Software Project Management

Organization of this Lecture:

- Introduction to Project Planning
- Software Cost Estimation
  - Cost Estimation Models
  - Software Size Metrics
  - Empirical Estimation
  - Heuristic Estimation
  - COCOMO
- Staffing Level Estimation
- Effect of Schedule Compression on Cost
- Summary

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 to work efficiently towards successful completion of a software project.

Responsibility of project managers

- Project proposal writing,
- Project cost estimation,
- Scheduling,
- Project staffing,
- Project monitoring and control,
- Software configuration management,
- Risk management,
- Managerial report writing and presentations, etc.

Project Planning Activities

- Estimating the following attributes of the project:
  - **Project size:** What will be problem complexity in terms of the effort and time required to develop the product?
  - **Cost**
  - **Duration**
  - **Effort**

- The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.
  - **Scheduling** manpower and other resources
  - **Staff organization** and staffing plans
  - **Risk identification**, analysis
  - **Miscellaneous plans** such as quality assurance plan, configuration, management plan, etc.

Sliding Window Planning

- **Planning a project over a number of stages** protects managers from making big commitments too early. This technique of staggered planning is known as **Sliding Window Planning.**

- In the sliding window technique, starting with an initial plan, the project is **planned more accurately in successive development stages.**

Organization of SPMP Document

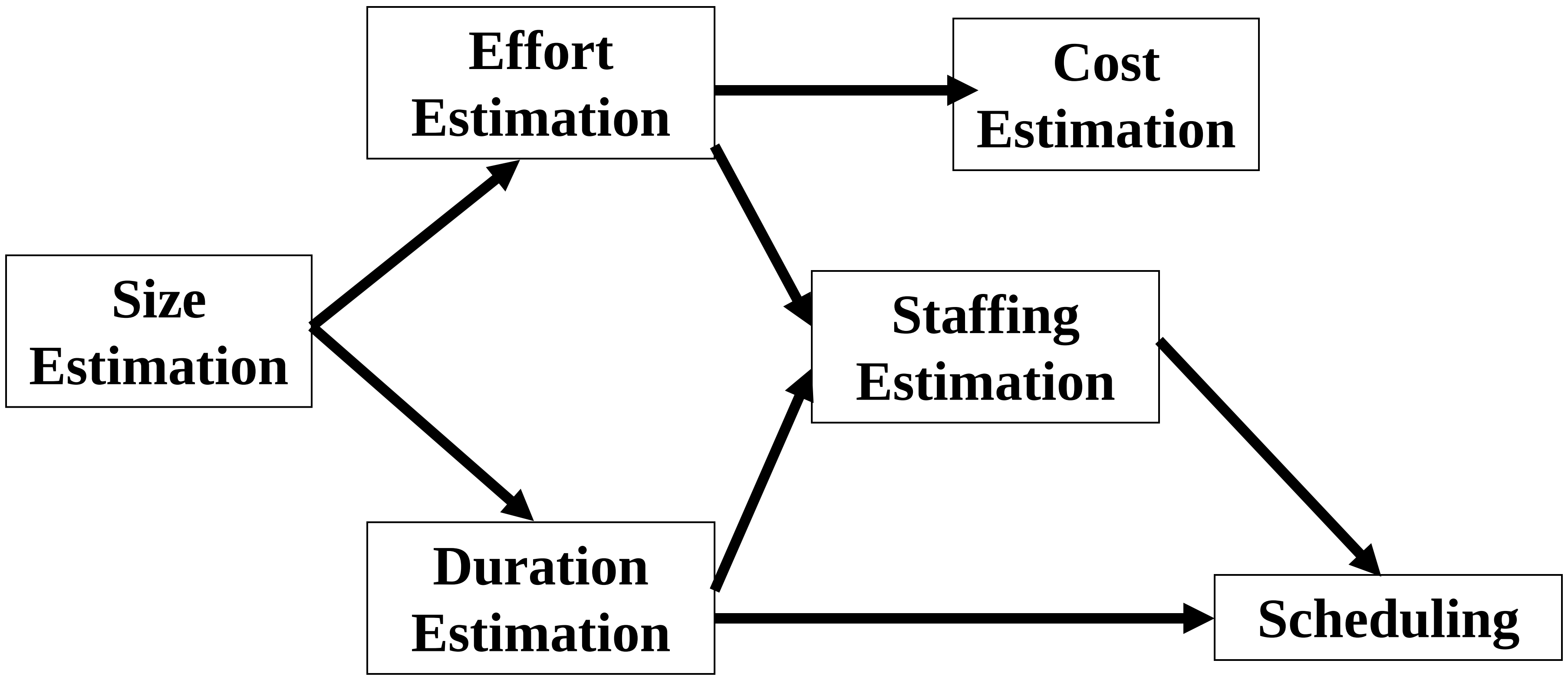
- After planning is complete:
  - Document the plans in a **Software Project Management Plan(SPMP) document.**
    - Introduction (Objectives, Major Functions, ...)
    - Project Estimates (Historical Data, Estimation Techniques, Effort, Cost, ...)
    - Project Resources Plan (People, Hardware and Software, ...)
    - Schedules (Work Breakdown Structure, Gantt Chart Represent...)
    - Risk Management Plan (Risk Analysis, Risk Identification, ...)
    - Project Tracking and Control Plan
    - Miscellaneous Plans(Process Tailoring, Quality Assurance)

Software Project Size Estimation

- The project size is a measure of the problem complexity in terms of the **effort and time** required to develop the product.

- Two metrics widely used to estimate size:
  - lines of code (LOC)
  - function point (FP).

Software Project Size Estimation



Software Size Metrics

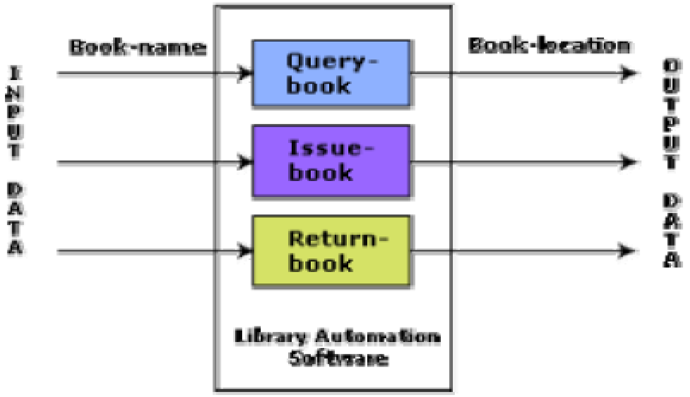
- **LOC (Lines of Code):**
  - Using this metric, the project size is estimated by **counting** the **number of source instructions** in the developed program.
  - While counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.
  - Accurate estimation of the LOC count at the beginning of a project is very difficult.
  - To estimate project managers usually divide the problem into modules, submodules until the sizes of the different leaf-level modules can be approximately predicted.

