Software Project Management

Introduction

- Many software projects fail:
  - due to faulty project management practices:
    - It is important to learn different aspects of software project management.
- Goal of software project management:
  - Enable a group of engineers/developers to work efficiently towards successful completion of a software project.
- A project manager's activities are varied.
  - can be broadly classified into:
    - project planning,
    - project monitoring and control activities.
- Once a project is found to be feasible,
  - project managers undertake project planning.

Project Planning Activities

- Estimation:
  - Effort, cost, resource, and project duration
- Project scheduling:
- Staff organization:
- Risk handling:
  - identification, analysis, and abatement procedures
- Miscellaneous plans:
  - quality assurance plan, configuration management plan, etc.

Project planning

- Requires utmost care and attention --- commitments to unrealistic time and resource estimates result in:
  - irritating delays.
  - customer dissatisfaction
  - adverse affect on team morale
    - poor quality work
  - project failure.

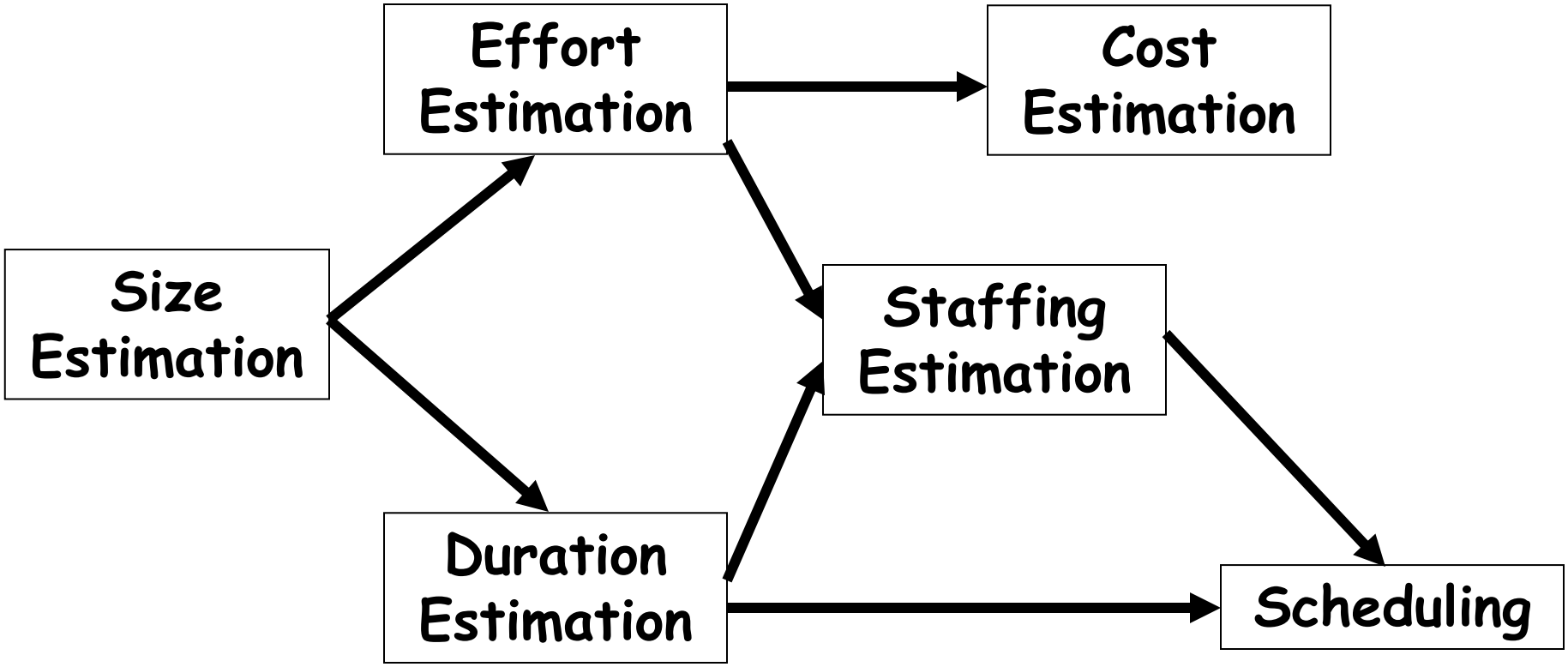
Sliding Window Planning

- Involves project planning over several stages:
  - protects managers from making big commitments too early.
  - More information becomes available as project progresses.
    - Facilitates accurate planning
- After planning is complete:
  - Document the plans:
  - in a Software Project Management Plan (SPMP) document.

Organization of SPMP Document

- Introduction (Objectives, Major Functions, Performance Issues, Management and Technical Constraints)
- Project Estimates (Historical Data, Estimation Techniques, Effort, Cost, and Project Duration Estimates)
- Project Resources Plan (People, Hardware and Software, Special Resources)
- Schedules (Work Breakdown Structure, Task Network, Gantt Chart Representation, PERT Chart Representation)
- Risk Management Plan (Risk Analysis, Risk Identification, Risk Estimation, Abatement Procedures)
- Project Tracking and Control Plan
- Miscellaneous Plans(Process Tailoring, Quality Assurance)

Software Cost Estimation



Software Cost Estimation Techniques

- Three main approaches to estimation:
- Empirical techniques:
  - an educated guess based on past experience.
- Heuristic techniques:
  - assume that the characteristics to be estimated can be expressed in terms of some mathematical expression.
- Analytical techniques:
  - derive the required results starting from certain simple assumptions.

Software Size Metrics

- LOC (Lines of Code):
  - Simplest and most widely used metric.
  - Comments and blank lines should not be counted.
  - Accurate estimation of LOC count at the beginning of a project is a very difficult task.
  - One can possibly estimate the LOC count at the starting of a project, only by using some form of systematic guess work.
  - Systematic guessing typically involves the following.

Software Size Metrics

- The project manager divides the problem into modules, and each module into sub-modules and so on, until the LOC of the leaf-level modules are small enough to be predicted.
- To be able to predict the LOC count for the various leaf-level modules sufficiently accurately, past experience in developing similar modules is very helpful.
- By adding the estimates for all leaf level modules together, project managers arrive at the total size estimation.
- In spite of its conceptual simplicity, LOC metric has several shortcomings when used to measure problem size.

Disadvantages of Using LOC

- Size can vary with coding style.
- Focuses on coding activity alone.
- Correlates poorly with quality and efficiency of code.
- Penalizes higher level programming languages, code reuse, etc.
- Measures lexical/textual complexity only.
  - does not address the issues of structural or logical complexity.
- Difficult to estimate LOC from problem description.
  - So not useful for project planning

Function Point Metric

- Overcomes some of the shortcomings of the LOC metric
- Proposed by Albrecht in early 80's:
- The size of a software product is directly dependent on the number of different high-level functions or features it supports.
- This assumption is reasonable, since each feature would take additional effort to implement.

Function Point Metric

- Different features may take very different amounts of efforts to develop.
- *For example, in a banking software, a function to display a help message may be much easier to develop compared to say the function that carries out the actual banking transactions.*
- This has been considered by the function point metric by counting the number of input and output data items and the number of files accessed by the function.

Function Point Metric

- Input :
  - A set of data items input by the user.
- Output:
  - A set of related outputs is counted as one output.
- Inquiries:
  - Each user query type is counted.
- Files:
  - Files are logically related data and thus can be data structures or physical files.
- Interface:
  - Data transfer to other systems.

Function point (FP) metric computation

- It is computed using the following three

Steps:

- Step 1: Compute the unadjusted function point (UFP) using a heuristic expression.
- Step 2: Refine UFP to reflect the actual complexities of the different parameters used in UFP computation.
- Step 3: Compute FP by further refining UFP to account for the specific characteristics of the project that can influence the entire development effort.

Step:1 UFP Computation

- The unadjusted function points (UFP) is computed as the weighted sum of five characteristics of a product.
- The weights associated with the five characteristics were determined empirically by Albrecht through data gathered from many projects.

*UFP = (Number of inputs)*4 + (Number of outputs)*5 + (Number of inquiries)*4 + (Number of files)*10 + (Number of interfaces)*10*

Step 2: Refine parameters

- The complexity of each parameter is graded into three broad categories—simple, average, or complex.
- The weights for the different parameters are determined based on the numerical values shown in Table.
- Based on these weights of the parameters, the parameter values in the UFP are refined.

Type	Simple	Average	Complex
Input(I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
Number of files (F)	7	10	15
Number of interfaces	5	7	10

Step 3: Refine UFP based on complexity

- Several factors that can impact the overall project size are considered to refine the UFP computed such as high transaction rates, response time requirements, scope for reuse, etc.
- 14 such parameters that can influence the development effort have been identified.
- Each of these 14 parameters is assigned a value from 0 (not present or no influence) to 6 (strong influence).
- The resulting numbers are summed, yielding the total degree of influence (DI).
- A technical complexity factor (TCF) for the project is computed.

Step 3: Refine UFP based on complexity

- TCF is computed as

  $TCF = (0.65 + 0.01 * DI)$.

- As DI can vary from 0 to 84, TCF can vary from 0.65 to 1.49.
- Finally, FP is given as the product of UFP and TCF.

  $FP = UFP * TCF$.

Function points relative complexity adjustment factors

Requirement for reliable backup and recovery
Requirement for data communication
Extent of distributed processing
Performance requirements
Expected operational environment
Extent of online data entries
Extent of multi-screen or multi-operation online data input
Extent of online updating of master files
Extent of complex inputs, outputs, online queries and files
Extent of complex data processing
Extent that currently developed code can be designed for reuse
Extent of conversion and installation included in the design
Extent of multiple installations in an organisation and variety of customer organisations
Extent of change and focus on ease of use

Function Point Metric

External Inputs – IFPUG Definition:

- An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary
- The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system
- **Example:**
  - Data entry by users
  - Data or file feeds by external applications

Fur

External Outputs – IFPUG Definition:

- An external output (EO) is an elementary process that sends data or control information outside the application boundary
- The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information
- The processing logic must contain at least one mathematical formula or calculation, create derived data, maintain one or more ILFs, or alter the behavior of the system

- **Example:**
- Reports created by the application being counted, where the reports include derived information

Fu

External Inquiries – IFPUG Definition:

- An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary
- The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF
- The processing logic contains no mathematical formulas or calculations, and creates no derived data
- No ILF is maintained during the processing, nor is the behavior of the system altered
- **Example:**
  - Reports created by the application being counted, where the report does not include any derived data

Internal Logical Files – IFPUG Definition:

- An ILF is a user-identifiable group of logically related data or control information maintained within the boundary of the application
- The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted

- **Example:**
  - Tables in a relational database
  - Files
  - Application control information, perhaps things like user preferences that are stored by the application

Complexity Assessment

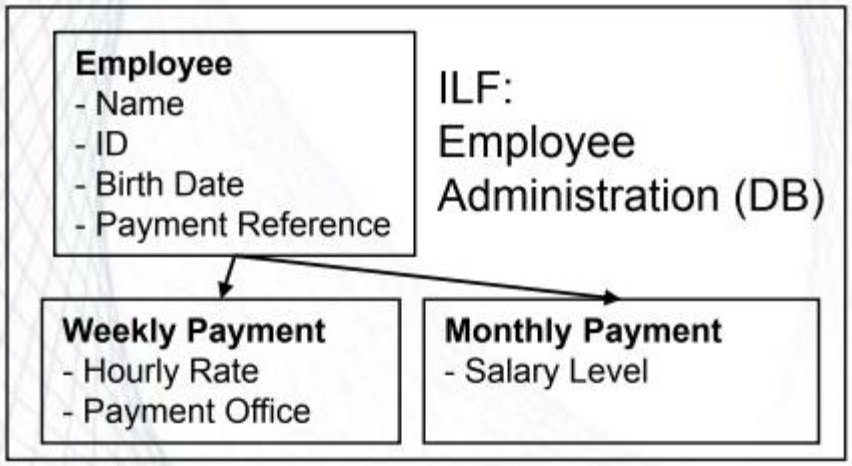
	Data Function Types		Transaction Function Types		
	Internal Logical Files (ILF)	External Interface Files (EIF)	External Input (EI)	External Output (EO)	External Inquiry (EQ)
Elements evaluated for technical complexity assessment	**REcord Types (RET)**: User recognizable sub groups of data elements within an ILF or an EIF. It is best to look at logical groupings of data to help identify them.		**File Type Referenced (FTR):** File type referenced by a transaction. An FTR must be an Internal Logical File (ILF) or External Interface File (EIF).		
	Data Element Types (DET): A unique user recognizable, non-recursive (non-repetitive) field containing dynamic information. If a DET is recursive then only the first occurrence of the DET is considered not every occurrence.				

