Estimation for Software Projects

Note :

1. **These slides are modified version of slides of Roger Pressman, 6th edition.**
2. **Prepared by Narayan D. G., Faculty,BVBCET Hubli.**

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, with quality!

Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources

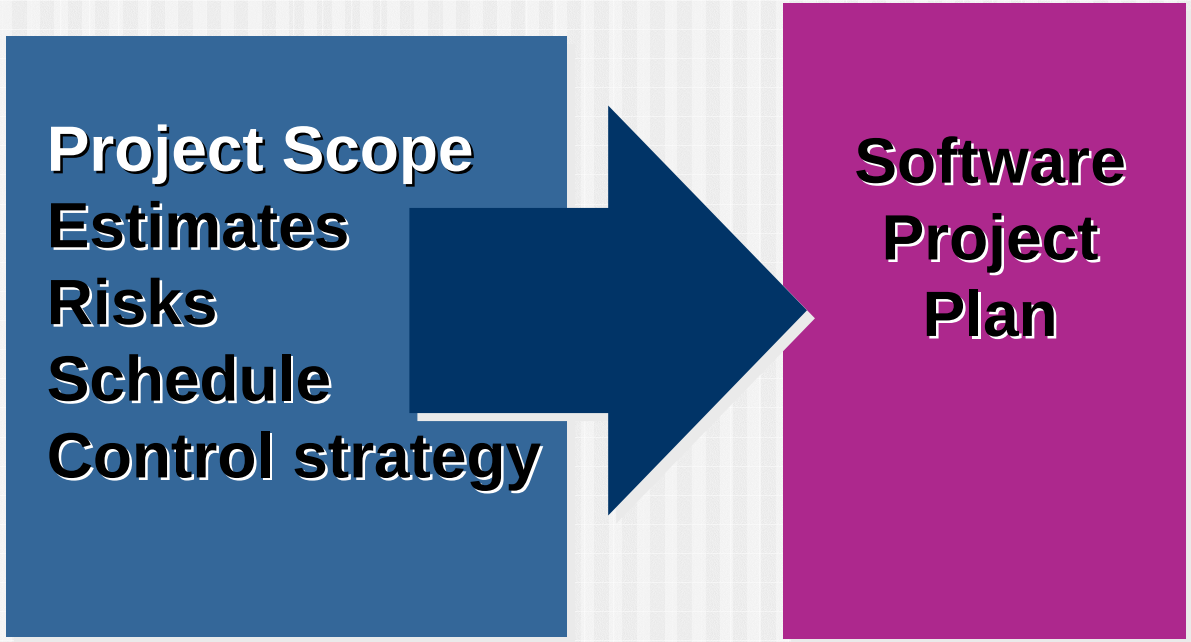
Project Planning Task Set-II

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates
- Develop a project schedule
  - Establish a meaningful task set
  - Define a task network
  - Use scheduling tools to develop a timeline chart
  - Define schedule tracking mechanisms

Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
  - experience
  - access to good historical information (metrics)
  - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

Write it Down!



To Understand Scope ...

- Understand the customers needs
- understand the business context
- understand the project boundaries
- understand the likely paths for change
- understand that ...

Even when you understand, nothing is guaranteed!

What is Scope?

- *Software scope* describes
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the “content” that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of two techniques:
  - A narrative description of software scope is developed after communication with all stakeholders.
  - A set of use-cases is developed by end-users.

