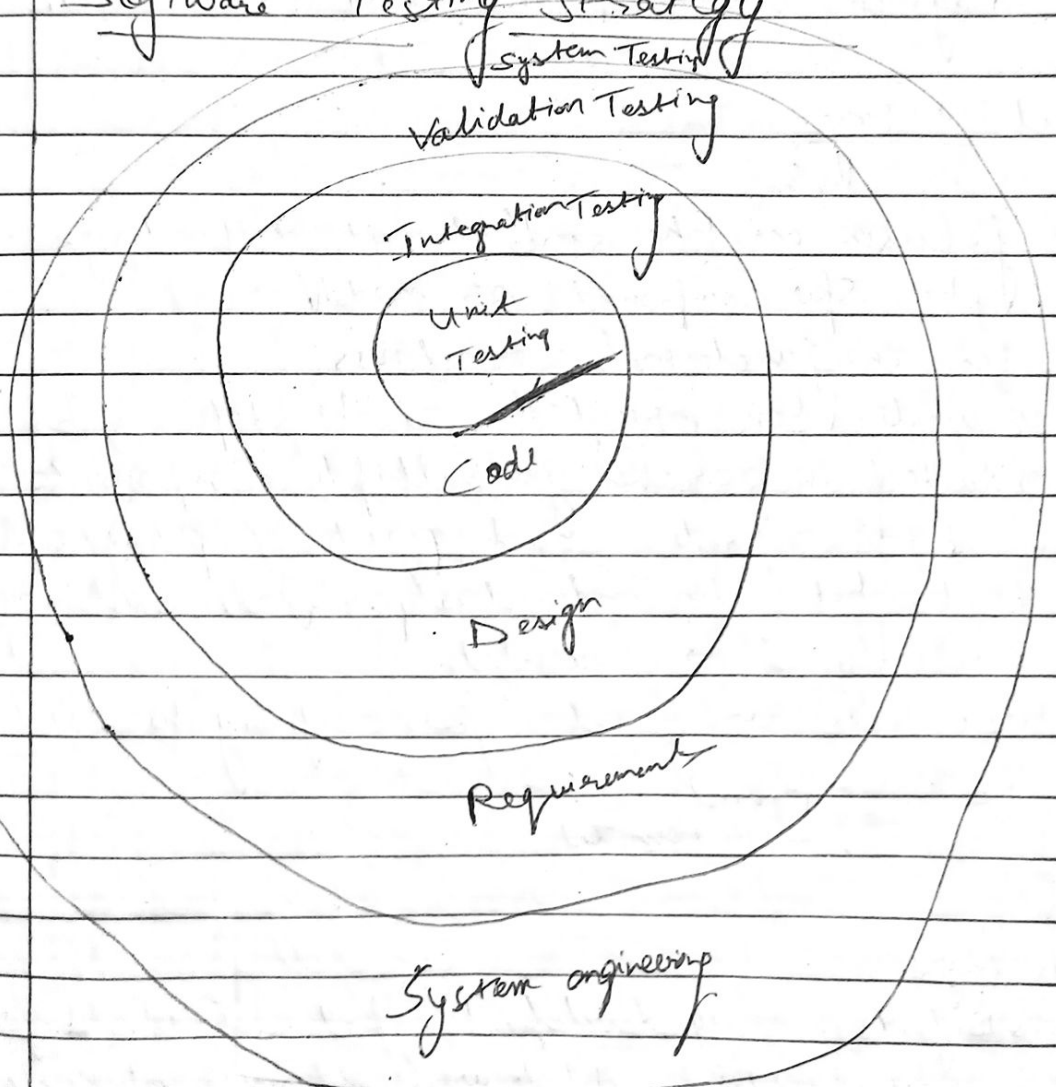
Are we building the product right?

Are we building the right product

Verification VS Validation

Verification	Validation
• It refers to activities that ensure that Software correctly implements a specific function	• It refers to activities that ensure that the built S/W is traceable to customer requirements
• Performed at every phase	• Performed after software is made

Software Testing Strategy



Date:
Page:

- Unit Testing begins at vertex of spiral & concentrates on each unit (component) of s/w as implemented in source code.
- Testing progress by moving outward along the spiral to integration testing, where the focus is on design & construction of s/w architecture.
- Taking another turn outward on spiral, we encounter validation testing, where requirements are established as part of s/w requirement analysis are validated against the s/w that has been constructed.
- Finally, system testing, where the s/w & other system elements are tested as a whole.

