Software Engineering

SW Measurements

(Cost, Price, Productivity and Size)

Software cost estimation

- Predicting the resources required for a software development process.

Fundamental estimation questions

- How much effort is required to complete an activity?
- How much calendar time is needed to complete an activity?
- What is the total cost of an activity?
- Project estimation and scheduling and interleaved management activities.

Software cost components

- Hardware and software costs
- Travel and training costs
- Effort costs (the dominant factor in most projects)
  - salaries of engineers involved in the project
  - Social and insurance costs
    - » Effort costs must take overheads into account
  - costs of building, heating, lighting
  - costs of networking and communications
  - costs of shared facilities (e.g library, staff restaurant, etc.)

Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system
- There is not a simple relationship between the development cost and the price charged to the customer
- Broader organisational, economic, political and business considerations influence the price charged

Software pricing 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

- A measure of the rate at which individual engineers involved in software development produce software and associated documentation
- Not quality-oriented although quality assurance is a factor in productivity assessment
- Essentially, we want to measure useful functionality produced per time unit

- 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.

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	4 weeks	6 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

Productivity measures

- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- 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

Factors affecting productivity

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.
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, configuration management systems, etc. can improve productivity.
Working environment	, a quiet working environment with private work areas contributes to improved productivity.

Common Measurement problems

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

Software Metrics

- Black Box Metrics
  - Function Oriented Metrics
    - » Feature Points
    - » *Function Points*
- White Box Metrics
  - *LOC*
  - Halstead's Software Science
  - McCabe's Cyclomatic Complexity

LOC Metrics

- Easy to use
- Easy to compute
- Language & programmer dependent

Size-oriented metrics

- Example of simple size-oriented metrics
  - Errors per KLOC (thousand lines of code)
  - Defects per KLOC
  - $ per KLOC
  - Pages of documentation per KLOC

Problems

- Size-oriented metrics are not universally accepted as the best way to measure the software process
- Opponents argue that KLOC measurements
  - Are dependent on the programming language
  - Penalize well-designed but short programs
  - Cannot easily accommodate nonprocedural languages
  - Require a level of detail that may be difficult to achieve

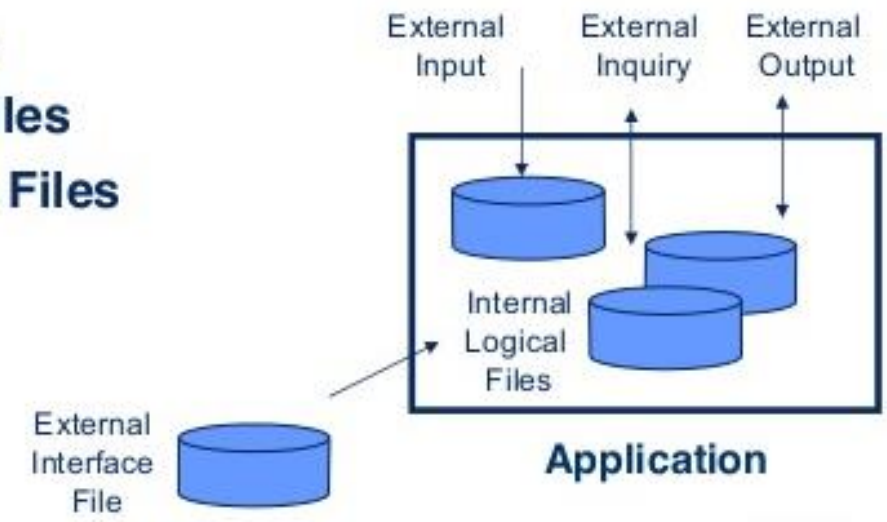
Function Oriented Metrics

- Mainly used in business applications
- The focus is on program functionality
- Most common are:
  - Function Points (FP)
  - Feature points