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.

Function Point Metric

- Estimate the size of a software product directly from the problem specification.
- Size of a software product is directly dependent on the number of different functions or features it supports.
- For example, the **issue book feature** of a Library Automation Software takes the **name of the book** as **input** and **displays** its **location and the number of copies available.**
- In addition to the number of basic functions that a software performs, **the size is also dependent on the number of files and the number of interfaces.**



Function Point Metric

- Proposed by Albrecht in early 80's:
  - **Count Total =4 #inputs + 5 #Outputs + 4 #inquiries + 10 #files + 7 #interfaces**
- **Number of inputs (EI):** Each data item input by the user is counted.
  - Example: employee pay roll software; the data items **name**, **age**, **address**, **phone number**, etc. are together considered as a **single input**.
- **Number of outputs (EO) :** The outputs considered refer to **reports printed, screen outputs, error messages etc.**
- **Number of inquiries (EQ) :** user commands which require specific action by the system. Ex: **print account balance, print student grades etc.**
- **Number of files (ILF) :** Each logical file is counted. Ex. File for storing **customer details**, file for **daily purchase record** in a supermarket.
- **Number of interfaces (EIF) :** Used to exchange information with other systems. Examples of such interfaces are data files on disks, communication links with other systems etc.

1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types as shown in Table 2.1:

Table 2.1 Types of FP Attributes

Measurement Parameter	*Examples*
1. Number of external inputs (EI)	Input screen and tables.
2. Number of external outputs (EO)	Output screens and reports.
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories.
5. Number of external interfaces (EIF)	Shared databases and shared routines.

All these parameters are then individually assessed for complexity.

2. FP characterizes the complexity of the software system and hence can be used to depict the project time and the manpower requirement.
3. The effort required to develop the project depends on what the software does.
4. FP is programming language independent.
5. FP method is used for data processing systems, business systems like information systems.
6. The 5 parameters mentioned above are also known as information domain characteristics.
7. All the above-mentioned parameters are assigned some weights that have been experimentally determined and are shown in Table 2.2.

Measurement Parameter	*Count*		*Weighing factor*			
			Simple	*Average*	*Complex*	
1. Number of external inputs (EI)	—	*	3	4	6 =	—
2. Number of external outputs (EO)	—	*	4	5	7 =	—
3. Number of external inquiries (EQ)	—	*	3	4	6 =	—
4. Number of internal files (ILF)	—	*	7	10	15 =	—
5. Number of external interfaces (EIF)	—	*	5	7	10 =	—
Count-total →						—

Note here that weighing factor will be simple, average or complex for a measurement parameter type.

CAF: Complexity Adjustment Factor

The Function Point (FP) is thus calculated with the following formula

$$\mathbf{FP = Count\text{-}total * [0.65 + 0.01 * \Sigma(F_i)]}$$
$$= Count\text{-}total * CAF$$

where Count-total is obtained from the above Table.

$$CAF = [0.65 + 0.01 * \Sigma(F_i)]$$

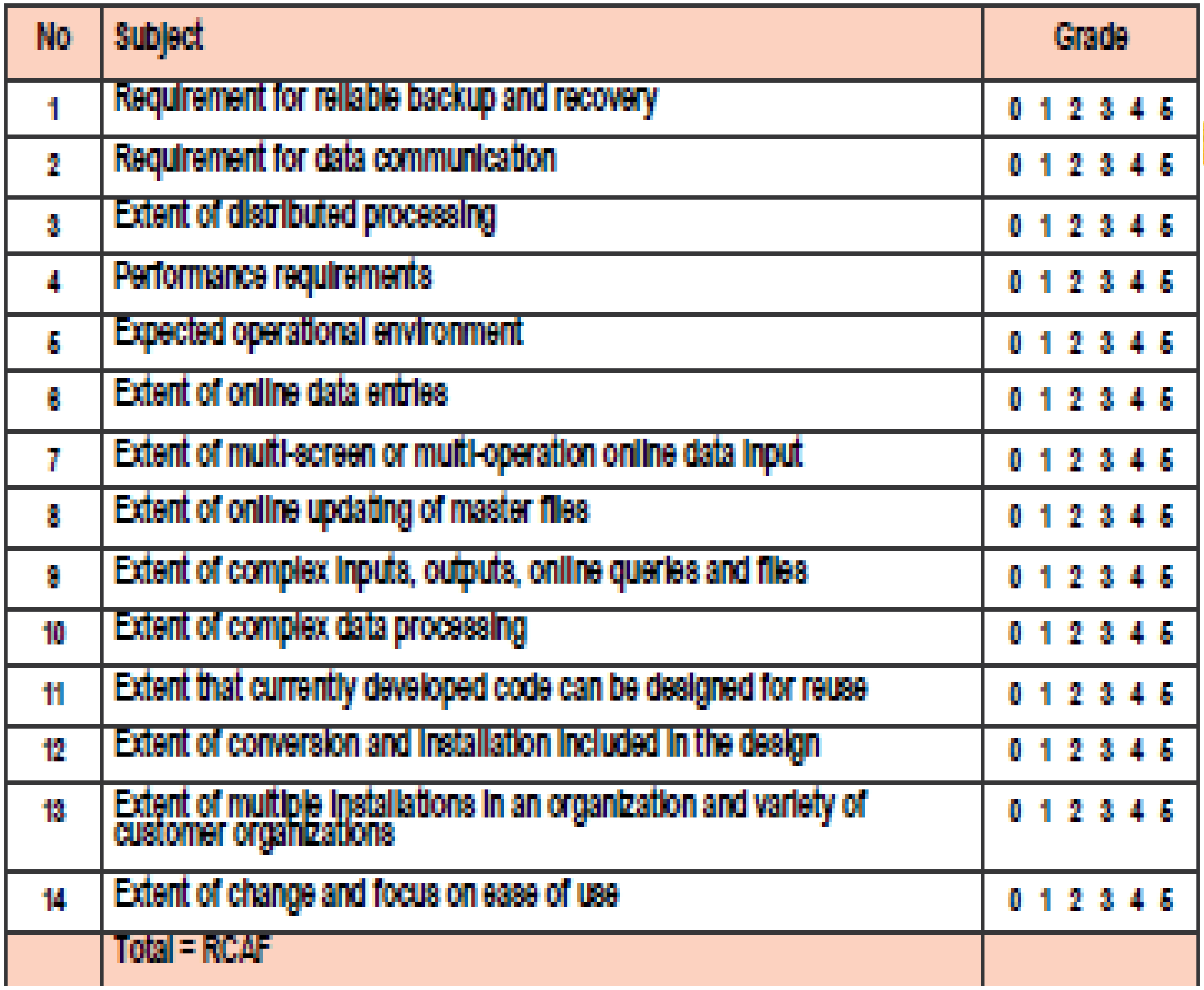
and $\Sigma(F_i)$ is the sum of all 14 questionnaires and show the complexity adjustment value/ factor-CAF (where i ranges from 1 to 14). Usually, a student is provided with the value of $\Sigma(F_i)$. **Also note that $\Sigma(F_i)$ ranges from 0 to 70, i.e.,**