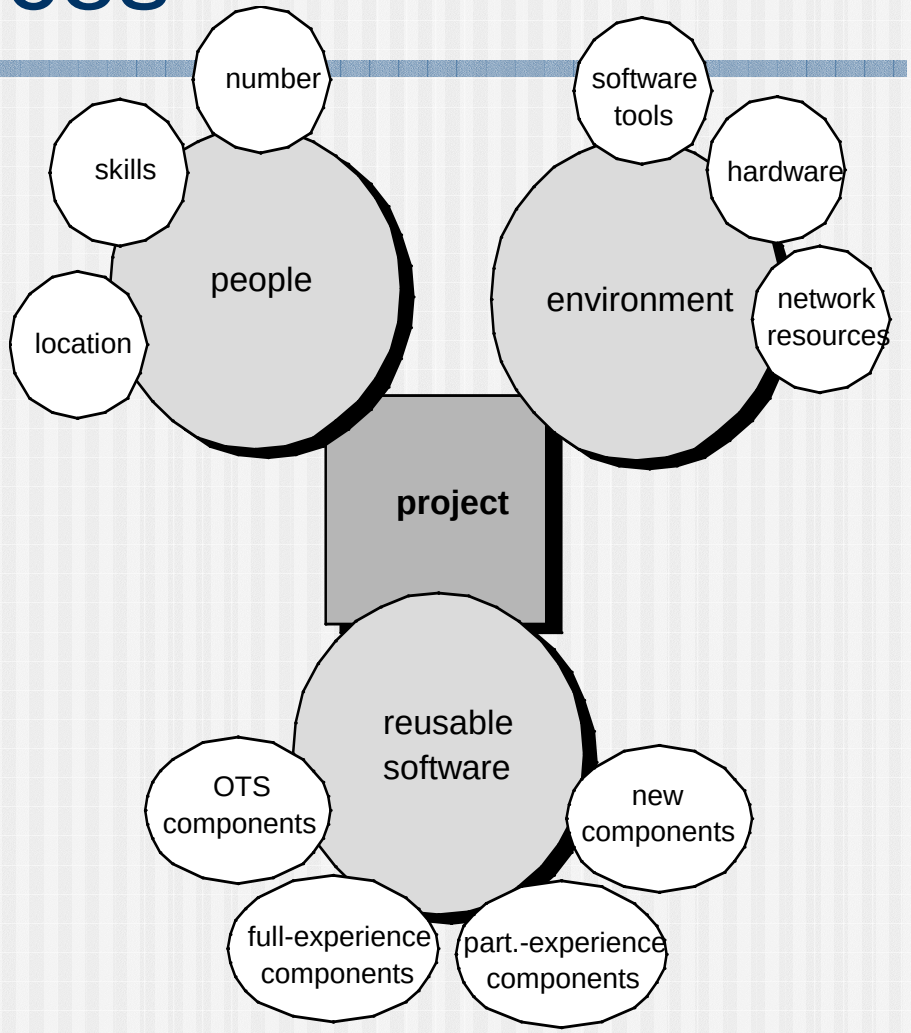
Feasibility

- After the scope is resolved, feasibility is addressed
- Software feasibility has four dimensions
  - **Technology** – Is the project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?
  - **Finance** – Is is financially feasible? Can development be completed at a cost that the software organization, its client, or the market can afford?
  - **Time** – Will the project's time-to-market beat the competition?
  - **Resources** – Does the software organization have the resources needed to succeed in doing the project?

Estimation: Resources

- Three major categories of software engineering resources
  - Human Resources
  - Environmental resources
  - Reusable software resources
    - Often neglected during planning but become a paramount concern during the construction phase of the software process
- Each resource is specified with
  - A description of the resource
  - A statement of availability
  - The time when the resource will be required
  - The duration of time that the resource will be applied

Resources


number
skills
people
location
software tools
hardware
environment
network resources
project
reusable software
OTS components
new components
full-experience components
part.-experience components

Human Resources

- Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

12

Development Environment Resources

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products
- Most software organizations have many projects that require access to the SEE provided by the organization
- Planners must identify the time window required for hardware and software and verify that these resources will be available

13

Reusable Software Resources

- **Off-the-shelf components**
  - **Components are from a third party or were developed for a previous project**
- **Full-experience components**
  - **Components are similar to the software that needs to be built**
  - **Software team has full experience in the application area of these components**
- **Partial-experience components**
  - **Components are related somehow to the software that needs to be built but will require substantial modification**
  - **Software team has only limited experience in the application area of these components**
- **New components**
  - **Components must be built from scratch by the software team specifically for the needs of the current project**
  - **Software team has no practical experience in the application area**

