

# UNIT 6

## Software Testing and Quality Assurance

(6 Hours) — Software Engineering (CACS253)

### 6.1 Verification and Validation

**Verification and validation (V&V)** is the process of checking that a software system meets its specification and fulfils the needs of its customer. It is a whole life-cycle process, applied at each stage.

| Verification                                      | Validation                                               |
|---------------------------------------------------|----------------------------------------------------------|
| “Are we building the product <b>right</b> ?”      | “Are we building the <b>right</b> product?”              |
| Checks conformance to specification and design    | Checks that the software meets the customer's real needs |
| Mostly static: reviews, inspections, walkthroughs | Mostly dynamic: testing the executing software           |
| Done during each development phase                | Done mainly at the end of a phase or of the project      |

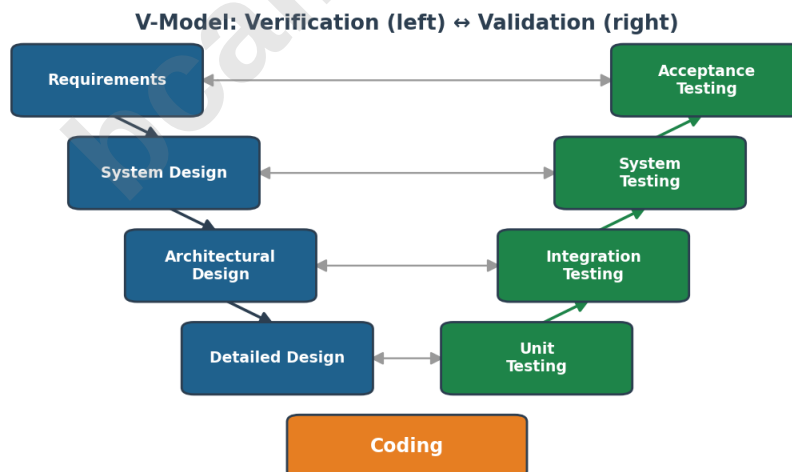


Figure 6.1: V-model relating development phases to testing levels

There are two complementary approaches to checking: **software inspections (static)** analyse the static system representation to discover problems; **software testing (dynamic)** exercises and observes product behaviour. Testing can show the **presence** of errors, but never their absence (Dijkstra).

### 6.2 Techniques of Testing

## Black-Box Testing (Functional Testing)

The tester knows only the inputs and expected outputs; the internal structure of the program is ignored. Test cases are derived from the specification.

- **Equivalence partitioning** – divide input domain into classes that are expected to behave similarly and test one value from each class.
- **Boundary value analysis** – test values at and around the boundaries of each class, where errors are most likely (min, min+1, max-1, max, and just outside).
- **Decision-table testing, state-transition testing, cause-effect graphing and use-case testing.**

### Example – Equivalence Partitioning + Boundary Value

A field accepts age from **18 to 60** inclusive.

**Equivalence classes:** invalid (<18), valid (18–60), invalid (>60).

**Boundary values to test:** 17, 18, 19, 59, 60, 61.

## White-Box Testing (Structural / Glass-Box Testing)

The tester has full knowledge of the source code and derives test cases from the program structure so that all logical paths are exercised.

- **Statement coverage** – every statement executed at least once.
- **Branch (decision) coverage** – every true/false outcome of each decision taken.
- **Condition coverage** – each condition in a decision takes both true and false values.
- **Path testing** – execute all independent paths, using **cyclomatic complexity** to count them.
- **Loop testing** – test loops with 0, 1, typical and maximum iterations.

### Cyclomatic Complexity

$V(G) = E - N + 2$  (E = edges, N = nodes of the flow graph) = number of decision points + 1.