Complexity Assessment: EI

- Identify number of File Types Referenced (FTR)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→ used to calculate FP count)

EI		#DET		
		1-4	5-15	> 15
#FTR	1	low (3)	low (3)	average (4)
	2	low (3)	average (4)	high (6)
	> 2	average (4)	high (6)	high (6)

Complexity Assessment :EI

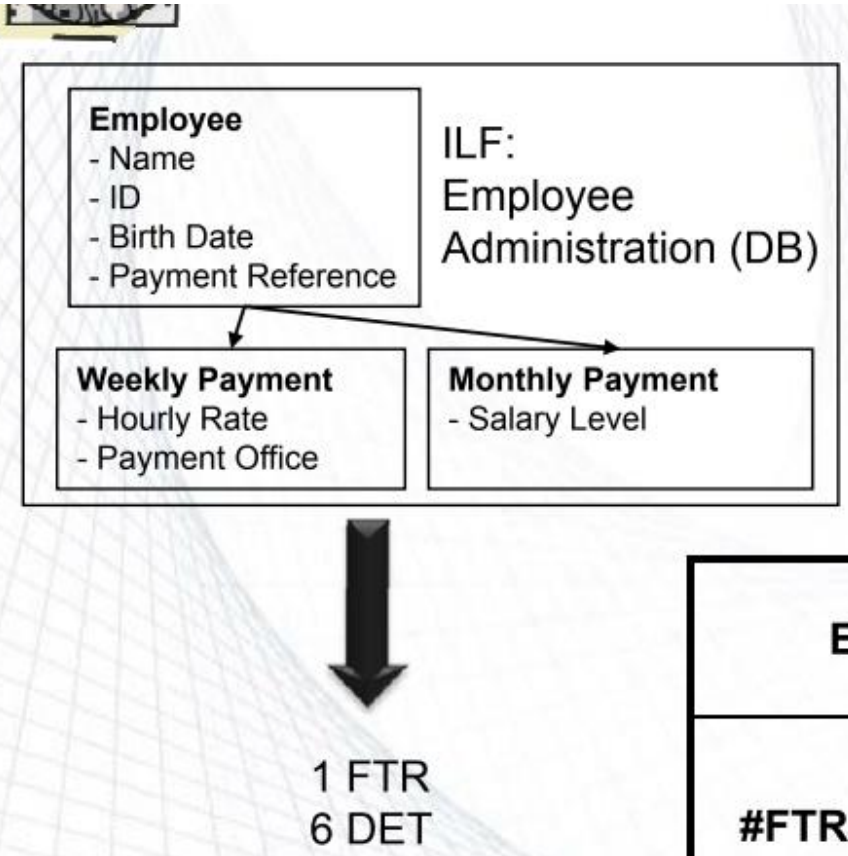


- Enter a new employee with Monthly Payment:
  - Name
  - ID
  - Birth Date
  - Payment Reference
  - Salary Level

1 FTR
5 DET

EI		#DET		
		1-4	5-15	> 15
#FTR	1	low (3)	low (3)	average (4)
	2	low (3)	average (4)	high (6)
	> 2	average (4)	high (6)	high (6)

Complexity Assessment: EI



- Enter a new employee with Weekly Payment:
  - Name
  - ID
  - Birth Date
  - Payment Reference
  - Hourly Rate
  - Payment Office

EI		#DET		
		1-4	5-15	> 15
#FTR	1	low (3)	low (3)	average (4)
	2	low (3)	average (4)	high (6)
	> 2	average (4)	high (6)	high (6)

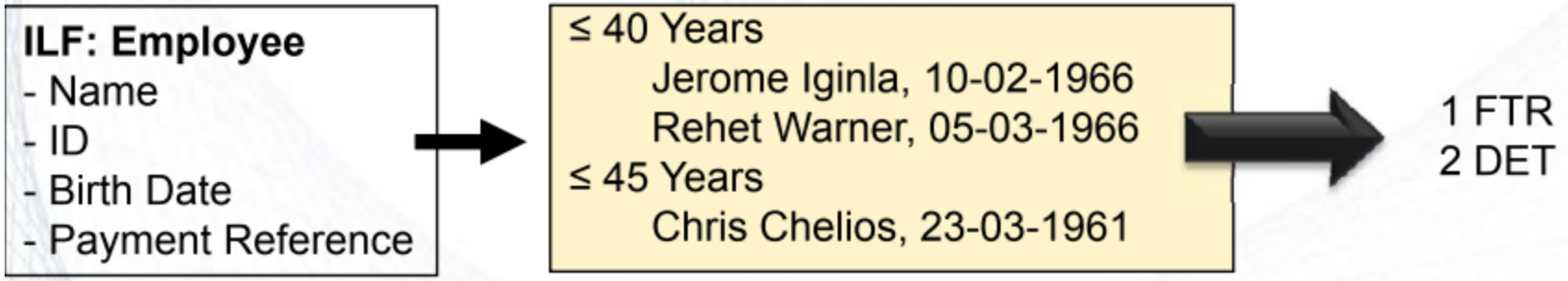
Complexity Assessment: EO

- Identify number of File Types Referenced (FTR)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→ used to calculate FP count)

EO		#DET		
		1-5	6-19	> 19
#FTR	1	low (4)	low (4)	average (5)
	2-3	low (4)	average (5)	high (7)
	> 3	average (5)	high (7)	high (7)

Complexity Assessment: EO

- Report of all Employees containing Names and Birth Dates, sorted by age.



EO		#DET		
		1-5	6-19	> 19
#FTR	1	low (4)	low (4)	average (5)
	2-3	low (4)	average (5)	high (7)
	> 3	average (5)	high (7)	high (7)

Complexity Assessment: EQ

- Identify number of File Types Referenced (FTR)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→ used to calculate FP count)

EQ		#DET		
		1-5	6-19	> 19
#FTR	1	low (3)	low (3)	average (4)
	2-3	low (3)	average (4)	high (6)
	> 3	average (4)	high (6)	high (6)

Complexity Assessment: EQ

- Report of all employees belonging to Department X containing Names, Birth Dates, and showing the Department Name
- Files (ILF): Employee, Department
- 2 FTR: Employee, Department
  3 DET: Name (Employee), Birth Date (Employee), Department Name (Department)

EQ		#DET		
		1-5	6-19	> 19
#FTR	1	low (3)	low (3)	average (4)
	2-3	low (3)	average (4)	high (6)
	> 3	average (4)	high (6)	high (6)

Complexity Assessment: ILF

- Identify number of Record Types (RET)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→ used to calculate FP count)

ILF		#DET		
		1-19	20-50	> 50
#RET	1	low (7)	low (7)	average (10)
	2-5	low (7)	average (10)	high (15)
	> 5	average (10)	high (15)	high (15)

Complexity Assessment: ILF

ILF: "Employee" = **Employee**
- Name
- ID
- Birth Date
- Payment Reference

→ 1 RET, 4 DET

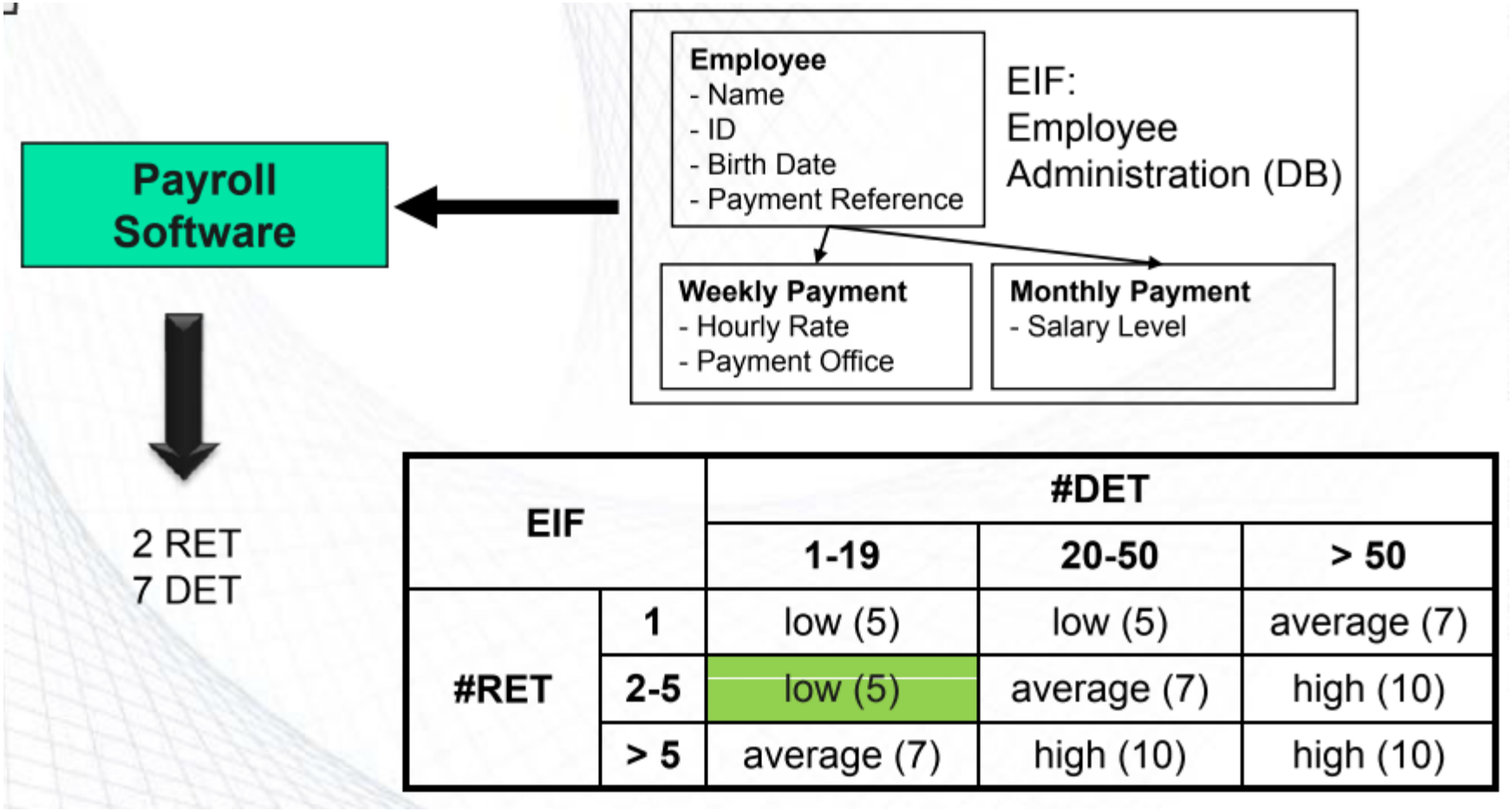
ILF		#DET		
		1-19	20-50	> 50
#RET	1	low (7)	low (7)	average (10)
	2-5	low (7)	average (10)	high (15)
	> 5	average (10)	high (15)	high (15)

Complexity Assessment: EIF

- Identify number of Record Types (RET)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→ used to calculate FP count)

EIF		#DET		
		1-19	20-50	> 50
#RET	1	low (5)	low (5)	average (7)
	2-5	low (5)	average (7)	high (10)
	> 5	average (7)	high (10)	high (10)

Complexity Assessment: EIF



EIF		#DET		
		1-19	20-50	> 50
#RET	1	low (5)	low (5)	average (7)
	2-5	low (5)	average (7)	high (10)
	> 5	average (7)	high (10)	high (10)

Function Point Metric: Example

- A supermarket needs to develop the following software to encourage regular customers. For this, the customer needs to supply his/her residence address, telephone number, and the driving license number. Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. Based on the generated CN, a clerk manually prepares a customer identity card after getting the market manager's signature on it. A customer can present his customer identity card to the check out staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22 caret gold coin to every customer whose purchase exceeded Rs. 10,000. The entries against the CN are reset on the last day of every year after the prize winners' lists are generated. Assume that various project characteristics determining the complexity of software development to be average.

Function Point Metric: Example

- **Step 1:** we find that there are two inputs, three outputs, two files, and no interfaces. Two files would be required, one for storing the customer details and another for storing the daily purchase records.

  $UFP = 2 \times 4 + 3 \times 5 + 1 \times 4 + 10 \times 2 + 0 \times 7 = 47$

- **Step 2:** All the parameters are of moderate complexity, except the output parameter of customer registration. Complexity of the output parameter of the customer registration function can be categorized as simple.
- The UFP can be refined as follows:

  $UFP = 3 \times 4 + 2 \times 5 + 1 \times 4 + 10 \times 2 + 0 \times 10 = 46$

- **Step 3:** The complexity adjustment factors have average values,

  $DI = 14 \times 4 = 56$

  $TCF = 0.65 + 0.01 * 56 = 1.21$

  $FP=46 * 1.21 = 55.66$

Function Point Metric: Example

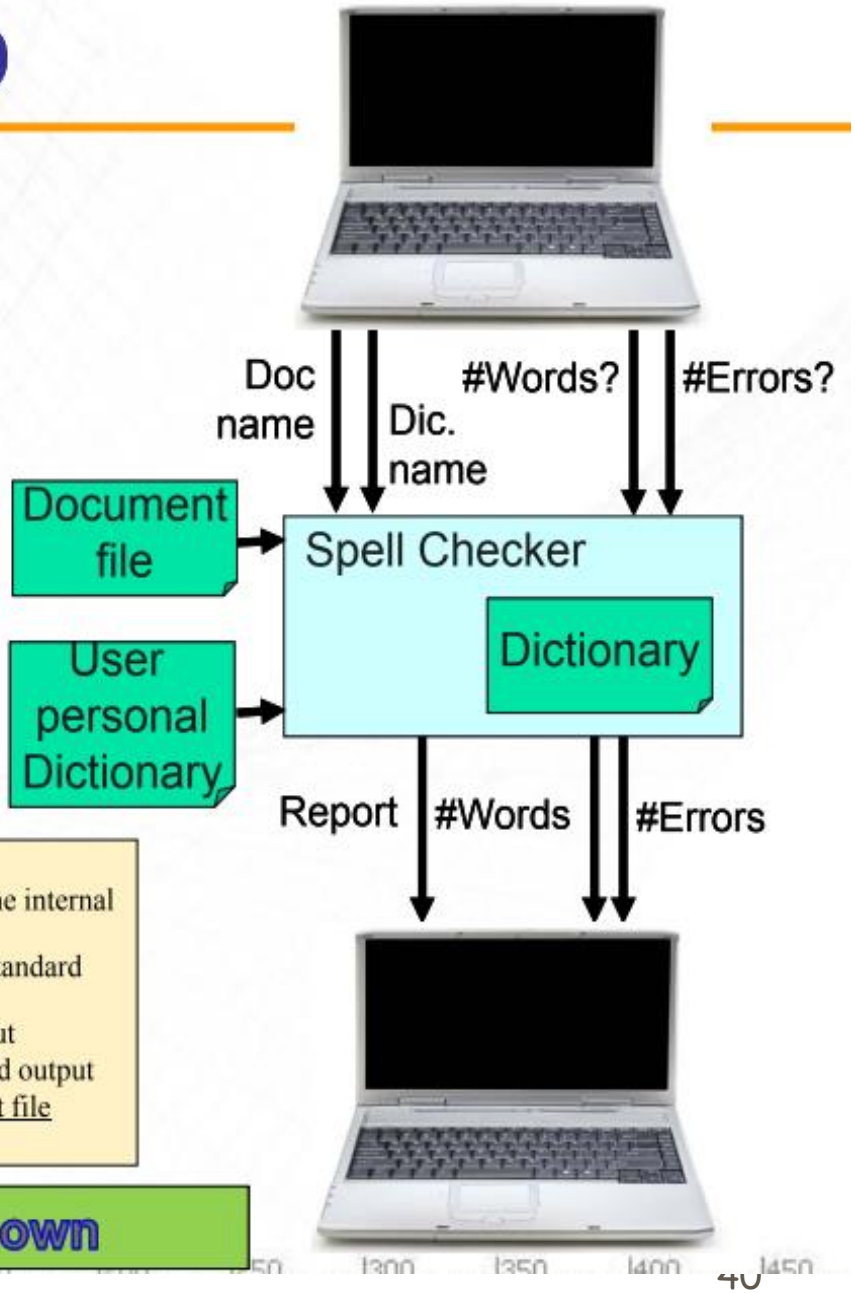
Specification: Spell Checker

- Checks all words in a document by comparing them to a list of words in the internal dictionary and an optional user-defined dictionary
- After processing the document sends a report on all misspelled words to standard output
- On request from user shows number of words processed on standard output
- On request from user shows number of spelling errors detected on standard output
- Requests can be issued at any point in time while processing the document file

