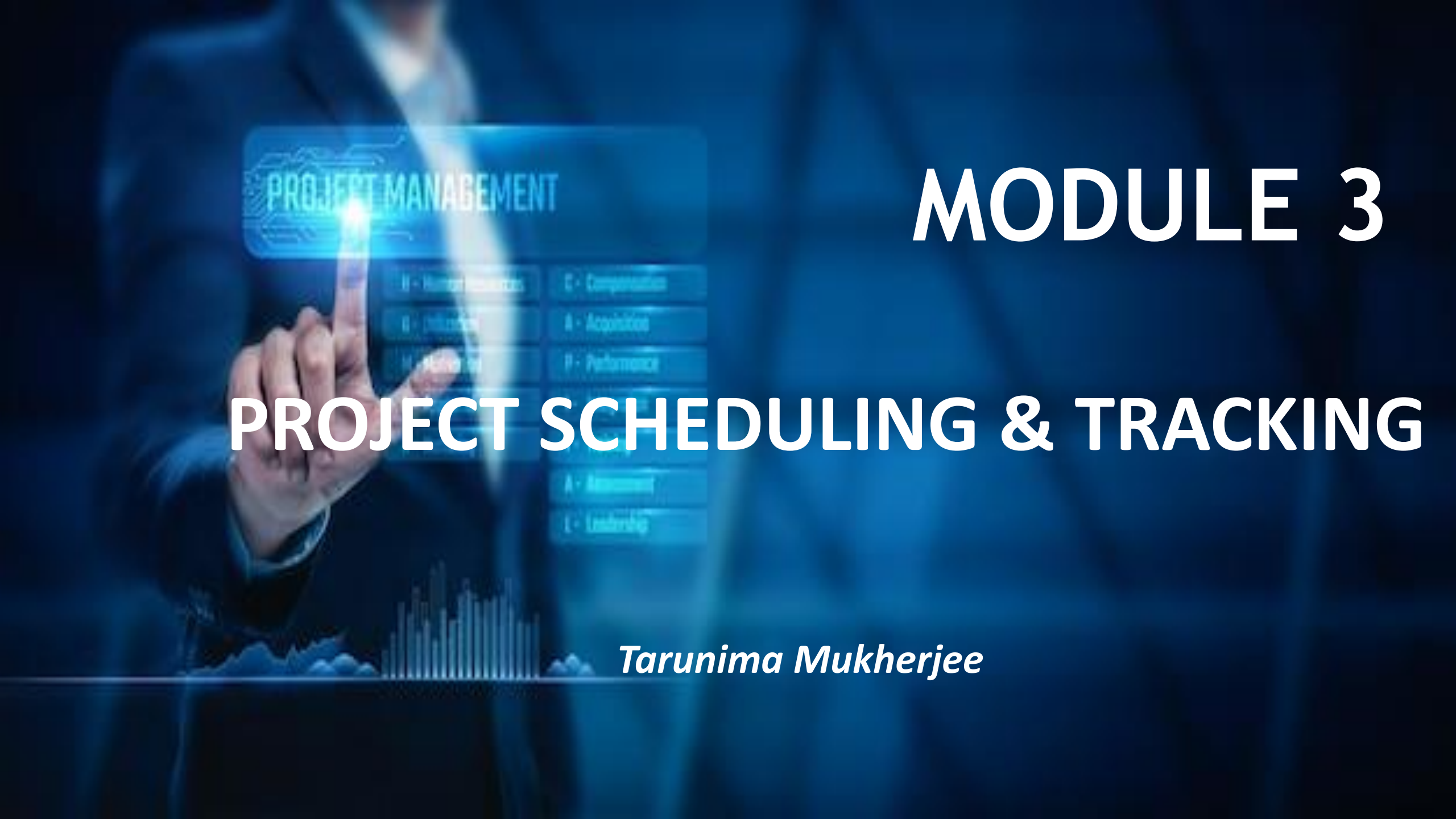



# MODULE 3

# PROJECT SCHEDULING & TRACKING

*Tarunima Mukherjee*

# *Software Project Estimation*

- Predicting the resources required for a software development process

- Fundamental estimation Questions..??

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

# *Software Cost Components*

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

# *Costing & Pricing*

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

# Software Pricing Factors

| Factor | Description |
|---|---|
| Market opportunity | A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed. |
| Cost estimate uncertainty | If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit. |
| Contractual terms | A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer. |
| Requirements volatility | If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements. |
| Financial health | Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business. |

# *Software Pricing Factors - Programmer Productivity*

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

# *Software Pricing Factors - Productivity Measures*

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

- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure

# *Software Pricing Factors - Measurement Problems*

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

# *Software Measurement*

- Software Measurements can be categorized in two ways: ***direct measures*** and ***indirect measures***.