Example: a module with 3 *if* statements:  $V(G) = 3 + 1 = 4 \rightarrow$  at least **4 independent paths** to test.

| Basis            | Black-Box          | White-Box                 |
|------------------|--------------------|---------------------------|
| Knowledge needed | Specification only | Source code and structure |
| Performed by     | Testers, users     | Developers                |
| Focus            | Functionality      | Internal logic and paths  |

| Basis | Black-Box                    | White-Box         |
|-------|------------------------------|-------------------|
| Level | Mostly system and acceptance | Mostly unit level |

## Inspections

Software inspection is a formal peer-review technique in which a team examines a work product (requirements, design or code) to find defects *without executing it*. Fagan inspections define roles: author, moderator, reader, recorder, inspectors. Inspections can find 60–90% of defects and detect many errors at once (testing may mask errors with other errors). They also help verify non-functional attributes like standards compliance.

- **Preparation:** team members study the material with a checklist of common errors.
- **Inspection meeting:** defects are identified (not fixed); typical rate ~ 100–200 lines of code per hour.
- **Rework and follow-up:** the author fixes the defects; moderator verifies.

## 6.3 Levels of Testing

| Level               | Description                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unit testing        | Individual functions, classes or modules tested in isolation, usually by the developer using drivers and stubs. Purpose: find local logic and coding errors.                              |
| Integration testing | Tested units are combined and interfaces between them tested. Approaches: <b>top-down</b> (stubs), <b>bottom-up</b> (drivers), <b>sandwich</b> and <b>big-bang</b> .                      |
| Interface testing   | Focuses on interface errors between components: parameter, shared-memory, procedural and message-passing interfaces. Errors include interface misuse, misunderstanding and timing errors. |
| System testing      | The complete integrated system is tested against the requirements. Includes functional, performance, stress, security, recovery and usability testing.                                    |
| Alpha testing       | Done at the <b>developer's site</b> by internal staff or selected users in a controlled environment.                                                                                      |
| Beta testing        | Done at the <b>customer's site</b> by real users in the real environment; developer is not present. Feedback used for final fixes.                                                        |
| Acceptance testing  | The customer validates that the system meets the contract requirements before accepting it.                                                                                               |
| Regression testing  | Re-running previous tests after a change to ensure that fixes or new features have not broken existing functionality; usually automated.                                                  |

## Stubs and Drivers

- **Stub** – a dummy module that replaces a lower-level module *called* by the module under test (top-down testing).
- **Driver** – a test program that *calls* the module under test and passes it data (bottom-up testing).

| Type of System Testing | Checks                                              |
|------------------------|-----------------------------------------------------|
| Performance testing    | Response time, throughput under expected load       |
| Stress testing         | Behaviour beyond design load – finds breaking point |
| Security testing       | Resistance to unauthorised access and attacks       |
| Recovery testing       | Ability to recover from crashes and failures        |
| Usability testing      | Ease of use and learnability                        |

## 6.4 Design of Test Cases

A **test case** is a set of inputs, execution conditions and expected results developed for a particular objective. Good test cases have a high probability of finding an as-yet-undiscovered error, are neither redundant nor too complex, and have a defined expected result.

| Field                  | Content (Login example)                                |
|------------------------|--------------------------------------------------------|
| Test case ID           | TC-01                                                  |
| Objective              | Verify login with valid credentials                    |
| Preconditions          | User account exists and is active                      |
| Test steps / input     | Enter user = 'ram', password = 'Secret@1'; click Login |
| Expected result        | User redirected to dashboard                           |
| Actual result / status | (filled during execution) – Pass / Fail                |

- Derive tests systematically from requirements (each requirement has at least one test – **traceability**).
- Include both **valid** and **invalid** inputs and boundary conditions.
- Record results, log defects and retest after fixes.
- Test-driven development (TDD): write the test *before* the code.

