Software Engineering (CSE3004)

Software Project Management

Puneet Kumar Jain

CSE Department

National Institute of Technology Rourkela

1/24/2022

Reference

- R. Mall, Fundamentals of Software Engineering, Fifth Edition, PHI Learning Pvt Ltd., 2018.

- Lecture PPT of Software Project, Process and Quality Management by Prof. D P Mohapatra.

- Few slides from: Sommerville, " Introduction to Software Engineering", 8th Edition, Addison-Wesley, 2007

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.

What is a project?

Some dictionary definitions:

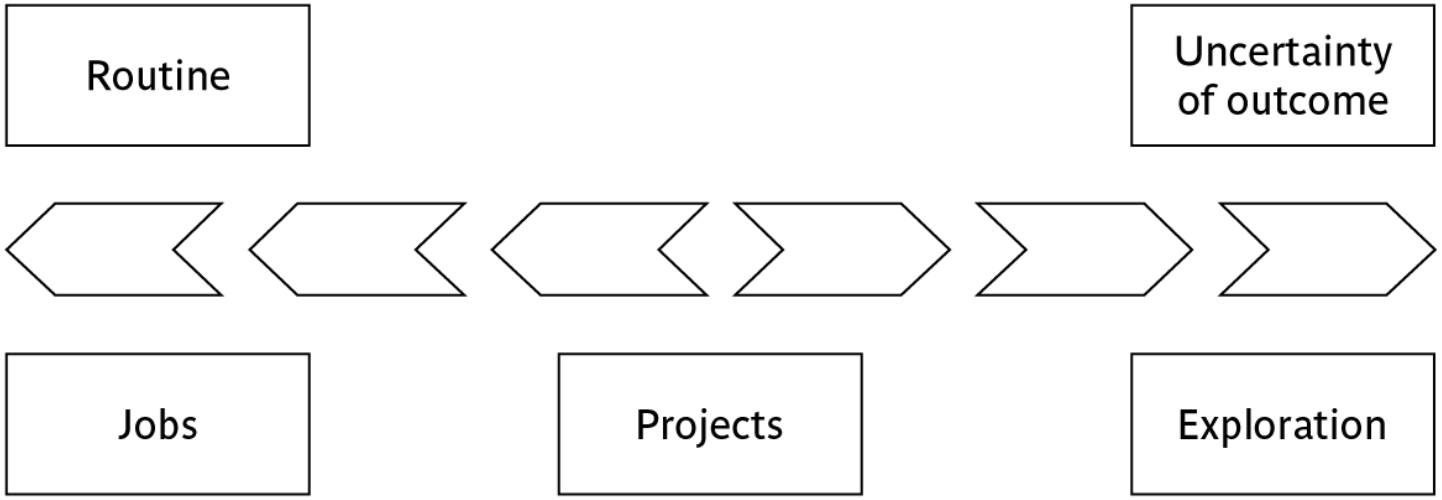
"A specific plan or design"

"A planned undertaking"

"A large undertaking e.g. a public works scheme"

Key points above are *planning* and *size* of task

Jobs versus projects



'Jobs' – repetition of very well-defined and well understood tasks with very little uncertainty

'Exploration' – e.g. finding a cure for cancer: the outcome is very uncertain

Projects – in the middle!

Why is project management important?

- Project often fail – Standish Group claim only a third of ICT projects are successful. 82% were late and 43% exceeded their budget.
- Poor project management is a major factor in these failures

SOFTWARE PROJECT MANAGEMENT COMPLEXITIES

- Management of software projects is much more complex than management of many other types of projects due to the following factors:

- **Invisibility:** Software remains invisible, until its development is complete and it is operational.
  - Invisibility of software makes it difficult to assess the progress of a project and is a major cause for the complexity of managing a software project.

- **Changeability:** Because the software part of any system is easier to change as compared to the hardware part, the software part is the one that gets most frequently changed.

SOFTWARE PROJECT MANAGEMENT COMPLEXITIES

- **Complexity:** Even a moderate sized software has millions of parts (functions) that interact with each other in many ways—data coupling, serial and concurrent runs, state transitions, control dependency, file sharing, etc.

- **Uniqueness:** Every software project is usually associated with many unique features or situations. This makes every project much different from the others.

- **Exactness of the solution:** the parameters of a function call in a program are required to be in complete conformity with the function definition.

- **Team-oriented and intellect-intensive work:** In a software development project, the life cycle activities not only highly intellect intensive, but each member has to typically interact, review, and interface with several other members, constituting another dimension of complexity of software projects.

Job Responsibilities for Managing Software Projects

- A software project manager takes the overall responsibility of
  - Project proposal writing,
  - Project cost estimation,
  - Scheduling,
  - Project staffing,
  - Software process tailoring,
  - Project monitoring and control,
  - Software configuration management,
  - Risk management,
  - Managerial report writing and presentation, and
  - Interfacing with clients

Job Responsibilities for Managing Software Projects

- Project manager's responsibilities can be classified into the following two major categories:

- **Project planning:**
  - Involves estimating several characteristics of a project and then planning the project activities based on these estimates made.
  - Undertaken immediately after the feasibility study phase and before the starting of the requirements analysis and specification phase.

- **Project monitoring and control:**
  - The focus of project monitoring and control activities is to ensure that the software development proceeds as per plan. Project monitoring and control activities are undertaken once the development activities start.

Project Planning

- The project manager performs the following activities.

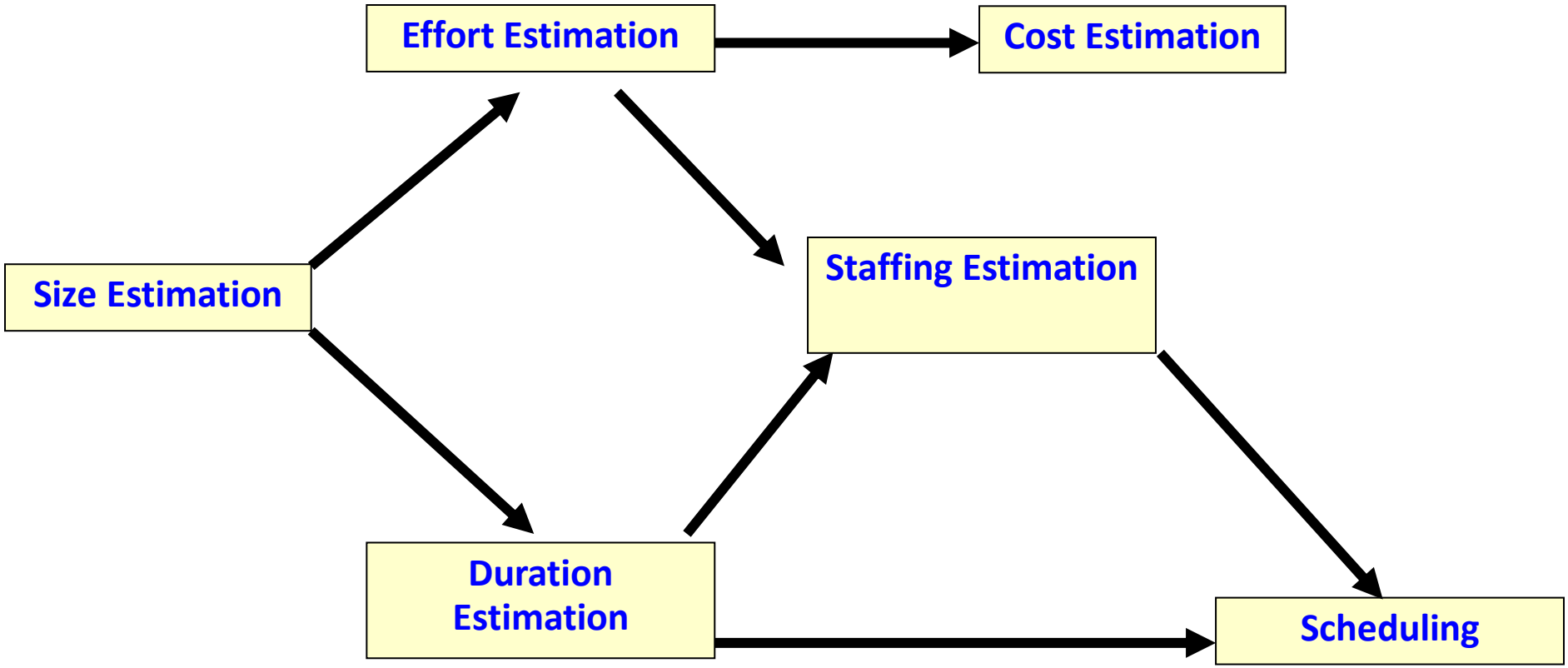
- **Estimation:** The following project attributes are estimated.
  - **Cost:** How much is it going to cost to develop the software product?
  - **Duration:** How long is it going to take to develop the product?
  - **Effort:** How much effort would be necessary to develop the product?
- **Scheduling:** After all the necessary project parameters have been estimated, the schedules for manpower and other resources are developed.
- **Staffing:** Staff organisation and staffing plans are made.
- **Risk management :** This includes risk identification, analysis, and abatement planning.
- **Miscellaneous plans:** This includes making several other plans such as quality assurance plan, and configuration management plan, etc.

Software Cost Estimation

Which Project Parameters to be Estimated?

- What is the size of the software to be developed?
- How much effort is required?
- How much calendar time is needed?
- What is the total cost?

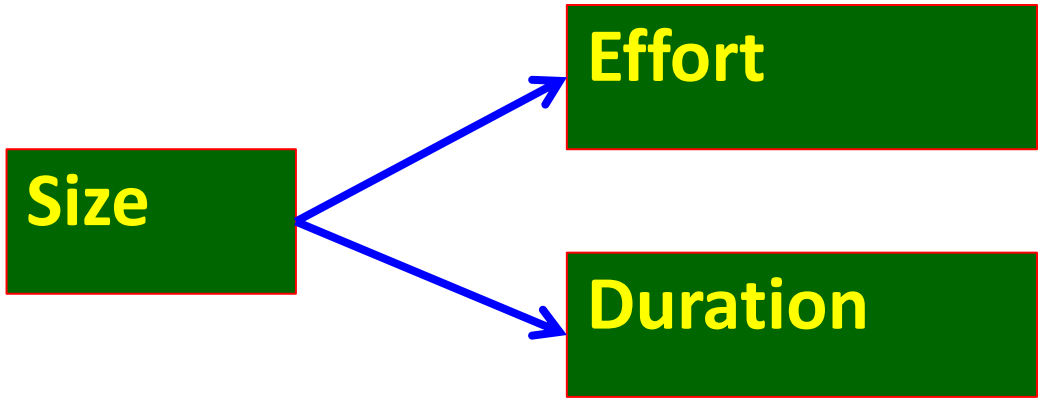
Activities Dependent on Estimation Results



- Size is the most fundamental parameter based on which all other estimations and project plans are made.

Parameters to be Estimated

- Size is a fundamental measure of work and also direct metric.
- Based on the estimated size, two important parameters are estimated:
  - **Effort**
  - **Duration**



- Effort is measured in person-months:
  - **One person-month is the effort an individual can typically put in a month.**

What is size? A Measure of Work…

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

- Two metrics are popularly used to measure project size:
  - **Source Lines of Code (SLOC)**
  - **Function point (FP)**

Lines of code

- What's a line of code?
  - Originally proposed when programs were typed on cards with one line per card;
  - What happens when statements in Java span several lines or where there can be several statements on one line?

- Which programs should be counted as part of the system? GUI? Built-in Class?

- Initially software development consisted of only writing code…

LOC: Few things counter intuitive…

- The lower the level of the language, the more productive the programmer is:
  - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity:
  - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code.

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

Major Shortcomings of SLOC

- **Difficult to estimate at the start of a project…**
  - Only way to estimate is to make a guess…
- Only a code measure…
- Programmer-dependent…
- Does not consider code complexity

Further Difficulties With SLOC

- SLOC has become an ambiguous metric due to rapid changes in programming methodologies, languages, and tools:

  -Language advancements

  -Automatic source code generation

  -Custom software and reuse

-Object orientation

Function points: Albrecht/IFPUG

- **IFPUG: International Function Point Users Group**
- It is developed by Albrecht and accepted by IFPUG
- Albrecht worked at IBM:
  - Needed a way of measuring the relative productivity of different programming languages.
  - **Needed some way of measuring the size of an application without counting lines of code.**

- **Identified five parameters:**
  - Counted occurrences of each type of functionality in order to get an indication of the size of an information system

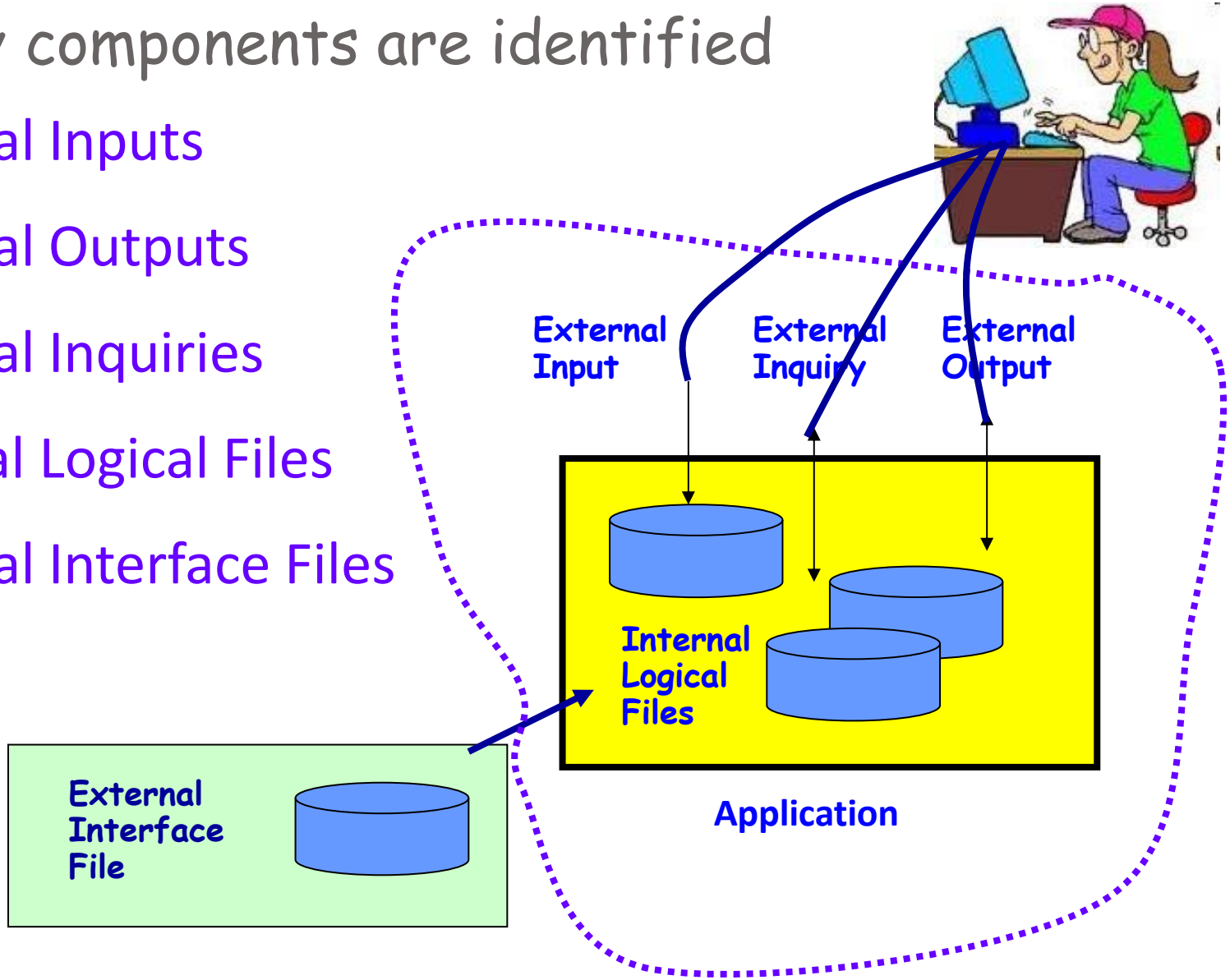
How Do Function Points Overcome LOC Problems?

- Function points are independent of the language, tools, or methodologies used for implementation
- Function points can be estimated early in analysis and design
- Simple enough to minimize the overhead of the measurement process…
- A consistent measure among various projects and organizations …
- Since function points are based on the user's external view of the system:
  - **Non-technical users of the software have a better understanding of what function points are measuring**

Function Point

Five key components are identified

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



Albrecht/IFPUG function points - continued

Five function types:

1. Logical interface file (LIF) types

- Equates roughly to a data store in the system that is created and accessed by the system

2. External interface file types (EIF)

- Represents data retrieved from a data store which is actually maintained by a different application.