$$\mathbf{0 <= \Sigma(F_i) <= 70}$$

and CAF ranges from 0.65 to 1.35 because

(a) When $\Sigma(F_i) = 0$ then CAF = 0.65
(b) When $\Sigma(F_i) = 70$ then CAF = 0.65 + (0.01 * 70) = 0.65 + 0.7 = 1.35

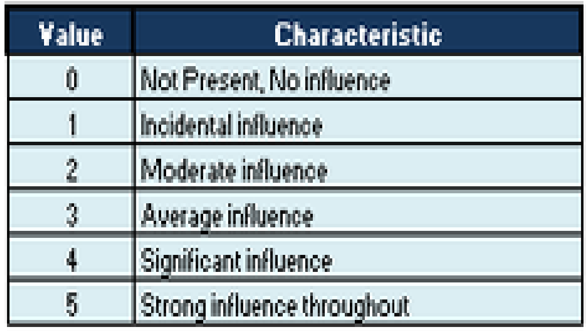
Relative complexity adjustment factor (RCAF) – form



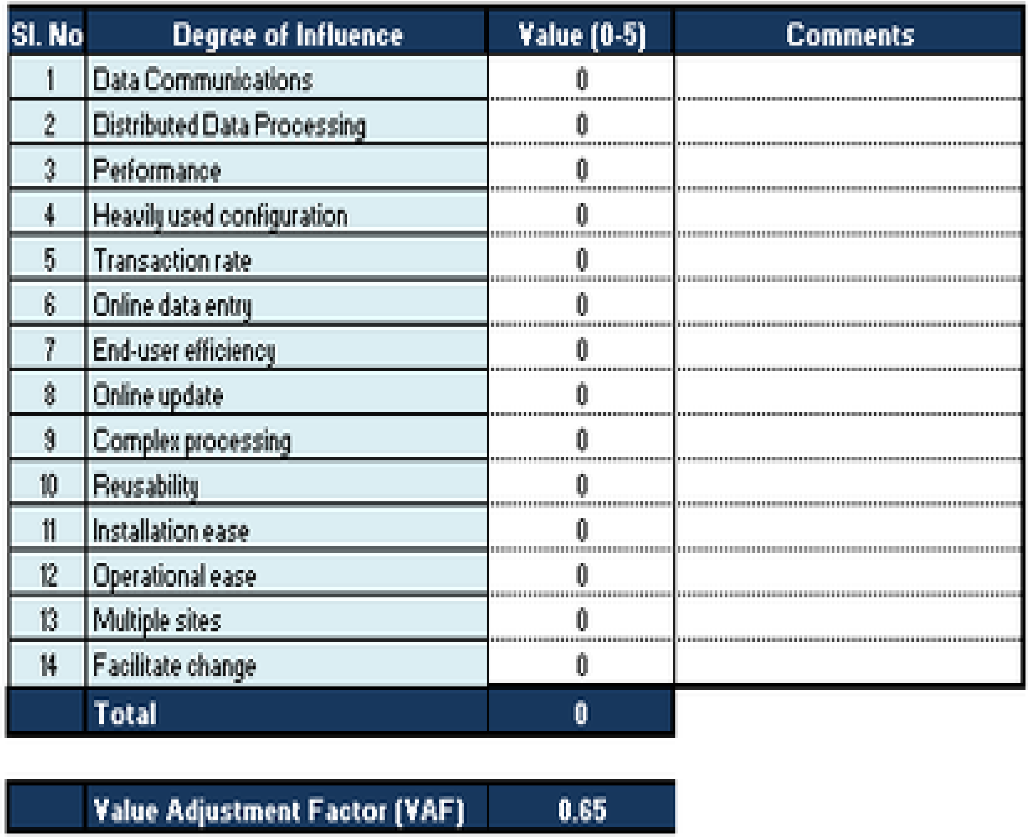
No	Subject	Grade
1	Requirement for reliable backup and recovery	0 1 2 3 4 5
2	Requirement for data communication	0 1 2 3 4 5
3	Extent of distributed processing	0 1 2 3 4 5
4	Performance requirements	0 1 2 3 4 5
5	Expected operational environment	0 1 2 3 4 5
6	Extent of online data entries	0 1 2 3 4 5
7	Extent of multi-screen or multi-operation online data input	0 1 2 3 4 5
8	Extent of online updating of master files	0 1 2 3 4 5
9	Extent of complex inputs, outputs, online queries and files	0 1 2 3 4 5
10	Extent of complex data processing	0 1 2 3 4 5
11	Extent that currently developed code can be designed for reuse	0 1 2 3 4 5
12	Extent of conversion and installation included in the design	0 1 2 3 4 5
13	Extent of multiple installations in an organization and variety of customer organizations	0 1 2 3 4 5
14	Extent of change and focus on ease of use	0 1 2 3 4 5
	Total = RCAF	

