SOFTWARE ENGINEERING

Spring 2024

THREE PRINCIPLES FOR SOFTWARE QUALITY MANAGEMENT

At the organizational level, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.

At the project level, quality management involves the application of specific quality processes and checking that these planned processes have been followed.

At the project level, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used.

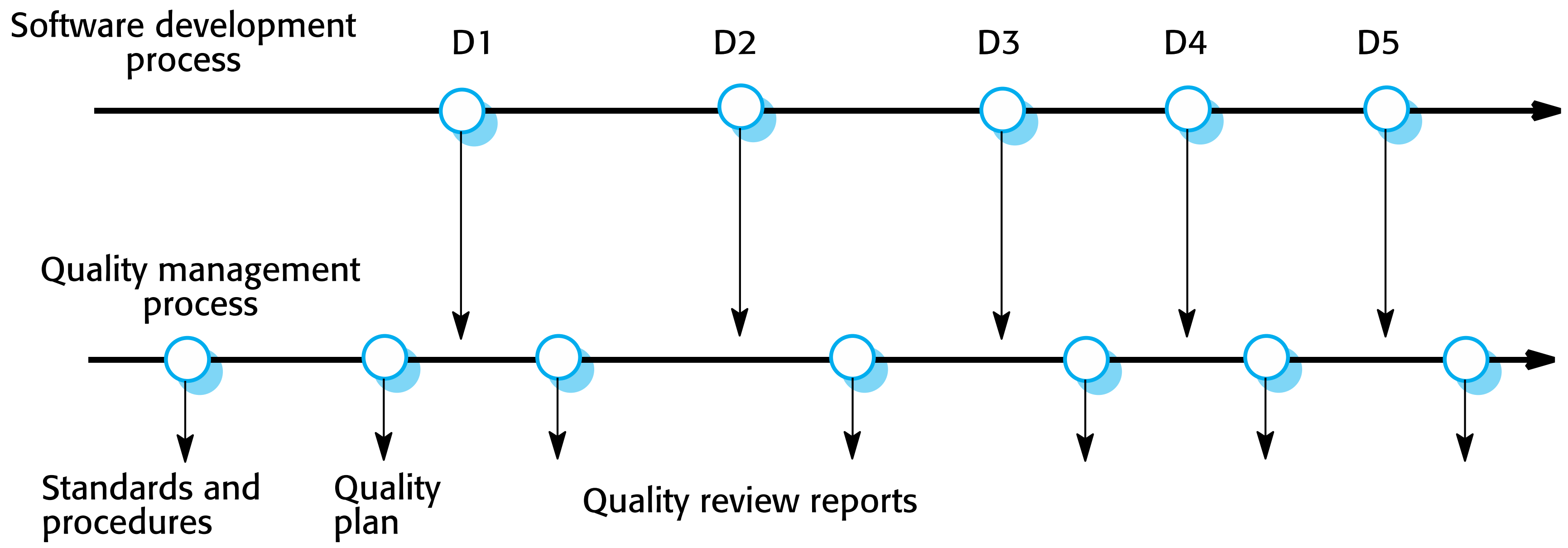
QUALITY MANAGEMENT ACTIVITIES

independent check on the software development process.

checks the project deliverables to ensure that they are consistent with organizational standards and goals

quality team should be independent from the development team so that they can take an objective view of the software.

QUALITY MANAGEMENT AND SOFTWARE DEVELOPMENT



QUALITY PLANNING

- sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.

Setting Product Qualities

- should define the quality assessment process.

Quality Assessment Process

- organisational standards should be applied and, where necessary, define new standards to be used.

Organizational Standards

QUALITY PLANS STRUCTURE

Product introduction

- description of the product
- its intended market
- the quality expectations for the product.

Product plans

- critical release dates
- responsibilities for the product
- plans for distribution and product servicing.

Process descriptions

- The development and service processes
- standards that should be used for product development and management.

Quality goals

- identification and justification of critical product quality attributes.

Risks and risk management

- risks that might affect product quality
- the actions to be taken to address these risks.

Quality plans should be short, succinct documents
If they are too long, no-one will read them.

SOFTWARE QUALITY

SOFTWARE QUALITY

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.
- The focus may be 'fitness for purpose' rather than specification conformance.

SOFTWARE FITNESS FOR PURPOSE

- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?
- Have programming and documentation standards been followed in the development process?