Albrecht/IFPUG function points - continued

3. **External input (EI) types:**
   - Transactions which update internal computer files

4. **External output (EO) types:**
   - Transactions which extract and display data from internal computer files.
   - Generally involves creating reports.

5. **External inquiry (EQ) types:**
   - **User initiated transactions which provide information but do not update computer files.**
   - Normally the user inputs some data that guides the system to the information the user needs.

FP Computation 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.

Step1: Unadjusted function point

- UFP = (Number of inputs) ***4**
  +(Number of outputs) ***5**
  +(Number of inquiries) ***4**
  +(Number of files) ***10**
  +(Number of interfaces) ***7**

Step2: Refine parameters

- In step 1, each parameter (input, output, etc.) has been implicitly assumed to be of average complexity
- 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

External user types	Low complexity	Medium complexity	High complexity
EI	3	4	6
EO	4	5	7
EQ	3	4	6
LIF	7	10	15
EIF	5	7	10

IFPUG file type complexity

Number of record types	Number of data types		
	< 20	20-50	> 50
1	Low	Low	Average
2 to 5	Low	Average	High
> 5	Average	High	High

Example - 1

Tom is starting a dental practice in a small town. He will have a dental assistant, a dental hygienist, and a receptionist. He wants a system to manage the appointments.

- When a patient calls for an appointment, the receptionist will check the calendar and will try to schedule the patient as early as possible to fill in the vacancy.
- If the patient is happy with proposed appointment, the receptionist will enter the appointment with the patient name and purpose of appointment.
- The system will verify the patient name and supply necessary details from the patient records, including the patient's ID number.
- After each exam or cleaning, the hygienist or assistant will mark the appointment as completed, add comments and, then schedule the patient for the next visit if appropriate.

Example – 1 cont..

- The system will answer queries by patient name and by date. Supporting details from the patient's records are displayed along with the appointment information.
- The receptionist can cancel appointments. The receptionist can point out a notification list for making reminder calls 2 days before appointments.
- The system includes the patient's phone numbers from the patient records.
- The receptionist can also print out daily and weekly work schedules with all the patients.

Calculate Unadjusted Function Point (UFP).

Solution

Type	Simple	Average	Complex	Total
Inputs	• Patient name • Appointment completed • Appointment purpose	• Cancel appointment		3*3+ 1*4=13
Outputs	• Comments	• Calendar • Supporting details • Appointment information • Notification list	• Daily schedule • Weekly schedule	1*4+ 4*5+ 2*7=38
Inquires	• Query by name • Query by date	• Verify patient • Check calendar • Available appointment		2*3+ 3*4=18
Files		• Patient data		10
Interfaces				
Total				79

Step3: Function Point Refinement

14 General Systems Characteristics are evaluated and used to compute a Value Adjustment Factor (VAF)/ Technical Complexity Factor (TCF)

General System Characteristics

Data Communication	On-Line Update
Distributed Data Processing	Complex Processing
Performance Objectives	Reusability
Heavily Used Configuration	Conversion & Install Ease
Transaction Rate	Operational Ease
On-Line Data Entry	Multiple-Site Use
End-User Efficiency	Facilitate Change

The final calculation is based upon the **UFP X TCF**

Degrees of Influence

- 0 Not present, or no influence
- 1 Incidental influence
- 2 Moderate influence
- 3 Average influence
- 4 Significant influence
- 5 Strong influence throughout

Function point calculation

- Evaluate each of the 14 general system characteristics on a scale from zero to five to determine the degree of influence (DI)
- Add the degrees of influence for all 14 general system characteristics to produce the total degree of influence (TDI).
- Insert TDI into the following equation to produce the Value Adjustment Factor (VAF) / Technical Complexity Factor (TCF)

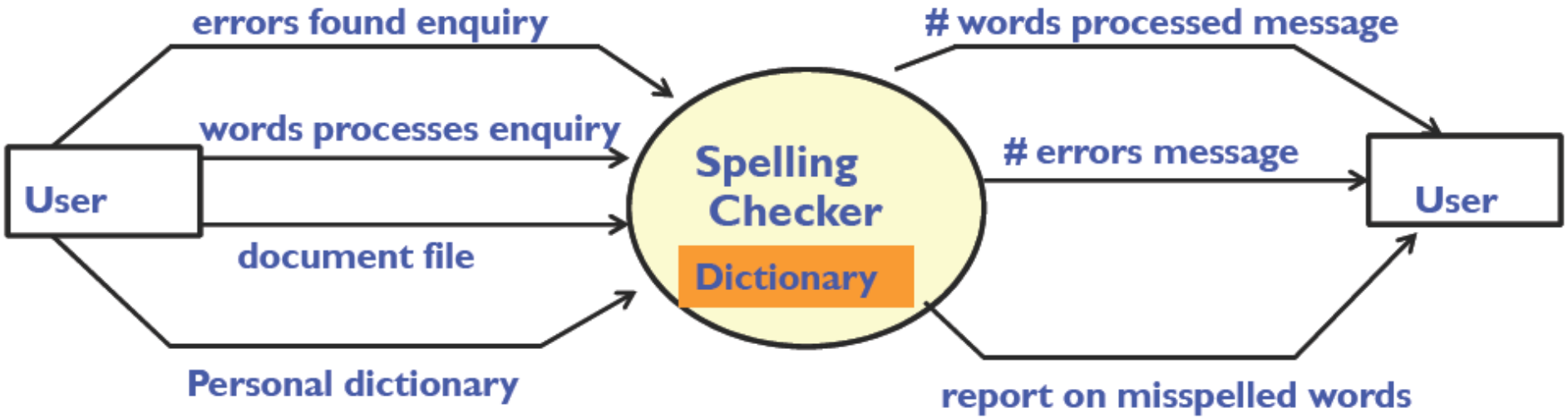
  TCF = (TDI * 0.01) + 0.65

- It expresses the overall impact of the corresponding parameter on the development effort. As TDI can vary from 0 to 70, TCF can vary from 0.65 to 1.35.
- Finally, the function point can be calculated as:

FP=UFP*TCF

Example: spell-checker

- The Spell-Checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed and the number of spelling errors found at any stage during processing.



Example: spell-checker

- 2 users inputs: document file name, personal dictionary name (average)
- 3 users outputs: error report, word count, misspelled error count (average)
- 2 users Inquiries: # processed words?, #spelling errors? (average)
- 1 internal file: dictionary (average)
- 2 external files: document file, personal dictionary (average).
- We know that, for average (medium) complexity parameters,
- UFP = # inputs*4+ # outputs*5+ # inquiries*4+ # files*10+ #interfaces*7
  = 2*4 + 3*5 + 2*4 + 1*10 + 2*7
  = 55

Example: spell-checker

- **TDI calculation:**

1. Data Communication	3
2. Distributed Data Processing	0
3. Performance Criteria	4
4. Heavily Utilized Hardware	0
5. High Transaction Rates	3
6. Online Data Entry	3
7. Online Updating	3
8. End-user Efficiency	3
9. Complex Computations	0
10. Reusability	3
11. Ease of Installation	3
12. Ease of Operation	5
13. Portability	3
14. Maintainability	3
Total Degree of Influence (TDI)=	**36**

Example: spell-checker

- So,

  TCF = (36 * 0.01) + 0.65 = 1.01

- The Adjusted Function Point is calculated as follows:

  FP = UFP * TCF

  = 55 * 1.01

  = 55.55

Function point metric shortcomings:

- 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.

Both metrics are subjective in nature

Project estimation techniques

- Estimation techniques can broadly be classified into three main categories:

- **Empirical estimation techniques**
  - Expert judgment
  - Delphi estimation

- **Heuristic techniques**
  - Single variable model (COCOMO)
  - Multivariable model (Intermediate COCOMO)

- **Analytical estimation techniques**
  - Halstead's software science

Empirical estimation techniques

- Empirical estimation techniques are essentially based on making an educated guess of the project parameters.

- Although empirical estimation techniques are based on common sense and subjective decisions, over the years, the different activities involved in estimation have been formalised to a large extent.

- We shall discuss two such formalisations of the basic empirical estimation techniques
  - Expert judgement and
  - Delphi techniques

Expert judgement

- An expert:
  - One familiar with and knowledgeable about the application area and the technologies
- This technique particularly appropriate if:
  - Experts worked with similar applications
  - Existing code is to be modified: Customization, maintenance, etc.
- Research shows that expert judgement in practice tends to be based on analogy…

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.

Expert Judgment: Pros and Cons

- Advantages: Relatively simple estimation method. Can be accurate if experts have direct experience of similar systems

- Disadvantages:
  - Subject to human error and individual bias
  - Very inaccurate if there are no experts available!

Delphi Estimation

- A variation of consensus estimation technique
- Team of Experts and a coordinator.
- Experts carry out estimation independently based on given SRS:
  - mention the rationale behind their estimation.
  - coordinator notes down any extraordinary rationale:
    - circulates the estimation rationale among experts.
- Experts re-estimate.
- This process is iterated several rounds.
- Experts never meet each other
  - to discuss their viewpoints.
- After the completion of several iterations, co-ordinator compiles the result

Expert Judgement: Cons

- It is often hard to document the specific factors used by the experts.
- Expert may be biased, optimistic, or pessimistic:
  - **Of course, these to some extent get decreased by the group consensus.**
- The expert judgment method can complement the other cost estimating methods such as algorithmic method.

Heuristic Techniques

- Heuristic techniques assume that the relationships that exist among the different project parameters can be satisfactorily modelled using suitable mathematical expressions.

- Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the values of the independent parameters in the corresponding mathematical expression.

- Different heuristic estimation models can be divided into the following two broad categories—single variable and multivariable models.

Single variable estimation model

- Single variable estimation models assume that various project characteristic can be predicted based on a single previously estimated basic (independent) characteristic of the software such as its size.

  Estimated Parameter = c_1 x e^{d1}

- e: a characteristic of the software that has already been estimated such as **effort**, **project duration**, **staff size**, etc.,

- c_1 and d_1: constants. The values of the constants c_1 and d_1 are usually determined using data collected from past projects (historical data).

- COCOMO model

Multivariable Model

- Assumes that the parameter to be estimated depends on more than one characteristic.

$$\textbf{Estimated resource} = C_1(p_1)^{d1} + C_2(p_2)^{d2} + \ldots$$

- Usually more accurate than single variable models, since project parameter is typically influenced by several independent parameters.

- Intermediate COCOMO model

COCOMO Model

- COCOMO (COnstructive COst MOdel)
- First published by Dr. Barry Boehm, 1981
- Consider not only the characterisitcs of the product but also of the development team and development environment.
- Divides software product developments into 3 categories:
  - Organic
  - Semidetached
  - Embedded
- Roughly correspond to:
  - Application Programs: Data processing and scientific programs
  - Utility Programs: Compilers, linkers, editors, etc.
  - System programs: Operating systems and real-time system programs, etc.

Elaboration of Product classes

- Organic:
  - Relatively small groups
    - working to develop well-understood applications.
  - Developed in familiar, stable environment
  - <50,000 DSIs (ex: accounting system)

- Semidetached:
  - Project team consists of a mixture of experienced and inexperienced staff.
  - Somewhere between Organic and Embedded

- Embedded:
  - The software is strongly coupled to complex hardware, or real-time systems.
  - New product requiring a great deal of innovation
  - Inflexible constraints and interface requirements

Modes

Feature	Organic	Semidetached	Embedded
Organizational understanding of product and objectives	Thorough	Considerable	General
Experience in working with related software systems	Extensive	Considerable	Moderate
Need for software conformance with pre-established requirements	Basic	Considerable	Full
Need for software conformance with external interface specifications	Basic	Considerable	Full

COCOMO Models

- Three increasingly complex models
  - Basic COCOMO Model
    - Used for early rough, estimates of project cost, performance, and schedule
    - Accuracy: within a factor of 2 of actuals 60% of time
  - Intermediate COCOMO Model
    - Uses Effort Adjustment Factor (EAF) from 15 cost drivers
    - Doesn't account for 10 - 20 % of cost (trng, maint, Qual, etc)
    - Accuracy: within 20% of actuals 68% of time
  - Complete COCOMO Model
    - Uses different Effort Multipliers for each phase of project (Most project managers use intermediate model)

Basic COCOMO Model (CONT.)

- Gives only an approximate estimation:
  - Effort = a_1 x $(KLOC)^{a2}$ PM
  - T_{dev} = b_1 x $(Effort)^{b2}$ Months
    - KLOC: estimated kilo lines of source code,
    - a_1,a_2,b_1,b_2: constants for different categories of software products
    - T_{dev}: 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

System type	A	k
Organic (broadly, information systems)	**2.4**	**1.05**
Semi-detached	**3.0**	**1.12**
Embedded (broadly, real-time)	**3.6**	**1.20**

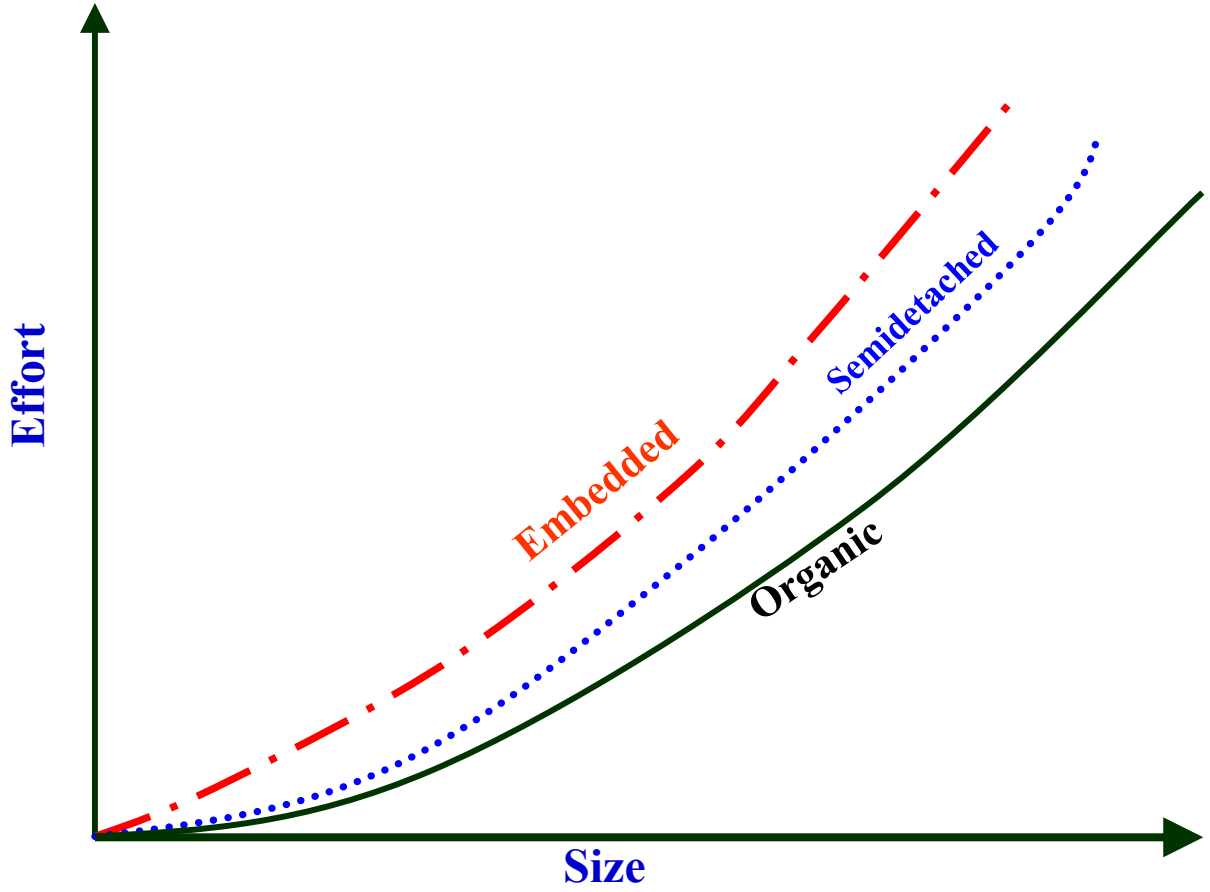
Development Time Estimation

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

System type	A	k
Organic (broadly, information systems)	**2.5**	**0.38**
Semi-detached	**2.5**	**0.35**
Embedded (broadly, real-time)	**2.5**	**0.32**

Basic COCOMO Model (CONT.)

