Date. _____
Page No. _____

Software Engineering

Date.
Page No.

8.2
58 18/1/18

→ In building a software, it is necessary that it should be effective. No unnecessary code will / should be written.

→ Should be of good quality, and should not lag much.

Software executing

→ 45% → Delivered but never successfully used
Client is not satisfied with software, points out lot of ineffectiveness, problems.

→ 20% → Used for a time, but later on abadoned.
→ Imp → Requirement should be added during runtime.

→ 30% → Paid for, but not delivered. (company took the money, but software wasn't given.

→ 5% → Is actualy used.
(2% + 3%)
↑ ↳ Usable rework.
used as delivered

Date
Page No.

→ Small project does not need much software engineering. But big project does.

Software engineering → A problem Solving Activity

→ Analysis : Analyze the problem. Understand the nature of the problem and break the problem into pieces. Convert them into sub-problems.

→ Synthesis: Put the pieces together into a large structure.

For problem solving we use (techniques) -

1) Technique (methods)

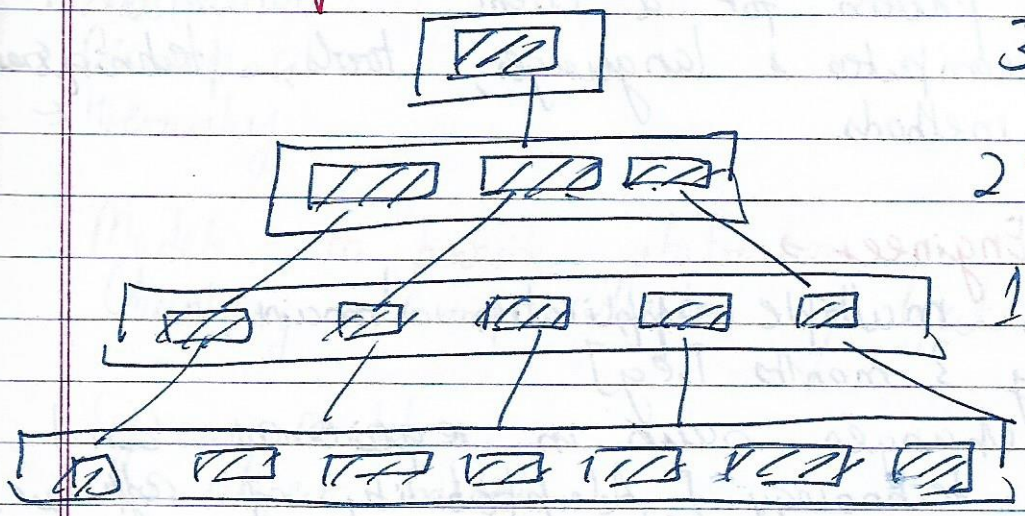
→ Using of some well-defined notation (Sorting and all) of which gives you the desired result.

2) Methodologies: -
Collection of techniques applied across software development and unified by a philosophical approach

3) Tools:
Instrument or automated systems to accomplish a technique. (eg → (comparing which cloud company to take).

Date.
Page No.

Principle of Abstraction.



Dividing problems into sub-problems.

Simulator → To simulate similar / real world environment. To test your software.

Software engineering → is a collection of techniques, methodologies and tools that help with production of -

→ a high quality software system
→ with a given budget.
→ before a given deadline.

while change occurs.

Computer Scientist → Prove theorems about algorithms, design languages, defines knowledge representation, schemes.
→ Has infinite time.

Date.
Page No.

Engineer → Develops a solution for an application-specific problem for a client.
→ Uses computers & languages, tools, techniques and methods.

Software Engineer →
→ Works in multiple application domains.
→ Has only 3 months [eg]
Imp → While changes occurs in requirements and available technology. [Adaptability of software].

Factors affecting quality of software system.

→ Complexity: Should be reduced. Divide the problem [If problem is very big and complex], into subproblems.

- The introduction of one bug fix causes another bug. [It should not happen this way].

→ Change: Change should be done in a way that quality should not degrade. "The entropy of a software ~~change~~ system increases with each change."

- As time goes on, cost to implement a change will be too high, [i.e. change on an old system is difficult]. True of all systems, independent of their technology.

