Software Engineering — Unit 1

Multiple reason — unskilled, software crisis, training
→ budgets
→ cancelled

Page No: pending
Date: restarts

30% Most of projects of SE are pending.

HW cost/software cost (graph vs. years)

Moore laws processor speed, memory doubles in 2 yrs

Software failures

↗ bit error

Hardware Cost reduce drastically

- Ariane 5 (EU, rocket, suddenly 39 sec destroy)
- Y2K problem. (year wi
- Patriot missile
- XP windows

Flexible

No Silver Bullet → SE: no single technology or method will dramatically improve software productivity or reduce complex.

Software Prod: Generic, Bespoke

SE → Process of designing, developing, maintain, testing software

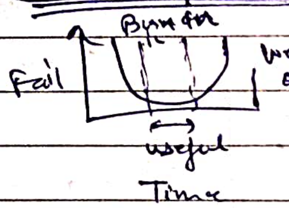
SE = P + D + OP (rqm, docu, operating procedure)

Manual: Analysis, Design, Imple, Testing

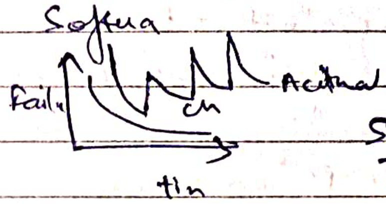
← Definition

- Maintainable
- Time Delivery
- Budget
- Requirement satisfied.

Bath Tub Curve



Software



Open Source — Source code to all

Advantages of SE

better suits for

reduce costs, improved efficiency, manual work, maintainability, productivity, security, scalability, Portable, Security, Accurate eff.

Role of Manpower: People, Product, Process, Project

Software Myths

- Management: easy to code, good tech = good success, more better engineers, complete
- Customers: more features, one time work, one time testing, SE = coding (design)
- Developer: SE = coding, TeamWork, Lots learning (Dynamic)

measurable events — Deliverables (code manual, op manual), Milestones (status, stage), Product Process, Process Metrics, Product

op → operating procedure

Productivity = Rate of O/P Per unit effort (Person Months)

↳ unit = LOC/PM (line of code/PM)

ER Diagram: Entity, Relationship (Participants), Integrity (R E), G & S, Attribute, Cardinality, Weak, Derived

Unit 2 — Software Life Cycle Models — Pg 47 SDesign

Page No.: 35 P49.
Date: 7 /

Build & Fix Model → 2 phase model
→ adhoc process
→ No prior Planning

For 200 lines
Build ⇄ Fix
unsatisfy
No structured Design.
no maintainance

Waterfall Model. (cannot go back; no risk taken)
→ not for complex
→ less flexible
→ complete requirement
→ should exist, unreal
R D I Z O
Req. us. imp nt per
D D D D

Iterative Enhancement Model (SRS)
→ less restrictive WM
→ usable product released at end of each cycle.

Process in several cycles
→ prioritize requirements
→ each cycle version.
big projects.

Incremental Process Model
→ req. defined properly
→ no confusion on any area
→ time bound process
break module.

(RAD) Rapid Development Application (flexible, Reuse, cost effective)
→ build prototype → give to user, get feedback
→ make refine improve (User)
skilled people | reduce time develop

Evolutionary Process Model
→ no product after each cycle
→ used for new / complex problems
→ unfamiliar requirements

Prototype Model
same as waterfall. but user involvement
→ linear, rapid. → experience gained
→ more cost for prototype → produced as per user
better than waterfall

Spiral Model → Risk Driven, layer (→) cost ↑. model
Planning, Risk Analysis, Develop, Ass. →
(PRDA)
Design cost, constrai | alternate factor. | Develop Test | customer
completed → review by people in project

Barry Boehm

→ **Unified Process Model** ←
IET
Inception | Elaboration | Construction | Transition
- Inception: Scope feasibi, constraints, vision, plan, prototypes, initial srs
- Elaboration: Architecture, Use case, Risks
- Construction: development, test, manuals, Product dest
- Transition: delivery, deploym, usage, Beta test

Selection of LCM
Requirements, Development team.
3. Users, ProType & Risk (RDUR)

Rational Unified Process (RUP)

working on project
higher quality maintainable s/w
Ex: iterative + incremental
• use-case driven
• architecture
• customer feedback

COCOMO

SDLC → structured Process used to design, develop, test s/w.