14

Project Estimation

- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process

Project Estimation Options

- Options for achieving reliable cost and effort estimates
  1) Delay estimation until late in the project (we should be able to achieve 100% accurate estimates after the project is complete)
  2) Base estimates on similar projects that have already been completed
  3) Use relatively simple decomposition techniques to generate project cost and effort estimates
  4) Use one or more empirical estimation models for software cost and effort estimation
- Option #1 is not practical, but results in good numbers
- Option #2 can work reasonably well, but it also relies on other project influences being roughly equivalent
- Options #3 and #4 can be done in tandem to cross check each other

16

Project Estimation Approaches

- Decomposition techniques
  - These take a "divide and conquer" approach
  - Cost and effort estimation are performed in a stepwise fashion by breaking down a project into major functions and related software engineering activities
- Empirical estimation models
  - Offer a potentially valuable estimation approach if the historical data used to seed the estimate is good

17

Decomposition Techniques

Introduction

- Before an estimate can be made and decomposition techniques applied, the planner must
  - Understand the scope of the software to be built
  - Generate an estimate of the software's size
- Then one of two approaches are used
  - Problem-based estimation
    - Based on either source lines of code or function point estimates
  - Process-based estimation
    - Based on the effort required to accomplish each task

19

Problem-Based Estimation

1) Start with a bounded statement of scope

2) Decompose the software into problem functions that can each be estimated individually

3) Compute an LOC or FP value for each function

4) Derive cost or effort estimates by applying the LOC or FP values .

5) Combine function estimates to produce an overall estimate for the entire project

20

Problem-Based Estimation (continued)

- For both approaches, the planner uses lessons learned to estimate an optimistic, most likely, and pessimistic size value for each function or count (for each information domain value)

- Then the expected size value S is computed as follows:

$$S = (S_{opt} + 4S_m + S_{pess})/6$$

- Historical LOC or FP data is then compared to S in order to cross-check it

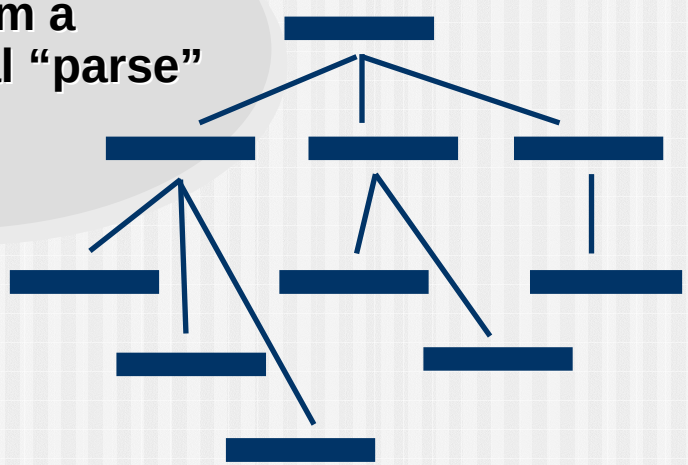
21

Functional Decomposition

Statement of Scope

Perform a Grammatical “parse”

functional decomposition



Example: LOC Approach

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate =$8000 per month, the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is **$431,000 and the estimated effort is 54 person-months.**