FP Example /1b

Spell Checker

- Convert spec to a diagram



Specification:

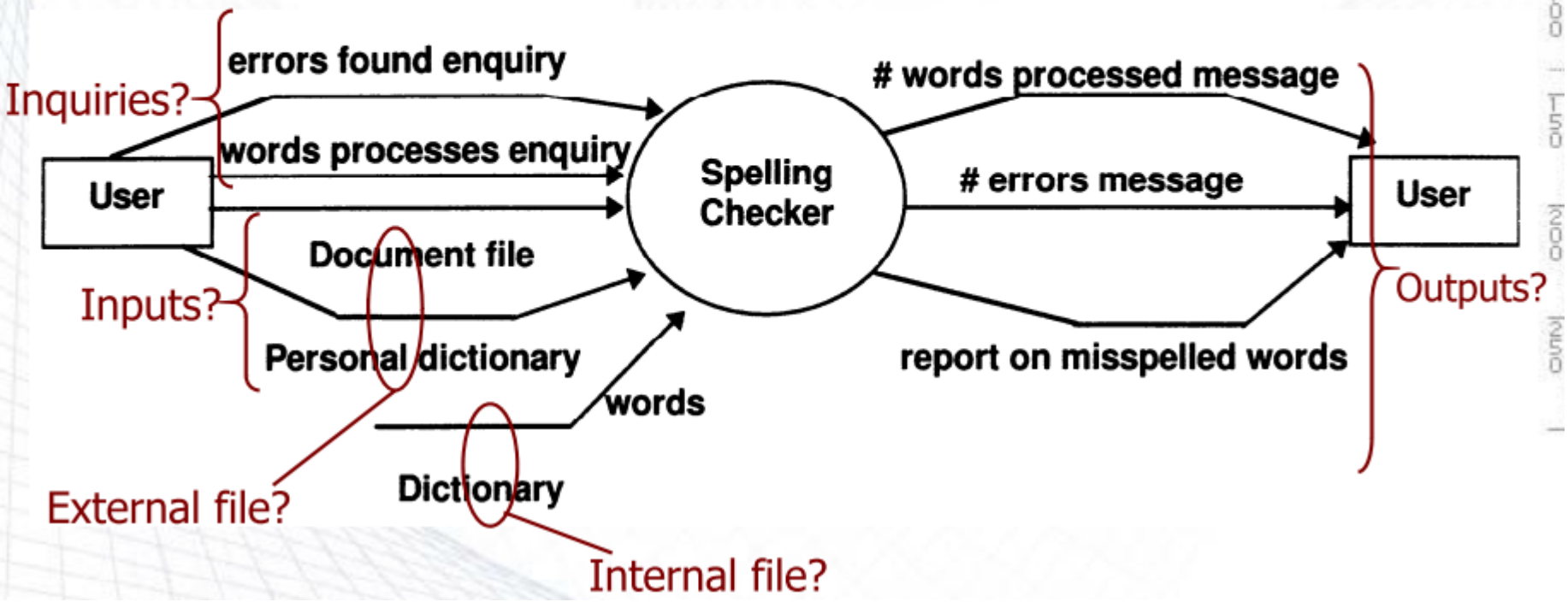
1. Checks all words in a document by comparing them to a list of words in the internal dictionary and an optional user-defined dictionary
2. After processing the document sends a report on all misspelled words to standard output
3. On request from user shows number of words processed on standard output
4. On request from user shows number of spelling errors detected on standard output
5. Requests can be issued at any point in time while processing the document file

Last req. cannot be shown

Function Point Metric: Example

Spell Checker

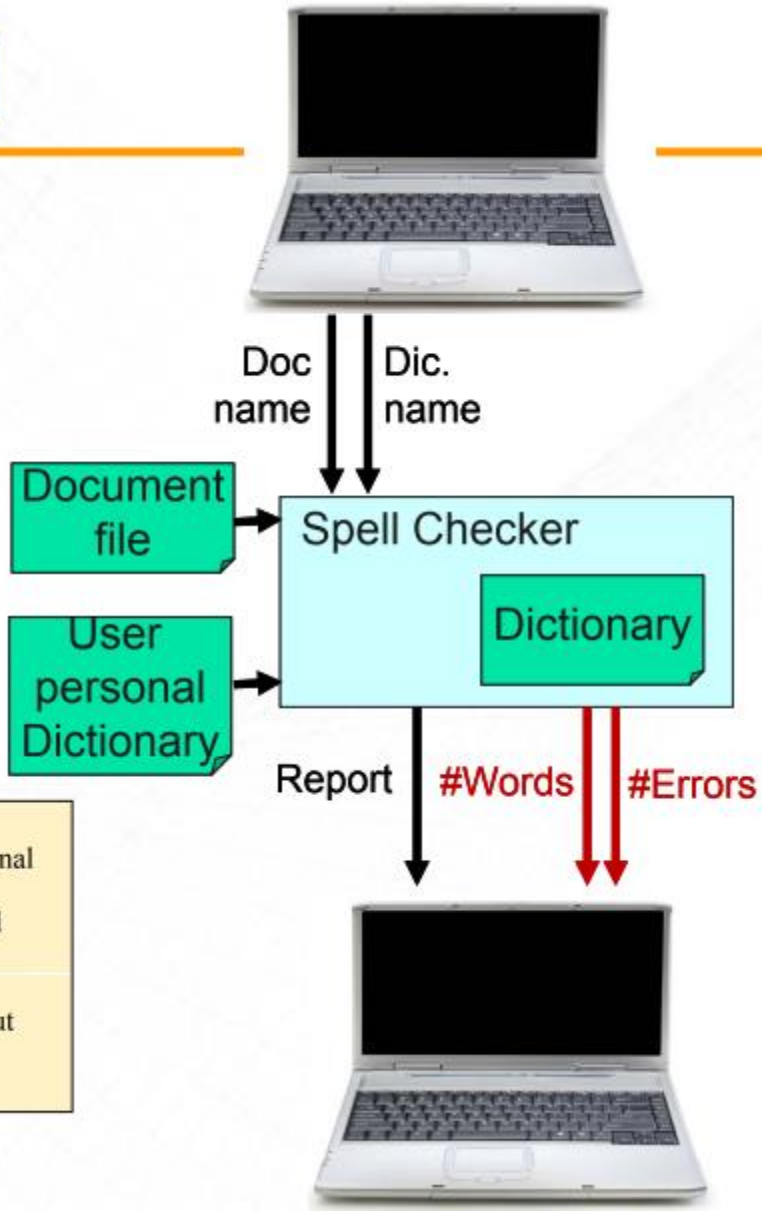
- Convert spec to a diagram



FP Example /1d

Solution A:

- EI: Doc. name + User Dic. name → 2
- EO: Report + #Words + #Errors → 3
- EQ: --
- EIF: Document + User Dictionary → 2
- ILF: Dictionary → 1



Specification:

1. Checks all words in a document by comparing them to a list of words in the internal dictionary and an optional user-defined dictionary
2. After processing the document sends a report on all misspelled words to standard output
3. On request from user shows number of words processed on standard output
4. On request from user shows number of spelling errors detected on standard output
5. Requests can be issued at any point in time while processing the document file

EI EO EQ EIF ILF

Function Point Metric: Example

Element	Weighting Factor		
	Simple	Average	Complex
External inputs (*EI*)	3	4	6
External outputs (*EO*)	4	5	7
External inquiries (*EQ*)	3	4	6
External interface files (*EIF*)	5	7	10
Internal logical files (*ILF*)	7	10	15

- $N_{EI}=2$ (number of EI): document name, user-defined dictionary name
- $N_{EO}=3$ (number of EO): misspelled word report, number-of-words-processed message, number-of-errors-so-far message
- $N_{EQ}=0$ (number of EQ): --
- $N_{EIF}=2$ (number of EIF): document file, personal dictionary
- $N_{ILF}=1$ (number of ILF): dictionary

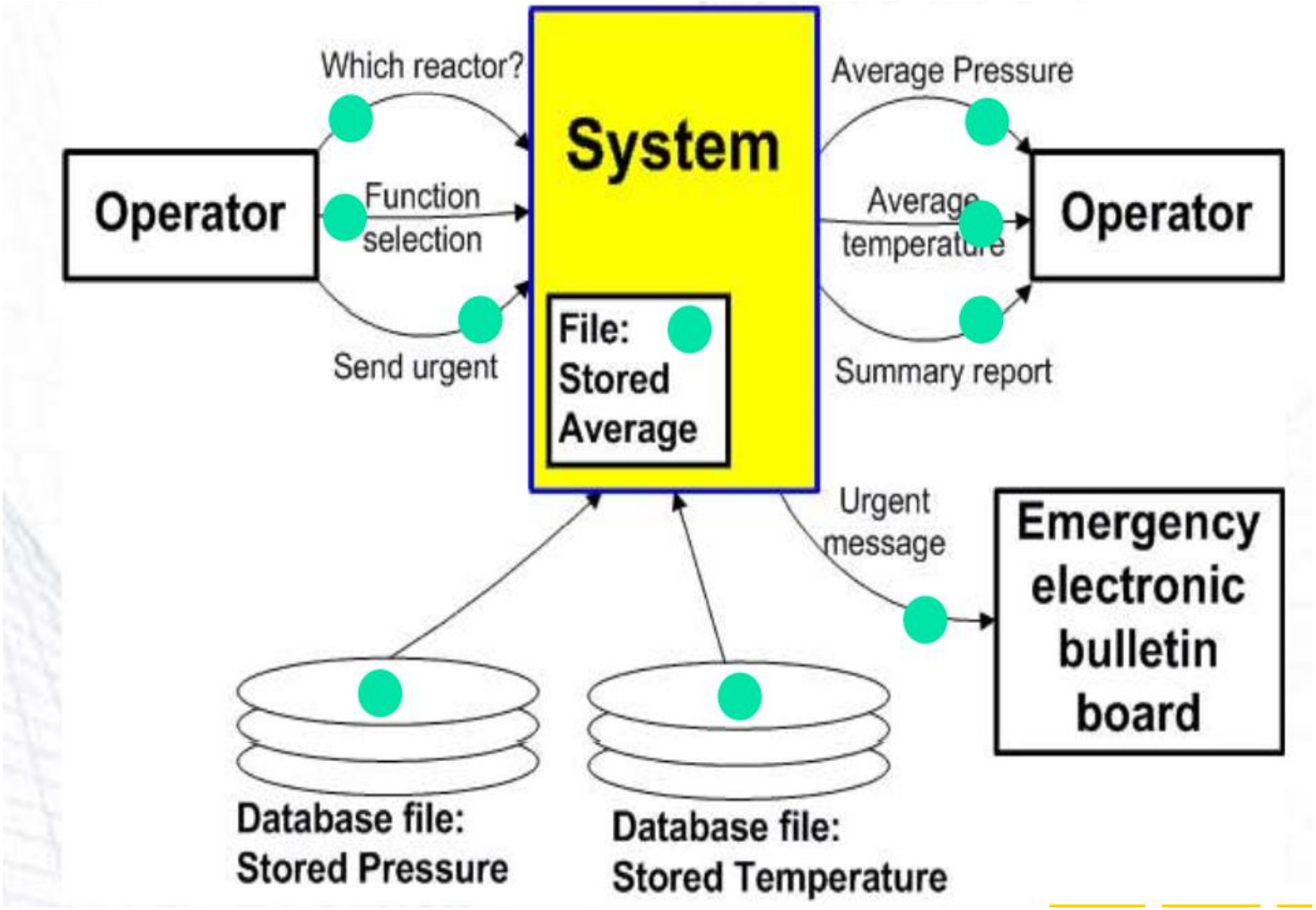
FP: Example

Consider the following system that processes various commands from the operator of a chemical plant. Various interactions of the system with the operator and internal and external files are shown.

The system consults two databases for stored pressure and temperature readings. In the case of emergency, the system asks the user whether to send the results to an electronic emergency bulletin board or not.

Calculate the unadjusted function point count (UFC) for this system.

FP: Example


Which reactor?
Function selection
Send urgent
Operator
System
File: Stored Average
Average Pressure
Average temperature
Summary report
Operator
Urgent message
Emergency electronic bulletin board
Database file: Stored Pressure
Database file: Stored Temperature

FP: Example

N_{EI} number of external inputs **2**

N_{EO} number of external outputs **3**

N_{EQ} number of external inquiries **1**

N_{EIF} number of external interface files **2**

N_{ILF} number of internal logical files **1**

UFC = 4x2 + 5x3 + 4x1 + 7x2 + 10x1 = 51

FP Vs. LOC

- A number of studies have attempted to relate LOC and FP metrics (Jones, 1996).
- The average number of source code statements per function point has been derived from case studies for numerous programming languages.
- Languages have been classified into different levels according to the relationship between LOC and FP.

