Software Project Planning & Software Design

UNIT II

 U2. 1

Learning Objectives

- **Software Project Planning**
  - Size Estimation
  - lines of Code
  - Function Count
  - Cost Estimation Models
  - COCOMO
  - COCOMO-II
  - Putnam resource allocation model
  - Risk Management.
- **Software Design**
  - Cohesion & Coupling
  - Classification of Cohesiveness & Coupling
  - Function Oriented Design
  - Object Oriented Design

 U2. 2

Software Project Planning

 U2. 3

Learning Objectives

- Characteristics of project manager
- Software Project Planning
- The Steps
- Activities during SPP
- Size Estimation
  - Lines of code (LOC)
  - Function count
- Cost Estimation
- Estimation Techniques
  - Empirical Estimation Models
  - The Constructive Cost Model (COCOMO)
  - COCOMO II
  - Putnam Resource Allocation Model

 U2. 4

Characteristics of Project Manager

A person with the ability to know what will go wrong before it actually does and the courage to estimate when the future is cloudy

 U2. 5

Software Project Planning

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

Why?

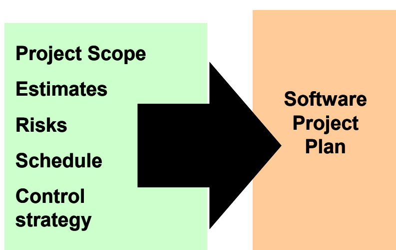
So the end result gets done on time, within budget and with quality!

 U2. 6

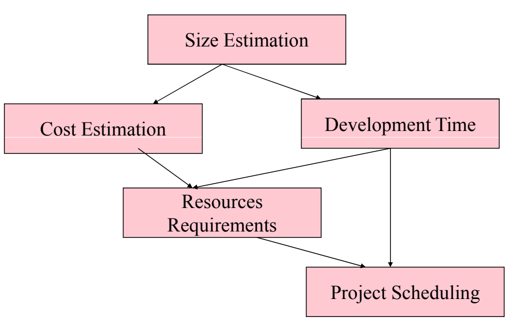
The Steps

- **Scoping**—understand the problem and the work that must be done
- **Estimation**—how much effort? how much time?
- **Risk**—what can go wrong? how can we avoid it? what can we do about it?
- **Schedule**—how do we allocate resources along the timeline? what are the milestones?
- **Control strategy**—how do we control quality? how do we control change?

Write it Down!



Activities During SPP



Size Estimation

Critical & Difficult area of the project planning

Conventional Methods (LOC/FP Approach)

compute LOC/FP using estimates of information domain values

use historical effort for the project

Lines of Code (LOC)

First measurement attempted

Easily recognizable uses different approaches

- Don't include data declarations, comments or any other lines that did not result in object code
- Include declaration and any other unexecutable statement but exclude blank lines and comments

LOC cont..

01	Int sort (int x[], int n)	13	x[I] = x[j];
02	{	14	x[j] = Save;
03	Int I, j, save, im1;	15	}
04	/* This function sorts array x in ascending order */	16	}
05	If (n <2) return 1;	17	Return 0
06	For (I=2; I<= n; I++)	18	}
07	{		
08	IM1 = I-1		
09	for(j=1; j<= im; j++)		
10	IF (X[I] < x[j])		
11	{		
12	Save = x[I];		

LOC cont..

Include all lines
- 18 LOC

When comments and blank lines are ignored
- 17 LOC

Executable statements
- 13 (5-17)

LOC cont..

Language dependent

Reflect what the system is rather than what it does

Useful in estimating programming time for a program during the build phase of a project

Usefulness is limited for functionality, complexity, efficiency etc.

Productivity Comparisons

- The lower level the language, the more productive the programmer
  - The same functionality takes more code to implement in a lower-level language than in a high-level language
- The more verbose the programmer, the higher the productivity
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code

High and Low Level Languages

Low-level language

Analysis	Design	Coding	Validation

High-level language

Analysis	Design	Coding	Validation

System Development Times

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	5weeks	5 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

Example: LOC Approach in SPP

Functions	estimated LOC	LOC/pm	$/LOC	Cost	Effort (months)
UICF	2300	315	14	32,000	7.4
2DGA	5300	220	20	107,000	24.4
3DGA	6800	220	20	136,000	30.9
DBM	3350	240	18	60,000	13.9
CGDF	4950	200	22	109,000	24.7
PCF	2100	140	28	60,000	15.2
DAM	8400	300	18	151,000	28.0
Totals	33,200			655,000	145.0

Function Count

- Introduce by Alan Albrecht while working for IBM in the 1970s
- Measures functionality from the user point of view
- Deals with the functionality being delivered

Albrecht's Function Point Analysis Principle

Decomposed into functional units

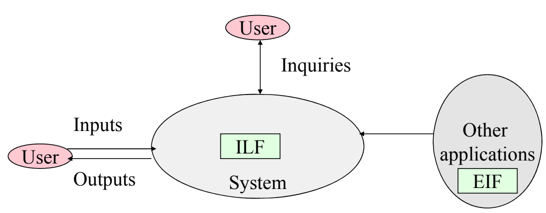
Transactional Function Types

- Inputs :
  - ✓Information entering the system
- Outputs:
  - ✓Information leaving the system
- Enquiries
  - ✓Requests for instant access to information

Data Function Types

- Internal logical files
  - ✓Information held within the system
- External interface files
  - ✓Information held by other systems that is used by the system being analyzed

FPA Functional Units



ILF: Internal Logical Files

EIF : External Interfaces

Special Features

- Independent of the language, tools or methodologies used for implementation
- Can be estimated from requirement specifications or design specifications, so possible to estimate development effort in early phases of development
- Directly linked to the statement of requirements; any change in requirements can easily be followed by a re-estimate
- Based on the system user's external view of the system, easy to understand by non-technical users
- FPs are very subjective. They depend on the estimator.
  - Automatic function-point counting is impossible

 U2. 22

Counting Function Points cont..

- Counted by considering a linear combination of five basic software components, each at one of three levels
  - Simple
  - Average
  - Complex

$$UFP = \sum_{I=1}^{5} \sum_{j=1}^{3} W_{ij} Z_{ij};$$

- Z_{ij} is the count for components i at level j
- W_{ij} Fixed weight

Also known as unadjusted function points (UFP).

 U2. 23

Counting Function Points

Software Components	Weighting Factors		
	Simple	Average	Complex
user inputs	3	4	6
user outputs	4	5	7
user inquiries	3	4	6
Internal logical files	7	10	15
external interfaces	5	7	10

 U2. 24

Counting Function Points cont..

- Final number is arrived by multiplying the UFP by an adjustment factor; determine by considering 14 aspects of processing complexity

FP = UFP * CAF

CAF; complexity adjustment factor, equal to **[0.65+0.01* $\sum F_i$]**

F_i (i = 1 to 14) ; degree of influence

- No : 0
- Incidental : 1
- Moderate : 2
- Average : 3
- Significant : 4
- Essential : 5

 U2. 25

Factors Considered

1. Does the system require reliable backup and recovery
2. Is data communication required
3. Are there distributed processing functions
4. Is performance critical
5. Will the system run in an existing heavily utilized operational environment
6. Does the require on line data entry
7. Does the the on line data entry requires the input transaction to be built over multiple screens or operations
8. Are the master files updated on line

 U2. 26

Factors Considered cont..

9. Is the inputs, outputs, files or inquiries complex
10. Is the internal processing complex
11. Is the code designed to be reusable
12. Are conversion and installation included in the design
13. Is the system designed for multiple installation in different organizations
14. Is the application designed to facilitate change and ease of use by the user

 U2. 27

Typical Function-Oriented Metrics

defects per FP; **Quality**
$ per FP; **Cost**
pages of documentation per FP; **Documentation**
FP per person-month; **Productivity**

Function Count Example

Compute the function point value for a project with the following information

Number of user inputs	= 32	**4**
Number of user outputs	= 60	**5**
Number of user inquiries	= 24	**4**
Number of files	= 08	**10**
Number of external interfaces	= 2	**7**

UFP = $\sum_{i=1}^{5}\sum_{j=1}^{3} W_{ij} Z_{ij}$ **;**

CAF= (0.65+ 0.01 $\sum F_i$**)**

FP = UFP* CAF

Assume that all complexity adjustment values and weighting factors are average. Assume that 14 algorithms have been counted

Function Count Example cont..