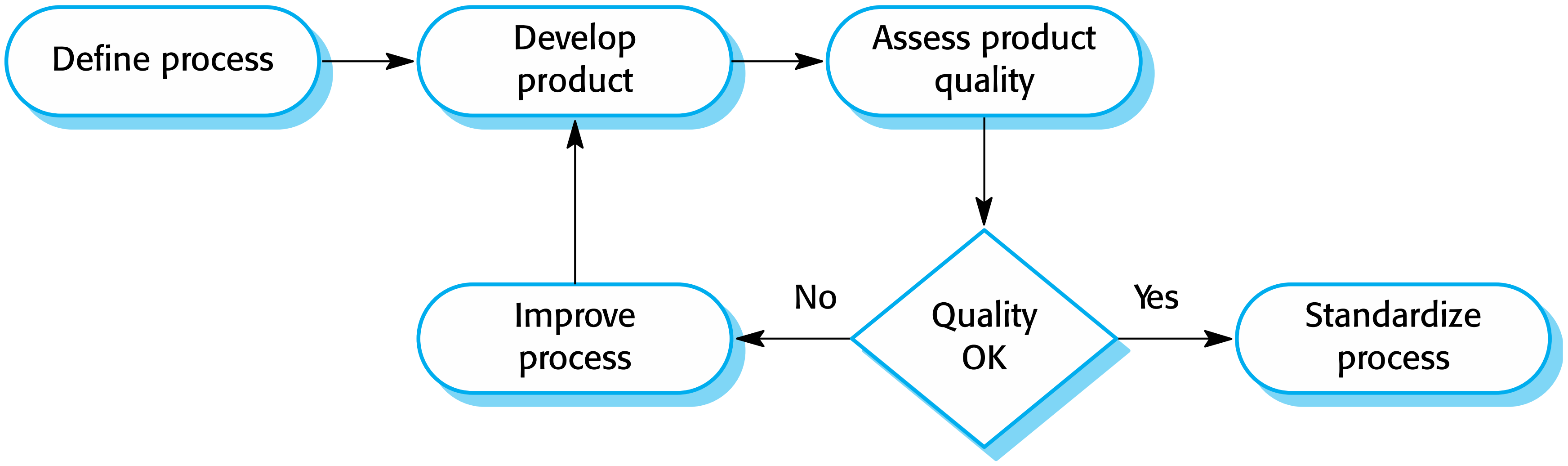
SOFTWARE QUALITY ATTRIBUTES

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

QUALITY CONFLICTS

- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.
- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.

PROCESS-BASED QUALITY



SOFTWARE STANDARDS

IMPORTANCE OF STANDARDS

- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for defining what quality means in a particular setting i.e. that organization's view of quality.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.

Software standards

Standards define the required attributes of a product or process. They play an important role in quality management.
Standards may be international, national, organizational or project standards.

PRODUCT AND PROCESS STANDARDS

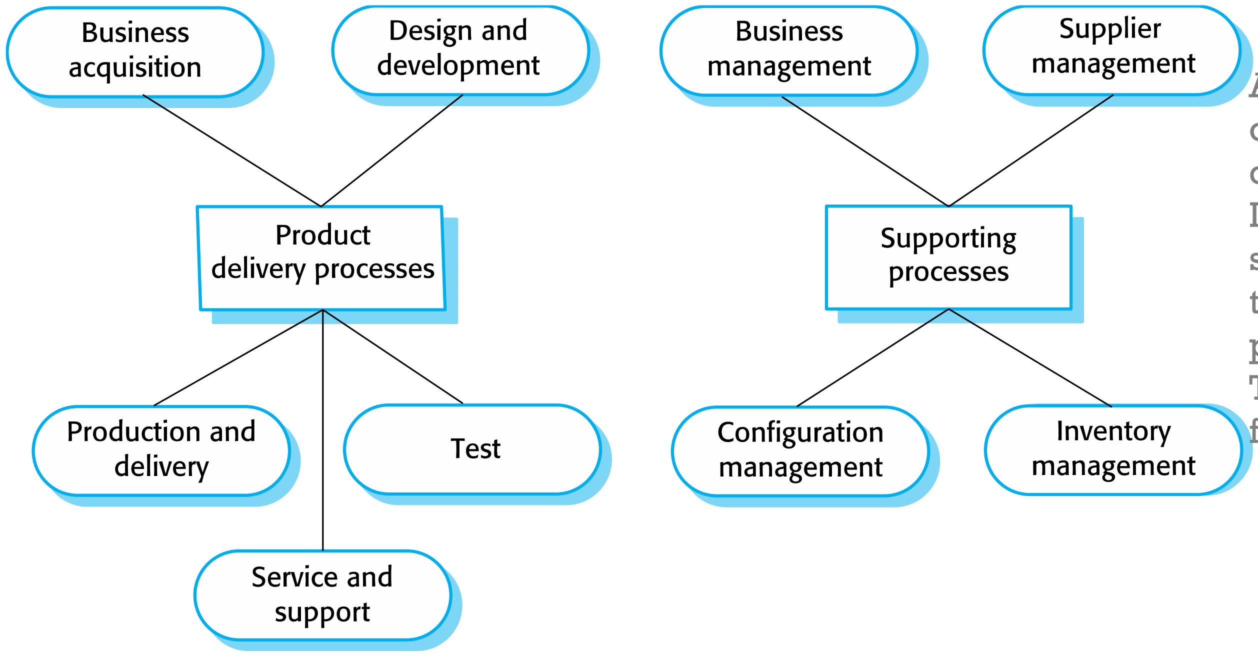
Problems with standards

They may not be seen as relevant and up-to-date by software engineers

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

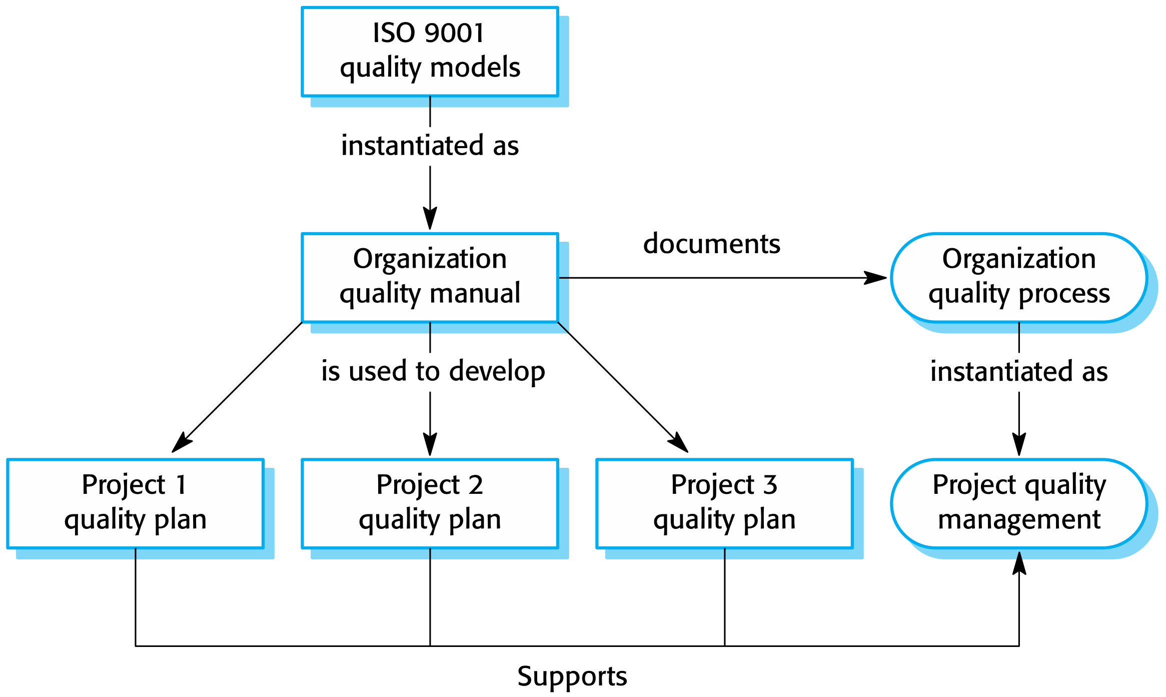
- *Product standards*
  - Apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.
- *Process standards*
  - These define the processes that should be followed during software development. Process standards may include definitions of specification, design and validation processes, process support tools and a description of the documents that should be written during these processes.

