Software Testing, Verification Validation and Quality Assurance

UNIT IV

NON-FUNCTIONAL TESTING

- Reliability testing
    - system performs its intended function without failure for a specific period under specific conditions.
    - Measures how long the system can run without crashing.
    - Associated with Mean Time Between Failures (MTBF).
    - A highly reliable system fails rarely.
- Availability testing
    - percentage of time a system is operational and accessible when required.
    - uptime vs total time
    - A highly available system is almost always online.
    - Availability = MTBF / (MTBF + MTTR) where MTTR = MTTR (Mean Time to Repair)

Reliability vs Availability

- Reliability = fewer failures
- Availability = less downtime
- Reliable but not Available
  - A car that runs perfectly but is locked in the garage most of the time.
  - The banking app processes transactions accurately. But it is down every night for maintenance for 3 hours.
- Available but not Reliable
  - A taxi that is always around but breaks down often
  - The banking app is always accessible. But payments fail, balances involve errors, or app crashes occasionally.

NON-FUNCTIONAL TESTING

- Usability testing
  - Is the application user-friendly or not?
- Efficiency testing
  - test the amount of code and testing resources required by a program to perform a particular function.
- Maintainability testing
  - how easy it is to maintain the system. E.g., version upgrade
- Portability testing
  - Can an application be moved from one environment to another?
  - Eg., Windows to Mac
- Baseline testing:
  - validation of documents and specifications on which test cases would be designed.
  - e.g., SRS validation
- Compliance testing
  - Follows standards by company or government?
  - GDPR, ISO

