Unit 4 speedrun

26 November 2024 13:54

TESTING

Lec1

Intro to testing

- Set of activities planned in advance and conducted systematically
- Makes sure that software performs as per expectations and exposes deviations
- Involves measuring attributes that'll enhance confidence in product
- Strategy incorporates:
  - Low level tests: verify that small source code segment is correctly implemented
  - High level tests: validate major system functions

Objectives of testing

- Demonstration
  - System used w acceptable risk
  - Functions under special conditions
- Detection
  - Discover defects, errors, deficiencies
  - Quality of components
- Prevention
  - Prevent/reduce errors
  - Avoid risks and problems

Verification

- Are we building the product right?
- Process of checking if software achieves goals to ensure that deliverables meet requirements
- Static testing
  - No code execution
  - Check docs, design, code
  - review, walkthrough, inspection
- Find bugs early in development
- Target software architecture
- Occurs before validation

Validation

- Are we building the right product?
- Focuses on product related activities that determine if system/project deliverables meet client expectations
- Dynamic testing
  - Includes code execution
  - Validate capabilities and features in project scope and requirements
  - Done by testing team
  - Black box testing, white box testing, non-functional testing

Terminology runthrough

- Defect:
  - Deviation from req
  - Faults due to code mistake
- Bug:
  - Coding error that prevent req working
  - Programmer mistake
- Failure:
  - Resulting state due to defect
  - Occurs during or after development lifecycle
- Issue:
  - Raised by end user when expectations not met

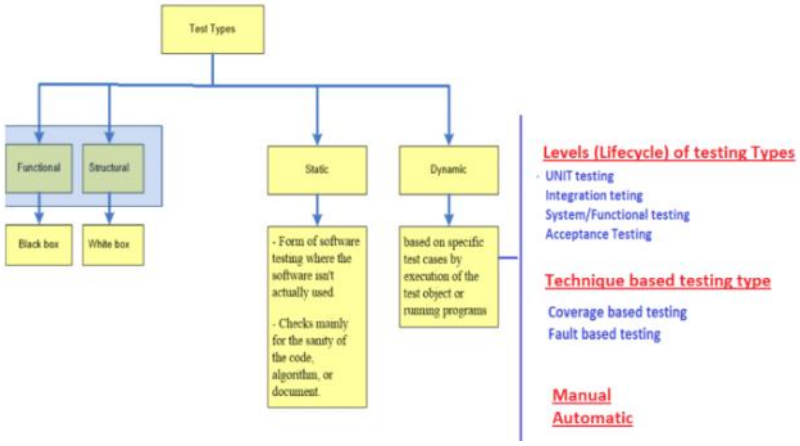
Characterizing testing:

- Why?
  - What observations are we looking for
- How much?
  - Size of sample
  - Exhaustive testing is not feasible
- How did we arrive at test cases?
  - Based off test strategy
- What do we execute?
  - Whole or part of product
- When do we test?
  - At what stage of lifecycle
- Which is the samples?
  - Based off test selection
  - Random/adhoc/algorithmic/statistical

- Where do we test?
  - For embedded systems

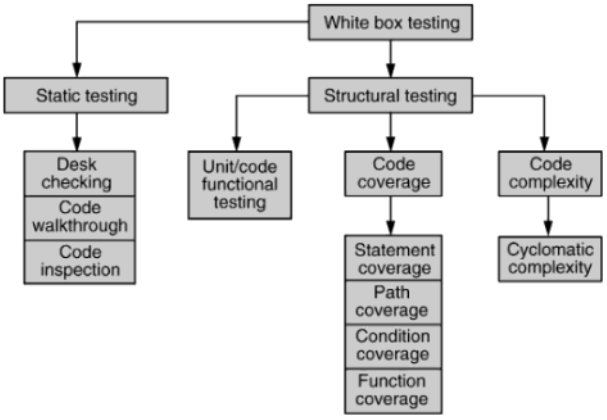
Lec 2:

Broad characterization



Black box testing

- Functional testing
- Treats software as black box- no regard to internal structure and logic
- Concerned only with external behavior
- Objective: identify defects in output as result of invalid input
- Tested from user pov
- Advantages
  - Best for large units of code
  - Early test planning
- Disadvantages
  - Some paths may not be tests
  - Can't direct tests to error prone code specifically

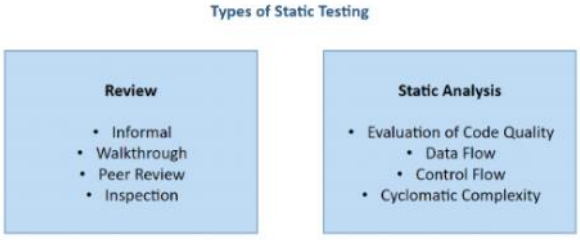


White box testing

- Structural testing
- Factors in internal logic and structure
- Test specifier uses knowledge of internal structure to derive test cases
  - Test cases not determined until code written
- Tested from developer pov
- Advantages
  - Partitioning by execution equivalence
  - Reveals hidden errors
- Disadvantages
  - Needs skilled testers
  - Hard to test all of the code