- *Direct measures* of the software engineering process include cost and effort applied, lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.

- *Indirect measures* of the product include functionality, quality, complexity, efficiency, reliability, maintainability, etc.

- The cost and effort required to build software, the number of lines of code produced, and other direct measures are relatively easy to collect, as long as specific.

- Conventions for measurement are established in advance.

- However, the quality and functionality of software or its efficiency or maintainability are more difficult to assess and can be measured only indirectly.

# *Software Measurement contd..*

**A. Size-Oriented Metrics**

Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the *size* of the software that has been produced.

- What's a **line of code?**
  - The measure was first proposed when programs were typed on cards with one line per card
  - How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line

# *Lines of Code*

- The units of LOC are:
  - KLOC- Thousand lines of code
  - NLOC- Non-comment lines of code
  - KDSI- Thousands of delivered source instruction

- The size is estimated by comparing it with the existing systems of the same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

- LoC assumes linear relationship between system size and volume of documentation

# Lines of Code

- A table of size-oriented measures, such as the one shown in Figure 4.4 below, can be created.

| Project | LOC | Effort | $(000) | Pp. doc. | Errors | Defects | People |
|---|---|---|---|---|---|---|---|
| alpha | 12,100 | 24 | 168 | 365 | 134 | 29 | 3 |
| beta | 27,200 | 62 | 440 | 1224 | 321 | 86 | 5 |
| gamma | 20,200 | 43 | 314 | 1050 | 256 | 64 | 6 |
| • | • | • | • | • | • | | |
| • | • | • | • | • | • | | |
| • | • | • | • | • | • | | |

# Lines of Code

- The table lists each software development project that has been completed over the past few years and corresponding measures for that project.
- Referring to the table entry for project alpha: 12,100 lines of code were developed with 24 person-months of effort at a cost of $168,000.
- Effort and cost recorded in the table represent all software engineering activities (analysis, design, code, and test), not just coding.
- Further information for project alpha indicates that 365 pages of documentation were developed, 134 errors were recorded before the software was released, and 29 defects were encountered after release to the customer within the first year of operation. Three people worked on the development of software for project alpha.
- In order to develop metrics that can be assimilated with similar metrics from other projects, we choose lines of code as our normalization value. From the rudimentary data contained in the table, a set of simple size-oriented metrics can be developed for each project.
  - Errors per KLOC (thousand lines of code).
  - Defects per KLOC.
  - $ per LOC.
  - Page of documentation per KLOC
- ***Drawbacks:***
  - This method is Language dependent.
  - It requires details, which are difficult to get in design phase.

| Project | LOC | Effort | $(000) | Pp. doc. | Errors | Defects | People |
|---|---|---|---|---|---|---|---|
| alpha | 12,100 | 24 | 168 | 365 | 134 | 29 | 3 |
| beta | 27,200 | 62 | 440 | 1224 | 321 | 86 | 5 |
| gamma | 20,200 | 43 | 314 | 1050 | 256 | 64 | 6 |
| • | • | • | • | • | • | | |
| • | • | • | • | • | • | | |
| • | • | • | • | • | • | | |

# *Lines of Code*

## Advantages

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to the developer's perspective.
- Both people throughout the world utilize and accept it.
- At project completion, LOC is easily quantified.
- It has a specific connection to the result.
- Simple to use.

# *Lines of Code*

**Disadvantages:**

- Different programming languages contain a different number of lines.
- No proper industry standard exists for this technique.
- It is difficult to estimate the size using this technique in the early stages of the project.
- When platforms and languages are different, LOC cannot be used to normalize.

# *Software Measurement contd..*

B. **Function-Oriented Metrics**

- Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value.

- Since ‘functionality’ cannot be measured directly, it must be derived indirectly using other direct measures.

