comparison: Different review types

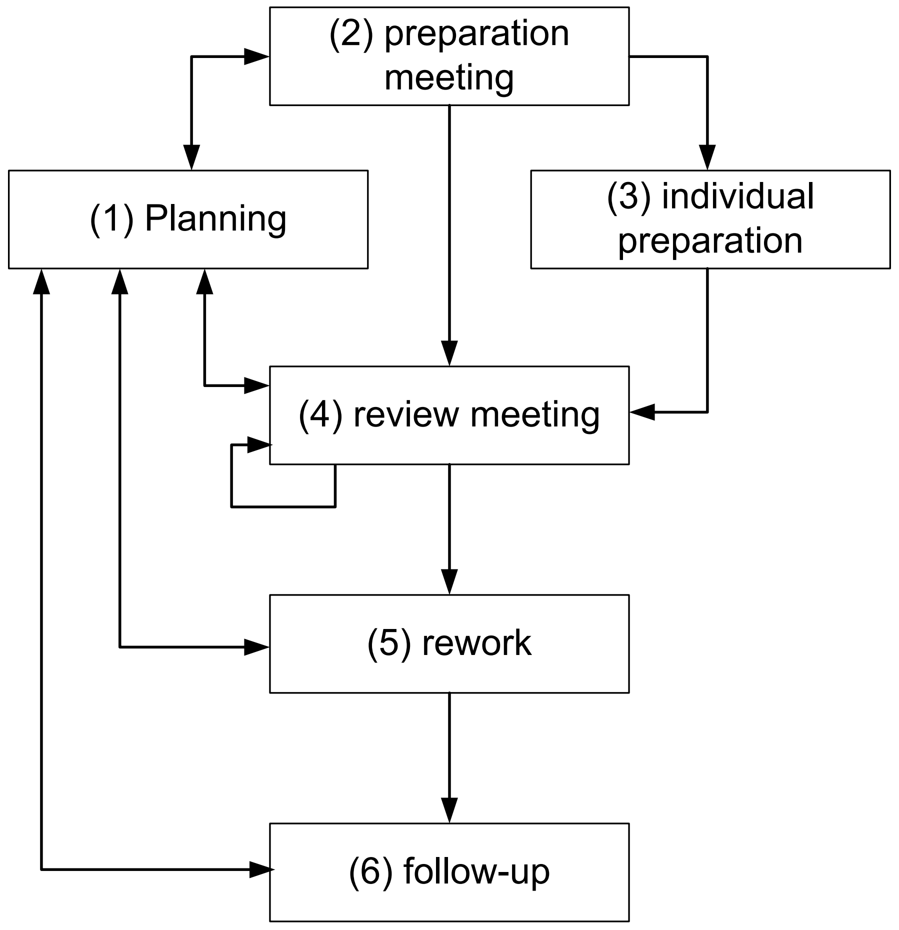
(Technical) Review	Software Inspection (formal review)	Code Walk-Through
Goals ▪ Identify strength and weakness of the object under review ▪ (Process improvement)	Goals ▪ Identify important defects ▪ Process improvement ▪ Collect measurement	Goals ▪ Identify defects and product deviations ▪ Training of users and employees.
Participants Moderator, author, assessor, writer.	Participants Moderator, author, assessor, writer, reader.	Participants Author (= moderator), assessor
Characteristics ▪ Trained moderator ▪ Assessment team provides recommendation for the management	Characteristics ▪ Trained moderator ▪ Reader presents the inspection object stepwise. ▪ Release by the moderator	Characteristics ▪ Author presents the inspection object acc to the sequence of execution ▪ Release by the author.

Review roles

- Size of the review team depends on the project type and the project size.
- Typical team size: 3-6 persons.
- Involved roles:
  - Moderator („keeper of the process“)
    manages the review meeting (including preparation and rework)
  - Reader („keeper of focus and pace“)
  - Writer („preserver of knowledge“)
    prepares the minutes during the review meeting
  - Assessor („Reviewer“)
    Comments on the inspection object.
  - Author
    Explanation of unclear situations, answers open questions.
    BUT: no comments on the solution or defense of the solution.