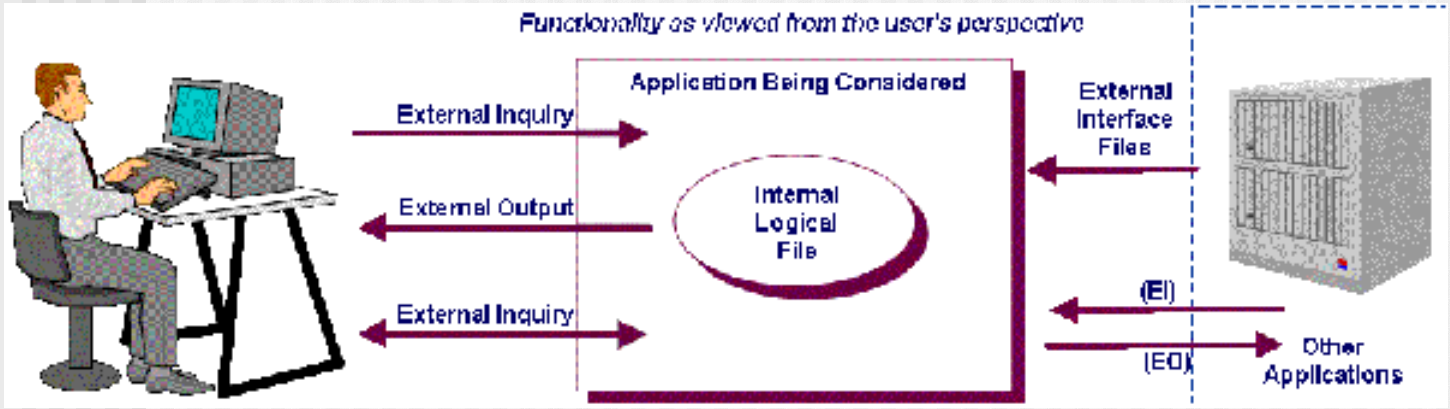
Function Point Analysis

- Function Point Analysis has been proven as a reliable method for measuring the size of computer software, estimating projects, managing change of scope and measuring productivity.
- This method uses the requirements specification to assess inputs, outputs, file accesses, user interactions and interfaces and calculates the size based on these.
- Based on a combination of program characteristics
  - External inputs (I)
  - External outputs (O)
  - User interactions/enquiries (E)
  - Internal logical files used by the system (L)
  - External interfaces to other applications (F)
- Each of these is individually assessed for complexity and given a weighting value that varies from 3 (for simple external inputs) to 15 for complex internal files.

Function points…

ITEM	WEIGHTING FACTOR		
	simple	average	complex
External input	3	4	6
External output	4	5	7
User inquiry	3	5	6
External file	7	10	15
Internal file	5	7	10

Function Point Analysis



Example: CAD software

FIGURE 23.3
Estimating information domain values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						*320*

$FP_{estimated}$ = count-total * [0.65 + 0.01 * S (F_i)]

$FP_{estimated}$ = 375

organizational average productivity = 6.5 FP/pm.

burdened labor rate = $8000 per month, approximately $1230/FP.

Based on the FP estimate and the historical productivity data, **total estimated project cost is $461,000 and estimated effort is 58 person-months.**

Example….

Each of the complexity weighting factors is estimated and the value adjustment factor is computed as described in Chapter 15:

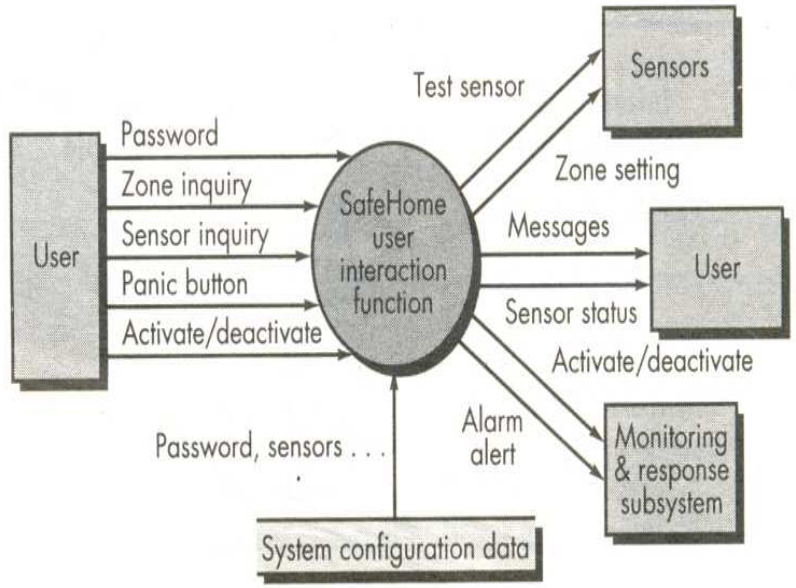
Factor	Value
1. Backup and recovery	4
2. Data communications	2
3. Distributed processing	0
4. Performance critical	4
5. Existing operating environment	3
6. On-line data entry	4
7. Input transaction over multiple screens	5
8. ILFs updated online	3
9. Information domain values complex	5
10. Internal processing complex	5
11. Code designed for reuse	4
12. Conversion/installation in design	3
13. Multiple installations	5
14. Application designed for change	5
Value adjustment factor	**1.17**