- 0 = No Influence
- 1 = Incidental
- 2 = Moderate
- 3 = Average
- 4 = Significant
- 5 = Essential

VALUE ADJUSTMENT FACTOR (VAF)



Value	Characteristic
0	Not Present, No influence
1	Incidental influence
2	Moderate influence
3	Average influence
4	Significant influence
5	Strong influence throughout



Sl. No	Degree of Influence	Value (0-5)	Comments
1	Data Communications	0	
2	Distributed Data Processing	0	
3	Performance	0	
4	Heavily used configuration	0	
5	Transaction rate	0	
6	Online data entry	0	
7	End-user efficiency	0	
8	Online update	0	
9	Complex processing	0	
10	Reusability	0	
11	Installation ease	0	
12	Operational ease	0	
13	Multiple sites	0	
14	Facilitate change	0	
	Total	0	

	Value Adjustment Factor (VAF)	0.65

Q. Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input = 50, User Output = 40, User Inquiries = 35, User Files = 6

External Interface = 4

Q. Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input = 50, User Output = 40, User Inquiries = 35, User Files = 6

External Interface = 4

Step-1: As complexity adjustment factor is average (given in question), hence, scale = 3. F = 14 * 3 = 42

Step-2:CAF = 0.65 + (0.01 * 42) = 1.07

Step-3: As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628

Step-4: Function Point = 628 * 1.07 = 671.96

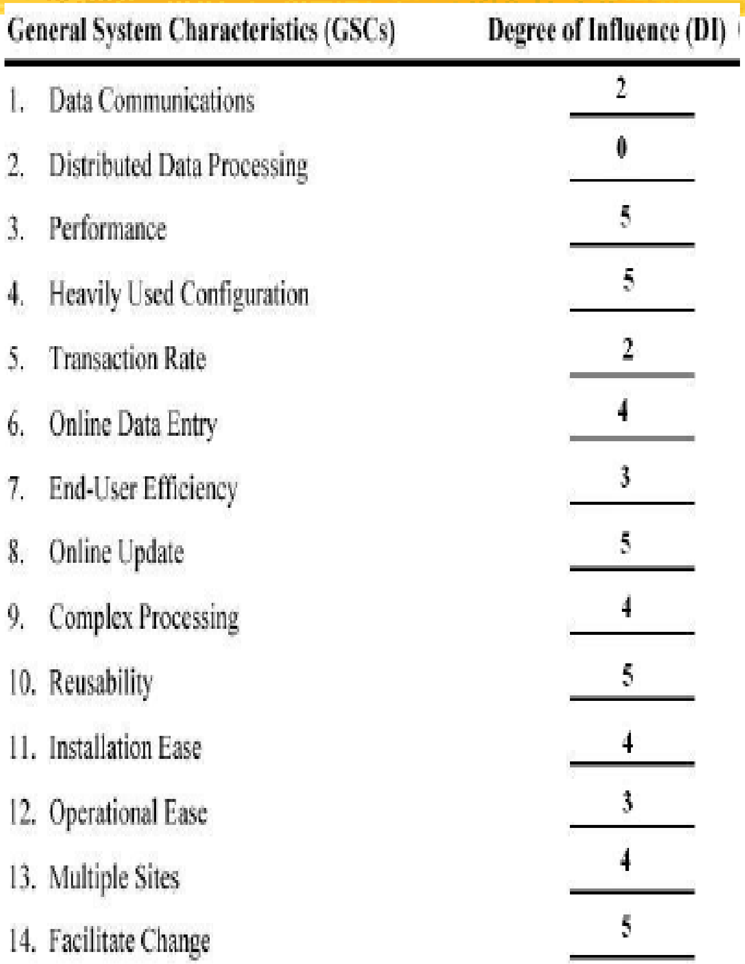
Example 4 Compute the function point value for a project with the following information domain characteristics:

(1) No. of user inputs = 24
(2) No. of user outputs = 65
(3) No. of user inquiries = 12
(4) No. of files = 12
(5) No. of external interfaces = 4

Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

Compute the fuction point value for an Complex condition.

Function Type	Estimated Count
EI	24
EO	16
EQ	22
ILF	4
ELF	2



General System Characteristics (GSCs)	Degree of Influence (DI)
1. Data Communications	2
2. Distributed Data Processing	0
3. Performance	5
4. Heavily Used Configuration	5
5. Transaction Rate	2
6. Online Data Entry	4
7. End-User Efficiency	3
8. Online Update	5
9. Complex Processing	4
10. Reusability	5
11. Installation Ease	4
12. Operational Ease	3
13. Multiple Sites	4
14. Facilitate Change	5

Function Point Metric (CONT.)

- 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:** Greater is the effort required to develop it and therefore its size should be larger compared to simpler functions.
- **Opponents claim:**
  - it is subjective --- Different people can come up with different estimates for the same problem.
- **Proponents claim:**
  - FP is language independent. Size can be easily derived from problem description

Software Project Size/Cost 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.

Empirical Size Estimation Techniques

- **Expert Judgement:** Estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate. **Suffers from individual bias.**
  - Experts divide a software product into component units:
    - e.g. GUI, database 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 unusual characteristic of the product which has influenced his estimation.
  - coordinator notes down any extraordinary rationale:
    - circulates among experts. Experts re-estimate.
- Experts never meet each other to discuss their viewpoints as many estimators may easily get Influenced.

Heuristic Estimation Techniques

- Single Variable Model:
- Models provide a means to estimate the desired characteristics of a problem, **using some previously estimated basic (independent) characteristic** of the software product such as its size, staff etc.
    - 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)d2+…
    - Usually more accurate than single variable models.

COCOMO Model

- COCOMO (COnstructive COst MOdel) is a Constructive Cost Estimation Model proposed by Boehm.

- COCOMO Product classes 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.**

COCOMO Model

Divides software product developments into 3 categories:

- Organic: Relatively small groups. Working to develop well-understood applications.
  - **Examples** of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

- Semidetached: Project team consists of a mixture of experienced and inexperienced staff.
  - **Example** of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

- Embedded: The software is strongly coupled to complex hardware, or real-time systems.
  - **For Example**: ATM, Air Traffic control.

The Modes

- Organic
  - 2-50 KLOC, small, stable, little innovation
- Semi-detached
  - 50-300 KLOC, medium-sized, average abilities, medium time-constraints
- Embedded
  - > 300 KLOC, large project team, complex, innovative, severe constraints

COCOMO Model (CONT.)

