# Software Testing

Short notes for B.Tech CSE / IT · BCA · MCA · M.Tech

Uncodemy Handwritten Notes · free for students

## I. Introduction

- **Software testing** = running a program to find defects and check that it meets requirements.
- Testing shows the presence of bugs, not their absence.

### ➤ Error, fault, failure

| Term | Meaning |
|---|---|
| Error (mistake) | Human action that is wrong (e.g. wrong logic) |
| Fault (defect / bug) | The wrong part in the code or document |
| Failure | System behaves differently from what is expected when run |

- An error causes a fault; a fault, when executed, may cause a failure.

### ➤ Seven testing principles

1. Testing shows presence of defects
2. Exhaustive testing is impossible
3. Early testing saves time and money
4. Defect clustering (few modules hold most bugs)
5. Pesticide paradox (same tests stop finding new bugs)
6. Testing is context dependent
7. Absence-of-errors fallacy (bug-free but useless software is still a failure)

### ➤ Verification vs validation

| Verification | Validation |
|---|---|
| "Are we building the product right?" | "Are we building the right product?" |
| Checks documents, design, code | Checks the actual working software |
| Static: reviews, walkthroughs, inspections | Dynamic: running tests |
| Done before validation | Done after verification |

**EXAM POINT** QA (quality assurance) is process-oriented and prevents defects. QC (quality control) is product-oriented and finds defects. Testing is part of QC.

# 2. Testing in the SDLC

## ➤ STLC (Software Testing Life Cycle)

1. Requirement analysis
2. Test planning
3. Test case design
4. Test environment setup
5. Test execution
6. Test closure

## ➤ V-Model

```
Requirements  ------------------>  Acceptance testing
  System design  ------------>  System testing
    Architecture design  --->  Integration testing
      Module design  ----->  Unit testing
                \  Coding  /
```

- Each development phase on the left has a matching test phase on the right. Test planning starts early.

## ➤ Levels of testing

| Level | Tests | Done by |
|---|---|---|
| Unit | One function / module | Developer |
| Integration | Interfaces between modules | Developer / tester |
| System | Whole system against requirements | Testing team |
| Acceptance | Ready for users? | Customer / users |

- **Alpha testing**: by users at the developer's site. **Beta testing**: by real users at their own place.

# 3. Static Testing

- Testing without running the code.

- **Reviews**: informal review, walkthrough (author leads), technical review, **inspection** (most formal, has moderator, roles, checklists).
- **Static analysis** tools find unused variables, unreachable code, standard violations.
- Benefits: finds defects early when they are cheap to fix.

# 4. Black Box Testing Techniques

- Tests based on requirements only; the tester does not look at the code.

## ➤ Equivalence Class Partitioning (ECP)

- Divide input into groups that should behave the same. Test one value from each group.

**EXAMPLE** Age field accepts 18 to 60. Classes: invalid (< 18), valid (18–60), invalid (> 60). Test values: 10, 35, 70.

## ➤ Boundary Value Analysis (BVA)

- Bugs often occur at the edges. Test min − 1, min, min + 1, max − 1, max, max + 1.

**EXAMPLE** Range 18 to 60. BVA values: 17, 18, 19, 59, 60, 61.

**FORMULA** BVA for n variables (single-fault, nominal values) = 4n + 1 test cases. Robust BVA = 6n + 1.

**EXAMPLE** 2 input variables → 4(2) + 1 = 9 test cases. Robust → 6(2) + 1 = 13.

## ➤ Decision table testing

- For combinations of conditions. Each column (rule) = one test case.

| Conditions / Actions | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| Valid user? | Y | Y | N | N |
| Valid password? | Y | N | Y | N |
| Action: login | Yes | No | No | No |
| Action: show error | No | Yes | Yes | Yes |

- n binary conditions → 2^n rules (before merging).

### ➤ Other black box techniques

- **State transition testing**: for systems with states (ATM card blocked after 3 wrong PINs).
- **Use case testing**: test end-to-end user scenarios.
- **Error guessing**: use experience to guess likely bugs (empty input, null, special characters).
- **Cause-effect graphing**: convert causes and effects to a decision table.

## 5. White Box Testing Techniques

- Tests based on the internal code structure.

### ➤ Coverage criteria

| Coverage | Every … executed at least once |
|---|---|
| Statement | Statement |
| Branch / decision | True and false outcome of each decision |
| Condition | True and false of each individual condition |
| Path | Independent path |

- Branch coverage is stronger than statement coverage. 100% branch coverage implies 100% statement coverage.