Static testing (subset of white box testing)

- Test for defects in software without executing code
- Done in early stages
- Easier to find source and solution when discovered earlier
- Types:
  - Review
  - Static analysis

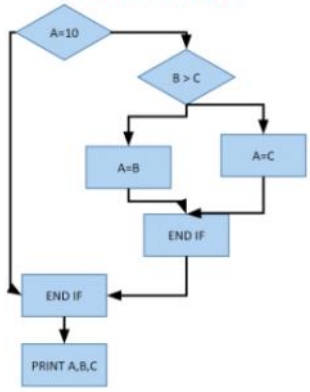


- Cyclomatic complexity
  - Quantitative measure of number of linearly independent paths in code section
  - Indicates complexity of a program -> computed using control flow graph
  - Control flow graph = directed graph where
    - node = smallest group of commands
    - Edge = connects two blocks of commands
  - Cyclomatic complexity M = E - N + 2P
    - E= number of edges
    - N= number of nodes
    - P = number of connected components

Code

```
IF A = 10 THEN
    IF B > C THEN
        A = B
    ELSE
        A = C
    ENDIF
ENDIF
Print A
Print B
Print C
```

Control Flow Graph



Computing Cyclomatic Complexity

Cyclomatic Complexity = E– N + 2*P
Where, E = Number of Edges
N = Number of Nodes
P = Number of Nodes with Exit Points

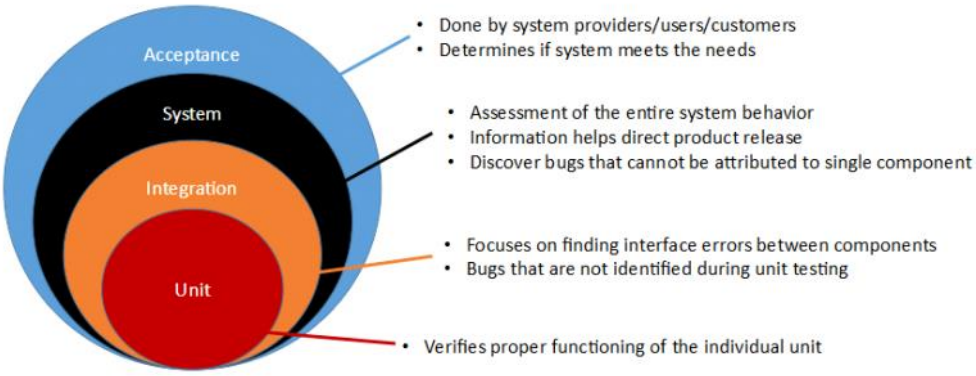
Here, E = 8, N = 7, P = 1
Cyclomatic Complexity = 8 - 7 + 2
= 3

Dynamic testing:

- Involves execution of code for analyzing dynamic behavior
- Provide input values, observe output values
- Advantage: find difficult and complex defects
- Disadvantage: time and budget
- Kinds of dynamic testing are based on:
    - Code or fault based
        - Control flow based
            - Execute every path at least once
            - No branch leads to abnormal behavior
        - Data flow based criteria
            - Selecting paths through control flow order to explore sequences of events related to variables or objects
            - Executes all statements at least once
            - No side effects
    - Approach for testing
        - Equivalence partitioning and boundary value analysis
            - Uncover errors and reduce test cases
            - Divide input into partitions
            - If it's in the range (1, 100) your test cases will the 0, 1, 100, 101
        - Specification based
            - Test functionality according to stated requirements
            - Test system behavior based on specific input
        - Intuition based
            - Rely on tester's experience to make test cases
        - Usage based
            - Find defects that are revealed when users interact with product
        - Application domain
            - Use specific knowledge of product domain to make test cases
    - How testing is done
        - Manual testing
            - Human performs test- step by step- no test scripts
            - Used in complex testing when automation is expensive
            - Slow and tedious
            - Hard to get test coverage
            - Acceptance testing, black box testing, white box testing, unit tests
        - Automates testing
            - Tests which are executed without human assistance
            - Needs coding, test framework maintenance
            - Fast and repeatable
            - Most efficient for periodic and regular tests
    - Levels of testing (next heading. It's big)