UFP $= \sum_{I=1}^{5} W_{ij} Z_{ij}$; J = 2

= 32*4+60*5+24*4+8*10+2*7

= 618

CAF = (0.65+ 0.01 $\sum F_i$)

= 0.65+0.01(14*3)

= 1.07

FP = UFP* CAF

= 661.26

Cost Estimation

Common attributes for any cost estimation technique

- project scope must be explicitly defined
- task and/or functional decomposition is necessary
- historical measures (metrics) are very helpful
- at least two different techniques should be used
- remember that uncertainty is inherent

Estimation Techniques

To achieve reliable cost & schedule estimates , a number of options arise:

- Delay estimation until late in project
- past (similar) project experience
- conventional estimation techniques
  - ✓ task breakdown and effort estimates
- Develop empirical models for estimation
- Acquire one or more automated estimation tools

Empirical Estimation Models

Model is concerned with the representation of the process to be estimated

Two types of model

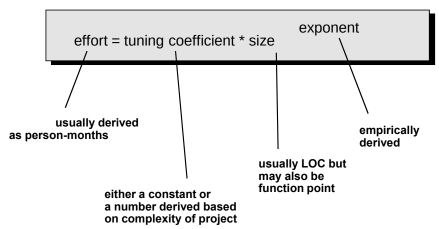
- Static
  - ✓ A unique variable (say, size) is taken as a key element for calculating all others (say, cost, time etc.)
  - ✓ The form of equation used is the same for all calculations.
- Dynamic
  - ✓ All variables are interdependent and there is no basic variable as in the static model

Empirical Estimation Models cont..

- Single variable model :
  - Use single basic variable to calculates all others
- Multivariable model:
  - Several variables are needed to describe the software development process, and selected equations combine these variables to give the estimate of time and cost
- Predicator
  - The variables, that are input to the model to predict the behavior of a software development are called predictors

Empirical Estimation Models cont..

General form:



Static, Single Variable Models

- Use an equation to estimate the desired values such as cost, time, effort etc
- Depend on the same variable used as predictor (say, size)

  $C = a L^{b}$

  C : cost

  L : size

  a & b : constants derived from the historical data of the organization
- A & b depend on the local development environment, these models are not transportable to different organizations

Static, Single Variable Models cont..

- The software Engineering Laboratory of the university of Maryland has established a model, the SEL model, for estimating its own software production

$E = 1.4\ L^{0.93}$

- Effort (person-month)

$DOC = 30.4\ L^{0.90}$

- Documentation (number of pages)

$D = 4.6\ L^{0.26}$

- Duration (month)

L : number of lines of code (thousands of lines) used as predictor

 U2. 37

Static, Multi Variable Models

- Depends on several variables representing various aspects of the software development environment, for example, method used, user participation, customer oriented changes, memory constraints etc.
- Develop by Walston and Felix at IBM

$E = 5.2\ L^{0.91}$

$D = 4.1\ L^{0.36}$

L ; source code in thousands of lines

E ; effort in person-month

D ; development in month

 U2. 38

Static, Multi Variable Models..

Productivity index uses 29 variables

$$I = \sum_{I=1}^{29} W_i X_i ;$$

W_i weight factor for i^{th} variable and X_i one of the values –1,0 or +1 depending on whether the variable decreases, has no effect, or increases the productivity respectively

 U2. 39

Example

Compare the Walston-Felix model with SEL model on a software development expected to involve 8 person-years of effort

- Calculate the number of lines of source code that can be produced
- Calculate the duration of the development
- Calculate the productivity in LOC/PY
- Calculate the average manning

SEL	Walston-Felix
$E = 1.4\ L^{0.93}$	$E = 5.2\ L^{0.91}$
$DOC = 30.4\ L^{0.90}$	$D = 4.1\ L^{0.36}$
$D = 4.6\ L^{0.26}$	

 U2. 40

Example cont..

The amount of man power involved = 8 PY = 96 person-month

a) $L = (E/a)^{1/b}$

$L\ (SEL) = (96/1.4)^{1/0.93} = 94264$ LOC

$L\ (W\text{-}F) = (96/5.2)^{1/0.91} = 24632$ LOC

b) $D\ (SEL) = 4.6(94.264)^{0.26} = 15$ months

$D\ (W\text{-}F) = 4.1(24.632)^{0.36} = 13$ months

c) P (SEL) = LOC/PY = 94264/8 = 11783 LOC/ PY

P (W-F) = LOC/PY = 24632 /8 = 3079 LOC/ PY

d) Average manning is the average number of persons required per month in the project.

M (SEL) = 96pM/15M = 6.4 persons

M (W-F) = 96pM/13M = 7.4 persons

 U2. 41

The Constructive Cost Model (COCOMO)

- one of the most widely used software estimation model in the world.
- Introduced by Barry Boehm in 1981 in his classic book "Software Engineering Economics"
- predicts the effort and schedule for a software product development based on inputs relating to the size of the software and a number of cost drivers that affect productivity.
- has three different models according to the complexity of the model:
  - the Basic Model,
  - the Intermediate Model,
  - the Detailed Model

 U2. 42

Basic Model

- Applicable to small to medium sized software projects
- Use for a quick and rough estimates
- Three modes of software development are considered
  - Organic
  - Semi-detached
  - Embedded

Organic Mode

- A small team of experienced programmers develop software in a very familiar environment
- Require little Innovation
- Size range (0-50 KLOC)

Basic Model cont..

Embedded mode

- Project has tight constraints
- Problem to be solved is unique
- Hard to find experienced persons
- Require significant Innovation
- Development environment is complex
- Size range (over 300 KLOC)

Basic Model cont..

Semi-detached mode

- An intermediate mode between the organic mode and embedded mode
- Depending on the problem at hand, the team include the mixture of experienced and less experienced people
- Require medium Innovation
- Development environment is medium
- Size range (50 - 300 KLOC)

COCOMO - Model

Develo pment Model	Project Characteristics			
	Size	Innovati on	Deadline/ Constrain ts	Developm ent Environme nt
Organic	Small	Little	Not Tight	Stable
Semi-detache d	Medium	Mediu m	Medium	Medium
Embed ded	Large	Greater	Tight	Complex hardware/ Customer Interface

Basic Model cont..

The basic COCOMO equation

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

$D = C_b(E)^{db}$

E; effort applied in Person-Months

D; development time in months

a_b, b_b , c_b, d_b ; the coefficients

Basic COCOMO Coefficients

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

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

$D = C_b(E)^{db}$

Basic COCOMO Model Example

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes I.e., organic, semidetached and embedded

The basic COCOMO equation take the form:

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

$D = C_b(E)^{db}$

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

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Nitish Pathak U2. 49

Basic COCOMO Model Example cont..

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

$D = C_b(E)^{db}$

Organic Mode $E = 2.4(400)^{1.05} = 1285.31$ PM

$D = 2.5(1285.31)^{0.38} = 38.07$ M

Semidetached Mode $E = 3.0(400)^{1.12} = 2462.79$ PM

$D = 2.5(2462.79)^{0.38} = 38.45$ M

Embedded Mode $E = 3.6(400)^{1.20} = 4772.81$ PM

$D = 2.5(2462.79)^{0.32} = 38$ M

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Nitish Pathak U2. 50

Intermediate Model

- Boehm introduces set of 15 predictors called cost drivers in the intermediate model to take account of the software development environment
- Cost drivers are used to adjust the nominal cost of a project to the actual project environment

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Nitish Pathak U2. 51

Intermediate Model cont..

- The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF).
- Typical values fro EAF range from 0.9 to 1.4
- The intermediate COCOMO equation

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

$$D = c_i\,(E)\,d_i$$

very tedious to use on a product with many components

 U2. 52

Cost Drivers

Grouped into four categories:

Product attributes

- Required software reliability (RELY)
- Database Size (DATA)
- Product Complexity (CPLX)

Computer attributes

- Execution time constraint (TIME)
- Main storage constraint (STORE)
- Virtual machine volatility (VIRT)
- Computer turnaround time (TURN)

 U2. 53

Cost Drivers cont..

Personnel attributes

- Analyst capability (ACAP)
- Application experience (AEXP)
- Programmer capability (PCAP)
- Virtual machine experience (VEXP)
- Programming language experience (LEXP)

Project attributes

- Modern programming practices (MODP)
- Use of software tools (TOOL)
- Required development schedule (SCED)

Each cost driver is rated for a given project environment using a scale very low, low, nominal, high, very high, extra high

 U2. 54

Cost Drivers Product Attributes

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.4	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product Complexity	.70	.85	1.00	1.15	1.30	1.65

Cost Drivers Computer Attributes

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.3	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility		.87	1.00	1.07	1.15	
TURN Computer turnaround time		.87	1.00	1.07	1.15	

Cost drivers Personnel Attributes

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Application experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience	1.21	1.10	1.00	.90		
LEXP Programming lang. experiences.	1.14	1.07	1.00	.95		

Cost Drivers Project Attributes

Cost Driver	Very Low	Lo w	Nominal	High	Very High	Extra High
Project Attributes						
MODP Use of modern prog. practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

 U2. 58

Intermediate Model cont..

- The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF).
- Typical values fro EAF range from 0.9 to 1.4
- The intermediate COCOMO equation
  - **$E = a_i\ (KLOC)^{b_i} * EAF$**
  - **$D = c_i\ (E)\ d_i$**
  - *very tedious to use on a product with many components*

 U2. 59

Coefficient for Intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

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

$D = c_i\ (E)\ d_i$

 U2. 60

Example

Project A is to be a 32,000 LOC semi-detached software. It is only a feasibility demonstration model (RELY=very low=0.75). Then we can estimate:

$E = a_i\,(KLOC)^{b_i} * EAF$; 3.0; 1.12

$D = c_i\,(E)d_i$; 2.5; 0.35

Effort	$= 3.0*(32)^{1.12} * 0.75 = 109$ PM
Productivity	= 32,000 LOC/109 PM = 291 LOC /PM
Schedule	$= 2.5*(109)^{0.35} = 13$ months
Average Staffing	= 109 PM/13 months = 8.4 Persons

Detailed COCOMO Model

The Detailed model differs from the intermediate model in two aspects:

- Phase-sensitive Effort Multipliers
  - ✓The effort multipliers for every cost drivers are different during the software development phases.
  - ✓ These multipliers are used to determine the amount of effort required to complete each phase.
- The Module-Subsytem-System Hierarchy
  - ✓The software product is estimated in the three level hierarchical decomposition.
  - ✓The 15 cost drivers are related to module or subsystem level.