ISO 9001 CORE PROCESSES



An international set of standards that can be used as a basis for developing quality management systems.
ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
The ISO 9001 standard is a framework for developing software standards.

ISO 9001 and Quality Management



REVIEWS AND INSPECTIONS

- A group examines part or all of a process or system and its documentation to find potential problems.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.
- There are different types of review with different objectives
  - Inspections for defect removal (product);
  - Reviews for progress assessment (product and process);
  - Quality reviews (product and standards).

PHASES IN THE REVIEW PROCESS

Pre-review activities

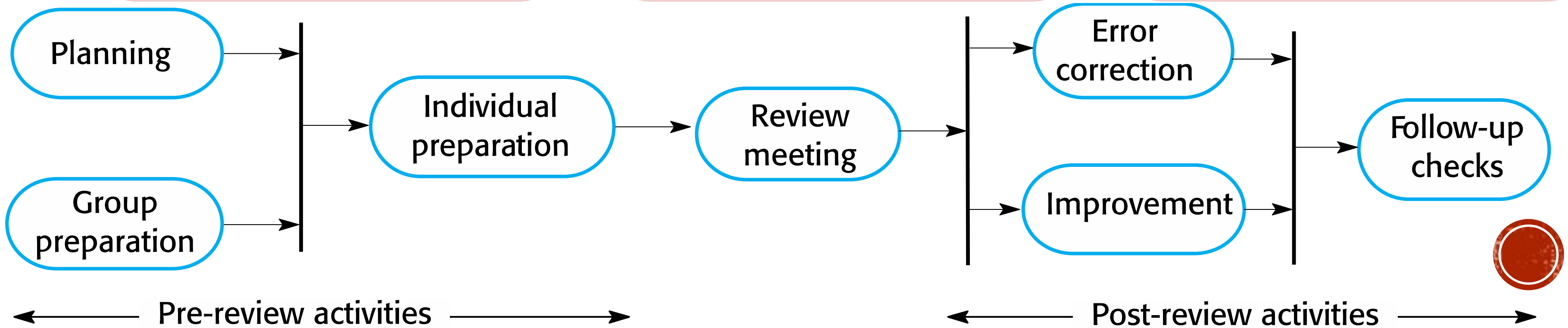
Pre-review activities are concerned with review planning and review preparation

The review meeting

During the review meeting, an author of the document or program being reviewed should 'walk through' the document with the review team.

Post-review activities

These address the problems and issues that have been raised during the review meeting.



PROJECT ESTIMATION

SOFTWARE PROJECT ESTIMATING

- Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project.
- Various measures are used in project size estimation. Some of these are:
  - LOC
  - Functional Point

SOFTWARE PROJECT ESTIMATING

- LOC and FP data are used in two ways during software project estimation:
  - (1) as estimation variables to "size" each element of the software
  - (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common.

Baseline productivity metrics (e.g., LOC/pm or FP/pm)

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 are interleaved management activities.
- Project Estimation is done on the basis of :
- • Project size
- • Project Cost
- • Effort required
- • Project duration

SOFTWARE COST COMPONENTS

- Hardware and software costs.
- Travel and training costs.
- Effort costs (the dominant factor in most projects)
  - The 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.

SOFTWARE 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.

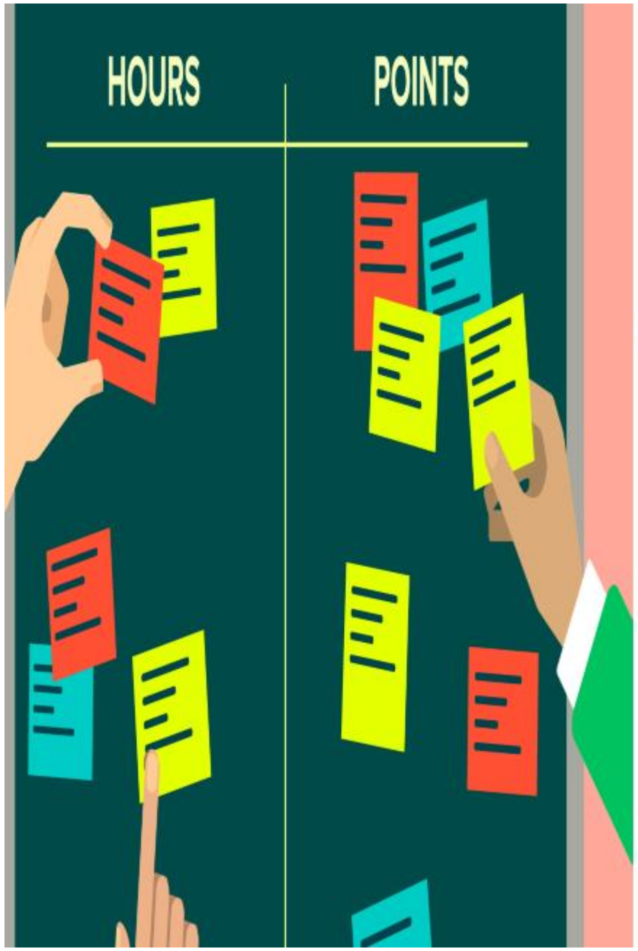
PRODUCTIVITY MEASURES

- **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.
- Metrics used for project size estimation
- Lines of codes (LOC)
- Functional points (FPs)

PRODUCTIVITY COMPARISONS

- 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.
- 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.