Lec 3

Levels of testing



- Unit testing
  - Focus: Test for coding errors prior to quality engineering
  - Tests smallest individually executable code units- BY programmers
  - Test cases for
    - Algo and logic
    - Data struct
    - Interfaces
    - Independent paths
    - Boundary conditions
    - Error handling
- Integration testing
  - Focus: verify interface between components against a software design
  - Performed by programmers to isolate timing and resource contention
  - Abstract away unit issues and look for defects between units

Strategies

Big Bang Approach

Integrate everything and run to see if system functions as expected

Iterative Approach

- Software components are integrated iterative until software works
- Top down or Bottoms up

- System testing:
  - Focus: completely integrated system to verify against compliance with srs
  - Detects both inter assemblage defects and system defect
  - Black box testing: tests end-to-end flow
  - Process:
    - Test env setup
    - Create test case
    - Create test data
    - Execute test case
    - Defect reporting and logging
- Acceptance testing:
  - Focus: verifies completeness of user story during sprint
  - Running a suite of tests on completed system
  - Each test case exercises a particular operating condition
    - Test environment is designed to be as identical to real life as possible
  - Provides confidence that delivered system meets requirements of sponsors and users
  - Final quality gateway

Alternate testing categorization

Acceptance / qualification testing	Checks the system behavior against the customer's requirements, however these may have been expressed.
Installation testing	Verifies the installation in the target environment. May be identical to system testing in a new environment.
Alpha and beta testing	Before the software is released, it is sometimes given to a small, representative set of potential users for trial use, either in-house (*alpha* testing) or external (*beta* testing). ... Alpha and beta use is often uncontrolled, and is not always referred to in a test plan.
Performance testing	Aimed at verifying that the software meets the specified performance requirements (capacity and response time, for instance). A specific kind of performance testing is volume testing, in which internal program or system limitations are tried.

Lec 4

System testing

- Smoke and sanity testing
  - Ensure most impt functions work
  - Decide whether build is fit for further testing
- Regression testing
  - Determine whether any changes have unintended side effects
- Installation and uninstallation testing
  - Verify is sw is installed with all necessary components and application works as you want it to
  - Verify if all components of the application have been removed
- Functional and non functional testing
  - FT: test features and functionality covering all scenarios (failure and boundary)
  - NFT: verify attributes of the system (performance or robustness)
- Destructive testing
  - Application fails in uncontrolled manner to test robustness and find point of failure
- Software performance testing
  - Responsiveness, scalability, stability and reliability of product
- Usability
  - Ease of use of system form user pov
  - Level of skill to learn/use software and acquire skill
- Localization
  - Verify quality of product localization for particular target
- Boundary testing
  - Blackbox testing using boundary values
- Startup/shutdown testing
  - Ensure system hasn't left uncleared lock files or inconsistent data when shutdown
  - Ensure product starts in deterministic and consistent state
- Platform/cross platform tests
  - To evaluate behavior of product in different environments
- Load tests
  - Determine behavior od system under varying loads
- Stress tests
  - System is pushed to maximum design load and beyond to test its limits
- Security testing
  - To uncover vulnerabilities and ensure data, resources are protected
- Compliance testing
  - Compliance with internal and external standards
- Recovery tests
  - To check restart capabilities after unanticipated shut down
- Scalability tests
  - To determine capability to scale up or down
- Cloud testing
  - Application tested using cloud computing services

TEST PLANNING

Lec 5 + 6

Ensuring scope is understood

- Understanding the context where the product will be used
  - Review use case
  - Discuss with designer and developer
  - Review project documentation
  - Product walk through
- Optimal testing based on
  - Customer req
  - Proj schedule and schedule
  - Product specification
  - Skills of team

Establish test adequacy criteria

- Criteria to determine when to stop testing or consider test complete
- Testing a subset of a possible combo doesn't guarantee absence of issues
- Fixing issues may not be possible due to
  - Volume of errors
  - Schedule
  - Resources
- Evolve test strategy to be followed
  - Test approach- defines how testing is carried out
  - Testing mindset
    - Demonstration
      - Make sure software runs and solves problem
      - Only tests what succeeds
    - Preventive
      - Prevents faults in early stages
      - Reviews and test driven development
    - Destruction

        - Try to make software fail and find as many faults
        - Difficult to decide when to stop testing
    - Evaluation
        - Detect faults through lifecycle phases
        - Focus on analysis
- Testing type
    - Lifecycles have outcomes which can be tested

Feasibility Phase	Acceptance test
Requirements	Requirement specification (functional and system test cases)
Architecture/design phase	Functional, system, integration test cases
Implementation phase	Code unit tests
Testing phase	Reviews and execution of test cases
Maintenance	Review and execution of regression and other tests