Detailed COCOMO Model cont..

Module level cost drivers

- CPLX, PCAP, VEXP, LEXP

Subsystem level cost drivers

- RELY, DATA, TIME, STOR, …
- The subsystem level cost drivers vary from subsystem to subsystem, but tend to be the same for all the modules within a subsystem

System Level

- overall project relations such as nominal effort and schedule equations

Detailed COCOMO Model cont..

Development phases

Plan/ requirement
- 6% to 8% of the effort
- 10% to 40% of the development time

Product design
- 16% to 18% of the effort
- 19% to 38% of the development time

Programming
- 48% to 68% of the effort
- 24% to 64% of the development time

Integration/ Test
- 16% to 34% of the effort
- 18% to 34% of the development time

Detailed COCOMO Model cont..

- uses the **same equations** for estimations as the Intermediate Model
- uses a complex procedure to calculate estimation.
- procedure uses the LOCs for subsystems and modules, and module level and subsystem level effort multipliers as inputs

Detailed COCOMO Model cont..

Assume five distinct life cycle phases
- Plan & Requirement
- System Design
- Detail Design
- Module Code & Test
- Integration & Test

Efforts and schedule for each phase are assumed to be given

$Ep = \mu_p E$; $Dp = \zeta_p D$

Detailed COCOMO Model cont..

- Ignores software safety & security issues
- Ignores many hardware and customer related issues
- Silent about the involvement and responsiveness of the customer
- doesn't give proper importance to software requirements and specification phase

 U2. 67

COCOMO II

- Revised version of the original COCOMO
- Developed at University of Southern California under the leadership of Dr. Barry Boehm
- Categories of application/project identified by COCOMO II
  - End User Programming
  - Infrastructure Sector
  - Intermediate Sectors
    - ✓**Application Generators and composition aids**
    - ✓**Application composition sector**
    - ✓**System Integration**

 U2. 68

Objective of COCOMO 2

- To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's.
- To develop software cost database and tool support capabilities for continuous model improvement.
- To provide a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.

 U2. 69

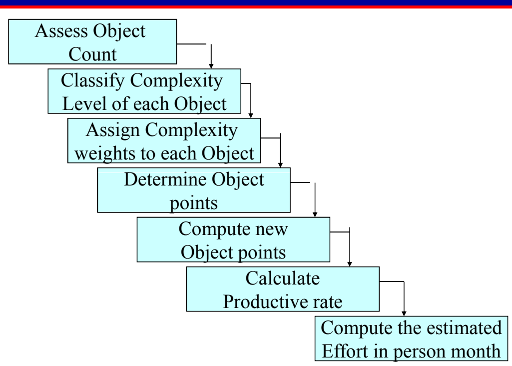
Levels of COCOMO II

- 3 level model that allows increasingly detailed estimates to be prepared as development progresses
- **Early prototyping level/ Application composition**
  - Use for Application composition projects
  - Estimates based on object points and a simple formula is used for effort estimation
- **Early design level**
  - Use for Application generators, infrastructure & system integration projects
  - Estimates based on function points that are then translated to LOC
- **Post-architecture level**
  - Use for Application generators, infrastructure & system integration projects
  - Estimates based on lines of source code

Early Prototyping/ Application Composition

- Supports prototyping projects and projects where there is extensive reuse
- Size is first estimated using object points
- Based on standard estimates of developer productivity in object points/month
- Takes CASE tool use into account
- Suitable for projects built with modern GUI-builder tools. Based on new Object Points

Steps for the Estimation of Efforts in PM



1. Access Object Counts

- estimate the number of screens, reports, and 3GL components that will comprise this application.

 U2. 73

2. Classification of Complexity Levels

Classify each object instance into simple, medium and difficult complexity levels depending on values of characteristic dimensions. Use the following scheme:

For Screens				For Reports			
Number of Views contained	# and source of data tables			Number of Sections contained	# and source of data tables		
	Total < 4 (< 2 srvr < 3 clnt)	Total < 8 (2/3 srvr 3-5 clnt)	Total 8+ (> 3 srvr > 5 clnt)		Total < 4 (< 2 srvr < 3 clnt)	Total < 8 (2/3 srvr 3-5 clnt)	Total 8+ (> 3 srvr > 5 clnt)
< 3	simple	simple	medium	0 or 1	simple	simple	medium
3 - 7	simple	medium	difficult	2 or 3	simple	medium	difficult
> 8	medium	difficult	difficult	4 +	medium	difficult	difficult

 U2. 74

3. Assign Complexity Weights to Each Object

- Weigh the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level.:

Object Type	Complexity-Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Component			10

 U2. 75

Steps 4-6

Step 4 :Determine Object-Points: add all the weighted object instances to get one number, the Object-Point count.

Step 5 : Compute new object points

- **NOP = (object points) * (100 -%reuse)/100**

Step 6 : Calculation of productivity rate

- **Productivity rate(PROD) = NOP/Person month** (*Table given*)

Step 7

Step 7 : Compute the effort in Person-Months using following scheme

✓**Effort in PM = NOP/PROD**

Developers' experience and capability	Very Low	Low	Nominal	High	Very High
ICASE maturity and capability	Very Low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

Example

- Consider a database application project with the following characteristics:
  - The application has 4 screens with 4 views each and 7 data tables for 3 servers and 4 clients
  - The application may generate two report of 6 sections each from 07 data tables for two server and 3 clients.
  - There is 10% reuse of object points
  - The developer's experience and capability in the similar environment is low. The maturity of organization in terms of capability is also low. Calculate the object point count, new object points and effort to develop such a project

Example cont..

Number of screens = 4 with 4 views each (medium)
Number of reports = 2 with 6 section each (difficult)
Object point count = 4*2+2*8 =24
NOP = 24 *(100-10)/100 = 21.6
Efforts in PM = NOP/PROD
Effort = 21.6/7 = 3.086 PM

The Early Design Model

- can use this model to get rough estimates of a project's cost and duration before you've determined it's entire architecture.
- uses a small set of new Cost Drivers, and new estimating equations. Based on Unadjusted Function Points or KSLOC
- Uses Unadjusted Function Points

The Early Design Model cont..

Base equation used

$$PM_{nominal} = A \times (Size)^B$$

- Effort is expressed as Person Months PM
- A = a constant to capture the multiplicative effects on effort with projects of increasing size , provisionally set to 2.5
- B = Scale factor;

Value of B is computed on the basis of scaling factors (or drivers) that may cause drop in productivity with increase in size

COCOMO II – Scale Drivers

The Early Design & Post Architecture Models

- •Precedentedness (PREC)
- •Development Flexibility (FLEX)
- •Architecture/Risk Resolution (RESL)
- •Team Cohesion (TEAM)
- •Process Maturity (PMAT)

 U2. 82

COCOMO II – Scale Drivers (The Early Design & Post Architecture Models)

Scale Factor	Explanation
Precedentedness (PREC)	Reflects the previous experience of the organisation with this type of project. Very low means no previous experience, Extra high means that the organisation is completely familiar with this application domain.
Development Flexibility (FLEX)	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; Extra high means that the client only sets general goals.
Architecture/Risk Resolution RESL)	Reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete and a thorough risk analysis.
Team Cohesion (TEAM)	Reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.
Process Maturity (PMAT)	Reflects the process maturity of the organisation. The computation of this value depends on the CMM Maturity Questionnaire but an estimate can be achieved by subtracting the CMM process maturity level from 5.

 U2. 83

Exponent Scaling Factor to Calculate B

Scaling factors	Very low	low	Nominal	High	Very High	Extra High
Precedentness	6.20	4.96	3.72	2.48	1.24	0.0
Development flexibility	5.07	4.05	3.04	2.03	1.01	0.0
Architecture/ Risk resolution	7.07	5.65	4.24	2.83	1.41	0.0
Team cohesion	5.48	4.38	3.29	2.19	1.10	0.0
Process Maturity	7.80	6.24	4.68	3.12	1.56	0.0

 U2. 84

Exponent Scaling Factor to Calculate B

B = 0.91+0.01 * (Sum of rating on scaling factor for the project)

Range from **0.91**(Extra High) to **1.23**(Very Low)

$$PM_{nominal} = A \times (Size)^B$$

Effort-Multiplier Cost Drivers

- COCOMO 2.0 uses a set of effort multipliers to adjust the nominal person-month estimate obtained from the project's size and exponent drivers:

$$PM_{adjusted} = PM_{nominal} \times \left(\prod_i EM_i\right)$$

Early Design Cost drivers

1. FCIL Facilities
2. PDIF Platform Difficulty
3. PERS Personnel Capability
4. PREX Personnel Experience
5. RCPX Product Reliability and Complexity
6. RUSE Required Reusability
7. SCED Required Development Schedule

Early Design Parameters

EDCD	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
RCPX	.73	.81	.98	1.0	1.30	1.74	2.38
RUSE	-	-	0.95	1.0	1.07	1.15	1.24
PDIF	-	-	0.87	1.0	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.0	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.0	0.87	0.71	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	-	1.43	1.14	1.0	1.0	1.0	-

 U2. 88

Example

A software project of application generator category with estimated 50 KLOC has to b developed. The scale factor (B) has low precedentece [4.96]; high development flexibility [2.03]; low tem cohesion [4.38] . Other factors are nominal Architeture/ Risk resolution [4.24]; Process maturity[4.68].

