ASESummary ISTQB (beliebig lang)

19. Februar, 2024; rev. 1. April 2025

Linda Riesen (rieselin)

1 Fundamentals of Testing

Testing: not just verifying test obj. but also requirements

- Evaluating work products such as requirements, user stories, designs, and code
- Causing failures and finding defects
- Ensuring required coverage of a test object
- Reducing the risk level of inadequate software quality
- Verifying whether specified requirements have been fulfilled
- Verifying that a test object complies with contractual, legal, and regulatory requirements
- Providing information to stakeholders to allow them to make informed decisions
- Building confidence in the quality of the test object
- Validating whether the test object is complete and works as expected by the stakeholders

1.1 Quality Assurance (QA)

Preventative (Testing is corrective)

1.2 Testing Principles

1. Testing shows the presence, not the absence of defects
2. Exhaustive testing is impossible
3. Early testing saves time and money.
4. Defects cluster together.
5. Tests wear out.
6. Testing is context dependent
7. Absence-of-defects fallacy: software verification does not ensure successful system

1.3 Test activities / tasks => all of these produce work products

this needs traceability

- **Test planning:** define test objective, select approach
- **Test monitoring and test control:** ongoing checking of all test activities and compare actual progress against plan
- **Test analysis:** identify testable features, prioritize/define test conditions
- **Test design:** make test conditions into test cases (coverage)
- **Test implementation:** creating or acquiring the testware
- **Test execution:** run tests
- **Test completion:** usually occurs at project milestones

1.4 Roles in Testing

- a test management role
- a testing role

2 Testing Throughout the Software Development Lifecycle

2.1 Software Development Lifecycle (SDLC)

Early testing (= shift left)

- acceptance test-driven development (ATDD)
  - Derives tests from acceptance criteria as part of the system design process
- behavior-driven development (BDD)
  - Expresses the desired behavior of an application with test cases written in a simple form of natural language, which is easy to understand by stakeholders – usually using the Given/When/Then format
- domain-driven design (DDD)
- extreme programming (XP)
- feature-driven development (FDD)
- Kanban
- Lean IT
- Scrum
- test-driven development (TDD)

2.2 Test Levels

- Component testing (also known as unit testing)
- Component integration testing (also known as unit integration testing)
- System testing focuses (overall behavior and capabilities)
- System integration testing (test interfaces of system)
- Acceptance testing (validation and on demonstrating readiness for deployment)

2.3 Test Types

- **Functional testing** evaluates the functions that a component or system should perform
- **Non-functional testing** “how well the system behaves”
  - Performance efficiency
  - Compatibility
  - Usability (also known as interaction capability)
  - Reliability
  - Security
  - Maintainability
  - Portability (also known as flexibility)
  - Safety
- **Black-box / White-box Testing**
- **Confirmation testing** confirms that an original defect has been successfully fixed
- **Regression testing** confirms that no adverse consequences have been caused by a change
- **Maintenance Testing:** Test whole system after new releases/hot fixes

3 Static Testing

- **dynamic testing:** software needs to be executed for testing
  - does not find defects directly but after failure
- **static testing:** software does not need to be executed for testing
  - less effort (no test cases / tools required)
  - usually done before dynamic testing
  - can also be done on hard to execute / non executable code
  - detect: specific code defects, evaluate maintainability and security (e.g. spelling checkers and readability tools)

3.1 Review Process Activities

- Planning (scope of review, purpose, quality characteristics, focus area, ...)
- Review Initiation (access rights, all have the correct knowledge)
- Individual Review
- Communication and analysis
- Fixing and Reporting