Finally, the estimated number of FP is [illegible]

Example 2:

FIGURE 15.3

A data flow model for *SafeHome* software



Example 2….

FIGURE 15.4 Computing function points

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total							50

Example 2….

Each of the complexity weighting factors is estimated and the value adjustment factor is computed as described in Chapter 15:

Factor	Value
1. Backup and recovery	4
2. Data communications	2
3. Distributed processing	0
4. Performance critical	4
5. Existing operating environment	3
6. On-line data entry	4
7. Input transaction over multiple screens	5
8. ILFs updated online	3
9. Information domain values complex	5
10. Internal processing complex	5
11. Code designed for reuse	4
12. Conversion/installation in design	3
13. Multiple installations	5
14. Application designed for change	5
Value adjustment factor	**1.17**

Finally, the estimated number of FP is [illegible]

Example 2….

- The estimated number of FP is derived:

  FPestimated = count-total * [0.65 + 0.01 * S (Fi)]

  = 50 X [.65 + .01 * 46]

  FPestimated = 56

Eample 3…

Exercise:

The specification for a spelling checker program is as follows:

> *The checker accepts two external inputs: a document file name and a personal dictionary file name; these refer to the two external files used by the system. The major output of the checker is a list of misspelt words, i.e. all words not contained in either the dictionary or personal dictionary files. Two other outputs are a 'number of words processed' message and 'number of words used from personal dictionary' message. At any point in the checking process, the user can query the number of words processed and the number of spelling errors. The standard dictionary file used by the checker is considered an internal logical file.*

- The complexity of each data type is rated average, except 'standard dictionary file' and 'list of misspelt words' which are considered complex.

Function points…

1. calculate the unadjusted function point value

ITEM	WEIGHTING FACTOR		
	simple	average	complex
External input	3	4	6
External output	4	5	7
User inquiry	3	4	6
External file	7	10	15
Internal file	5	7	10

2. Calculate an adjusted function point value where DI is 28.

Process-Based Estimation

1) Identify the set of functions that the software needs to perform as obtained from the project scope

