Systems Engineering for DoD

Testing

Basics

- Management Involvement
- Corporate Culture
- Objectives
- Integrity
- A ‘commitment to competence’
- Good testing does not just happen, it must be planned, and a testing policy should be the cornerstone of that plan.
- The structure that is put in place to develop and update policies, standards, and procedures must involve both staff and management.

Negative Impact Issues

- What the CSTE calls ‘incentives and temptations’
- Pressure to meet performance targets (realistic or unrealistic)
- Nonexistent or ineffective controls
- Over decentralization
- Weak or ineffective audit policies
- Ineffective deterrents

MIL-STD-810 & MIL-STD-2105D

MIL-STD-810 = “Will my equipment survive the environment it’s used in?”

MIL-STD-2105D = “Will my munition stay safe if something goes wrong?”

MIL-STD-810

- MIL-STD-810 is a U.S. military standard that establishes a series of laboratory and field test methods to evaluate how equipment performs under various environmental conditions. It is widely used not only by the Department of Defense but also in commercial industries (such as aerospace, automotive, and electronics) to ensure ruggedness and durability. The original standard was issued in 1962, and it has been updated several times (current version is MIL-STD-810H, released in 2019).

MIL-STD-810

MIL-STD-810 includes procedures for a wide range of environmental stresses, such as:

Temperature extremes (high, low, rapid changes)

Humidity

Shock and vibration

Sand and dust exposure

Salt fog / corrosion resistance

Rain and water immersion

Altitude and pressure changes

Solar radiation (UV exposure)

Fungus resistance

Each test method specifies how to replicate these environments in controlled conditions

MIL-STD-810

Test Category	What It Simulates
High Temperature	Operation and storage in hot climates (deserts, enclosed vehicles in sun).
Low Temperature	Operation and storage in cold climates (arctic, high-altitude).
Temperature Shock	Sudden temperature transitions (e.g., moving from hot outdoor air to a cold aircraft).
Humidity	Exposure to warm, moist environments that may cause condensation or mold.
Rain / Drip	Heavy rainfall, dripping water, blowing rain conditions.
Immersion	Submersion in water (short-term or accidental).
Sand & Dust	Wind-blown particles in deserts or dusty environments.
Salt Fog (Corrosion)	Salt-laden air that causes corrosion (common in coastal/marine environments).
Solar Radiation (UV)	Long-term sun exposure leading to heat buildup or UV degradation.
Altitude / Pressure	High-altitude, low-pressure environments (aircraft operations, mountain regions).
Explosive Atmosphere	Operation in fuel-vapor-rich environments (testing ignition safety).
Shock	Impacts, drops, and sudden physical stresses.
Vibration	Continuous vibration from vehicles, aircraft, or machinery.
Gunfire Vibration	Shock and vibration specific to weapon-mounted equipment.
Fungus	Warm, damp conditions promoting fungal growth.
Icing / Freezing Rain	Exposure to freezing precipitation that forms ice on surfaces.
Acoustic Noise	Exposure to high noise/vibration from jet engines, machinery, or artillery.
Toxic Fluids	Exposure to fuels, oils, hydraulic fluids, cleaning agents, etc.
Ballistic Shock	High-intensity shock loads (e.g., from nearby explosions).

MIL-STD-2105D Hazard Assessment Tests for Non-Nuclear Munitions

Test	Purpose
Fast Cook-off Test	Evaluates response when exposed to **sudden, intense fire** (e.g., fuel fire).
Slow Cook-off Test	Measures how munitions react to **gradual heating** over time.
Bullet Impact Test	Determines reaction to **small arms fire**.
Fragment Impact Test	Assesses response to **high-velocity fragments** (e.g., nearby explosion).
Sympathetic Detonation	Evaluates whether one munition detonates when a nearby one explodes.
Shaped Charge Jet Impact	Tests vulnerability to **armor-piercing shaped charges**.
Shock Tests	Simulates impact or drops during handling/transport.

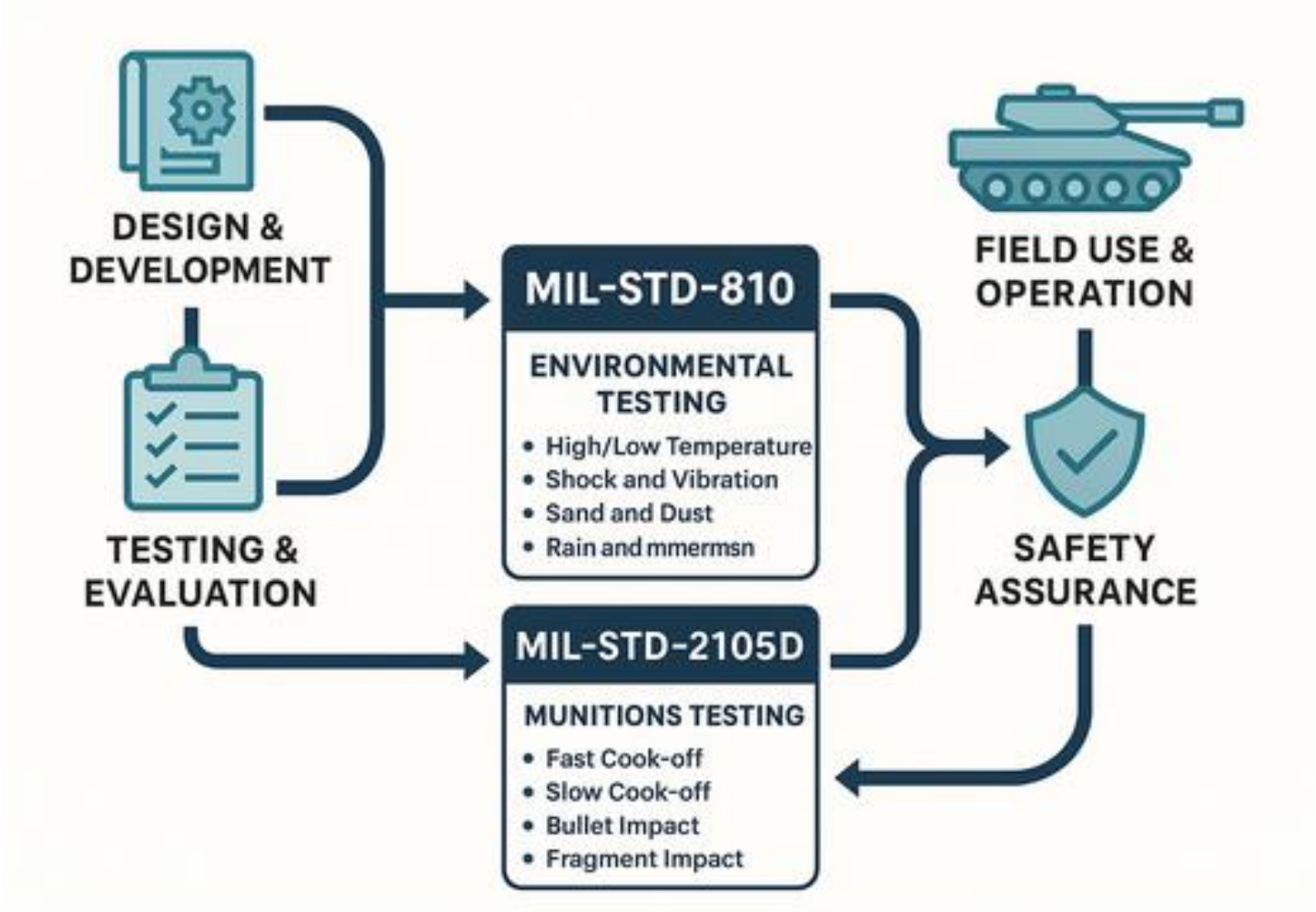
MIL-STD-2105D Hazard Assessment Tests for Non-Nuclear Munitions

- Munitions tested under MIL-STD-2105D are classified based on reaction severity:
- **Type I – Detonation**: Violent, catastrophic explosion.
- **Type II – Deflagration**: Rapid burning/explosion, less violent than detonation.
- **Type III – Explosion**: Localized but significant blast.
- **Type IV – Burning**: Sustained combustion without detonation.
- **Type V – No Reaction**: No hazardous response observe

MIL-STD-2105D Hazard Assessment Tests for Non-Nuclear Munitions

Aspect	MIL-STD-810	MIL-STD-2105D
Full Title	Environmental Engineering Considerations and Laboratory Tests	Hazard Assessment Tests for Non-Nuclear Munitions
Primary Purpose	Evaluate environmental durability of equipment and materials.	Assess safety hazards of munitions under accidental conditions.
Scope	General equipment: vehicles, electronics, aerospace, consumer gear.	Ordnance only: bombs, rockets, missiles, warheads, propellants, explosives.
Focus	How items perform in real-world environments (heat, vibration, dust, etc.).	How munitions react to unintended stimuli (fire, impact, shock, etc.).
Key Tests	- High/low temperature- Humidity- Shock & vibration- Sand/dust- Salt fog- Rain/immersion- Solar radiation- Fungus- Altitude	- Fast cook-off- Slow cook-off- Bullet impact- Fragment impact- Sympathetic detonation- Shaped charge jet impact- Shock/impact tests
Outcome	Demonstrates ruggedness and survivability for operational use.	Classifies munitions' reaction severity (Detonation, Deflagration, Burn, etc.).
Applicability	Military & commercial (aerospace, automotive, electronics, rugged devices).	Defense industry (DoD, NATO, contractors for explosives and ordnance systems).
First Issued	1962 (current: MIL-STD-810H, 2019).	1980s (current: MIL-STD-2105D).
Alignment	Tailored testing—select methods based on mission environment.	Standardized safety assessment—tests required for Insensitive Munitions (IM) compliance.
End Goal	Ensure equipment functions reliably under expected environmental conditions.	Ensure munitions are safe to store, transport, and handle, reducing accidental hazards.

MIL-STD-810 & MIL-STD-2105D



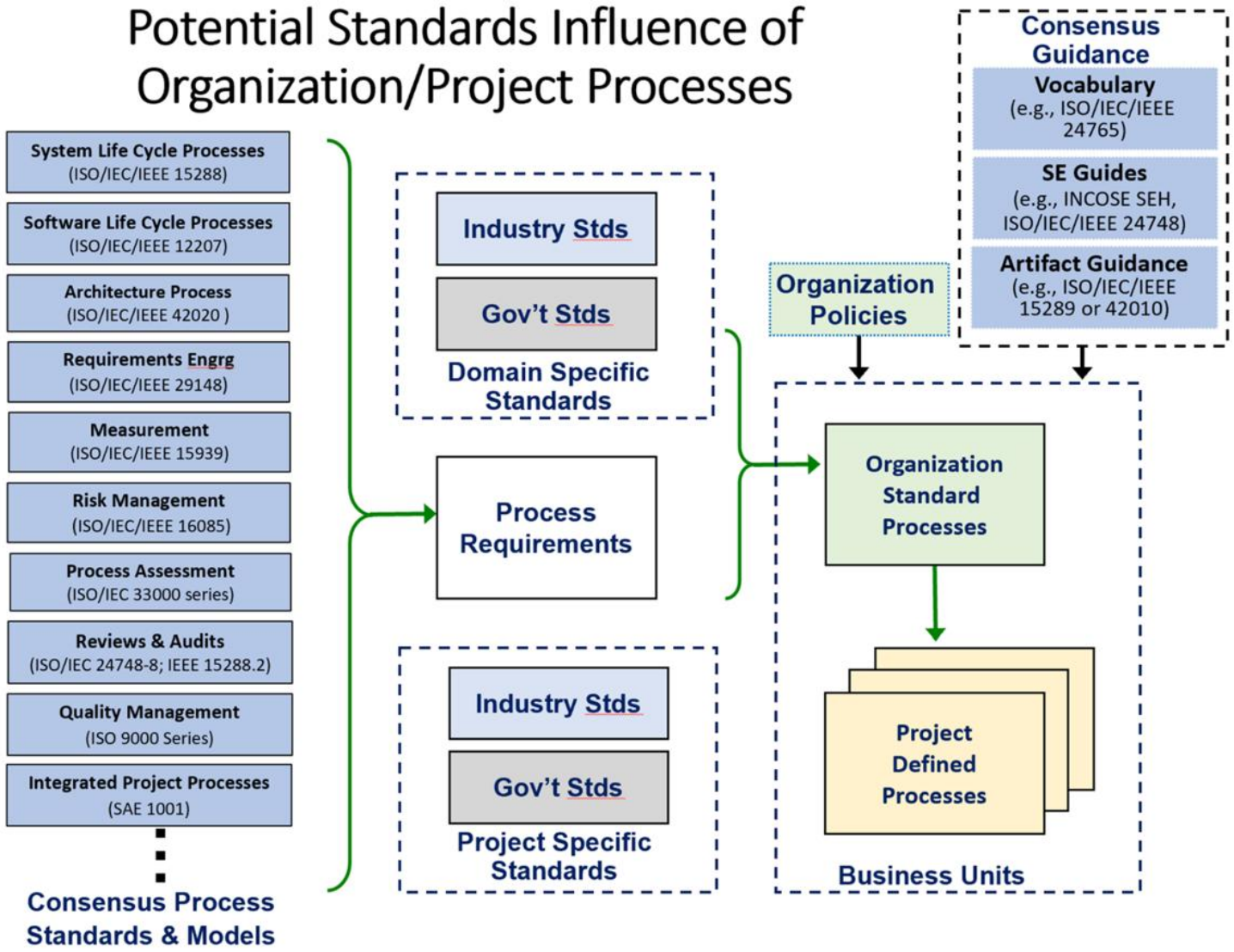
Some Military Testing Standards

- MIL-STD-810 – Environmental Engineering Considerations and Laboratory Tests
- MIL-STD-202 – Electronic and Electrical Component Parts (environmental test methods)
- MIL-STD-883 – Microcircuits environmental test methods and procedures
- MIL-STD-167 – Mechanical Vibrations of Shipboard Equipment
- MIL-STD-740 – Airborne Sound Measurement
- MIL-STD-2105D – Hazard Assessment Tests for Non-Nuclear Munitions
- MIL-STD-1576 – Electroexplosive Subsystem Safety Requirements
- MIL-STD-464 – Electromagnetic Environmental Effects (E3) Requirements
- MIL-STD-1316 – Safety Criteria for Fuzes
- MIL-STD-1901A – Safety Assessment for Non-Nuclear Munitions
- Focused on materials strength, shock, and survivability.
- MIL-STD-810 (shock/vibration portions)
- MIL-S-901D – Shock Tests, High-Impact Shipboard Machinery, Equipment
- MIL-STD-167-1A – Mechanical Vibrations of Shipboard Equipment
- MIL-STD-740-1/2 – Airborne and Structure-Borne Noise Measurements