Date.
Page No.

Dealing with complexity:
→ Abstraction
→ Decomposition
→ Hierarchy

Models to provide abstraction.
Chunking: Group collection of objects.

System model
→ Object model.

Synopsis → PERT Chart - Time schedule chart and dependencies between them.

System model → Object model
functional model
Dynamic model

Task model → who does what.

Decomposition → Functional
→ Object Oriented

→ To find class for new software system → green field engineering

→ To identity classes in an existing system → reengineering

→ Integrate engineering.

Date.
Page No.

Software process: organizing a structured set of activities to develop software systems.

Plan driven processes are processes where all of process activities are planned in advance and progress is measured against this plan.

Agile process, planning is incremental and it is easier to change the process to reflect changing customer requirements.

Waterfall model

1) Analyze the requirement
2) System design → Algorithm [SRS] → software requirement
   Software design → Code.
3) Implementation → design team
   Unit testing → testing team
4) Integration
   system testing.] – Testing
5) Operation → Launch
   maintenance → Time to time updates.

Drawback → Difficulty of accommodating change after process is underway [started].
We cannot move from one phase to another.

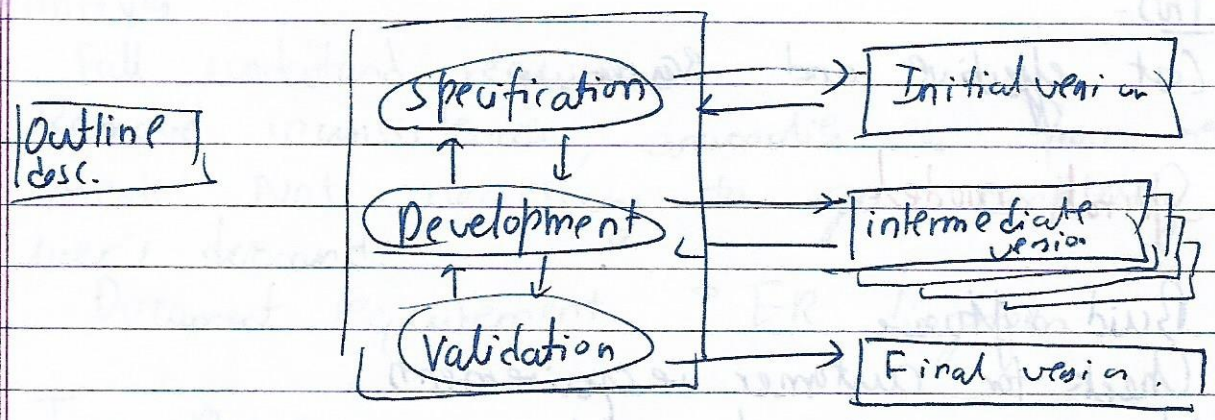
Date.
Page No.

Problems

→ Changing the requirement is very difficult, everything should be known beforehand.

→ Mostly used for large system engineering projects, where project developed is at several sites. So, implementing changes is very difficult and costly at diff. sites.

Incremental Development



Accomodates change, so multiple intemediate version. But in waterfal there is no intemediate version.

→ Cost of accomodating changes is reduced.
- small project.

→ Customer can give feedback on development work

→ Rapid delevery.

Problems → Procen is not visible → Not cost-effective to keep record of all intemodicle changes.

→ System structure / old code. tends to get degrade as new increments are added.

Date.
Page No.

Re-use oriented software engineering

Activities

→ Component analysis → to identify which components are useful and which to discard.
→ Modification of requirements.
→ System design with reuse.
→ Development and integration.

Pros -

→ Cost effective and easier.

Spiral model

→ Build software
→ Check for customer requirements.
→ If no goto 1 else stop.

High risk (As less input for customer requirements, most of are our assumptions).

Rapid Prototyping

Key idea: Customers are non-technical and don't know what they have.

Date.
Page No.

Unit - 2

Require Analysis and Specification

Basically like ER diagram where you need to identify everything.

Many projects fail because they start implementing the system without deciding whether they are meeting the needs of a customer or not.