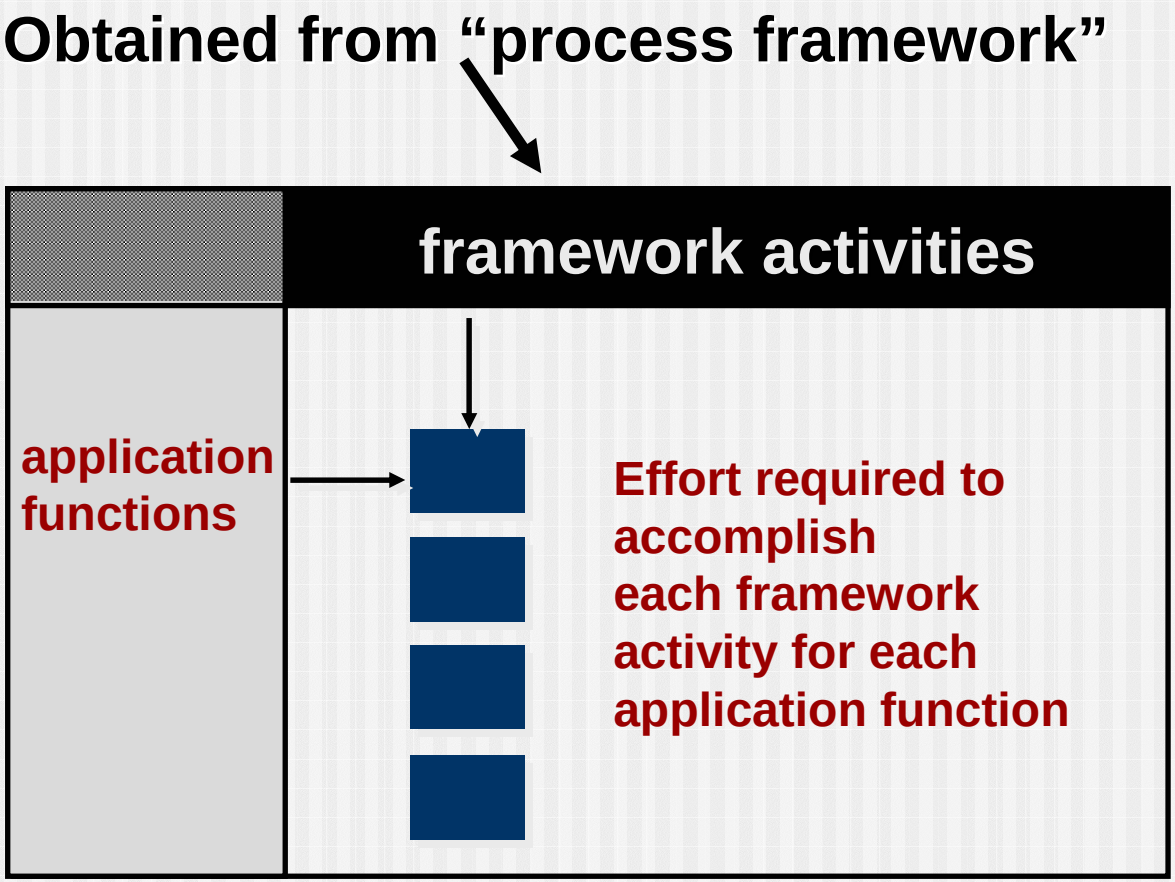
2) Identify the series of framework activities that need to be performed for each function

3) Estimate the effort (in person months) that will be required to accomplish each software process activity for each function

(More on next slide)

35

Process-Based Estimation



Process-Based Estimation Example

Activity →	CC	Planning	Risk Analysis	Engineering		Construction Release		CE	Totals
Task →				analysis	design	code	test		
Function									
↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DSM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

Based on an average burdened labor rate of $8,000 per month, **the total estimated project cost is $368,000 and the estimated effort is 46 person-months.**

Estimation with use-cases

- Use-cases provide insight into software scope and requirements
- However, developing an estimation approach with use-cases is problematic:
  - There is no standard form for use-cases
  - They represent an external view (the user's view) of the software, and vary in levels of abstraction
  - Use-cases do not address the complexity of a feature
  - Use-cases do not describe complex interactions between many functions and features

Estimation with use-cases

- One person's "use-case" could require months of effort, another just a day or two
- Estimation using use-cases has been investigated, but no proven method has emerged to date.
- Smith suggests a method for using use-cases, but in a strict, hierarchical manner

Use-case estimation

- A structural hierarchy is described for the project
- Any level of the hierarchy can be described by no more than 10 use-cases
- Each of the use-cases would encompass no more than 30 distinct scenarios

Use-case estimation

- Use-cases for a large system are described at a much higher level of abstraction then use-cases for individual sub-systems
- Before use-cases can be used:
  - The level within the structural hierarchy is established
  - Average length (in pages) of each use-case is determined
  - The type of software (business, scientific, etc) is defined
  - Rough architecture for the system is considered