FP Vs. LOC

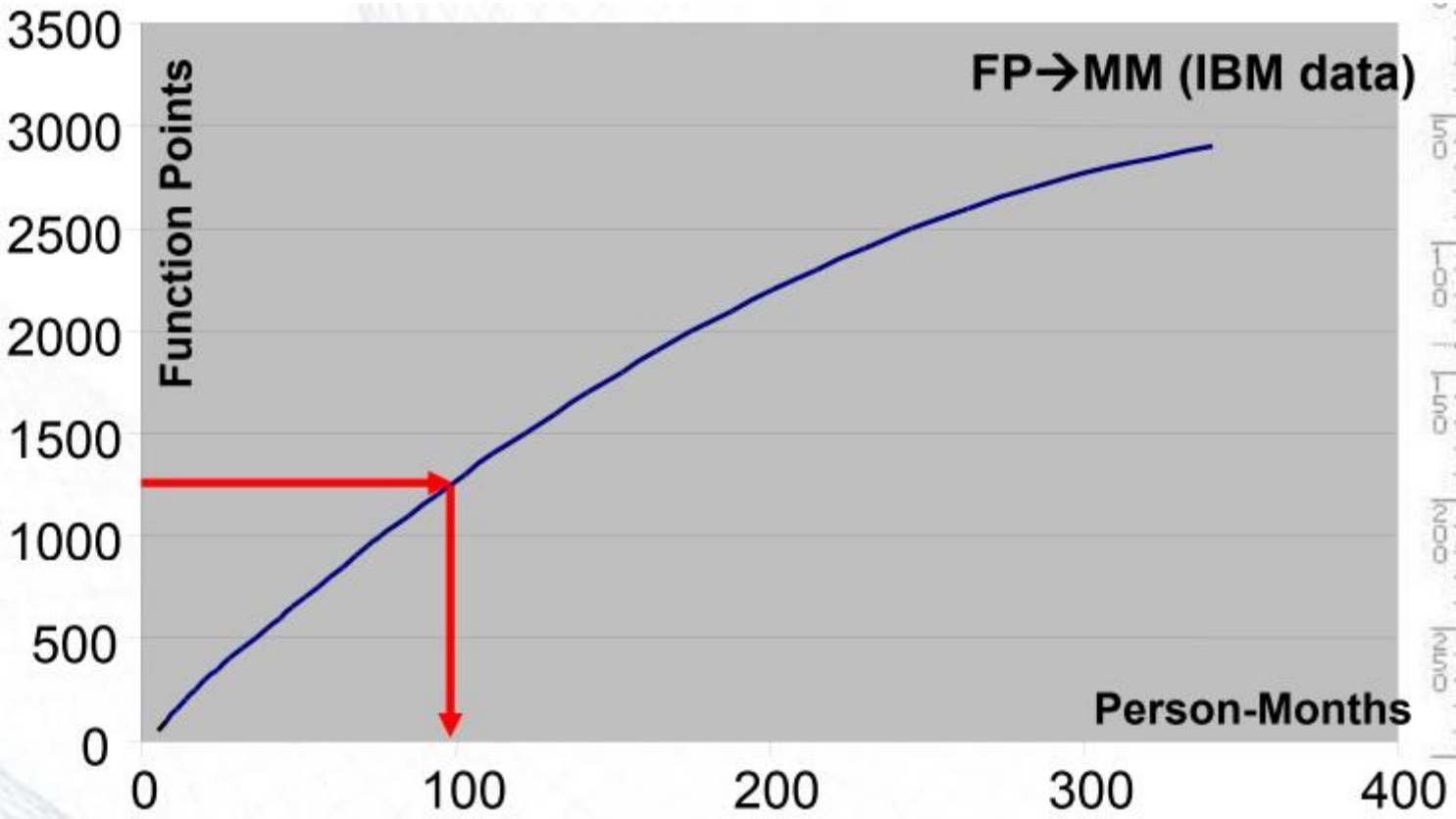
Language	Typical SLOC per FP
1GL Default Language	320
2GL Default Language	107
3GL Default Language	80
4GL Default Language	20
Code generators	15
Assembler	320
C	148
Basic	107
Pascal	90

FP Vs. LOC

Language	Typical SLOC per FP
C#	59
C++	60
PL/SQL	46
Java 2	60
Visual Basic	50
Delphi	18
HTML 4	14
SQL	13
Excel	47

FP Vs. LOC

Effort estimation based on organization-specific data from past projects.



Function Point Metric

- Suffers from a major drawback:
  - the size of a function is considered to be independent of its complexity.
- Extend function point metric:
  - Feature Point metric:
  - considers an extra parameter:
    - Algorithm Complexity.

Function Point Metric

- Proponents claim:
  - FP is language independent.
  - Size can be easily derived from problem description
- Opponents claim:
  - it is subjective --- Different people can come up with different estimates for the same problem.

Empirical Size Estimation Techniques

- Expert Judgement:
  - An euphemism for guess made by an expert.
  - Suffers from individual bias.
- Delphi Estimation:
  - overcomes some of the problems of expert judgement.

Expert judgement

- Experts divide a software product into component units:
  - e.g. GUI, database module, data communication module, billing module, etc.
- Add up the guesses for each of the components.

Delphi Estimation

- Team of Experts and a coordinator.
- Experts carry out estimation independently:
  - mention the rationale behind their estimation.
  - coordinator notes down any extraordinary rationale:
    - circulates among experts.
- Experts re-estimate.
- Experts never meet each other
  - to discuss their viewpoints.

Halstead's Software Science

- An analytical technique to estimate:
  - size,
  - development effort,
  - development time.

Halstead's Software Science

- Halstead used a few primitive program parameters
  - number of operators and operands
- Derived expressions for:
  - over all program length,
  - potential minimum volume
  - actual volume,
  - language level,
  - effort, and
  - development time.

Halstead's Software Science

- For a given program, let:
  - h1 be the number of unique operators used in the program,
  - h2 be the number of unique operands used in the program,
  - N1 be the total number of operators used in the program,
  - N2 be the total number of operands used in the program.

Halstead's Software Science

- For instance,
  - assignment, arithmetic, and logical operators are usually counted as operators.
  - A pair of parentheses, as well as a block begin Block end pair, are considered as single operators.
  - A label is considered to be an operator, if it is used as the target of a GOTO statement.
  - The constructs if ... then ... else ... endif and a while ... Do are considered as single operators.
  - termination operator ';' is considered as a single operator.
  - Subroutine declarations and variable declarations comprise the operands.
  - Function name in a function call statement is considered as an operator, and the arguments of the function call are considered as operands.
  - etc.

Halstead's Software Science

- Program Volume – Proportional to program size, represents the size, in bits, of space necessary for storing the program.
- This parameter is dependent on specific algorithm implementation.
- Potential Minimum Volume – The potential minimum volume V* is defined as the volume of the most succinct program in which a problem can be coded.
- Program Level – To rank the programming languages, the level of abstraction provided by the programming language, Program Level (L) is considered.
- Programming Effort – Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language.
- Language Level – Shows the algorithm implementation program language level.

Halstead's Software Science

- length N = N1 + N2
- program vocabulary h = h1 + h2
- Program Volume V = N log2 h
- Potential Minimum Volume V* = (2 + h2) log2 (2 + h2)
- program level L = V*/V
- Effort and Time E = V /L
- programmer's time T = E/S
- **(S=18 is recommended)*

Halstead's Software Science: Example

```
int sort (int x[ ], int n)
{
    int i, j, save, im1;
    If (n< 2) return 1;
    for (i=2; i< =n; i++)
    {
        im1=i-1;
        for (j=1; j< =im1; j++)
            if (x[i] < x[j ])
            {
                Save = x[i];
                x[i] = x[j ];
                x[j ] = save;
            }
    }
    return 0; }
```

Halstead's Software Science: Example

operators	occurrences	operands	occurrences
int	4	sort	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3
;	11	im1	3
for	2	2	2
=	6	1	3
-	1	0	1
<=	2	-	-
++	2	-	-
return	2	-	-
{}	3	-	-
n1=14	N1=53	n2=10	N2=38

Halstead's Software Science: Example

N = 91

n = 24

V = 417 bits

N^ = 86.45

n2 = 3 (x: array holding integer o be sorted. This is used both as input and output)*

V = 11.6*

L = 0.027

D = 37.03

L^ = 0.038

T = 610 seconds

Heuristic Estimation Techniques

- Single Variable Model:
  - Parameter to be Estimated = C1 * (Estimated Characteristic)^{d1}
- Multivariable Model:
  - Assumes that the parameter to be estimated depends on more than one characteristic.
  - Parameter to be Estimated= C1(Estimated Characteristic)^{d1}+ C2(Estimated Characteristic)d^{2}+…
  - Usually more accurate than single variable models.

COCOMO Model: A Heuristic Estimation Technique

- COCOMO (COnstructive COst MOdel) proposed by Boehm.
- Divides software product developments into 3 categories:
  - Organic
  - Semidetached
  - Embedded

COCOMO Product classes

- Roughly correspond to:
  - application, utility and system programs respectively.
    - Data processing and scientific programs are considered to be application programs.
    - Compilers, linkers, editors, etc., are utility programs.
    - Operating systems and real-time system programs, etc. are system programs.

Elaboration of Product classes

- Organic:
  - Relatively small groups
    - working to develop well-understood applications.
- Semidetached:
  - Project team consists of a mixture of experienced and inexperienced staff.
- Embedded:
  - The software is strongly coupled to complex hardware, or real-time systems.

COCOMO Model

- For each of the three product categories:
  - From size estimation (in KLOC), Boehm provides equations to predict:
    - project duration in months
    - effort in programmer-months
- Boehm obtained these equations:
  - examined historical data collected from a large number of actual projects.

COCOMO Model

- Software cost estimation is done through three stages:
  - Basic COCOMO,
  - Intermediate COCOMO,
  - Complete COCOMO.

Basic COCOMO Model

- Gives only an approximate estimation:
  - Effort = a1 * (KLOC) * a2
  - Tdev = b1 * (Effort) * b2
    - KLOC is the estimated kilo lines of source code,
    - a1,a2,b1,b2 are constants for different categories of software products,
    - Tdev is the estimated time to develop the software in months,
    - Effort estimation is obtained in terms of person months (PMs).

Development Effort Estimation

- Organic :
  - Effort = 2.4 $(KLOC)^{1.05}$ PM
- Semi-detached:
  - Effort = 3.0$(KLOC)^{1.12}$ PM
- Embedded:
  - Effort = 3.6 $(KLOC)^{1.20}$PM

Development Time Estimation

- Organic:
  - Tdev = 2.5 (Effort)$^{0.38}$ Months
- Semi-detached:
  - Tdev = 2.5 (Effort)$^{0.35}$ Months
- Embedded:
  - Tdev = 2.5 (Effort)$^{0.32}$ Months

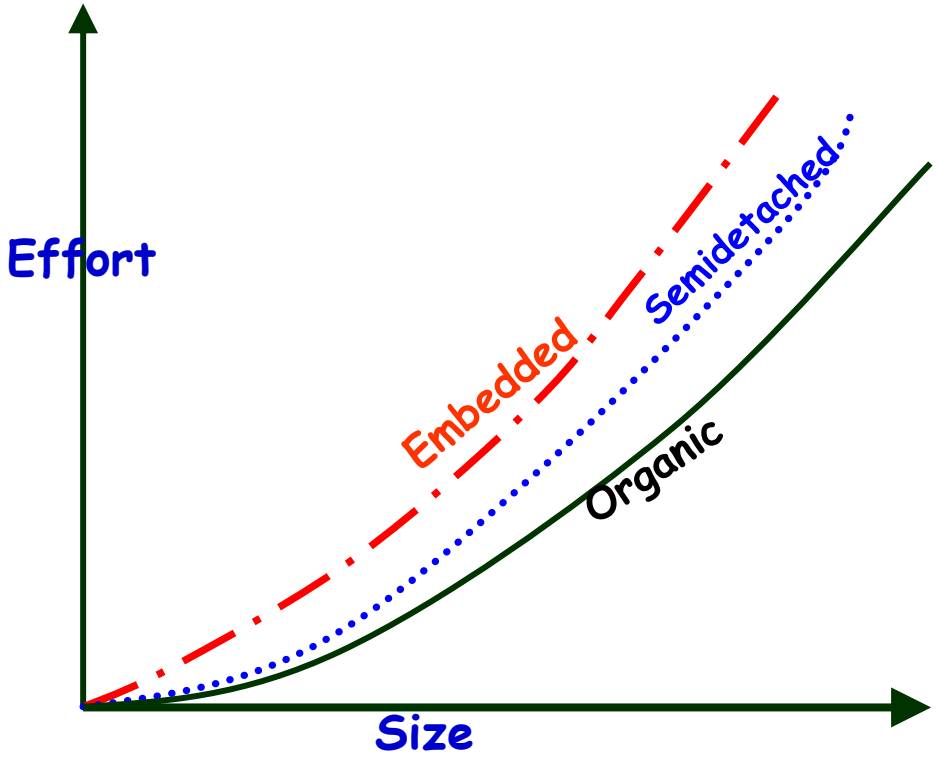
Example

- The size of an organic software product has been estimated to be 32,000 lines of source code.