Some Military Testing Standards

- MIL-STD-461 – Electromagnetic Interference (EMI) Control
- MIL-STD-464 – Electromagnetic Environmental Effects (E3)
- MIL-STD-704 – Aircraft Electric Power Characteristics
- MIL-STD-1399 – Shipboard Power Interface Standards
- MIL-STD-1472 – Human Engineering (ergonomics, displays, controls)
- MIL-STD-882 – System Safety Program Requirements
- MIL-S-901D – Shock Tests for Shipboard Equipment (Heavy)
- MIL-STD-167-1A – Shipboard Vibration
- MIL-STD-2036 – General Requirements for Electronic Equipment (shipboard)
- MIL-STD-1553 – Digital Time Division Command/Response Multiplex Data Bus
- MIL-STD-1760 – Aircraft Stores Electrical Interconnection System
- MIL-STD-1530 – Aircraft Structural Integrity Program

Standards for SE



https://sebokwiki.org/wiki/Application_of_Systems

ISO-IEC-IEEE 29919-2 **Software and systems engineering — Software testing- Test Processes**

- The goal of each layer is as follows:
  - Organizational test process
    - Defining a process for the creation and maintenance of organizational test specifications, such as organizational test policies, practices, processes, procedures and other assets.
  - Test management processes
    - Defining processes that cover the management of testing for a whole project or any test level
    - (e.g. system testing) or test type (e.g. performance testing) within a project (e.g. project test
    - management, system test management, performance test management).
  - The test management processes are:
    - test strategy and planning process ;
    - test monitoring and control process ;
    - test completion process
  - Dynamic test processes
    - Defining generic processes for performing dynamic testing. Dynamic testing may be performed at a particular test level (e.g. unit, integration, system, and acceptance) or for a particular test type (e.g. performance testing, security testing, and functional testing) within a project.
    - The dynamic test processes are:
      - test design and implementation process ii) test environment and data management process
      - test execution process and
      - test incident reporting process.

ISO-IEC-IEEE 29919-2 **Software and systems engineering — Software testing- Test Processes**

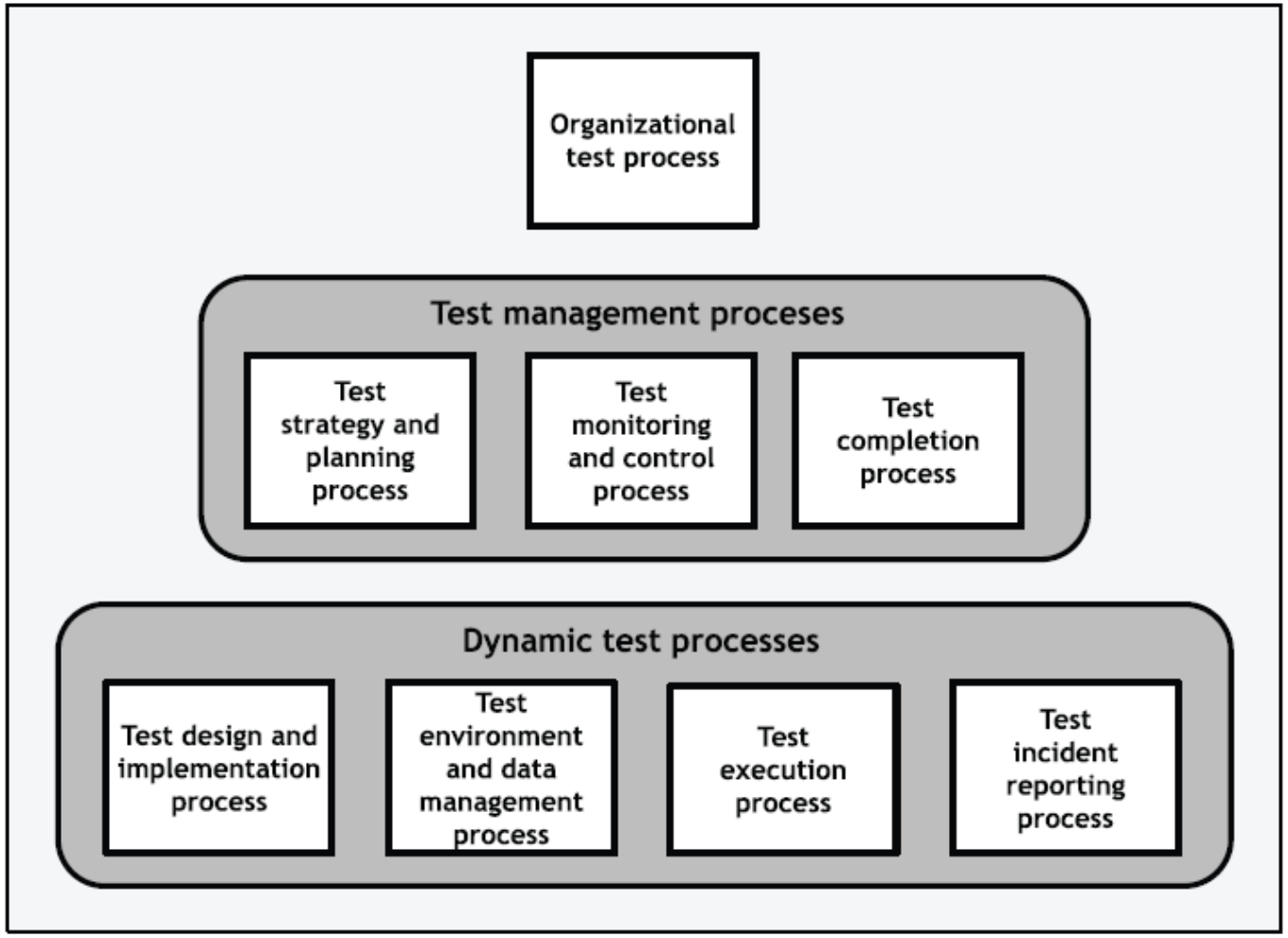


Figure 2 — The multi-layer model showing all test processes

ISO-IEC-IEEE 29919-2
Software and systems engineering —Software testing- Test Processes

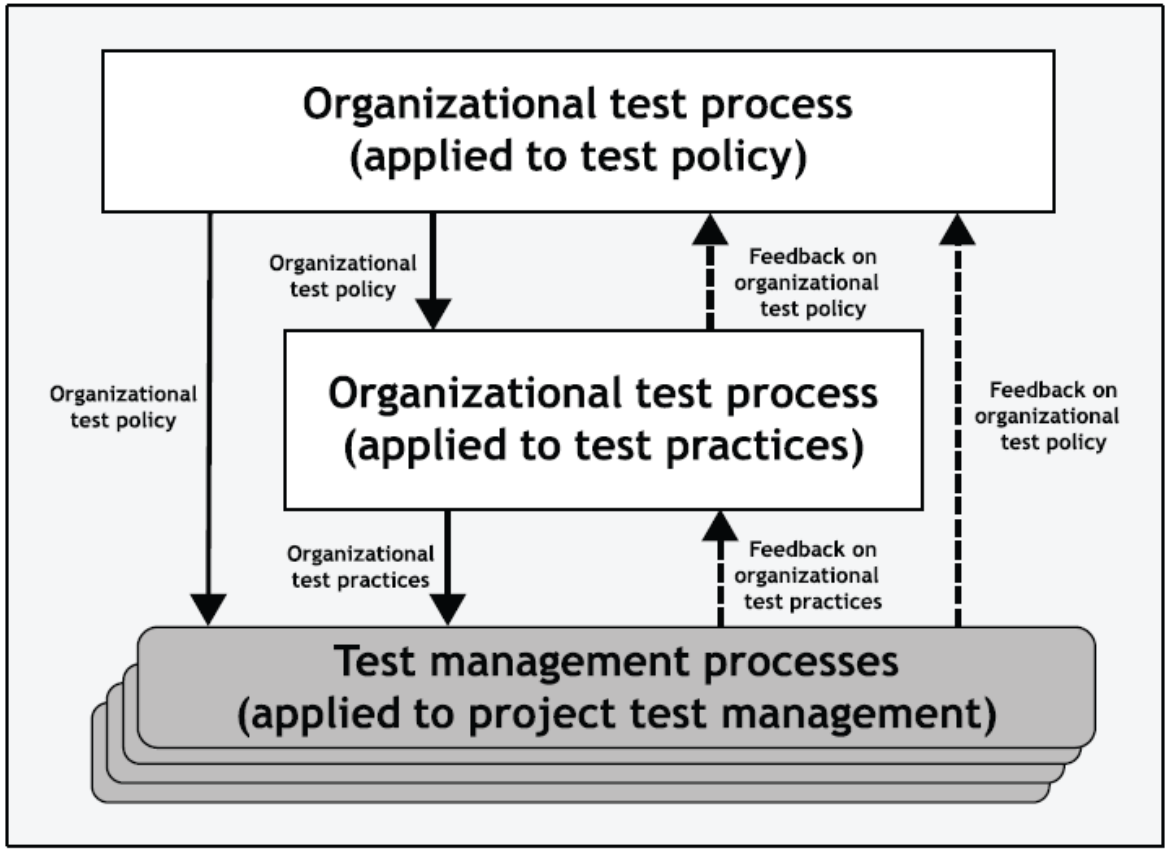


Figure 3 — Example organizational test process implementation

ISO-IEC-IEEE 29919-2 **Software and systems engineering — Software testing- Test Processes**