The early design cost drivers like platform difficult [1.29] and personal capability [0.83] are high . and others are nominal.

Calculate the effort in person months for the development of the project.

B = 0.91 +0.01 * (sum of rating on scaling factors for the project)

$PM_{nominal} = A * (size)^{B}$

$PM_{adjusted} = PM_{nominal} * (\Pi\ EMi)$

 U2. 89

Example cont..

B = 0.91 +0.01 * (sum of rating on scaling factors for the project)

= 0.91+0.01 *(4.96+2.03+4.24+4.38+4.68)

= 0.91 +0.01(20.29)

= 1.1129

$PM_{nominal} = A * (size)^{B}$

$= 2.5 * (50)^{1.1129} = 194.41$ Person month

$PM_{adjusted} = PM_{nominal} * (\Pi\ EMi)$

= 194.41 * [1.29* 0.83] = 194.41 *1.07

= 208.155 Person-month

 U2. 90

Post Architecture Cost drivers

1. ACAP Analyst Capability
2. AEXP Applications Experience
3. CPLX Product Complexity
4. DATA Database Size
5. DOCU Documentation to match lifecycle needs
6. LTEX Language and Tool Experience
7. PCAP Programmer Capability
8. PCON Personnel Continuity
9. PDIF Platform Difficulty

 U2. 91

Post Architecture Cost drivers cont..

10. PEXP Platform Experience
11. PREX Personnel Experience
12. PVOL Platform Volatility
13. RELY Required Software Reliability
14. SCED Required Development Schedule
15. STOR Main Storage Constraint
16. TIME Execution Time Constraint
17. TOOL Use of Software Tools

 U2. 92

Mapping of Cost Drivers

Early Design Cost Driver	Counterpart Combined Post-Arch. Cost Driver
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

 U2. 93

Cost Driver Rating

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
DATA		DB bytes/Pgm SLOC < 10	$10 \le D/P < 100$	$100 \le D/P < 1000$	$D/P \ge 1000$	
RUSE		none	across project	across program	across product line	across multiple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	

 U2. 94

Cost Driver Rating cont..

	Very Low	Low	Nominal	High	Very High	Extra High
TIME			≤ 50% use of available execution time	70%	85%	95%
STOR			≤ 50% use of available storage	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	

 U2. 95

Cost Driver Rating cont..

	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
AEXP	≤ 2 months	6 months	1 year	3 years	6 years	
PEXP	≤ 2 months	6 months	1 year	3 years	6 year	
LTEX	≤ 2 months	6 months	1 year	3 years	6 year	

 U2. 96

Cost Driver Rating cont..

	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	edit, code, debug	simple, frontend, backend CASE, little integration	basic lifecycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive lifecycle tools, well integrated with processes, methods, reuse	
SITE: Collocation	International	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro. area	Same building or complex	Fully collocated
SITE: Communications	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communication.	Wideband elect. comm, occasional video conf.	Interactive multimedia
SCED	75% of nominal	85%	100%	130%	160%	

 U2. 97

Post Architecture Cost Driver

CD	VL	Low	Nominal	High	VH	EH
RELY	0.75	0.88	1.00	1.15	1.39	
DATA		0.93	1.00	1.09	1.19	
CPLX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
DOCU	0.89	0.95	1.00	1.06	1.13	
TIME			1.00	1.11	1.31	1.67
STOR			1.00	1.06	1.21	1.57
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.50	1.22	1.00	0.83	0.67	
PCAP	1.37	1.16	1.00	0.87	0.74	
PCON	1.24	1.10	1.00	0.92	0.84	
AEXP	1.22	1.10	1.00	0.89	0.81	
PEXP	1.25	1.12	1.00	0.88	0.81	
LTEX	1.22	.10	1.00	0.91	0.84	
TOOL	1.24	1.12	1.00	0.86	0.72	
SITE	1.25	1.10	1.00	0.92	0.84	0.78
SCED	1.29	1.10	1.00	1.00	1.00	

 U2. 98

Schedule Estimation

B = 0.91 +0.01 * (sum of rating on scaling factors for the project)

$PM_{nominal} = A *(size)^{B}$

$PM_{adjusted} = PM_{nominal} * (\Pi\ EMi)$

$TDEV_{nominal} = [Ǿ * PM_{adjusted})^{(0.28+0.2(B-0.091)} * SCED\%/100$

Ǿ= constant, provisionally set to 3.67

$TDEV_{nominal}$ =Calendar Time in months

 U2. 99

SLOC per FP

Language	SLOC per FP
Assembler	320
Macro Assembler	213
C	150
Algol	106
Chill	106
Cobol	106
Fortran	106
Jovial	106
Pascal	91
RPG	80
PL/I	80
Modula-2	71

 U2. 100

SLOC per FP

Ada	71
Prolog	64
Lisp	64
Forth	64
Basic	64
Logo	53
4th Generation Database	40
Strategem	35
APL	32
Objective - C	26
Smalltalk	21
Query Languages	16
Spreadsheet Languages	6

 U2. 101

Putnam Resource Allocation Model

- based on an equation of staffing profiles for research and development projects (Putnam, 1978), (Londeix, 1987), (Putnam and Myers, 1992).
- Its major assumption is that the Rayleigh curve can be used to model staff levels on large (>70,000 KDSI) software projects.
  - KDSI : Kilo delivered source instructions

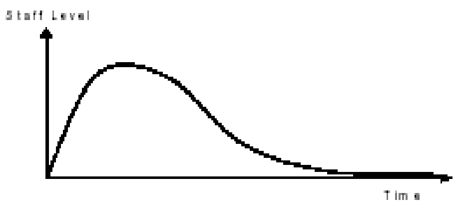
 U2. 102

Putnam Resource Allocation Model cont..

- Plotting the number of people working on a project is a function of time and a project starts with relatively few people.
- The manpower reaches a peak and falls off and the decrease in manpower during testing is slower than the earlier build up.
- Putnam assumes that the point in time when the staff level is at its peak should correspond to the project development time.
- Development effort is assumed to be 40% of the total life cycle cost.
- Putnam explicitly excludes requirements analysis and feasibility studies from the life cycle.

 U2. 103

Putnam Resource Allocation Model cont..



 U2. 104

Putnam Resource Allocation Model cont..

The basic Rayleigh form is characterized through a differential equation

$M(t) = dy/dt = 2Kat \exp(-at^2)$

- dy/dt : manpower utilization rate per unit time; staff build-up rate,
- t: elapsed time
- a : parameter, affects the shape of the curve
- K : area under the curve in the interval [0,inf..]

 U2. 105

Putnam Resource Allocation Model cont..

Y(t) :Cummulative man power used up to time t

$Y(t) = k[1 - \exp(-at^2)]$

$Y(0) = 0$

$Y(inf..) = K$

a can be obtained from peak time

$d^2y/dt^2 = 2Ka \exp(-at^2)[1 - 2at^2] = 0$

$t_d^2 = 1/2a$

$a = 1/(2 * t_d^2)$

$M(t) = dy/dt = 2Kat \exp(-at^2)$

$t_d^2 = 1/2a$

$a = 1/2 t_d^2$

- Larger the value of **a** earlier the peak time; steeper the person profile

Putnam Resource Allocation Model cont..

t_d is nearly equal to project development time

$E = y(t) = k[1 - \exp(-t_d^2/2t_d^2)]$

$= 0.39535K$

Cumulative man power required at t_d

$M(t) = dy/dt = 2Kat \exp(-at^2)$
$Y(t) = k[1 - \exp(-at^2)]$
$d^2y/dt^2 = 2Ka \exp(-at^2)[1 - 2at^2]$
$a = 1/2 t_d^2$

Putnam Resource Allocation Model cont..