The Function Point Methodology

Five key components are identified based on logical user view

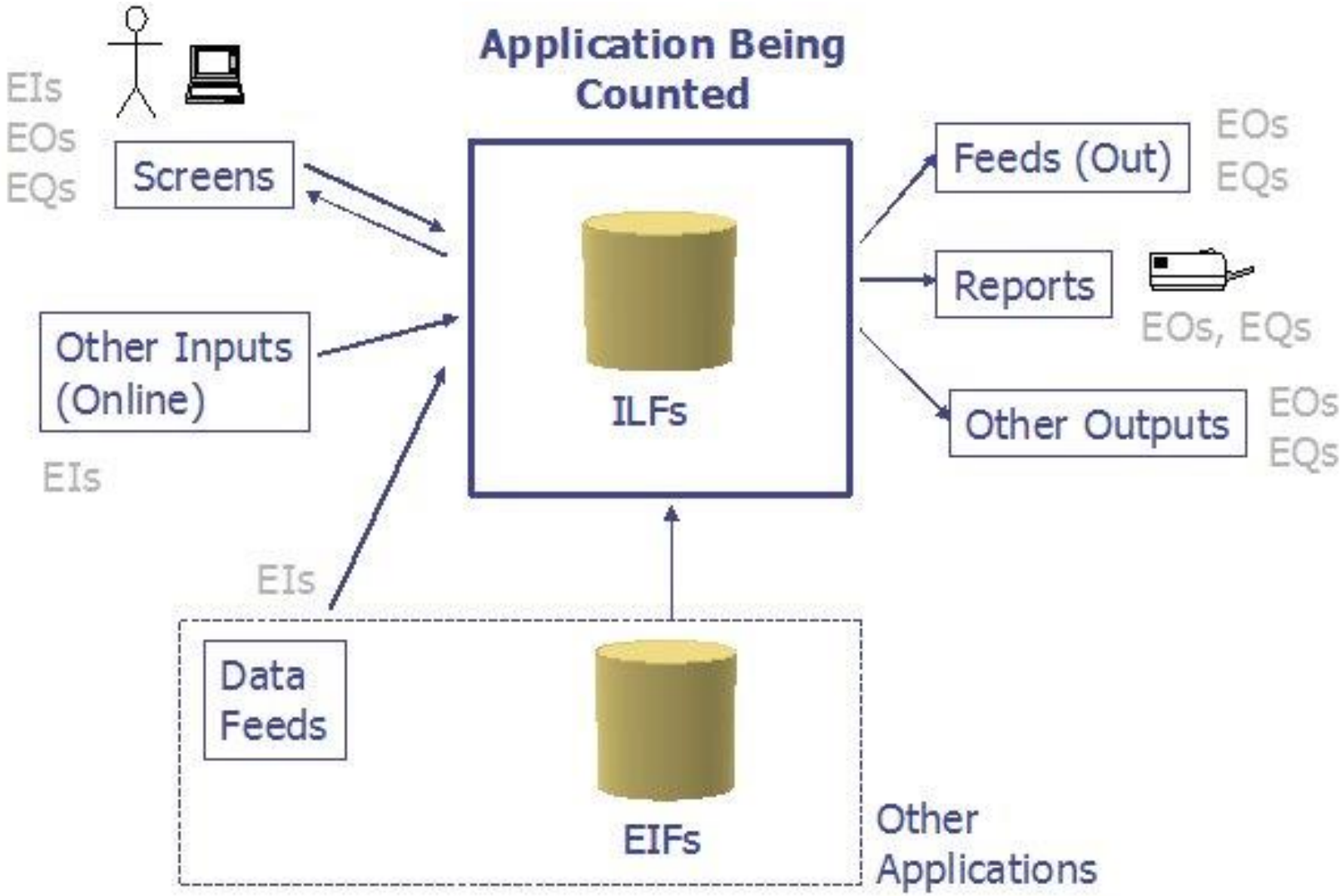
- External Inputs
- External Outputs
- External Inquiries
- Internal Logical Files
- External Interface Files



Function-Oriented Metrics

- use “functionality” to measure
- derived from “function point”
- using an empirical relationship
- based on countable (direct) measure of SW information domain and assessments of software complexity
- Measuring scale of a project

Information domain values



Information domain values

- Number of external inputs (EI)
  - Originates from a user or is transmitted from another application and provides distinct application-oriented data or control information
- Number of external outputs (EO)
  - Derived within the application and provides information to the user
- Number of external inquiries (EQ)
  - Interactive inputs requiring a response

Information domain values

- Number of internal logical files (ILF)
  - A logical grouping of data that resides within the application's boundary and is maintained via external inputs.