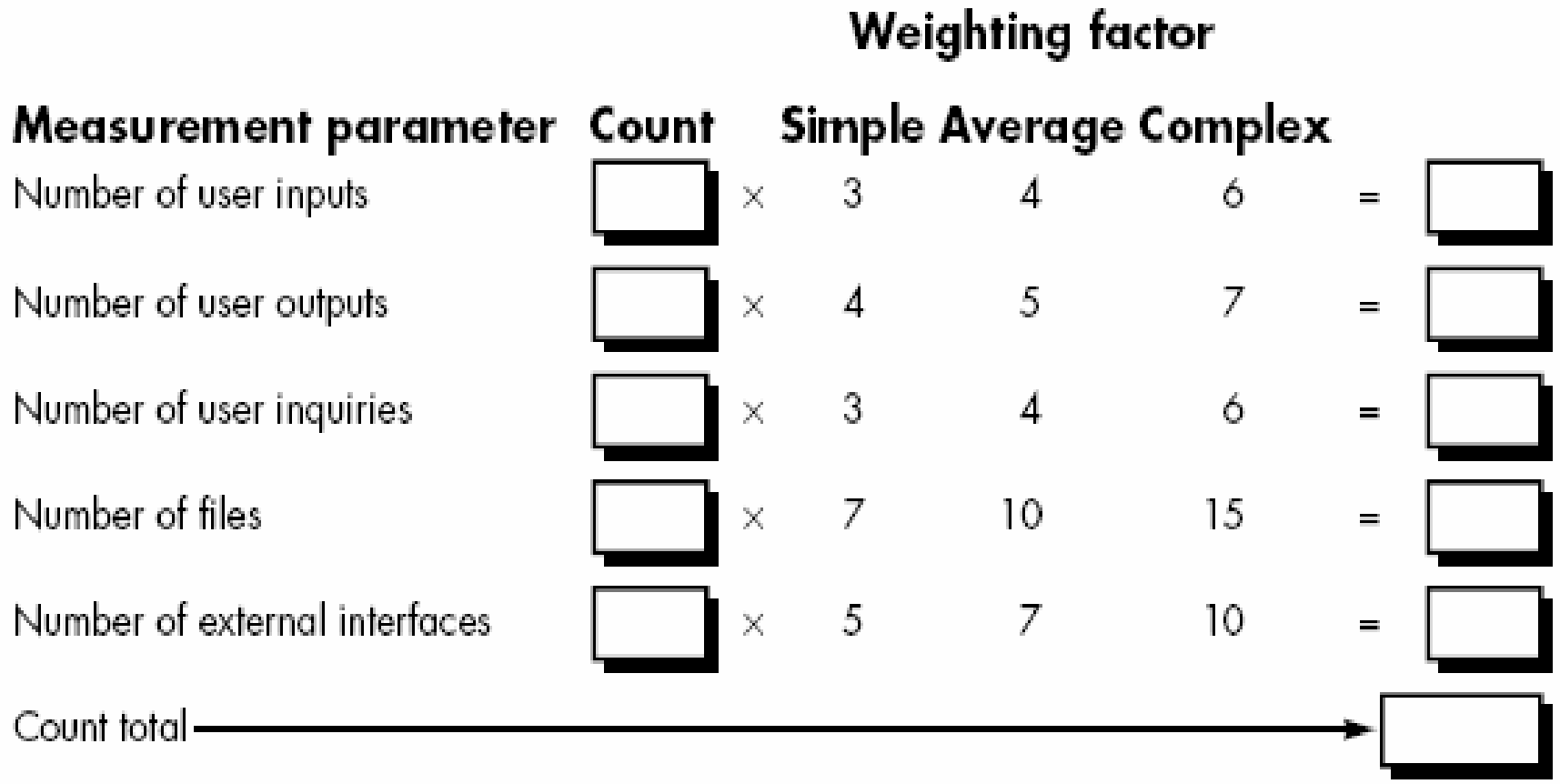
- Function-oriented metrics were first proposed by Albrecht, who suggested a measure called the *function point.*

- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

- Function points are computed by completing the table shown in figure. Five information domain characteristics are determined and counts are provided in the appropriate table location.

# Software Measurement contd..

**FIGURE 4.5**
Computing function points



# *Software Measurement contd..*

Information domain values are defined in the following manner:

- **Number of user inputs.** Each user input that provides distinct application-oriented data to the software is counted.

- **Number of user outputs.** Each user output that provides application-oriented information (i.e. reports, screens, error messages) to the user is counted.

- **Number of user inquiries.** An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

- **Number of files.** Each logical master file is counted.

- **Number of external interfaces.** All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

# *Software Measurement contd..*

- Once these data have been collected, a complexity value is associated with each count.

- Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex.

- To compute function points (FP), the following relationship is used:

**FP = count total * [0.65 + 0.01 * Σ(*Fi*)] (4-1)**

Where count total is the sum of all FP entries obtained from Figure 4.5.

# *Software Measurement contd..*

The *Fi* (*i* = 1 to 14) are "complexity adjustment values" based on responses to the following questions :

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?