Number of people involved at peak time

- $m(t) = 2Kat \exp(-at^2)$
- $m(t) = (2K/2t_d^2)t \exp(-t^2/2t_d^2)$
- $m(t) = (K/t_d^2)t \exp(-t^2/2t_d^2)$

At $t = t_d$, the peak manning, $m(t_d)$ is obtained

- $m_o = K/(t_d * \text{sqrt}(e))$
- K : total project cost (or effort) in person year
- t_d : the delivery time in years
- m_o : number of persons employed at the peak

$M(t) = dy/dt = 2Kat \exp(-at^2)$
$Y(t) = k[1 - \exp(-at^2)]$
$d^2y/dt^2 = 2Ka \exp(-at^2)[1 - 2at^2]$
$a = 1/2 t_d^2$

Difficulty Metric

The slope of the manpower distribution at start time (t =0) used to represent the difficulty denoted by D

- $m'(t)=d^2y/dt^2 = 2Ka\ exp(-at^2)[\ 1 - 2\ at^2]$
- For t = 0
- $m'(0) = 2\ Ka = 2K/\ 2t_d^2 = k\ /t_d^2$

Shows that a project is more difficult to develop when manpower demand is high or when the time schedule is short

$M(t) = dy/dt = 2Kat\ exp(-at^2)$
$Y(t) = k[1 - exp(-at^2)]$
$d^2y/dt^2 = 2Ka\ exp(-at^2)[\ 1 - 2\ at^2]$
$a = 1/2\ t_d^2$

 U2. 109

Difficulty Metric cont..

Peak manning is defined as

- $m_o = K / (t_d * sqrt(e))$

D is also related to peak manning "m_o" and the development time "t_d" by

- $D = K / t_d^2 = m_o\ sqrt(e)/t_d$

Difficult projects tend to have a higher peak manning for a given development time

$M(t) = dy/dt = 2Kat\ exp(-at^2)$
$Y(t) = k[1 - exp(-at^2)]$
$d^2y/dt^2 = 2Ka\ exp(-at^2)[\ 1 - 2\ at^2]$
$a = 1/2\ t_d^2$

 U2. 110

Manpower Buildup

D is dependent upon "K" and t_d.
$D'(t_d) = -2K/\ t_d^3$ person/year^2
$D'(K) = I/\ t_d^2$ year^{-2}

Putnam observed that the difficulty derivative relative to time played an important role in explaining the behaviour of software development.

PA: Cost = 20PY & t_d = 1Year
PB: Cost = 120PY & t_d = 2.5Year
PA : $D'(t_d) = -40$ & $D'(K) = 1$
PB : $D'(t_d) = -15.36$ & $D'(K) = 0.16$

$M(t) = dy/dt = 2Kat\ exp(-at^2)$
$Y(t) = k[1 - exp(-at^2)]$
$d^2y/dt^2 = 2Ka\ exp(-at^2)[\ 1 - 2\ at^2]$
$a = 1/2\ t_d^2$

 U2. 111

Manpower Buildup cont..

If project scale is increased the development time also increases to such an extent that the quantity K/t_d^3 remains constant around a value, which could be 8, 15 or 27; represented by D_0

$D_0 = K/t_d^3$ person/year^2

D_0 is related to the nature of s/w developed in the following way

$M(t) = dy/dt = 2Kat\ exp(-at^2)$
$Y(t) = k[1 - exp(-at^2)]$
$d^2y/dt^2 = 2Ka\ exp(-at^2)[1 - 2at^2]$
$a = 1/2\ t_d^2$

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Nitish Pathak U2. 112

Manpower Buildup cont..

- D_0 is related to the nature of s/w developed in the following way
- D_0 = 8 refers to entirely new s/w with many interfaces and interactions with other systems
- D_0 = 15 refers to new stand alone system
- D_0 = 27 refers to the s/w that is rebuilt from existing s/w
- D_0 has a strong influence on the shape of the manpower distribution.
- The larger D_0 is, the steeper manpower distribution is and the faster the necessary manpower build up will be
- So quantity D_0 is called manpower buid up

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Nitish Pathak U2. 113

Productivity Versus Difficulty

Productivity is proportional to difficulty

$P \propto D^b$

P = lines of code produced/ cumulative manpower used to produce code

$P = S/E$

S : lines of code produced

E : cumulative manpower used from t =0 to t = td (inception of the project to the delivery time)

© Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi-63, by Nitish Pathak U2. 114

Productivity Versus Difficulty cont..

According to Putnam

$p = Ó\ D^{-2/3}$

$S = Ó\ D^{-2/3}\ E$

$= Ó\ D^{-2/3}\ (0.3935\ K)$

$S = Ó[K/t_d^{2}]^{-2/3}\ K(0.3935)$

$= 0.3935\ Ó\ K^{1/3} t_d^{4/3}$

$= C\ K^{1/3} t_d^{4/3}$

$C = S\ K^{-1/3} t_d^{-4/3}$

Easy to use size, cost and development time of past projects to determine the value of C

 U2. 115

Productivity Versus Difficulty cont..

- Putnum proposed 20 values of C ranging from 610 to 57314 reflecting the effect of various factors effect on productivity
  - h/w constraint, PCPLX, PEXP, programing environment

 U2. 116

The Trade off Between Time versus Cost

$K^{1/3} t_d^{4/3} = S/C$

Raise power by 3

$Kt_d^{4} = [S/C]^3$

Kt_d^{4} is constant for a constant size software

- Compression in development time will produce an increase in manpower cost
- According to Boehm time scale should never be reduced to less than 75% of its initial calculated value

 U2. 117

Development Sub-Cycle

- Let $m_d(t)$ and y_d (t) be the design manning & the cumulative design manpower cost and can be represented as

$m_d(t) = K_d bt \exp(-bt^2)$

$y_d(t) = K_d[1-\exp(-bt^2)]$

$m_d(t)$ is non zero value for m_d at t_d

So design and coding still completing this activity after t_d
By assumption development will be completed 95% by the time t_d

 U2. 118

Development Sub-Cycle cont..

$y_d(t) / K_d = 1-\exp(-bt^2) = 0.95$
$b = 1/2\, t_{od}^2$

t_{od} is the time at which the development exhibits a peak manning

the relationship between t_d and t_{od}
$t_{od} = t_d /\text{sqrt}(6)$

Consider the relationship between K and K_d
$K_d = K/6$
Difficulty : $D = K/ t_d^2 = K_d/ t_{od}^2$
Manpower build up; $D0 = K/t_d^3 = K_d/\text{sqrt}(6)\, t_{od}^3$

 U2. 119

What we Learnt

- ✓ Characteristics of project manager
- ✓ Software Project Planning
- ✓ The Steps
- ✓ Activities during SPP
- ✓ Size Estimation
- ✓ Cost Estimation
- ✓ Estimation Techniques

 U2. 120

Risk Management

 U2. 121

Project Risks

 U2. 122

Risk Management

- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
- A risk is a probability that some adverse circumstance will occur.
  - Project risks affect schedule or resources
  - Product risks affect the quality or performance of the software being developed
  - Business risks affect the organisation developing or procuring the software

 U2. 123

Software Risks

Risk	Risk type	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule

Software Risks

Risk	Risk type	Description
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The Risk Management Activities

- **Risk Assessment**

Examining a project & identifying areas of potential risk

- Risk identification
  - Identify project, product and business risks
- Risk analysis
  - Assess the liklihood and consequences of these risks
- Risk Prioritisation
  - can be done in a quantitative way, by estimating the probability and relative loss

The Risk Management Process cont..

Risk Control

Process of managing risks to achieve desired outcomes

Risk planning

- Draw up plans to avoid or minimise the effects of the risk

Risk monitoring

- Monitor the risks throughout the project

Risk resolution

- Execution of the plans for dealing with each risk

Risk Identification

- Technology risks
- People risks
- Organisational risks
- Requirements risks
- Estimation risks

Risks and Risk Types

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components which should be reused contain defects which limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget

Risks and Risk Types

Risk type	Possible risks
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements which require major design rework are proposed. Customers fail to understand the impact of requirements changes
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

 U2. 130

Risk Analysis

- Assess probability and seriousness of each risk
- Probability may be very low, low, moderate, high or very high
- Risk effects might be catastrophic, serious, tolerable or insignificant

 U2. 131

Risk Analysis

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components which should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements which require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious

 U2. 132

Risk Analysis

Risk	Probability	Effects
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable

Risk Planning

- Consider each risk and develop a strategy to manage that risk
- **Avoidance strategies**
  - The probability that the risk will arise is reduced
- **Minimisation strategies**
  - The impact of the risk on the project or product will be reduced
- **Contingency plans**
  - If the risk arises, contingency plans are plans to deal with that risk

Risk Management Strategies

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.

Risk Management Strategies

Risk	Strategy
Requirements changes	Derive tractability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator.

 U2. 136

Risk Monitoring

- Assess each identified risks regularly to decide whether or not it is becoming less or more probable
- Also assess whether the effects of the risk have changed
- Each key risk should be discussed at management progress meetings

 U2. 137

Risk Management - Example

A company XYZ Inc. wants to developer a software to perform CAD for the home construction industry. This is the first customer of XYZ Inc. The XYZ Inc. is a new company & though they want to be the best in CAD systems, they are still, overall, a bit inexperienced.

The initial analysis of the problem leads to requirements calling for 3-major modules. The programmers hired by XYZ Inc. are among the best in the business, but most have never used C++ before, which will be the language of implementation.

The system must run fairly fast, since the user will be impatient if drawing takes too long. XYZ Inc. also would like to make their market as large as possible, which means the package should run on slower as well as faster PCs. Many models will require over time; the house simulation routines & solid modeling routines will require a great deal of memory to operate efficiently.