  - Typically the author must not be a member of the review team.
  - Very important: Assess the product – NOT the author!

Basic Review Process



Reviews are supported by checklists.
Typical duration: 2h

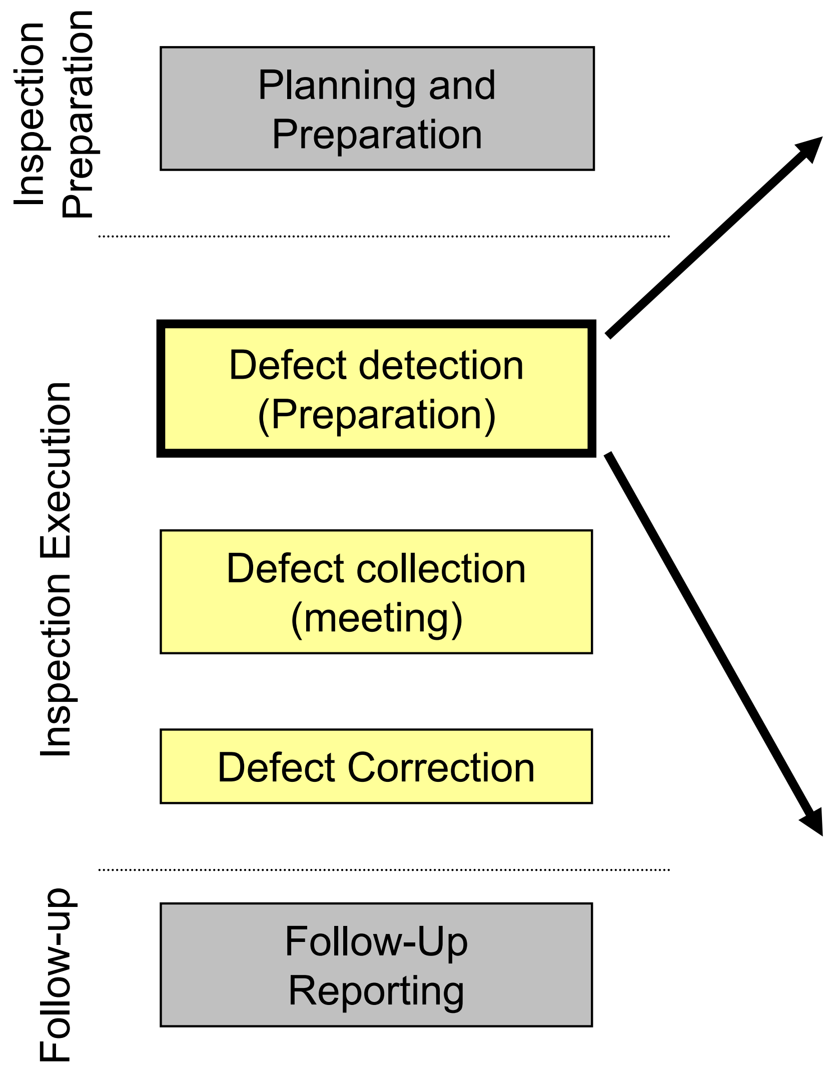
- Planning:
  Definition of objects, inspection goals (exit criteria), pre-conditions (entry criteria), participants, room, time.
- Preparation meeting:
  Introduction to the inspection object (new and complex products).
- Individual preparation
- Review execution: common reading of the artifact, recording of problems;
  Note: in a review meeting problems must be recorded, not fixed!
- Rework: check of the recorded problems and rework of the object.
- Follow up: Review-Report, Feedback
- Repetition of reviews are possible (e.g., in case of a high amount of reported problems).