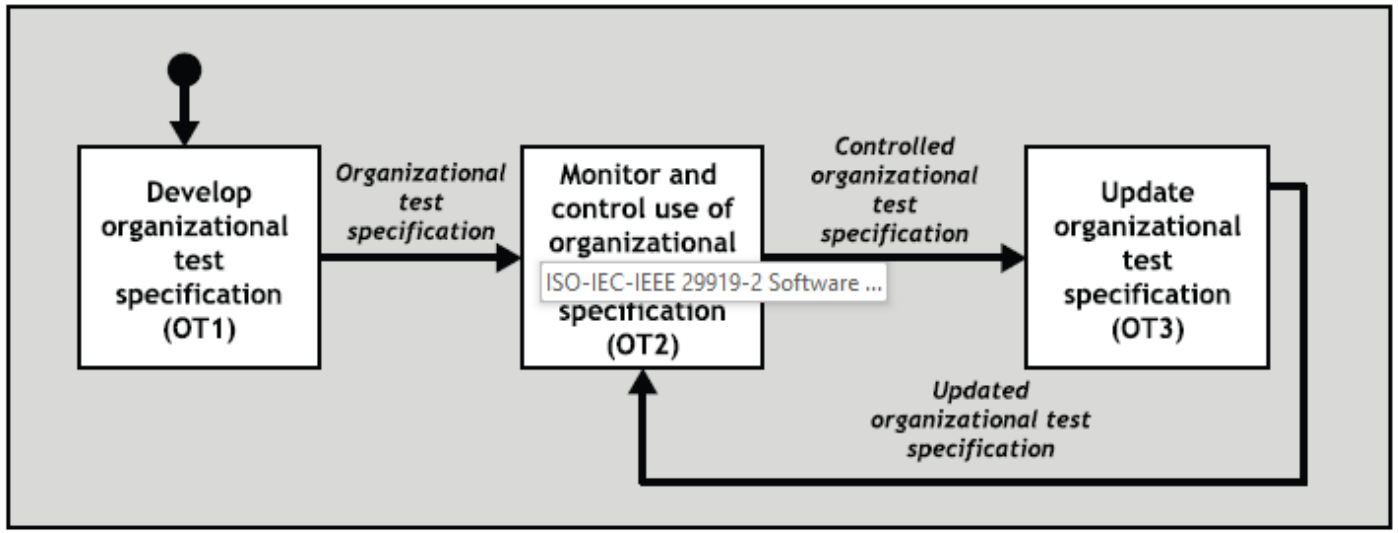


Figure 4 — Organizational test process

ISO-IEC-IEEE 29919-2

Software and systems engineering — Software testing- Test Processes

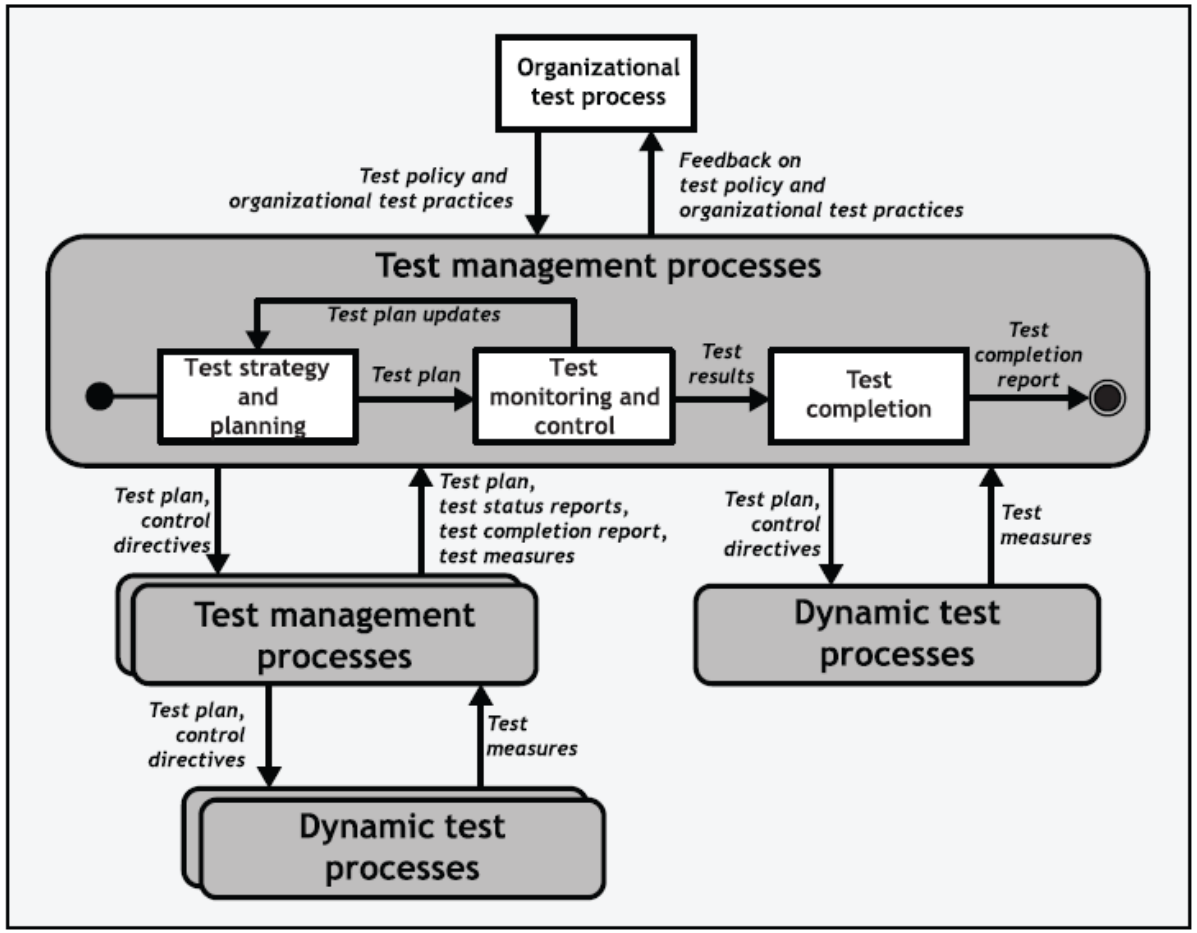


Figure 5 — Example test management process relationships

System Integrity Levels

- An integrity level (or *integrity level claim*) is a declared statement about a system's behavior or properties (e.g., reliability, safety, security), specifying its acceptable limits and the allowable level of uncertainty.

- Safety: In functional safety (e.g., IEC 61508), systems use Safety Integrity Levels (SILs) ranging from SIL 1 to SIL 4, where SIL 4 indicates the highest reliability (lowest probability of dangerous failure)

- Automotive: ISO 26262 uses Automotive Safety Integrity Levels (ASILs)—from ASIL A (lowest) to ASIL D (highest)—to quantify required risk reduction for hazards

Software Integrity Levels

Criticality	Description	Level
High	Selected function affects critical performance of the system.	4
Major	Selected function affects important system performance.	3
Moderate	Selected function affects system performance, but workaround strategies can be implemented to compensate for loss of performance.	2
Low	Selected function has noticeable effect on system performance but only creates inconvenience to the user if the function does not perform in accordance with requirements.	1

Levels/Types of Testing

Testing for software/systems

- Verification Testing
  - testing of development interim deliverables
- Unit Testing
- Integration Testing
- System Testing
  - simulate operation of the entire system
- User Acceptance Testing

Software Product Defects

- Software Design Defects
  - Designing software with incomplete or erroneous decision-making criteria
  - Failing to program as designed
  - Failure to validate data
- Data Defects
  - Incomplete data used by automated decision-making applications.

Reasons for Software Defects

- IT improperly interprets requirements
- Users specify the wrong requirements
- Requirements are incorrectly recorded
- Design specifications are incorrect
- Errors in program coding
- Data entry errors
- Testing errors
- Tests falsely detect an error
- The corrected condition causes another defect

Test Planning

Select and rank test objectives.

Identify the system development phases.

Identify the operational risks associated with the system under development.

Place risks in the matrix.

Testing Constraints

- Limited schedule and budget
- Lacking or poorly written requirements
- Changes in technology
- Limited tester skills

Source of most problems in testing

- Poor objectives
- Wrong Techniques
- Incomplete Testing

Examples

- Requirements Reviews – Done by developers and users
- Design Reviews - Developers
- Code Walkthroughs – Developers
- Code Inspections - Developers
- Unit Testing – Developers
- Integrated Testing – Developers with assist from independent test team
- System Testing – Independent Test Team
- User Acceptance Testing- users with assist from independent test team

Functional vs Structural

Functional testing is sometimes called black-box testing because no knowledge of the internal logic of the system is used to develop test cases.

Structural testing is sometimes called white-box testing because knowledge of the internal logic of the system is used to develop hypothetical test cases.

Functional Testing Requirements

- Requirements - System performs as specified.
- Regression – Verifies anything unchanged still performs correctly.
- Error Handling – Errors prevented or detected.
- Manual Support – Support process works.
- Inter-system – Data is correctly passed from system to system.
- Control Controls reduce system risk to acceptable level.
- Parallel - Old system and new system are run and the results compared

Structural Testing

Stress

Recovery

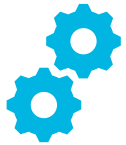
Operations

Compliance

Security

General Testing Guidelines

Verify Data Entry



Validate Logical Operations

Validate Output

Test normal conditions

Test outlier/extreme conditions

Regression testing

- *Regression testing* on the other hand is the act of repeating other tests in 'parallel' areas to ensure that the applied fix or a change of code has not introduced other errors or unexpected behavior.

Classes of Test Tools

- Defect Management Tools
- Performance/Load Testing Tools
- Manual Tools
- Traceability Tools
- Code Coverage
- Test Case Management Tools

Basics

- Test Planning – assesses the systems risks, and then develops a plan to determine if the software minimizes those risks.
- Budgeting – the resources to accomplish the test objectives.
- Scheduling – dividing the test project into accountable pieces and establishing start and completion dates.
- Staffing – obtain the testers to achieve the plan.
- Customization of the test process – determining whether or not the standard test process is adequate for a specific test project, and if not, customizing the test process for the project.

SMART Goals

- Specific
- Measureable
- Agreed Upon
- Realistic
- Time Frame

Cost Estimation Models

- Cost Models: Direct estimates of effort. They typically have a primary cost factor such as lines of code (LOC) and a number of secondary adjustment factors.
- Constraint Models: These models demonstrate the relationship over time between two or more parameters of effort, duration, or resource.
- Function Points Model: Function points (FP) measure the size in terms of the amount of functionality in a system. Function points are computed first by calculating an unadjusted function point count (UFC) for different categories.
- COCOMOII Model: an enhancement over the original COCOMO (Constructive Cost Model). The original COCOMO model is based on inputs relating to the size of the system and a number of cost drivers that affect productivity. COCOMOII is useful for a wider collection of techniques and technologies.

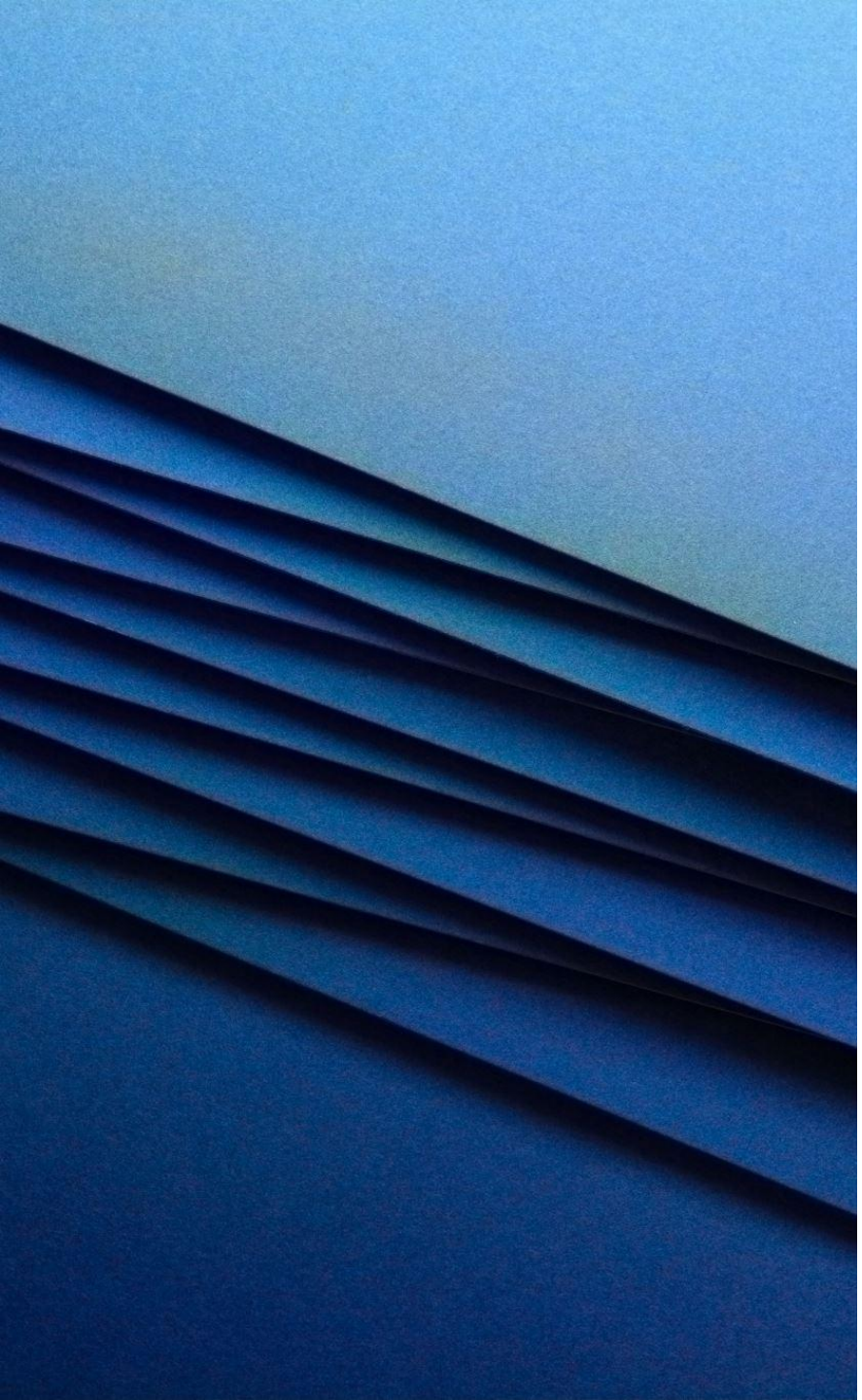
Test Team Approaches

1. Developers become the Test Team Approach

2. Independent IT Test Team Approach

3. Non-IT Test Team Approach

4. Combination Test Team Approach



Test Planning Vocabulary

Test Case: Test cases are how the testers validate that a system function meets the specifications (i.e., expected results).

Test Data: Test data is information used to build a test case.

Test Scripts: Test scripts are an online entry of test cases.

Risk: Risk is the potential loss to an organization.