SOFTWARE PROJECT ESTIMATION

LOC and FP data are used in two ways during software project estimation:

- as estimation variables to "size" each element of the software.
- as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.
- LOC This is widely used metric for measuring the size of a software system. It counts the number of lines of code in the software system and is used as a basis for estimating the effort required to develop the software.
- Measured in LOC/pm or LOC/pm^6

LINES OF CODE ESTIMATION

LOC

- Count every line of source code except blanks (improve code readability) or comments
- (improves code understandability).
- Count each declaration, actual code containing logic and computations
- Advantages:

  ✓ Easy to count and calculate

- Disadvantages:

  ✓ Not a good metric for estimation as its highly dependent on the programming language used.

LOC

- Break projects into features
- Estimate lines of code by analyzing past data. For example, a similar component has been created before and consider its optimistic, most likely and pessimistic values.

$$\text{Expected size} = (S_{\text{optimistic}} + 4 * S_{\text{most likely}} + S_{\text{pessimistic}}) / 6$$

- In upcoming example, the values acquired for the task 3D geometric analysis for past data are :

Optimistic =4600 LOC, most likely = 6900 LOC, pessimistic = 8600 LOC.

So the Estimated LOC will be calculated using above formula :

$$\text{Expected LOC} = \frac{4600+(4*6900)+8600}{6} = 6800 \text{ LOC}$$

LOC BASED ESTIMATION EXAMPLE 1

- The mechanical CAD software will accept two- and three-dimensional geometric data from an engineer.
- The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design.
- All geometric data and other supporting information will be maintained in a CAD database.
- Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices.
- The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

LOC BASED ESTIMATION EXAMPLE 1

Functions	Estimated LOC
2D geometric analysis	5300
3D geometric analysis	6800
UI and control facilities	2300
Database management	3350
Design analysis modules	8400
Computer graphics display facilities	4950
Peripheral control	2100

LOC BASED ESTIMATION EXAMPLE 1

A "person month" is the metric for expressing the effort (amount of time) personnel devote to a specific project.

- As per historic data, it is deduced that:
  - Organization can write = 620 LOC/pm
  - Payment to a personnel = \$ 8000/month
  - Cost/LOC = $\frac{8000}{620}$ = 12.9 ≈ \$ 13
  - Effort required = Total LOC / LOC per person-month
  - Effort required for the project = $\frac{5300+6800+2300+3350+8400+4950+2100}{620} = \frac{33200}{620}$ ≈ 53 person month
  - Project cost to be paid = estimated LOC * Cost/LOC

    = (5300+6800+2300+3350+8400+4950+2100) * 13

    = \$ 4,31,600

LOC BASED ESTIMATION EXAMPLE 2

- Login and registration module: 1000 LOC
- Search and booking module: 3000 LOC
- Payment and checkout module: 2000 LOC
- User profile module: 1500 LOC
- Feedback and rating module: 500 LOC
- Use historical data to estimate the effort required to develop the project:
- Let's assume that the historical data suggests an average of 600 LOC per person-month and an average cost of $8000 per person-month.

LOC BASED ESTIMATION EXAMPLE 2

- Calculate the total number of lines of code:
- Total LOC = 1000 + 3000 + 2000 + 1500 + 500 = 8000
- Cost per LOC = $8000 / 600 = 13.3
- Effort required = Total LOC / LOC per person-month = 8000 / 600 = 13.3 person-months.
- Project cost = Estimated LOC* Cost per LOC = 8000 * $13.3 = $106,400

LOC BASED ESTIMATION EXAMPLE 3

- Suppose we have a software project that is estimated to be 50,000 lines of code. The development team consists of 5 developers, each with a salary of $50 per hour. The project is expected to take 12 months to complete. Assuming 160 hours per month per developer and a productivity rate of 600 lines of code per developer per month.
  - ✓ 1. Calculate the total effort required for the project
  - ✓ 2. Calculate the total cost of each developer per month
  - ✓ 3. Calculate the total cost of the project