Software Inspection

- Software Inspection is a (more formal) systematic static analysis technique to verify quality properties of software.
- Supports structured quality improvement (repeatable, traceable and auditable) in an efficient way.
- Inspection does not require executable code
  → applicable to design specifications
  → defect detection in early stages of development.
- Inspection focuses on different defect types (important or less important defects) and locations in the inspected object.

- Guidance of inspectors with reading techniques and guidelines (how to read / traverse a software document).
- A reading technique is a structured approach to support inspectors in their reading approaches. i.e., how to traverse the document under inspection.

Inspection and Reading Techniques



- Inspection preparation is crucial for cost-effective defect detection approaches.
- Often: non-systematic approaches (ad hoc)
- Reading techniques (RTs) are designed to support inspectors in defect detection tasks:
  - Checklists (CL)
  - Perspective Based Reading (PBR) (e.g. from User-, Designer-, Tester view)
  - Usage-Based Reading (UBR) (e.g. based on use-cases)
  - …
- Team meetings focus on defect collection / aggregation (reported by individuals) e.g. net gain vs. net loss; real vs. nominal teams.

Selected Reading Techniques (1)

- **Checklist-Based Reading (CBR)**
  - Predefined CL-Items, which are applied to the inspection object sequentially
  - CLs can spot on different defect types, severity classes, defect locations (design, code) and must be prepared according to the application domain.
  - Typically a checklist consists of a generic part (for general purposes) and a project specific part, focusing on the individual project requirements.
  - CBR is applicable very easy (also by less experienced inspectors), but defect detection performance depends strongly on inspector capability.

- **Perspective Based Reading (PBR)**
  - Defect detection from the view point of different stakeholders: users, developers, tester.
  - Different perspectives enable different types of (additional) defects, e.g., a user will focus on the functionality and the user interface and a developer will focus on architecture and software design.
  - Domain and method experts are necessary.

Selected Reading Techniques (2)

- **Usage-Based Reading (UBR)**
  - Use cases govern inspection process (user focus).
  - Application of use cases and scenarios to requirements documents in a pre-defined order of use cases.
  - Prioritization of the use-cases and scenarios by a group of experts; e.g. domain experts, customers.
  - Goal: focus on crucial and most important defects first.
  - Typical process of UBR inspection:
    (1) Selection of the most important use case,
    (2) Check all use case related parts (supported by guidelines)
    (3) Record of defects and deviations
    (4) Selection of the next use cases (based on the prioritized list).
- In empirical studies, UBR inspection has turned out to be the most effective and efficient way to identify defects early in the domain of administrative systems.

Costs & Benefits of Inspections

- **Cost**
  - Effort of inspectors (person hours);
  - Additional effort if team meetings are conducted.
  - Inspection must be planned within a project plan in advance.

- **Benefit**
  - Early defect detection and correction reduce overall cost and effort.
  - Inspection can provide detailed product and project information as input for resource planning (project and quality management).
  - Inspection output (e.g. defect list) is the baseline for product assessment (e.g., estimation models on remaining defects).
  - Increased knowledge on the product for all inspectors.
  - Common view of the inspection team on the product.
  - Inspection can be used as training for new team members.

Software Testing

- Software testing is an analytical quality assurance approach for software product improvement.
- Testing is considered with program execution to find deviations and defects.
- In traditional software processes test cases are generated in early cycles of development (e.g. in the analysis / design phase) and they are executed during / after software implementation (module, integration, acceptance tests).
- Flexible and agile software processes include test case generation and execution at the same time (e.g., test driven development).
- Testing strategies:
  - Private tests: testing by the developer, “testing” his own piece of software.
  - Module tests: test for identifying implementation defects (test realization against the module specification)
  - Integration tests: testing the interaction of different modules/components.
  - System- and acceptance test: testing the integrated system against customer requirements (system test performed by the developers, acceptance test usually performed by the customer).

Test Cases

- Tests consist of a set of test cases (test suite).