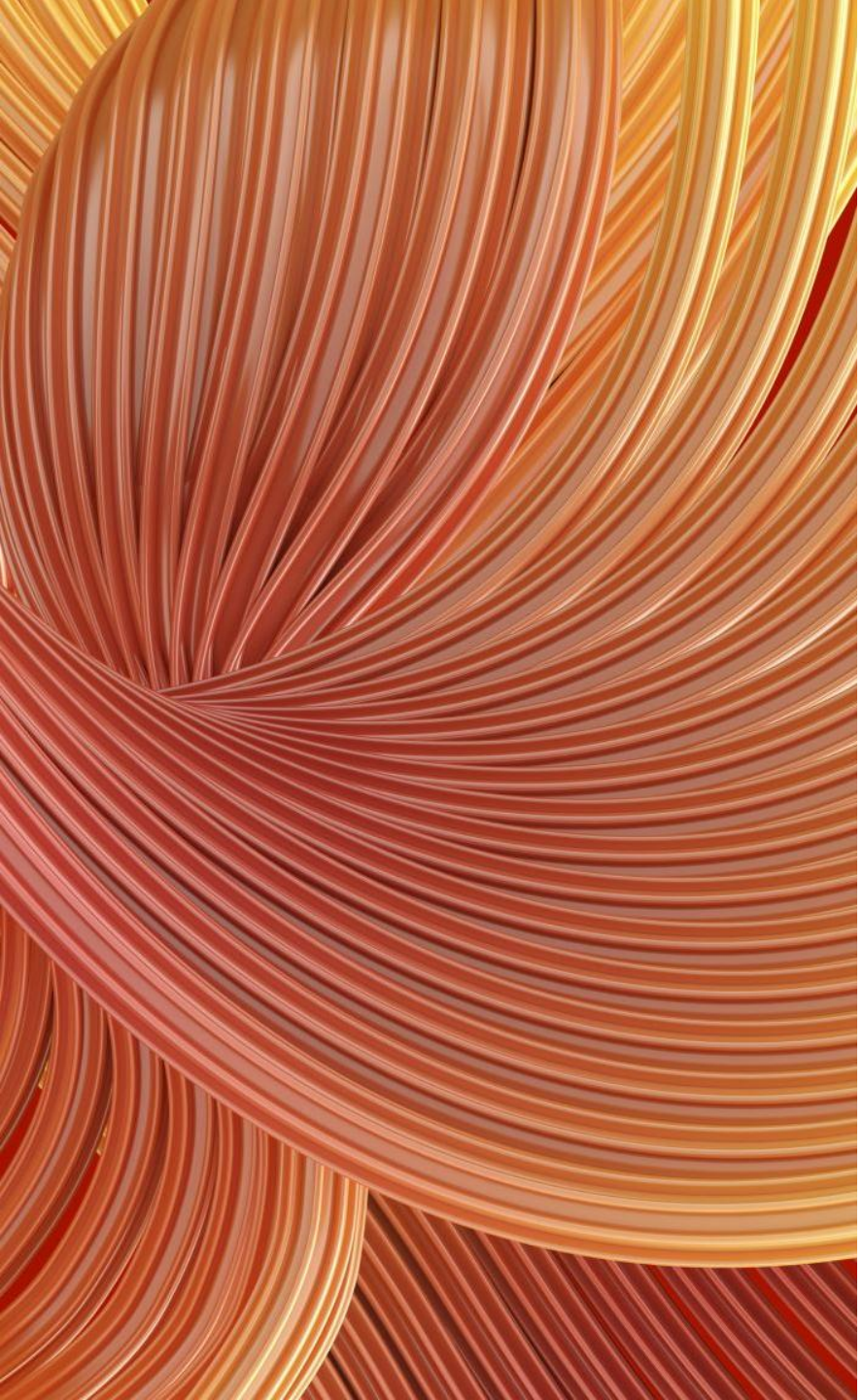
Risk Analysis: Risk analysis is an analysis of an organization’s information resources, its existing controls, and its remaining organization and computer system vulnerabilities. It combines the loss potential for each resource or combination.

Test Planning Vocabulary

Threat: A threat is something capable of exploiting vulnerability.

Vulnerability: Vulnerability is a design, implementation, or operations flaw that may be exploited by a threat.

Control: Control is anything that tends to cause the reduction of risk.



Test Planning

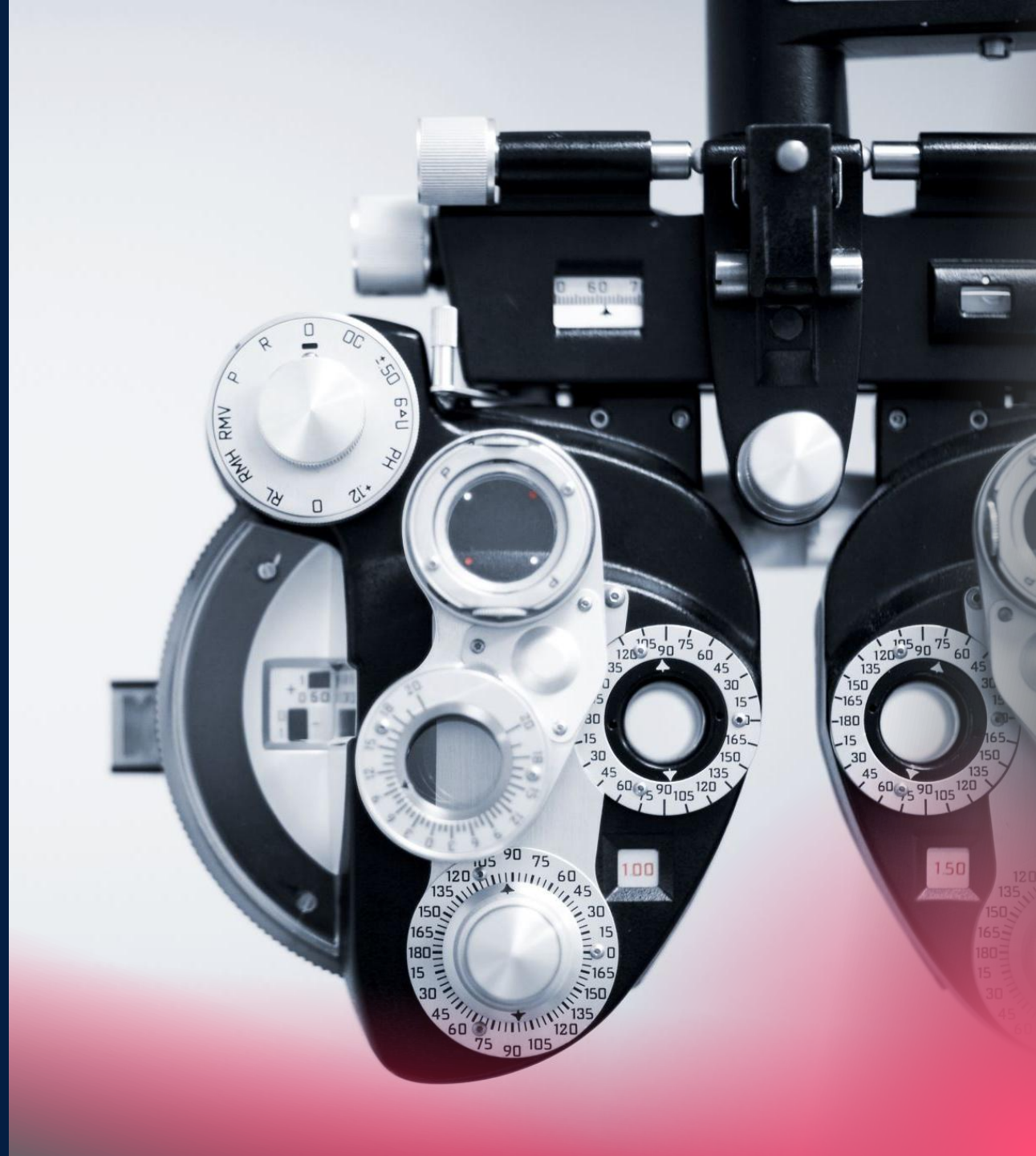
- Test Objectives
- Acceptance Criteria
- Assumptions
- People Issues
- Constraints

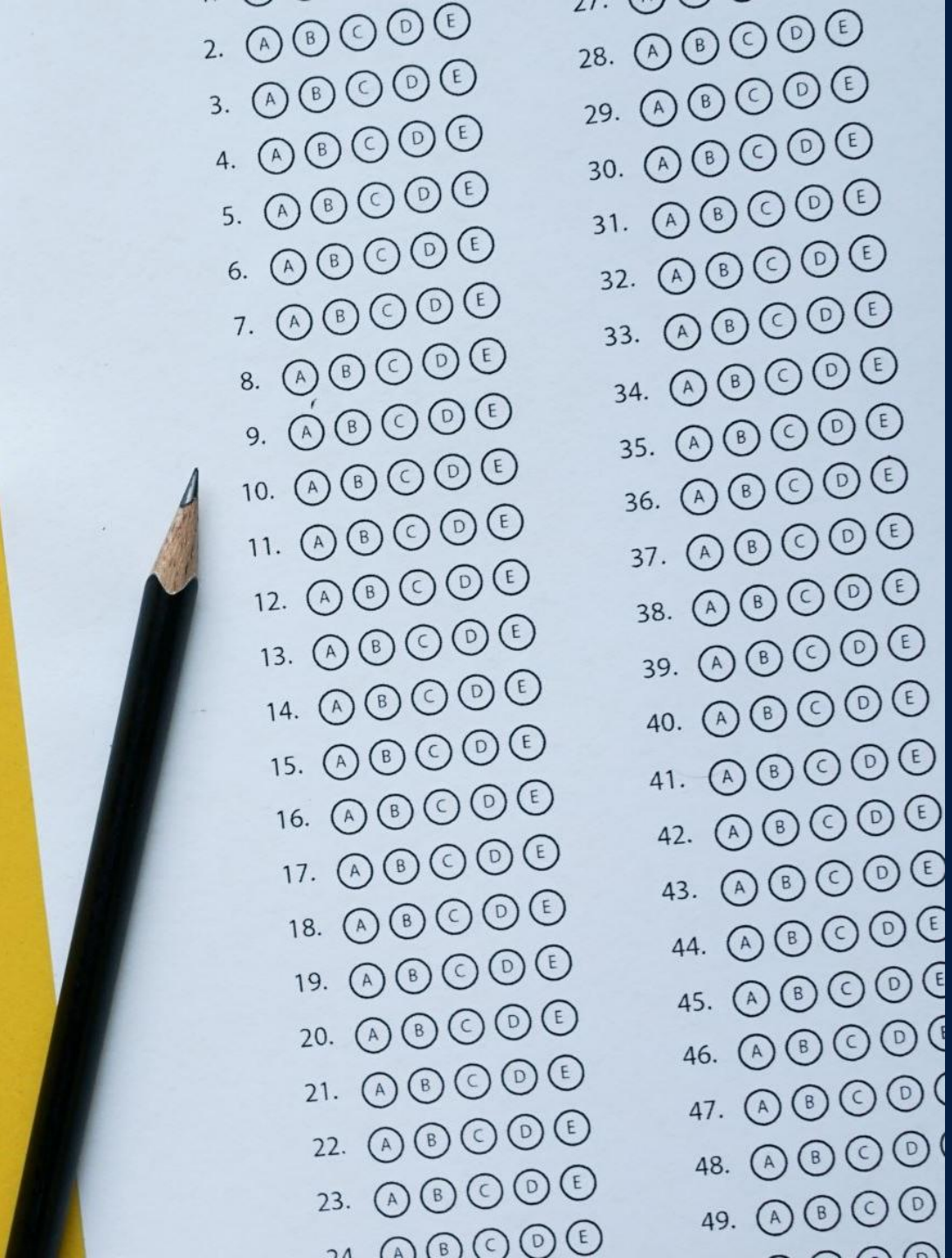
Types of Test Cases

- Functional
- Structural
- Erroneous
- Stress
- Scripts
- Use Cases

Test Case Objective

- Test Condition
- Operator Action
- Input Specifications
- Output Specifications
- Pass or Fail
- Comments



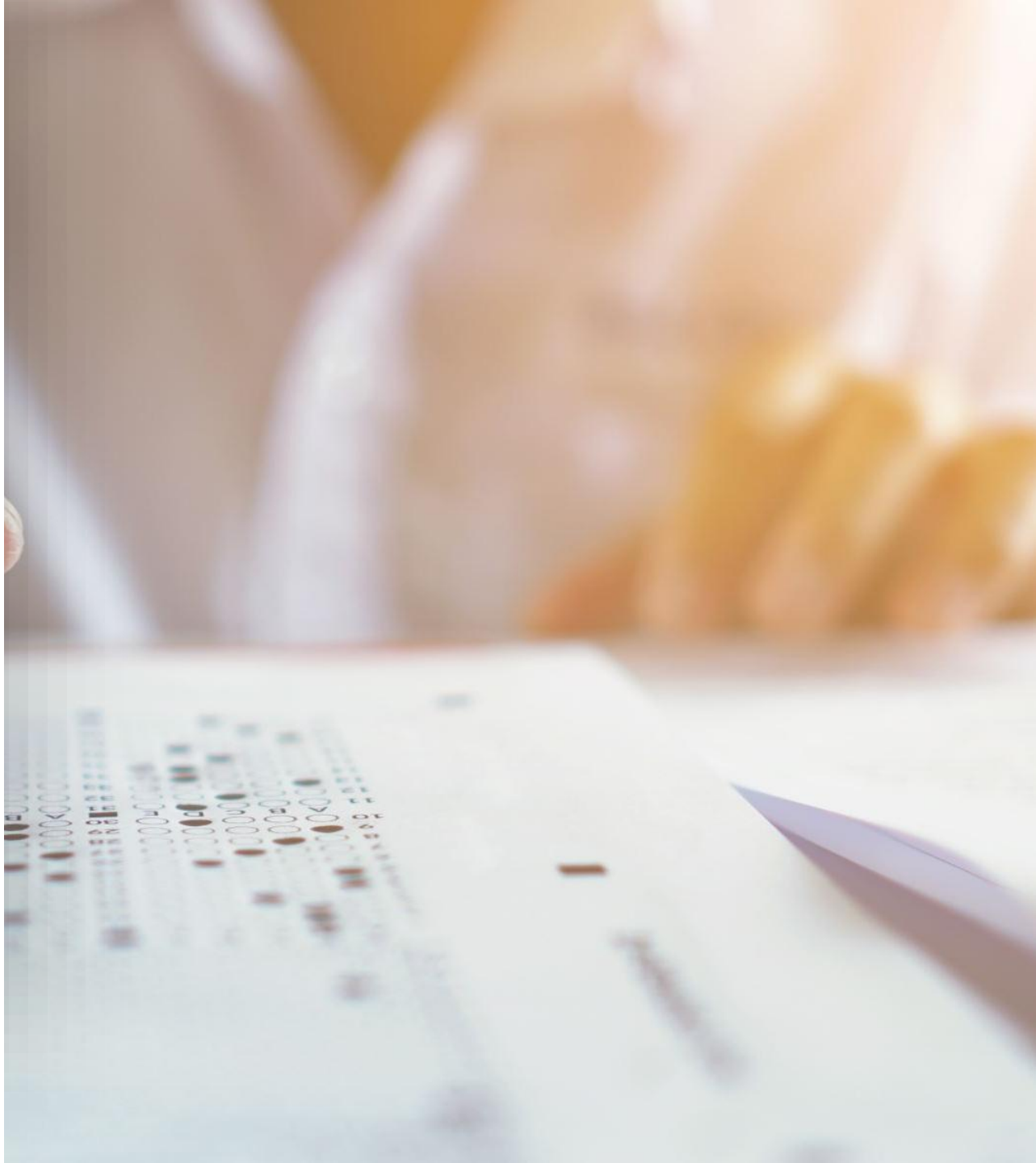


Process for building test cases

- Identify test resources.
- Identify conditions to be tested.
- Rank test conditions.
- Select conditions for testing.
- Determine correct results of processing.
- Create test cases.
- Document test conditions.
- Conduct test.
- Verify and correct.