FUNCTION POINT ESTIMATION

FUNCTION POINTS

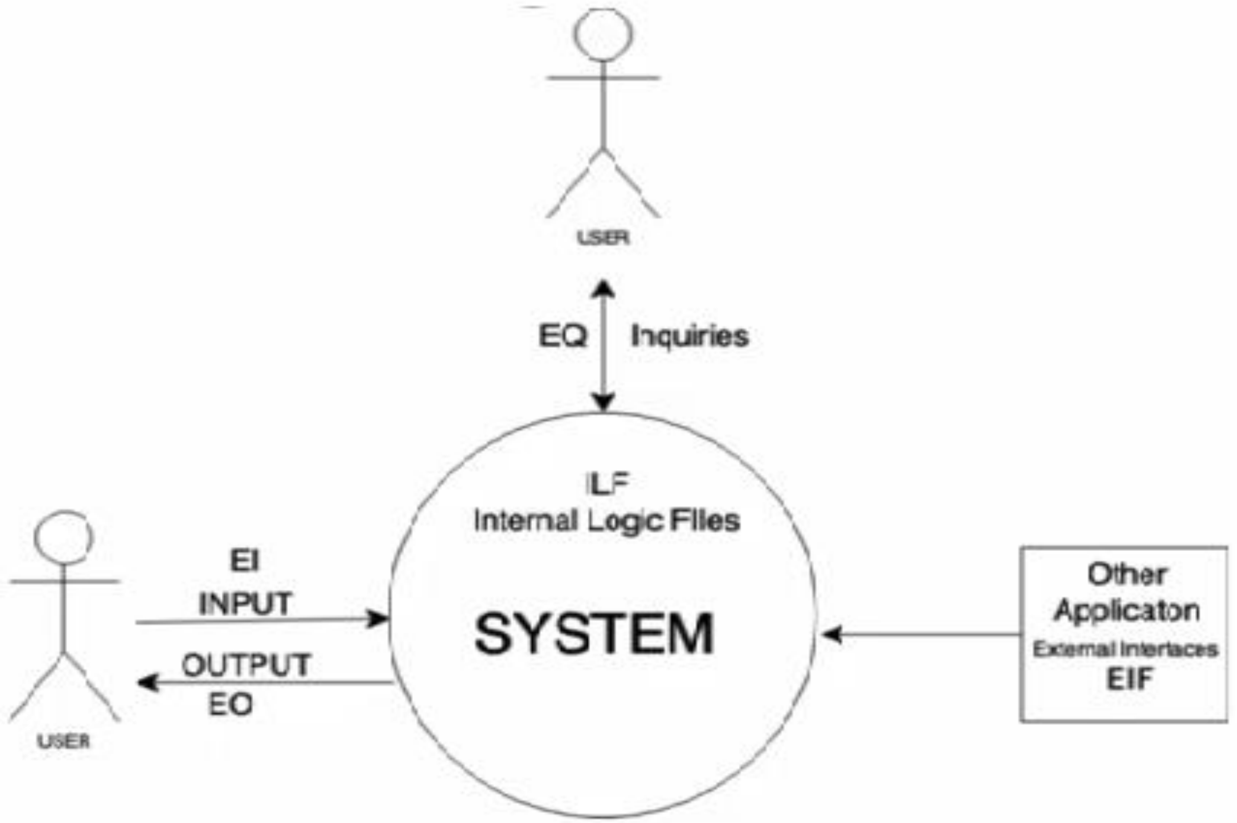
- Based on a combination of program characteristics
  - external inputs and outputs;
  - user interactions;
  - external interfaces;
  - files used by the system.
- A weight is associated with each of these and the function point count is computed by multiplying each raw count by the weight and summing all values.