- Testing strategy
    - Testing in the small -> testing in the large
    - Top down and bottoms up
    - Positive and negative testing
    - Dynamic and heuristics based approach
- Test execution environment
    - Setup of software and hardware for testing teams to execute
        - Configured as per need of application
        - Includes test data, database, frontend, OS, servers, storage, network
    - Env management involves maintenance and upkeep of test bed
- Tools used
    - Application under test (AUT) needs to be compatible with test bed
    - Testers need to be comfortable with tool
    - Tools need to be balanced in terms of features, generate reports all that
    - Cross platform support is an expectation
    - Acceptability of tool in industry + availability of trained personnel and support
    - Cost
- Risk analysis w contingency planning
    - Risk = probability of an unwanted incident during or towards testing
    - Could be in terms of
        - Changes in business or tech competition
        - Resources
        - Quality of software
        - Test models not being able to be used
        - Test env and state
    - Any risks found would need to be planned for addressing as part of mitigation and contingency

Evolve list of deliverables

- Includes
    - Test specifications for each module
    - Test cases for different conditions planned

Creation of test schedule

- Includes estimates for all steps before
- Includes work breakdown schedule

Planning, identification, allocation of resources

- Test schedule creation and this step are done together or iteratively
- Initially involves
    - Identifying number of servers, storage, test tools, network resources
    - Identifying number of people and type of people working on project
    - Identifying test environment
- Schedule is reworked after resource and skill identification

Identification of milestones

- Identified after considering expected deliverables, schedule and commitments
- Track progress and control overruns
- Identify risk triggers

Risk management

Establishment of measures and metrics

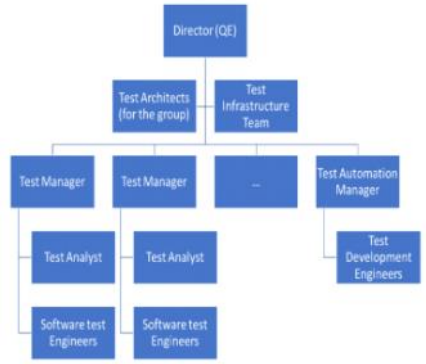
- Measurements are identified
    - Number of test cases passed and created
    - Number of test cases run
    - Amount of time spent on creation
    - Number of errors found

TESTING ROLES

Lec 7

Structure of testing organization

- Considered to be part of quality engineering



Roles and responsibilities

Test Director

Provide oversight, co-ordination, strategic vision, customer and stake holder connect

Test Manager

Prepare test strategy, plans for project, monitors and controls testing

Test Infrastructure Manager

Manages all the infrastructure, capacity planning, maintenance, support and configurations

Test Automation Manager

Manages the development of tools and scripts for automating

Test Architect

Designs test infrastructure, picks appropriate tools, validates test strategy

Test Analyst

Mapping of customer environment and test conditions and features to testing conditions and documentation

Software Test Engineer

Tests the product using appropriates testing techniques and tools

4

TEST PROCESS

Planning and control

- Identifying requirements
- Analyze test requirements, product architecture, interfaces
- Create test sufficiency criteria
- Create testing strategy, plan for resources, build schedule
- Setup review points, status reporting mechanisms

Design

- Identify test conditions, design tests, test environment

Implementation and execution

- Develop test cases, test data, test automation
- Execute tests, collect metrics, log results, compare
- Track defects, fix bugs

Evaluate exit criteria

- Evaluate test completion/stopping criteria based on functionality

Test closure activities

- When testing is complete/product is cancelled
- verify all planned deliverables
- Archive test scripts and reports
- Perform retrospective

Software test execution

- Subset of test cases selection for execution
- Test cases assigned to testers for execution
- Environment is set up along with test data
- Tests are executed, results are logged, status is captured, bugs are logged
- If roadblocks, resolve before continuing
- Results are reported and metrics are analyzed

Lec 8+9

Test cases

- Key part of test documentation
- Tells tester what to test, how to test, and what expected results are
- Acts as starting point for test execution
- Tests case parameters are:

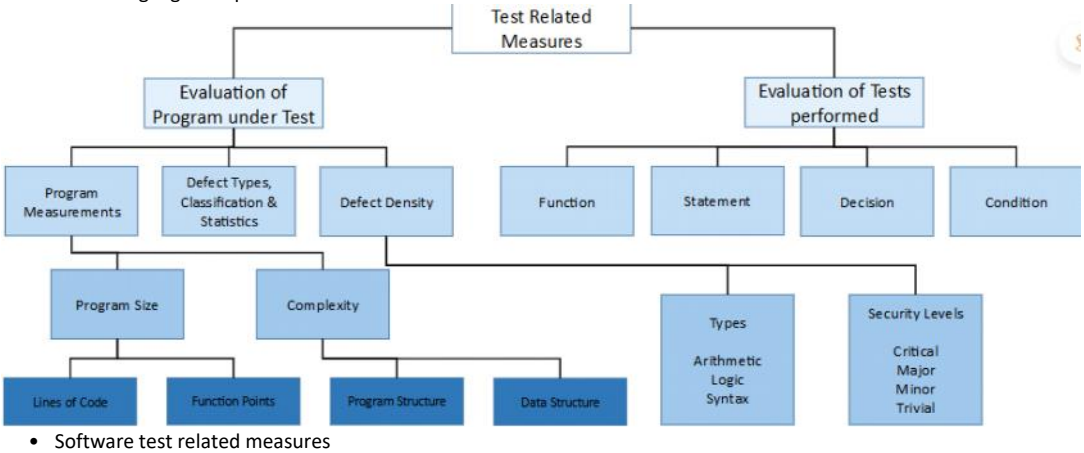
    - Test Case ID
    - Test scenario
    - TC description
    - Test setup
    - Test steps
    - Prerequisites
    - Test parameters
    - Expected result
    - Actual result
    - Comments
- Value of test cases:
    - Ensures good test coverage
    - Allows tester to find ways of validating a feature
    - Negative test cases are also documented
    - Reusable for the future
- Test case types:
    - Positive: verify that software does what it's supposed to
    - Negative: verify that software is not doing what it isn't supposed to
    - Destructive: scenarios that software handles before it breaks
- Best practices:
    - Short titles
    - Strong desc
    - Reusable
    - Clear and concise
    - Include expected result

Test suite

- Container that has a set of tests
- Three states
    - Active
    - In progress
    - Completed
- Test case can be added to multiple suites and plans
- Suites are created based on cycle or scope
- Works for any type of test
    - Functional
    - Non functional

Software test measurements and metrics

- STM = quantitative measure of testing process
- Characteristics
    - Quantitative
    - Understandable
    - Applicability
    - Repeatable
    - Economical
    - Language independent

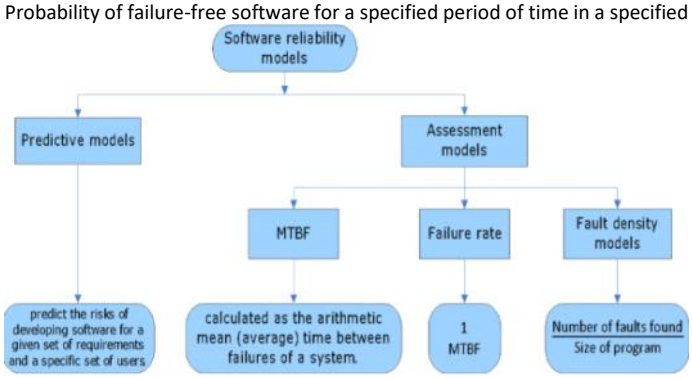


- Software test related measures
    - SLOC (size in lines of code)
    - Fault density (number of fault/program size)
    - MTBF (mean time bw failure)
    - Failure rate (inverse of MTBF)
    - Defect distribution (% of defects in one phase)
    - Defect density (faults in product/ faults in product)
    - Defect leakage: Test Efficiency = [(Total number of defects found in UAT) / (Total number of defects found before UAT)] x 100
- Software test related metrics

Metrics	Description
Percent Test Coverage	Includes: Statement Coverage, Branch/Decision Coverage, Condition Coverage, Loop Coverage, Path Coverage, Data Flow Coverage, etc.
Percent completed and Percent Defects corrected	# of test cases executed and closed compared to total # of test cases & # of defects fixed/closed compared to total # of defects
Defect Removal Effectiveness	% of defects found at the later phase of development.
Requirement Compliance Factor	Using the traceability matrix, the RCF measures the coverage provided by a test case to one or set of requirements.
Defect Discovery Rate	Number of defects found per line of code (LOC).
Defect Removal Cost	The cost associated with finding and fixing a defect.
Cost of Quality	Total cost of prevention, appraisal, rework/failure, to the total cost of the project.
Percent Injected Fault Discovered	Number of defects injected found, compared to total number of faults injected.
Mutation Score	Number of killed mutants to the total number of mutants. Again, any mutant left alive points to the weakness in the testing effort.

Software reliability

- Probability of failure-free software for a specified period of time in a specified env



Test driven development

- Prevention model
  - Writes tests first, then implementation
  - Agile approaches- XP
  - Tools: Junit
- Steps associated
  - Add test
  - Run test + earlier test and see if system fails
  - Continue incrementally until all planned tests run properly
  - Refactor system to improve design and reduce redundancies

Issues in software testing

- Test selection
  - Prioritize:
    - Highest priority first
    - Complex code first
    - Largest modules first
    - Most often modified first
- Objectives for a test
- Testing for defect identification
- Theoretical and practical limitations
- Infeasible paths
- Testability