Unit Testing

- It focuses on the smallest unit of s/w design i-e the s/w component or module.
- Focuses on functional correctness
- It is white-box oriented, & the step can be conducted in parallel for multiple components
- Goal is to separate each part of program & test that the individual parts are working correctly & as intended.
- These tests are great for confirming the correctness

→ Types → Manual
→ Automated

→ Advantages

- Reduces defects in newly developed features or reduces bugs
- Reduces cost of testing as defects are captured in early phase
- Improves design & allows better refactoring of code
- Unit tests, when integrated with build gives the quality of the builds as well

Date:
Page:

Stub and Drivers

These are actually computer programs that work as substitution of some computer modul which are not available for testing. These simulates the functionality of other modules and allows us to perform the testing activity

→ Stubs

- Used in Top-Down approach, when you have upper modules but you don't have bottom modules
- Now you can't test the upper modules without the bottom modules so just for testing phase instantly you make the simulation of bottom modules & check the working of upper module
- Kind of dummy programs, simulating the functionality

→ Drivers

- Used in Bottom-Up approach when bottom level modules are prepared but the top level main program is not prepared
- Drivers are main programs used to call other program / module
- As we know, to run any module we always need a main program so driver is the main program.

Date:
Page:

11 Integration Testing

- In this phase, individual S/w modules are combined & tested as a group.
- It takes as its input modules that have been unit tested, groups them in larger aggregates, applies test defined in integration ~~testing~~ test plan to those aggregates, & delivers as its output the integration system ready for system testing.
- Ensures Functionality, performance & Reliability

→ Bottom-Up Integration Testing

- In this modules at lowest level are developed first & other modules which go toward the main program are integrated & tested one at a time
- Uses test drivers to drive & pass appropriate data to the level modules
- As & when code for other module gets ready these drivers are replaced with actual module.

→ Top-Down Integration Testing

- It is an incremental testing technique which begins by testing the top-level module & progressively adds in lower level module one by one
- Lower level modules are simulated by stubs which mimics the functionality of lower level module.
- As you add lower level code, you will replaces stubs with actual components

Validation Testing

- The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements
- Validation Testing ensures that the product actually meets the client's needs

→ Advantages

- To ensure Customer satisfaction
- To be confident about the product
- To fulfill the client's requirement until optimum capacity
- S/w acceptance from end user.

→ Stages of Validation Testing Process

- Validation Planning: This is a project-specific plan that defines the scope & goals of your validation project
- Define Requirements: Whether you're just starting the s/w development process or you're making changes to an existing product, you'll need to establish a set of requirements to work toward. These requirements stipulate what features your product should include & how these features should work

Date:
Page:

- Selecting a Team: You'll need to field an experienced & competent development team
- Developing Documents: Develop a user specifications document where you describe the operating conditions
- Estimation Evaluation: Next, we proceed with the s/w testing stage, verify it operates as originally intended & submit a validation report
- Fixing Bugs or Incorporating Changes: Finally, update the s/w to remove any bugs or issues found during evaluation

System Testing

- It is a level of testing that validates the complete & fully integrated s/w product
- The purpose of a system test is to evaluate the end-to-end system specification
- System Testing is a black box testing

→ Types of system testing

- Performance Testing: It is a type of s/w testing that is carried out to test the speed, scalability, stability & reliability of s/w product or application
- Load Testing: Carried out to determine the behaviour of a system or s/w product under extreme load
- Stress Testing: Performed to check the robustness of system under varying loads

Date:
Page:

→ Steps of system testing

1. Test Environment Setup: Create testing env. for better quality testing
2. Create Test Case: Generate Test case for testing process
3. Create Test Data: Generate the data that is to be tested
4. Execute Test Case: After generation of test case & test data, test cases are executed
5. Defect Reporting: Defects in the system are detected
6. Regression Testing: It is carried out to test the side effects of testing process
7. Log Defects: Defects are fixed in this step
8. Retest: If the test is not successfull then again test is performed