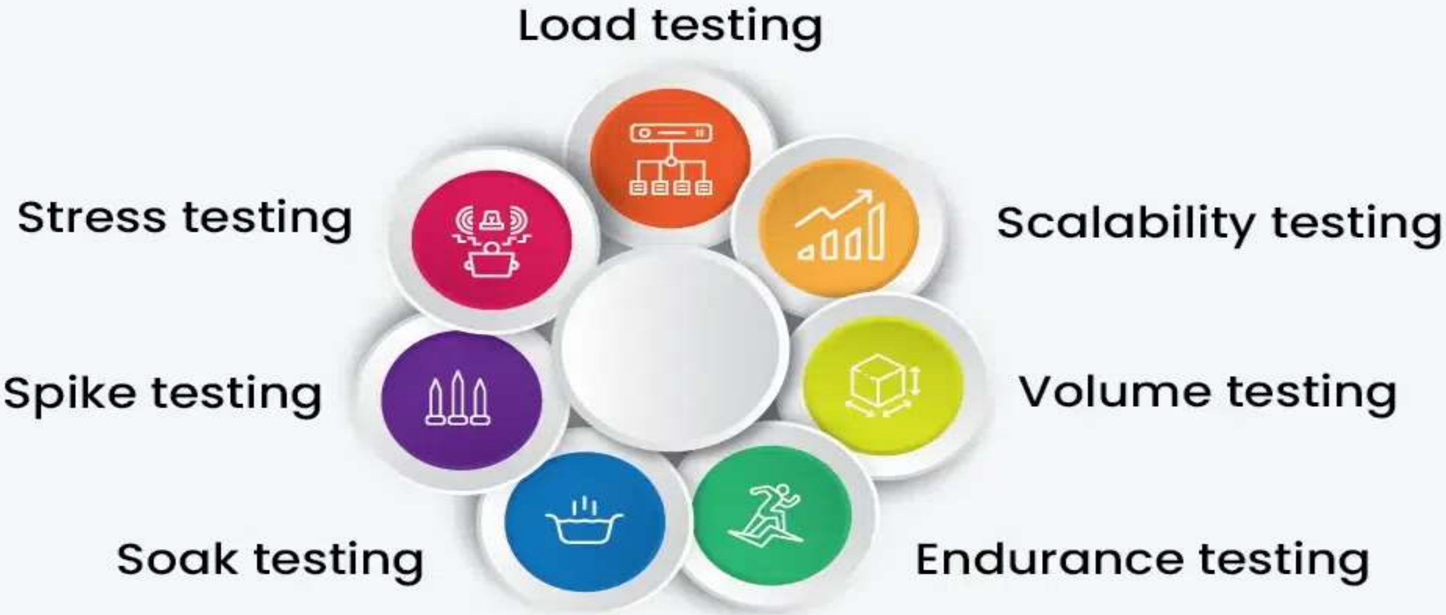
NON-FUNCTIONAL TESTING

- Endurance testing(Soak)
  - application is tested under normal or peak workload for an extended period of time.
  - Goal: Ensure that the software behaves correctly even after continuous usage for a long duration.
  - Many defects appear only after long periods of execution
    - Unreleased memory slowly filling RAM
    - Log files growing infinitely
  - After 10 hours, response time increases from 300ms → 3 seconds

Performance testing

- Evaluates how a system behaves under a given workload in terms of:
    - Speed (Response Time)
    - Scalability (Handling growth in users)
    - Stability (Reliability over time)
    - Resource usage (CPU, memory, network)
    - Throughput (Requests handled per second)
- Goal: Ensure the application is fast, stable, and scalable under expected load.

Types of Performance testing



Load testing

Stress testing

Scalability testing

Spike testing

Volume testing

Soak testing

Endurance testing

Performance testing types

- Load Testing
  - Tests behavior under expected user load.
  - Both normal and at peak conditions.
    - Identify performance bottlenecks
    - Validate capacity planning
  - It ensures
    - Acceptable response time
    - Stability under concurrent users
    - No failures at peak load

Types of Load Testing

a) Baseline Load Test

- Run with minimal users to record standard performance.

b) Peak Load Test

- Test with the maximum expected users.

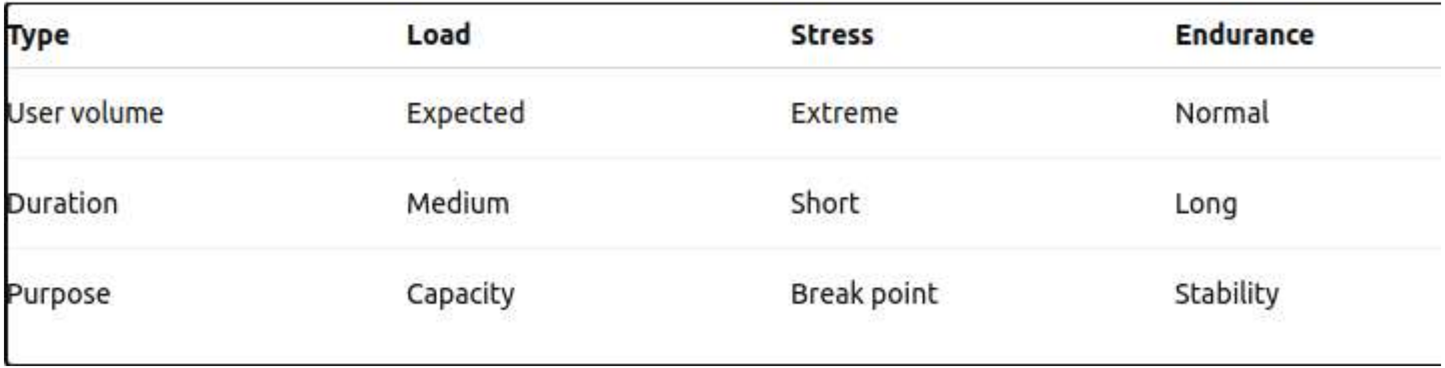
c) Incremental Load Test

- Gradually add users.

d) Scalability Load Test

- Check performance after adding resources.

Load vs Stress vs Endurance



Type	Load	Stress	Endurance
User volume	Expected	Extreme	Normal
Duration	Medium	Short	Long
Purpose	Capacity	Break point	Stability