# *Software Measurement contd..*

**7.** Does the on-line data entry require the input transaction to be built over multiple screens or operations?

**8.** Are the master files updated on-line?

**9.** Are the inputs, outputs, files, or inquiries complex?

**10.** Is the internal processing complex?

**11.** Is the code designed to be reusable?

**12.** Are conversion and installation included in the design?

**13.** Is the system designed for multiple installations in different organizations?

**14.** Is the application designed to facilitate change and ease of use by the user?

# *Software Measurement contd..*

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

$\sum F = 14 * X$

# *Software Measurement contd..*

**C. Extended Function Point Metrics**

- The function point measure was originally designed to be applied to business information systems applications.
- So, the data dimension (the information domain values discussed previously) was emphasized to the exclusion of the functional and behavioral (control) dimensions.
- For this reason, the function point measure was inadequate for many engineering and embedded systems (which emphasize function and control).
- A number of extensions to the basic function point measure have been proposed to remedy this situation.

# *Software Measurement contd..*

- A function point extension called *feature points* is a superset of the function point measure that can be applied to systems and engineering software applications.

- The feature point measure accommodates applications in which algorithmic complexity is high.

- Real-time, process control and embedded software applications tend to have high algorithmic complexity and are therefore suitable for feature point.

- To compute the feature point, information domain values are again counted and weighted as described in Section FUNCTION ORIENTED METRICS.

# *Software Measurement contd..*

## ADVANTAGES OF FP

- FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages.

- It is based on data that are more likely to be known early in the evolution of a project, making FP more attractive as an estimation approach.

## DISADVANTAGES OF FP

- The method requires some "sleight of hand" in that computation is based on subjective rather than objective data.

- The counts of the information domain (and other dimensions) can be difficult to collect after the fact.

- FP has no direct physical meaning—it's just a number.

# COCOMO Model

- COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e., the number of Lines of Code.
- The COCOMO Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.
- First proposed by Dr. Barry Boehm in 1981.
- Is a heuristic estimation technique- this technique assumes that relationship among different parameters can be modeled using some mathematical expression.
- This approach implies that size is primary factor for cost, other factors have lesser effect.
- “Constructive” implies that the complexity.
- COCOMO prescribes a three stage process for project estimation.
- An initial estimate is obtained, and over next two stages the initial estimate is refined to arrive at a more accurate estimate.
- projects used in this model have following attributes :-
    - ranging in size from 2,000 to 100,000 lines of code
    - Programing languages ranging from assembly to PL/I
    - These projects were based on waterfall model of software development

# BASIC COCOMO MODEL

- Basic COCOMO model computes software development effort, time and cost as a function of program size.
- Program size is expressed in estimated thousands of source lines of code (SLOC, KLOC).

$$\textbf{EFFORT} = a_1 \times (\text{KLOC})^{a2}\ \text{PM}$$

$$T_{dev} = b_1 \times (\text{Effort})^{b2}\ \text{Months}$$

- Where • KLOC is the estimated number of delivered lines (expressed in thousands) of code for project, estimated size of software product. •
- The coefficients a1, a2, $b_1$ and $b_2$ are constants for each category of software products.
- $T_{dev}$ is the estimated time to develop the software, in months.
- Effort is the total efforts required to develop the software product expressed in Person Months (PM)

# COCOMO Model

**1. Organic**

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

**2. Semidetached:**

- If the development team consists of a combination of both experienced and inexperienced staff.
- Team members have limited experience about some aspects but are totally unfamiliar with some aspects of the system being developed.
- Mixed Experience.

**3. Embedded:**

- If the software being developed is strongly coupled to complex hardware.
- Software projects that must be developed within a set of tight software, hardware and operational constraints.

# COCOMO Model

| Mode | Project size | Nature of Project | Innovation | Deadline of the project | Development Environment |
|---|---|---|---|---|---|
| Organic | Typically 2-50 KLOC | Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc. | Little | Not tight | Familiar & In house |
| Semi detached | Typically 50-300 KLOC | Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc. | Medium | Medium | Medium |
| Embedded | Typically over 300 KLOC | Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc. | Significant | Tight | Complex Hardware/ customer Interfaces required |

# COCOMO Model

## Estimation Of Development Effort

Organic : $\text{Effort} = 2.4(KLOC)^{1.05}$ PM

Semi-detached : $\text{Effort} = 3.0(KLOC)^{1.12}$ PM

Embedded : $\text{Effort} = 3.6(KLOC)^{1.20}$ PM

# COCOMO Model

## Calculating Effort & Productivity

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size } (SS) = \frac{E}{D} \; Persons$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity } (P) = \frac{KLOC}{E} \; KLOC / PM$$