Unit 3 — Requirement Engineering

large document in natural language with description of what to do.

Expectations is built and better understand

Functional requirements

SRS → contract

Requirements:
- Elicitation
- Analysis
- Documentation
- Review

Requirements
1) known
2) unknown
3) undreamt
4) poor communication
5) lack of resources

① Feasibility Study ←
evaluation / analysis of impact of strategy.
risks tech, budget, business, market
User feedback.

IEEE 830

① Interviews
② Group Discuss → creative, brain storm, ideas
③ Delphi

④ FAST (facilitated application Specification Technique) → make teams & work

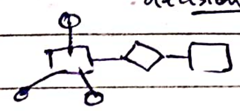
Use Case Approach.
structured outline template. User requi
(why?)

(DFD)
Data Dictionary

Use Case, Actors, Relationships, decision table

ER diagrams

EAR



not all details, only significant

(SRS) correct, unambiguous, complete, consistency, Modifiable, Traceable

Unit 5 — SOFTWARE DESIGN → Software Design Document

Design Clarity | Conceptual Design (customers) | Technical Design (System builder) | informal outline → informal formal, final? Design

A modular system consist of well defined manageable units with well defined interfaces among units.

MODULARITY

Module Coupling → measure of interdependence b/w modules

uncoupled, loosely couple, highly coupled.

Types of Module Coupling
[DS CEEC] — coext, Content, Control

B/W

Dependency b/w module A & B

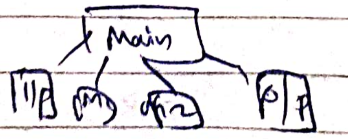
Module Cohesion: Strength of relations within modules.

functional, Sequential, Coincidental

FSCPTLC

→ High Cohesion
→ Low Coupling

Top Down Design
identify main modules, decompose it to level requir.



Function Oriented Design
system designed from function point of view.

Transaction centred structure

Object Oriented Design
function on the real data then functions.

SDD → Blue print of implementation activity

Design description

Design entity & attributes

Design Notation

- DFD
- DD
- structure charts
- Pseudocode

Page No:
Date: / /

Scope
References

SDD → Representation of software system

Design View

1. Decomposition
2. Dependency
3. Interface
4. Detail (internal)

Description (relation)

Level 0 → procedure name
1 → Parameters
2 → data structs
3 → body structure

Functional / Procedural / Layers

level 0123

attributes → properties

Operations → functions / task

OOD → class, objects, OOPS, attributes, hierarchy, operations

messages → procedure / function call

Class → blue print of objects

Inheritance → inherit properties / methods of another class

Steps for OOD model

→ see screenshot

UAI (sc d d)
CSD (d c d)

Class Diagram

Association Dependency
Generalization Aggregation

Entity Interface Control Object

Estimation

COCOMO model → predict effort cost / schedule

Constructive Cost Model

↓

Basic, Intermediate, Detailed

measured in LOC

cost estimation, resource management, risk mgm, operation,

Phases

Planning, System Design, Detailed Design, Module code & test, Integration and testing, (PSDMIC)

COCOMO

Iterative

easy error finding

DFD → visual maps that provide clear understanding

logical physical

functions of system → Rules for DFD

Data cannot flow b/w
→ two entities
→ entity & data storages
→ two data storages
→ Data flows cannot cross e.o.
→ input → Process entity → Output
atleast one

Process, Data flow,
Data Store, Ext. Entity.

