

**Software Engineering**

**Software Cost Estimation**

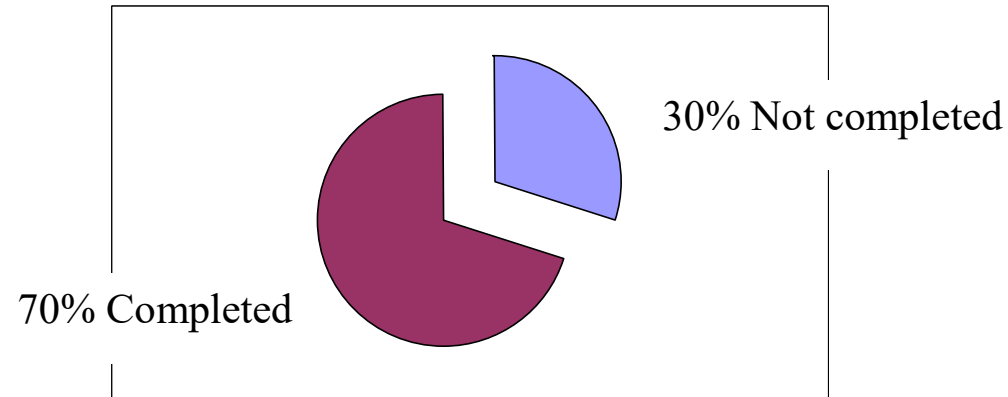
# Objectives

- To introduce the fundamentals of software costing and pricing
- To explain software productivity metric
- To explain why different techniques for software estimation:
  - LOC model
  - Function points model
  - Object point model
  - COCOMO (CONstructive COst MOdel): 2 algorithmic cost estimation model
  - UCP: Use Case Points

# What is Software Cost Estimation

- Predicting the cost of resources required for a software development process

# Software is a Risky Business



- 53% of projects cost almost 200% of original estimate.
- Estimated \$81 billion spent on failed U.S. projects in 1995.
  - All surveyed projects used waterfall lifecycle.

# Software is a Risky Business

- British Computer Society (BCS) survey:
  - 1027 projects
  - Only 130 were successful !
- Success was defined as:
  - deliver all system **requirements**
  - within **budget**
  - within **time**
  - to the **quality** agreed on

# Why **early** Cost Estimation?

- Cost estimation is needed **early** for s/w pricing
- $S/W \text{ price} = \text{cost} + \text{profit}$

# Fundamental estimation questions

- **Effort**
  - How much effort is required to complete an activity?
  - Units: man-day (person-day), man-week, man-month,,...
- **Duration**
  - How much **calendar time** is needed to complete an activity? Resources assigned
  - Units: hour, day, week, month, year,...
- **Cost of an activity**
  - What is the total cost of an activity?
- Project estimation and scheduling are interleaved management activities

# Software Cost Components

1. Effort costs (dominant factor in most projects)
  - salaries
  - Social and insurance & benefits
2. Tools costs: Hardware and software for development
  - Depreciation on relatively small # of years 300K US\$
3. Travel and Training costs (for particular client)
4. Overheads(OH): Costs must take overheads into account
  - costs of building, air-conditioning, heating, lighting
  - costs of networking and communications (tel, fax, )
  - costs of shared facilities (e.g library, staff restaurant, etc.)
  - depreciation costs of assets
  - Activity Based Costing (ABC)



# S/W Pricing Policy

S/W price is influenced by

- economic consideration
- political consideration
- and business consideration

# Software Pricing Policy/Factors

Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

# Programmer Productivity

- Rate of s/w production
  - Needs for measurements
  - Measure software produced per time unit (Ex: LOC/hr)
    - rate of s/w production
    - software produced including documentation
- Not quality-oriented: although quality assurance is a factor in productivity assessment

# Productivity measures

S/W productivity measures are based on:

- Size related measures:
  - Based on some output from the software process
  - Number lines of delivered source code (LOC)
- Function-related measures
  - based on an estimate of the functionality of the delivered software:
    - Function-points (are the best known of this type of measure)
    - Object-points
    - UCP