White Box Testing

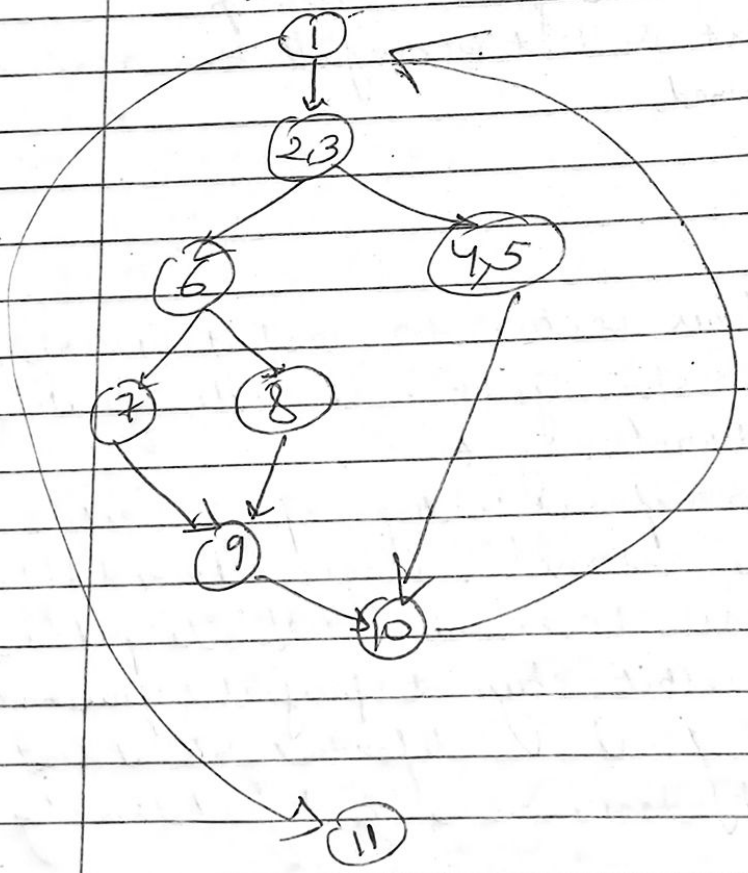
1. This approach allows testers to inspect & verify inner working of a s/w system — its code, infrastructure & integration with external system
2. It is also known as transparent testing, open box testing
3. Used to test s/w internal logic, flow & structure
4. Tester creates test cases to examine the code paths & logic flows to ensure that they meet specified requirements
5. It guarantees traversal of all independent paths at least once
6. Evaluates all logical decisions & find whether they are true or false
7. Evaluates all loops — to check their boundaries.

→ Types of white box testing

1. Basis path Testing

- Based on control structure of a program or code
- Technique of selecting the paths in control flow graph that provide a basis set of execution paths through program or module.
- Objective is to define number of independent paths so the number of test cases needed can be defined explicitly to maximize test coverage

(i) Flow graph notation



(ii) Independent program paths

Path 1: 1 → 11

Path 2: 1 → 2 → 3 → 4 → 5 → 10 → 1 → 11

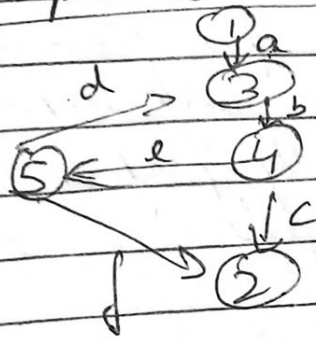
Path 3: 1 → 2 → 3 → 6 → 8 → 9 → 10 → 1 → 11

Path 4: 1 → 2 → 3 → 6 → 7 → 9 → 10 → 1 → 11

(iii) Deriving Test Cases

prepare test cases that will force execution of each path in basis set

(iv) Graph matrices



	1	2	3	4	5
1			a		
2					
3				b	
4		c			e
5			d		

2. Control structure testing

(i) Statement coverage: Aim is to traverse all statements at least once. Hence, each line of code is tested.
In case of flow chart, every node must be traversed at least once.

(ii) Branch coverage: Here test cases are designed so that each branch from all decision points are traversed at least once.
In case of flow chart, all edges must be traversed at least once.

(iii) Data flow testing: Focuses on two points
① In which statement, variable are defined
② ——————, variables are used
It designs the test cases that cover control flow paths around variable definations & their uses in the modules

Black-Box Testing

- A testing method that tests functionalities of S/W applications without having knowledge of internal code structure, implementation details & internal paths.
- mainly focuses on output & input of S/W application.
- Also known as specifications based testing
- Independent Testing Team usually performs this type of testing during S/W testing cycle.