Quality functions
Deployment (QFD)
Value to requirements
degree of imp

Boundary box

Software / Hardware
→ No wear tear
→ specific needs fulfilled
→ college development
→ no physical touch

- inadequate requirements
- budgets & time
- poor communication
- wrong tech,
- wrong planning } management
- no risk assessment
- no skillset

- dynamic Risk Handling
- flexible

Modularity → division into separate manageable units, brought together.

Software Quality / Attributes

Correctness	Maintainability	Secure
Usability	Portability	Modular
Efficiency	Scalable	Reusable.

Decision tables → table tells which actions to perform when → simply & clear representation

Condition, Action

Decision tree - seq & choices

Requirements Review

(SQA) software quality Assurance

→ ensure it meets quality standards
→ quality control & management, code reviews, audit, testing
→ reliability, maintain, feasibility | process monitor. | metrics

Verification	Validation
white box	black box
how work is done	end product is correct.
are we building product? right	end user requirement
requirements & standards	are we building right product!

Quality Factors || McCall
- operation · quality
- revision ·
- transition ·

SRS — Design — SDD
readable, easy, maintain, complete picture.
Interface, Architecture, Detailed (Types of Design) 3 step

Bottom up — when problem is big, so start from small.

Top Down — systematic, modular, small medium

Time & Cost

Types of estimation: Past, Base, Decomposition
- Past → after completion
- Base → rough
- Decomposition → break up → Direct, Indirect

Direct: (size oriented metric) KLOC — length of code

Indirect: function oriented metric (FP) — value of code, logical complexity

larger the product more it is complex — FP

(IO InF(EI)) i/p, o/p, inquiries, files, external interface → effort

Size = Effort × Productivity (S=EP)

Effort = Duration × Team size

Cost = Effort × Pay

$(S_O + 4S_{ML} + S_P)/6 = \text{Size}$

$EAF = 0.65 + \frac{\Sigma f_i}{100}$

FP = T.C. × EAF (estimated adjustment factor)

Empirical Model → formula for estimation using LOC / FP

COCOMO

COCOMO — basic, intermediate, advanced (for all)

classes
- organic — small, experienced, in-house
- semidetached
- embedded

[OSE]

scope, objectives, requirements

Basic COCOMO ←

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

$D_D = c_b (KLOC)^{d_b}$

Team size = D_E → effort, D_D → durati

2 tables

Page No:
Date: / /

simple straight cut estimation

effort & duration

many factors

Intermediate → $D_E = a (KLOC)^b \times EAF$ only effort

IS cost drive use of EAF

modules accurate constraints

Detailed

Cyclomatic Complexity → testing

measure complexity of pgm

no. of linearly independent path

no overhead testing effort

$V(G) = E - N + 2$ (E: edges, N: nodes)

Software Testing

$V(G) =$ (no. of bounded region $+1$) = no. of regions

Coding → translate design or algorithm into code of particular lang.

High quality & maintainable code.

Tech phase

readable, simple, modular, modify, comment

Good practices (See)

Testing → phase of SD in pgm with intention of finding bugs / fault in code.

Human → fault → failure

err Bug

Test, Debugging

difficult to test 100% acc

Condition: SRS, budget, time, duration, good team (Scope).

Principles of Testing → user requirement, limited cost, skilled team, third party, test plan, P & NP

Unit Testing → first level of testing

ind. components to work correctly.

for specific use

if bug then resolve, test case, logic

Communic

Integration Test → ind. modul combined & tested as a group

expose fault, multiple approaches

time consuming | costly | multiple teams

IT

Incremental | Big Bang

BU | TD | Sandwich. Approach.

Big Bang → all modules integrated simultaneously & tested as complete system

not big, team friendly

simple, time ↓, mod

34375
454107
676151

Top Bottom → top level modules first & lower
testing

stub are temporary modules
↳ effect of lower lvl module not yet integral

progressive testing, incompleta, test managem diffi

Page No:
Date: / /

• Bottom Up → lower level first, need (Drivers)
→ early detection
→ no stub
→ simultaneous develop

Sandwich ↓ combinat ↑ of TD BU
consumes
efficient
flexible

SYSTEM TESTING → Def complete & integrated software system tested as a whole.
to check compliance
real world environ.