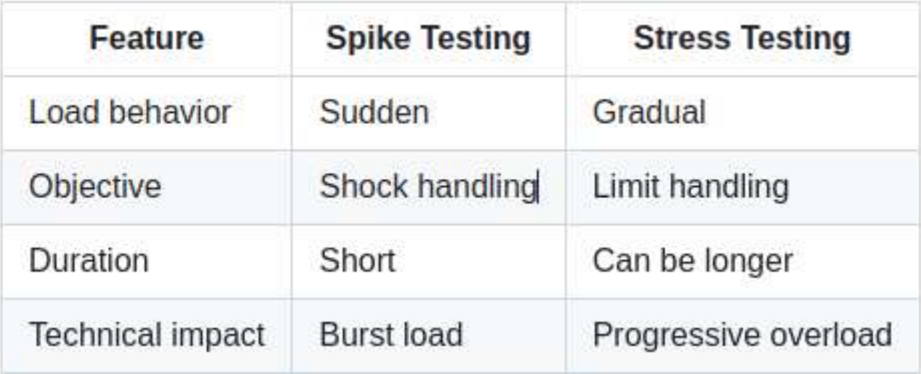
Stress Testing

- System is evaluated by applying an extreme workload beyond its normal capacity in order to determine how it behaves under failure conditions.
- To observe
  - When the system breaks
  - How it breaks
  - Whether it recovers after failure
- Goals
  - Identify the maximum capacity of the system
  - Detect system breaking point
  - Measure recovery time after failure
- Example:
  - Banking System -> Simulate 10× expected user traffic.

Spike Testing

- Tested by applying a sudden and extreme increase or decrease in load to observe how the application behaves during abrupt traffic changes.
- *“Can the system handle unexpected surges and drops in users?”*
- Goals
  - To verify system behavior under sudden high load
  - To detect failures caused by abrupt traffic changes
  - To check stability after load drops suddenly
- NPL ticket booking—test sudden spike in users -> Timeout errors
- Flash sale

Spike vs Stress Test



Feature	Spike Testing	Stress Testing
Load behavior	Sudden	Gradual
Objective	Shock handling	Limit handling
Duration	Short	Can be longer
Technical impact	Burst load	Progressive overload

Scalability Testing

- Evaluates a system’s ability to increase or decrease its capacity (users, transactions, data, or resources) while maintaining acceptable performance
- *“What happens when the system grows?”*
    - Users
    - Hardware/Resource
    - Workload
- **Goals = Future readiness**
    - Verify system behavior under growth
    - Identify performance limits
    - Check effect of adding resources
    - Estimate future capacity

Scalability Testing

- Example
  - An online shop expands from one city to nationwide use.
  - Bank App: New branches introduced.
- **Vertical Scaling (Scale Up)**
  - Increase system power (RAM, CPU).
  - Example: Upgrade server from 16GB → 64GB RAM.
- **Horizontal Scaling (Scale Out)**
  - Add more machines.
  - Example: Add more web servers behind load balancer.

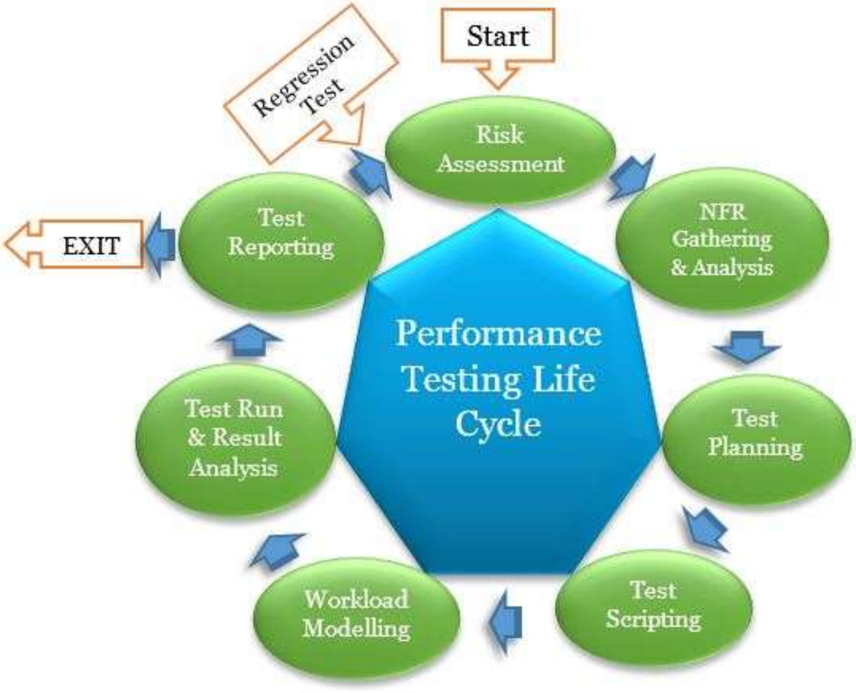
Volume Testing

- how a system behaves when a very large amount of data is processed, stored, or retrieved from the system.
- *“Can the system handle a massive amount of data?”*
- Goals
    - Validate database performance
    - Identify data-handling bottlenecks
    - Test storage capacity limits