### ➤ Cyclomatic complexity (McCabe)

- Number of independent paths in the control flow graph = minimum test cases for basis path testing.

**FORMULA** $V(G) = E - N + 2P$ ($E$ = edges, $N$ = nodes, $P$ = connected parts, usually 1). Also $V(G)$ = number of decision nodes + 1 = number of regions.

**EXAMPLE** A flow graph has 9 edges and 7 nodes. $V(G) = 9 - 7 + 2 = 4$, so 4 independent paths to test.

```
if (a > b)          // decision 1
    x = a;
else
    x = b;
while (x > 0)       // decision 2
    x--;
// 2 decisions -> V(G) = 2 + 1 = 3
```

TIP Range of V(G): 1–10 simple, 11–20 moderate, above 20 complex and risky.

### ➤ Other white box methods

- **Data flow testing**: follow definitions and uses of variables (def-use pairs).
- **Mutation testing**: make small changes (mutants) in code; good tests should "kill" them.

FORMULA Mutation score = killed mutants / (total mutants − equivalent mutants) × 100.

## 6. Integration and System Testing

### ➤ Integration approaches

| Approach | Order | Needs |
|---|---|---|
| Big bang | All modules at once | Nothing, but bugs hard to locate |
| Top-down | From main module downwards | Stubs (dummy called modules) |
| Bottom-up | From lowest modules upwards | Drivers (dummy calling modules) |
| Sandwich (hybrid) | Both directions | Stubs and drivers |

REMEMBER Stub is called by the module under test. Driver calls the module under test.

### ➤ Non-functional testing

- **Performance**: response time under load. **Load**: expected users. **Stress**: beyond the limit until it breaks.
- **Security**, **usability**, **compatibility**, **reliability**, **recovery**, **volume** testing.

### ➤ Other types

- **Smoke testing**: quick check that the main build works (build verification).

- **Sanity testing**: narrow check that a specific fix works.
- **Regression testing**: re-test after changes to make sure old features still work.
- **Retesting**: run the failed test again after the fix.
- **Exploratory testing**: learn and test at the same time, no fixed scripts.

# 7. Test Management and Defects

- **Test plan** (IEEE 829 style): scope, approach, resources, schedule, entry / exit criteria, risks, deliverables.
- **Test case** parts: ID, title, preconditions, steps, test data, expected result, actual result, status.
- **Traceability matrix (RTM)**: maps requirements to test cases so nothing is missed.

### ➤ Defect life cycle

```
New -> Assigned -> Open -> Fixed -> Retest -> Verified -> Closed
                     |                 |
                Rejected/Deferred   Reopened -> Assigned
```

- **Severity** = impact on the system (set by tester). **Priority** = how soon to fix (set by manager / business).

> **EXAMPLE** Company logo spelt wrong on the home page = low severity, high priority. Crash in a rarely used report = high severity, lower priority.

> **FORMULA** Defect density = number of defects / size (e.g. per KLOC). Defect removal efficiency = defects found before release / total defects × 100.

# 8. Automation and Tools

- **Automation** suits regression, repeated, data-driven and load tests. Manual suits exploratory, usability and one-time tests.

| Tool | Use |
|---|---|
| Selenium | Web UI automation |
| JUnit / TestNG | Unit testing in Java |
| PyTest | Unit testing in Python |
| JMeter | Load and performance testing |
| Postman | API testing |
| Jira | Defect and test management |

```
@Test
public void testAdd() {
    Calculator c = new Calculator();
    assertEquals(5, c.add(2, 3));
}
```

- **TDD** (test-driven development): write a failing test → write code to pass → refactor.
- **BDD**: tests in plain language (Given / When / Then), e.g. Cucumber.
- **Agile testing**: testers work inside each sprint; continuous testing in CI/CD.

# 9. Most Asked Questions (practice)

1. Define error, fault and failure. Explain the seven testing principles.
2. Verification vs validation. QA vs QC.
3. Explain the V-model and levels of testing.
4. Explain equivalence partitioning and BVA with an example.
5. Prepare a decision table for a login screen.
6. Find cyclomatic complexity of a given code and list independent paths.
7. Statement vs branch vs path coverage.
8. Top-down vs bottom-up integration. Stub vs driver.
9. Explain the defect life cycle. Severity vs priority.
10. Smoke vs sanity vs regression testing.

> **TIP** BVA / ECP test cases, cyclomatic complexity and decision tables are the main numericals. Learn the defect life cycle and V-model diagrams.