Test Coverage

Based upon the risk, and criticality associated with the application under test, the project team should establish a coverage goal during test planning. The coverage goal defines the amount of code that must be executed by the tests for the application.



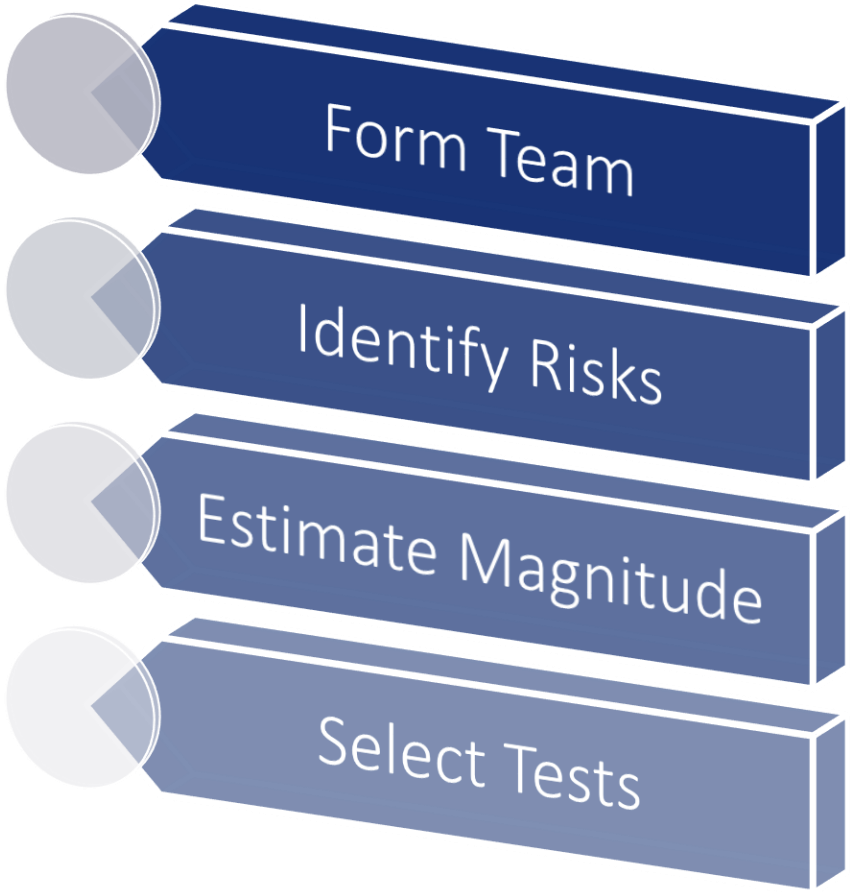
Risks Associated with Software Development

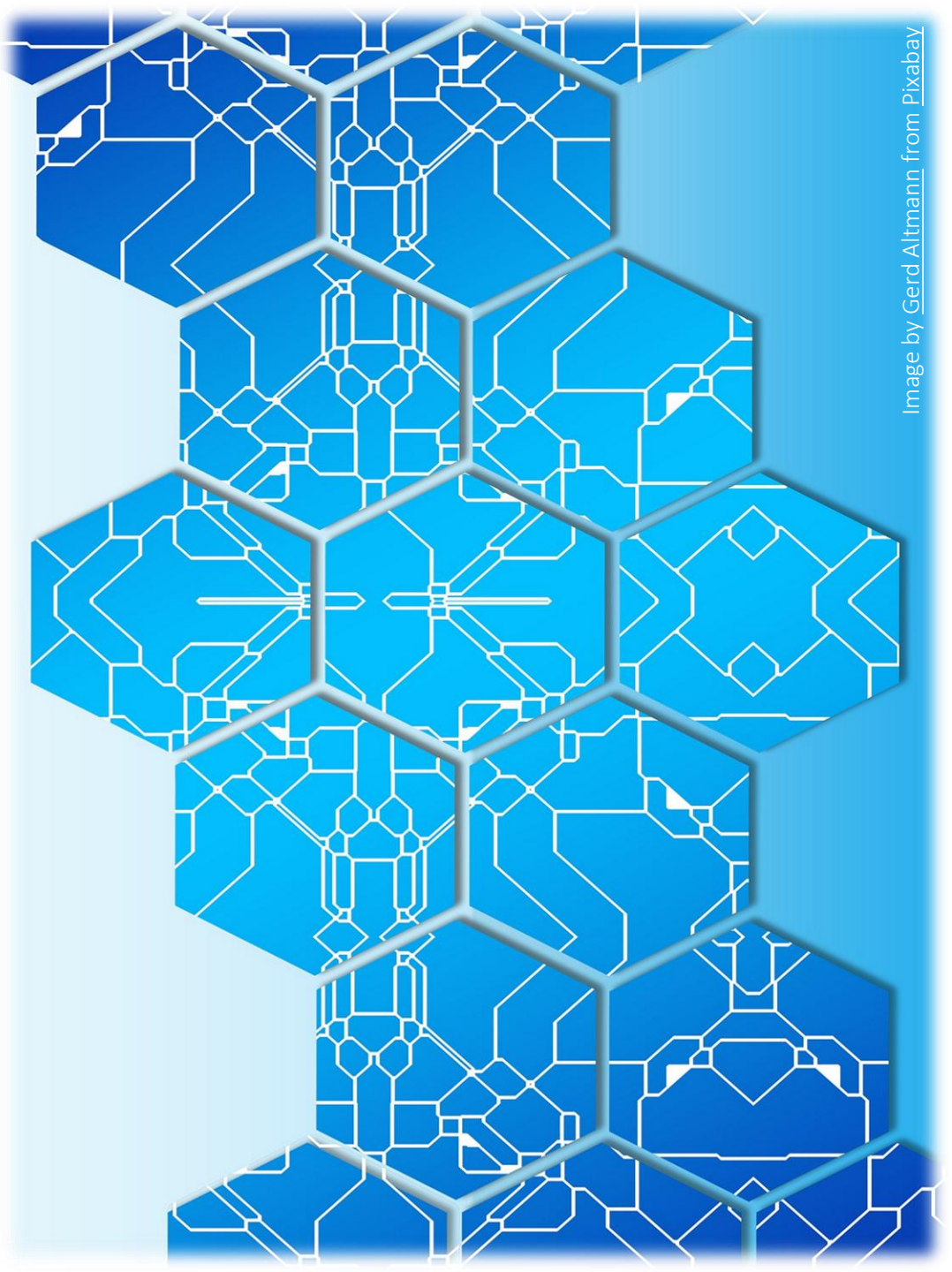
- • Improper use of technology
- • Repetition of errors
- • Cascading of errors
- • Illogical processing
- • Inability to translate user needs into technical requirements
- • Inability to control technology
- • Incorrect entry of data
- • Concentration of data
- •
- Inability to react quickly
- • Inability to substantiate processing
- • Concentration of responsibilities
- • Erroneous or falsified input data
- • Misuse by authorized end users
- • Uncontrolled system access
- • Ineffective security and privacy practices for the application
- • Procedural errors during operations
- • Program errors
- • Operating system flaws
- • Communications system failure

Risks Associated with Systems Testing

- Not Enough Training/Lack of Test Competency.
- Us versus Them Mentality
- Lack of Test Tools
- Lack of Management Understanding and Support of Testing
- Lack of Customer and User Involvement
- Not Enough Schedule or Budget for Testing
- Over Reliance on Independent Testers
- Rapid Change
- Testers are in a Lose-Lose Situation
- Having to Say “No”
- Test Environment
- New technology
- New developmental processes

Risk Analysis Process





Black-box testing

An approach to testing where the program is considered as a ‘black-box’

The program test cases are based on the system specification

Test planning can begin early in the software process

Boundary value testing

Partition system inputs and outputs into
'equivalence sets'

- If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are < 10,000, 10,000 - 99, 999 and > 10, 000

Choose test cases at the boundary of these
sets

- 00000, 09999, 10000, 99999, 10001

Functional

- Functional analysis seeks to verify, without execution, that the code faithfully implements the specification.
  - Input Testing, Syntax Checking, Equivalence Partitioning
  - Special-Value Testing: Selecting test data on the basis of features of the function to be computed is called

Structural Testing

- In structural analysis, programs are analyzed without being executed. The techniques resemble those used in compiler construction.
  - Complexity Measures
  - Data Flow Analysis
  - Symbolic Execution

Structural Testing

- Statement Testing: Statement testing requires that every statement in the program be executed.
- Branch Testing: Branch testing seeks to ensure that every branch has been executed.
- Conditional Testing: In conditional testing, each clause in every condition is forced to take on each of its possible values in combination with those of other clauses.
- Expression Testing: Expression testing requires that every expression assume a variety of values during a test in such a way that no expression can be replaced by a simpler expression and still pass the test.
- Path Testing: In path testing, data is selected to ensure that all paths of the program have been executed.

Scripts

Unit Scripting – Develop a script to test a specific unit or module.

Pseudo-concurrency Scripting – Develop scripts to test when there are two or more

users accessing the same file at the same time.

Integration Scripting – Determine that various modules can be properly linked.

Regression Scripting – Determine that the unchanged portions of systems remain unchanged when the system is changed. *Stress and Performance Scripting* – Determine whether the system will perform

correctly when it is stressed to its capacity. This

Items for Scripts

Test Item – a unique item identified of the test condition.

- Entered by – Who will enter the script.
- Sequence – The sequence in which the actions are to be entered.
- Action – The action or scripted item to be entered.
- Expected Result – The result expected from entering the action.
- Operator Instructions – What the operator is to do if the proper result is received, or if an improper result is returned.

Scripts

Test Level	Single Transaction	Multiple Transactions	Single Terminal	Multiple Terminals
Unit	X		X	
Concurrent	X			X
Integration		X	X	
Regression		X		X
Stress		X		X

White-box testing

- Sometime called structural testing or glass-box testing
- Derivation of test cases according to program structure
- Knowledge of the program is used to identify additional test cases
- Objective is to exercise all program statements (not all path combinations)

Types of structural testing

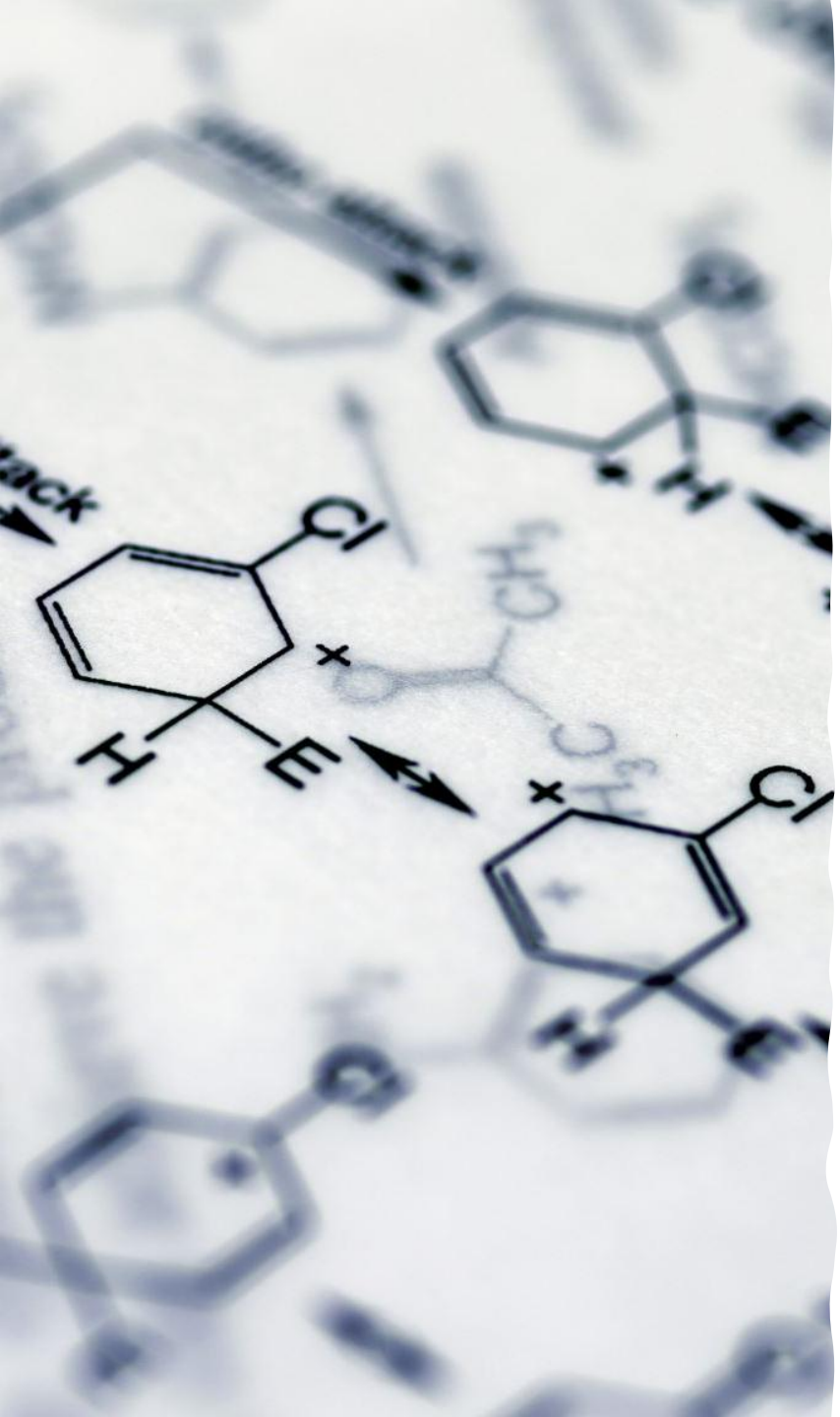
- Statement coverage -
  - Test cases which will execute every statement at least once.
  - Tools exist for help
  - No guarantee that all branches are properly tested. Loop exit?
- Branch coverage
  - All branches are tested once
- Path coverage - Restriction of type of paths:
  - Linear code sequences
  - Definition/Use checking (all definition/use paths)
  - Can locate dead code