Use-case estimation

- Once those characteristics are established, empirical data can be used to determine estimated LOC or FP per each use-case for each level of the hierarchy
- Historical data are then used to calculate the effort required to develop the system

Sample use-case estimation

$$LOC = N \times LOC_{avg} + [(S_a/S_h - 1) + (P_a/P_h - 1)] \times LOC_{adjust}$$

N = actual number of use-cases

LOC_{avg} = historical average LOC per use-case for this type of system

S_a = actual scenarios per use-case

S_h = average scenarios per use-case for this type of system

P_a = actual pages per use-case

P_h = average pages per use-case for this type of system

LOC_{adjust} = represents an adjustment based on *n* percent of LOCavg where *n* is defined locally.

Example:

List of Actual use cases

- Total Use cases – 6
- Scenarios – 10
- Pages – 6
- Loc adjust – 30 %

List of Average Use Cases values

- Scenarios – 12
- Pages – 5
- LOC – 560

LOC = 6 x560 [(10/12 -1) + (6/5 – 1)] x 168

LOC = 3366

Estimation with Use-Cases

	use cases	scenarios	pages	Ź	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	Ź	12	5	560	3,366
Engineering subsystem group	10	20	8	Ź	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	Ź	10	6	1650	7,970
				Ź	Ź	Ź	Ź	
Total LOC estimate				Ź	Ź	Ź	Ź	42,568

Using 620 LOC/pm as the average productivity for systems of this type and a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the use-case estimate and the historical productivity data, **the total estimated project cost is $552,000 and the estimated effort is 68 person-months.**

Reconciling estimates

- The various estimation methods encountered result in multiple estimates which must be reconciled
- The goal of a reconciliation process is to produce a single estimate of effort, project duration, or cost
- “Complicated methods might not yield a more accurate estimate, particularly when developers can incorporate their own intuition into the estimate.” -- Philip Johnson, et al.

Empirical Estimation Models

- An estimation model uses empirically derived formulas to predict effort as a function of LOC or FP
- The empirical data that support most estimation models are derived from a limited sample of projects

The Structure of Estimation Models

- $E = A + B \times (ev)^c$
  - E – effort in person-months
  - A, B, C are empirically derived constants
  - ev is the estimation variable (either LOC or FP)
- Examples of LOC-oriented
  - $E = 5.2 \times (KLOC)^{0.91}$ - Walston-Flix model
  - $E = 5.5 + 0.73 \times (KLOC)^{1.16}$ – Bailey-Basili model
  - $E = 2.4 \times (KLOC)^{1.05}$ – Boehm simple model
  - $E = 5.288 \times (KLOC)^{1.047}$ – Doty model for KOCL > 9

- Examples of FP-oriented
  - E = -13.39 + 0.0545 FP
    - – Albrech and Gaffney model
  - $E = 60.62 \times 7.728 \times 10^{-8} FP^3$
    - – Kemerer model
  - E = 585.7 + 15.12 FP
    - – Matson, Barnett, and Mellichamp model

The COCOMO model

- An empirical model based on project experience
- Well-documented, ‘independent’ model which is not tied to a specific software vendor
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 81

Project complexity	Formula	Description
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

The COCOMO II Model (1)

- ***CO**nstructive **CO**st **Mo**del*
- A hierarchy of estimation models
- Addresses the following areas

  Application composition model - Used during the early stages of software engineering when the following are important
  - Prototyping of user interfaces
  - Consideration of software and system interaction
  - Assessment of performance
  - Evaluation of technology maturity

  Early design stage model – Used once requirements have been stabilized and basic software architecture has been established

  Post-architecture stage model – Used during the construction of the software

- Three different sizing options are available
  - Object points
  - Function points
  - Lines of source code

COCOMO II Early prototyping level

- Also called as application composition model
- Suitable for projects built using modern GUI-builder tools
  - Based on **Object-Points**
- Supports prototyping projects and projects where there is extensive reuse
- Based on standard estimates of **developer productivity** in object points/month
- Takes CASE tool use into account