- Number of external interface files (EIF)
  - A logical grouping of data that resides external to the application but provides data that may be of use to the application

Steps In Calculating (FP)

- Count the information domain values.
- Assess the complexity of the values.
- Calculate the raw FP
- Rate the complexity factors to produce the **Complexity Adjustment Aalue (CAV)**

$$\sum_{i=1}^{14} F_i$$

- Calculate the adjusted FP as follows:

  FP = raw FP x [0.65 + 0.01 x CAV]

Step 1

- Count the information domain values Number of external inputs
  - Number of user inputs
  - Number of external outputs
  - Number of external inquiries
  - Number of internal logical files
  - Number of external interface files

Step 2

- Assess the complexity of the values
  - Simple, Average, Complex
  - Determined based on organisational experience based criteria. Nonetheless, the determination of complexity is somewhat subjective
  - Compute count total

Step 3:

- Calculate the raw FP

			Weighting Factor			
Parameter	Count		Simple	Average	Complex	
Inputs		x	3	4	6	=
Outputs		x	4	5	7	=
Inquiries		x	3	4	6	=
Files		x	7	10	15	=
Interfaces		x	5	7	10	=
				Count-total (raw FP)		

Complexity Adjustment Value

- Complexity Adjustment Values are rated on a scale of 0 (not important) to 5 (very important):
  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. System to be run in an existing, heavily utilized environment?
  6. Does the system require on-line data entry?
  7. On-line entry requires input over multiple screens or operations?
  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. Are conversion and instillation included in the design?
  13. Multiple installations in different organizations?
  14. Is the application designed to facilitate change and ease-of-use?

14 GSCs [General System Characteristics]

1. Data Communication
2. Distributed data processing
3. Performance
4. Heavily used configuration
5. Transaction rate
6. Online data entry
7. End user efficiency
8. Online update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. Facilitate change

Complexity Adjustment Value

- The rating for all the factors, F_1 to F_{14}, are summed to produce the complexity adjustment value (CAV)
  - varies between 0 and 70.
- Assign grades (0 to 5)

  0 - No influence/not important

  1 - Incidental

  2 - Moderate

  3 - Average

  4 - Significant

  5 – Essential
- CAV is then used in the calculation of the function point (FP) of the software

Step 4

- Assess the complexity of the values

$$\sum_{i=1}^{14} F_i$$

Step 5

- Calculate the adjusted FP as follows:

FP = raw FP x (0.65 + 0.01 x CAV)

Exercise: Function Points

- Compute the function point value for a project with the following information domain characteristics:
  - Number of user inputs: 32
  - Number of user outputs: 60
  - Number of user enquiries: 24
  - Number of files: 8
  - Number of external interfaces: 2
  - Assume that weights are average and external complexity adjustment values are not important.

Solution

			Weighting Factor			
Parameter	Count		Simple	Average	Complex	
Inputs	32	x	3	4	6	= 128
Outputs	60	x	4	5	7	= 300
Inquiries	24	x	3	4	6	= 96
Files	8	x	7	10	15	= 80
Interfaces	2	x	5	7	10	= 14
				Count-total (raw FP)		618

Solution

- **CAV= 0**
- **Raw FP=618**

$$FP = raw\ FP \times (0.65 + 0.01 \times CAV)$$
$$= 618 \times (0.65 + 0.01 \times 0)$$
$$= 401.7$$

Example

- **Compute the function-point value for the project with the following information domain characteristics: Assume that all complexity adjustment values are average.**

Number of inputs:	32
Number of outputs:	60
Number of files:	8
Number of inquiries:	24
Number of external interfaces:	2

Comparing LOC and FP

- LOC Per Function Point

Programming Language	LOC/FP (average)
Assembly Language	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Object-oriented language	30
Fourth-generation language	20
Code generators	15

LOC and FP

- LOC and FP-based metrics have been found to be relatively accurate predictors of software development effort and cost
- Typical metrics
  - $ per LOC, $ per FP
  - LOC per person-month, FP per person-month
  - errors per KLOC, errors per FP
  - pages of documentation per KLOC, pages of documentation per FP

Object points

- Object points are an alternative function-related measure to function points
- 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 modules that must be developed

END