Erroneous Testing

- Techniques that focus on assessing the presence or absence of errors in the programming process are called error-oriented. There are three broad categories of such techniques: statistical assessment, error-based testing, and fault-based testing. These are stated in order of increasing specificity of what is wrong with the program without reference to the number of remaining faults.
- Error-based testing attempts to show the absence of certain errors in the programming process. Fault-based testing attempts to show the absence of certain faults in the code. Since errors in the programming process are reflected as faults in the code, both techniques demonstrate the absence of faults.

Erroneous Testing

- Fault Estimation: Fault seeding is a statistical method used to assess the number and characteristics of the faults remaining in a program. Harlan Mills originally proposed this technique and called it error seeding.
- Input Testing: The goal of input testing is to discover input faults by ensuring that test data limits the range of undetected faults.
- Perturbation Testing: Perturbation testing attempts to decide what constitutes a sufficient set of paths to test. Faults are modeled as a vector space, and characterization theorems describe when sufficient paths have been tested to discover both computation and input errors.
- Fault-Based Testing: Fault-based testing aims at demonstrating that certain prescribed faults are not in the code. It functions well in the role of test data evaluation. Test data that does not succeed in discovering the prescribed faults is not considered adequate.

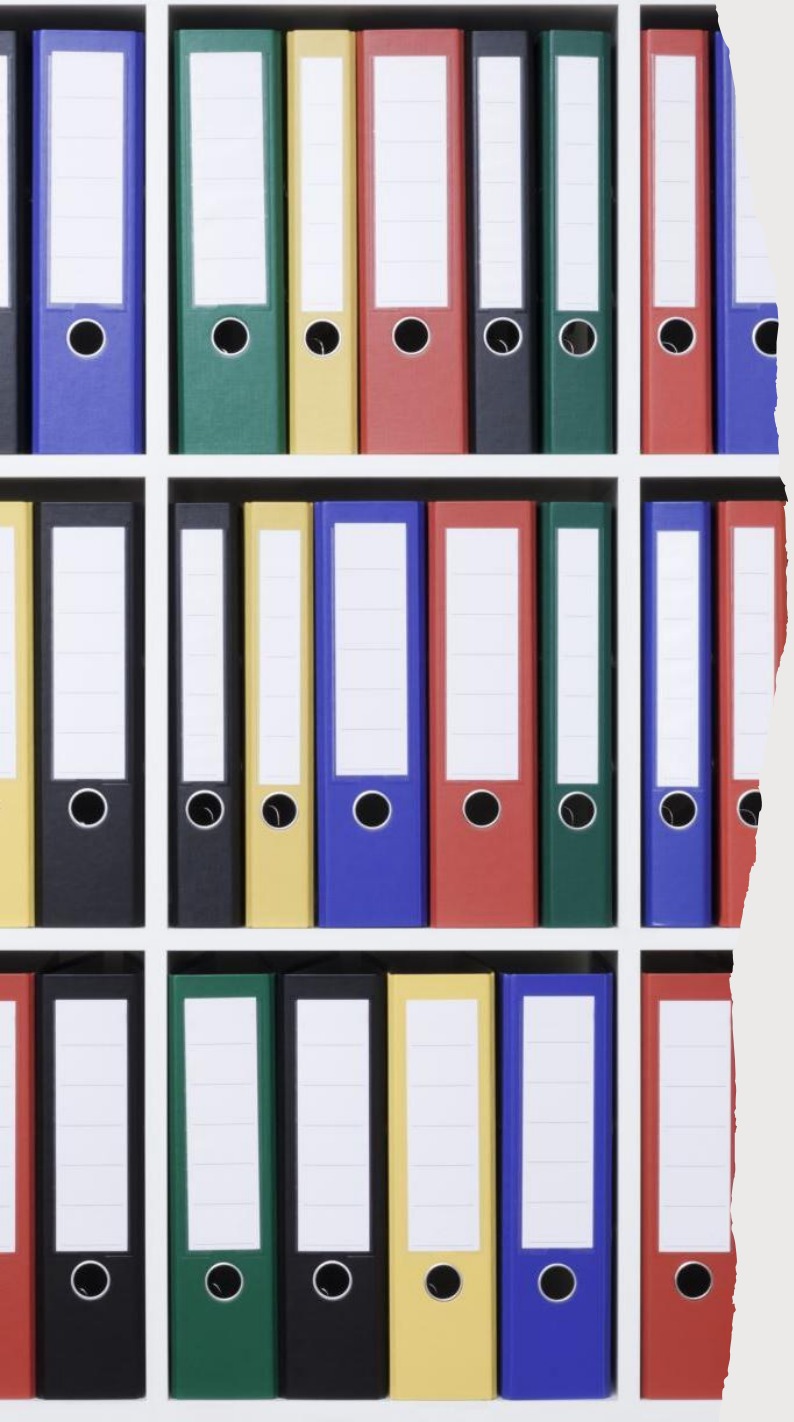


Erroneous Testing

- Local Extent, Finite Breadth: Input-output pairs of data are encoded as a comment in a procedure, as a partial specification of the function to be computed by that procedure. The procedure is then executed for each of the input values and checked for the output values.
- Global Extent, Finite Breadth: In mutation testing, test data adequacy is judged by demonstrating that interjected faults are caught. A program with interjected faults is called a mutant, and is produced by applying a mutation operator. Such an operator changes a single expression in the program to another expression, selected from a finite class of expressions.
- Local Extent, Infinite Breadth: Rules for recognizing error-sensitive data are described for each primitive language construct. Satisfaction of a rule for a given construct during testing means that all alternate forms of that construct have been distinguished.

Erroneous Testing

- Global Extent, Infinite Breadth: We can define a fault-based method based on symbolic execution that permits elimination of infinitely many faults through evidence of global failures. Symbolic faults are inserted into the code, which is then executed on real or symbolic data..



Stress Testing

The types of internal limitations that can be evaluated with volume testing include:

- Internal accumulation of information, such as tables.
- Number of line items in an event, such as the number of items that can be included within an order.
- Size of accumulation fields.
- Data-related limitations, such as leap year, decade change, switching calendar years, etc.
- Field size limitations, such as number of characters allocated for people's names.
- Number of accounting entities, such as number of locations, state/country in which operations are performed, etc.

Audit Hook

The audit hook technique involves embedding code in application systems for the examination of selected transactions. This helps an IS auditor to act before an error or an irregularity gets out of hand. An embedded audit module involves embedding specially-written software in the organization's host application system so that application systems are monitored on a selective basis. An integrated test facility is used when it is not practical to use test data, and snapshots are used when an audit trail is required.

Automated Code Comparison

An automated code comparison is the process of comparing two versions of the same program to determine whether the two correspond. It is an efficient technique because it is an automated procedure. Test data runs permit the auditor to verify the processing of preselected transactions, but provide no evidence about unexercised portions of a program. Code review is the process of reading program source code listings to determine whether the code contains potential errors or inefficient statements. A code review can be used as a means of code comparison but it is inefficient. The review of code migration procedures would not detect program changes.

System testing metrics

- Defects rates
- Errors rates
- Number of errors
- Number of errors found per person hours expended
- Measured by:
  - individual
  - module
  - during development
- Errors should be categorized by origin, type, cost

Defect Severity

Critical - The defect(s) would stop the software system from operating.

Major - The defect(s) would cause incorrect output to be produced.

Minor - The defect(s) would be a problem but would not cause improper output to be produced, such as a system documentation error.

Defect Classes

Missing

Wrong

Extra

Defect Naming

- **Name** – Requirement defect
- **Severity** – Minor
- **Type** - Procedural
- **Class** – Missing

Defect Reporting

- Data the defect uncovered
- Name of the defect
- Location of the defect
- Severity of the defect
- Type of defect
- How the defect was uncovered (i.e., test data/test script)
- Where the defect originated
- When it was corrected
- When it was entered for retest

Good Metrics

RELIABILITY

VALIDITY

EASE OF USE
AND SIMPLICITY

TIMELINESS

CALIBRATION

Test Metric Categories

- Metrics unique to test
- Complexity measurements
- Project metrics
- Size measurements
- Defect metrics
- Product measures
- Satisfaction metrics
- Productivity metrics

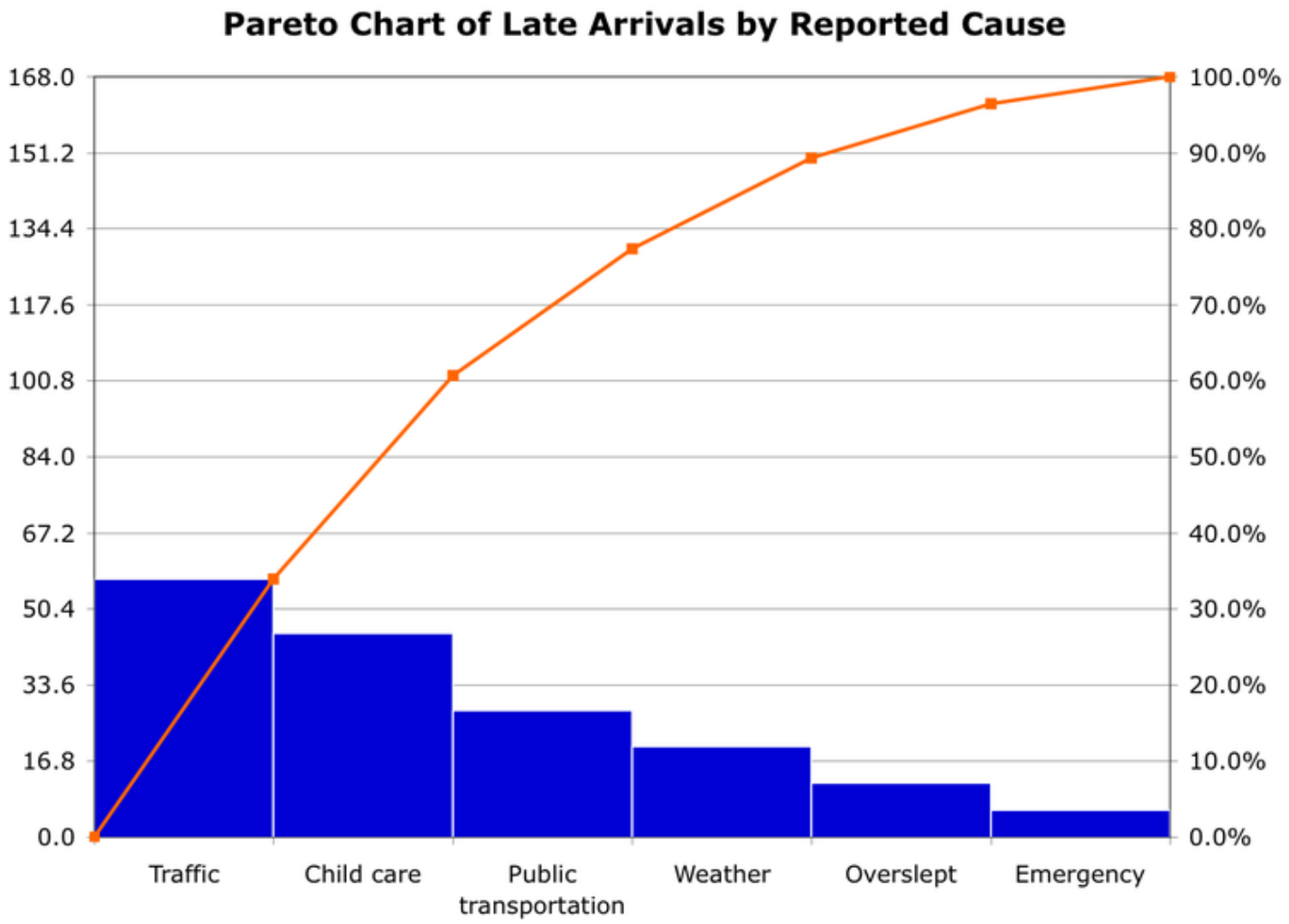
Pareto Chart

A Pareto chart is a special type of bar chart to view the causes of a problem in order of