$$UFC = \sum(\text{number of elements of given type}) \times (\text{weight})$$

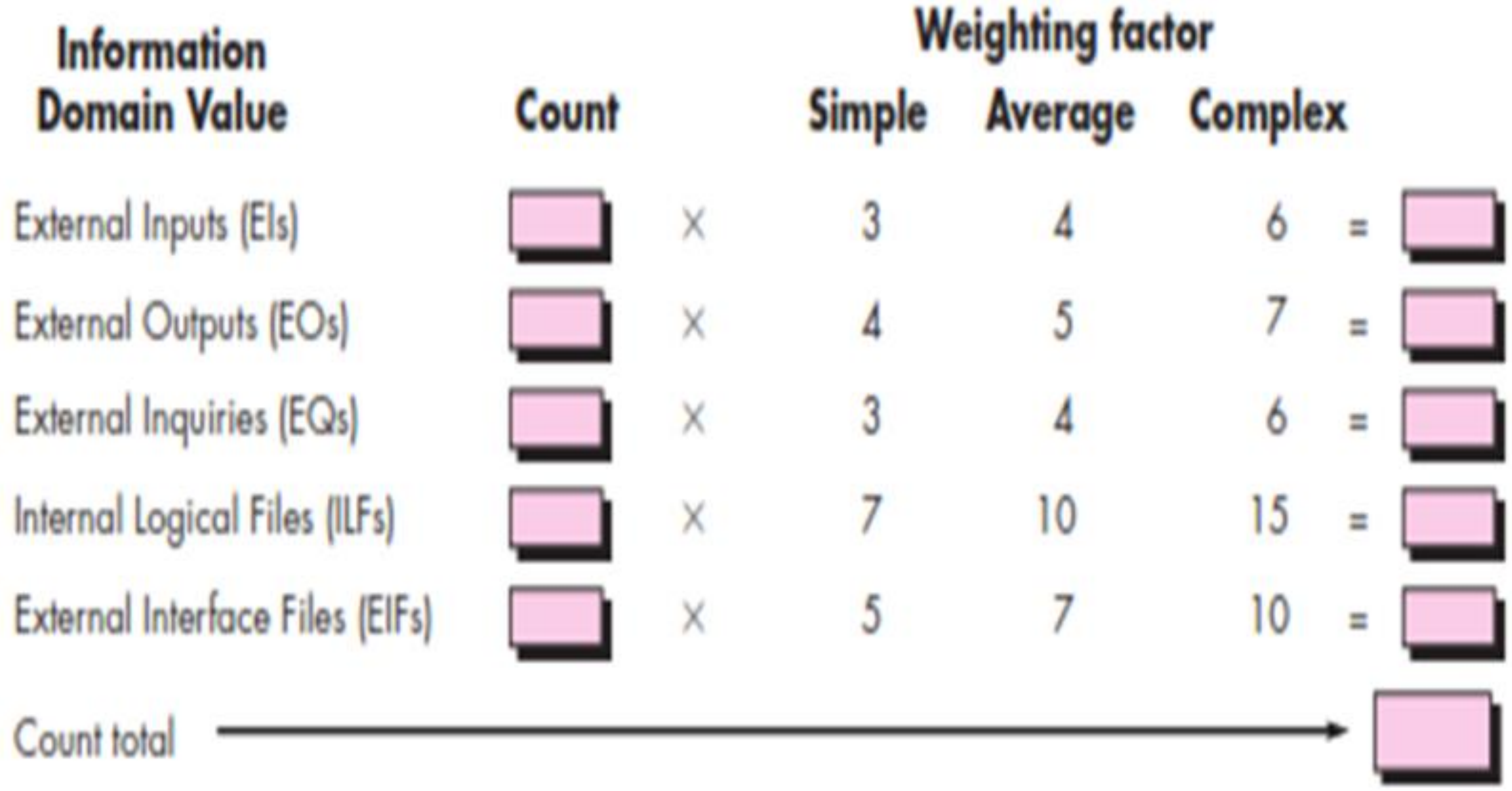
$$F.P = UFP \;\; X \;\; CAF$$

FP INFORMATION DOMAIN

- Number of external inputs (EIs). Each external input originates from a user and provides distinct application-oriented data or control information. Inputs are often used to update internal logical files (ILFs). Inputs should be distinguished from inquiries, which are counted separately.
- Number of external outputs (EOs). Each external output is derived data within the application that provides information to the user.
- Number of external inquiries (EQs). An external inquiry is defined as an online input that results in the generation of some immediate software response in the form of an online output (often retrieved from an ILF).
- Number of internal logical files (ILFs). Each internal logical file is a logical grouping of data that resides within the application's boundary.
- Number of external interface files (EIFs). Each external interface file is a logical grouping of data that resides external to the application but provides information that may be of use to the application.


USER
EQ
Inquiries
ILF
Internal Logic Files
SYSTEM
EI
INPUT
OUTPUT
EO
USER
Other
Applicaton
External Interfaces
EIF

1. COMPUTING FPS



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

2. CALCULATING CAF
COMPLEXITY ADJUSTMENT FACTOR

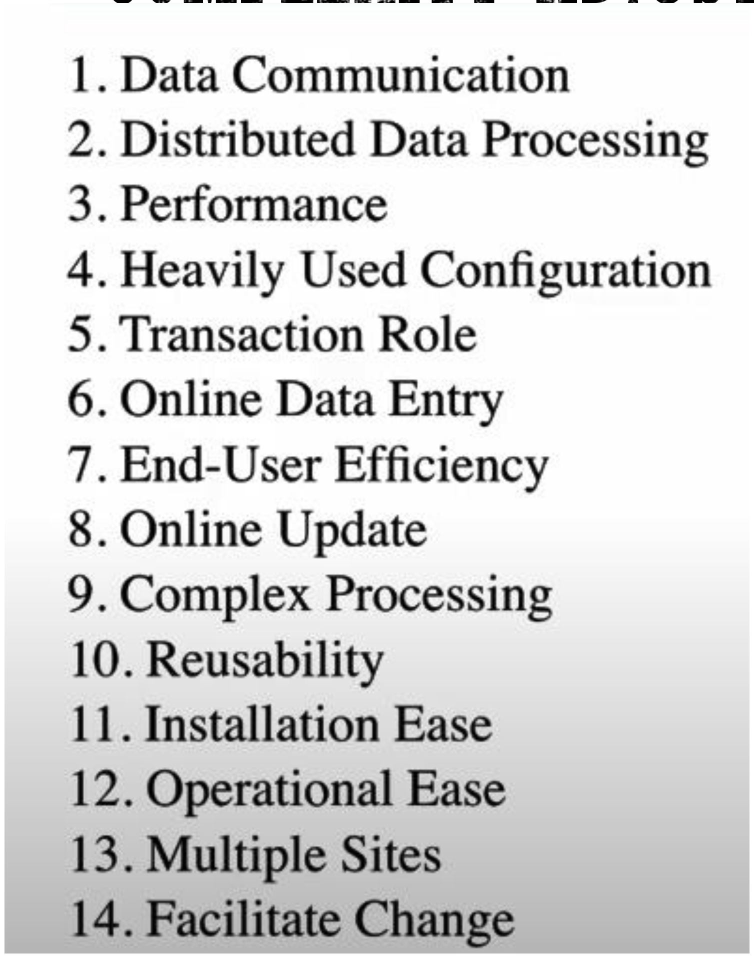
$$F.P = UFP \ X \ CAF$$

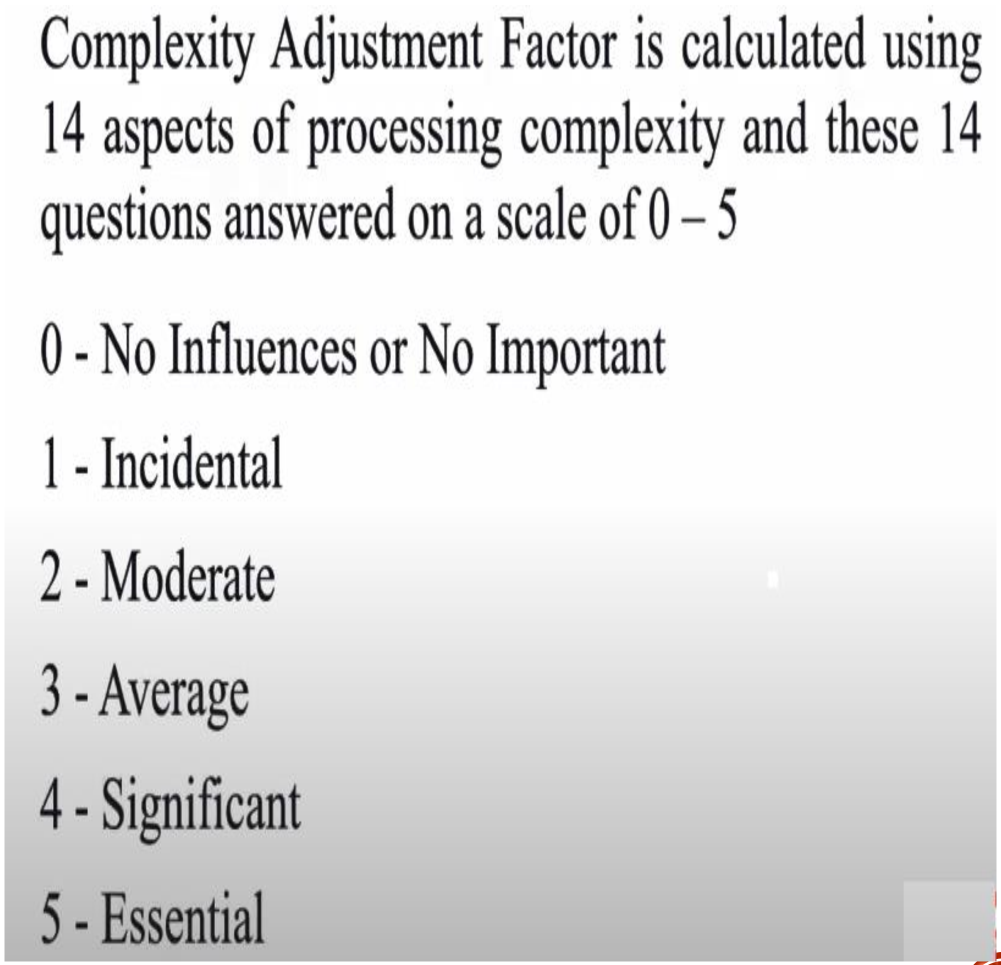
$$CAF = 0.65 + (0.01 \times \Sigma Fi)$$