Motives

→ Full understand requirement.
→ Remove inconsistencies, anamolies etc from requirement. Not neccessary to go completely as per user's demand.
→ Document requirement - ER diagram

Two Phases

→ Requirements gathering and analysis.
→ Specification.

System analyst → Collects data pertaining to the product, analyses collected data then make the software requirement specification (SRS) document.
→ Take customer reviews.

How to gather requirements?

→ Observation of existing system
→ Study existing procedures.
→ Discussion with customers and end-user.
→ Analysis of what needs to be done.

Date.
Page No.

eg. of above - planning a trip.

- If no existing system, lot of imagination and creativity are required.
- Interacting with customer to gather relevant data, requires a lot of experience.
  - eg - As if we have don't know the syllabus.

Characteristics of Good System Analyst

→ Good Interaction Skills
→ Imagination and creativity
→ Experience.

→ Inconsistent requirement - Some part of the requirements contradicts with some other part.

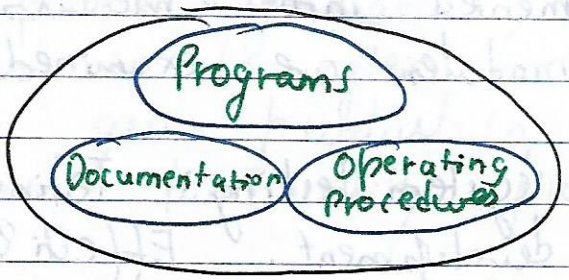
→ Incomplete requirements - some requirements have been omitted due to oversight.

Date.
Page No.

Software Engineering

The discipline whose aim is the production of quality software, which delivered in time, within budget and also satisfies its requirements.

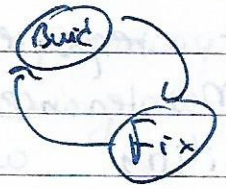
Program vs Software



Software

Build and Fix model

First phase is write some code. Serond is to corect any error or to add more function ality suitable for 100 - 200 line.



Waterfall model

1) Requirement analysis and specification phase:
In this exact requirements of customer is noted and documented properly. A long document, written in natural language, specifying "what" the software does and not "how" it does is prepared known as software requirements specification (SRS)

Date.
Page No.

2) Design phase: The goal of this phase is to convert the above requirement specification into a structure that can be implemented in a suitable high level programming language (known as software design description (SDD).

3) Implementation and unit testing: In this above SDD, is implemented into modules. In unit testing modules are examined and modified

4) Integration and system testing: Testing constitutes 1/3 to 1/2 of development. Effective testing will lead to high quality software, lower maintenance, more satisfied customers.

As unit testing tests for single module, integration tests the interface between modules.

5) ~~mai~~ Operation and maintenance phase:
Time spent and effort required to keep the software operational. Maintenance → error correction enhancement of capabilities and optimization.

Problems

→ Not possible to determine all requirements before hand.
→ cannot accomodate change.
→ not suitable for large projects.
→ real projects are rarely sequential.
→ working version is not seen

→ easy to understand
→ define before design
→ design before code

Date.
Page No.

Incremental - effective in situations where requirements are defined precisely and there is no confusion about the functionality of the software.

Iterative enhancement model

→ delivers a subset of product at each release
→ Does a portion of work.
→ lets customer do after first release.
→ Customer gets product in phase.
→ Gets first functionality product very fast.

Rapid Application Development model (RAD)
Incremental process model by IBM

Rapid prototype is build, given to user for feedback, feedback is incorporated and prototype is refined.

1) Requirements planning phase :- Requirements are captured using and group elicitation technique.

2) User description :- Team of developers and users, prepare, understand and review the requirement

3) Construction phase :- Detail design, coding, and testing phase of waterfall.

4) Cut-over phase :- acceptance testing by users, installation of system and user training.

Date.
Page No.

Pros -
→ Quick delivery of product.
→ less development of time.
→ Increase productivity.
→ Involvement of user so better acceptance.

cons -
→ User may not be involved throughout life cycle.
→ Highly specialized developes needed.
→ Not effective if system not modularised.