TESTING TOOLS

Lec 10

Selenium

- Open source- automating web browsers
- Allows testers to write scripts that can interact w web applications
- Languages:
  - Java
  - Python
  - C#
  - Ruby
  - JS
- Selenium operation suite:
  - Selenium IDE
    - Record and playback for quick test scripts
  - Selenium RC
  - Selenium WebDriver
    - Directly interacts with browser to execute automation scripts
    - Supports many browsers

    - Selenium grid
        - Parallel execution of tests across diff machines and browsers
- Advantages
    - Open source
    - Highly extensible
    - Runs across diff browsers
    - Supports various OS
    - Supports mobile devices
    - Executes tests in parallel
- Disadvantages
    - Only web applications
    - No in built object repo
    - Automates slowly
    - Data driven testing is hard
    - Cannot access elements outside web application under test
    - No official user support

Junit

- Simple framework for repeatable tests: Java
- Eliminates need to verify and tally results-> faster product dev
- Annotation:

@before	Execute statement before each test case
@beforeClass	Execute statements before all test cases
@after	Execute statement after each test case
@afterClass	Execute statements after all test cases
@ignore	Ignore some statements during execution

- Features of Junit:
    - Fixtures
    - Test suites
    - Test runners
    - JUnit classes
- Advantages
    - Alternate front ends for displaying test results
    - Separate class loaders for each unit test
    - setUp and tearDown for resource utilization
    - Set of assert method to check test results
    - Integration with tools such as Ant, Maven and IDE's such as Eclipse and Jbuilder
- Disadvantages
    - Cannot do dependency testing
    - Not suitable for high level testing
    - Large test suites
    - Cannot test various JVM at once

Apache Jmeter

- Open source to load test functional behavior and measure performance
- Run on any env that accepts JVM
- Simulates group of users sending requests to a target server and returns statistical info of target server
- Components:
    - Thread group
    - Samplers
    - Listeners
    - Configuration
- Advantages
    - Open source
    - Friendly UI
    - Platform independent
    - Multi threading framework
    - Highly extensible
    - Visualize test result
    - Unlimited testing capabilities
    - Easy installation
    - Supports multi protocols
- Disadvantages
    - Scripting involves some understanding of Jmeter elements, regular expressions, session handling
    - No network visualization
    - Single normal configuration nor enough for load tests
    - Doesn't support ajax, js, and flash
    - Very limited realtime test monitoring

SOFTWARE MAINTENANCE

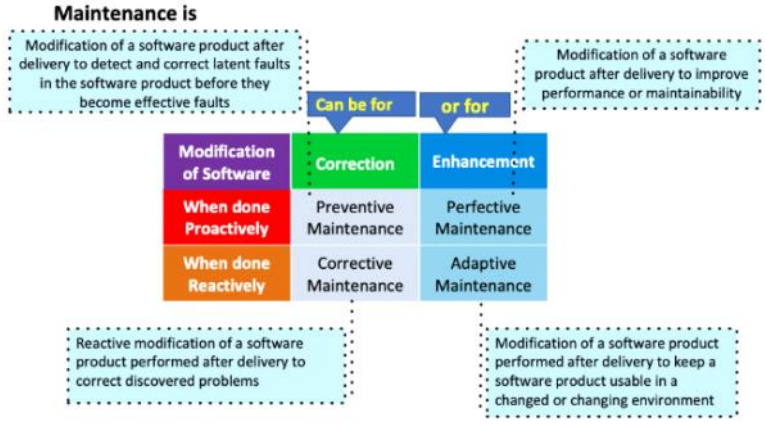
Lec 11

Software maintenance

- Sustaining process of modifying software system or component after delivery to correct faults, improve performance, adapt to changed env
- Activities expected:
  - Maintaining control over software's day to day functioning
  - Maintaining control over software modifications
  - Perfecting existing functions
  - Preventing software performance degrading to unacceptable levels
- Activities that support it:
  - Understand product to be maintained
  - Discuss with architect + developers
  - Documentationnnnnnnn
  - Look at present and change request at hand
  - Look at future and analyze consequences of action
- Factors affecting costs:
  - Application type
  - Software novelty
  - Software maintenance staff
  - Software lifespan
  - Hardware characteristics
  - Quality of software design and all that
- Key issues:
  - Technical
    - Code and documentation quality
    - Limited understanding
    - Availability of test env
    - Impact analysis
  - Management
    - Staffing
    - Economic objectives
    - Outsourcing issues
      - Protection of IP
      - Control over development and quality control
      - Learning curve for product
      - Scope of maintenance
  - Cost
  - Predictability