→ Techniques of Black-box Testing

(i) Graph Based Testing: First step is to understand the objects & the relationships among them. Next step is to define series of tests that verify "all objects have the expected relationship to one another"

(ii) Equivalence partitioning method :- In this, we divides the input domain of a program into classes of data from which test cases can be derived.
An equivalence class represents a set of valid or invalid states for input conditions.

(iii) Boundary Value Analysis technique: A greater no of errors occurs at boundaries of input domain. Test both sides of each boundary. Look at output boundaries for test cases

(iv) Decision Table Testing: A systematic approach where diff input combinations & their corresponding system behaviours are captured in tabular form. Used for testing & requirements management.

Eg:

condition	Rule 1	Rule 2	Rule 3	Rule 4
Username	F	T	F	T
Password	F	F	T	T
	F	F	F	T

Next page

Date:
Page:

Project Management Spectrum

(i) People: The most imp element of successful project
(ii) Process: The set of framework activities & s/w engineering tasks to get the job done
(iii) Product: The software to be built
(iv) Project: All work required to make the product reality

S/w Metrics

- Standard of measuring the quality
- help in measuring progress & health of s/w
- Three types of s/w metrics are

(1) Process Metrics

- These metrics refer to process quality
- These metrics help us in improving
  - s/w development → How effectively & timely removing defects
  - s/w maintenance → Response time to fix the issues
- These are collected across all projects & over long periods of time
- We derive a set of metrics based on the outcomes that can be derived from the process. Outcomes include:
  - (a) Errors uncovered before release of s/w
  - (b) Defects delivered to & reported by end-users
  - (c) Work products delivered (productivity)
  - (d) Human Effort expended
  - (e) Calendar time expended, etc

(ii) Project Metrics

- Measure of s/w project ⊂ are used to monitor and control the project
- They enable s/w project manager to:
  (a) Minimize the development time by making the adjustments necessary to avoid delays & potential problems & risks
  (b) Assess product quality on an ongoing basis & modify the technical approach to improve quality
- Metrics collected from past projects are used as a basis from which effort & time estimates are made for current s/w project
- As a project proceeds, actual values of human effort & calendar time expended are compared to original estimates.
- The data is used by project manager to monitor & control project

(iii) Product Metrics

- Measure of s/w product at any stage of its development from requirements to installed system.
- used to measure:
  (a) the size of final program
  (b) the feature of final program
  (c) complexity of s/w design
  (d) performance of final program
  (e) no of pages of documentation produced

Normalization of Metrics

Date:
Page:

(i) Size oriented metrics (KLOC measurements)

- measure size of sfw produced
- Any lines in code apart from comments & blanks
- Normalized by counting in Thousan Lines of Code (KLOC)
- Size oriented metrics of project computed by :
  - (a) Errors per KLOC
  - (b) Defects per KLOC
  - (c) $ per KLOC pages of documentation per KLOC

- Advantages
  - Easy to count or calculate from developed code
- Disadvantage
  - Programming Language dependent
  - Penalized well-designed but short programs
  - Not universally accepted as best way

(ii) Function Oriented Metrics

- measure of functionality delivered by application
- most widely used metric here is Functional point
- Using historical data it can :
  - (a) Estimate cost or effort req to design, code & test
  - (b) predict no of errors that will be encountered during testing
  - (c) forecast no of components & no of project source lines in implemented system

- Advantages
  - Programming language independent
  - Based on data that are more likely to be known in early stages
- Disadvantages : Computation is based on subjective data
  - Count of info domain can be difficult to collect
  - Has no direct physical meaning, it's just a number

Project Planning

- S/w project management begins with a set of activities that are collectively called project planning
- The S/w project planner must estimate three things before a project begins:
  (a) How long it will take
  (b) How much effort will be required
  (c) How many people will be involved.
- In addition, the planner must predict the resources (h/w & s/w) that will be required & the risk involved
- S/w project planning includes five major activities
  (a) Estimation
  (b) Scheduling
  (c) Risk Analysis
  (d) Quality management planning
  (e) Change management planning

The overall goal of project planning is to establish a logical strategy for controlling, tracking & monitoring a complex technical project.

→ Project Planning process

- The objective of S/w project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost & schedule
- The project plan must be adapted & updated as project proceeds.