- Effort = $2.4*(32)^{1.05}$ = 91 PM
- Nominal development time = $2.5*(91)^{0.38}$ = 14 months

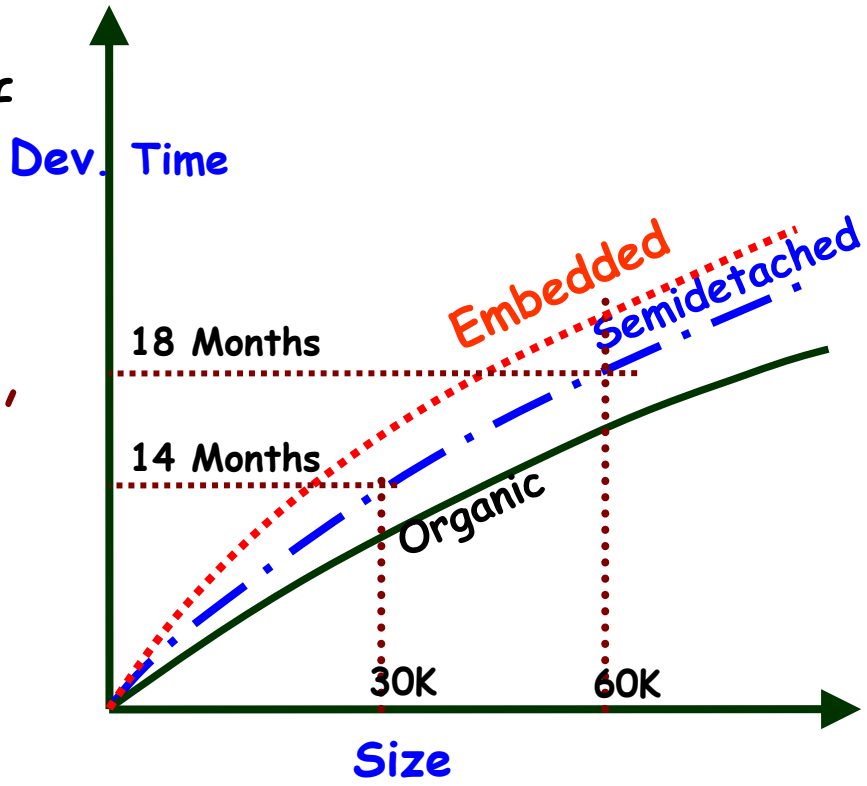
Basic COCOMO Model (CONT.)

- Effort is somewhat super-linear in problem size.



Basic COCOMO Model

- Development time
  - sublinear function of product size.
- When product size increases two times,
  - development time does not double.
- Time taken:
  - almost same for all the three product categories.



Basic COCOMO Model

- Development time does not increase linearly with product size:
  - For larger products more parallel activities can be identified:
    - can be carried out simultaneously by a number of engineers.

Basic COCOMO Model

- Development time is roughly the same for all the three categories of products:
  - For example, a 60 KLOC program can be developed in approximately 18 months
    - regardless of whether it is of organic, semi-detached, or embedded type.
  - There is more scope for parallel activities for system and application programs,
    - than utility programs.

Intermediate COCOMO

- Basic COCOMO model assumes
  - effort and development time depend on product size alone.
- However, several parameters affect effort and development time:
  - Reliability requirements
  - Availability of CASE tools and modern facilities to the developers
  - Size of data to be handled

Intermediate COCOMO

- For accurate estimation,
  - the effect of all relevant parameters must be considered:
  - Intermediate COCOMO model recognizes this fact:
    - refines the initial estimate obtained by the basic COCOMO by using a set of 15 cost drivers (multipliers).

Intermediate COCOMO

- If modern programming practices are used,
  - initial estimates are scaled downwards.
- If there are stringent reliability requirements on the product :
  - initial estimate is scaled upwards.
- Rate different parameters on a scale of one to three:
  - Depending on these ratings,
    - multiply cost driver values with the estimate obtained using the basic COCOMO.

Intermediate COCOMO

- Cost driver classes:
  - Product: Inherent complexity of the product, reliability requirements of the product, etc.
  - Computer: Execution time, storage requirements, etc.
  - Personnel: Experience of personnel, etc.
  - Development Environment: Sophistication of the tools used for software development.

Shortcoming of basic and intermediate COCOMO models

- Both models:
  - consider a software product as a single homogeneous entity:
  - However, most large systems are made up of several smaller sub-systems.
    - Some sub-systems may be considered as organic type, some may be considered embedded, etc.
    - for some the reliability requirements may be high, and so on.

Complete COCOMO

- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduces the margin of error in the final estimate.

Complete COCOMO Example

- A Management Information System (MIS) for an organization having offices at several places across the country:
  - Database part (semi-detached)
  - Graphical User Interface (GUI) part (organic)
  - Communication part (embedded)
- Costs of the components are estimated separately:
  - summed up to give the overall cost of the system.

COCOMO 2

- The present day software projects are much larger in size and reuse of existing software to develop new products has become pervasive.
- To make COCOMO suitable in the changed scenario, Boehm proposed COCOMO 2 in 1995.
- COCOMO 2 provides three models to arrive at increasingly accurate cost estimations.
- These can be used to estimate project costs at different phases of the software product.
- As the project progresses, these models can be applied at the different stages of the same project.

COCOMO 2

Application composition model:

- This model as the name suggests, can be used to estimate the cost for prototype development. (uses object points)

Early design model:

- This supports estimation of cost at the architectural design stage. (uses Functional points)

Post-architecture model:

This provides cost estimation during detailed design and coding stages. (Uses LOC)

Application composition model

- It is based on counting the number of screens, reports, and modules (components).
- Each of these components is considered to be an object.
- These are used to compute the object points of the application.

Application composition model

- Effort is estimated in the application composition model as follows:
- 1. Estimate the number of screens, reports, and modules (components) from an analysis of the SRS document.
- 2. Determine the complexity level of each screen and report, and rate these as either simple, medium, or difficult.
- The complexity of a screen or a report is determined by the number of tables and views it contains.

Application composition model

- Use the weight values in Tables.
- The weights have been designed to correspond to the amount of effort required to implement an instance of an object at the assigned complexity class.

SCREEN Complexity Assignments for the Data Tables

Number of views	Tables < 4	Tables < 8	Tables ≥ 8
< 3	Simple	Simple	Medium
3–7	Simple	Medium	Difficult
>8	Medium	Difficult	Difficult

Application composition model

Report Complexity Assignments for the Data Tables

Number of views	Tables < 4	Tables < 8	Tables ≥ 8
0 or 1	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
4 or more	Medium	Difficult	Difficult

- Add all the assigned complexity values for the object instances together to obtain the object points.

Application composition model

Table of Complexity Weights for Each Class for Each Object Type

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

- Estimate percentage of reuse expected in the system. Note that reuse refers to the amount of pre-developed software that will be used within the system.
- Then, evaluate New Object-Point count (NOP) as follows

$$NOP = \frac{(\text{Object-Points})(100 - \%\text{ of reuse})}{100}$$

Application composition model

- Determine the productivity using Table. The productivity depends on the experience of the developers as well as the maturity of the CASE environment used.
- Finally, the estimated effort in person-months is computed as ***E = NOP/PROD***.

Productivity Table

Developers' experience	Very low	Low	Nominal	High	Very high
CASE maturity	Very low	Low	Nominal	High	Very high
PRODUCTIVITY	4	7	13	25	50

Early design model

- The unadjusted function points (UFP) are counted and converted to source lines of code (SLOP).
- In a typical programming environment, each UFP would correspond to about 128 lines of C, 29 lines of C++, or 320 lines of assembly code.
- The conversion from UFP to LOC is environment specific, and depends on factors such as extent of reusable libraries supported.
- Seven cost drivers that characterize the post-architecture model are used.

Early design model

- These are rated on a seven points scale.
- The cost drivers include product reliability and complexity, the extent of reuse, platform sophistication, personnel experience, CASE support, and schedule.
- The effort is calculated using the following formula:
- $\textbf{\textit{Effort} = (KSLOC)}^{\textit{scaling factor}} \textbf{\textit{* Effort Adjustment Factors * 2.45}}$
- *Example*

Early design model: Scale factor

- Precedentness : It reflects the experience on similar projects previously.
  - This is applicable to individuals as well as organizations both in terms of expertise and experience.
  - High value would imply that organization is quite familiar with application formula and very low value means no previous experience or expertise.
- Development Flexibility : It reflects degree of flexibility in development process.
  - Low value would imply well defined process being used.
  - Very high value would imply that the client offers very general idea of the product or project.
- Architecture Risk and Resolution : It represents degree of risk analysis being carried out during course of project.
  - Low value would indicate little analysis and very high value would represent complete and thorough risk analysis.

Early design model: Scale factor

- Team Cohesion : Reflects the team management skills of the employees developing the project.
  - Very low value would imply very low interaction and hardly any relationship among the members however high value would imply great relationship and good team work.
- Process maturity : Reflects process maturity of organization.
  - Very low value would imply organization has no level at all and high value would imply that organization is rated as highest level of the SEI-CMM.

Early design model: Cost Driver

Product attributes			
RELY	Required system reliability	DATA	Size of database used
CPLX	Complexity of system modules	RUSE	Required percentage of reusable components
DOCU	Extent of documentation required		
Computer attributes			
TIME	Execution time constraints	STOR	Memory constraints
PVOL	Volatility of development platform		
Personnel attributes			
ACAP	Capability of project analysts	PCAP	Programmer capability
PCON	Personnel continuity	AEXP	Analyst experience in project domain
PEXP	Programmer experience in project domain	LTEX	Language and tool experience
Project attributes			
TOOL	Use of software tools	SITE	Extent of multi-site working and quality of site communications
SCED	Development schedule compression		

Staffing Level Estimation

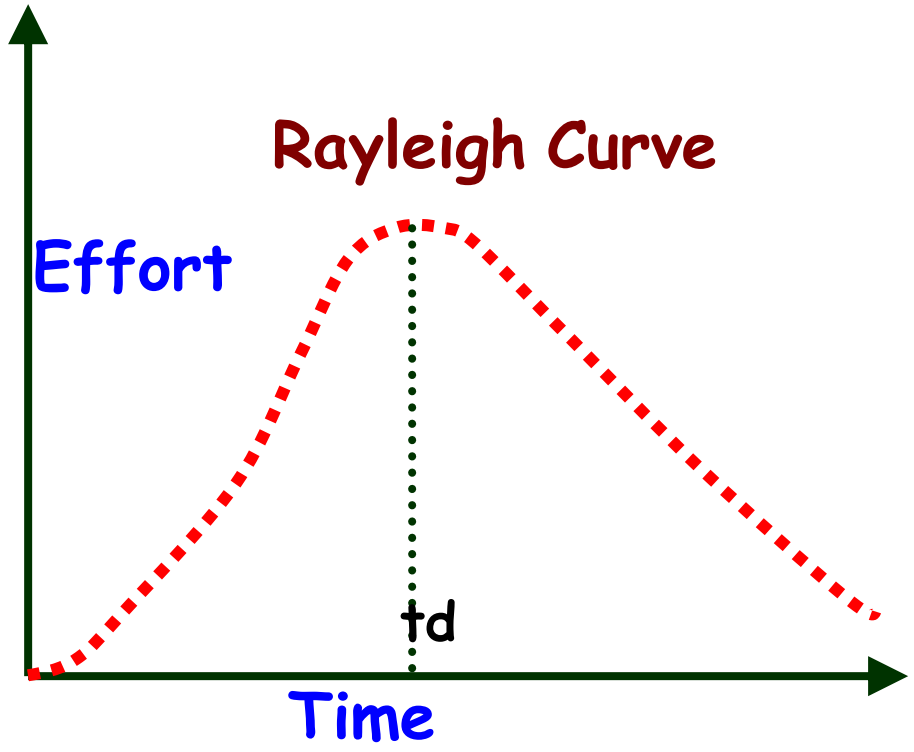
- Number of personnel required during any development project:
  - not constant.
- Norden in 1958 analyzed many R&D projects, and observed:
  - Rayleigh curve represents the number of full-time personnel required at any time.

Norden's work

- Norden concluded that the staffing pattern for any R&D project starting from a low level, increases until it reaches a peak value.
- It then starts to diminish.
- This pattern can be approximated by the Rayleigh distribution curve.

Rayleigh Curve

- Rayleigh curve is specified by two parameters:
  - td the time at which the curve reaches its maximum
  - K the total area under the curve.
- L=f(K, td)



Norden's work

- Norden represented the Rayleigh curve by the following equation

$$E = \frac{K}{{t_d}^2} * t * e^{\frac{-t^2}{2t{d^2}}}$$

- where, *E* is the effort required at time *t*.
- *E* is an indication of the number of developers at any particular time during the duration of the project,
- *K* is the area under the curve, and *td* is the time at which the curve attains its maximum value.

Norden's work

- Results of Norden are applicable to general R&D projects and,
- were not meant to model the staffing pattern of software development projects.

Putnam's Work:

- In 1976, Putnam studied the problem of staffing of software projects:
  - observed that the level of effort required in software development efforts has a similar envelope.
  - found that the Rayleigh-Norden curve
    - relates the number of delivered lines of code to effort and development time.

Putnam's Work

- Putnam analyzed a large number of army projects, and derived the expression:

$$L = C_k K^{1/3} t_d^{4/3}$$

  - K is the effort expended and L is the size in KLOC.
  - td is the time to develop the software.
  - C_k is the state of technology constant reflects factors that affect programmer productivity.

Putnam's Work

- C_k =2 for poor development environment
  - no methodology, poor documentation, and review, etc.
- C_k =8 for good software development environment
  - software engineering principles used
- C_k =11 for an excellent environment

Rayleigh Curve

- Very small number of engineers are needed at the beginning of a project
  - carry out planning and specification.
- As the project progresses:
  - more detailed work is required,
  - number of engineers slowly increases and reaches a peak.

Rayleigh Curve

- Putnam observed that:
  - the time at which the Rayleigh curve reaches its maximum value
    - corresponds to system testing and product release.
  - After system testing,
    - the number of project staff falls till product installation and delivery.
- From the Rayleigh curve observe that:
  - approximately 40% of the area under the Rayleigh curve is to the left of td
  - and 60% to the right.

Effect of Schedule Change on Cost

- Using the Putnam's expression for L,

$$K = L^3 / C_k^3 t_d^4$$

$$\text{Or, } K = C_1 / t_d^4$$

- For the same product size,
- $C_1 = L^3 / C_k^3$ is a constant.

Effect of Schedule Change on Cost

- From this expression, it can easily be observed that when the schedule of a project is compressed, the required effort increases in proportion to the fourth power of the degree of compression.
- It means that a relatively small compression in delivery schedule can result in substantial penalty on human effort.
- For example, if the estimated development time using COCOMO formulas is 1 year, then in order to develop the product in 6 months, the total effort required (and hence the project cost) increases 16 times.