- **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**

- 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)1.20PM

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

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

Example

⌘ The size of an organic software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the **effort** required to develop the software product and the **nominal development time and total cost**..

....

Example

- The size of an organic software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- **per month**. Determine the **effort** required to develop the software product and the **nominal development time and total cost**.

....

- Effort = 2.4*(32)1.05 = 91 **PM**
- Nominal development time = 2.5*(91)0.38 = 14 months
- Cost required to develop the product = **91 x 15,000**

  = **Rs. 13,65,000/-**

Average Staff Size

$$SS = \frac{E}{TDEV} = \frac{[\text{staff - }\cancel{\text{months}}]}{\cancel{[\text{months}]}} = [\text{staff}]$$

Productivity

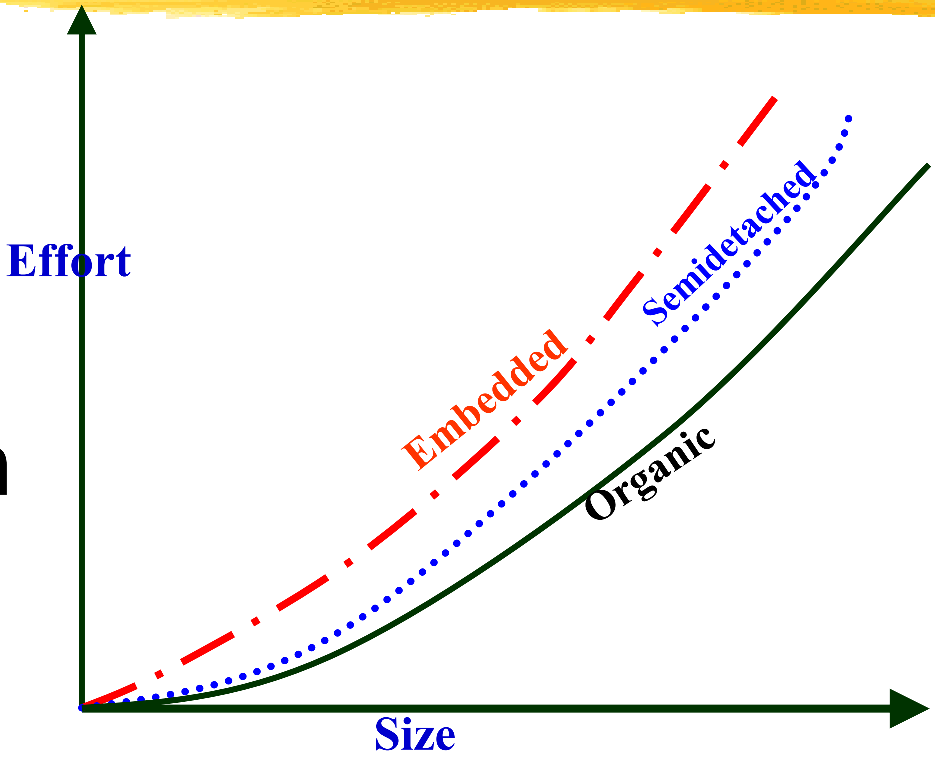
$$P = \frac{Size}{E} = \frac{[\text{KLOC}]}{[\text{staff - months}]} = \text{KLOC} \big/ \text{staff - month}$$

Suppose an organic project has 7.5 KLOC,

- Suppose an organic project has 7.5 KLOC,
  - Effort $2.4(7.5)^{1.05}$ = 20 staff–months
  - Development time $2.5(20)^{0.38}$ = 8 months
  - Average staff 20 / 8 = 2.5 staff
  - Productivity 7,500 LOC / 20 staff-months = 375 LOC / staff-month

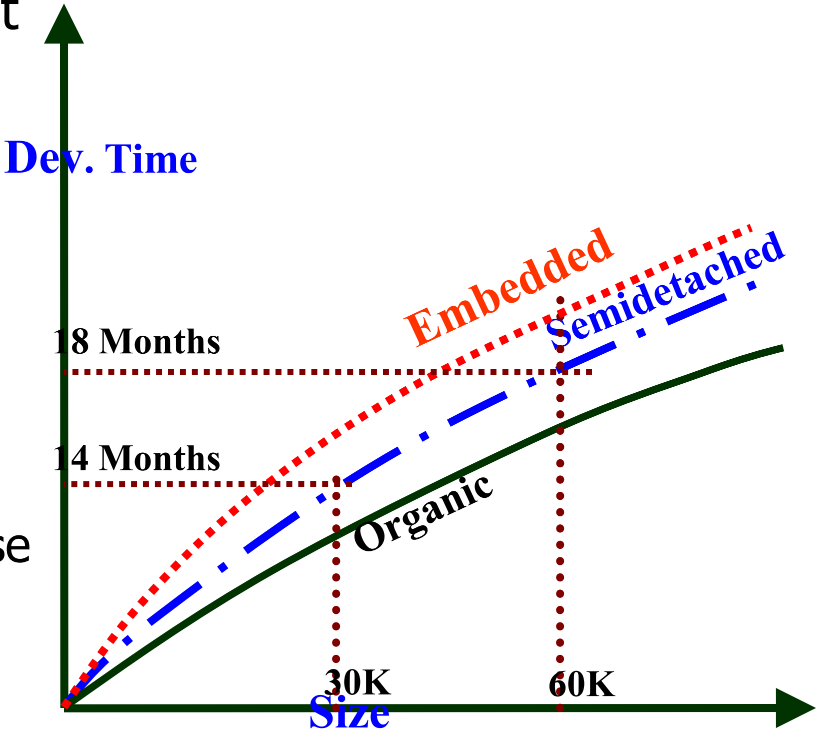
Basic COCOMO Model (CONT.)

- Effort is somewhat super-linear in problem size.



Basic COCOMO Model (CONT.)

- **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.
- Development time does not increase linearly with product size:
  - For larger products more parallel activities can be identified and performed by engineers



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
- 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).
    - Rate different parameters on a scale of **one to three** and multiply cost driver values with the estimate obtained using the basic COCOMO.
    - These 15 values are then multiplied to calculate the EAF **(Effort Adjustment Factor).**

Effort Adjustment Factor

- The Effort Adjustment Factor in the effort equation is simply the **product of the effort multipliers** corresponding to each of the cost drivers for your project.