3.2 Roles and responsibilities in reviews

- **Manager**: decides what is to be reviewed and provides resources, such as staff and time for the review
- Author: creates and fixes the work product under review
- **Moderator (also known as the facilitator)**: ensures the effective running of review meetings, including mediation, time management, and a safe review environment in which everyone can speak freely
- **Scribe (also known as recorder)**: collates anomalies from reviewers and records review information, such as decisions and new anomalies found during the review meeting
- **Reviewer**: performs reviews. A reviewer may be someone working on the project, a subject matter expert, or any other stakeholder
- **Review leader**: takes overall responsibility for the review such as deciding who will be involved, and organizing when and where the review will take place

3.3 Review Types

- Informal Review (no defined process)
- Walkthrough (performed by author)
- Technical Review (performed by technically qualified reviewer, led by moderator)
- Inspection (most formal type)

3.4 Success Facotrs for Reviews

- Define clear objectives and measurable exit criteria (excluding participant evaluation).
- Choose the appropriate review type based on objectives, work product, participants, and context.
- Review small chunks to maintain reviewer concentration.
- Provide feedback to stakeholders and authors for improvement.
- Allow adequate preparation time for participants.
- Ensure management support for the review process.
- Integrate reviews into the organization’s culture for learning and improvement.
- Provide training so participants understand their roles.
- Facilitate review meetings effectively.

4 Test Analysis and Design

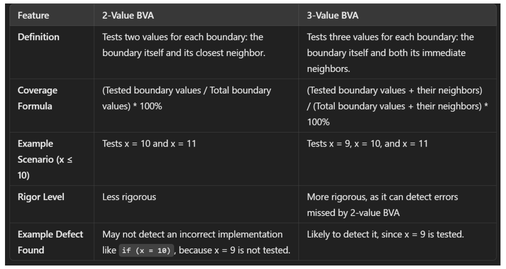
4.1 Black-box test techniques (also known as specification-based techniques)

4.1.1 Equivalence Partitioning (EP)

divides data into partitions based on expectation that all elements of a given partition are processed the same way by test object => only 1 test/partition is sufficient

4.1.2 Boundary Value Analysis (BVA)

black-box testing technique used to test the boundaries of input partitions, as developers often make mistakes at these edges.



Feature	2-Value BVA	3-Value BVA
Definition	Tests two values for each boundary: the boundary itself and its closest neighbor.	Tests three values for each boundary: the boundary itself and both its immediate neighbors.
Coverage Formula	(Tested boundary values / Total boundary values) * 100%	(Tested boundary values + their neighbors) / (Total boundary values + their neighbors) * 100%
Example Scenario (x ≤ 10)	Tests x = 10 and x = 11	Tests x = 9, x = 10, and x = 11
Rigor Level	Less rigorous	More rigorous, as it can detect errors missed by 2-value BVA
Example Defect Found	May not detect an incorrect implementation like `if (x = 10)`, because x = 9 is not tested.	Likely to detect it, since x = 9 is tested.

Abbildung 1: 2-Value BVA and 3-Value BVA

4.1.3 Decision Table Testing

testing the implementation of requirements that specify how different combinations of conditions result in different outcomes

Table conditions

- “T” (true) means that the condition is satisfied
- “F” (false) means that the condition is not satisfied.
- –” means that the value of the condition is irrelevant for the action outcome
- “N/A” means that the condition is infeasible for a given rule
- For actions: “X” means that the action should occur
- For actions: “ ” means that the action should not occur

4.1.4 State Transition Testing

A test case based on a state diagram or state table is usually represented as a sequence of events, which results in a sequence of state changes (and actions, if needed).

- **all states coverage** covers the states
- **valid transitions coverage** covers single valid transitions
- **all transitions coverage** all transitions from state table/diagram

4.2 White-box test techniques (also known as structure-based techniques)

4.2.1 Statement Testing

covers executable statements

4.2.2 Branch Testing

a transfer of control between two nodes in the control flow graph (possible sequences of exec. code)

4.3 Experience-based test techniques

4.3.1 Error guessing

anticipate the occurrence of errors, defects, and failures (based on how app has worked in past, typical types of errors of devs / similar apps)

4.3.2 Exploratory testing

tests are simultaneously designed, executed, and evaluated while the tester learns about the test object