Effect of Schedule Change on Cost

- **Example:** The nominal effort and duration of a project have been estimated to be 1000PM and 15 months. The project cost has been negotiated to be Rs. 200,000,000. The needs the product to be developed and delivered in 12 month time. What should be the new cost to be negotiated?
- **Answer:** The project can be classified as a large project. Therefore, the new cost to be negotiated can be given by the Putnam's formula:
- new cost = Rs. 200, 000, 000 × $(15/12)^4$ = Rs. 488,281,250.

Effect of Schedule Change on Cost

- Observe:
  - a relatively small compression in delivery schedule
    - can result in substantial penalty on human effort.
- Also, observe:
  - benefits can be gained by using fewer people over a somewhat longer time span.

Example

- If the estimated development time is 1 year, then in order to develop the product in 6 months,
  - the total effort and hence the cost increases 16 times.
  - In other words,
    - the relationship between effort and the chronological delivery time is highly nonlinear.

Effect of Schedule Change on Cost

- Putnam model indicates extreme penalty for schedule compression
  - and extreme reward for expanding the schedule.
- Putnam estimation model works reasonably well for very large systems,
  - but seriously overestimates the effort for medium and small systems.

Effect of Schedule Change on Cost

- Boehm observed:
  - "There is a limit beyond which the schedule of a software project cannot be reduced by buying any more personnel or equipment."
  - This limit occurs roughly at 75% of the nominal time estimate.

Effect of Schedule Change on Cost

- If a project manager accepts a customer demand to compress the development time by more than 25%
  - very unlikely to succeed.
    - every project has only a limited amount of parallel activities
    - sequential activities cannot be speeded up by hiring any number of additional engineers.
    - many engineers have to sit idle.

Jensen Model

- Jensen model is very similar to Putnam model.
  - attempts to soften the effect of schedule compression on effort
  - makes it applicable to smaller and medium sized projects.

Jensen Model

- Jensen proposed the equation:
  - $L=C_{te}t_dK^{1/2}$
  - Where,
    - C_{te} is the effective technology constant,
    - t_d is the time to develop the software, and
    - K is the effort needed to develop the software.

SCHEDULING

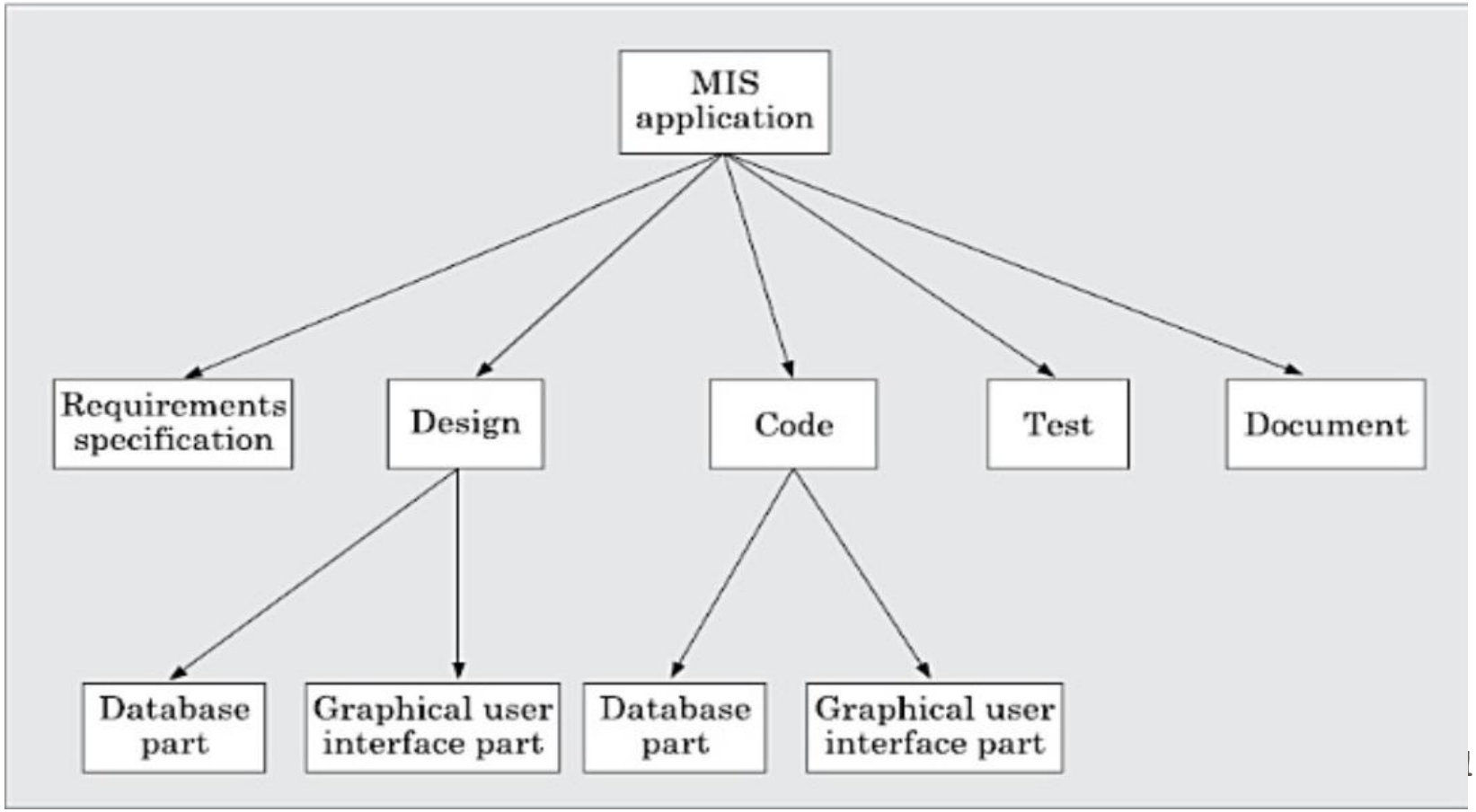
- The scheduling problem, in essence, consists of deciding which tasks would be taken up when and by whom.
- In order to schedule the project activities, a software project manager needs to do the following:
  1. Identify all the major activities that need to be carried out to complete the project.
  2. Break down each activity into tasks.
  3. Determine the dependency among different tasks.

SCHEDULING

- 4. Establish the estimates for the time durations necessary to complete the tasks.
- 5. Represent the information in the form of an activity network.
- 6. Determine task starting and ending dates from the information represented in the activity network.
- 7. Determine the critical path. A critical path is a chain of tasks that determines the duration of the project.
- 8. Allocate resources to tasks.

Work Breakdown Structure

- Work breakdown structure (WBS) is used to recursively decompose a given set of activities into smaller activities. Example



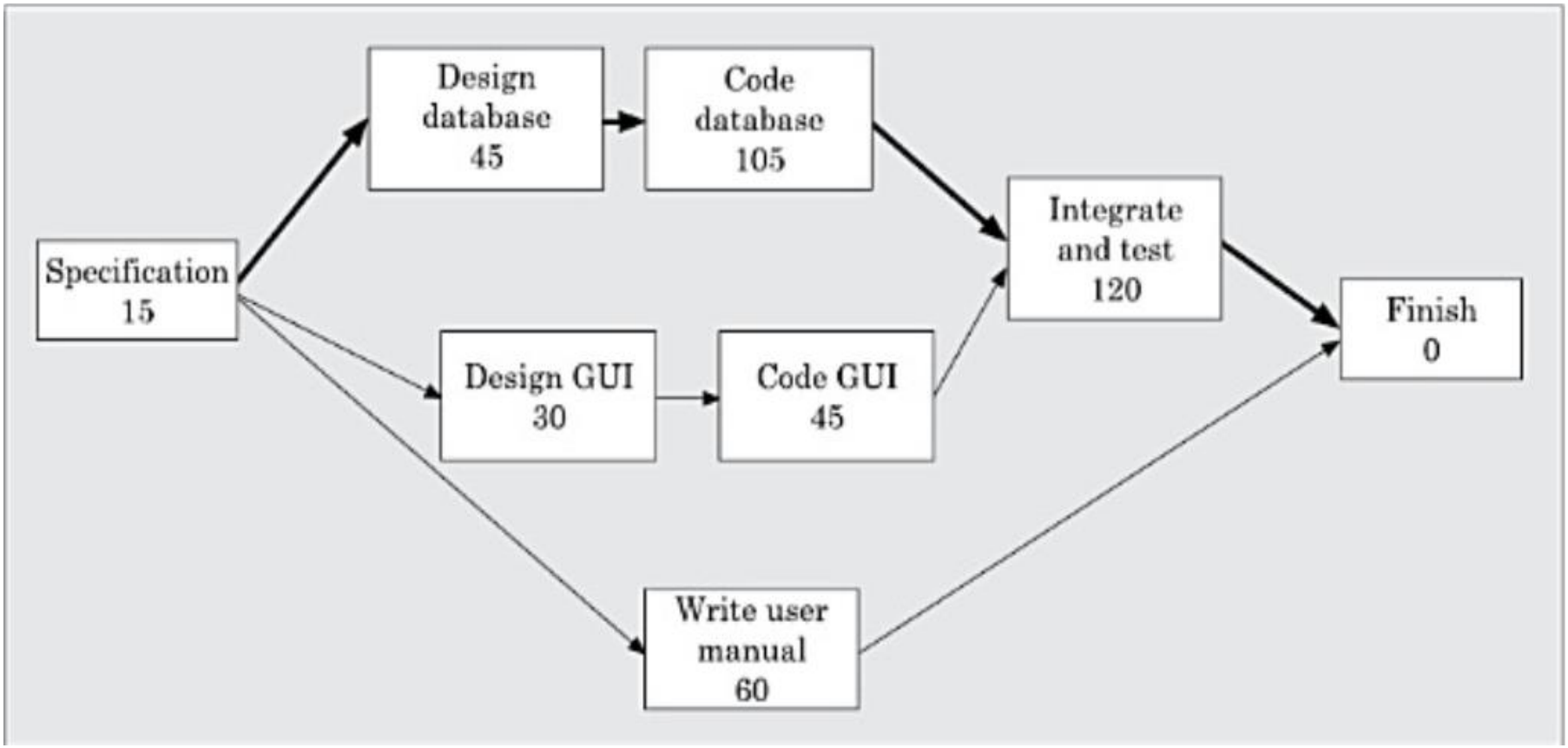
How long to decompose?

- The decomposition of the activities is carried out until any of the following is satisfied:
- A leaf-level sub-activity (a task) requires approximately two weeks to develop.
- Hidden complexities are exposed, so that the job to be done is understood and can be assigned as a unit of work to one of the developers.
- Opportunities for reuse of existing software components is identified.

Activity Networks

- An activity network shows the different activities making up a project, their estimated durations, and their interdependencies.
- **Activity on Node (AoN):** In this representation, each activity is represented by a rectangular (some use circular) node and the duration of the activity is shown alongside each task in the node.
- The inter-task dependencies are shown using directional edges

Activity Networks



Activity Networks

- Example: Determine the Activity network representation for the MIS development project. Assume that the manager has determined the tasks to be represented from the work breakdown structure, and has determined the durations and dependencies for each task.

Activity Networks

- The activity network representation is

Task Number	Task	Duration	Dependent on Tasks
T1	Specification	15	–
T2	Design database	45	T 1
T3	Design GUI	30	T 1
T4	Code database	105	T 2
T5	Code GUI part	45	T 3
T6	Integrate and test	120	T 4 and T 5
T7	Write user manual	60	T 1

Critical Path Method (CPM)

- A **path** in the activity network graph is any set of consecutive nodes and edges in this graph from the starting node to the last node.

- A **critical path** consists of a set of dependent tasks that need to be performed in a sequence and which together take the longest time to complete.

Critical Path Method (CPM)

- A critical path in project management is certain tasks that need to be performed in a clear order and for a certain period.
- If part of one task can be slowed down or postponed for a term without leaving work on others, then such a task is not critical.
- While tasks with a critical value cannot be delayed during the implementation of the project and are limited in time.
- Critical Path Method (CPM) is an algorithm for planning, managing and analyzing the timing of a project.

Critical Path Method (CPM)

- The step-by-step CPM system helps to identify critical and non-critical tasks from projects' start to completion and prevents temporary risks.
- Critical tasks have a zero run-time reserve. If the duration of these tasks changes, the terms of the entire project will be "shifted."
- That is why critical tasks in project management require special control and timely detection of risks.

Critical Path Method (CPM)

Benefits of Critical Path Analysis

1. The method visualizes projects in a clear graphical form.
2. It defines the most important tasks.
3. Saves time and helps in the management of deadlines.
4. Helps to compare the planned with the real status.
5. Identifies all critical activities that need attention.
6. Makes dependencies clear and transparent.

CPM loses its usefulness in more chaotic projects.

Critical Path Method (CPM)

- CPM is an algorithmic approach to determine the critical paths and slack times for tasks not on the critical paths involves calculating the following quantities:
- **Minimum time (MT):** It is the minimum time required to complete the project. It is computed by determining the maximum of all paths from start to finish.
- **Earliest start (ES):** It is the time of a task, maximum of all paths from the start to this task. The ES for a task is the ES of the previous task plus the duration of the preceding task.