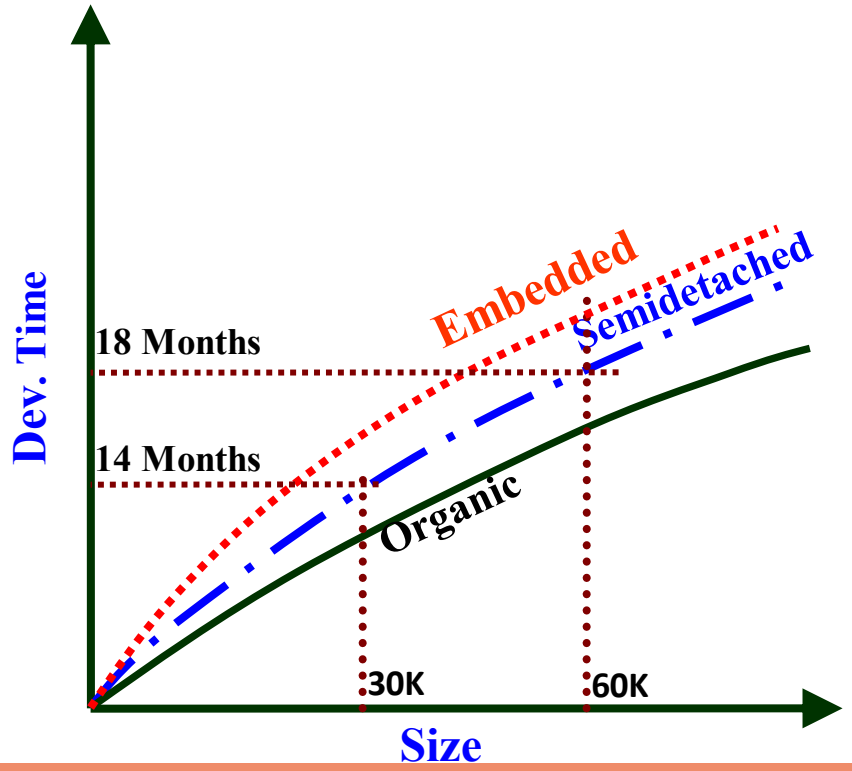
- Effort is somewhat super-linear in problem size.

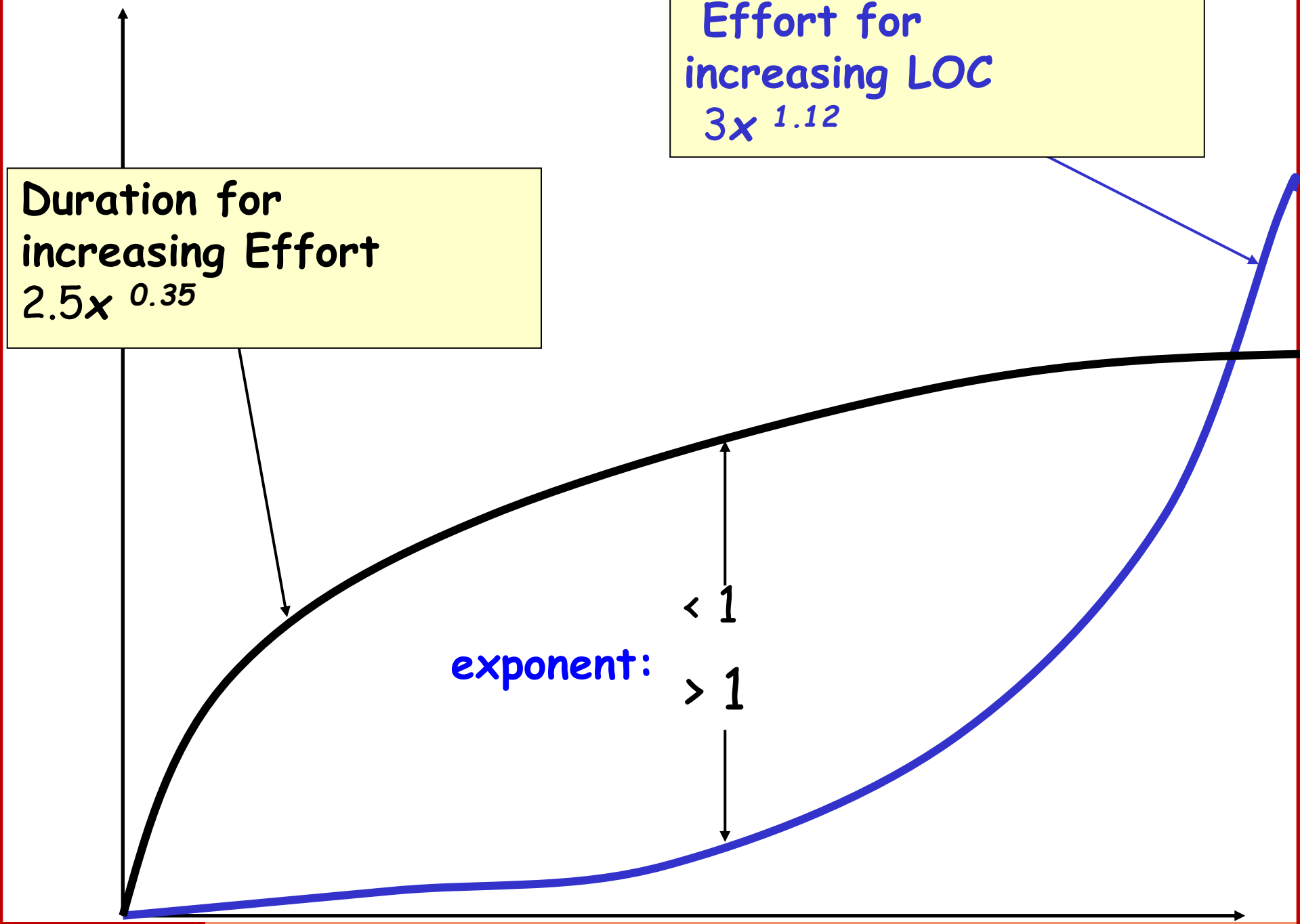


- Increase in effort with size is not bad
  - COCOMO assumes that projects are carefully designed and developed

Basic COCOMO Model (CONT.)

- Development time is sublinear function of product size.
- When product size increases two times,development time does not double.
  - Project development is carried out by a team of developers
- Time taken:
  - almost same for all the three product categories.




Effort for increasing LOC
3x 1.12
Duration for increasing Effort
2.5x 0.35
exponent:
< 1
> 1

Basic COCOMO Model (CONT.)

- 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.

- Development time is roughly the same for all the three categories of products:
  - 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.

Example

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

.....

- Effort = $2.4*(32)^{1.05}$ = 91 PM
- Nominal development time = $2.5*(91)^{0.38}$ = 14 months
- Staff cost required to develop the product

  =91 x Rs. 15,000=Rs. 1,465,000

Example - 2

Suppose you are developing a software product in the organic mode. You have estimated the size of the product to be about 100,000 lines of code. Compute the nominal effort and the development time.

- Given that the size is 100 KLOC and the project is organic.
- Nominal effort=$2.4 \times 100^{1.05}=2.4 \times 125.893=302.1$ man-months
- Nominal development time=$2.5 \times (Effort)^{0.38}=2.5 \times 302.1^{0.38}=8.6$ months

Example - 3

Suppose that a certain software product for business application costs Rs. 50,000 to buy off-the-shelf and that its size is 40 KLOC. Assuming that in-house developers cost Rs. 6000 per PM (including overheads), would it be more cost-effective to buy the product or build it?

The product is for business application and can be classified as organic type. So,

- Nominal effort=2.4 x $40^{1.05}$=2.4 x 48.1=115.5 man-months
- In-house engineers cost Rs. 6000/-.
- So, the cost of development is 115.5 x 6000=Rs. 693,000/-
- But, purchasing the above S/W will cost Rs. 50,000.
- So, it is better to go for buying the product.

Example - 4

Suppose an organic project has 7.5 KLOC. Find the effort, development time, average staff required and productivity.

- Effort $2.4x(7.5)^{1.05}$= 20 staff–months
- Development time $2.5x(20)^{0.38}$= 8 months
- Average staff required 20 / 8 = 2.5 staff
- Productivity 7,500 LOC / 20 staff-months = 375 LOC / staff-month

Example - 5

Suppose an embedded project has 50 KLOC. Find the effort, development time, average staff required and productivity.

- Effort $3.6x(50)^{1.20}$= 394 person–months
- Development time $2.5x(394)^{0.32}$= 17 months
- Average staff 394 / 17 = 23 staff
- Productivity 50,000 LOC / 394 staff-months = 127 LOC / staff-month

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).

Intermediate COCOMO (CONT.)

- 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.

Cost driver classes and attributes

- Product:
  - Reliability requirement
  - Database size
  - Product complexity
- Computer:
  - Execution time constraints
  - Storage constraints
  - Virtual machine volatility
  - Computer turnaround time

Attributes

- Personnel:
  - Analyst capability
  - Virtual machine experience
  - Programmer capability
  - Programming language experience
  - Application experience

- Development Environment:
  - Modern programming practices
  - Software tools
  - Required development schedule

COCOMO effort multipliers (cost drivers)

- Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value).
- The product of all effort multipliers results in an effort adjustment factor (EAF).
- Typical values for EAF range from 0.9 to 1.4.

COCOMO – cost drivers

Cost Driver	Very low	Low	Nominal	High	Very High	Extra High
Required reliability	0.75	0.88	1.0	1.15	1.40	
Database size		0.94	1.0	1.08	1.16	
Product complexity	0.70	0.85	1.0	1.15	1.30	
Execution time constraint			1.0	1.11	1.30	
Memory constraint			1.0	1.06	1.21	
Virtual machine volatility		0.87	1.0	1.15	1.30	
Computer turnaround time		0.87	1.0	1.07	1.15	
Analyst capability	1.46	1.19	1.0	0.86	0.71	
Application experience	1.29	1.13	1.0	0.91	0.82	
Programmer capability	1.42	1.17	1.0	0.86	0.70	
Virtual machine experience	1.21	1.10	1.0	0.90		
Programming language experience	1.14	1.07	1.0	0.95		
Modern programming practices	1.24	1.10	1.0	0.91	0.82	
Use of software tools	1.24	1.10	1.0	0.91	0.83	
Development schedule	1.23	1.08	1.0	1.04	1.10	

Intermediate COCOMO

- **Effort=EAF*c*(size)^k**
- EAF (effort adjustment factor) is the product of effort multipliers corresponding to each cost driver rating
- c is a constant based on the developmental mode

- Some papers have taken value of c same as that of Basic COCOMO, some other papers have taken value of c as
  - for organic products, c = 3.2
  - for semi-detached products, c = 3.0
  - for embedded products, c = 2.8

- size = 1000s Delivered Source Instruction (KDSI)
- k is a constant for a given mode, same as Basic COCOMO

- The development time calculation uses effort in the same way as in the Basic COCOMO

  Nominal Development time= 2.5*(Effort)$^{\text{exponent}}$

Example

- Determine effort, duration, and staffing level for the following scenario:
  - Estimated size 10,000 LOC = 10 KLOC.
  - Small project, familiar development
  - Analyst capability: Low
  - Application experience: Low
  - programmer capability: low
  - Programmer experience: High

Example

- Need to produce 10,000 LOC = 10 KLOC.
- Since a small project and familiar development, use organic model:

- Effort =$2.4(10)^{1.05}$ = 26.9 Person-Months

- Development-time = $2.5(26.9)^{0.38}$ = 8.73 Months

- Average Staff = 26.9 PM/8.73 Months = 3.08 ~= 3 People

Example

- Now, the attribute multipliers will be as follows:
  - Analyst capability - 1.19 (low)
  - Application experience - 1.13 (low)
  - Programmer capability - 1.17 (low)
  - Programming experience - 0.95 (high)

- **So, Adjustment factor = 1.19*1.13*1.17*0.95 = 1.49**

- Effort = 26.9*1.49 = 40 Person-months

- Development time = $2.5*(40)^{0.38}$ = 10.2 Months

- Average Staff = 40PM/10.2M=3.9 (approx. 4) People

Example 2

- Suppose the project to be developed is a flight control system (mission critical) with 319,000 DSI in embedded mode.
- Reliability must be very high (RELY=1.40). So we can calculate:
  - Effort = $1.40*3.6*(319)^{1.20}$ = 5093 PM(approx.)
  - Duration = $2.5*(5093)^{0.32}$ = 38.4 months (approx.)
  - Average Staffing = 5093 PM/38.4 months = 133 People (approx.)

Example 3

- An embedded software system on microcomputer hardware to be developed.
- Basic COCOMO predicts a 45 person-month effort requirement
- Attributes = RELY (1.15), STOR (1.21), TIME (1.10), TOOL (1.10)
- Intermediate COCOMO predicts
- 45 * 1.15 * 1.21 * 1.10 *1.10 = 76 person-months.
- Assume total cost of person month = Rs. 50,000.
- Total cost = 76 * 50,000 = Rs.38,00, 000

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, some of them may be considered as organic type, some semidetached, and some even embedded

- 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.

Drawbacks of COCOMO 81

- For better accuracy:
  - COCOMO has to be calibrated to an organizations' environment.
- Very sensitive to parameter change:
  - Over a person-year difference in a 10 KLOC project with minor adjustments
- Broad brush model that can generate significant errors

Drawbacks of COCOMO 81

- COCOMO 81 was developed with the assumption:
  - Waterfall process would be used and that all software would be developed from scratch.

- Since its formulation, there have been many changes in software engineering practices:
  - Software reuse
  - Application generation of programs
  - Object oriented approaches
  - Need for rapid development
    - Agile models

COCOMO 2 (1995)

- Main objectives of COCOMO 2:
  - To develop a software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's
  - To develop software cost database and tool support capabilities for continuous model improvement

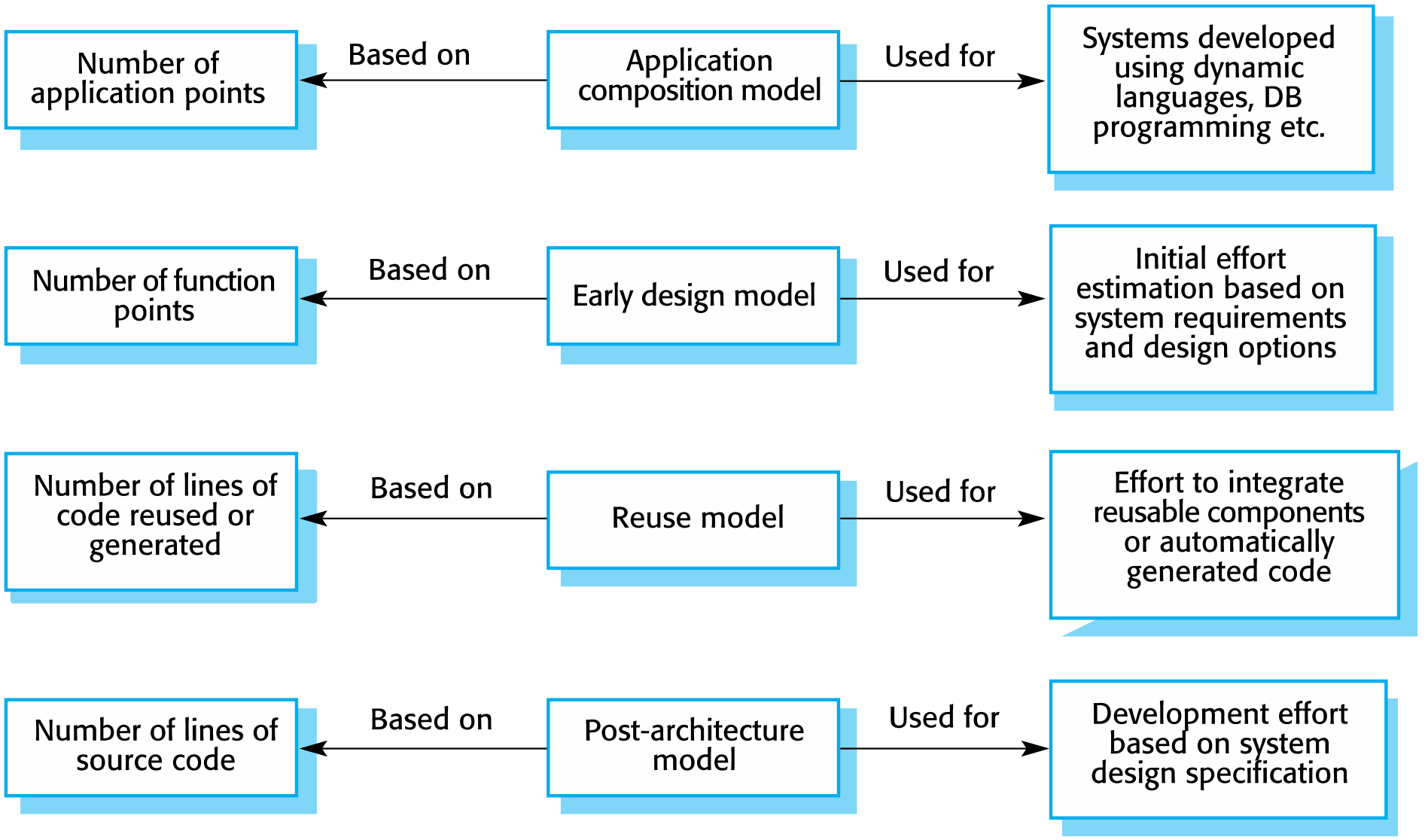
- **From "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," *Annals of Software Engineering*, (1995).**

COCOMO 2 model

- COCOMO 2 incorporates a range of sub-models:
  - Produces increasingly accurate estimates.