4.3.3 Checklist-based testing

a tester designs, implements, and executes tests to cover test conditions from a checklist (not good repeatability as devs learn not to make same errors)

4.4 Collaboration-based Test Approaches

4.4.1 Collaborative User Story Writing

User Story Components

- Card – the medium describing a user story (e.g., an index card, an entry in an electronic board)
- Conversation – explains how the software will be used (can be documented or verbal)
- Confirmation – the acceptance criteria
- “As a [role], I want [goal to be accomplished], so that I can [resulting business value for the role]”

Acceptance Criteria conditions that an implementation of the user story must meet to be accepted by stakeholders => can be test conditions too

4.4.2 Acceptance Test-driven Development (ATDD)

ATDD is a test-first approach, test cases are created by team members with different perspectives

5 Managing the Test Activities

5.1 Test Planning

A **test plan** describes the test objectives, resources and processes for a test project

- Context
- Assumptions and Constraints of test project
- Stakeholders
- Communication
- Risk Register
- Test approach
- Budget and Schedule

5.1.1 Entry and Exit Criteria

- should be defined for each test level
- Entry Criteria e.g. availability of resources / testware, init. quality
- Exit Criteria (= Definition of Done) e.g. measures of thoroughness, yes/no criteria, time/budget

5.1.2 Estimation Techniques

- Estimation based on ratios (figures from previous projects / standards (like dev:test = 3:2)
- Extrapolation
- Wideband Delphi (e.g. Planning Poker)
- Three-Point Estimation (most optimistic (a), most likely (m), most pessimistic (b) => $E = (a + 4 * m + b)/6$

5.1.3 Test Case Prioritization

- Risk-based prioritization
- Coverage-based prioritization (most branches/statements/function covered in test = highest priority of test)
- Requirements-based prioritization (priorities of requirements)

5.1.4 Test Pyramid

Higher Layer: lower granularity, lower isolation, higher exec time (e.g. end-to-end tests)

5.1.5 Testing Quadrants

- **Quadrant Q1 (technology facing, support the team)**. This quadrant contains component tests and component integration tests. These tests should be automated and included in the CI process.
- **Quadrant Q2 (business facing, support the team).** This quadrant contains functional tests, examples, user story tests, user experience prototypes, API testing, and simulations. These tests check the acceptance criteria and can be manual or automated.
- **Quadrant Q3 (business facing, critique the product).** This quadrant contains exploratory testing, usability testing, user acceptance testing. These tests are user-oriented and often manual.
- **Quadrant Q4 (technology facing, critique the product).** This quadrant contains smoke tests and non-functional tests (except usability tests). These tests are often automated.

5.2 Risk Management

5.2.1 Risk Analysis

risk identification and risk assessment

- Risk Likelihood
- Risk impact (harm)

Project and Product Risks

- Project
  - Organizational Issues
  - People issues
  - Technical Issues
  - Supplier Issues
- Product
  - User dissatisfaction
  - Loss of revenue, trust, reputation
  - Damage to third parties
  - high maintenance costs, overload of help desk
  - criminal penalties / physical damage / injuries / death

5.2.2 Risk Control

risk mitigation and risk monitoring

5.3 Test Monitoring, Test Control and Test Completion

Test monitoring: gather info about testing (test progress, check exit criteria / acceptance criteria)
Test Control: use monitoring to provide necessary corrective actions and guidance

5.4 Configuration Management (CM)

a discipline for identifying, controlling, and tracking work products (test plans, test strategies, test conditions, test cases, test scripts, test results, test logs, and test reports) as configuration items.

5.5 Defect Management

create reports for findings (to provide resolvers sufficient info, means of tracking quality, ideas of improvement)

6 Test Tools

nvm.

regression tests test analysis

Boundary Value Analysis (BVA) valid transitions coverage count test cases how equivalence partitioning

three-point estimation know formula