The COCOMO II Model

- Object point is computed using counts of the number of
  - Screens (at the user interface)
  - Reports
  - Components likely to be required to build the application

The COCOMO II Model (3)

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	10	10	10

Complexity weighting for object types

The COCOMO II Model (4)

Developer's experience/capability And Environment maturity/capability	Very low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

Productivity rate for object points

The COCOMO II Model (5)

- NOP = (object points) × [(100-%reuse)/100]
  - NOP = New Object Points
  - Object Points = Weighted Total
  - %reuse = Percent of reuse
- Estimated effort = NOP/PROD
  - PROD = Productivity Rate
  - PROD = NOP/person-month

Example:

- Assessment of a software system shows that:
  - ●The system includes
    - 6 screens: 2 simple + 3 medium + 1 difficult
    - 3 reports: 2 medium + 1 difficult
    - 2 3GL components
  - 30 % of the objects could be supplied from previously developed components
  - Productivity is high
- Compute the estimated effort PM ‘Person-months’ needed to develop the system.

Example…

- Object counts:

  2 simple screens x 1 = 2
  3 medium screens x 2 = 6
  1 difficult screen x 3 = 3
  2 medium reports x 5 = 10
  1 difficult report x 8 = 8
  2 3GL components x10= 20

Total OP = 49

Example …

- NOP = (object points) × [(100-%reuse)/100]
  - NOP = New Object Points
  - Object Points = Weighted Total
  - %reuse = Percent of reuse

  NOP = 49 * [(100-30)/100]

  = 35

- Estimated effort = NOP/PROD
  - PROD = Productivity Rate
  - PROD = NOP/person-month

  E= 35 / 25

  = 1.4 PM

The Software Equation

- Suggested by Putnam & Myers
- It is a multivariable model
- It assumes a specific distribution of effort over life of software project
- It has been derived from productivity data collected for over 4000 modern-day software projects
- $\mathbf{E = [LOC \times B^{0.333} / P]^3 \times (1/t^4)}$

The Software Equation

- **$E = [LOC \times B^{0.333} / P]^3 \times (1/t^4)$**
  - E = effort in person-months or person-years
  - B = special skills factor
  - P = productivity factor
  - t = project duration (months or years)
- **P** reflects
  - Overall process maturity
  - Management practices
  - Extent to which good s/w engineering practices are used
  - Level of Programming Languages used
  - State of Software environment
  - Skills & experience of team
  - Application complexity

The Software Equation

- Typical values of P (2,000-12,000 typical)
  - P= 2000 - for a real-time embedded software
  - P= 10,000 - for telecomm. & systems software
  - P= 28,000 for business applications
- Value of **B**
  - increases slowly as “the need for integration, testing, quality assurance, documentation and management skills grows”.
  - For small programs (KLOC=5 to 15), B= 0.16, for larger programs (KLOC=more than 70), B=0.39

The Software Equation

- Software equation has two independent parameters
  - LOC
  - t
- Minimum development time equations derived from software equation
  - $\mathbf{t_{min}} = 8.14\ (LOC/P)^{0.43}$
    - in months for $t_{min} > 6$ months
  - $\mathbf{E} = 180\ Bt^3$
    - In person-months for E>= 20 person-months
    - t is represented in years

Example

- For
  - LOC = 33200
  - P = 12000
  - B = 0.28
- Then
  - **t_{min} = 12.6 months**
  - **E = 180 $\times$ 0.28 x (12.6/12)^3**
  - **= 58 person-months**

Example2:

- The LOC is 6450, P=3000. Calculate the effort required.

  24 PM

Estimation for OO Projects-I

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented requirements modeling (Chapter 6), develop use-cases and determine a count.
- From the analysis model, determine the number of key classes (called analysis classes in Chapter 6).
- Categorize the type of interface for the application and develop a multiplier for support classes:

Interface type	**Multiplier**
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

Estimation for OO Projects-II

- Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
- Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
- Cross check the class-based estimate by multiplying the average number of work-units per use-case