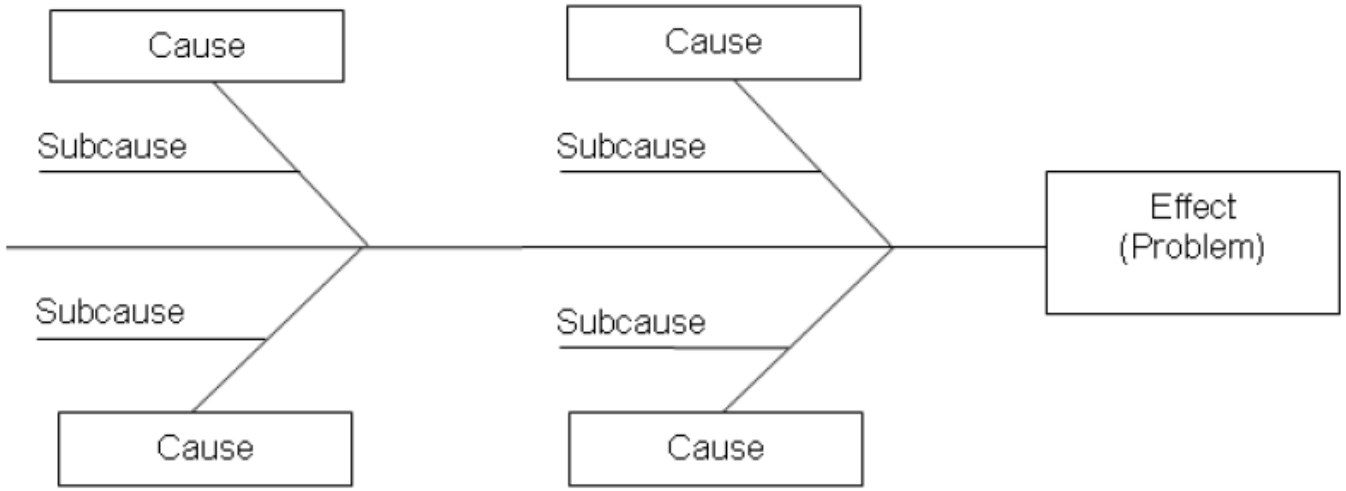
severity: largest to smallest. The Pareto chart provides an effective tool to graphically show

where significant problems and causes are in a process.

Pareto Chart



Cause and Effect Diagram



FMEA

Failure Mode and Effects Analysis (FMEA) is a systematic, structured method used to identify potential failures in a product, process, or system, evaluate their effects, and prioritize actions to reduce the likelihood or consequences of those failures.
It is one of the most widely used reliability and risk management tools in systems engineering, manufacturing, and quality assurance. The core concepts are:

- **Failure Mode** – The specific way in which a component or process could fail (e.g., short circuit, leakage, fracture).
- **Effect** – The consequence of the failure mode on the next higher level of the system, or on the overall system.
- **Cause** – The underlying reason for the failure mode (e.g., material fatigue, human error, design flaw).
- **Current Controls** – The existing measures in place to prevent or detect the failure.
- **Recommended Actions** – Additional steps to reduce occurrence, improve detection, or mitigate the effect.

FMECA

Failure Mode, Effects, and Criticality Analysis (FMECA) is a systematic, bottom-up reliability and risk assessment method used in engineering to identify potential failure modes of a system, evaluate their effects, and prioritize them based on criticality—a combination of severity and likelihood.

- Process Steps
- Define the scope and system breakdown
  - Decompose the system into subsystems, assemblies, and components.
- Identify failure modes
  - List all the possible ways each component can fail (e.g., short circuit, mechanical jam, corrosion).
- Determine the effects
  - For each failure mode, describe the **local effect** (on the component), **next-higher-level effect** (on the subsystem), and **end effect** (on the whole system).
- Assess severity
  - Rank how serious each effect is (e.g., from negligible to catastrophic).
- Estimate occurrence probability
  - Rank or quantify how likely each failure is to occur, based on data or estimates.
- Evaluate detection (optional in some approaches)
  - Assess how likely it is that the failure will be detected before it causes harm.

FMECA

Perform criticality analysis

- Calculate a **Criticality Index** (quantitative) or assign a criticality category (qualitative).
- Often based on:

$$C_m = \beta \times \alpha \times \lambda_p \times t$$

- Where:
- β = conditional probability the failure will result in the defined severity
- α = proportion of failures of that mode
- λ = part failure rate
- t = operating time

Failure Modes, Effects, and Diagnostic Analysis (FMEDA)

Failure Modes, Effects, and Diagnostic Analysis (FMEDA) is an extension of Failure Modes and Effects Analysis (FMEA) that adds a quantitative diagnostic coverage assessment—making it especially useful in functional safety and reliability-critical systems. It's widely used in industries like automotive (ISO 26262), process safety (IEC 61508), aerospace, and medical devices to help meet Safety Integrity Level (SIL) or Automotive Safety Integrity Level (ASIL) targets. FMEDA goes beyond standard FMEA by:

- Identifying failure modes and their effects (like FMEA).
- Quantifying how each failure mode is detected by built-in diagnostics or monitoring.
- Calculating metrics such as diagnostic coverage (DC), safe failure fraction (SFF), and failure rates.
- Providing input for probabilistic safety analysis and hardware fault tolerance (HFT) assessments.

FMEDA vs FMEA

Aspect	FMEA	FMEDA
Focus	Qualitative analysis of potential failures and effects	Quantitative analysis including diagnostic detection and coverage
Outputs	Failure modes, effects, risk priority	Failure modes, effects, detection rates, failure rates, safety metrics
Application	Reliability improvement & risk prioritization	Functional safety compliance & quantitative reliability targets

Beta-factor model

The Beta-factor model is a quantitative method used in Common Cause Analysis (CCA) to account for dependent failures in redundant systems—specifically, common cause failures (CCFs) that affect multiple components simultaneously.

- For two identical redundant components:
- λ = total failure rate of a single component
- β = fraction of failures due to a *common cause*
- $(1 - \beta)$ = fraction of failures that are truly independent
- Then:
- CCF failure rate=$\beta \cdot \lambda$\
- Independent failure rate=$(1-\beta) \cdot \lambda$

- In a 2-out-of-2 system (both components must work), the total probability of system failure includes:
- Common cause failures (both fail together via the same cause)
- Independent sequential failures (one fails, then the other before repair

Beta-factor model

- Interpretation of β
- $\beta = 0$ → all failures are independent (ideal redundancy)
- $\beta = 1$ → all failures are due to common causes (redundancy is useless)
- Typical real-world values: **0.01–0.3** depending on system type and environment.

RAMS

- Reliability, Availability, Maintainability, and Safety (RAMS) is a collective term used in systems engineering to describe the key dependability attributes of a system.
It’s widely applied in industries such as aerospace, defense, rail, automotive, energy, and medical devices to ensure that systems perform as intended throughout their lifecycle.
- Measured as a probability (0 to 1) or Mean Time Between Failures (MTBF).Focuses on failure prevention.Influenced by design quality, component selection, manufacturing processes, and operating environment.

CCA

Common Cause Analysis (CCA) is a systems engineering and reliability method used to identify and manage situations where multiple components fail due to the same underlying cause. The key idea is that some failures are not independent—if a single cause can trigger several component failures at once, it can undermine redundancy and safety measures.
Purpose

- Detect potential single points of vulnerability that can defeat fault-tolerant designs.
- Identify causes that can produce simultaneous or correlated failures in otherwise independent elements.
- Ensure that redundancy actually improves reliability rather than giving a false sense of security.

CCA

Common CCA Methods

- Qualitative:
  - *Cause–effect mapping*: Brainstorming and mapping shared failure triggers.
  - *Checklists*: Based on known industry CCF patterns.
  - *FMEA/FMECA extensions*: Adding “common cause” columns to standard analyses.
- Quantitative:
  - *Beta-factor model*: Assumes a proportion (β) of total component failures are due to common causes.
  - *Multiple Greek letter models*: More granular modeling of partial dependencies.

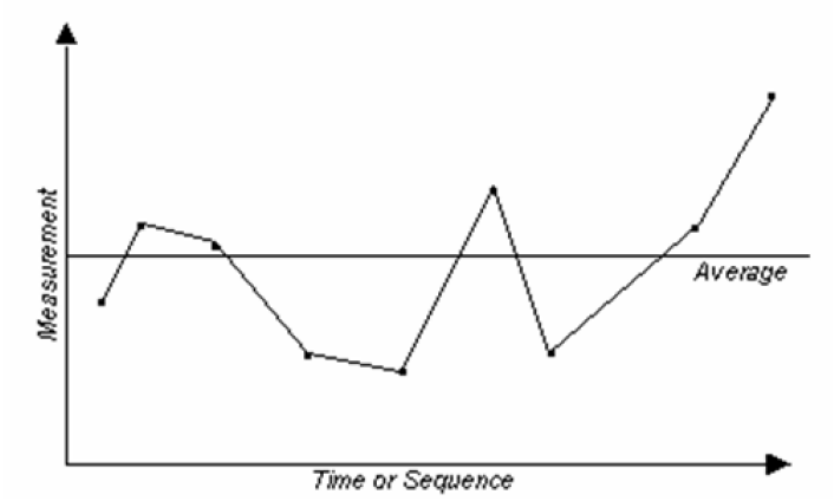
Check Sheet

A check sheet is a technique or tool to record the number of occurrences over a specified interval of time; a data sample to determine the frequency of an event. The recording of data, survey, or sample is to support or validate objectively the significance of the event.

Run Chart

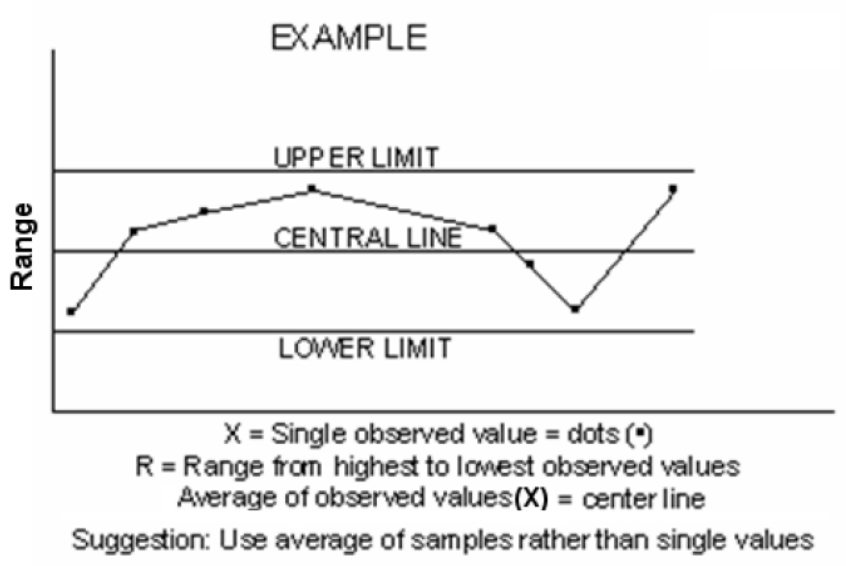
A run chart is a graph of data (observation) in chronological order displaying shifts or trends

in the central tendency (average).



Control Chart

The intent of a control chart is to monitor the variation of a statistically stable process where activities are repetitive. Two types of variation are being observed: 1) common, or random; and, 2) special or unique events.



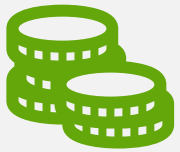
Other Tools

- Histogram
- Pie Chart
- Bar Graph
- Scatter Plot
- Regression Analysis
- Multivariate Analysis

Budgeting Techniques

Top-Down Estimation

Expert Judgment



Bottom-Up Estimation

Cost Estimation Models

- Cost Models: Direct estimates of effort. They typically have a primary cost factor such as lines of code (LOC) and a number of secondary adjustment factors.
- Constraint Models: These models demonstrate the relationship over time between two or more parameters of effort, duration, or resource.
- Function Points Model: Function points (FP) measure the size in terms of the amount of functionality in a system. Function points are computed first by calculating an unadjusted function point count (UFC) for different categories.
- COCOMOII Model: an enhancement over the original COCOMO (Constructive Cost Model). The original COCOMO model is based on inputs relating to the size of the system and a number of cost drivers that affect productivity. COCOMOII is useful for a wider collection of techniques and technologies.

Cost Estimation Models

- Cost Models: Direct estimates of effort. They typically have a primary cost factor such as lines of code (LOC) and a number of secondary adjustment factors.
- Constraint Models: These models demonstrate the relationship over time between two or more parameters of effort, duration, or resource.
- Function Points Model: Function points (FP) measure the size in terms of the amount of functionality in a system. Function points are computed first by calculating an unadjusted function point count (UFC) for different categories.
- COCOMOII Model: an enhancement over the original COCOMO (Constructive Cost Model). The original COCOMO model is based on inputs relating to the size of the system and a number of cost drivers that affect productivity. COCOMOII is useful for a wider collection of techniques and technologies.

Scheduling

- What tasks will be done?
- Who will do them?
- When will they do them?