# Measurement problems

- Estimating the size of the measure
- Estimating the total number of programmer-months which have elapsed
- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate

# Lines Of Code (LOC)

- Program length (LOC) can be used to predict program characteristics e.g. person-month effort and ease of maintenance
- What's a line of code?
  - The measure was first proposed when programs were typed on cards with one line per card
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line?
- What programs should be counted as part of the system?
- Assumes linear relationship between system size and volume of documentation

# Versions of LOC

- DSI : Delivered Source Instructions
- KLOC Thousands of LOC
- DSI
  - One instruction is one LOC
  - Declarations are counted
  - Comments are not counted

# LOC

- Advantages
  - Simple to measure
- Disadvantages
  - Defined on code: it can not measure the size of specification
  - Based on one specific view of size: length.. What about complexity and functionality !!
  - Bad s/w may yield more LOC
  - Language dependent
- Therefore: Other s/w size attributes must be included



# LOC Productivity

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

# Function Points: FP



Function Points is used in 2 contexts:

- Past: To develop **metrics** from historical data
- Future: Use of available metrics to size the s/w of a **new** project

# Function Points

- Based on a combination of program characteristics
- The number of :
  - External (user) inputs: input transactions that update internal files
  - External (user) outputs: reports, error messages
  - User interactions: inquiries
  - **Logical** internal files used by the system:  
Example a purchase order logical file composed of 2 physical files/tables  
Purchase\_Order and Purchase\_Order\_Item
  - External interfaces: files shared with other systems
- A weight (ranging from **3 for simple** to **15 for complex** features) is associated with each of these above
- The function point count is computed by multiplying each raw count by the weight and summing all values

# Function Points - Calculation

<u>measurement parameter</u>	<u>count</u>		<u>weighting factor</u>				
			<u>simple</u>	<u>avg.</u>	<u>complex</u>		
number of user inputs	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of user outputs	<input type="text"/>	X	4	5	7	=	<input type="text"/>
number of user inquiries	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of files	<input type="text"/>	X	7	10	15	=	<input type="text"/>
number of ext.interfaces	<input type="text"/>	X	5	7	10	=	<input type="text"/>
count-total							<input type="text"/>
complexity multiplier							<input type="text"/>
function points							<input type="text"/>

# Function Points – Taking Complexity into Account -14 Factors Fi

Each factor is rated on a scale of:

**Zero:** not important or not applicable

**Five:** absolutely essential

1. Backup and recovery
2. Data communication
3. Distributed processing functions
4. Is performance critical?
5. Existing operating environment
6. On-line data entry
7. Input transaction built over multiple screens

## Function Points – Taking Complexity into Account -14 Factors Fi (cont.)

8. Master files updated on-line
9. Complexity of inputs, outputs, files, inquiries
10. Complexity of processing
11. Code design for re-use
12. Are conversion/installation included in design?
13. Multiple installations
14. Application designed to facilitate change by the user

## Function Points – Taking Complexity into Account -14 Factors $F_i$ (cont.)

$$FP = UFC * \left[ 0.65 + 0.01 * \sum_{i=1}^{i=14} F_i \right]$$

UFC: Unadjusted function point count

$$0 \leq F_i \leq 5$$

# FP: Advantages & Disadvantages

- Advantages
  - Available early .. We need only a detailed specification
  - Not restricted to code
  - Language independent
  - More accurate than LOC
- Disadvantages
  - Ignores quality issues of output
  - Subjective counting .. depend on the estimator
  - Hard to automate.. Automatic function-point counting is impossible



# Function points and LOC

- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - $LOC = AVC * \text{number of function points}$
  - AVC is a language-dependent factor varying from approximately 300 for assemble language to 12-40 for a 4GL

# Relation Between FP & LOC

Programming Language	LOC/FP (average)
Assembly language	320
C	128
COBOL	106
FORTRAN	106
Pascal	90
C++	64
Ada	53
Visual Basic	32
Smalltalk	22
Power Builder (code generator)	16
SQL	12