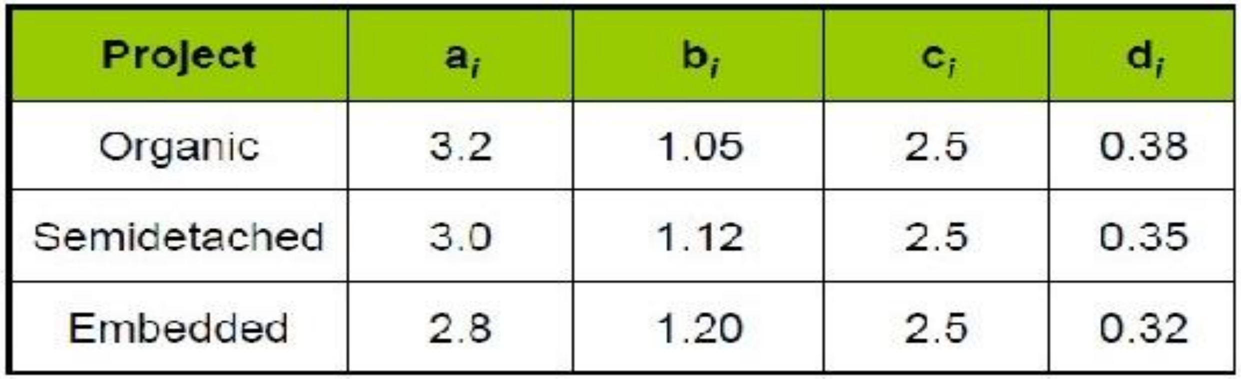
- For example,
- if your **project** is rated **Very High** for Complexity (effort multiplier of 1.34), and
- **Low** for **Language & Tools Experience** (effort multiplier of 1.09),
- and all of the **other cost drivers** are rated to be **Nominal** (effort multiplier of 1.00),
- the EAF is the product of 1.34 and 1.09.

Effort Adjustment Factor = EAF = 1.34 * 1.09 = 1.46

⌘Intermediate COCOMO equation:

$E=a_i\ (KLOC)\ b_i*EAF$

$Tdev=\ c_i\ (E)d_i$



Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Coefficients for intermediate COCOMO

Intermediate COCOMO (CONT.)

- Cost driver classes:
  - Product: characteristics of the product that include inherent complexity of the product, reliability requirements of the product, etc.
  - Computer: Execution time, storage requirements, etc.
  - Personnel: Experience of personnel, programming capability, analysis capability, etc.
  - Development Environment: Sophistication of the tools used for software development.

Complete COCOMO

- Both models:
  - consider a software product as a single homogeneous entity:
  - 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.

- 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.
  - **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.

Analytical Estimation Techniques

- **Halstead's software** analytical technique to estimate:
  - size,
  - development effort,
  - development time.

- **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.

Staffing Level Estimation

- Once the **effort** required to develop a software has been determined, it is necessary to determine the **staffing requirement for the project.**

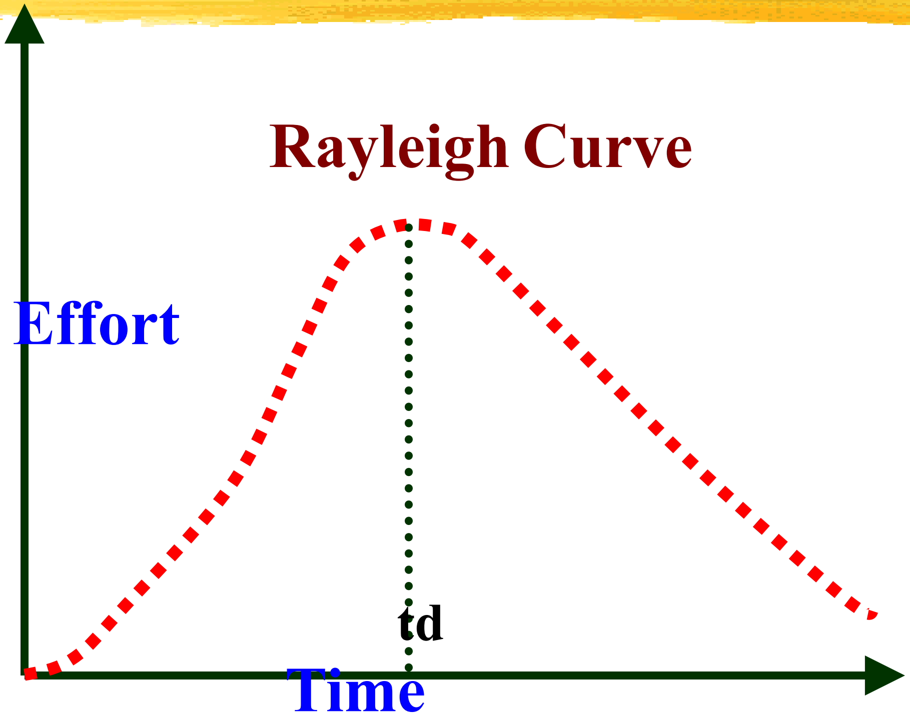
- Norden in 1958 analyzed many R&D projects, and observed:
  - **Rayleigh curve** represents the number of full-time personnel required at any time.

- Very small number of engineers are needed at the beginning of a project to carry out planning and specification.

- As the project progresses:
  - more detailed work is required for which number of engineers slowly increases and reaches a peak.
  - This is 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.

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)



Project Scheduling

- It involves deciding which tasks would be taken up when. In order to schedule the project activities, a software project manager needs to do the following task:
  - Identify all the tasks needed to complete the project.
  - Break down large tasks into small activities.
  - Determine the dependency among different activities.
  - Establish the most likely estimates for the time durations necessary to complete the activities.
  - Plan the starting and ending dates for various activities.
  - Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Work breakdown structure

- Decompose a given task set recursively into small activities.
- Root of the tree is labelled by the problem name.
- Each node of the tree is broken down into smaller activities that are made the children of the node.
- Each activity is recursively decomposed into smaller sub-activities until at the leaf level.