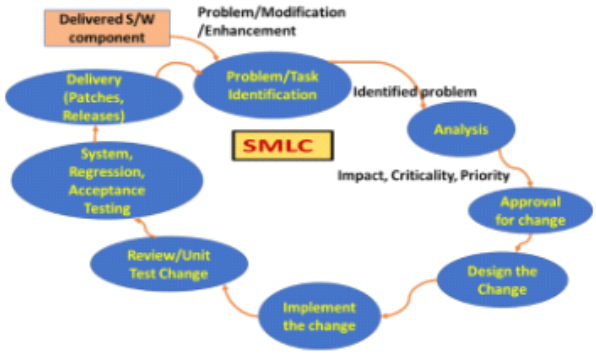
Lec 12

Categories of maintenance



Maintenance process activities

- Implementation of processes
  - Develop, document and execute maintenance process plans
  - Procedures for handling user reports
- Follow SMLC



- Migration
  - For the fixed/new component and execute the plan
    - Notifying and replacing the product
    - Applying path with restart/no-restart and patch
    - Migration, verification of migration, support for older data
- Software retirement
  - Develop retirement plan for system
  - Retire older version and running of new one in parallel
  - Dispose superseded hardware and software

Reverse engineering

- Passive technique to understand piece of software prior to re-engineering
- Recover specification about system from source code
- Creates a representation of the software in different format
- Doesn't modify product
- Output: control flow graphs

Re-engineering

- process of modifying the software to make it easier to understand, change and extend
- Expected to result in reduction of CPU utilization, readability, usability, and performance
- Steps:
  - Identify current process
  - Document planned reconstruction
  - Reverse engineer
  - Refactor
    - Change software system such that external behavior doesn't change but improves internal structure
    - AFTER design/code of a component is done
    - Types:

DATA REFACTORING	CODE REFACTORING	ARCHITECTURE REFACTORING
Once data analysis is complete, data refactoring involves standardizing data definitions and making physical modifications like database migration	Although code refactoring can resolve immediate problems associated with debugging, it is not reengineering. Real benefit is achieved only when data and architecture are refactored as well.	The most time-consuming form of refactoring. Involves completely redesigning, recoding, and testing of the code 😱

  - Reconstruct code and data

SOFTWARE ENG IN GLOBAL ENV

Lec 13

Types of development

- Colocated development
- Multisite development
- Global development
  - Why?
    - Cost savings
    - Taking advantage of time difference to extend productive hours
    - Larger and global pool of devs
    - Advantage of diversity of stakeholders
  - Challenges
    - Communication
    - Coordination
    - Control
    - Cultural
  - Overcoming challenges
    - Collaboration models
      - Team leasing

        - Team extension
        - Project teams
    - Teaming models
        - Assembly line
        - Construction
        - Engineering
- Project management practices
    - Common ground
    - Decoupling work
    - Collaboration
    - Empowerment
    - Communication

Hacking

- Using deep understanding of computer systems to exploit systems
    - Whitehat hackers:
        - To improve security of org's security systems
    - Blackhat hackers:
        - Subvert tech for malicious reasons

Hacker
- Solve problem with non standard approaches or exploit weakness
- Need novelty, attracted by challenges and toys
- Weakly motivated by conventional rewards

Software Engineer
- Take a problem, fits it within criteria, to meet performance benchmark, exceed reliability requirements, make it aesthetically pleasing, meet arbitrary regulations
- Creative and original thinking given constraints

Ethics

Personal Ethics
- Values or viewpoints an individual uses to influence and guide his or her personal, social, or professional behavior
- Values are reinforced and strengthened from childhood

Business Ethics
- Values or viewpoints which a business and "individuals when doing business" use to influence and guide behavior
- Focus on behaviors pertaining to work

Professional Ethics
- Principle that govern the behavior and judgements while performing his profession

- Considering ethics as a framework
    - Reason we make a choice matters
    - Most people have an intrinsic sense of what's right
- Utilitarian framework
    - Cost analysis based on risks, costs, and probabilities of outcomes
    - Includes considering legal violations
- Framework of individual rights
    - Can't put price on person's life
    - People are entitled to certain rights
- Golden rule
    - Treat others as you'd wanna be treated
- Principle of beneficence
    - Prioritize doing good
- Principle of least harm
    - Deals with situs where no choice is beneficial
- Principle of respect for autonomy
    - Allow people to make decisions that apply to their situations without interference

Intellectual property

- Tangible creations of human intellect
- Types:
    - Patents
        - Methods to do something (utility)
        - Look and feel (design)
    - Copyrights
        - Focuses on expressions of implementations
        - Authorship is impt to creators
        - Provides protection of investement
- IP infringement is costly

- Patents have higher value

IT Service Lifecycle

- Service strategy
  - Understands organizational objectives and customer needs and provides strategic guidance
- Service design
  - Turns strategy into a plan for delivering business objectives and technology
- Service transition
  - Develops and improves capabilities for introducing new services
- Service operation
  - Manages services in supported environments
- Continual service improvement
  - Incremental and large scale improvements to services