Where,

Fi = Value adjustment factors based on responses to the following 14 questions

2. CALCULATING CAF
COMPLEXITY ADIUSTMENT FACTOR


1. Data Communication
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Role
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 Factor is calculated using 14 aspects of processing complexity and these 14 questions answered on a scale of 0 – 5

0 - No Influences or No Important

1 - Incidental

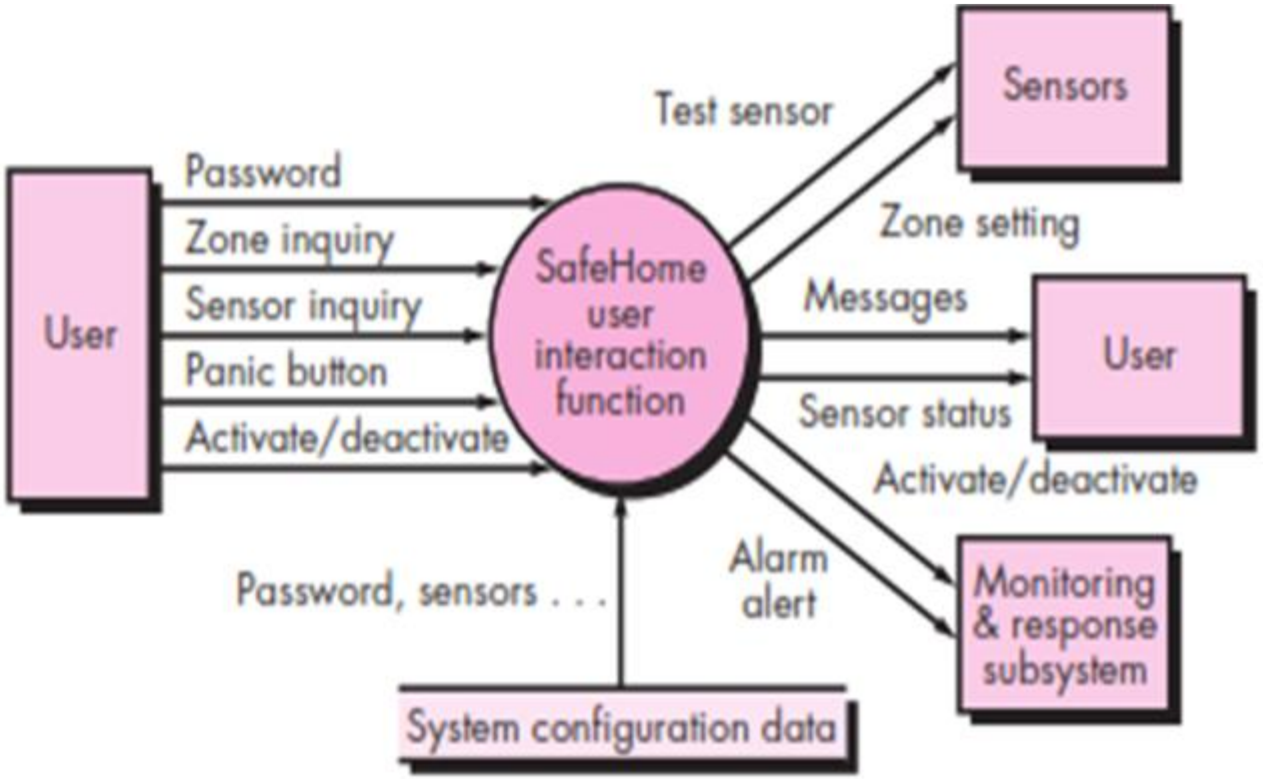
2 - Moderate

3 - Average

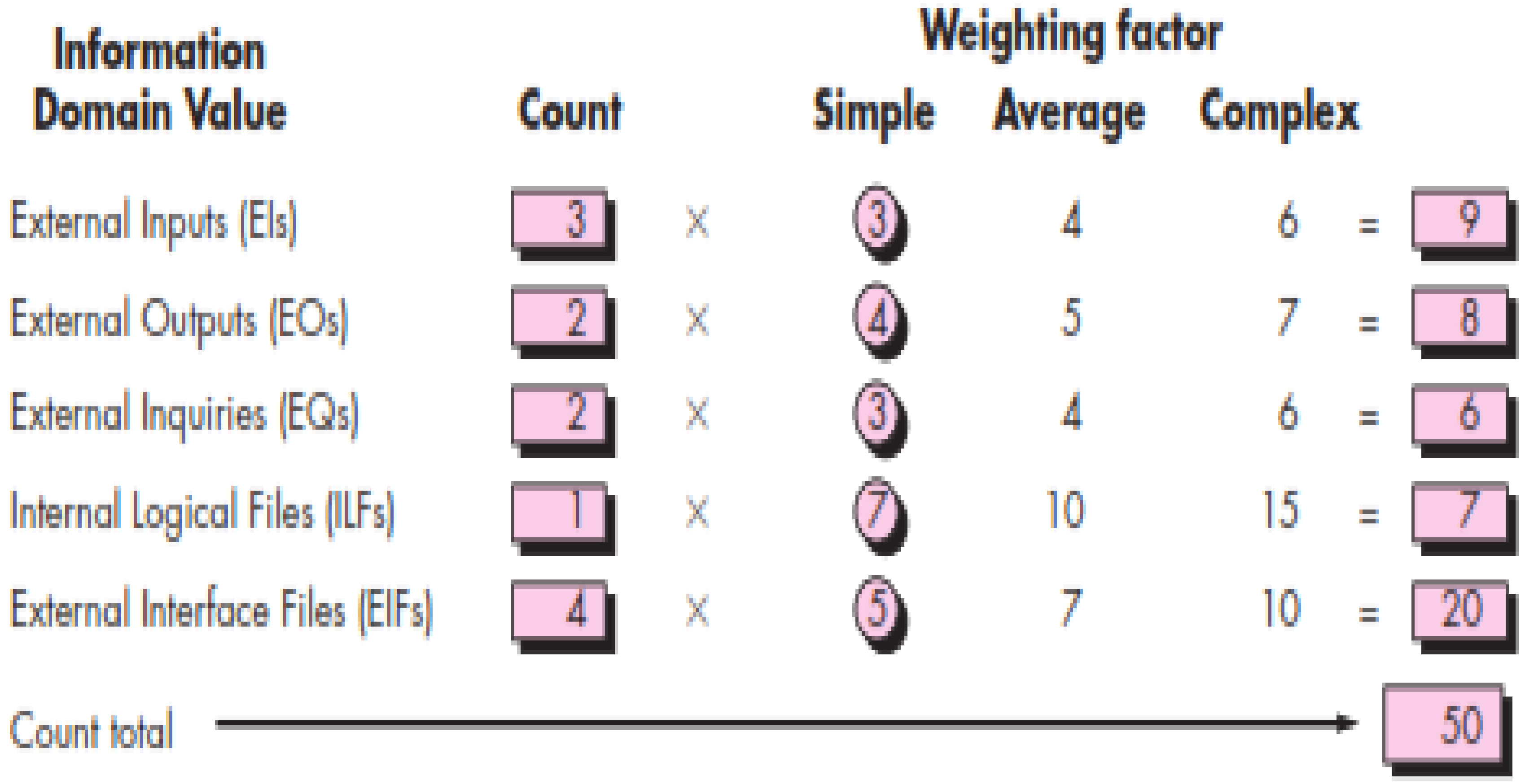
4 - Significant

5 - Essential

EXAMPLE: DFD FOR COMPUTING FP



COMPUTING FP



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

F.P = UFP X CAF

CAF = 0.65 + (0.01 x ΣFi) Moderately complex product

Then,

ΣFi = 14 x 2 = 28 2 - Moderate

CAF = 0.65 + (0.01 x 28)

CAF = 0.93

F.P = UFP X CAF

F.P = 50 X 0.93 = 46.5

RECONCILING LOC AND FP METRICS /PRODUCTIVITY ESTIMATES

- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
  - LOC = AVC * number of function points;
  - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL;
  - The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design.
  - A number of studies have attempted to relate FP and LOC measures. The following table 4 [QSM02] provides rough estimates of the average number of lines of code required to build one function point in various programming languages:

Programming Language	LOC per Function Point			
	Average	Median	Low	High
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP 69	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Clipper	38	39	27	70
COBOL	77	77	14	400
Cool:Gen/IEF	38	31	10	180
Culprit	51	—	—	—
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Excel47	46	—	31	63
Focus	43	42	32	56
FORTRAN	—	—	—	—
FoxPro	32	35	25	35
Ideal	66	52	34	203
IEF/Cool:Gen	38	31	10	180
Informix	42	31	24	57
Java	63	53	77	—
JavaScript	58	63	42	75
JCL	91	123	26	150
JSP	59	—	—	—
Lotus Notes	21	22	15	25