## 6.5 Quality Management Activities

Software quality management provides an independent check on the software development process. It ensures that project deliverables are consistent with organisational standards and goals. Three main activities:

| Activity               | Description                                                                                                                                                        |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Quality assurance (QA) | Establishing a framework of organisational procedures and standards that lead to high-quality software; <i>process-oriented</i> , preventive.                      |
| Quality planning       | Selecting applicable procedures and standards for a specific project and modifying them as required; produces the quality plan.                                    |
| Quality control (QC)   | Ensuring that the project team follows the quality procedures and standards; detects defects in the product through reviews and testing; <i>product-oriented</i> . |

- **Reviews** (technical, management, progress) are the main validation method for quality.
- **Software measurement and metrics** assess quality attributes (defect density, MTTF, complexity, size).
- **Documentation standards** and **configuration management** support quality.

## 6.6 Product and Process Quality

The quality of the software product depends on the quality of the process used to develop it, especially for large projects. For manufactured goods the link is direct; for software the relationship is more complex because design and creativity are involved, but process quality still strongly influences product quality.

| Product Quality                                                                            | Process Quality                                                                  |
|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| Attributes of the delivered software – reliability, usability, efficiency, maintainability | How the software is developed – standards, reviews, planning, tools, team skills |
| Assessed by testing, metrics and user feedback                                             | Assessed by process audits and maturity models                                   |
| Directly experienced by the customer                                                       | Indirectly improves the product; reduces variability                             |

Factors affecting quality include development technology, process quality, people quality, cost, time and schedule.

## 6.7 Standards: ISO 9000 and CMM

### ISO 9000

ISO 9000 is a family of international standards for quality management systems. **ISO 9001** is the model that applies to organisations that design, develop, produce and service products

including software (guided by ISO 9000-3). The standard defines a set of quality-process requirements – not the quality of the product itself. An organisation is **certified** after an external audit.

- Focus on documented procedures, management responsibility, resources, product realisation and continual improvement.
- Certification shows that a documented quality process exists and is followed; it does not guarantee product quality.
- Key elements: customer focus, leadership, involvement of people, process approach, improvement, evidence-based decisions and relationship management.

## Capability Maturity Model (CMM)

Developed by the Software Engineering Institute (SEI), CMM assesses the maturity of an organisation's software process on a five-level scale. Each level (except Level 1) has **Key Process Areas (KPA's)** that must be satisfied.

| Level | Name       | Characteristics and Key Focus                                                                                                            |
|-------|------------|------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | Initial    | Ad hoc, chaotic processes; success depends on individual heroics.                                                                        |
| 2     | Repeatable | Basic project management (requirements management, planning, tracking, configuration management, QA). Earlier successes can be repeated. |
| 3     | Defined    | Standard, documented process across the organisation (organisation process definition, training, peer reviews, integrated management).   |
| 4     | Managed    | Process and product quality measured quantitatively and controlled using metrics.                                                        |
| 5     | Optimising | Continuous process improvement using quantitative feedback, defect prevention and new technologies.                                      |

| Basis       | ISO 9001                                  | CMM                             |
|-------------|-------------------------------------------|---------------------------------|
| Type        | Pass/fail certification by external audit | Five-level maturity rating      |
| Scope       | Any industry                              | Software processes specifically |
| Improvement | Minimum acceptable quality system         | Path for continuous improvement |

*Note: CMM has since evolved into CMMI (Capability Maturity Model Integration), which covers development, services and acquisition.*

### Key Takeaways

- ✓ Verification = building the product right (static); validation = building the right product (dynamic).
- ✓ Black-box tests use spec (equivalence partitioning, boundary value); white-box tests use code (statement, branch, path coverage, cyclomatic complexity).
- ✓ Levels: unit → integration → interface → system → alpha/beta → acceptance; regression testing after every change.
- ✓ Test cases need ID, inputs, conditions and expected results.
- ✓ Quality management = quality assurance, quality planning and quality control.
- ✓ Good process leads to good product. ISO 9001 certifies quality systems; CMM has five levels: Initial, Repeatable, Defined, Managed, Optimising.

— *End of Unit 6* —

bcalink.com