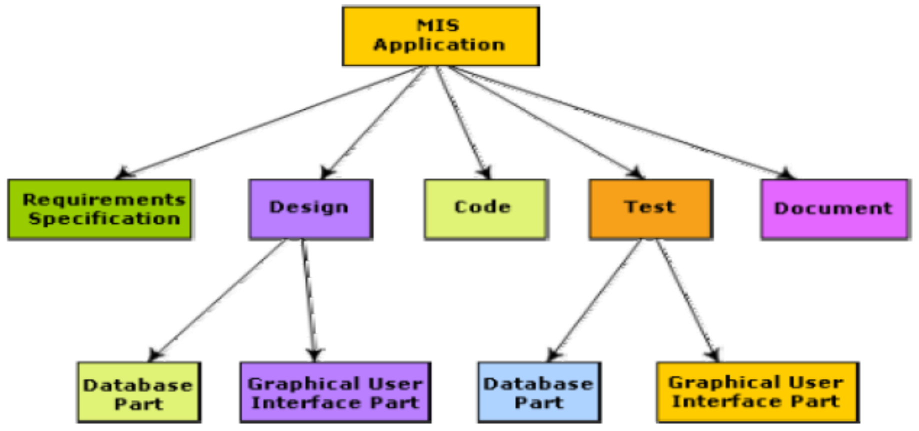

MIS Application
Requirements Specification
Design
Code
Test
Document
Database Part
Graphical User Interface Part
Database Part
Graphical User Interface Part

Fig. 11.7: Work breakdown structure of an MIS problem

Software Configuration Management

Software Configuration Management

- The deliverables of a SW product consist of a number of objects, e.g. **source code, design document, SRS document, test document, user's manual, etc**.
- These objects are modified by many software engineers through out development cycle.
- The state of each object changes as bugs are detected and fixed during development .

- **The configuration of the software is the state of all project deliverables at any point of time.**

- **SCM deals with effectively tracking and controlling the configuration of a software during its life cycle.**

Release vs. Version vs. Revision of SW

- A new version results from a significant change in functionality, technology, or the platform
  - Example: one version of a SW might be Unix-based, another Windows based.
- revision refers to minor bug fix .
- A release results from bug fix, minor enhancements to the functionality, usability
- A new release of SW is an improved system replacing the old one.
- Systems are described as Version m, Release n;

  or simple m.n.

Necessity of SCM

- To control access to deliverable objects with a view to avoiding the following problems
  - **Inconsistency problem when the objects are replicated.**
  - **Problems associated with concurrent access.**
  - **Providing a stable development environment.**
  - **System accounting and status information.**
  - **Handling variants.** If a bug is found in one of the variants, it has to be fixed in all variants

SCM activities

- **Configuration identification** :
  - deciding which objects (configuration items) are to be kept track of.
- **Configuration control :**
  - ensuring that changes to a system happen without ambiguity.

Baseline

- When an effective SCM is in place, the manager freezes the objects to form a **base line.**
- A **baseline** is the status of all the objects under configuration control. When any of the objects under configuration control is changed , a new baseline is formed.

Configuration item identification

- **Categories of objects:**
  - **Controlled objects:** are under Configuration Control (CC) . Formal procedures followed to change them.
  - **Precontrolled objects:** are not yet under CC, but will eventually be under CC.
  - **Uncontrolled objects:** are not subjected to CC
  - **Controllable objects** include both **controlled and precontrolled objects**; examples:

    SRS document, Design documents, Source code , Test cases

    The SCM plan is written during the project planning phase and it lists all controlled objects.

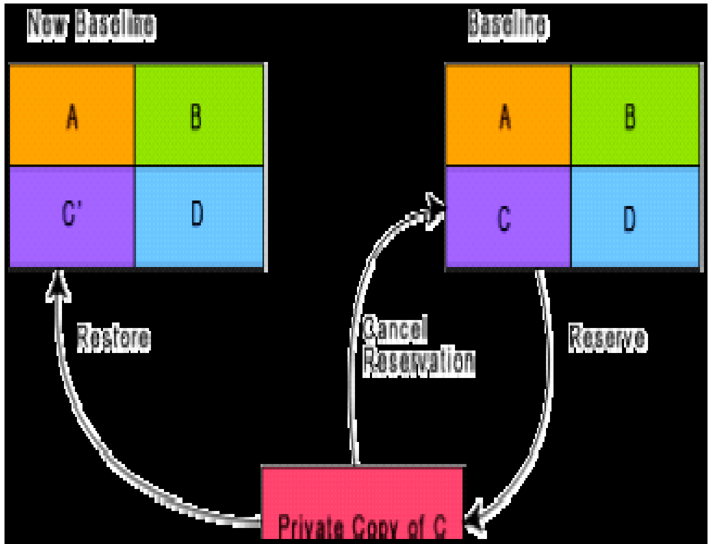
- **Configuration Control (CC):** process of managing changes to controlled objects. Allows authorized changes to the controlled objects and prevents unauthorized changes .

Changing the Baseline

- When one needs to change an object under configuration control, he is provided with a copy of the base line item.
- The requester makes changes to his private copy.
- After modifications, updated item replaces the old item and a new base line gets formed .

Reserve and restore operation in configuration control

- obtains a private copy of the module through a reserve operation.
- carries out all changes on this private copy.
- restoring the changed module to the baseline requires the permission of a change control board(CCB).
- Except for very large projects, the functions of the CCB are discharged by the project manager himself.