 U2. 138

Risk Management - Example

#	Risk	Probability	Impact	Risk Exposure	Mitigation Plan
1	Change in customer coordinator	0.50	5.0	2.5	Better status monitoring.
2	Working on new technology	0.40	8.0	3.2	Training on new technology planned.
					Develop a proof-of-concept application.
3	Availability of functional/ technical group for review	0.80	9.0	7.2	Decide on possible dates for review at least one month earlier, so that group members can plan early.
4	Frequent requirements changes	0.50	9.0	4.5	Sign-off on the initial requirements.
					Early prototyping.
5	Failure to meet performance requirements	0.50	5.0	2.5	Better design will be done to ensure that performance criteria are satisfied, the design will be reviewed by the technical group.

Recording Risk Information

Project: Embedded software for XYZ system
Risk type: schedule risk
Priority (1 low ... 5 critical): 4
Risk factor: Project completion will depend on tests which require hardware component under development. Hardware component delivery may be delayed
Probability: 60 %
Impact: Project completion will be delayed for each day that hardware is unavailable for use in software testing
Monitoring approach:
  Scheduled milestone reviews with hardware group
Contingency plan:
  Modification of testing strategy to accommodate delay using software simulation
Estimated resources: 6 additional person months beginning 7-1-96

Software Design

Learning Objectives

- Software Design
- Design Framework
- Design Principle
- Modularity
- Cohesion & Coupling
- Classification of Cohesiveness & Coupling
- Function Oriented Design
- Object Oriented Design

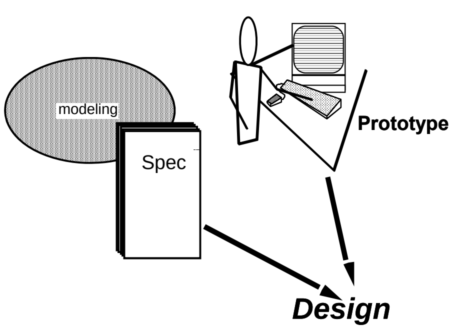
 U2. 142

Software Design

- Designer plans HOW
- Transformation of ideas into detailed implementation descriptions, with the goal of satisfying the software requirements
- Designer must satisfy both customers and system builders

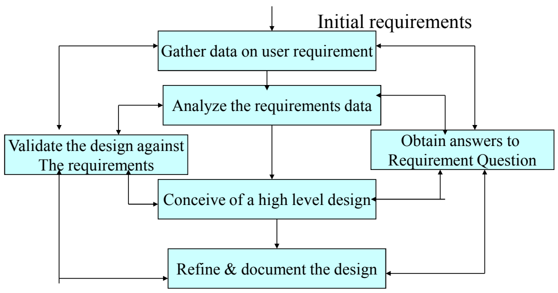
 U2. 143

Where Do We Begin?



 U2. 144

Design Framework



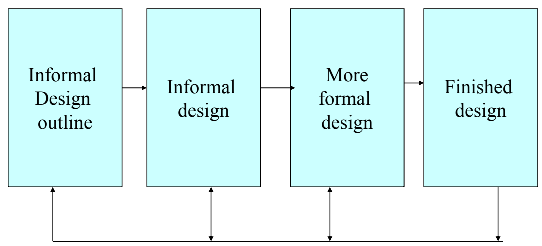
Design Principles

- should be traceable to the analysis model.
- should "minimize the intellectual distance" [DAV95] between the software and the problem as it exists in the real world.
- should exhibit uniformity and integration.
- should be structured to accommodate change.
- Design is not coding, coding is not design.
- should be reviewed to minimize conceptual (semantic) errors.

Objective of Design

- Correct & Complete
- Understandable
- At the right level
- Maintainable & to facilitate maintenance of the produced

Transformation of an Informal Design to a Detailed Design



If not Design

- Fail when small changes are made
- Difficult to maintain
- Quality cannot be assessed until late in the software process

What Features Must be Designed

- Elements of the Design
  - **Data flows, Data Stores, Processes (Activities), Procedures, Controls (Standards and guidelines for determining whether are occurring in the anticipated or accepted manner i.e. under control), Roles**
- Design of output
- Design of files
- Design of database interactions, example
  - **Data needed from the database**
  - **Actions that will affect database**
- Design of Input

What Features Must be Designed cont..

Design of Control, for example input control provides the way to
- Ensures only authorized user access
- Guarantee that transactions are acceptable
- Validate the data for accuracy
- Determine whether any necessary data have been omitted

Design of procedures
- Data entry procedures
- Run time procedures
- Error handling procedures
- Security and backup procedures

Design of program specifications

 U2. 151

Design Methodologies

- **Requirements for design techniques & methodologies.**
- Improve productivity of analyst and programmer
- Improve, documentation, maintenance and enhancements
- Cut down cost overruns and delays
- Improve communication among the user, analyst, designer and programmer
- Standardize approach to analysis and design
- Simplify design by segmentation

 U2. 152

Module

What is a module?
- lexically contiguous sequence of program statements, bounded by boundary elements, with aggregate identifier

Examples
- procedures & functions in classical PLs
- objects & methods within objects in OO PLs

 U2. 153

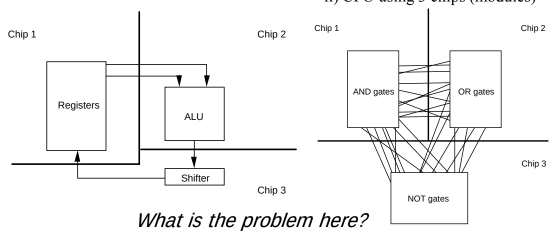
Good vs. Bad Modules

Modules in themselves are not "good"

Must design them to have good properties

i) CPU using 3 chips (modules)

ii) CPU using 3 chips (modules)



What is the problem here?

 U2. 154

Good vs. Bad Modules

Two designs functionally equivalent, but the 2nd is

- hard to understand
- hard to locate faults
- difficult to extend or enhance
- cannot be reused in another product. Implication?
- -> expensive to perform maintenance

Good modules must be like the 1st design

- maximal relationships within modules (cohesion)
- minimal relationships between modules (coupling)
- this is the main contribution of structured design

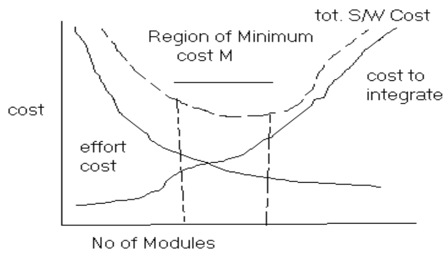
 U2. 155

Modular System

- consists of discrete components
- each component can be implemented separately
- change to one component has minimal impact on other components
- Under modularity and over modularity should be avoided

 U2. 156

Modularity cont..



Module Independence

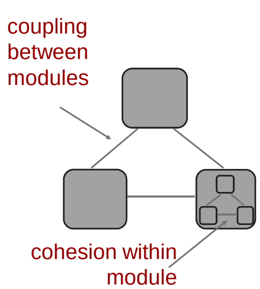
COHESION - the degree to which module performs one and only one function.

COUPLING - the degree to which module is "connected" to other modules in the system.

Cohesion & Coupling

Cohesion
- Extent to which module has a single responsibility
- Informal (Defined in terms of purpose)
- High cohesion is good

Coupling
- Extent to which modules depend on each other
- Can be defined formally
- Loose coupling is good



Coupling

- Refers strength of the relationship between modules in a system
- Loose coupling minimizes the interdependence between modules
- can be control in the following ways
  - Control the number of parameters passed between modules
  - Avoid passing unnecessary data to called module
  - Pass data only when needed
  - Maintain superior/subordinate relationship between calling and called modules

 U2. 160

Levels of coupling

5.Data Coupling
4.Stamp Coupling
3.Control Coupling
2.Common Coupling
1.Content Coupling

 U2. 161

1. Content Coupling

Def.

- one module directly references contents of the other

Example

- module **a** modifies statements of module **b**
- module **a** refers to local data of module **b** in terms of some numerical displacement within **b**
- module **a** branches into local label of module **b**

Why is this bad?

- almost any change to **b** requires changes to **a**

 U2. 162

2. Common Coupling

Def.

- two modules have write access to the same global data

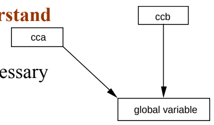
Example

- two modules have access to same database, and can both read and write same record

Why is this bad?

- resulting code is unreadable
  - ✓**modules can have side-effects**
  - ✓**must read entire module to understand**
- difficult to reuse
- module exposed to more data than necessary

```
while (global variable == 0)
    if (argument xyz > 25)
        module 3 ();
    else
        module 4 ();
```



 U2. 163

3. Control Coupling

Def.

- one module passes an element of control to the other

Example

- control-switch passed as an argument

Why is this bad?

- modules are not independent
  - ✓module **b** must know the internal structure of module **a**
  - ✓affects reusability

 U2. 164

4. Stamp Coupling

Def.

- data structure is passed as parameter, but called module operates on only some of individual components

Example

calculate withholding (employee record)

Why is this bad?