- A test case includes a set of information
  - Ongoing number (reference to the test case)
  - Test case classification:
    - *normal case*: a test case that should not fail
    - *special case*: response to special cases, e.g., div / 0
    - *error cases*: a test case that should not work and produce helpful responses.
  - Pre-conditions: starting state of the system e.g., DB states
  - Input values
  - Actions and activities: what happen with the input values, e.g., a calculation
  - Expected output values.
  - Result: response on the test case

Some Principles on Testing

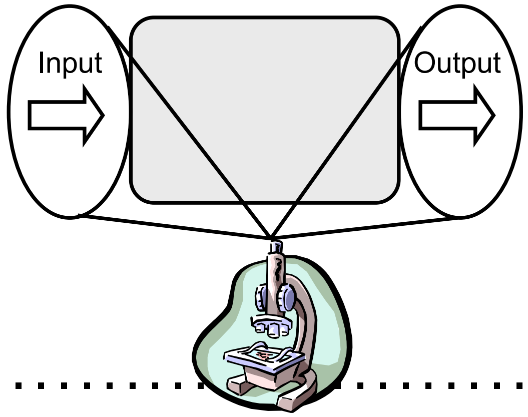
- Every program contains defects – the goal of testing is to identify them!
- The main goal of a tester is to identify defects and to „destroy“ the program (destructive activity).
- No defects found doesn’t mean that a software product doesn’t contain any defects (think about refining – strengthen – the test cases)
- Every identified (and fixed) defect increases the quality of the system (value of testing).
- It is impossible to proof the correctness of a system with functional testing approaches (formal methodologies would be necessary)!
- A developer is responsible for the software product, not the tester!
- Defined role distribution within a software project (Tester <> Developer) – „4-eyes-principles“
- Testing is a core activity in software development, not an additional task at the end of the life-cycle.
- Test case / result reporting is necessary to repeat testing phases (regression testing) after fixing already found defects.

Basic Test Strategies

"Testing is a quality assurance activity in order to find defects".

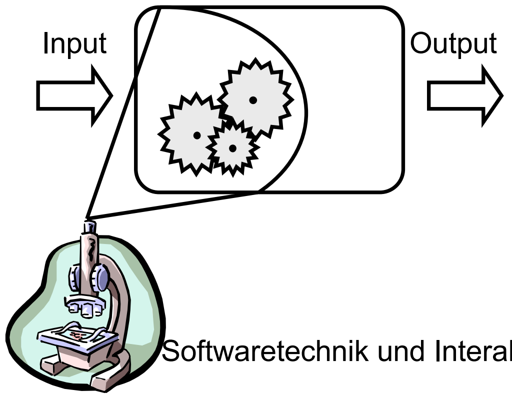
Black Box Tests

- Based on the specification document.
- Independent on the realization of the module.
- Data-driven (Input/Output).
- Requirements coverage.
- Equivalence classes of input data.
- No defect localization possible.



White Box Tests

- Based on software code.
- Knowledge of internal representation necessary.
- logic-driven tests.
- Control-flow coverage.
- Equivalence classes of internal branches and loops.
- Enables defect localization.