- The 4 sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not yet started.
  - Reuse model. Used to compute the effort of integrating reusable components.
  - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

COCOMO 2 model



COCOMO 2 model

COCOMO 81 DSI - “Delivered Source Instructions”
COCOMO II SLOC - “Source Lines Of Code”

- The original COCOMO 81 model was defined in terms of Delivered Source Instructions, which are very similar to SLOC.
- The major difference between DSI and SLOC is that a single Source Line of Code may be several physical lines.
- For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI.

COCOMO 2 model

- The core model is:
  $\text{Effort (pm)} = A \times (\text{size})^{(sf)} \times (em_1) \times (em_2) \times (em_3) \ldots$
  where,
- pm = person months,
- A = 2.94,
  - size is the number of thousands of source lines of code,
  - sf is the scale factor
  - em is an effort multiplier

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.

Staffing level estimation requirements

- Staff required can't be computed by diving the development time by the required schedule.

- Number of personnel required during any development project:
  - not constant.
- 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.

- The more people who work on the project, the more total effort is usually required.

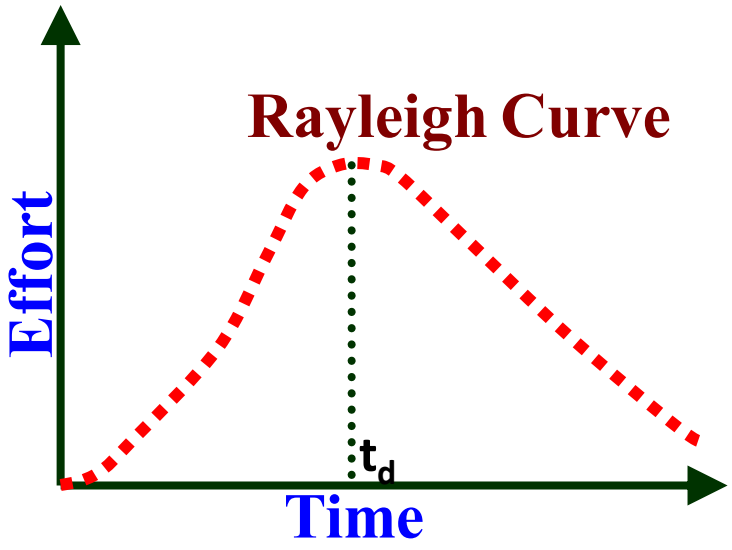
Staffing Level Estimation

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

- Rayleigh curve is specified by two parameters:
  - t_d : the time at which the curve reaches its maximum
  - K : the total area under the curve.

Norden represented the Rayleigh curve by the following equation

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



Putnam's Work:

- In 1978, 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 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.

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 total effort expended in PM
  - L is the size in KLOC.
  - t_d is the time to develop the software.
  - C_k is the state of technology coefficient
    - reflects factors that affect programmer productivity.

Putnam's Work (CONT.):

- 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

Effect of Schedule Change on Cost

- Using the Putnam's expression for L,

$$K = \frac{L^3}{\left(C_K^3 t_d^4\right)}$$

Or,
$$K = \frac{C}{\left(t_d^4\right)}$$

- For the same product size L^3/C_k^3 is a constant C.

- Ratio of the effort required for the aame project with different time schedule

$$\frac{K_1}{K_2} = \frac{{t_{d_2}}^4}{{t_{d_1}}^4}$$

Effect of Schedule Change on Cost

- If the estimated development time is 1 year as per COCOMO model, 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.

Putnam's Model

Example:

given L =100,000

C1=10,040

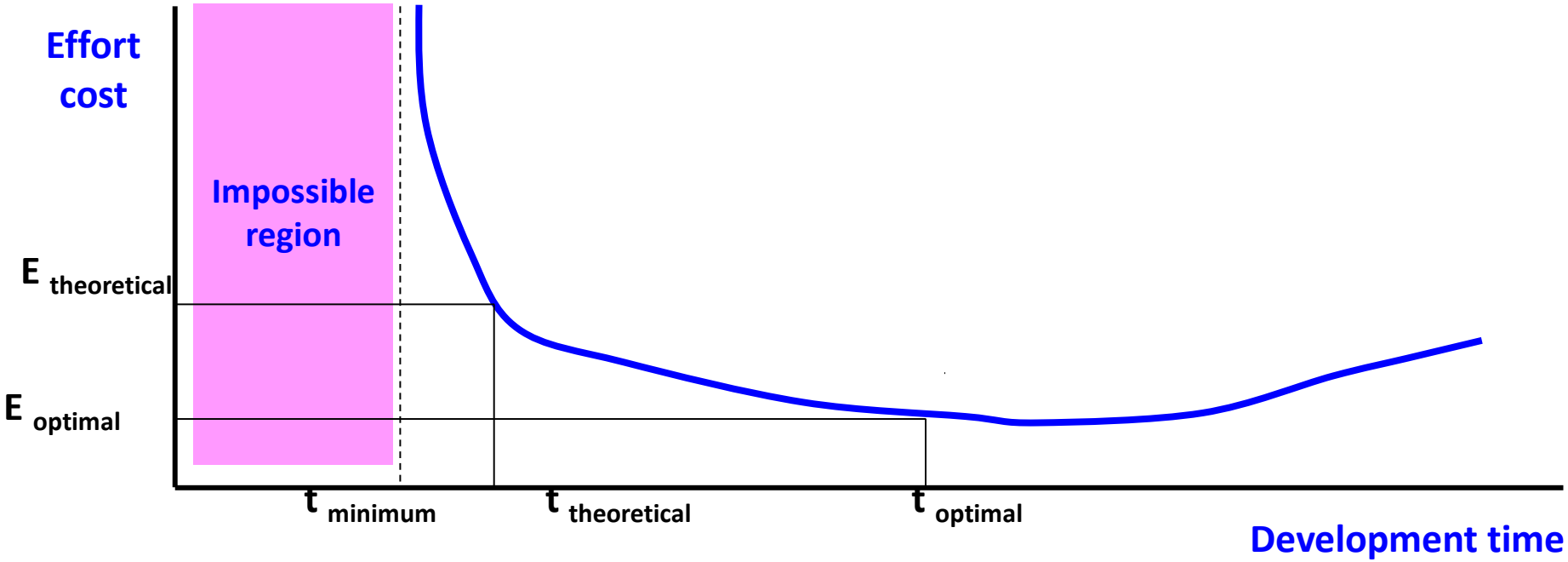
t_d = 12 ,18, 24months

compute K

t_d	K
12	988 person-month
18	195 person-month
24	62 person-month

Effort Applied vs. Delivery Time

- There is a nonlinear relationship between effort applied and delivery time (Putnam-Norden-Rayleigh Curve)
  - **Effort increases rapidly as the delivery time is reduced**



Putnam's Work (CONT.):

- 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 (CONT.)

- Boehm [1981] 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.

- 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 proposed the equation:

$$L = C_{te} t_d K^{1/2}$$

  - 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.

  - Thus, the ratio is: $\frac{K_1}{K_2} = \frac{{t_{d_2}}^2}{{t_{d_1}}^2}$

Project Scheduling

(Scheduling of activity)

Basics

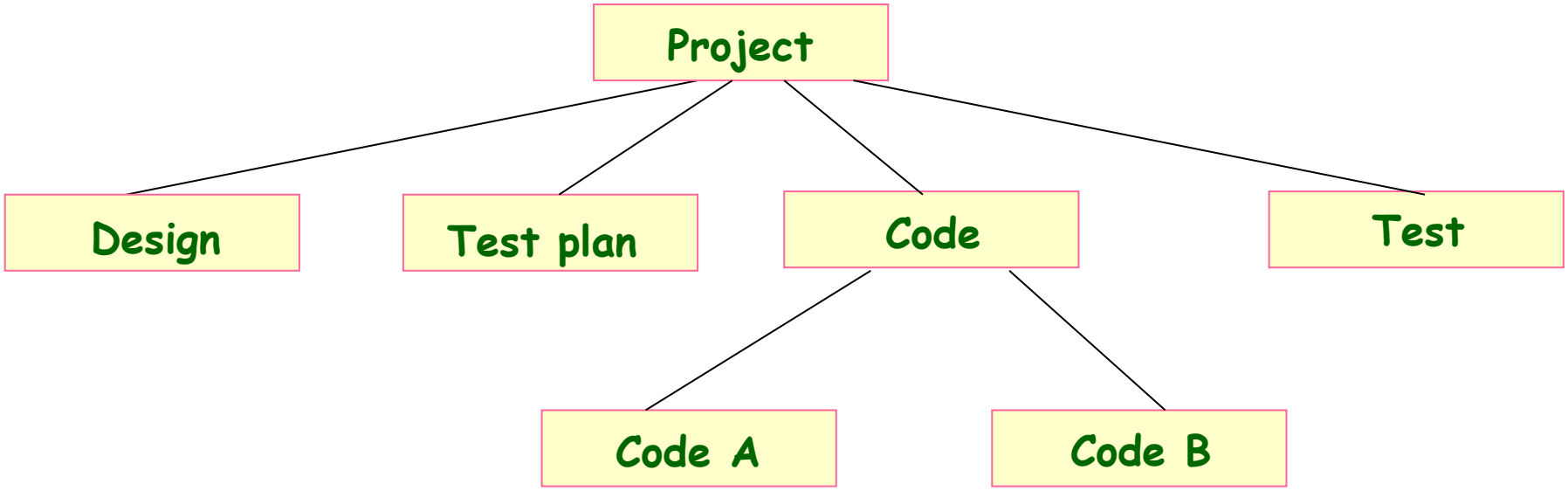
- Schedule converts action plan into an operating time table…
- Basis for monitoring and controlling projects…
- "Scheduling more important in projects than in production jobs."
- **Let us now discuss a few points:**
  - **Every project manager must remember these during project scheduling.**

Project scheduling activities

- Determining Major Activities
- Breakdown the activities into tasks
- Identifying precedence relationships among tasks
- Determining activity times (activity network): duration, start and end
- Determining critical path
- Determining available resources and allocating resources

Work breakdown structure (WBS)

- Hierarchical decomposition of a project into subtasks
  - Shows how tasks are decomposed into subtasks
  - Does not show duration
  - Does not show precedence relations (e.g. task A must be finished before task B can start)



How long to decompose?

- The decomposition of the activities is carried out until any of the following is satisfied:
  - A leaf-level subactivity (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

- Decomposition at too coarse level vs too fine level
  - When the granularity of the tasks is several months, by the time a problem (schedule delay) is noticed and corrective actions are initiated, it may be too late for the project to recover.
  - On the other hand, if the tasks are decomposed into very small granularity (one or two days each), then the milestones get too closely spaced. This would require frequent monitoring of the project status and entail frequent revisions to the plan document.

A Task Network

- Also called an **Activity Network**

- Depict task flow for a project
  - Task length, sequence, concurrency, and dependency
  - Represent inter-task dependencies

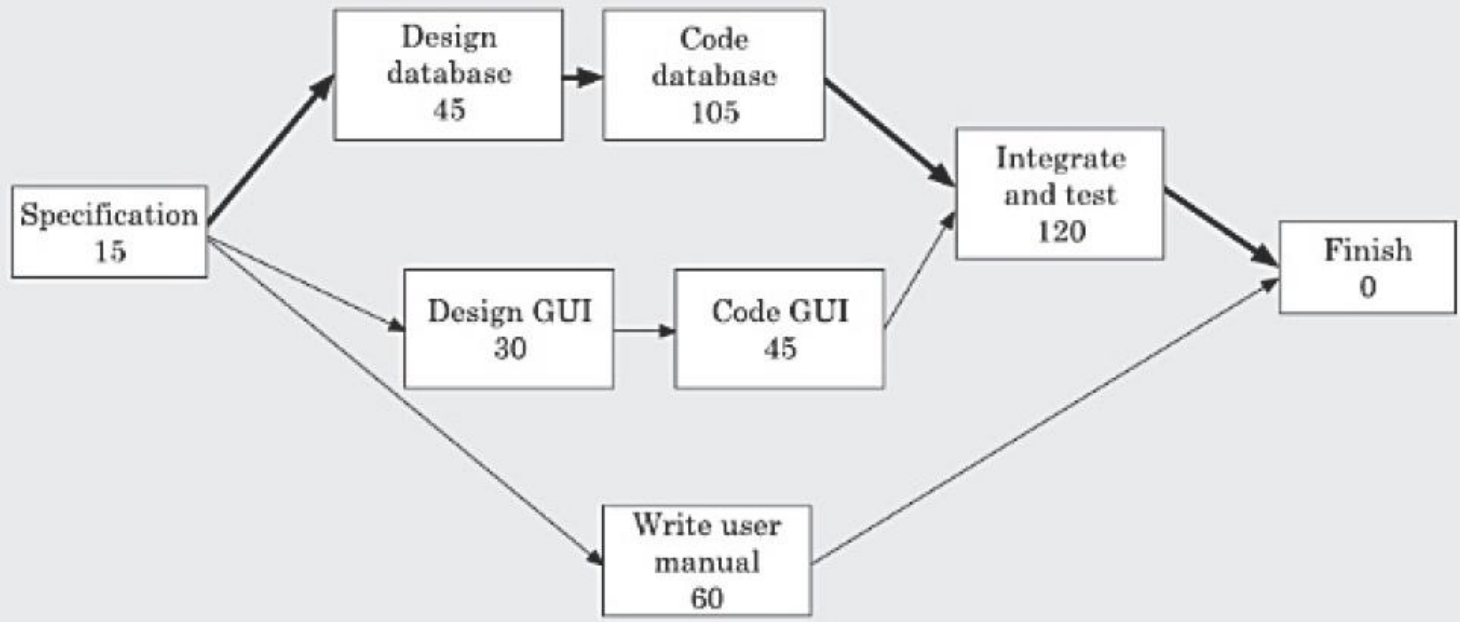
Activity Networks

- Activity on Node (AON)
  - 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 on Edge (AOE)
  - Tasks are associated with the edges.
  - The edges are also annotated with the task duration.
  - The nodes in the graph represent project milestones

Activity network example

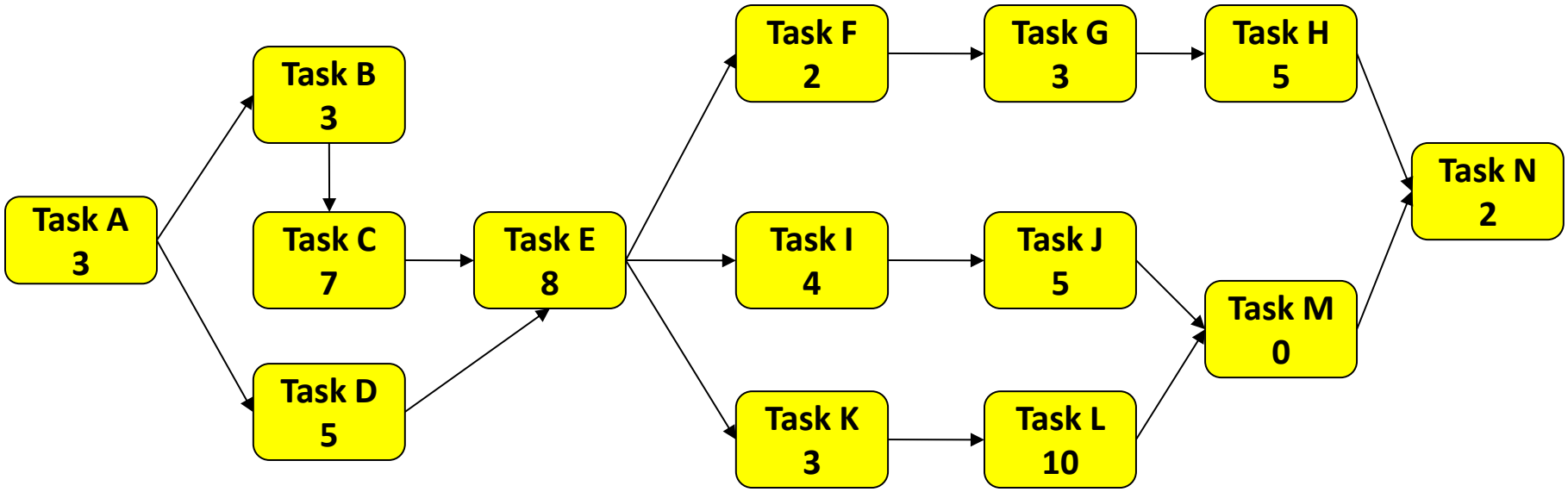
Task Number	Task	Duration	Dependent on Tasks
T1	Specification	15	–
T2	Design database	45	T 1
T3	Design GUI	30	T 1
T4	Code database	105	T 2
T5	Code GUI part	45	T 3
T6	Integrate and test	120	T 4 and T 5
T7	Write user manual	60	T 1