Mantis	71	27	22	250
Mapper	118	81	16	245
Natural	60	52	22	141
Oracle	30	35	4	217
PeopleSoft	33	32	30	40
Perl	60	—	—	—
PL/1	78	67	22	263
Powerbuilder	32	31	11	105
REXX	67	—	—	—
RPG II/III	61	49	24	155
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
VBScript36	34	27	50	—
Visual Basic	47	42	16	158

FP based estimation Example 2

- User input = 55
- User outputs = 35
- User enquiries = 40
- User files = 8
- External interfaces =5
- Calculate FP = UFP * CF
- Calculate CAF
- CAF = 0.65 +(0.01 * Σ Fi)
- Lets say, the product is significantly complex product so all 14 questions answered in moderately form i.e. 4
- Σ Fi = ? (if complexity factor is differing for each FP then calculate separately.)
- CAF = 0.65 + (0.01 * ?) = ?
- FP = UFP * CAF
- FP = ? * ? = ?

FP BASED ESTIMATION EXAMPLE 3

- Suppose we have a software project that has an estimated size of 5000 function points. We have historical data that shows that it takes an average of 10 person hours to complete one function point. The team consists of 8 developers, each with a salary of $60 per hour. The project is expected to take 12 months to complete. Calculate the effort required and total cost of the project.

To calculate the effort required for the project, we'll first find the total person-hours needed to complete the project, then calculate the total cost.

Effort Calculation:

Total Function Points (FP) = 5000

Average Person Hours per Function Point = 10

Total Person Hours = Total Function Points * Average Person Hours per Function Point

= 5000 FP * 10 hours/FP= 50,000 hours

Cost Calculation:

Total Person Hours = 50,000 hours

Number of Developers = 8

Salary per Developer = $60/hour

Total Cost = Total Person Hours * Salary per Hour * Number of Developers

= 50,000 hours * $60/hour * 8 developers

= $24,000,000

So, the effort required for the project is 50,000 person-hours, and the total cost of the project is $24,000,000.