Spiral model → (PRDA)
Traditional software does not incorporate risk, whereas spiral does.
→ many spiral path of 360°.
→ many prototype are build.
→ refinement of prototype at every cycle.
→ Each cycle into 4 sector -

1) Planning - of objectives, alternatives and constraints
2) Risk analysis - analyze alternatives to identify and resolve risks.
3) Development - product development and testing
4) Assesment - Customer evaluation.

Pros -
1) Becomes equivalent to other life cycle models in appropriate condition.
2) incorporates software quality objectives in software development.

Date.
Page No.

3) Risk analysis and validation steps eliminate errors in early phases.

Cons -

→ Lack of explicit process guidance in determing
- objectives
- constraints
- alternatives

→ Relying on risk assement expertise.

→ providing more flexibility than required for ma application.

Prototyping

As product is not available late into life cycle. We ~~can~~ develop a software prototype having limited functionality, lower realiability and unthuste peformance. ~~Then~~ This is evaluated by the user and helps in refining the requirements and prepation of SRS. Sole use is to determin customer's need. Prototype is thrown away, it increases cost but overall cost comes down. Then actual system is developed using waterfall model. It act as an input to waterfall model and produces maintainable and good quality software.

Date.
Page No.

Crucial process steps

Requirement engineering is disciplined application of proven methods, techniques, tools and notation which describes and intended behaviour of proposed system and its associated constraints.
It consists of -

1) Requirements elicitation - Gathering of required information, with the help of ~~user~~ or existing system, if available.

2) Requirements analysis - Analysis of above requirements for any inconsistencies, omissions and defects etc.

3) Requirements documentation - After elicitation and analysis comes documentation. Whose main aim to produce a software requirement specification (SRS) document.

4) Requirements review - Review ~~shoot~~ and verification should not be treated as a discrete activity it is continous activity during elicitation, analysis and documentation phase.

The main output of requirements engineering is the requirements specification.

Date.
Page No.

Types of requirements

A stakeholder is a person having direct or indirect influence on the system requirements.

1) Known requirements - Something a stakeholder believes has to be implemented.

2) Unknown requirements - Forgotten by stakeholder as it may not be needed right now or needed by some other stakeholder.

3) Undreamt requirements - Something a stakeholder cannot think of due to limited domain knowledge

These all can be functional or non-functional.

Functional and non-functional requirements

Functional - Describe what a software has to do, known as product features. Sometimes also specify what a software shall not do.

Non-functional - mostly quality requirements that stipulate how well a software does what it has to do. Concerns with performance, availibi reliability, usability and flexibility.

Functional requirements are needed for customer acceptance whereas non-functional requirements makes a customer happy and satisfied.

Date.
Page No.

User and System requirements

User - Written for user includes functional and non-functional requirements. In simple language, highlight only overview of system without design characteristics.

System - Expanded form of user requirements. Input for designes for software design documents.

Both constitute to SRS.

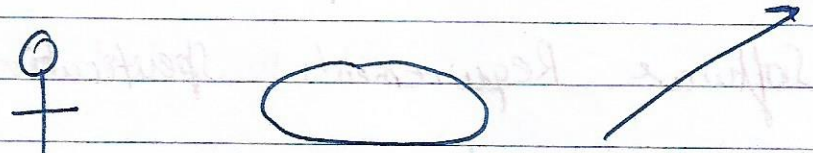
Use case

A primary actor is one needing the assitance of a system. Whereas a secondary actor is one from which the system needs assistance.

Guideline -

1) Identify the users of system.
2) Prepare a profile of each user, roles they have to play, goals which system has to support.
3) For each goal prepare a use case, using use case template.
4) Structure the cases, avoid over structuring.
5) Review and validate with the users.

Date.
Page No.

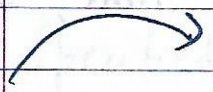


Actor Use case R/sp b/w actor and use case.

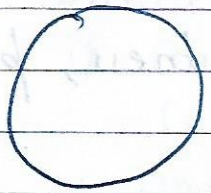
Throwaway - Quick and dirty, no rigor, build only difficult parts, optimize development time, throw away the prototype.
eg → As in prototyping model.

Evolutionary - no slopiness, rigorous, build understood parts first, solid foundation, optimize modifiability, evolve the existing prototype.
eg → Spiral model.