- affects understanding
  - ✓not clear, without reading entire module, which fields of record are accessed or changed
- unlikely to be reusable
  - ✓other products have to use the same higher level data structures
- passes more data than necessary
  - ✓e.g., uncontrolled data access can lead to computer crime

 U2. 165

5. Data Coupling

Def.

- every argument is either a simple argument or a data structure in which all elements are used by the called module

Example

display time of arrival (flight number)

get job with highest priority (job queue)

Slippery slope between stamp & data (see queue)

Why is this good?

- maintenance is easier
- good design has high cohesion & weak coupling

Levels of cohesion

1. Coincidental cohesion
2. Logical cohesion
3. Temporal cohesion
4. Procedural Cohesion
5. Communicational Cohesion
6. Sequential Cohesion
7. Functional Cohesion

7. Functional Cohesion

Def.

- module performs exactly one action

Examples

- **get temperature of furnace**
- **compute orbital of electron**
- **calculate sales commission**

Why is this good?

- more reusable
- corrective maintenance easier
  - ✓fault isolation
  - ✓reduced regression faults
- easier to extend product

Examples

- COMPUTE COSINE OF ANGLE
- VERIFY ALPHABETIC SYNTAX
- READ TRANSACTION RECORD
- DETERMINE CUSTOMER MORTGAGE REPAYMENT
- COMPUTE POINT OF IMPACT OF MISSILE
- CALCULATE NET EMPLOYEE SALARY
- ASSIGN SEAT TO AIRLINE CUSTOMER

 U2. 169

6. Informational Cohesion

Def.

- *one whose elements are involved in activities such that output data from one activity serves as input data to the next*

- Repaint Car Body
- CLEAN CAR BODY
- FILL IN HOLES IN CAR
- SAND CAR BODY
- APPLY PRIMER
- If add Put on final Coat

 U2. 170

Example

module format and cross-validate record
    uses raw record
    format raw record
    cross-validate fields in raw record
    returns formatted cross-validated record
  endmodule

 U2. 171

5. Communicational Cohesion

Def.

- module performs series of actions related by procedure to be followed by product, but in addition all the actions operate on same data

Why is this bad?

- still leads to less reusability -> break it up

Example

use the same input or output data.

find out some facts about a book: For instance, we may wish to

1.FIND TITLE OF BOOK

2.FIND PRICE OF BOOK

3.FIND PUBLISHER OF BOOK

4.FIND AUTHOR OF BOOK

Example

module determine customer details
  use customer account no
  find customer name
  find customer loan balance
  return customer name, customer loan balance
endmodule

4. Procedural Cohesion

Def. ?

- module performs series of actions related by procedure to be followed by product

Example

- update part number and update repair record in master db

Why is this bad?

- actions are still weakly related to one another
- not reusable

Solution

- break up!

Example

```
module write read and edit somewhat
        uses out record
        write out record
        read in record
        pad numeric fields of in record with zeros
        return in record
      endmodule
```


Example

procedurally cohesive module that averages two completely unrelated tables, **TABLE-A** and **TABLE-B**, which both just happen to have 100 elements each.

```
module compute table-A-avg and table-B-avg
    uses table-A, table-B
    returns table-A-avg, table-B-avg
      table-A-total : = 0
      table-B-total : = 0
      for i = i to 100
        add table-A (i) to table-A-total
        add table-B (i) to table-B-total
      endfor
      table-A-avg : = table-A-total/100
      table-B-avg : = table-B-total/100
  endmodule
```

3. Temporal Cohesion

Def. ?

- module performs series of actions related in time

Initialization example

open old db, new db, transaction db, print db, initialize sales district table, read first transaction record, read first old db record

Why is this bad?

- actions weakly related to one another, but strongly related to actions in other modules
- code spread out -> not maintainable or reusable

Initialization example fix

- define these intializers in the proper modules & then have an initialization module call each

Example

module initialize
updates a-counter, b-counter, items table, totals table, switch-a, switch-b
rewind tape-a
set a-counter **to** 0
rewind tape-b
set b-counter **to** 0
clear items table
clear totals table
set switch-a **to off**
set switch-b **to on**
endmodule

Example

- PUT OUT MILK BOTTLES
- PUT OUT CAT
- TURN OFF TV
- BRUSH TEETH

2. Logical Cohesion

Def.

- module performs series of related actions, one of which is selected by calling module

Example

function code = 7;

new operation (op code, dummy 1, dummy 2, dummy 3);

// dummy 1, dummy 2, and dummy 3 are dummy variables,

// not used if function code is equal to 7

Why is this bad?

- interface difficult to understand
- code for more than one action may be intertwined
- difficult to reuse

Example

- GO BY CAR
- GO BY TRAIN
- GO BY BOAT
- GO BY PLANE

1. Coincidental Cohesion

Def. ?

- module performs multiple, completely unrelated actions

Example

- module prints next line, reverses the characters of the 2nd argument, adds 7 to 3rd argument

How could this happen?

- hard organizational rules about module size

Why is this bad?

- degrades maintainability & modules are not reusable

Easy to fix. How?

- break into separate modules each performing one task

Example

- FIX CAR
- BAKE CAKE
- WALK DOG
- FILL OUT ASTRONAUT-APPLICATION FORM
- HAVE A BEER
- GET OUT OF BED
- GO TO THE MOVIES

Strategy of Design

Bottom up design

- Need to use lot of intution
- Chances Recoding is High
- Testing is easy

Top-Down Design

- Stepwise refinement
- Suitable of specifications are clear
- Testing is typical

Hybrid Design

Function-oriented Design

Design with functional units which transform inputs to outputs

Functional Design Process

Data flow design
- Model the data processing in the system using data flow diagrams

Structural decomposition
- Model how functions are decomposed into sub-functions using graphical structure charts

Detailed design
- The entities in the design and their interfaces are described in detail. These may be recorded in a data dictionary and represented as Pseudode

 U2. 187

Structural Decomposition

- Structural decomposition is concerned with developing a model of the design which shows the dynamic structure; i.e., function calls.
- The aim of the designer should be to derive design units which are highly cohesive and loosely coupled.
- In essence, a data flow diagram is converted to a structure chart

 U2. 188

Design Notations

- Data flow Diagrams
- Data Dictionaries
- Structure Charts
- Pseudocode

 U2. 189

Structure Chart

- A design tool that visually displays the relationships between modules
- Not intended to express procedural logic done by flow charts or pseudocode
- Identify the data passes existing between individual modules that interact with one another
- Assist the analyst in developing a S/W that meets the *objectives of good S/W design*

 U2. 190

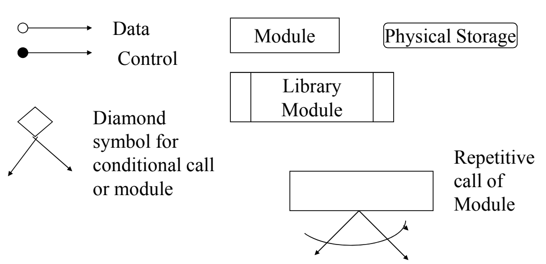
Structure Chart cont..

Notation use in Structure Chart

- Program modules identified by rectangles, with the module name inside the rectangle
- Arrows indicates calls
- Annotations indicate the parameters that are passed and the direction of the data movement
- Two types of data are transmitted
  - ✓**Parameter data** -> items of data needed in the called module
  - ✓**Control Information** (flag data)-> to assist in the control of processing by indicating the occurrence of, say, errors or end of file conditions

 U2. 191

Notations



 U2. 192

Functional Procedure Layers

Level0

- Function or procedure name
- Relationship to other system components
- Brief description of the function purpose
- Author date

Level 1

- Function parameters
- Global variables
- Routines called by the function
- Side effects
- Input/Output Assertions

Functional Procedure Layers cont..

Level2

- Local data structure
- Timing constraints
- Exception handling
- Any other limitation

Level 3

- Body

Object Oriented Design Steps

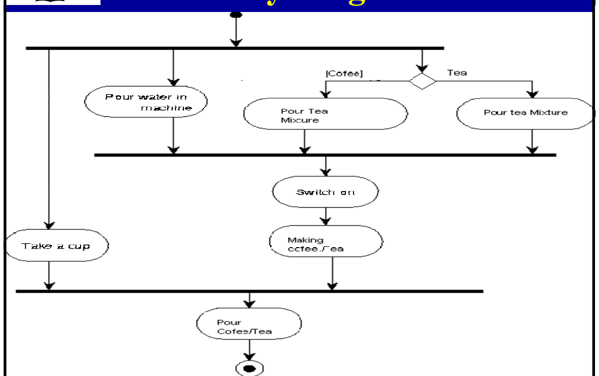
- Create Use case model
- Draw activity diagram
- Draw the interaction diagram
- Draw the class diagram
- Design of state chart diagrams
- Draw component and development diagram

Activity Diagrams cont..

- a graph in which the nodes represent activities and the arrows represent transitions between activities.
- Represents
  - Activity
  - Transition
  - Choice nodes
  - Initial state
  - final state

 U2. 196

Activity Diagrams cont..