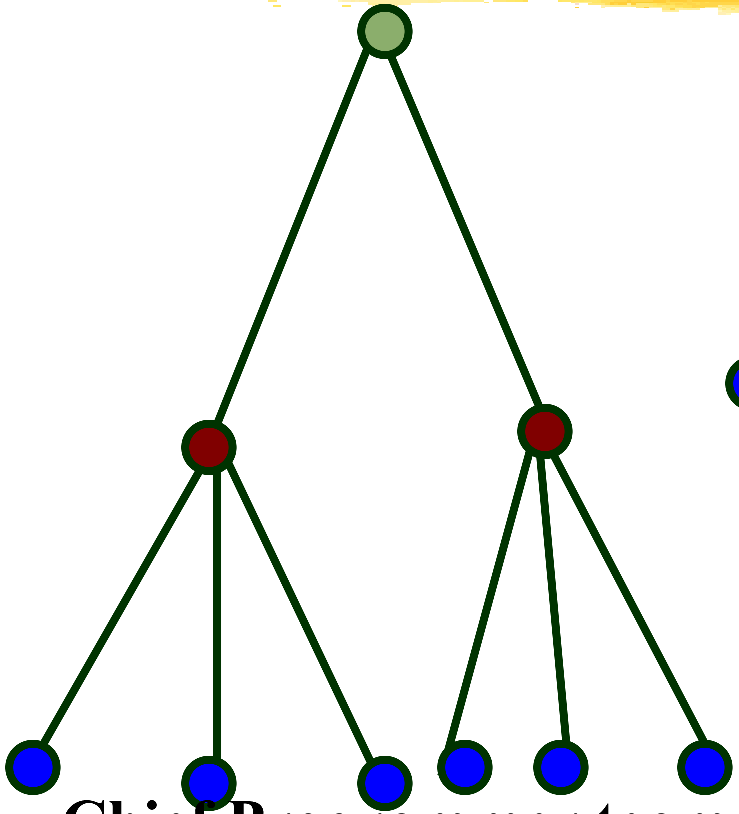


Organization Structure

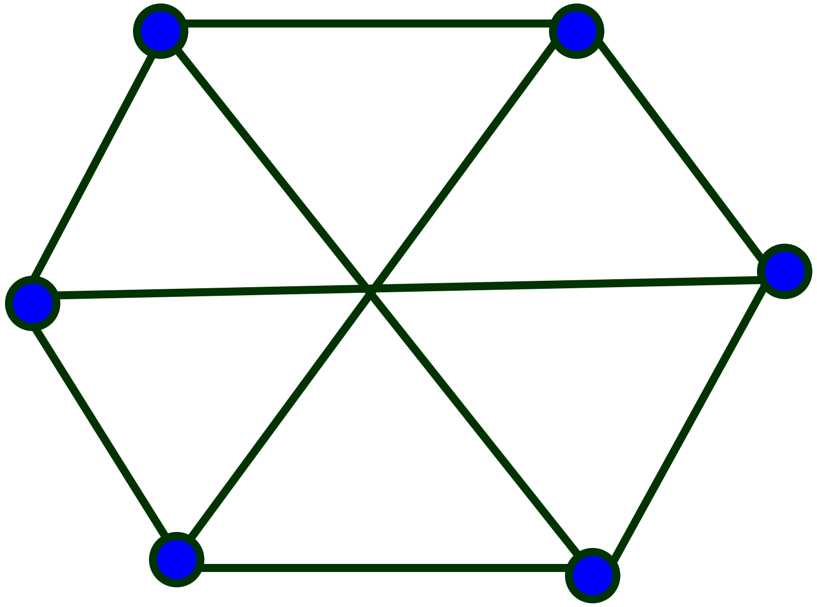
- Functional Organization:
  - Engineers are organized into functional groups, e.g.
    - specification, design, coding, testing, maintenance, etc.
  - Engineers from functional groups get assigned to different projects

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

Team Organization

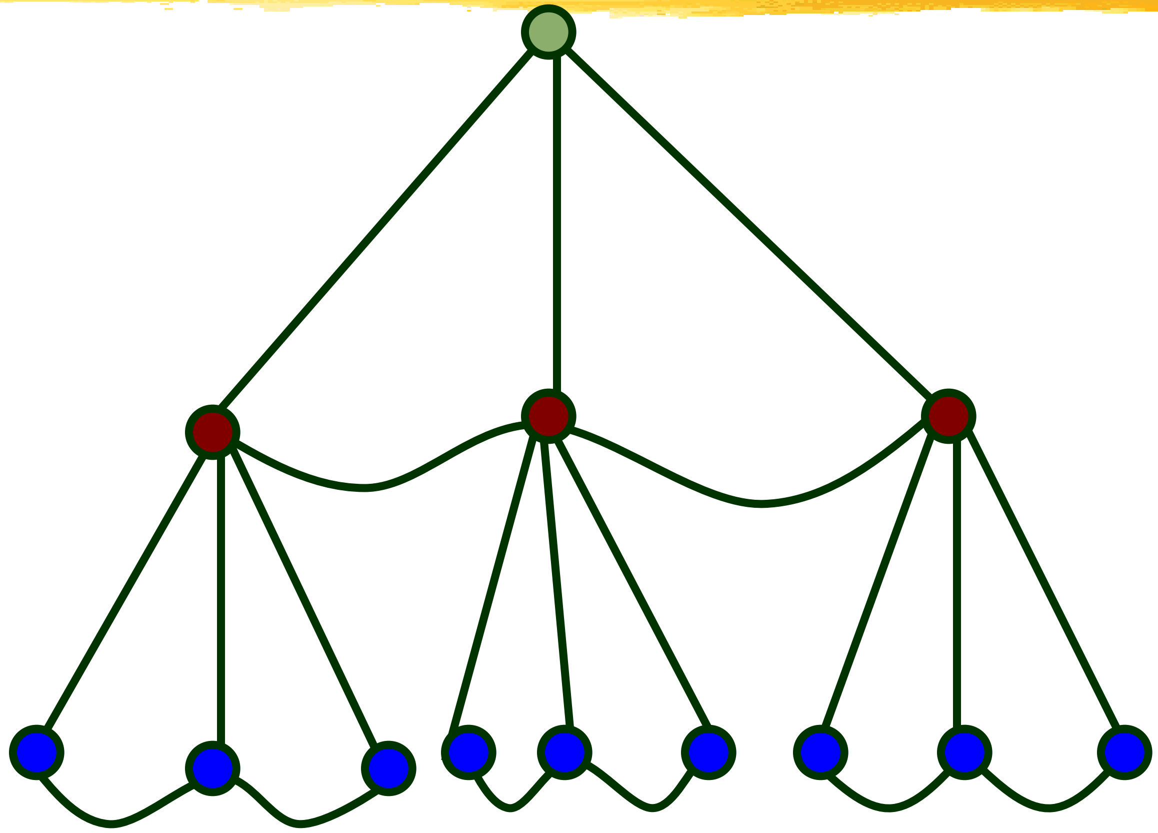


Chief Programmer team



Democratic Team

Mixed team organization



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.

- 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.

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 organization provides
  - higher morale and job satisfaction to the engineers
  - therefore leads to less employee turnover.
- Disadvantage:
  - team members may waste a lot time arguing about trivial points:
    - absence of any authority in the team.

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.