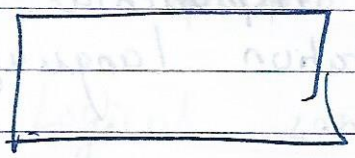
DFD - Data Flow Diagram



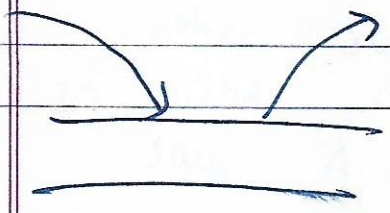
Data Flow



Process



Source or Sink



Data repository

Date.
Page No.

SRS - Software Requirements Specification.

1) SRS can be prepared by the customer.
   - states needs and expectations by customer.

2) SRS can be prepared by the developer.
   - serves as a contract document b/w customer and developer.

Nature of SRS

Issues SRS writer shall address -

1) Functionality - what a software is supposed to do?

2) External Interfaces - How software will interact with people, other hardware, and software.

3) Performance - Availability, response time, speed, recovery time of software.

4) Attributes - considerations of Correctness, portability, security, maintainability.

5) Design Constraints imposed on implementation : constraint for implementation language, policies for database integrity.

Date.
Page No.

Characteristics of SRS

1) Correct - SRS iff every requirement stated is one that software shall meet.

2) Unambiguous - If every requirement have only one interpretation. Diff. users shall not have diff. meaning.

3) Complete - iff
   1) All requirements such as functionality, external interfaces, performance, attributes and design constraints are met.
   2) Responses to valid and invalid input
   3) Full labels, references of diagrams and definitions, and units of measure.

4) Consistent - iff no conflict like

1) Specified characteristics of real world object conflict.
   1) One say output in tabular while other says textual.
   2) One specifies lights in blue while other specifies green.

2) Logical conflict between two requirements.
   1) One says to add A and B, while other says to multiply.
   2) One says, A follow B, while other says A and B simultaneously.

Date.
Page No.

3) Two or more requirements may describe same real-world object but with different terms. One use - "prompt" other uses "cue".

5) Ranked for importance.

Requirements should be ranked which are critical to diff. b/w essential, conditioned and optional!

6) Verifiable.
Requirement should be finite one, which can be measured.
Not like - "works well", "fast", "usually happen".
But like - "In 20 second".

7) Modifiable
Changes should take place easily, no redundancy as it will affect modification.
[Same as update anamoly].

8) Traceble
Requirements should be traceable to their origin.

→ Backward traceability: Depends on each requirement explicitly referencing it source in earlier documents.

Date.
Page No.

→ Forward traceability: each requirement in SRS should have a unique name or reference number.

Date.
Page No.

LOC

A line of code is any line of program text that is not a comment or a blank line, regardless of number of ~~frag~~ statements or fragments of statements on line. This specifically includes program header, declaration, executable and non-executable ~~co~~ statements.

FPA

Functional point analysis, decomposed into functional units:

- → Inputs: information entering the system.
- → Output: information leaving the system.
- → Enquires: request for instant access of information
- → Internal Logical files: information held within the system.
- → External Interface file: information held by other systems that is used by the system being analyzed.

CFP

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} w_{ij}$$

	Low	Average	High
External I	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Date.
Page No.

~~Undetermined~~ Unadjusted Functional Point

FP = UFP * CAF

Complexity Adjustment Factor = $(0.65 + 0.01 \sum F_i)$

$\sum F_i = \sum$ 14 Q's

14 Qs x Rating.

0 - no influence
1 - incidental
2 - moderate
3 - average
4 - significant
5 - essential.

EAF = 0.9 - 1.4

COCOMO

Basic

$E = a_b (KLOC)^{b_b}$

$D = c_b (E)^{d_b}$

Organic - 2 - 50 KLOC
Semidetached - 50 - 300 KLOC
Embedded - Over 300 KLOC

	a_b	b_b	c_b	d_b
Org.	2.4	1.05	2.5	0.38
Sem.	3.0	1.12	2.5	0.35
Emb.	3.6	1.20	2.5	0.32

E - PM D - M

Date.
Page No.

Staff size = $\frac{E}{D}$ persons.

Productivity = $\frac{KLOC}{E}$. KLOC/PM.

