# Requirements and Architecture Testing

**Today’s Topics (p. 3)**

1. **The Big Questions**
2. **Square of Opposition**
3. **Testing Requirements**
   * Revisiting the ClassifyTriangle Requirements
4. **Requirements Reviews**
5. **Traceability Matrices**
6. **Checklists**
7. **Prototypes**
8. **Architecture Review Boards**
9. **Capability Maturity Model**

**The Big Questions for Early Decisions (p. 4)**

* What can we do **early on** to **minimize faults** in the system later?
* Can we take steps early to **reduce failures** in software projects?
* What general **approaches improve software quality**?

**Understanding Requirements Oppositions (p. 5)**

* **We contradict ourselves** – Different requirements may conflict.
* Logical contradictions arise from different perspectives in software.

**Static Interactions Known to Aristotle (p. 6)**

* Aristotle defined **four logical propositions**:
  1. **Universal Affirmative** (All S is P)
  2. **Universal Negative** (No S is P)
  3. **Particular Affirmative** (Some S is P)
  4. **Particular Negative** (Some S is not P)
* These propositions form the **Square of Opposition**, widely used in programming.

**Aristotle's Square of Opposition (p. 7)**

* **Contradictory Relations**: Exactly one condition is true.
* **Contrary Relations**: Two conditions **cannot both** be true.
* **Sub-contrary Relations**: Two conditions **cannot both** be false.
* **Subalterns**: If the **super-altern is true**, the **subaltern must be true**.

**Programming Examples of the Four Relations (p. 8)**

* **Contradictory:**
  + select-case statements → Only **one** clause is true at a time.
* **Contrary:**
  + **Call forwarding loops** → May cause faults due to global contradictions.
* **Sub-contrary:**
  + if-else statements → Either the if or else must be satisfied.
* **Subalternation:**
  + Redundant if-else statements → Some conditions **never execute**.

**Requirements Testing (p. 9)**

* **Example Requirement:**
  + Function classify\_triangle(a, b, c) → Determines triangle type.
* **Questions to Consider:**
  + Are these **good requirements**?
  + Are they **complete**?
  + Are **important details missing**?

**Revisiting the Classify Triangle Requirements (p. 10)**

* **What happens if the parameters describe an invalid triangle?**
* **What are the return values?**
* **Are there valid constraints?** (Min/max values, negative values, decimals)
* **Handling precision:**
  + Does 3.69, 3.7, 3.71 qualify as an **equilateral triangle**?
  + Is 4, 5, 41 a valid **right triangle**?

**Analyzing the Requirements (p. 11-13)**

* **Clarifications Needed:**
  + Should return values include **"right scalene"** or **"right isosceles"?**
  + How should **invalid triangles** be handled? (None, exception, or special value)
  + **Decimal inputs:** What level of **precision** is required?

**Gathering Requirements Approaches (p. 14)**

1. **Traditional Requirements (Plan-Driven)**
2. **Use Cases (RUP)**
3. **User Stories (Agile)**

**Plan-Driven Requirements (p. 15)**

* **Business Analysts** interview customers to **define features**.
* **Example Requirements:**
  + System **must allow** users to enter name, address, email.
  + System must be **available 24/7/365** with **99.9% reliability**.
* **Business Requirements Document (BRD):**
  + Serves as a **contract** between customer and developers.

**Use Cases (p. 16)**

* **Describe all scenarios** (expected, alternate, error flows).
* Example: **Music Streaming Service Use Case**
  + User plays a song.
  + System fetches and streams the music.

**User Stories (Agile) (p. 17-18)**

* **Short statements from customers**:
  + "As a user, I want to…"
* **User Story Components:**
  + **Title**
  + **Acceptance Tests**
  + **Priority**
  + **Story Points**
  + **Description**

**What Could Possibly Go Wrong? (p. 19)**

* **NASA Mars Climate Orbiter failure:**
  + NASA **sent metric data**, but orbiter **expected imperial** measurements.
* **Lesson:** Misalignment of **requirements can lead to major failures**.

**Requirements Testing Approaches (p. 20)**

* **Static Reviews:**
  + Inspections, Walkthroughs, Peer Reviews, Checklists.
* **Dynamic Testing:**
  + Prototypes, Models, Verification & Validation.

**Formal Inspections (p. 22-26)**

* **Inspect everything**: Requirements, designs, code, test cases.
* **When?** Before baseline approval.
* **Common Issues:**
  + **Ambiguity, missing details, standards violations, design flaws.**
* **Effectiveness:**
  + **60-70% defect removal efficiency.**

**Checklists for Reviews & Inspections (p. 33-36)**

* **Requirements Checklist:**
  + Prioritization, clarity, traceability, verifiability, consistency.
* **Code Reviews Checklist:**
  + Code readability, security threats, coverage ≥ 80%.

**Requirements Traceability (p. 46-51)**

* Ensures that **every requirement is implemented & tested**.
* **Benefits of Traceability:**
  + Helps measure **test coverage**.
  + Identifies **missing features**.
  + Useful for **audits & inspections**.

**Prototypes as a Testing Tool (p. 54-60)**

* **Prototyping Benefits:**
  + **Rapid Testing of UI, Features, and Usability.**
  + **Two types:**
    1. **Throwaway Prototypes** (Quick tests, discarded afterward).
    2. **Evolutionary Prototypes** (Gradually improved into the final product).

**Architecture Review Boards (ARB) (p. 62-71)**

* **Ensures quality before construction.**
* **Key Principles:**
  + **Independent Experts** conduct reviews.
  + **Open Review Process** benefits the project.
  + **Detect problems early, spread knowledge.**

**Capability Maturity Model (CMM) (p. 73-75)**

* **Maturity levels for software development organizations:**
  1. **Initial:** Chaotic, undefined processes.
  2. **Repeatable:** Basic project management.
  3. **Defined:** Standardized software processes.
  4. **Managed:** Measurable quality control.
  5. **Optimized:** Continuous process improvement.

**Final Thoughts (p. 76)**

* **Defining precise requirements is critical for software success.**
* **Traceability and inspections save costs and improve quality.**
* **Architecture reviews ensure scalable and robust designs.**
* **Prototypes & user testing enhance usability and prevent failures.**