Critical Path Method (CPM)

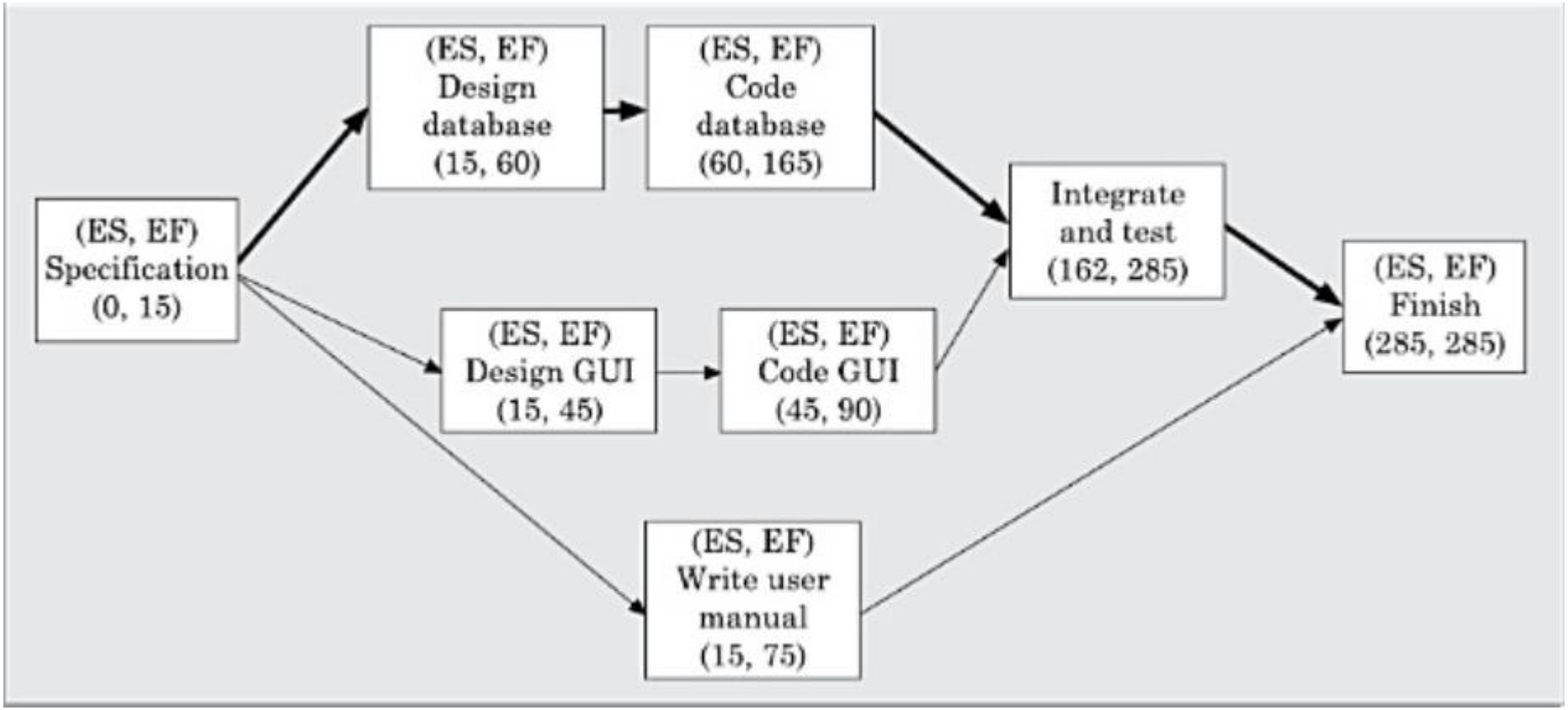
- **Latest start time (LST):** It is the difference between MT and the maximum of all paths from this task to the finish. The LST can be computed by subtracting the duration of the subsequent task from the LST of the subsequent task.
- **Earliest finish time (EF):** The EF for a task is the sum of the earliest start time of the task and the duration of the task.
- **Latest finish (LF):** LF indicates the latest time by which a task can finish without affecting the final completion time of the project. LF of a task can be obtained by subtracting maximum of all paths from this task to finish from MT.

Critical Path Method (CPM)

- **Slack time (ST):** The slack time (or float time) is the total time that a task may be delayed before it will affect the end time of the project. The slack time indicates the "flexibility" in starting and completion of tasks.
- ST for a task is **LS-ES** and can equivalently be written as **LF-EF**.

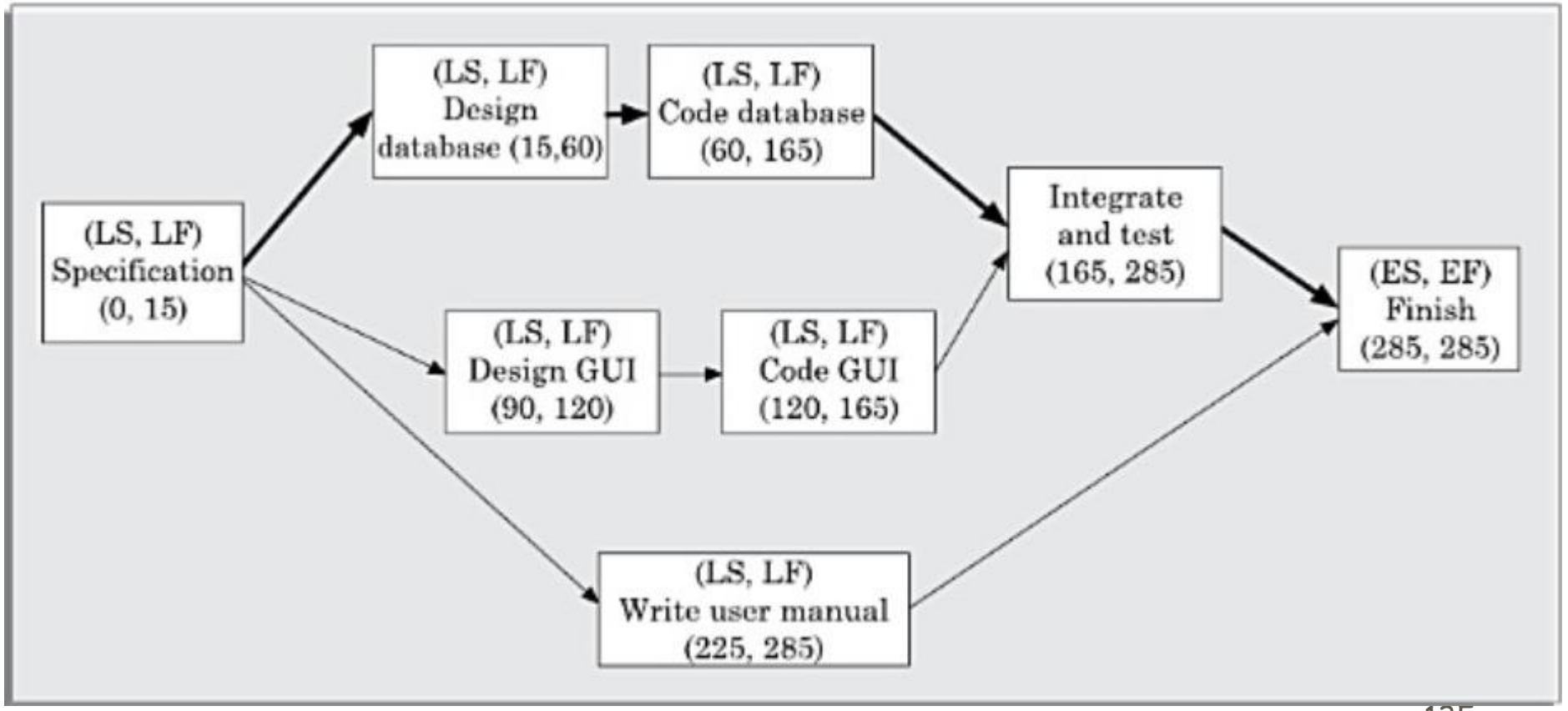
Critical Path Method (CPM)

Use the Activity network to determine the ES and EF for every task for the MIS problem



Critical Path Method (CPM)

Use the Activity network to determine the LS and LF for every task for the MIS problem



Critical Path Method (CPM)

The project parameters for different tasks for the MIS problem can be computed as follows:

1. Compute ES and EF for each task. Use the rule: ES is equal to the largest EF the immediate predecessors

2. Compute LS and LF for each task. Use the rule: LF is equal to the smallest LS of the immediate successors

3. Compute ST for each task. Use the rule: ST=LF-EF

Critical Path Method (CPM)

Table 3.8: Project Parameters Computed From Activity Network

Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design data base	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code data base	60	165	60	165	0
Code GUI part	45	90	120	165	75
Integrate and test	165	285	165	285	0
Write user manual	15	75	225	285	210

PERT (Project evaluation and review technique) Charts

- The activity durations computed using an activity network are only estimated duration.
- It is therefore not possible to estimate the worst case (pessimistic) and best case (optimistic) estimations using an activity diagram.
- Since, the actual durations might vary from the estimated durations, the utility of the activity network diagrams are limited.
- The CPM can be used to determine the duration of a project, but does not provide any indication of the probability of meeting that schedule.

PERT Charts

- **Project evaluation and review technique (PERT)** charts are a more sophisticated form of activity chart.
- Project managers know that there is considerable uncertainty about how much time a task would exactly take to complete.
- PERT charts can be used to determine the probabilistic times for reaching various project milestones, including the final mile stone.
- PERT charts like activity networks consist of a network of boxes and arrows.

PERT Charts

- The boxes represent activities and the arrows represent task dependencies.
- A PERT chart represents the statistical variations in the project estimates assuming these to be normal distribution.
- PERT allows for some randomness in task completion times,
- and therefore provides the capability to determine the probability for achieving project milestones based on the probability of completing each task along the path to that milestone.

PERT Charts

- Each task is annotated with three estimates:
- **Optimistic (O):** The best possible case task completion time.
- **Most likely estimate (M):** Most likely task completion time.
- **Worst case (W):** The worst possible case task completion time.
- The optimistic (O) and worst case (W) estimates represent the extremities of all possible scenarios of task completion.
- The most likely estimate (M) is the completion time that has the highest probability. The three estimates are used to compute the expected value of the standard deviation.

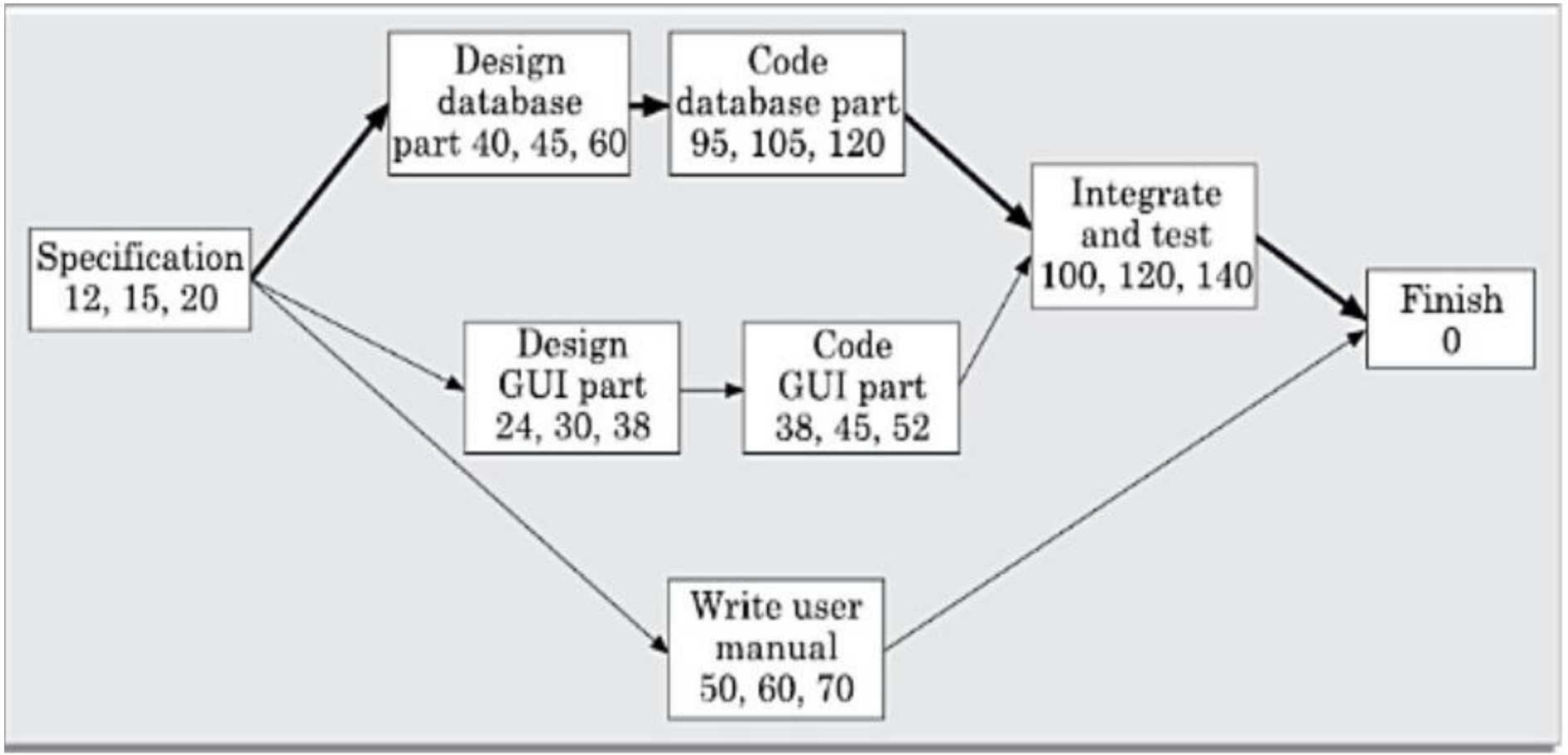
PERT Charts

- The mean estimated time is calculated as

$$ET = (O + 4M + W)/6$$

- Since all possible completion times between the minimum and maximum duration for every task has to be considered, there can be many critical paths,
- depending on the various permutations of the estimates for each task.
- This makes critical path analysis in PERT charts very complex. A critical path in a PERT chart is shown by using thicker arrows.

PERT Charts


Design
database
part 40, 45, 60
Code
database part
95, 105, 120
Integrate
and test
100, 120, 140
Specification
12, 15, 20
Finish
0
Design
GUI part
24, 30, 38
Code
GUI part
38, 45, 52
Write user
manual
50, 60, 70

RISK MANAGEMENT

- A risk is any anticipated unfavorable event or circumstance that can occur while a project is underway.
- Risk management aims at reducing the chances of a risk becoming real as well as reducing the impact of a risks that becomes real.
- Risk management consists of three essential activities—risk identification, risk assessment, and risk mitigation.