Critical path

- **A single path leading from start to finish in a task network**
  - Contains the sequence of tasks that must be completed on schedule, and which together take the longest time to complete
  - Also determines the minimum duration of the project

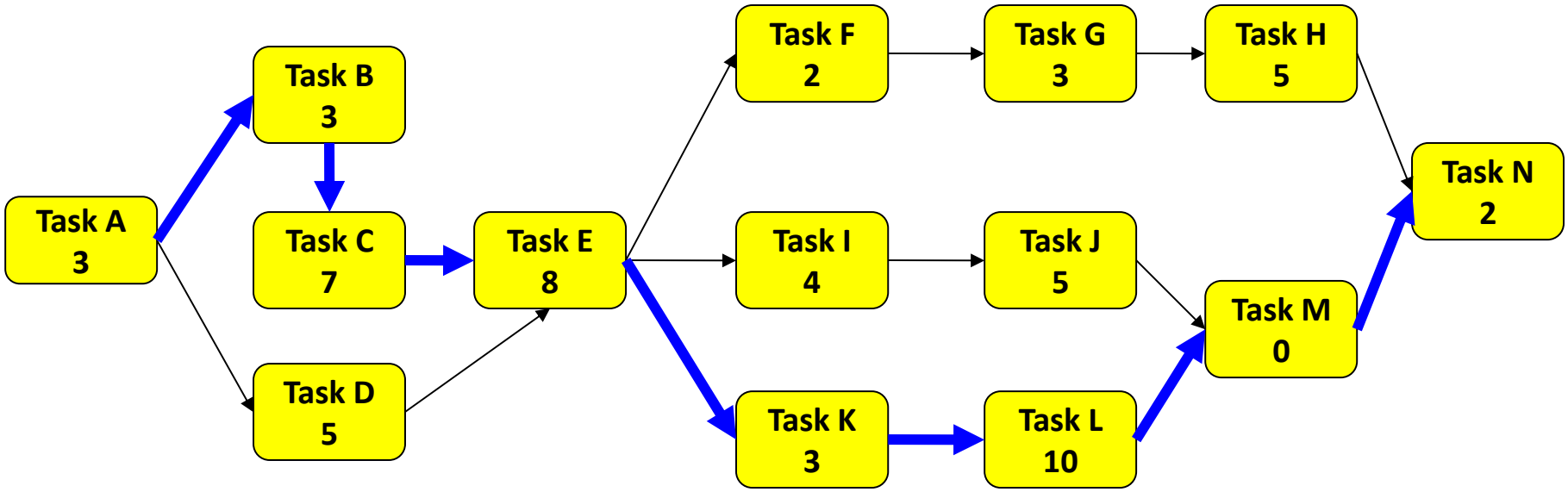
Example Task Network



Where is the critical path and what tasks are on it?

Is it possible to have two critical paths?

Example with Critical Path Marked



Critical path: A-B-C-E-K-L-M-N

PERT & CPM

- Both task Network-based techniques
- Developed in 1950's
  - CPM by DuPont for chemical plants
  - PERT by U.S. Navy for Polaris missile
- Consider precedence relationships & interdependencies
- Each uses a different estimate of activity times
- Graphically display the precedence relationships & sequence of activities
- Estimate the project's duration
- Identify **critical activities**:
  - Activities that cannot be delayed without delaying the project
- Estimate the amount of slack associated with non-critical activities

Critical Path Method (CPM)

- CPM is an algorithmic approach to determine the critical paths and slack times for tasks not on the critical paths.

- Identifies critical path
  - *Longest* path in network
  - *Shortest* time project can be completed
  - Any delay on activities delays project
  - Activities have 0 slack

Critical Path Method (CPM)

- A path in the activity network (graph) is any set of consecutive nodes and edges 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.
- A critical task is one with a zero slack time.
- A path from the start node to the finish node containing only critical tasks is called a critical path.
- There can be more than one critical path in a project.

- A critical path is a:
  - path along which every milestone is critical to meeting the project deadline
  - chain of tasks that determine the duration of the project.

Critical Path Method (CPM)

- **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 is the 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.

- **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.

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.

- **Latest finish (LF):** LF indicates the latest time by which a task can finish without affecting the final completion time of the project. A task completing beyond its LF would cause project delay. LF of a task can be obtained by subtracting maximum of all paths from this task to finish from MT.

- **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.

Earliest start & earliest finish time

- We are interested in the critical path = longest path through the network.
- Starting at the network's origin (node 1) and using a starting time of 0,
  - Compute an **earliest start** (ES) and **earliest finish** (EF) time for each activity in the network.
- The expression EF = ES + t can be used to find the earliest finish time for a given activity.

  For example, for activity A, ES = 0 and t = 5; thus the earliest finish time for activity A is

$$EF = 0 + 5 = 5$$

CPM Steps

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

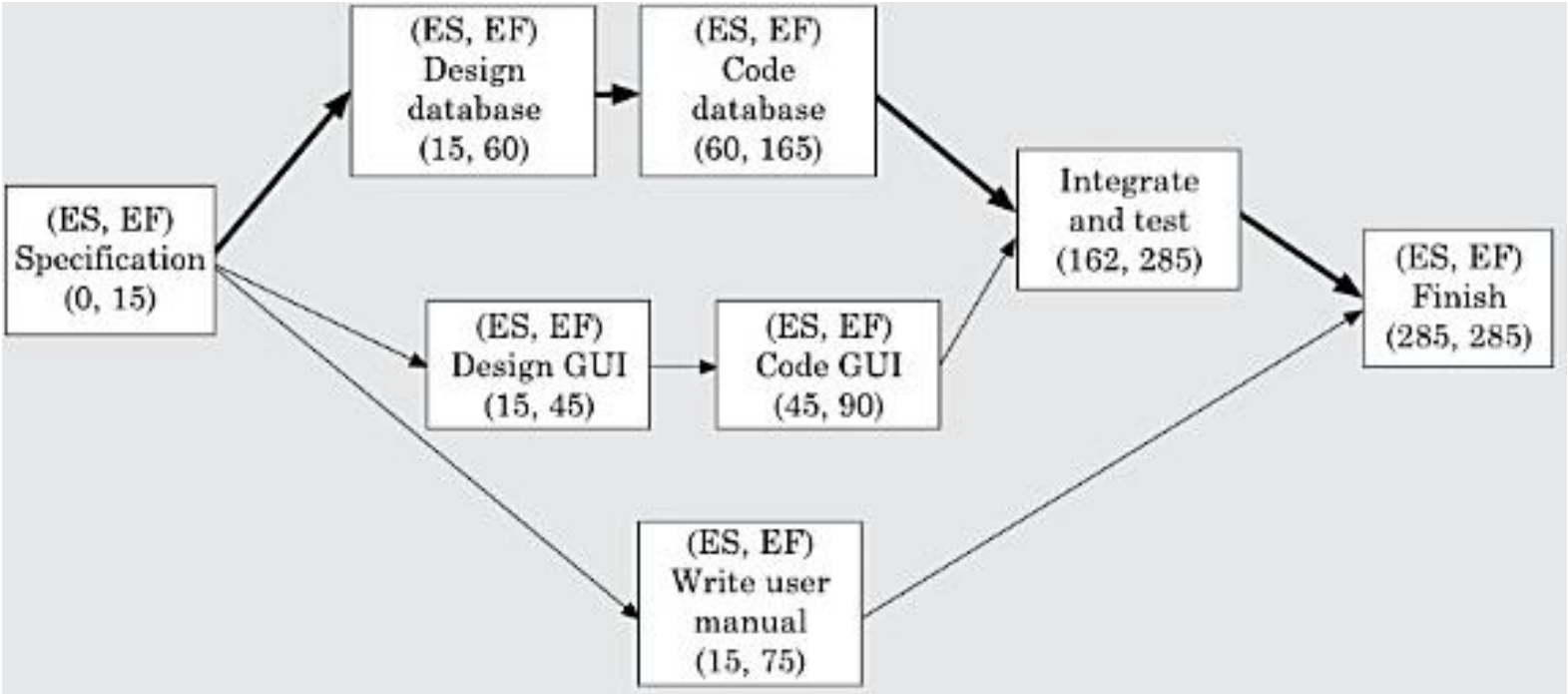
2. Compute LS and LF for each task. Use the rule: LF is equal to the smallest LS of the immediate successors

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

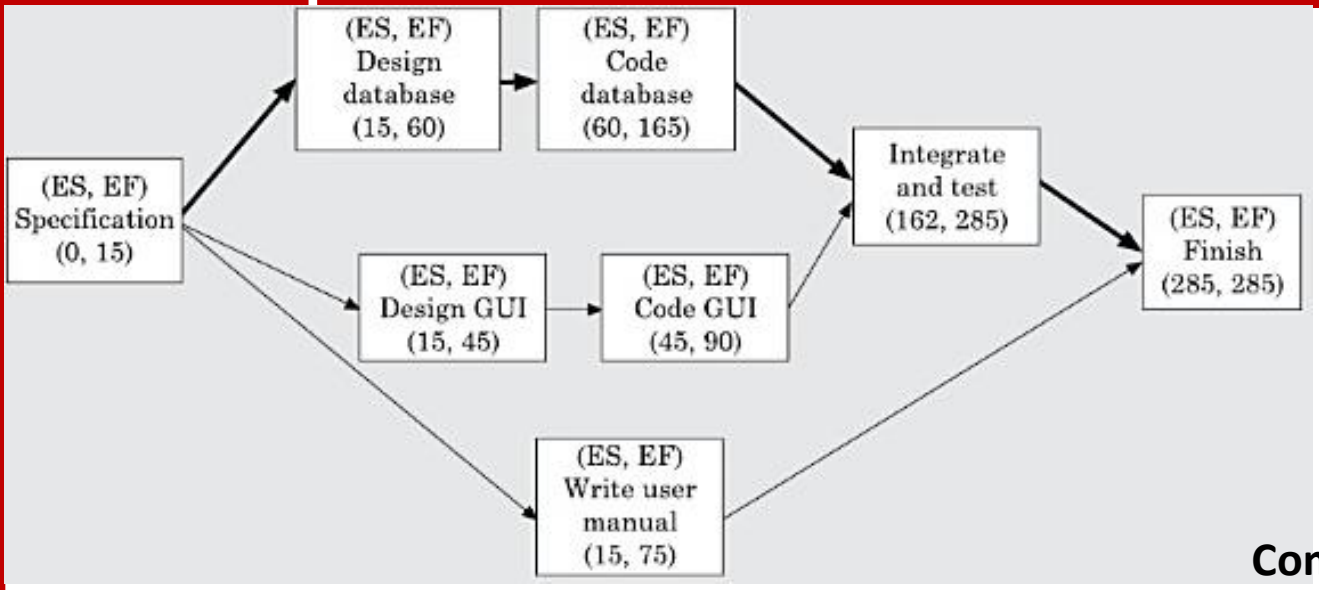
Example: CPM

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

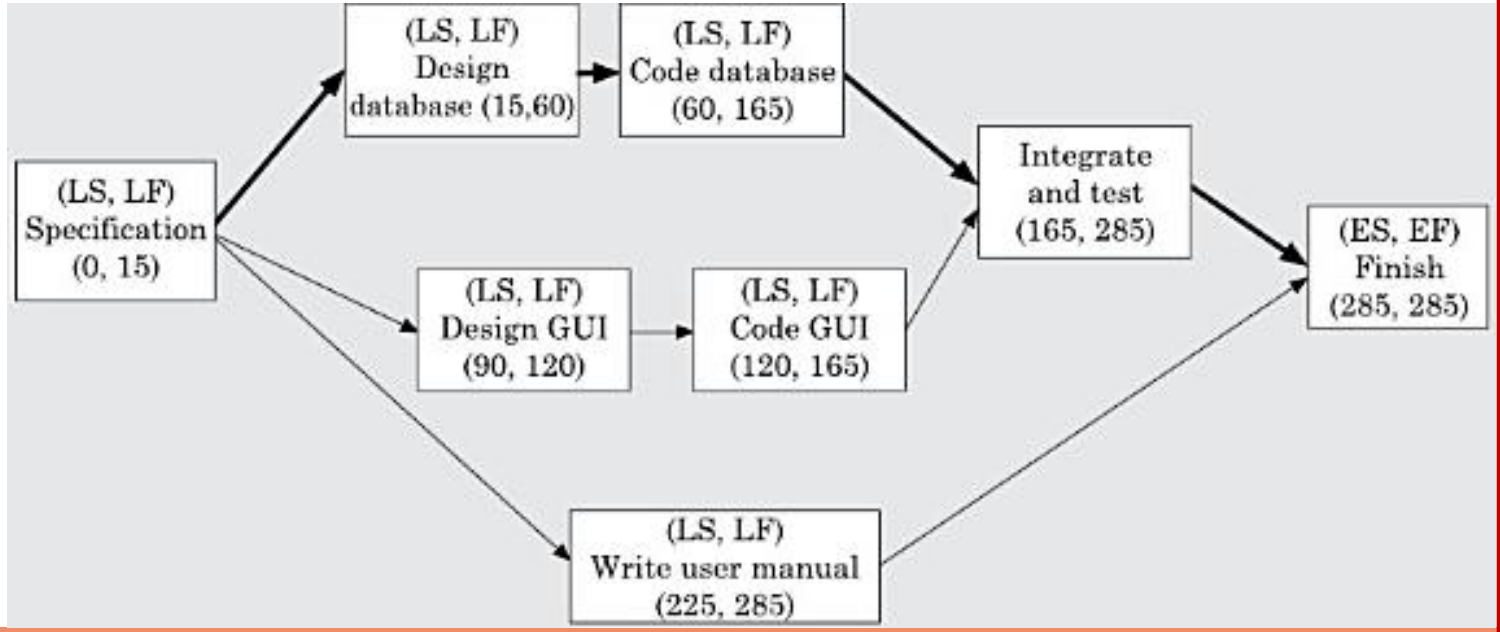
Compute ES and EF for each task



Example: CPM



Compute LS and LF for each task



Example: CPM

- Compute ST for each task

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

Formulating a Network Model

The first stage in creating a network model is to represent the activities and their interrelationships as a graph". In activity-on-node, we do this by representing activities as nodes (boxes) in the graph. The edges between nodes represent dependencies.

Some rules for constructing precedence networks

- A project network should have only one start node
- A project network should have only one end node
- A node has some duration
- Links normally have no duration
- Precedents are the immediate preceding activities
- Time moves from left to right
- A network may not contain loops
- A network should not contain dangles

Example

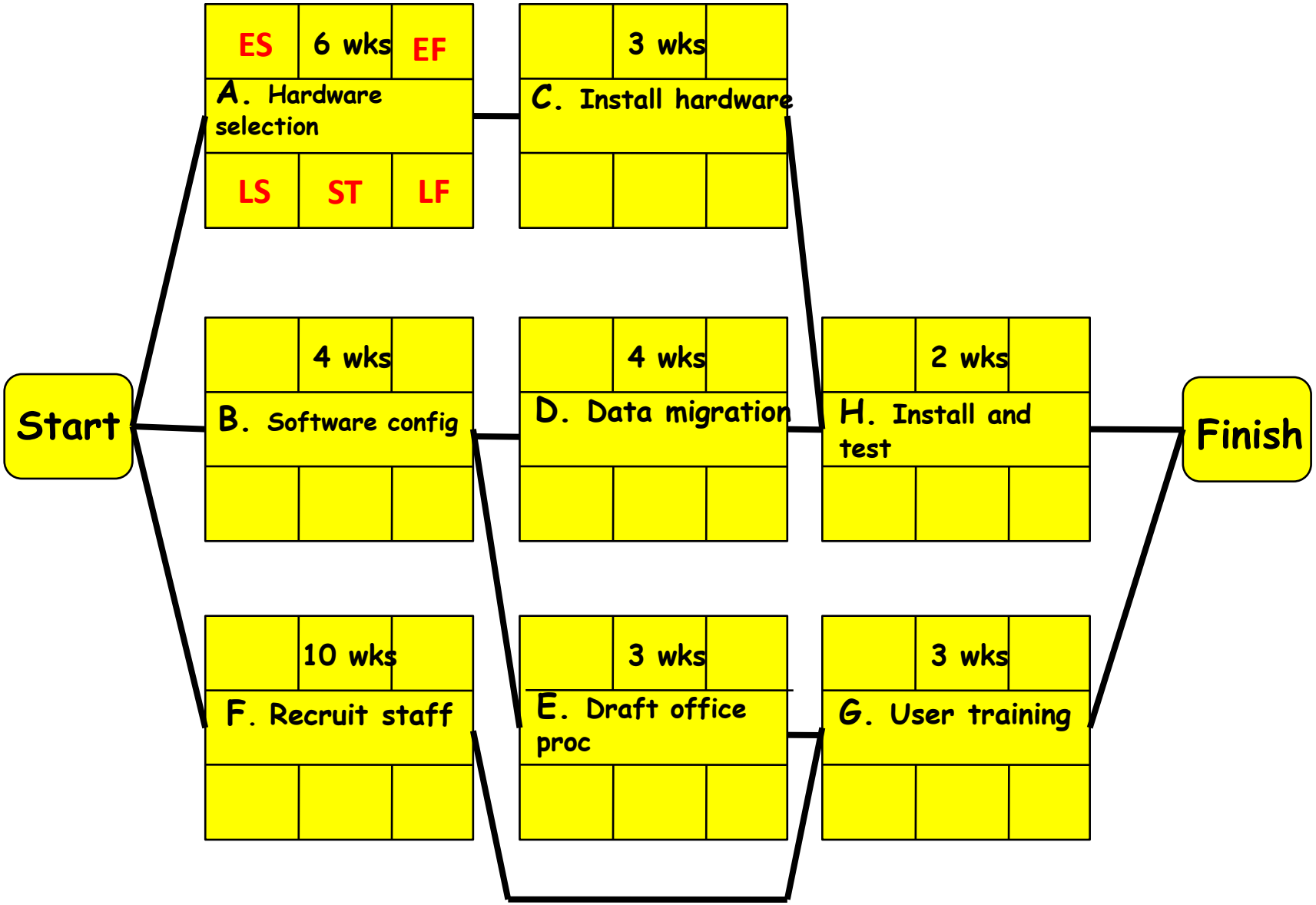
Find the critical activities and the critical path for the following example.