# Function Points & Normalisation

- Function points are used to normalise measures (same as for LOC) for:
  - S/w productivity
  - Quality
- Error (bugs) per FP (discovered at programming)
- Defects per FP (discovered after programming)
- \$ per FP
- Pages of documentation per FP
- FP per person-month

# Expected Software Size

- Based on three-point
- Compute Expected Software Size (S) as weighted average of:
  - Optimistic estimate: S(opt)
  - Most likely estimate: S(ml)
  - Pessimistic estimate: S(pess)
- $S = \{ S(\text{opt}) + 4 S(\text{ml}) + S(\text{pess}) \} / 6$ 
  - Beta probability distribution

# Example 1: LOC Approach

- A system is composed of 7 subsystems as below.
- Given for each subsystem the size in LOC and the  
2 metrics: productivity LOC/pm (pm: person month) ,Cost \$/LOC
- Calculate the system total cost in \$ and effort in months .

Functions	estimated LOC	LOC/pm	\$/LOC
UICF	2340	315	14
2DGA	5380	220	20
3DGA	6800	220	20
DSM	3350	240	18
CGDF	4950	200	22
PCF	2140	140	28
DAM	8400	300	18

# Example 1: LOC Approach

Functions	estimated LOC	LOC/pm	\$/LOC	Cost	Effort (months)
UICF	2340	315	14	32,000	7.4
2DGA	5380	220	20	107,000	24.4
3DGA	6800	220	20	136,000	30.9
DSM	3350	240	18	60,000	13.9
CGDF	4950	200	22	109,000	24.7
PCF	2140	140	28	60,000	15.2
DAM	8400	300	18	151,000	28.0
Totals	33,360			655,000	145.0

# Example 2: LOC Approach

Assuming

- Estimated project LOC = 33200
- Organisational productivity (similar project type) = 620 LOC/p-m
- Burdened labour rate = 8000 \$/p-m

Then

- Effort =  $33200 / 620 = (53.6) = 54$  p-m
- Cost per LOC =  $8000 / 620 = (12.9) = 13$  \$/LOC
- Project total Cost =  $8000 * 54 = 432000$  \$

# Example 3: FP Approach

	A	B	C	D	E	F	G
1	Info Domain	Optimistic	Likely	Pessim.	Est Count	Weight	FP count
2	# of inputs	22	26	30	26	4	104
3	# of outputs	16	18	20	18	5	90
4	# of inquiries	16	21	26	21	4	84
5	# of files	4	5	6	5	10	50
6	# of external inter	1	2	3	2	7	14
7	<b>UFC: Unadjusted Function Count</b>						<b>342</b>
8			Complexity adjustment factor				1.17
9						<b>FP</b>	<b>400</b>



# Example 3: FP Approach (cont.)

## Complexity Factor

Complexity factor: Fi	value=0	value=1	value=2	value=3	value=4	value=5	Fi
Backup and recovery	0	0	0	0	1	0	4
Data communication	0	0	1	0	0	0	2
Distributed processing functions	0	0	0	0	0	0	0
Is performance critical?	0	0	0	0	1	0	4
Existing operating environment	0	0	0	1	0	0	3
On-line data entry	0	0	0	0	1	0	4
Input transaction built over multiple screens	0	0	0	0	0	1	5
Master files updated on-line	0	0	0	1	0	0	3
Complexity of inputs, outputs, files, inquiries	0	0	0	0	0	1	5
Complexity of processing	0	0	0	0	0	1	5
Code design for re-use	0	0	0	0	1	0	4
Are conversion/installation included in design?	0	0	0	1	0	0	3
Multiple installations	0	0	0	0	0	1	5
Application designed to facilitate change by the user	0	0	0	0	0	1	5
						Sigma (F)	52
Complexity adjustment factor	0.65 + 0.01 * Sigma (F) =			1.17			