Intermediate

$E = a_i (KLOC)^{b_i} * EAF$

$D = c_i (E)^{d_i}$

	a_i	b_i	c_i	d_i
Org.	3.2	1.05	2.5	0.38
Sem.	3.0	1.12	2.5	0.35
Emb.	2.8	1.20	2.5	0.32

EAF → Effort adjustment factor
Multiplication of all related factor.

→ 0.82 × 1.14] Skilled
→ 1.29 × 0.95 ↑ Unskilled.

Static Single variable SEL → $E = 1.4L^{0.93}$
$D = 4.6L^{0.26}$

Static multivar → W-F: $E = 5.2L^{0.91}$
$D = 4.1L^{0.36}$

Date. _____
Page No. _____

Boundary Value Analysis

Based on fact that input value near the boundary have higher chances of error.

3 types → Values lying on boundary
Values just above boundary [lower bound
Values just below boundary [upper bound

for each variable: - 1) Minimum value.
2) Just above min
3) Nominal value
4) Just below max. value
5) Max. value.

Single Fault assumption - Out of 2 variables, only 1 variable can be assumed faulty and hence can be assigned any boundary value while other variable has to take a correct or nominal values.

For n variables → (Max y) $(4n+1)$ test cases

eg. $(x, y) \Rightarrow (100 \cdot 300)$

x	y
100,	200)
(101,	200)
(299,	200)
(300,	200)
(200,	100)
(200,	101)
(200,	299)
(200,	300)

x is faulty.

y is faulty.

Date.
Page No.

Equivalence Class Testing.

→ Input and Output Domain is partioned into mutualy exclusive parts called equivalence classes.

→ Any one sample from a class is representative of entire class.

Steps to make a class:

and output condition.

→ Take input condition, and find valid and invalid classes.

→ Generate test cases.

eg 1 to 100

all -ve ≤ 0 | 1 ≤ n ≤ 100 | all +ve ≥ 100

Invalid. valid Invalid

Steps

1) Make → Set of Valid cases
Set of Invalid cases
For each variable.

2) Test the conditions same in BVA but use only Nominal Value with single fault assumption.

Date.
Page No.

Note - 1 If → 'if' execute then it's corresponding brace is executed. [true.

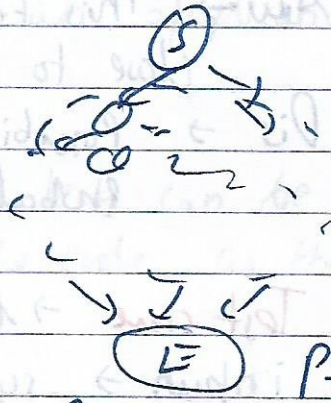
Cyclomatic

1) $V(G) = e - n + 2p$ ← no. of connected components [Usually P=1]
   - e: no. of edges
   - n: no. of nodes

2) $V(G) = \pi + 1$
   - π: predicate nodes [Having exact 2 outgoing arrows → decision node]

3) $V(G)$ = no. of region.

eg →



P = 1

As 1 start part
1 end part.

Date. _____
Page No. _____

White box testing - test cases and data are constructed based on the code that implements the software.

Functional / Behavioral
Black box testing - test cases and data are constructed based solely on software's specification.

- Testing of complexed application to determine that it provides all behaviors required of it.

~~Searches~~
- Searches for defects that are variances b/w actual operations of system and requirement.
- System is treated as a black box.

Adv → This testing type is conducted in conditions close to customer's one.

Dis → Possibility of omitting logical mistakes
Probability of redundant testing

Test case → basic component of testing.
inputs → system state, user commands and data value to be processed
expected result → visible / audible interface changes + changes in system state.

Running of test case providing input specified in test case and observing output specified in test case is called test case execution.

Date.
Page No.

Glass / White box Testing

It is the test case design method that uses control structu described as part of component level design to drive test cases.

Advantages -

1) Introspection - WBT enables to look inside the application, means that testers can identify objects programatically.

2) Stability - WBT can deliver greater stability and reusabili of test cases.

3) Thoroughness - WBT offers testers the ability to be mor thorough in terms of how much of an application they can test.