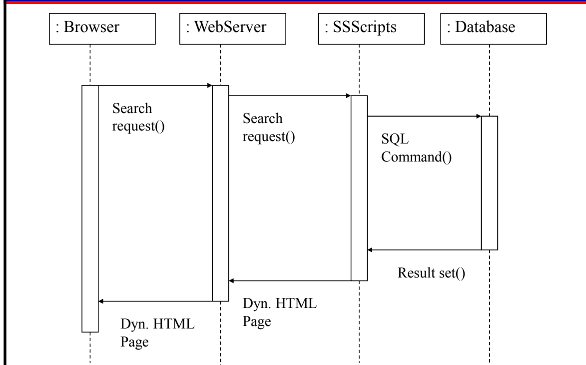
 U2. 197

Interaction Diagram

- Dynamic collaboration between objects for sequence of messages sent between them in a sequence of time
- Read from top to bottom, left to right

 U2. 198

Interaction Diagram



 U2. 199

Class Diagrams

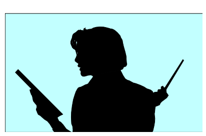
- describes the static structure of the system in terms of the classes and relationships between classes
- Classes and objects (instances of classes) are drawn with the same icon.
- Objects are distinguished from classes by underlining the name of the object.

 U2. 200

Representing Classes

A class is represented using a rectangle with compartments.

Professor
- name - employeeID : UniqueId - hireDate - status - discipline - maxLoad
+ submitFinalGrade() + acceptCourseOffering() + setMaxLoad() + takeSabbatical()

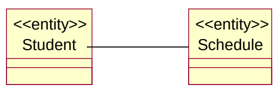


Professor Amit

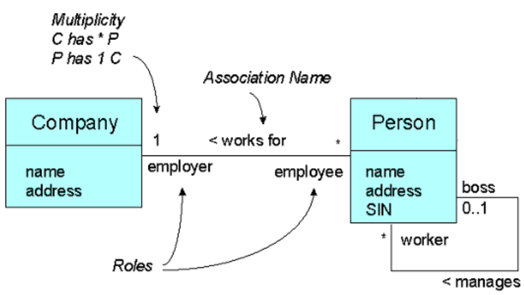
 U2. 201

Association

denote collaborations of classes.



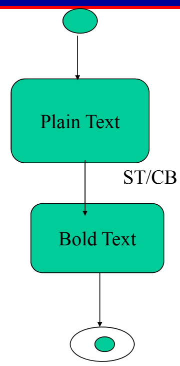
Class Diagram



State Transition Diagram

- Show all the possible states of an object and the event responsible for the state change
- Good to use when a class has complex lifecycle behaviour

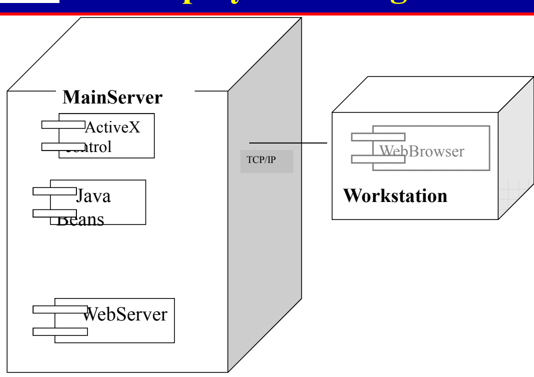
State Transition Diagram



Component and Deployment Diagram

- Component diagram are used for modeling the physical aspects of OOP
- Useful to get static view of the system
- Deployment diagram consists of node and their relationship
- A node consist of related components
- Illustrate the physical deployment of a system in actual use

Deployment Diagram



What we Learnt

- ✓ Software Design
- ✓ Design Framework
- ✓ Design Principle
- ✓ Modularity
- ✓ Cohesion & Coupling
- ✓ Classification of Cohesiveness & Coupling
- ✓ Function Oriented Design
- ✓ Object Oriented Design

UNIT II Learnings

- ✓ **Software Project Planning**
  - ✓ Size Estimation
  - ✓ Cost Estimation Models
  - ✓ Putnam resource allocation model
  - ✓ Risk Management.
- ✓ **Software Design**
  - ✓ Cohesion & Coupling
  - ✓ Classification of Cohesiveness & Coupling
  - ✓ Function Oriented Design
  - ✓ Object Oriented Design

Objective Questions

Q1. How technology factor 'C' is defined in Putnam Resource allocation model? What is its significance?

Q2. Differentiate between flow chart and structure chart.

Q3. Give at least one example for each of cohesion .The example should be either from O.S or from any of widely used software.

Q4. Discuss the categorization Empirical Estimation Models.

Q5. What are the activities during Software Project Planning?

Q6. What are the FPA functional units?

Q7. Compare DFD with ER Diagram.

Q8. What are the Risk management activities?

Q9. Define variable span with example.

Objective Questions

Q10. Which one is not an infrastructure software?

(a) Operating system (b) Database management system

(c) Compilers (d) Result management system

Q11. How many stages are in COCOMO-II?

(a) 2 (b) 3 (c) 4 (d) 5

Short Questions

Q1. Discuss difference between object oriented and function oriented design.

Q2. What problems are likely to arise if module has high complexity?

Q3. Define module cohesion. List different types of cohesion.

Q4. Can we have inheritance without polymorphism? Explain.

Q5. List points of a simplified design process.

Short Questions

Q1. Q6.You are the manager of a new project charged with developing a 100000 lines embedded system. You have a choice of hiring from two pools of developers; highly capable with very little experience in the programming language being used; or developers of low quality but a lot of experience with the programming language. What is the impact of hiring all the developers from one or the other group in terms of efforts and duration?

Q7. Consider a large-scale project for which the manpower requirement is K= 600PY and the development time is 3 years and 6 months.

- Calculate the peak manning and peak time
- What is the manpower cost after 1 year and two months?
- Calculate the difficulty and manpower build up

Long Questions

Q1. Define coupling and explain various types of coupling? Which one is best and why?

Q2. Describe the Albrecht's function count method with suitable example.

Q3. What are risk management activities? Is it possible to prioritize the risks?

Q4. What is risk exposure? What techniques can be used to control each risk?

Q5. If a module has logical cohesion, what kind of coupling is the module likely to have with others? Justify..

Q6. Explain Walston-Felix model and compare it with SEL model.

Q7.Assuming Putnam model with S=100, 000, C=5000, D0=15, compute development time td and manpower development kd.

Long Questions

Q8. Suppose that a project was estimated to be 600 KLOC. Calculate the effort and development time for each of the three models i.e., organic, semidetached and embedded.

Q9. software development requires 90 PM during total development subcycle. The development time is planned for duration of 3 yrs and 5 months.

- Calculate the manpower cost expanded until development time.
- Determine development peak time.
- Calculate the difficulty and manpower build-up.

Q10. Discuss the Categories of application/project identified by COCOMO II. Write an example and applicable level of COCOMO II estimation model for each application/project identified.

Q11. Discuss the steps for efforts estimation using COCOMOII Application Composition estimation model.

Long Questions

Q12. What are software metrics? Discuss Halistead software sciences metrics along with its limitations.

Q13. Discuss Putnam resource allocation model to derive the cumulative effort, a parameter at peak time, difficulty metric and man power build up.

Q14. The size of a software product to be developed has been estimated to be 22000 LOC. Predict the manpower cost by Waltson-Felix Model and SEL Model.

Q15. Describe various stages of COCOMO-II.

Q16. Explain with example different type of functional independence of the individual modules.

Q17. Discuss information flow metrics with its limitation. How a more sophisticated Information Flow model can overcome them?

Research Problems

Q1. You want to monitor the effort spent on different phases in the project and the time spent on different components. Design a time sheet or form to be filled in by the programmers that can be used to get this data. The design should be such that automated processing is possible.

Q2. For a student project being done in a semester course, list the major risks and risk mitigation strategy for them.

Q3. For a group student project in the software engineering course, device a suitable monitoring plan, and plans for data collection for this monitoring

 U2. 217

References

1. K. K. Aggarwal & Yogesh Singh, "Software Engineering", 2nd Ed., New Age International, 2005.
2. R. S. Pressman, "Software Engineering – A practitioner's approach", 5th Ed., McGraw Hill Int. Ed., 2001.
3. Pankaj Jalote, "An Integrated Approach to Software Engineering", Narosa, 3rd Ed., 2005.
4. Stephen R. Schach, "Classical & Object Oriented Software Engineering", IRWIN, 1996.
5. James Peter, W. Pedrycz, "Software Engineering: An Engineering Approach", John Wiley & Sons.
6. Sommerville, "Software Engineering", Addison Wesley,8th Ed., 2009.

 U2. 218

References

7. Frank Tsui and Orlando Karan, "Essentials of Software Engineering", Joes and Bartlett, 2nd Ed., 2010.
8. Kassem A. Saleh, "Software Engineering", Cengage Learning, 2009.
9. Rajib Mall, "Fundamrntal of Software Engineering", PHI, 3rd Ed., 2009.
10. Carlo Ghizzi , Mehdi Jazayeri and Dino Mandrioli, " Fundamental of Software Engineering", PHI, 2nd Ed.,2003.
11. Carol L. Hoover, Mel Rosso-Llopart and Gil Taran, "Evaluating Project Decision Case Studies in Software Engineering", Pearson, 2010.

 U2. 219