## Example 3: FP Approach (cont.)

Assuming

$$\sum_{i=1}^{52} F_i$$

$$FP = UFC * \left[ 0.65 + 0.01 * \sum_{i=1}^{52} F_i \right]$$

$$FP = 342 * 1.17 = 400$$

Complexity adjustment factor = 1.17

## Example 4: FP Approach (cont.)

Assuming

- Estimated FP = 401
- Organisation average productivity (similar project type) = 6.5 FP/p-m (person-month)
- Burdened labour rate = 8000 \$/p-m

Then

- Estimated effort =  $401/6.5 = (61.65) = 62$  p-m
- Cost per FP =  $8000/6.5 = 1231$  \$/FP
- Project cost =  $8000 * 62 = 496000$  \$

# Object Points (for 4GLs)

- Object points are an alternative function-related measure to function points **when 4GLs** or similar languages are used for development
- Object points are **NOT** the same as object classes
- The number of object points in a program is a weighted estimate of
  - The number of separate **screens** that are displayed
  - The number of **reports** that are produced by the system
  - The number of **3GL modules** that must be developed to supplement the 4GL code
  - <C:\Software Eng\Cocomo\Software Measurement Page, COCOMO II, object points.htm>

# Object Points – Weighting

Object Type	Simple	Meduim	Difficult
Screen	1	2	3
Report	2	5	8
Each 3GL module	10	10	10

# Object Points – Weighting (cont.)

- **svr**: number of server **data tables** used with screen/report
- **clnt**: number of client **data tables** used with screen/report

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 svr < 3 clnt)	Total < 8 (2/3 svr 3-5 clnt)	Total 8+ (> 3 svr > 5 clnt)		Total < 4 (< 2 svr < 3 clnt)	Total < 8 (2/3 svr 3-5 clnt)	Total 8+ (> 3 svr > 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

# Object Point Estimation

- Object points are **easier** to estimate from a specification than function points
  - simply concerned with screens, reports and 3GL modules
- At an **early** point in the development process:
  - Object points can be easily estimated
  - It is very difficult to estimate the number of lines of code in a system

# Productivity Estimates

- LOC productivity
  - Real-time embedded systems, 40-160 LOC/P-month
  - Systems programs , 150-400 LOC/P-month
  - Commercial applications, 200-800 LOC/P-month
- Object points productivity

Developer's experience and Capability / ICASE maturity and capability	Very low	Low	Nominal	High	Very high
<b>PROD: Productivity</b> Object-point per person-month	4	7	13	25	50



# Object Point Effort Estimation

- Effort in p-m =  $NOP / PROD$ 
  - NOP = number of OP of the system
  - Example: An application contains 840 OP (NOP=840) & Productivity is very high (= 50)
  - Then, Effort =  $840/50 = (16.8) = 17$  p-m

# Adjustment for % of Reuse

- Adjusted NOP =  $\text{NOP} * (1 - \% \text{ reuse} / 100)$
- Example:
  - An application contains 840 OP, of which 20% can be supplied by existing components.

$$\text{Adjusted NOP} = 840 * (1 - 20/100) = 672 \text{ OP}$$

$$\text{Adjusted effort} = 672/50 = (13.4) = 14 \text{ p-m}$$

# Factors affecting productivity

Factor	Description
Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers who already understand a domain are likely to be the most productive.
Process quality	The development process used can have a significant effect on productivity. This is covered in Chapter 31.
Project size	The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.
Technology support	Good support technology such as CASE tools, supportive configuration management systems, etc. can improve productivity.
Working environment	As discussed in Chapter 28, a quiet working environment with private work areas contributes to improved productivity.

# Quality and Productivity

- All **metrics** based on **volume/unit** time are flawed because they **do not take quality into account**
- Productivity may generally be increased **at the cost of quality**
- If change is constant, then an approach based on *counting lines of code* (**LOC**) is not meaningful

# Estimation techniques

- There is no simple way to make an accurate estimate of the effort required to develop a software system:
  - Initial estimates may be based on inadequate information in a user requirements definition
  - The software may run on unfamiliar computers or use new technology
  - The people in the project may be unknown
- Project cost estimates may be self-fulfilling
  - The estimate defines the budget and the product is adjusted to meet the budget