The more you know, the better you estimate. So update your estimates as project progress

→ Task Set for Project planning

(i) Establish project scope
(ii) Determine feasibility
(iii) Analyze risks
(iv) Define required resources
- determine human resources required
- define reusable s/w resources
- identify environmental resources

(v) Estimate cost and effort
- decompose the problem
- develop two or more estimates using size, FP.
- Reconcile the estimates

(vi) Develop a project schedule
- Establish a meaningful task set
- Define a task N/w.
- Use scheduling tools to develop a time chart
- Define schedule tracking mechanisms

→ Estimation

- involves how much money, effort, resources, & time it will take to build a specific s/w.
- Estimation of resources, cost, and schedule for a s/w engineering effort requires
  (a) Experience
  (b) Access to good historical info (metrics)
  (c) the courage to commit to quantative predictions when qualitative info is all that exists
- Estimation carries inherent risk & this risk leads to uncertainty.

→ Software Scope

- It describes:
  - (a) Functions & features that are to be delivered to end user
  - (b) Data that are input & output
  - (c) Performance, constraints, interfaces & reliability that bound the system
- Scope is defined using one of two techniques.
  - (a) A narrative description of s/w scope is developed after communication with all stakeholders
  - (b) A set of use-cases is developed by end-users

→ Feasibility

- Once s/w scope is well-understood, it is necessary to determine the feasibility of s/w.
- It is a crucial part of estimation process
  - (a) A study that determines whether a requested system makes economic, technical, & operational sense for an organization
  - (b) Conducted by project manager
  - (c) Takes place before the system is constructed

→ Resource Estimation

- The s/w project planner must conduct an estimation of the resources required to accomplish the s/w development effort
- Three major categories of s/w engineering resources:
  - (a) ~~People~~ (Human resources) → no of people & skills needed
  - (b) Reusable s/w components → full exp components, partial exp components, new components
  - (c) ~~Development env~~ (End. resources) (h/w & s/w tools)

Date:
Page:

⊙ Each of the resources is specified with four char —
(a) Description of resource
(b) A statement of availability
(c) When the resource will be required
(d) Duration of time that resources will be appl

Decomposition

→ Problem-Based Estimation

(i) start with bounded statement of scope
(ii) Decompose the s/w into problem function that can be estimated individually
(iii) Compute an LOC and FP value for each func
(iv) Derive cost or effort estimates by applying LOC or FP values to your baseline productivity metrics
(v) Combine function estimate to produce an overall estimate for entire project

→ Process Based Estimation

(i) identify the set of functions that s/w needs to perform as obtained from project scope
(ii) identify the series of framework activities that need to be performed for each func
(iii) estimate the efforts (in person-months) that will be required to accomplish each s/w process activity for each funct
(iv) Apply avg labour rates to effort estimated for each process activity
(v) Compute total cost or effort for each func & each framework activity
(vi) Compare the resulting values to those obtained by way of LOC or FP estimates.

COCOMO — empirical estimation model

- It is known as ~~Cost~~ Constructive Cost Model
- one of the most widely used s/w estimation models in world.
- developed by Barry Boehm in 1981
- It predicts the efforts & schedule for a s/w product development based on inputs relating to size of s/w & no of cost drivers that affect productivity
- It has three different models that reflect the complexity:
  - the Basic Model
  - the Intermediate Model
  - the Detailed Model

→ Basic model

- Applicable to small to medium sized s/w projects
- Use for a quick & rough estimates
- Three modes of s/w dev are considered

(i) Organic
- small team of exp programmers, develops s/w in familiar env
- Require little innovation
- Size range (0 - 50 KLOC)

(ii) Semi-detached mode
- intermediate mode b/w organic & embedded
- Depending on problem at hand, the team include the mixture of experienced & less exp people
- Require medium innovation
- Dev. env is medium
- size range (50 - 300 KLOC)

(iii) Embedded mode

- project has high constraints
- Hard to find exp persons
- requires significant innovation
- Development env is complex
- Size Range (over 300 KLOC)

→

	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi det	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- Efforts applied in person-month

$$E = a_b (KLOC \text{ or } KDSI)^{b_b} \text{ months}$$

- Development Time in months / Schedule

$$D = c_b (E)^{d_b} \text{ months}$$

- No of people required / Average staffing FSP

$$P = E/D$$

- Productivity

$$= \frac{\text{Lines of Code (not in KLOC)}}{E} \quad DSI/M$$

- Average staffing

$$= \frac{E}{D}$$