Volume Testing

- Examples
  - Banking Database: Transaction histories across 10+ years.
    - Test: Query database with 100 million records.
    - Observation: Balance calculation slows
  - University Records System: A university stores data for 20 years of students.
    - Test: Insert lakhs of records into the database.
    - Observation: Search becomes slow

Performance Testing Process and Test Cases



Performance Testing Process and Test Cases

- Risk assessment
    - decide the scope of performance testing in a project
    - neediness of performance testing for each component in a software system based on risk score
- Requirement Gathering & Analysis
    - Gathering performance-related (NFR) requirements.
        - Expected number of concurrent users
        - Peak traffic conditions
        - Page load time requirements
    - Define KPI
        - System must support 5,000 concurrent users with response time under 2 seconds.

Performance Testing Process and Test Cases

- Test Planning
  - The scope of performance testing (components to be tested)
  - Test scenarios (realistic user interactions to simulate) or Entry/exit criteria
  - Required test environments (hardware, software, network configurations)
  - Performance testing tools (JMeter, LoadRunner, Gatling, Locust, etc.)
- Test Environment Setup
  - Hardware setup
  - Application servers
  - Database servers
- Test Script Development
  - Scripts are designed to simulate user behavior and interactions with the system
  - Should include parameterization for testing different input values.
  - Example: Create scripts for Login users

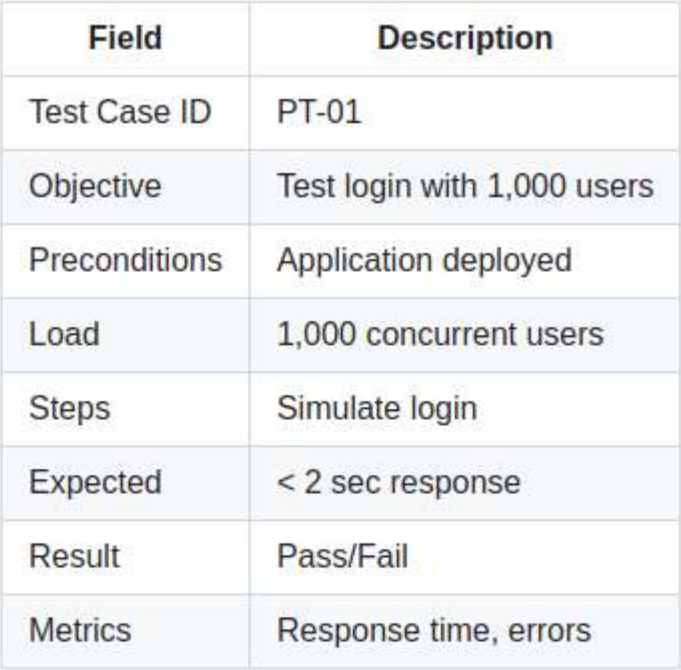
Performance Testing Process and Test Cases

- Workload Modelling
  - create the workload scenario as per the approved Performance Test Plan using test scripts
  - Example profile:
    - 60% browsing
    - 30% login
    - 10% purchase
- Test Execution & Monitoring, and Result Analysis
  - Run test scenarios: Load, Stress, Spike, Volume
  - Monitor
    - CPU, memory, and disk usage
    - Database query performance
    - API response times
  - Analyse to identify performance issues
    - Poorly managed thread execution causing deadlocks.
    - Complex Query causing Slow responses