Equivalence Classes

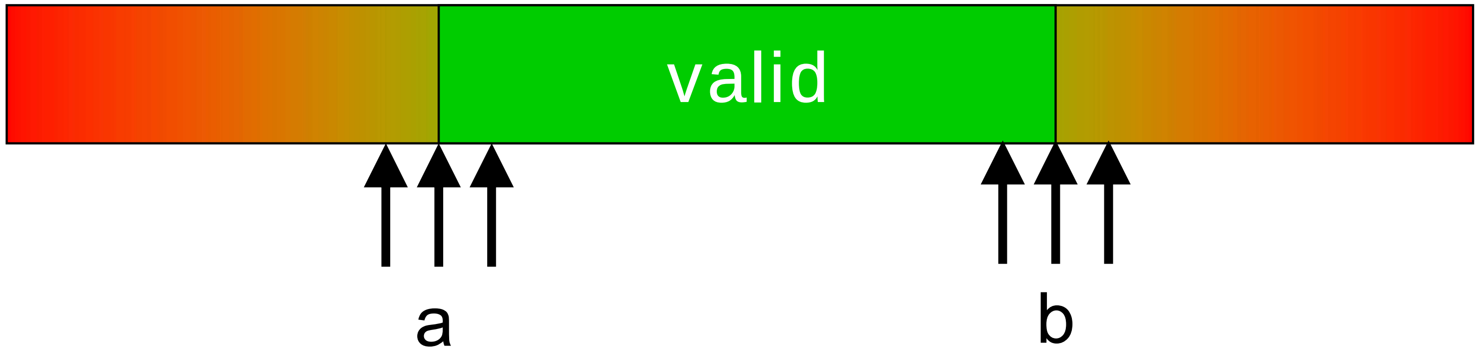
- It is impossible to test all possible test cases (very high effort)!
  e.g., age: integer → all test cases will include age = 1, 2, 3 … 9, 20, 21, …
- Thus,
  - structure input / output variables in (equivalence) classes,
  - and test one representative of each class

- Example: customer requirement defines a range of
  18 ≤ age ≤ 65 years
  → three equivalence classes
    - A1: age < 18 (invalid value)
    - A2: 18 ≤ Alter ≤ 65 (valid value)
    - A3: Alter > 65 (invalid value)

  → three basic test cases:
    - x ∈ A1, e.g., 10
    - x ∈ A2, e.g., 35
    - x ∈ A3, e.g., 70.

Threshold analysis

- Extension of equivalence classes for better test case coverage!
- Typical defects happens at border values …
- Apply threshold values as representatives of a test case class
  → choose a defined value per „class border“.
- Example:
  customer requirement defines a range of 18 ≤ age ≤ 65 years:
  17 (invalid), 18 (valid), 19 (valid), 65 (valid), 66 (invalid)



Summary

- Software defects have a heavy impact on project quality, project duration and project budget.
- Analytical QA approaches must be an integral part of the software development process.
- Reviews and inspection are systematic static analysis technique to verify quality properties of software. Both approaches are applicable to all types of software products (e.g., design specification), because they do not require software code.
- Reading techniques (CBR, UBR, PBR) support reviewers and inspectors in reading software documents.
- Software tests are core activities in software engineering, not just add-ons at the end of the development process.

Referenzen

- Biffl, Stefan: „*Software Inspection Techniques to support Project and Quality Management*", Shaker Verlag, 2001, ISBN: 3-8265-8512-7
- Boehm B.: Software Risk Management: Principles and Practices, IEEE Software 8(1), pp32-41, 1991.
- Dustin E., Rashka J., Paul J.: "Automated Software Testing", Addison-Wesley, 1999, ISBN 0-201-43287-0
- Kappel G.: On Models and Ontologies – or what you always wanted to know about Model-Driven Engineering, Keynote SEE Conference, Munich, 2007.
- Kaner C., Falk J., Nguyen H-Q: "Testing Computer Software", Wiley, 1999, ISBN 0-471-35846-0
- Laitenberger, Oliver: "Cost-Effective Detection of Software Defects Through Perspective-based Inspections"; PhD Theses in Experimental Software Engineering Vol. 1, 2000
- Software Engineering – Best practices: http://best-practice-software-engineering.blogspot.com/
- Sommerville, Ian: „Software Engineering", 8th Edition, Addison Wesley, 2007.

Thank you for your attention

Contact:
Dipl.-Ing. Dietmar Winkler

Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstr. 9-11/188, A-1040 Vienna, Austria

dietmar.winkler@qse.ifs.tuwien.ac.at
http://qse.ifs.tuwien.ac.at

This research work has been supported by a Marie Curie Transfer of Knowledge Fellowship of the European Community's 6th Framework Programme under the contract MTKD-CT-2005-029755: CzechVMXT.