4) More efficient automated testing - unit tests can be defined that isolate particular areas of the code, as they can be tested independently.

5) Transparency of internal coding structure helps in deriving type of input data needed to test an application effectively.

6) Covers all possible paths of code thereby, empowering Software engineering team to conduct thorough appl testing.

7) Test cases can be easily automated.

8) Code optimization by revealing hidden errors.

Date.
Page No.

Disadvantages

1) Complexity - High - degree of complexity requires a much more highly skilled individual to develop a test case.

2) Changes to code will often cause white-box script to break.

3) Some conditions might be untested as it is not realistic to test every single one.

4) Is more time consuming as it need full range of inputs to test each path.

5) WBT requires more expertise and training.

WBT vs BBT.

	BBT	WBT
Definition	BBT is a software testing method in which intend structure / design / implementation of software being tested is NOT known to the tester.	WBT is a software testing method in which the intend structure / design / implementation of item being tested is known to the tester.
Application	Mainly applicable to higher levels of Testing like Acceptance testing, System Testing.	Mainly applicable to lower levels of testing like unit testing, integration testing.
Responsibility	Independent Software Tester.	Software Developer.

Date. _____
Page No. _____

Programming Knowledge.	Not Required	Required
Implementation Knowledge.	Not Required	Required.
Basis for Test Cases	Requirement Specification	Detail Design.

Date.
Page No.

Halstead

$\eta = \eta_1 + \eta_2$

η = vocabulary of program

η_1 = no. of unique operators

η_2 = no. of unique operands.

$N = N_1 + N_2$

N = program length

N_1 = total occurence of operators

N_2 = total occurence of operands.

→ For machine langue : $N = 2 * LOC$.

Volume $(V) = N * \log_2 \eta$ bits.

Potential Volume

$V^* = (2 + \eta_2^*) \log_2 (2 + \eta_2^*)$

↳ unique input/output parameters.

Length

$L = V^* / V$

Difficulty

$D = 1/L \Rightarrow V/V^*$

Effort

$E = V/L = D * V$

$E = V/\hat{L} = \frac{N \log_2 \eta \cdot \eta_1 N_2}{2\eta_2}$

Date.
Page No.

Estimated Program length

$$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

Estimated Program Level / Difficulty

$$\hat{L} = 2n_2 / (n_1 N_2)$$

$$\hat{D} = \frac{1}{\hat{L}} = \frac{n_1 N_2}{2 n_2}$$

Language Level

$$\lambda = L * V^* = L^2 V$$

Operators (n_1)	Operands (n_2)
algorithmic action. punctuation marks. + - / * while for printf → considered in pair ( ), [ ], { }, eof -- functional. switch, do, while return, continue, default break, size of GOTO. → array [ ] → . : ;	Datatype not counted. variables constants. local variable with same name labels. array name, index. 0, 1, 2 ----

Date. _____
Page No. _____

Cost Benifit Evaluation Techniques

1) Net Profit = Total Cost(Revenue) - Total Investment

2) Return of Investment (ROI) :- $\frac{\text{Average Annual Profit}}{\text{Total Investment}} * 100$

3) Average Annual Profit = $\frac{\text{Net Profit}}{\text{No. of Years}}$

4) Discount Factor = $\frac{1}{(1+d\%)^n}$ → (*100) - if in %.

5) Payback period → time taken to recover the initial investment

6) NPV (Net Profit Value) = Amount * Dicount Factor

7) IRR → discount rate at which NPV vanish
(Internal Rate of return)
NPV ≥ 0 → Project is feasible or accepted.
NPV < 0 → Project must be rejected
NPV = 0 → There is no profit/loss.

0.6@909
1
0.6091

Imp ∑ upto 4 DP. 4th one round of

Years	Cash flow	Payback	D.F(i%)	PV = Cashflow ~~Payback~~

Date. _____
Page No. _____

Payback ⇒ at which payback is positive

Take that year − Payback Cashflow

X DF

Date. _____
Page No. _____

Software Design

Design Objectives

Design fills the gap between specification and coding, taking specification, deciding how the design will be organized and methods it will use, in sufficient detail as to be directly codable.