Performance Testing Process and Test Cases

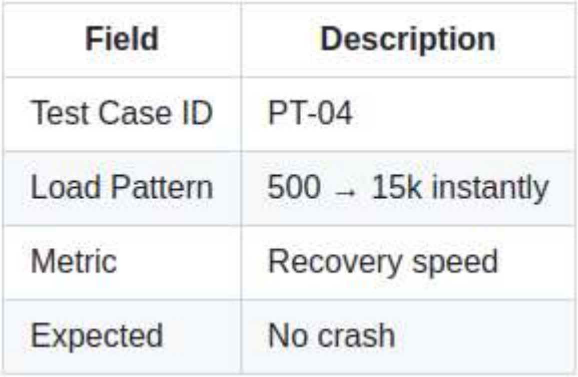
- Reporting
  - To prepare and publish the test report and provide the recommendations
  - Based on the analysis, development work on
    - Optimizing database queries (e.g., adding indexes, query caching).
    - Refining load balancing strategies
- Optimization and retesting
  - Based on the analysis, developers and DevOps work on issues
  - Performance tests are rerun to validate improvements. (Regression)
  - The process continues iteratively until the application meets performance benchmarks.

TEST CASE 1: Load Testing – Login Function



Field	Description
Test Case ID	PT-01
Objective	Test login with 1,000 users
Preconditions	Application deployed
Load	1,000 concurrent users
Steps	Simulate login
Expected	< 2 sec response
Result	Pass/Fail
Metrics	Response time, errors

TEST CASE 1: Spike Testing – Sudden Traffic



Field	Description
Test Case ID	PT-04
Load Pattern	500 → 15k instantly
Metric	Recovery speed
Expected	No crash

Reading material

- https://www.geeksforgeeks.org/software-testing/performance-testing-software-testing/

Regression Testing

- re-testing the existing software application to ensure that new changes (bug fixes, enhancements, or updates) have not broken existing functionality.
- “Old features still work after changes?”
- Goals
    - Ensure system stability after updates
    - Validate backward compatibility
    - Detect side-effects caused by changes

Types of Regression Testing

- **Full Regression**: Entire system tested
- **Partial Regression**: Only affected modules
- **Unit Regression**: Specific components
- **Selective Regression**: Key areas only
- **Automated Regression**: Script-driven tests

Regression Testing

- When
  - New feature added
  - Bug fixed
  - Environment changed
  - Database change
  - Code refactoring
  - Security patch applied
- Example: Banking Application
  - Change made: Interest calculation logic updated.
- Regression Test Action: Balance calculation, Fund transfer
- Result: A previous working feature (withdrawal) fails → regression defect.

Sanity Test

- Sanity Testing is a quick, focused testing
- Checks whether a **specific functionality** or bug fix works as expected
- Usually narrow in scope and shallow, not a full regression test
- Helps determine if the build is stable enough for further testing
- Often done without documentation, based on tester experience
- Types
  - Build Sanity: Checks if build is testable
  - Feature Sanity: Checks specific feature functionality

Sanity Test

- Banking Application
  - Change: Updated fund transfer feature
  - Sanity Test: Make a small transfer
  - Result: If transfer works → proceed with further tests
- Used for critical bug fix.(in prod)