Activity Label	Activity Name	Duration (Weeks)	Precedence
A	Hardware Selection	6	---
B	System configuration	4	---
C	Install hardware	3	A
D	Data migration	4	B
E	Draft office procedures	3	B
F	Recruit staff	10	---
G	User training	3	E,F
H	Install and test	2	C,D

Steps

- First Draw Task Network
- Compute ES, EF, LS and LF
- Compute the slack time (float time) for each activity
- Identify the critical activities, i.e. activities for which slack time is zero.
- Identify the critical path, i.e. the which contains only the critical activities.
- There may be more than one critical path.

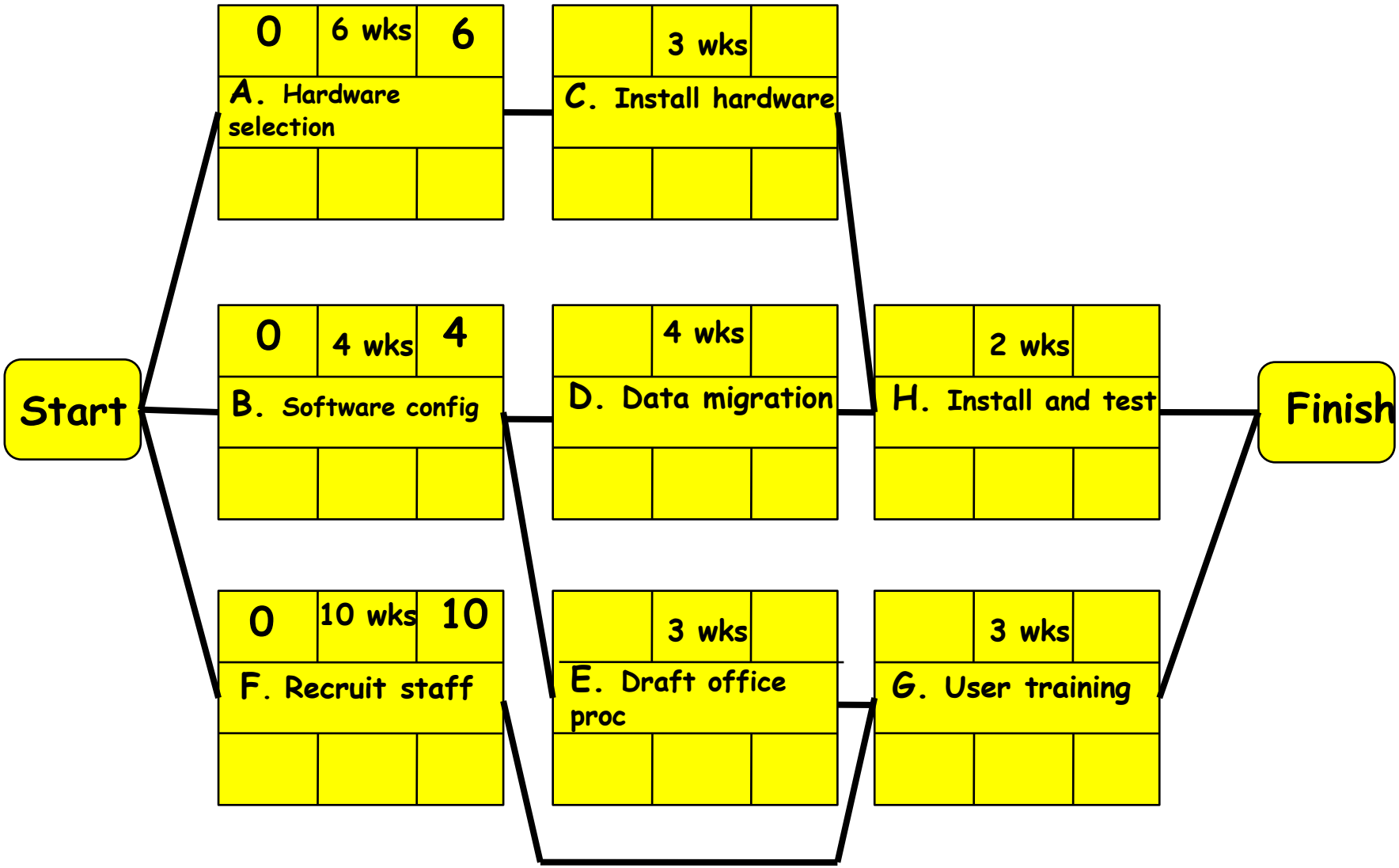
Activity Network



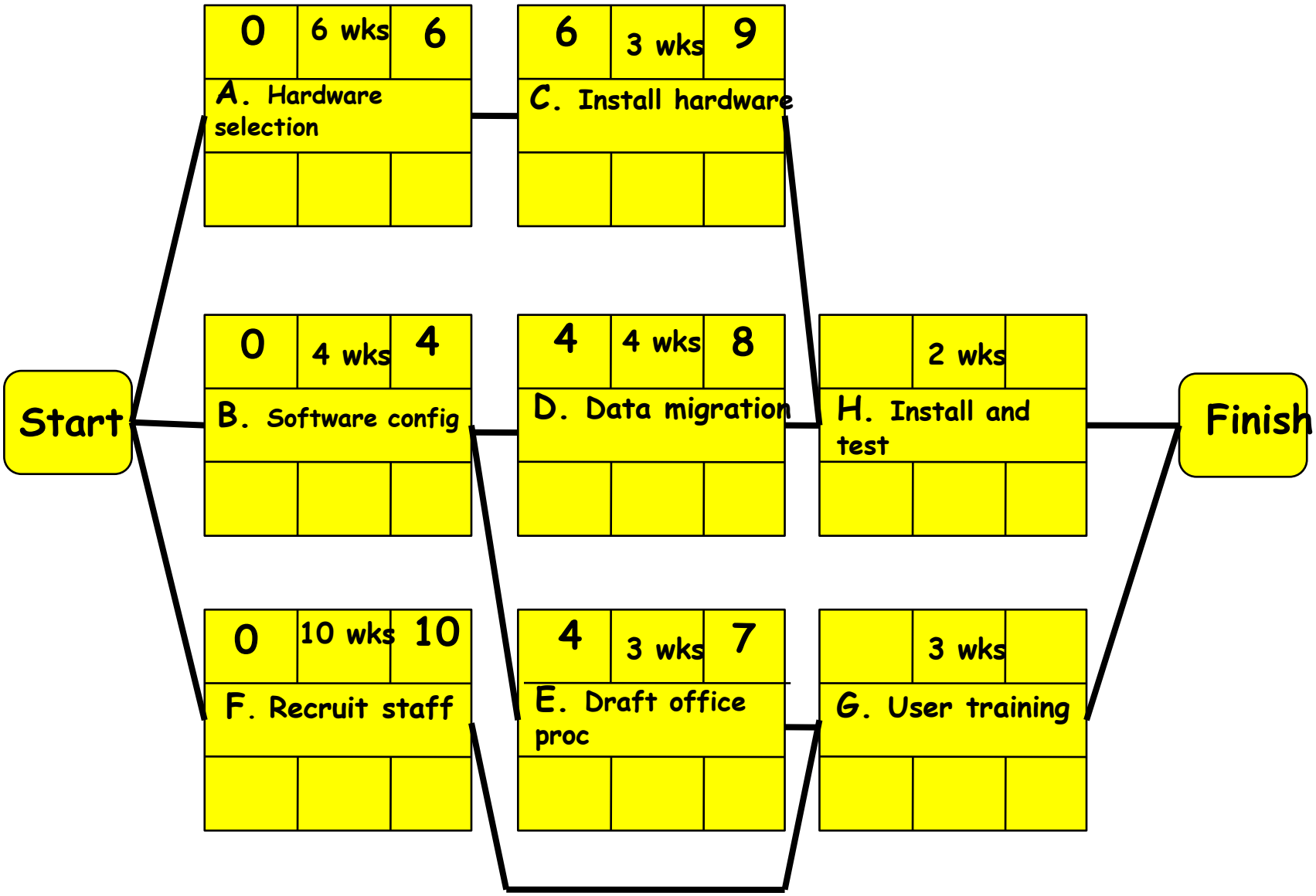
Forward Pass

- The forward pass is carried out to calculate the earliest dates on which each activity may be started and completed.
- Rule:
  - The earliest start date for an activity is the earliest finish date for the preceding activity.
  - Where there is more than one immediately preceding activity we take the latest of the earliest finish dates for those activities.

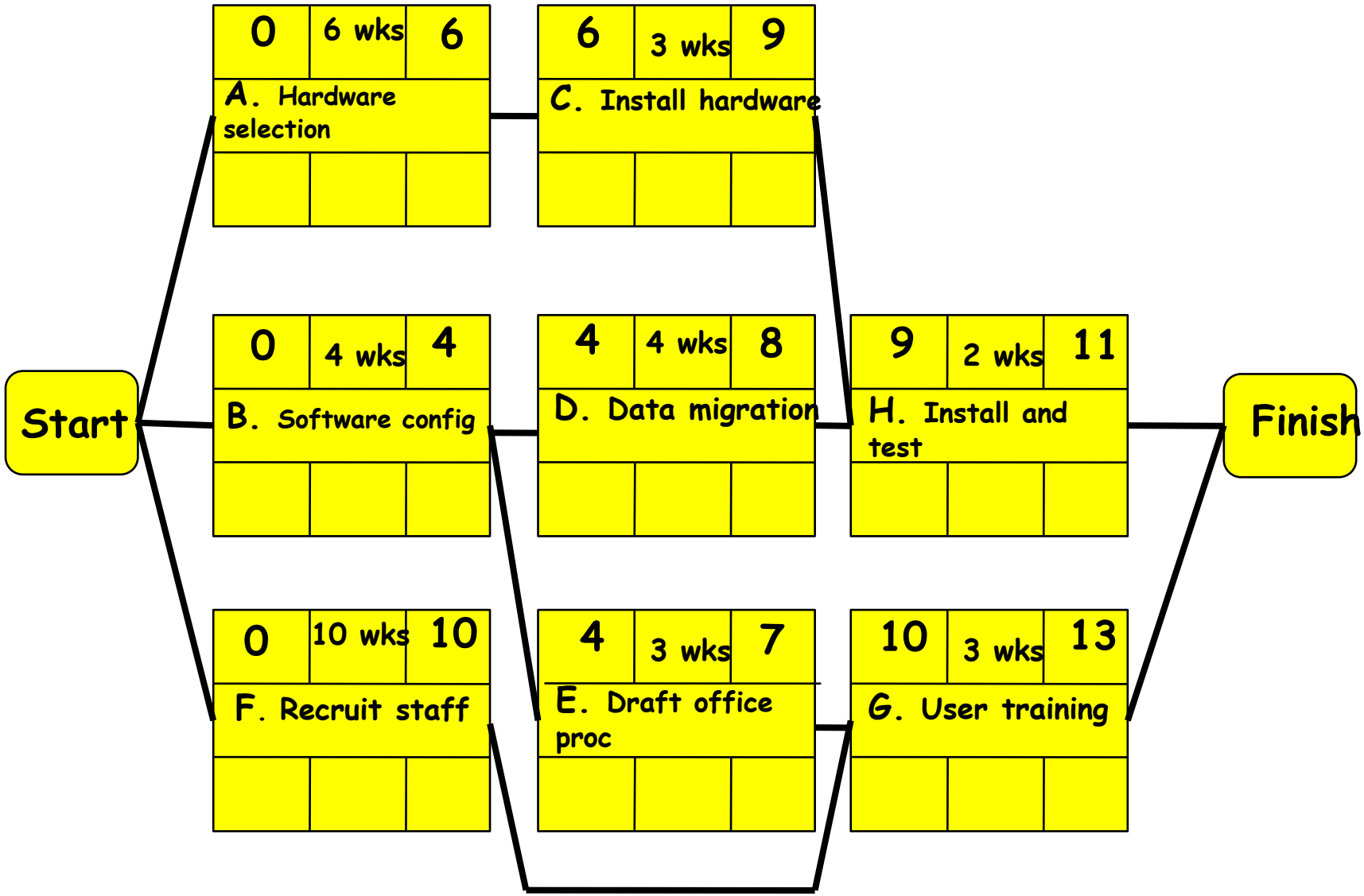
After Forward Pass Step 1



After Forward Pass Step 2



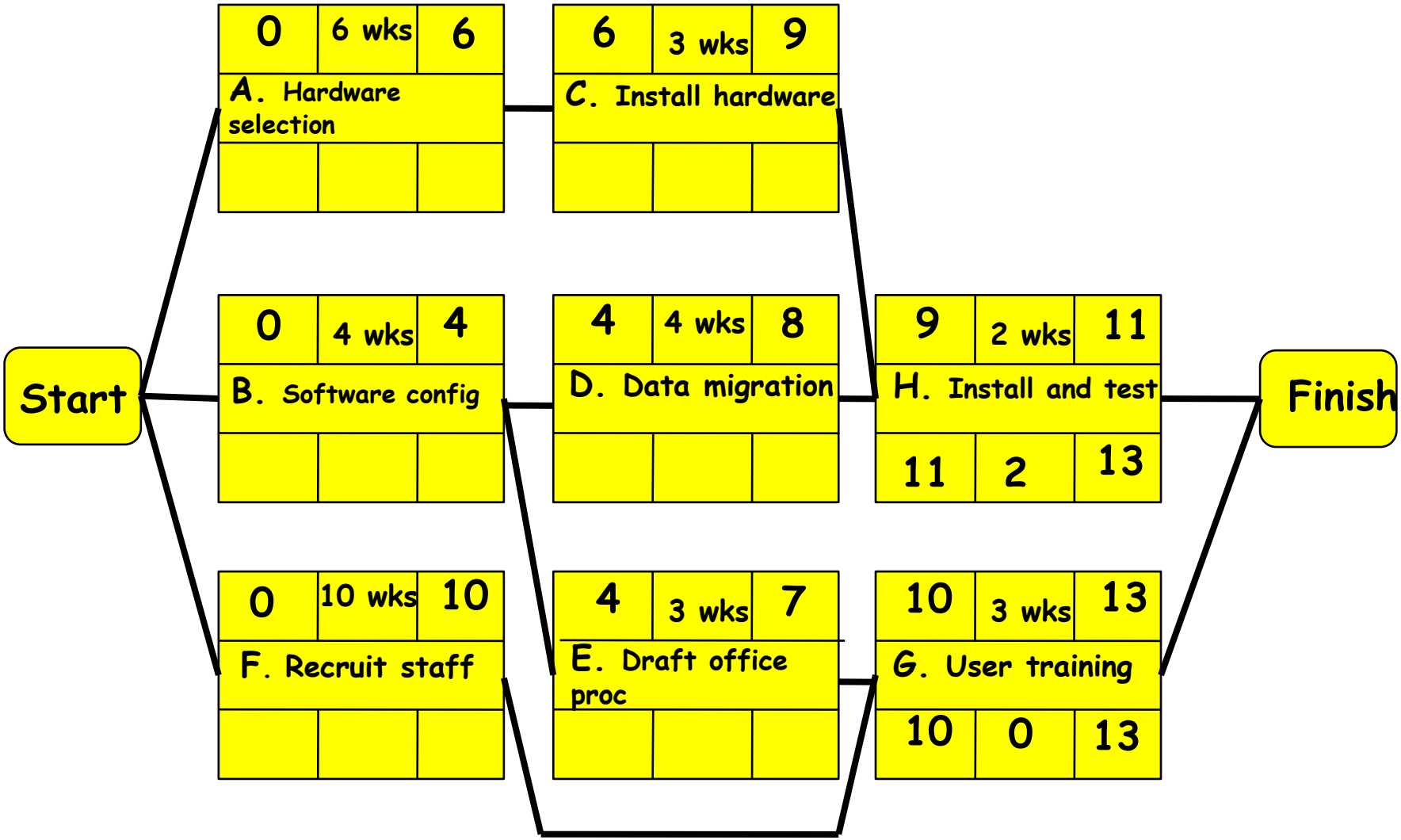
After Forward Pass Step 3 (Final)