Holistic service management

- Four dimensions to look at:
  - Organizations and people
  - Information and technology
  - Partners and supplies
  - Value streams and processes
- Service value chain defines 6 key activities
  - Plan
  - Design and transition
  - Improve
  - Obtain/build
  - Engage
  - Deliver and support
- Guiding principles
  - Focus on value
  - Start where you are
  - Progress iteratively
  - Collaborate and promote visibility
  - Think and work holistically
  - Simple and practical
  - Optimize and automate

DEVOPS

Lec 15

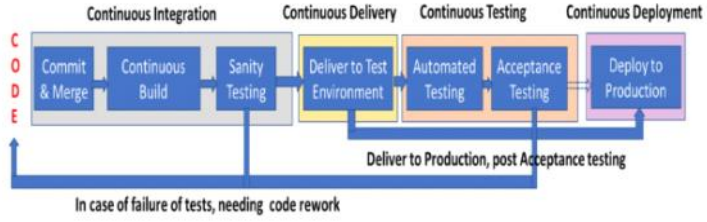
Devops

- Combination of cultural philosophies, practices and tools that increase org's ability to delivery applications at high velocity
- Represents collaborative or shared approach to tasks performed by the company's application development teams and IT operations teams
- Extension of agile-> deliver functionality incrementally and faster

Version control

- Record changes to files stored within the system
- Developers can make changes individually or within groups and save these changes through commits or revisions within the system in one way or the other

Devops pipeline



- Continuous integration
  - Continuously merge code written by developers with master branch
  - Change sets are small and hence work to resolve conflicts is small and incremental
  - Avoids difficult integrations
  - Reduced debugging time
  - Supports early detection of issues
- Continuous build
  - Involves compiling, handling dependencies and generating an executable product
  - Involves running some static code analysis tools like lint
- Sanity testing
  - Executable built is run with some sanity tests to ensure that total compilation is successful and all dependencies are resolved
- Continuous delivery
  - Frequent shipping of code to env

    - Integrated bits are ready to be deployed and this supports release of new changes to users
    - Process must be triggered in last step of integration
- Continuous testing
    - Process of executing predominantly automated tests to validate code
    - Designed to be executed with minimum wait times
- Continuous deployment
    - Software release process
    - Used in highly mature devops teams
    - Significant number of times, validated integrated components are batched together and are then deployed into customer environment

Agile vs Devops

Agile	**Devops**
Methodology/iterative SDLC which supports changes and collaborations between developers and other stakeholders	Culture and approaches which look to remove the silos of dev activities and increasing velocity of deployment
Focuses on constant changes	Focuses on constant testing and delivery
Target = software development	End to end business solutions and fast delivery
Functional and non-functional readiness	Operational and business readiness

Lec 16

Supporting pillars of devops:

- Collaboration
    - Process of working towards a specific outcome through supporting interactions
    - Effective collaboration:
        - Theory of mind
            - Ability to recognize one's perspective and understand that others have their own too
        - Equal participation
        - Communication
        - Shared goal
        - Hierarchies are less emphasized
- Affinity
    - Measure of strength of relationship between individuals
    - Building affinity = building inter team relationships
    - Achieved by
        - Shared values
        - Consistent team culture
        - Team cohesion
    - Can be measured by
        - Shared time
        - Intensity of relationships
        - Reciprocity of shared stories
        - Reciprocity of support
- Tools
    - Accelerator for change
    - Eg: system provisioning tools, Build Tools, Automation tools, Testing tools, Monitoring tools, Log extraction tools, Deployment tools
- Scaling
    - Focuses on processes and pivots organizations must adopt throughout the lifecycles
    - Takes into account how other pillars have been applied throughout organization

Jenkins

- Opensource automation tool in java with plugins for continuous integration
- Functionality:
    - Generates test reports
    - Integrated with many diff version control systems
    - Pushes to various repos
    - Deploys directly to test env
    - Notifies stakeholders of build status
    - Supports UI customization
- Pipeline:
    - Each stage represented by a cell
    - Header = name of job being run at that stage
    - Green = successful job, Red = job failed
- Automated testing facilitates higher velocity of deployment

Devsecops

- Native devops extension
- Accelerate high quality software delivery with automatic deployment, acceleration, and shutdown response
- Helps in various functions in SDLC
- Benefits of security measure available:

    - High speed and sharpness of defense teams
    - Ability to respond to change and need urgently
    - Better interaction and communication
    - Early detection of rick in code
- Key components:
    - Code analysis
    - Change management
    - Compliance monitoring
    - Threatening investigation
    - Risk assessment
    - Security training
- Build pipeline:
    - Planning and training teams
    - Automate security
    - Assess dependencies for risk
    - License compliance check