Risk Identification

- The project manager needs to anticipate the risks in a project as early as possible.
- There are three main categories of risks which can affect a software project:
  - Project risks,
  - Technical risks,
  - and business risks.

Project Risks

- It concerns various forms of budgetary, schedule, personnel, resource, and customer-related problems.
- An important project risk is schedule slippage. Since, software is intangible, it is very difficult to monitor and control a software project.
- The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule slippage.

Technical and Business Risks

- Technical risks concern potential design, implementation, interfacing, testing, and maintenance problems.
  - Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence.
  - Most technical risks occur due the development team's insufficient knowledge about the product.
- Business risks: This type of risks includes the risk of building an excellent product that no one wants, losing budgetary commitments, etc.

Example

- Let us consider a satellite based mobile communication product:
- What if the project cost escalates and overshoots what was estimated?: **Project risk.**
- What if the mobile phones that are developed become too bulky in size to conveniently carry?: **Business risk.**
- What if it is later found out that the level of radiation coming from the phones is harmful to human being?: **Business risk.**
- What if call hand-off between satellites becomes too difficult to implement?: **Technical risk.**

Risk Assessment

- The objective of risk assessment is to rank the risks in terms of their damage causing potential.
- For risk assessment, first each risk should be rated in two ways:
- The likelihood of a risk becoming real (r).
- The consequence of the problems associated with that risk (s)
- Based on these two factors, the priority of each risk can be computed.
- If all identified risks are prioritized, then the most likely and damaging risks can be handled.

Risk Mitigation

- **Avoid the risk:** Risks can be avoided in several ways. Risks often arise due to project constraints and can be avoided by suitably modifying the constraints.
- **Process-related risk:** These risks arise due to aggressive work schedule, budget, and resource utilization.
- **Product-related risks:** These risks arise due to commitment to challenging product features (e.g. response time of one second, etc.), quality, reliability etc.
- **Technology-related risks:** These risks arise due to commitment to use certain technology.

Risk Mitigation

- **Transfer the risk:** This strategy involves getting the risky components developed by a third party, buying insurance cover, etc.
- **Risk reduction:** This involves planning ways to contain the damage due to a risk.

- *For example, if there is risk that some key personnel might leave, new recruitment may be planned.*
- *The most important risk reduction techniques for technical risks is to build a prototype that tries out the technology that you are trying to use. For example, if you are using a compiler for recognizing user commands, you would have to construct a compiler for a small and very primitive command language first.*

Organization Structure

- Functional Organization:
  - Engineers are organized into functional groups, e.g.
    - specification, design, coding, testing, maintenance, etc.
  - Engineers from functional groups get assigned to different projects

Advantages of Functional Organization

- Specialization
- Ease of staffing
- Good documentation is produced
  - different phases are carried out by different teams of engineers.
- Helps identify errors earlier.

Project Organization

- Engineers get assigned to a project for the entire duration of the project
  - Same set of engineers carry out all the phases
- Advantages:
  - Engineers save time on learning details of every project.
  - Leads to job rotation

Team Structure

- Problems of different complexities and sizes require different team structures:
  - Chief-programmer team
  - Democratic team
  - Mixed organization

Democratic Teams

- Suitable for:
  - small projects requiring less than five or six engineers
  - research-oriented projects
- A manager provides administrative leadership:
  - at different times different members of the group provide technical leadership.

Democratic Teams

- Democratic organization provides
  - higher morale and job satisfaction to the engineers
  - therefore leads to less employee turnover.
- Suitable for less understood problems,
  - a group of engineers can invent better solutions than a single individual.

Democratic Teams

- Disadvantage:
  - team members may waste a lot time arguing about trivial points:
    - absence of any authority in the team.

Chief Programmer Team

- A senior engineer provides technical leadership:
  - partitions the task among the team members.
  - verifies and integrates the products developed by the members.

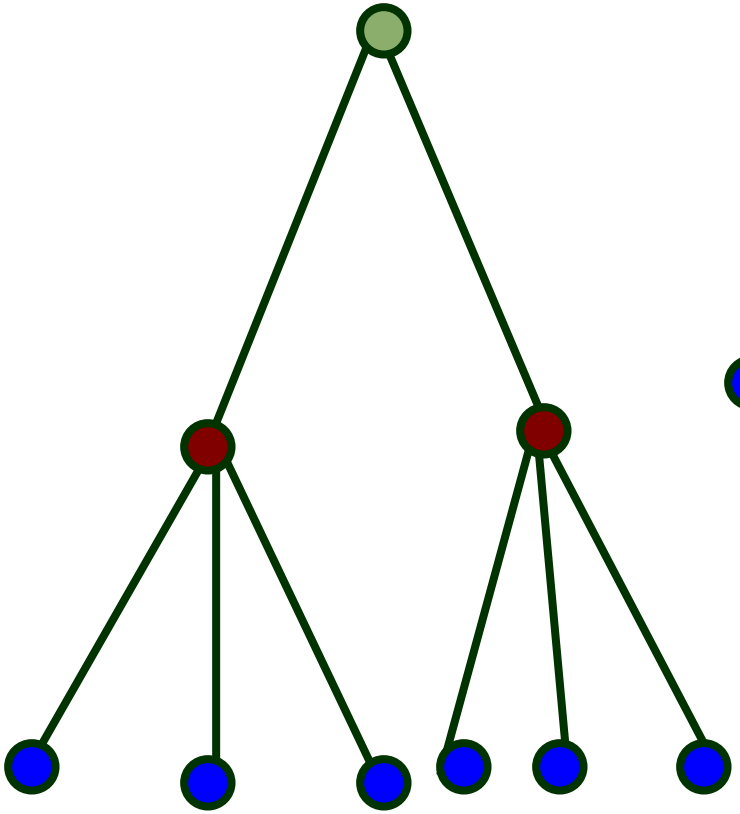
Chief Programmer Team

- Works well when
  - the task is well understood
    - also within the intellectual grasp of a single individual,
  - importance of early completion outweighs other factors
    - team morale, personal development, etc.
- Chief programmer team is subject to single point failure:
  - too much responsibility and authority is assigned to the chief programmer.

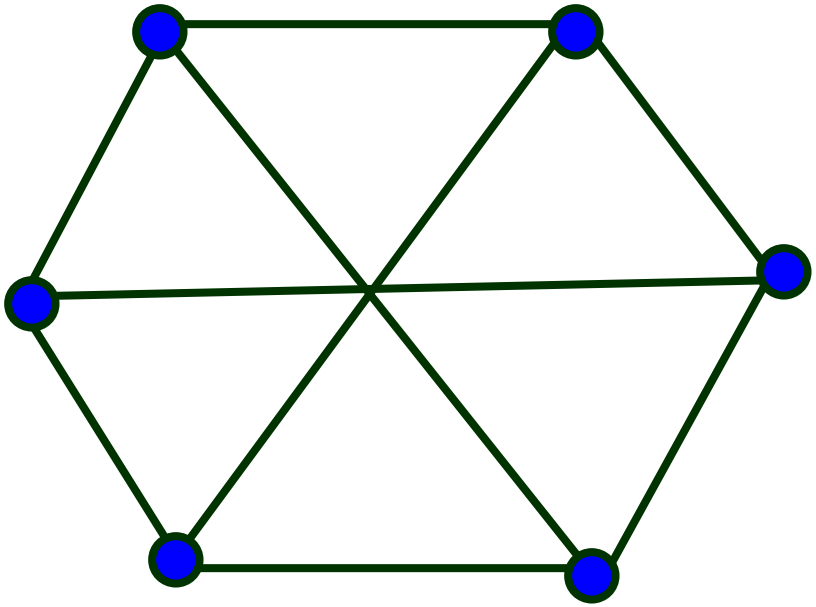
Mixed Control Team Organization

- Draws upon ideas from both:
  - democratic organization and
  - chief-programmer team organization.
- Communication is limited
  - to a small group that is most likely to benefit from it.
- Suitable for large organizations.

Team Organization

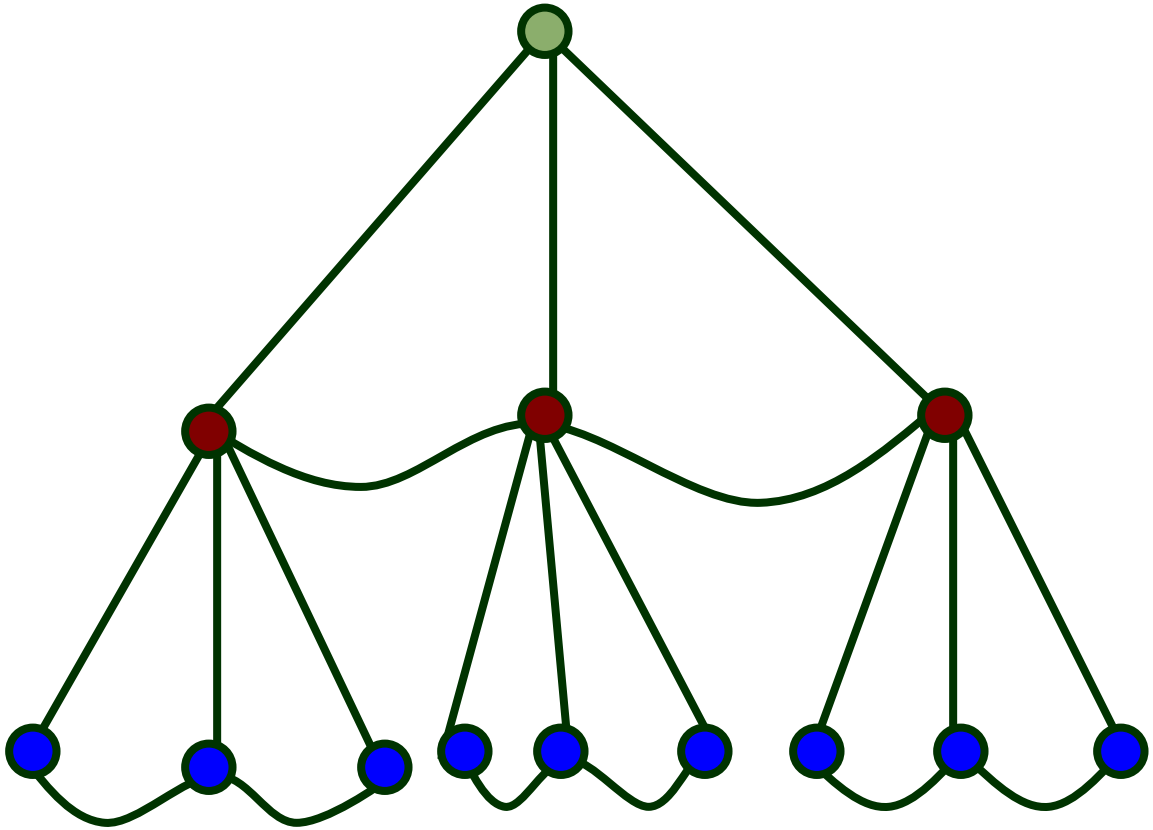


Chief Programmer team



Democratic Team

Mixed team organization



Object Point Metric

- Object points are used as an initial measure for size way early in the development cycle, during feasibility studies.
- An initial size measure is determined by counting the number of ***screens***, ***reports***, and third-generation ***components*** that will be used in the application.
- Each object is classified as ***simple***, ***medium***, or ***difficult***.

Object Point Metric

- **Object point complexity levels for screens.**

Number of views contained	Number and source of data tables		
	Total <4 <2 servers <2 clients	Total <8 2-3 servers 3-5 clients	Total 8+ >3 servers >5 clients
< 3	Simple	Simple	Medium
3 - 7	Simple	Medium	Difficult
8+	Medium	Difficult	Difficult

Object Point Metric

- **Object point complexity levels for reports.**

Number of views contained	Number and source of data tables		
	Total <4 <2 servers <2 clients	Total <8 2-3 servers 3-5 clients	Total 8+ >3 servers >5 clients
0 - 1	Simple	Simple	Medium
2 - 3	Simple	Medium	Difficult
4+	Medium	Difficult	Difficult

Object Point Metric

- The number in each cell is then weighted and summed to get the object point.
- The weights represent the relative effort required to implement an instance of that complexity level.
- **Complexity level for object point:**

Object Type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	-	-	10

$$\textit{New object points} = (\textit{object points}) \times (100 - r) / 100$$

Assuming that $\%\, r$ of the objects will be reused from previous projects.

How to calculate OP

1. Estimate the number of "screens" of the system and the approximate number of data entries for each screen (called view).
2. Estimate the number of "reports" that will be generated by the system (including any output file, writing to database, etc.) and the approximate number of data entries for each report (called view).
3. For each "screen" and "report" and their corresponding views, use Tables (previous pages) to determine whether that screen or report is "simple", "medium" or "difficult".
4. Weight the screens and reports using the weight table and sum them up to get the OP number.
5. If you have any acquired component, give it the weight of 10 and add it to the OP.
6. If you have any part of your system reused from a previous generation, define a percentage of reuse and then adjust the value of OP accordingly.

How to calculate OP

- Formula:

  Effort [Person-Month] = OP / PROD

- How to measure PROD?

Developer Experience/Skills	Very Low	Low	Average (Nominal)	High	Very High
PROD	4	7	13	25	50

- Example:

  Effort [Person-Month] = 13 / 13 = 1

  Assuming average experience and 0% reuse.

How to calculate OP

- Suppose that you have
  - 4 screens, 2 of them simple (weight 1) and 2 of them medium (weight 2)
  - 3 reports, 2 of them simple (weight 2) and 1 medium (weight 5),

  then the total number of OP is:

$$OP = 2\times1 + 1\times2 + 2\times2 + 1\times5 = 13$$

- If you have any acquired component, give it the weight 10 and add it to the OP.
- If you have any part of your system reused from a previous generation, define a percentage of reuse (say 10%) and then adjust the value of OP accordingly.

$$OP_{new} = 13 \times (100\text{-}10)/100 = 11.7$$