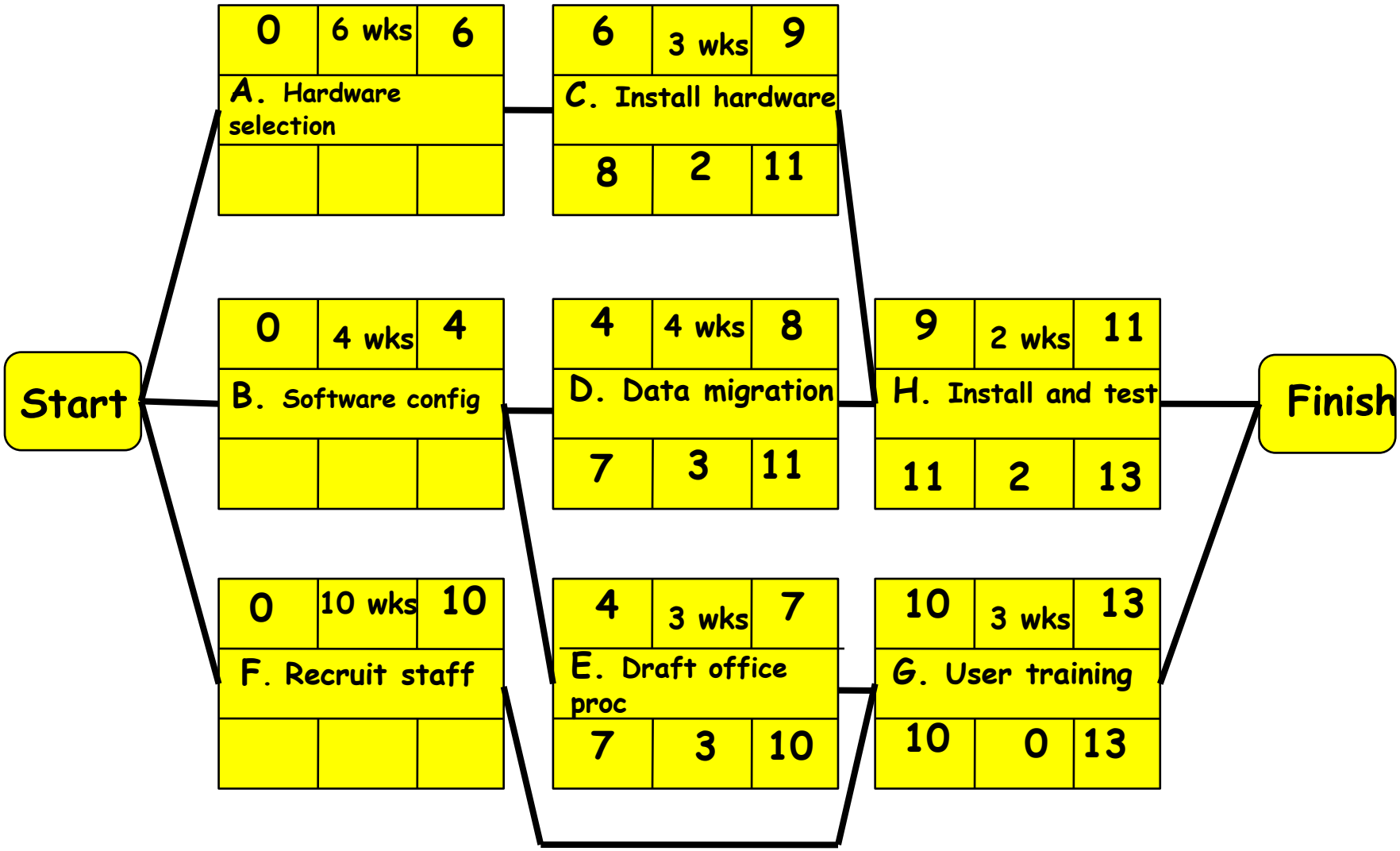
The Backward Pass

- The backward pass is carried out to calculate the latest start date at which each activity may be started and finished without delaying the end date of the project.
- Rule
  - The latest finish date for an activity is the latest start date for the activity that commences immediately that activity is complete.
  - Where more than one activity can commence we take the earliest of the latest start dates for those activities.

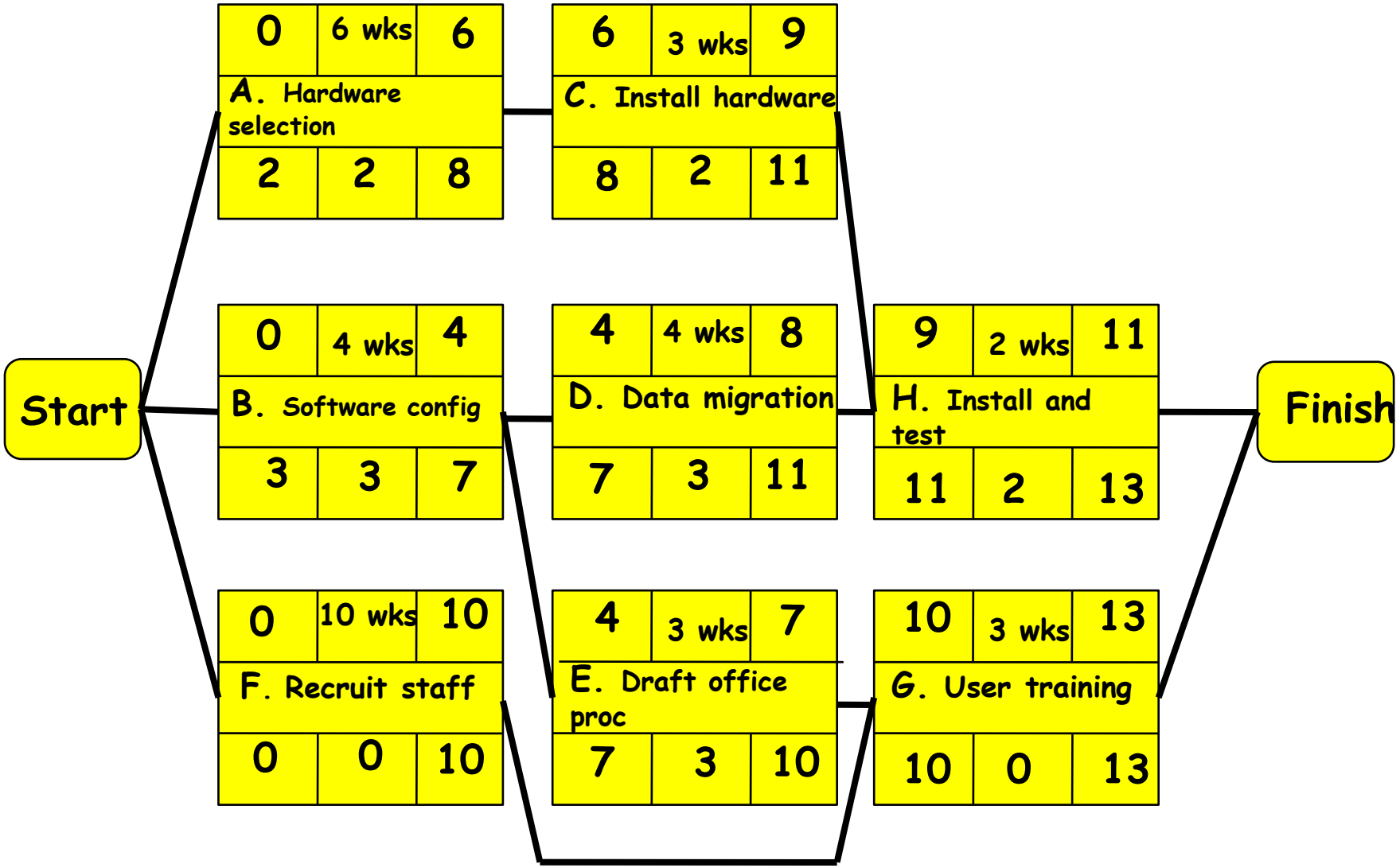
After Backward Pass Step 1



After Backward Pass Step 2



After Backward Pass Step 3 (Final)



Critical Path

Critical Activities – F, G

PERT (Program Evaluation and Review Technique)

- 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

- A network (graph) where the nodes represent tasks and arrows describe precedence relations
  - Used successfully in management of Polaris missile project in 50's
  - Shows probabilistic task durations (on the task node)

- Multiple time estimates were used for each task:
  - Allowed for variations due to uncertainty in determined activity times
  - **Activity times are assumed to be random, with certain assumed probability distribution**

Probabilistic Time Estimates

- PERT-type approach uses 3 time estimates for each activity

- **Optimistic time (O)**
  - shortest possible time (ideally)

- **Most likely time (M)**
  - subjective estimate of most frequent time

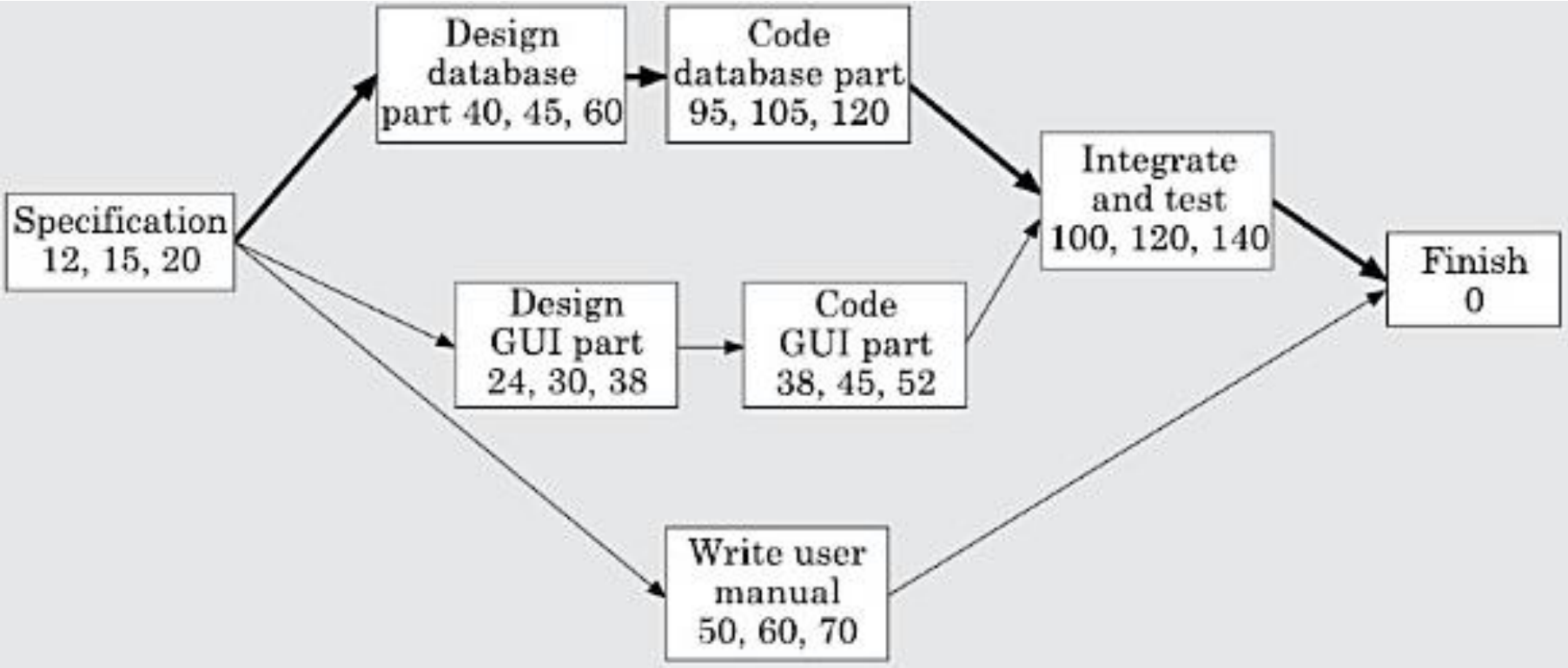
- **Pessimistic/Worst time (P/W)**
  - longest time possible if everything went wrong

- The entire distribution lies between the interval (M – 3xST) and (M + 3xST), where ST is the standard deviation
- Thus, the standard deviation for a task is ***ST=(W-O)/6***.
- The mean estimated time t_e***=(O+4M+W)/6***.

PERT

- 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.
- PERT was developed to take account of the uncertainty surrounding estimates of task durations.

Example: PERT



Example

Activity Label	Activity Name
A	Hardware Selection
B	System configuration
C	Install hardware
D	Data migration
E	Draft office procedures
F	Recruit staff
G	User training
H	Install and test

Example cont…

Activity Label	Optimistic (O)	Activity duration (weeks), Most Likely (m)	Pessimistic (W)	Precedents
A	5	6	8	---
B	3	4	5	---
C	2	3	3	A
D	3.5	4	5	B
E	1	3	4	B
F	8	10	15	---
G	2	3	4	E,F
H	2	2	2.5	C,D

Expected Time Estimates (t_e)

Activity	Activity duration (weeks)			
	Optimistic (O)	Most Likely (M)	Pessimistic (W)	Expected t_e=(O+4M+W)/6
A	5	6	8	(5+4*6+8)/6= 6.17
B	3	4	5	(3+4*4+5)/6= 4
C	2	3	3	(2+4*3+3)/6= 2.83
D	3.5	4	5	(3.5+4*4+5)/6= 4.08
E	1	3	4	(1+4*3+4)/6= 2.83
F	8	10	15	(8+4*10+15)/6= 10.5
G	2	3	4	(2+4*3+4)/6= 3
H	2	2	2.5	(2+4*2+2.5)/6= 2.08

Activity Deviation (s)

Activity	Optimistic (O)	Pessimistic (W)	Standard deviation s=(W-O)/6
A	5	8	(8-5)/6= 0.5
B	3	5	(5-3)/6= 0.33
C	2	3	(3-2)/6= 0.17
D	3.5	5	(5-3.5)/6= 0.25
E	1	4	(4-1)/6= 0.5
F	8	15	(15-8)/6= 1.17
G	2	4	(4-2)/6= 0.33
H	2	2.5	(2.5-2)/6= 0.08

Expected Time Estimates (t_e) & SD (s)

Activity	Activity duration (weeks)				
	Optimistic (O)	Most Likely (M)	Pessimistic (W)	Expected Time (t_e)	Standard Deviation (s)
A	5	6	8	6.17	(8-5)/6= 0.5
B	3	4	5	4	(5-3)/6= 0.33
C	2	3	3	2.83	(3-2)/6= 0.17
D	3.5	4	5	4.08	(5-3.5)/6= 0.25
E	1	3	4	2.83	(4-1)/6= 0.5
F	8	10	15	10.5	(15-8)/6= 1.17
G	2	3	4	3	(4-2)/6= 0.33
H	2	2	2.5	2.08	(2.5-2)/6= 0.08

Advantages of PERT

- PERT focuses attention on the uncertainty of forecasting
- The technique can be used to
  - calculate the standard deviation for each task and
  - rank them according to their degree of risk.
- Using this ranking, it can be seen that, for example, that
  - activity F has greatest uncertainty, whereas activity C should give us relatively little cause for concern.
- If we use the expected times and standard deviations for forward passes through the network,
- In particular, by setting target dates along the critical path,
  - we can focus on those activities posing the greatest risk to the project's schedule.

Benefits of CPM / PERT network

- Identifies activities with slacks :
  - These can be delayed for specified periods without penalty, or from which resources may be temporarily borrowed
- Determines the dates on which tasks may be started or must be started:
  - if the project is to stay in schedule.
- Shows which tasks must be coordinated to avoid resource or timing conflicts.
- Shows which tasks may run in parallel to meet project completion date
  - Shows interdependence of all tasks, work packages, and work units
  - Helps proper communications between departments and functions.
  - Determines expected project completion date.
  - Identifies critical activities, which can delay the project completion time.