Test Team Approaches

1. Developers become the Test Team Approach

2. Independent IT Test Team Approach

3. Non-IT Test Team Approach

4. Combination Test Team Approach

Customization of the Test Process

- Release cycle schedules
- System development methodology
- User schedules
- Project status reporting
- Interfacing with other projects
- Interfacing with enterprise-wide databases
- Assuring the same naming conventions/data definitions are used for testing as for

other projects

Test Supervision

- Communication skills
- Negotiation and complaint resolution skills
- Motivation, Mentoring, and Recognition

Test Leadership

- Chairing meetings
- Team building
- Quality Management Organizational Structure
- Code of ethics

Traditional v Quality Management

Traditional Management Philosophy	Quality Management Philosophy
Controls each result	Use the process
Who made the error?	What allowed the error?
Correct the error	Reduce variation and prevent the error
Employees are the problem	Refine the process
Management accountable to their manager	Management accountable to the customer
Competition between organizations	Teamwork
Motivation from fear of failure	Motivation from within (self)
Management of outputs (results)—focusing on detection of defects	Management of process inputs—methods or sources of variation that focus on preventing defects
Fire fighting	Continuous process improvement
Accomplishment from meeting quotas, the monthly or quarterly bottom line	Accomplishment from long-term impact of improving processes

User Acceptance Principles

- Acceptance decisions occur at pre-specified times when processes, support tools, interim documentation, segments of the software, and finally the total system system must meet predefined criteria for acceptance. Subsequent changes to the systems may affect previously accepted elements.

User Acceptance Accountability

Accountability for software acceptance belongs to the customer or user of the software, whose responsibilities are:

- Ensure user involvement in developing system requirements and acceptance criteria
- Identify interim and final products for acceptance, their acceptance criteria, and schedule
- Plan how and by whom each acceptance activity will be performed
- Plan resources for providing information on which to base acceptance decisions
- Schedule adequate time for buyer staff to receive and examine products and evaluations prior to acceptance review
- Prepare the Acceptance Plan
- Respond to the analyses of project entities before accepting or rejecting
- Approve the various interim software products against quantified criteria at

interim points

- Perform the final acceptance activities, including formal acceptance testing, at

delivery

- Make an acceptance decision for each product

FIT

Acceptance testing is designed to determine whether the software is fit for use. The concept of fit for use is important in both design and testing.

1. Data The reliability, timeliness, consistency, and usefulness of the data included in the

automated application.

2. People People should have the skills, training, aptitude, and desire to properly use and interact

with the automated application.

3. Structure The structure is the proper development of application systems to optimize technology

and satisfy requirements.

4. Rules The rules are the procedures to follow in processing the data.

Users Role

Defining acceptance criteria in a testable format

- Providing the use cases that will be used in acceptance testing
- Training user personnel in using the new system
- Providing the necessary resources, primarily user staff personnel, for acceptance testing
- Comparing the actual acceptance testing results with the desired acceptance testing results (NOTE: This may be performed using testing software)
- Making decisions as to whether additional work is needed prior to placing the system in operation, whether the software can be placed in operation with additional work to

be done, or whether the system is fully acceptable and can be placed into production as is

User Acceptance Principles

Acceptance decisions occur at pre-specified times when processes, support tools, interim documentation, segments of the software, and finally the total system must meet predefined criteria for acceptance. Subsequent changes to the software may affect previously

accepted elements.

User Acceptance Accountability

Accountability for system acceptance belongs to the customer or user of the software, whose responsibilities are:

- Ensure user involvement in developing system requirements and acceptance criteria
- Identify interim and final products for acceptance, their acceptance criteria, and schedule
- Plan how and by whom each acceptance activity will be performed
- Plan resources for providing information on which to base acceptance decisions
- Schedule adequate time for buyer staff to receive and examine products and evaluations prior to acceptance review
- Prepare the Acceptance Plan
- Respond to the analyses of project entities before accepting or rejecting
- Approve the various interim system products against quantified criteria at interim points
- Perform the final acceptance activities, including formal acceptance testing, at delivery
- Make an acceptance decision for each product

Users Role

Defining acceptance criteria in a testable format

- Providing the use cases that will be used in acceptance testing
- Training user personnel in using the new software system
- Providing the necessary resources, primarily user staff personnel, for acceptance testing
- Comparing the actual acceptance testing results with the desired acceptance testing results (NOTE: This may be performed using testing system)
- Making decisions as to whether additional work is needed prior to placing the software in operation, whether the system can be placed in operation with additional work to be done, or whether the software is fully acceptable and can be placed into production as is

Acceptance Test Planning

- Acceptance Criteria
- Acceptance Test Plan
- Use Case Test Data

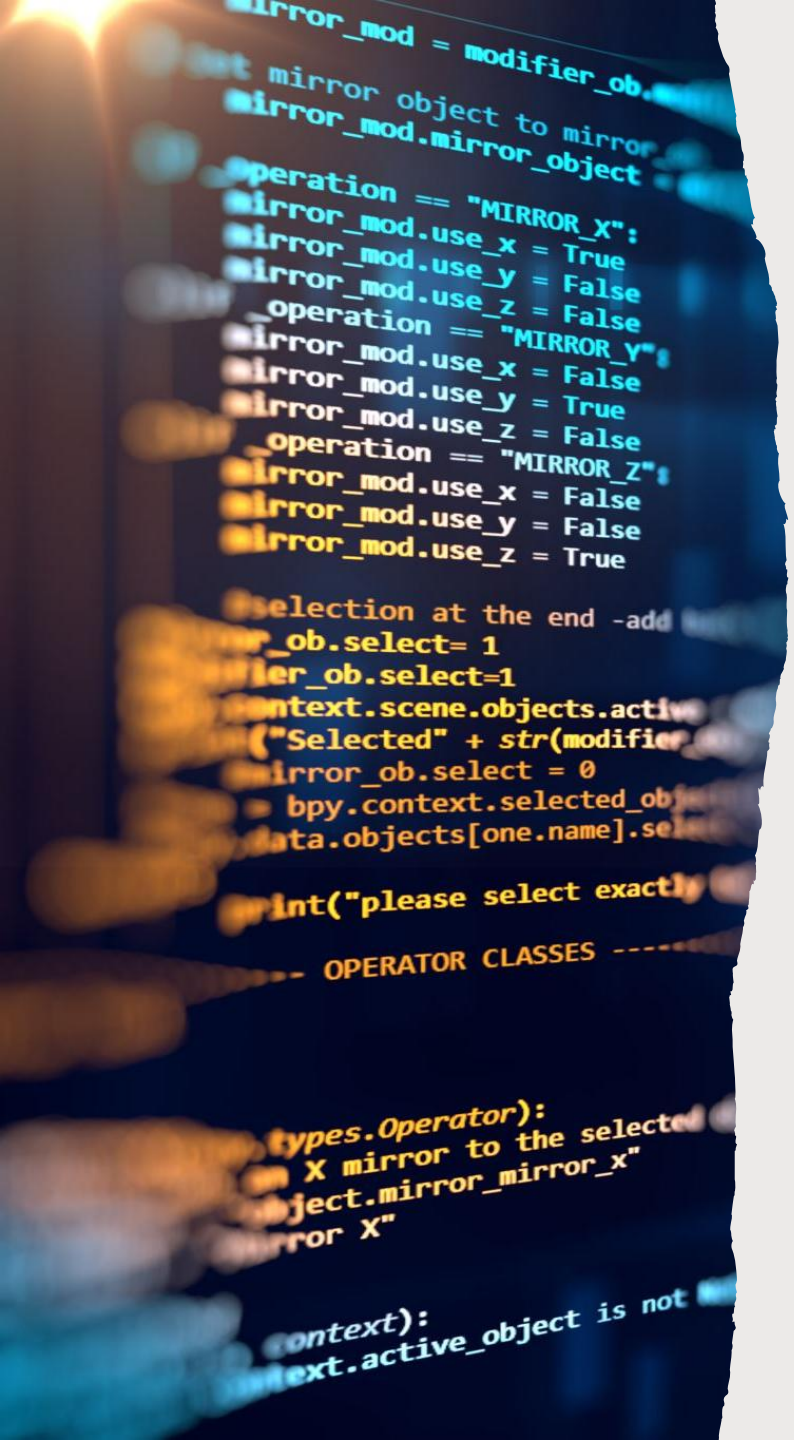
Example of Acceptance Criteria

Criteria	Action
Hardware/Software Project	The name of the project being acceptance-tested. This is the name the user or customer calls the project.
Number	A sequential number identifying acceptance criteria.
Acceptance Requirement	A user requirement that will be used to determine whether the corrected hardware/software is acceptable.
Critical / Non -Critical	Indicate whether the acceptance requirement is critical, meaning that it must be met, or non-critical, meaning that it is desirable but not essential.
Test Result	Indicates after acceptance testing whether the requirement is acceptable or not acceptable, meaning that the project is rejected because it does not meet the requirement.
Comments	Clarify the criticality of the requirement; or indicate the meaning of the test result rejection. For example: The software cannot be run; or management will make a judgment after acceptance testing as to whether the project can be run.

Table from CBK

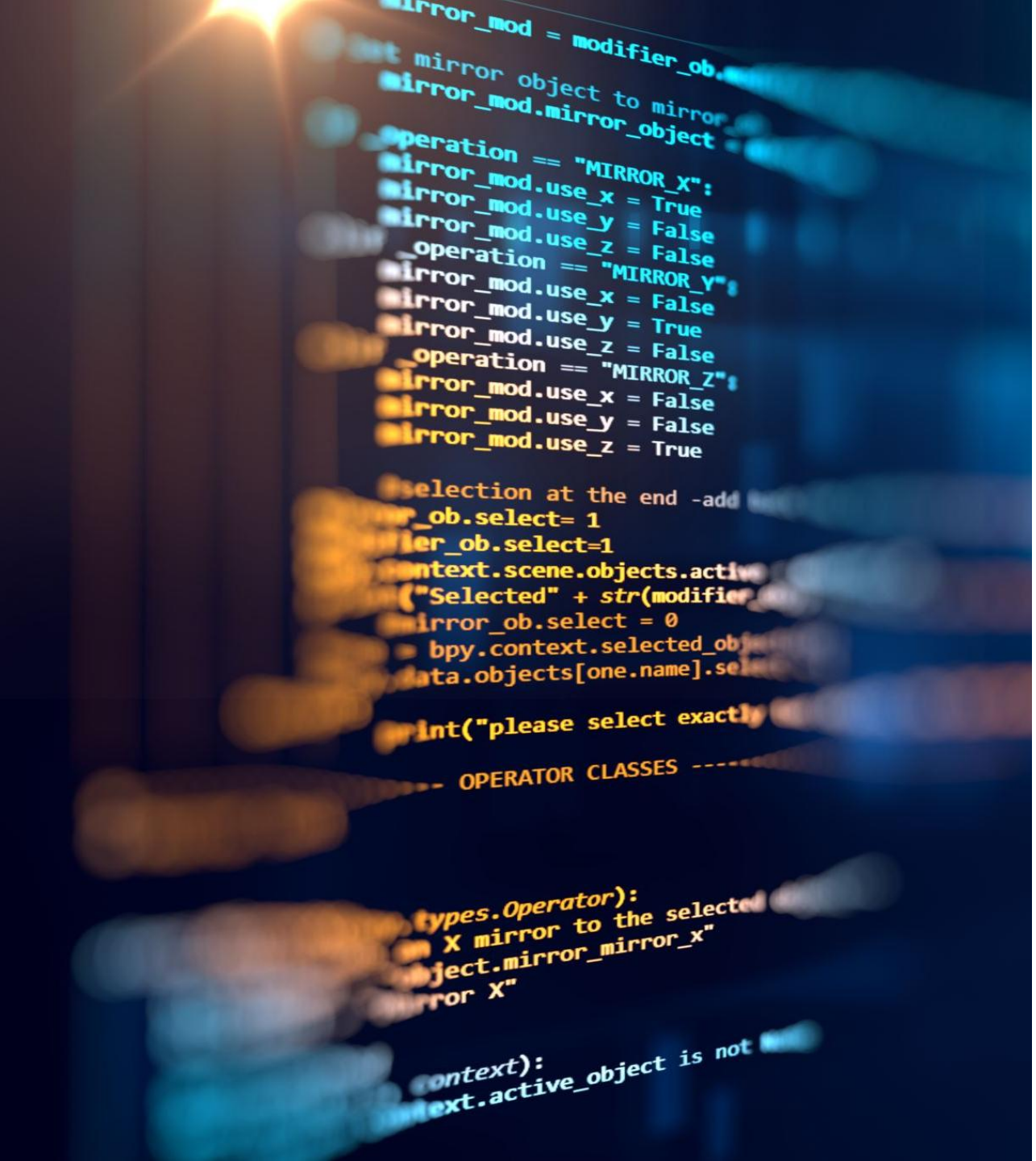
System Configuration Management

System Configuration Management (CM) is a key component of the infrastructure for any organization. The ability to maintain control over the changes made to all project artifacts is critical to the success of a project. The more complex an application is, the more important it is to implement change to both the application and its supporting artifacts in a controlled manner.



Artifacts in SCM

- Source code
- Requirements
- Analysis models
- Design models
- Test cases and procedures
- Automated test scripts
- User documentation, including manuals and online Help
- Hardware and software configuration settings
- Other artifacts as needed



Cleanroom process teams

developing
and maintaining the system specification.

- Development team. Responsible for developing and verifying the software. The software is NOT executed or even compiled during this process.
- Certification team. Responsible for developing
a set of statistical tests to exercise the software
after development. Reliability growth models used to determine when reliability is acceptable.

Component testing

- Component or unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Components may be:
  - Individual functions or methods within an object;
  - Object classes with several attributes and methods;
  - Composite components with

Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Particularly important for object-oriented development as objects are defined by their interfaces.