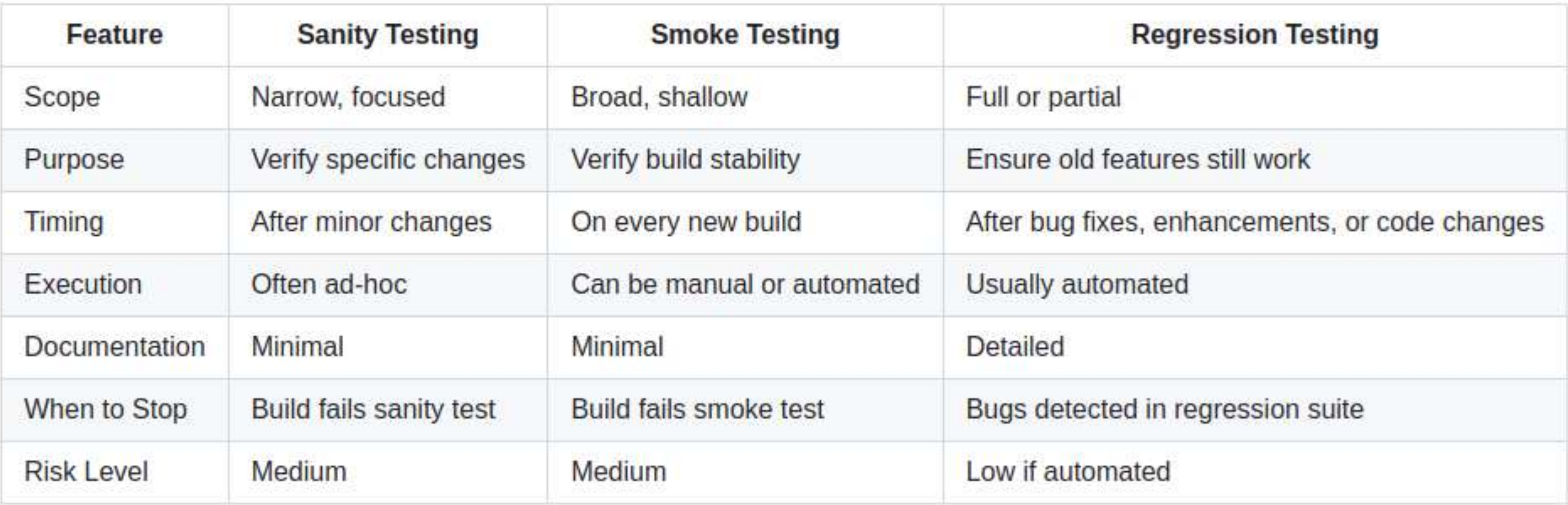
Smoke Testing

- Preliminary testing is performed on a new software build to ensure that the critical functionalities of the application are working.
- Check build is stable enough for detailed functional or regression testing.
- Focuses on **core application features** only.
- Goal
  - Verify that the main functions of the application work correctly.
  - Detect major issues early before detailed testing begins.

Smoke Testing

- Bank App
  - Critical Features: Login, Balance Inquiry, Fund Transfer
  - Test: Log in, check balance, transfer small amount
  - Result: Core functions work → proceed with detailed testing
- When
  - After receiving a new build
  - Before detailed functional or regression testing
  - After major integration or code changes

Sanity vs Smoke vs Regression



Feature	Sanity Testing	Smoke Testing	Regression Testing
Scope	Narrow, focused	Broad, shallow	Full or partial
Purpose	Verify specific changes	Verify build stability	Ensure old features still work
Timing	After minor changes	On every new build	After bug fixes, enhancements, or code changes
Execution	Often ad-hoc	Can be manual or automated	Usually automated
Documentation	Minimal	Minimal	Detailed
When to Stop	Build fails sanity test	Build fails smoke test	Bugs detected in regression suite
Risk Level	Medium	Medium	Low if automated

Compatibility testing

- Ensures that a software application works correctly across different environments
  - Browser Compatibility: Verify across different browsers
  - OS Compatibility: Test across Windows, macOS, Linux, Android, iOS
  - Device Compatibility: Desktop, tablet, mobile, wearable devices
  - Network Compatibility: Different bandwidths and latency conditions
  - Backward Compatibility: Works with older software versions or hardware
- Goal: Users have a seamless experience regardless of their system configuration.

Compatibility testing

- Example: Mobile App
  - Test on Android (different versions) and iOS devices
  - Check login, notifications, offline behavior
  - Result: Works on all targeted devices → compatible

Control Flow Testing (CFT)

- white-box testing that examines the sequence of execution of statements, branches, loops, and decision points in a program.
- Analyzes the paths and branches in code.
- Uses control flow graphs (CFGs) to represent program flow.
- Goal:
  - Helps in identifying unreachable code, infinite loops, or missing logic.
  - Improve code coverage (statement, branch, path coverage).

Data Flow Testing (DFT)

- white-box testing that focuses on the life cycle of variables in a program—their definition, usage, and deletion
- Tracks variable definitions, assignments, and usage.
- Detects errors like use-before-define, variable misuse, or memory leaks.
    - To detect errors such as undefined or unused variables.
    - Verify that variables are properly initialized, used, and updated.
    - Detect data anomalies in variable usage.

Reading materials

- https://www.geeksforgeeks.org/software-engineering/software-engineering-control-flow-graph-cfg/
- https://www.geeksforgeeks.org/software-testing/data-flow-testing/