GANTT charts

- Named after Henry Gantt (1861 - 1919) an American mathematical engineer.

GANTT charts

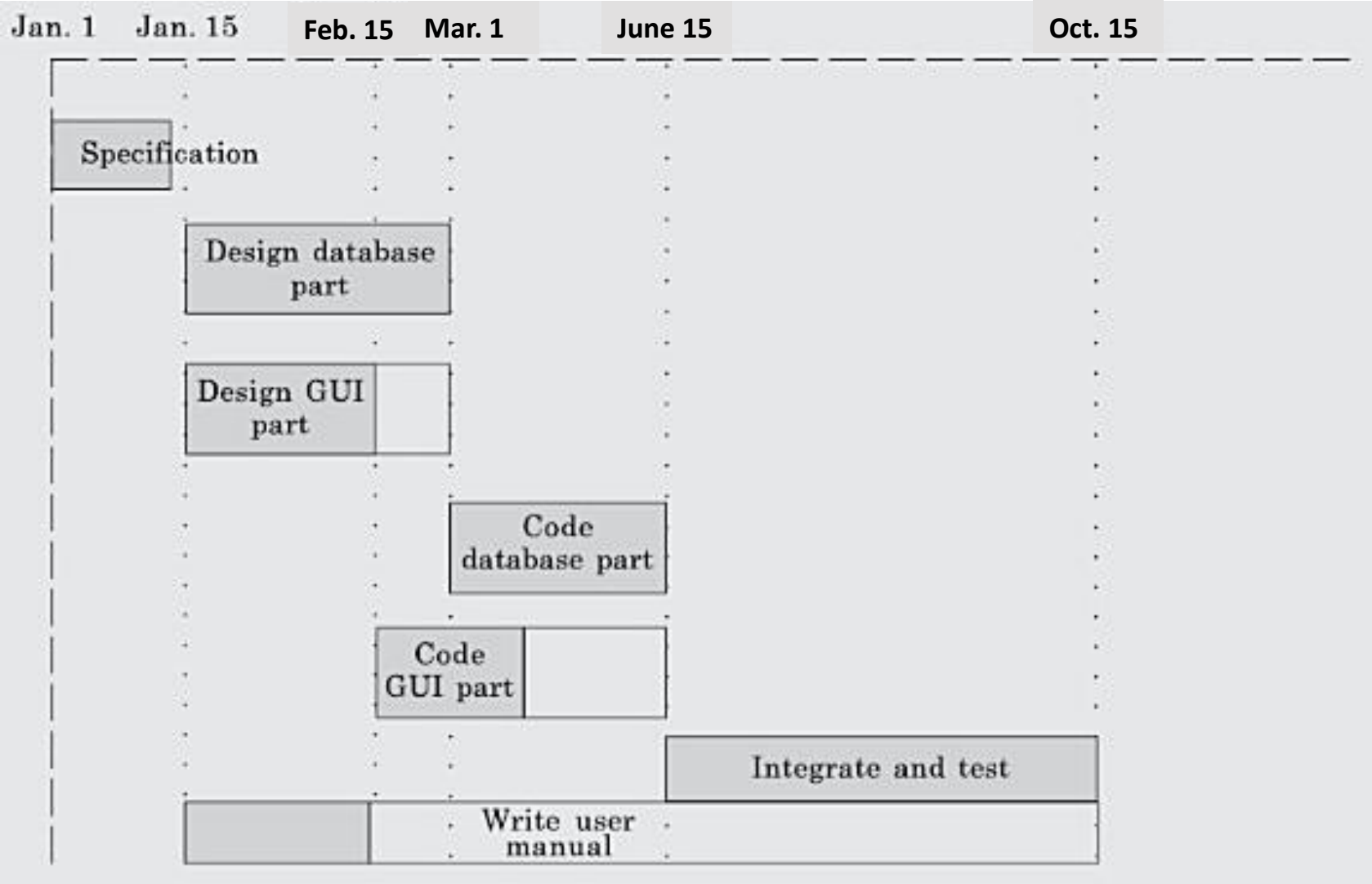
- A Gantt chart is a type of bar chart:
  - Captures start and finish dates of the activities

- Elements of a GANTT chart:
  - Task names
  - Start and finish dates of each tasks (graphically)
  - Dependency relationships
  - Task duration in an additional column
  - Name of the project worker responsible for the task or Resource specifications
  - other

Gantt Chart

- Characteristics:
  - The bar in each row identifies the corresponding task
  - The horizontal position of the bar identifies start and end times of the task
  - Bar length represents the duration of the task
  - Task durations can be compared easily
  - **Good for allocating resources and re-scheduling**
  - Precedence relationships can be represented using arrows
  - Critical activities are usually highlighted
  - Slack times are represented using bars with doted lines
  - Bar of each activity begins at the activity earliest start time (ES)
  - The bar of each activity ends at the activity latest finish time (LF).

Gantt Chart example



The unshaded part shows the slack time or lax time.

Gantt charts

- A graphical visualization of a *schedule*, where the time span for each activity is depicted by the length of a segment drawn on an adjacent calendar
  - **Generally does not show task decomposition**
  - Does not show duration, only the time span over which the task is scheduled
  - Can show activity of multiple developers in parallel
  - Makes it easy to monitor a project's progress and expenditures

Benefits of the GANTT chart

- Easy to read and comprehend
- Easy to create
- Identify the project network coupled with its schedule baseline
- Supports updating and project control
- Useful for resource planning

Limitations

- Can become quite unwieldy for projects with too many activities.
- Projects are often too complex for a Gantt chart.
- Gantt charts represent only a part of the project constraints.

Organisation structure and team structure

Organization Structure

- There are three broad ways in which a software development organisation can be structured:
  - Functional format
  - Project format
  - Matrix format

Functional format

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

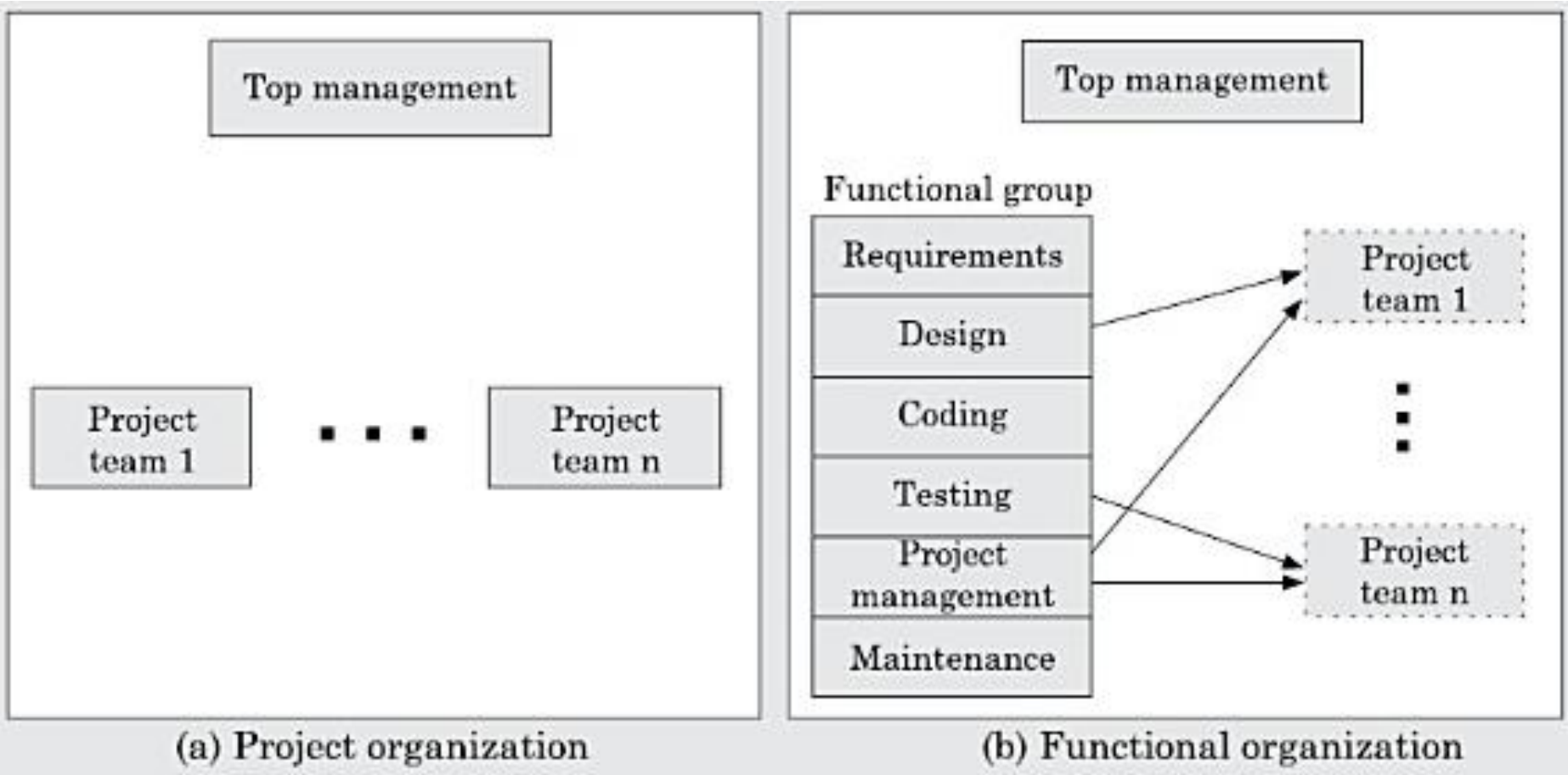
For obvious reasons the functional format is not suitable for small organisations handling just one or two projects

Project format

- 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.
    - That is, each team member takes on the role of the designer, co der, tester, etc during the course of the project.

Functional vs project format



(a) Project organization

(b) Functional organization

Matrix format

- The deployment of the different functional specialists in different projects can be represented in a matrix
- Therefore in a matrix organisation, the project manager needs to share responsibilities for the project with a number of individual functional managers.

	Project			
Functional group	#1	#2	#3	
#1	2	0	3	Functional manager 1
#2	0	5	3	Functional manager 2
#3	0	4	2	Functional manager 3
#4	1	4	0	Functional manager 4
#5	0	4	6	Functional manager 5
	Project manager 1	Project manager 2	Project manager 3	

- Two important problems that a matrix organisation often suffers from are:
  - Conflict between functional manager and project managers over allocation of workers.
  - Frequent shifting of workers in a firefighting mode

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

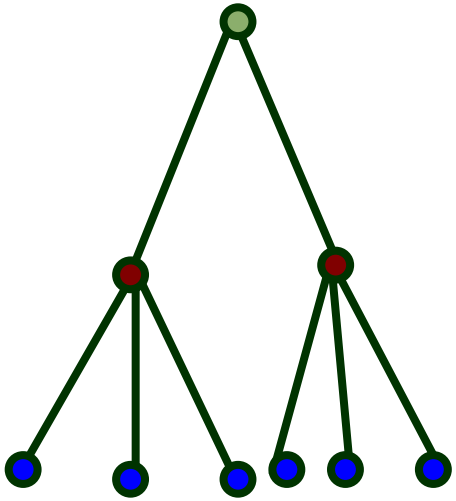
Chief Programmer Team

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

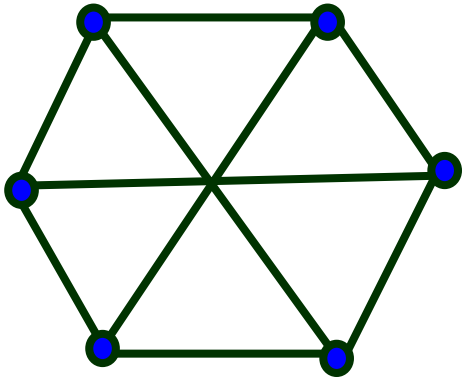
Mixed Control Team Organization

- 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.

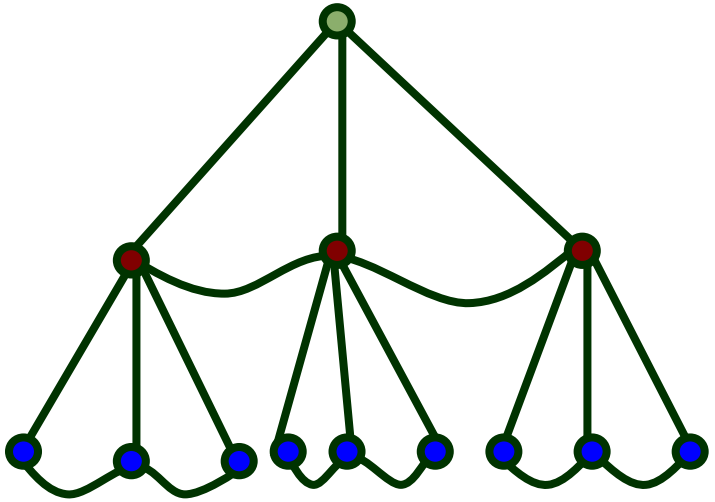
Team Organization



Chief Programmer team



Democratic Team



Mixed team organization

Risk Management

RISK MANAGEMENT

- **A risk is any anticipated unfavourable event or circumstance that can occur while a project is underway.**
- If a risk becomes real
  - it can adversely affect the project and hamper the successful and timely completion of the project.
- In this context, risk management aims at reducing the chances of a risk becoming real as well as reducing the impact of a risks that becomes real.
- Two risk management approaches:
  - Reactive approaches: This approach try to contain the adverse effect of the unfavorable event already occurred
  - Proactive approaches: Try to anticipate the risk the possible risks that the project is susceptible
- Risk management consists of three essential activities—
  - risk identification,
  - risk assessment, and
  - risk mitigation.

Risk Identification

- As soon as a risk is identified, effective risk management plans are made, so that the possible impacts of the risks is minimised
- There are three main categories of risks which can affect a software project
- **Project risks:**
  - budgetary, schedule, personnel, resource, and customer-related problems.
  - An important project risk is schedule slippage. The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule slippage.
- **Technical risks:**
  - Design, implementation, interfacing, testing, and maintenance problems.
  - 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.

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 as follows:

  p = r x s

Risk Mitigation

- After all the identified risks of a project have been assessed, plans are made to contain the most damaging and the most likely risks first.

- There are three main strategies for risk containment:

1. **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. The different categories of constraints that usually give rise to risks are:
   - Process-related risk: These risks arise due to aggressive work schedule, budget, and resource utilisation.
   - 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 (e.g., satellite communication).

- A few examples of risk avoidance can be the following: Discussing with the customer to change the requirements to reduce the scope of the work, giving incentives to the developers to avoid the risk of manpower turnover, etc.

Risk Mitigation

2. Transfer the risk: This strategy involves getting the risky components developed by a third party, buying insurance cover, etc.

3. Risk reduction: This involves planning ways to contain the damage due to a risk.

- 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.
- To choose the most appropriate strategy for handling a risk, the project manager must consider the cost of handling the risk and the corresponding reduction of risk.

Software configuration management

Software configuration management

- The configuration of the software is the state of all project deliverables at any point of time;
- software configuration management deals with effectively tracking and controlling the configuration of a software during its life cycle.

- **Software revision versus version:**
  - A new version of a software is created when there is significant change in functionality, technology, or the hardware it runs on, etc.
  - On the other hand, a new release is created if there is only a bug fix, minor enhancements to the functionality, usability, etc.

Necessity of Software Configuration Management

- **Inconsistency problem when the objects are replicated:** inconsistency in the local copy of the developer of the source code.

- **Problems associated with concurrent access:** Two developers may simultaneously carry out changes to different functions of the same module, and while saving overwrite each other.

- **Providing a stable development environment:** Suppose one developer is trying to integrate module A, with the modules B and C; since if developer of module C keeps changing C; this can be especially frustrating if a change to module C forces recompilation of the module.
- When an effective configuration management is in place, the manager freezes the objects to form a baseline.

Necessity of Software Configuration Management

- **System accounting and maintaining status information:** System accounting denotes keeping track of who made a particular change to an object and when the change was made.

- **Handling variants:** Suppose you have several variants of the same module, and find that a bug exists in one of them. Then, it should not be required to fix it in all versions and revisions.

Configuration Management Activities

- **Configuration identification:**
- Project managers normally classify the objects associated with a software development into three main categories
  - **Controlled:** Those objects that are already under configuration control. The team members must follow some formal procedures to change them.
  - **Precontrolled:** Precontrolled objects are not yet under configuration control, but will eventually be under configuration control.
  - **Uncontrolled:** Uncontrolled objects are not subject to configuration control. Controllable objects include both controlled and precontrolled objects.
- **Configuration control**
- Configuration control allows only authorised changes to the controlled objects to occur and prevents unauthorised changes.

- **End of Chapter**