Software Estimation

- What is an estimate?
  - A *prediction* regarding the effort required to complete a project
  - Might take one of several forms:
    - *Person-months:* Project X will need 26 person-months to complete
    - *Dollars:* Project X will cost $2 million
    - *Time:* Project X will be finished in one year
    - *Features:* Given the time and money we have, we will deliver features *a,b,...,g* in this release of project X
  - All of the above can also be given as *intervals*
    - E.g., Project X will cost between $1.8 and 2.5 million

Software Estimation

- An essential but overlooked characteristic of estimates is that they have a *probability* of being true

- This is a source of conflict:
  - Academics often mean “estimate = 50% likelihood”
  - Managers often mean “estimate = 80% likelihood”
  - Developers often mean “estimate = most optimistic outcome” (about 10% likelihood!)

- **Software cost estimation:** Predicting the resources required for a software development process

Software cost components

- **Effort costs** (the dominant factor in most projects)
  - salaries of engineers involved in the project
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g library, staff restaurant, etc.)
  - costs of pensions, health insurance, etc.
- Other costs
  - Hardware and software costs
  - Travel and training costs
  - …

Costing and pricing

- There is not a simple relationship between the development cost and the price charged to the customer
- Software pricing factors
  - Market opportunity – low price to enter the market, e.g., initially “free software”
  - Cost estimation uncertainty
  - Contractual terms
  - Requirements volatility
  - Financial health
  - …

Estimation techniques

- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Pricing to win
- Top-down estimation
- Bottom-up estimation
- Function point estimation
- Algorithmic cost modelling

Expert judgement

- One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached.
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!

Estimation by analogy

- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Advantages: Accurate if project data available
- Disadvantages: Impossible if no comparable project has been tackled. Needs systematically maintained cost database

Parkinson's Law

- The project costs whatever resources are available
- Advantages: No overspend
- Disadvantages: System is usually unfinished

Pricing to win

- The project costs whatever the customer has to spend on it
- Advantages: You get the contract
- Disadvantages: The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required

Top-down estimation

- Approaches may be applied using a top-down approach. Start at system level and work out how the system functionality is provided
- Takes into account costs such as integration, configuration management and documentation
- Can underestimate the cost of solving difficult low-level technical problems

Bottom-up estimation

- Start at the lowest system level. The cost of each component is estimated individually. These costs are summed to give final cost estimate
- Accurate method if the system has been designed in detail
- May underestimate costs of system level activities such as integration and documentation

Function Points

- The idea of function point was first proposed by Albrecht in 1979.
- The function point of a system is a measure of the "functionality" of the system.
- Steps
  - Counting the information domain – counting FPs
  - Assessing complexity of the software – adjusting FPs
  - Applying an empirical relationship to come up with LOC or P-months based on the adjusted FPs
- This method cannot be performed automatically

Counting Function Points

Counting the information domain

Measurement parameter	Count		Weighting factor Simple	Av.	Complex	
Number of user inputs	_____	x	3	4	6	= _____
Number of user outputs	_____	x	4	5	7	= _____
Number of user inquiries	_____	x	3	4	6	= _____
Number of files	_____	x	7	10	15	= _____
Number of ext. interfaces	_____	x	5	7	10	= _____
Count Total ----------→						= _____

Counting Function Points

- **User inputs**. Each user input that provides distinct application oriented data to the software is counted.
- **User outputs**. Each user output that provides application oriented information to the user is counted. Individual data items within a report are not counted separately.
- **User inquiries**. This is an on-line input that results in the generation of some response.
- **Files**. Each master file is counted.
- **External interfaces**. Each interface that is used to transmit information to another system is counted.

Adjusting Function Points

Answer the following questions using a scale of [0-5]: 0 not important; 5 absolutely essential. We call them influence factors (F_i).

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational env.?
6. Does the system require on-line data entry?

Adjusting Function Points

7. Does the on-line data entry require the input transaction to be built over multiple screens or operations (user efficiency)?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Is installation included in the design?
13. Is the system designed for multiple installations?
14. Is the application designed to facilitate change and ease of use by the user?

Map FPs to LOC

- Use an empirical relationship
  - Function point = count total × [0.65 + 0.01 × (sum of the 14 F_i)]
  - Companies may want to refine their own version
- According to a 1989 study, implementing a function point in a given programming language requires the following number of lines of code
  - Assembly 320
  - C 128
  - COBOL 106
  - C++ 64
  - Visual Basic 32
  - SQL 12
- See **www.ifpug.org** for more information on FP

Example: Your PBX project

# of user inputs: {on, off, ext. number} (3) x simple (3)	= 9
# of user outputs: {tone} (1) x simple (4)	= 4
# of user inquiries: 0 x simple	= 0
# of files: {mapping table} (1)x simple (7)	= 7
# of external interfaces: {memory map} (1) x simple (5)	= 5
Total count	= 25

Example: Your PBX project

- Total of FPs = 25
- $F_4 = 4$, $F_{10} = 4$, other F_i's are set to 0. Sum of all F_i's = 8.
- FP = 25 x (0.65 + 0.01 x 8) = 18.25
- Lines of code in C = 18.25 x 128 LOC = 2336 LOC
- In the past, students have implemented their projects using about 2500 LOC.