FUPS (
• functional
• Usability
• Performance
• Security
• Compatibility

• User Acceptance → final testing by cst
→ Hands on experie
→ real world
user friendliness
UR

• Regression Testing
ensure previously developed software still works as expected after making changes.

Black Box / White Box Testing
↓ user requirement analysis
→ go inside system & check if actual functionality

Alpha → test done on developer's sid
→ with artificial data

Beta Testing → testing done at customer side with real data

• Stress Testing endurance
evaluate system perform when it is stressed for shorter duration.
Load more than specifid

• Boundary Value Analysis
→ errors occur at extremes.
focus on values at boundaries

Equivalent Partitioning
→ save testing effort by testing only 1 value from each partition

• Version control

• Resource Allocation
• Critical Path Management
Maintenance

Modular Architecture
→ easy to explain, code, design, document & maintain

Software Testing | Design Standards
ACIS → Scalable Performance
Archit
component modularit
Interfaces
Das

Capability Maturity Model → strategy for improving software process to make quality software
US Defence

Initial → Repeatable → Defined → Managed → Optimized
Initial ↓ no management, no experience, new
Repeatable: prev work experience
IR(DMO)
Defined ↓ (standardization) developing maintaining s/w
Rich
Managed → Goal met → predictable
Optimized ↓ • Highly experi. • Skilled

TD, BU, Delphi, Parametric.
size, info, innovation, deadline, envo
nature

improve
cbo process improvement

V Semester
B.Tech.

END SEMESTER EXAMINATION *(November-2024)*

CO301 SOFTWARE ENGINEERING

Time: 3 Hours *Max. Marks: 40*

Note: Answer any four questions.
Assume suitable missing data, if any.

Q1. Answer the following questions briefly:

a) Differentiate between Black Box Testing and White Box Testing with suitable examples. **[CO3]**

b) What are the advantages of using DFDs for requirement analysis? Mention at least three benefits and provide a brief explanation for each **[CO2]**

c) Your team is developing a mobile application for online food delivery. Which software process model would you recommend for this project? Justify your choice. **[CO5, 6]**

[4+4+2=10] [BTL 4,2,5]

Q2. a) A software team is working on a project with the following metrics:

- Total lines of code (LOC): 10,000
- Effort per KLOC: 25 person-months
- Average productivity: 200 LOC/person-month

Calculate the total effort required and the estimated number of people required to complete the project in 6 months. **[4][CO4][BTL 3]**

b) A client is unhappy with the prototype of their software application. As a project manager, how would you handle this situation? Suggest three steps you would take to resolve their concerns. **[2][CO1,6][BTL 6]**

c) What is regression testing? Why is it important in the software lifecycle? **[4][CO5][BTL 2]**

Q3. a) Define coupling and cohesion in software design. How do they impact software quality? **[5] [CO2,3] [BTL 2]**

b) Explain and analyze the distinctions and similarities between incremental development and iterative development, focusing on their methodologies, practical applications, and impact on software project outcomes. **[5] [CO1] [BTL 4]**

Q4. a) A university wants to automate its admission process. As a software engineer, draw a context-level-0 and context-level-1 DFD for this system. Include:

- External entities: Applicant, Admin Office
- Processes: Submit Application, Verify Documents, Process Fee
- Data stores: Application Records, Payment Records

[6] [CO3] [BTL 6]

b) You are managing a software development team for an e-commerce website. How would you ensure the quality of the software during its development? Mention any three strategies you would follow. **[4][CO5] [BTL 3]**

Q5. a) Define software prototyping and explain its benefits and drawbacks. **[5] [CO4] [BTL 2]**

b) List the importance of requirement engineering in the software development process? Explain its main steps. **[5] [CO4] [BTL 1]**

END