Design needs to be:
1) Correct and complete.
2) Understandable
3) At the right level
4) Maintainable.

Principles / Characteristics of Design

Designs are -

1) A design should implement all the explicit requirements contained in the analysis model and should accomodate all the implicit requirements desired by the user.

2) A design should be readable, understandable for those who write the code, for those who test and subsequently support the software.

3) A design should a clear picture of software, addressing the data, functional and behavioural domains from an implementation perspective.

Date.
Page No.

Module Coupling

module coupling is the measure of interdependence between two modules. Higher is coupling more interdependent modules are.

1) Data Coupling
Two modules A and B are said to be data coupled if they communicate by passing of data.

2) Stamp coupling
Stamp coupling is when two modules communicate by passing of a data structure.

3) Control coupling.
Two modules communicate by passing of control informa

4) External Coupling.
External coupling occurs when a module is dependent on the module which is external to the software be developed.

5) Common Coupling.
Common Coupling is when two modules A and B share some common data - global variables.

6) Content Coupling.
When a module A changes the data of module B or pass the control information in the middle of another.

Date.
Page No.

Module Cohesion.

Cohesion is the measure of degree to which elemen of a module are functionally related.

In a procedure let there by two operations X and Y.

1) Functional Cohesion.
X and Y are part of ~~sin~~ single functional task.

2) Sequential Cohesion.
X outputs some data which becomes input for Y

3) Communicational cohesion.
X and Y operate on same inputs or contribute to same output.

4) Procedural Cohesion
X and Y both perform different operation, but yet have been combined due to the specific order in which they have to will execute.

5) Temporal Cohesion.
X and Y have to execute around same time.

6) Logical Cohesion
X and Y perform logically similar operations.

7) Coincidental Cohesion.
X and Y have not relation other than shared co

Date.
Page No.

OOPs

1) Object - Object is an entity, which has some state and operations to perform on state.

2) Messages - Objects communicate to each other by passing messages.

3) Abstraction - Abstraction is the elimination of irrelevant and amplification of essential.

4) Class - Class is a set of object with same behaviour and structure. It is a template for objects and describes their structure internally.

5) Attributes - Data held within objects.

6) Operations - Functions applied by objects in a class.

7) Inheritance - The low level classes inherit state and behaviour from high level classes.

→ State
→ operations
→ Interface of operations.

8) Polymorphism - ~~If the~~ We abstract just the interface of operation and leave the implementation to class this is called polymorphic operation and process is called Polymorphism.

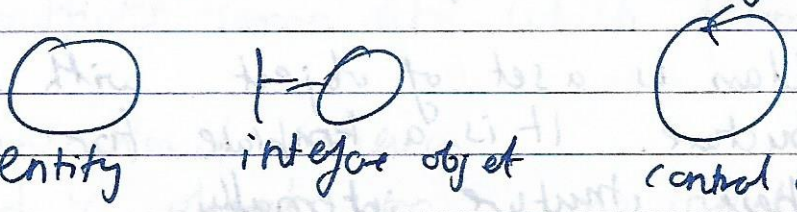
Date. _____
Page No. _____

→ Encapsulation - Process of "Information Hiding".
Ripple effect - change in one module effects other module.

→ ~~Hei~~ Hierarchy - involves organizing something in particular order or rank.

Sequence diagram

Sequence diagram is an interaction diagram which emphasizes on time ordering of messages



Class diagram

Shows relationship amongst classes.

Association - semantic relationship. It can be bi-directional or uni-directional. ——

Dependencies - One class depends on definitions of other class, unidirectional. - - - ->

Aggregations - Stronger form of associations. ◇—

Generalization - inheritance relationship b/w class ——▷

Date. _____

Page No. _____

Date. _____
Page No. _____

Category of metrics

→ Product metrics - defines the characteristics of product such as size, complexity, design features, efficiency, reliability etc.

→ Process metrics - defines the effectiveness and quality of process that produce the software product.
eg -
effort required in process
~~efficiency~~ / effectiveness of defect removal during development
number of defects found during testing
maturity of the process.

→ Project metrics - defines project characteristics and execution.
eg -
→ number of software developers
→ change in pattern of staff over the life cycle of project
→ cost and schedule
→ productivity.