# COCOMO Model

| Software Projects | $a_1$ | $a_2$ | $b_1$ | $b_2$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi - Detached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

**Example:** Consider a software project using semi-detached mode with 30,000 lines of code. We will obtain estimation for this project as follows:

(1) Effort estimation E= a1 (KLOC) Exp (a2) person-months

E=3.0(30)Exp(1.12) where lines of code=30000=30 KLOC

E=135 person-month

(2) Duration estimation D= $b_1$ x (Effort) $^{b2}$ Months months =2.5 $(135)^{0.35}$

D=14 months

(3)Person estimation N= E/D =135/14

N=10 persons approx

# COCOMO Model

## Merits-

- Basic COCOMO is good for quick, rough and early estimate of software costs.

## Demerits-

- It does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.
- The accuracy of this model is limited because it does not consider certain factors for cost estimation of software.

# *Intermediate COCOMO Model*

- Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes.
- The Intermediate COCOMO model refines the initial estimate obtained from the basic COCOMO by scaling the estimate up or down based on attributes of software development.
- This model uses a set of 15 cost drivers, these cost drivers are multiplied with the initial cost and effort estimates to scale the estimates up and down.
- This extension considers a set of "cost drivers", each with a number of subsidiary attributes:

**Product attributes**

1. Required software reliability
2. Size of application database
3. Complexity of the product

# *Intermediate COCOMO Model*

**Hardware attributes**

4. Run-time performance constraints
5. Memory constraints
6. Volatility of the virtual machine environment
7. Required turnabout time

**Personnel attributes**

8. Analyst capability
9. Software engineering capability
10. Applications experience
11. Virtual machine experience
12. Programming language experience

**Project attributes**

13. Use of software tools
14. Application of software engineering methods
15. Required development schedule

# Project Characteristics Table

Cost adjustments for computing the EAF (Effort Adjustment Factor)

| | v.low | low | nominal | high | v.high | ex.high |
|---|---|---|---|---|---|---|
| **product attributes** | | | | | | |
| required software reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | |
| database size | | 0.94 | 1.00 | 1.08 | 1.16 | |
| product complexity | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| **computer attributes** | | | | | | |
| execution time constraints | | | 1.00 | 1.11 | 1.30 | 1.66 |
| main storage constraints | | | 1.00 | 1.06 | 1.21 | 1.56 |
| virtual machine volitility | 0.87 | 1.00 | 1.15 | 1.30 | | |
| computer turnaround time | | 0.87 | 1.00 | 1.07 | 1.15 | |
| **personnel attributes** | | | | | | |
| analyst capability | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | |
| applications experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | |
| programmer capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | |
| virtual machine experience | 1.21 | 1.10 | 1.00 | 0.90 | | |
| programming language experience | 1.14 | 1.07 | 1.00 | 0.95 | | |
| **project attributes** | | | | | | |
| use of modern programming practices | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | |
| use of software tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | |
| required development schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | |

- 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). •

formula now takes the form: $E=a_1(KLOC)^{(b1)} \times (EAF)$

Where E: Effort applied in terms of person- months

KLOC: Kilo lines of code for the project

EAF: It is the effort adjustment factor •

The value of a and b for various class of software projects

| Software Projects | $a_1$ | $b_1$ |
|---|---|---|
| Organic | 3.2 | 1.05 |
| Semi - Detached | 3.0 | 1.12 |
| Embedded | 2.8 | 1.20 |

**Example:** Consider a project having 30.000 lines of code which in an embedded software with critical area hence reliability is high.
The estimation can be
$E=a_1 (KLOC)^{b1} x (EAF)$

As reliability is high EAF=1.15 (product attribute)
$a_1 =2.8$
$b_1=1.20$. for embedded software
E=2.8(30) 120 x 1.15
=191 person month

$D=b, (E)^{b2}$
$=2.5(191)^{0.32}$
=13 months approximately

N=E/D=191/13
N=15 persons approx.

**Merits:**

- This model can be applied to almost to entire software product for easy and rough cost estimation during early stage.
- It can be applied at the software product component level for obtaining more accurate cost estimation.

**Demerits:**

- The effort multipliers are not dependent on phases. A product with many components is difficult to estimate.

# 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.

➢ So, **Complete COCOMO** was proposed to overcome these limitations of basic and intermediate COCOMO.

# COMPLETE COCOMO MODEL

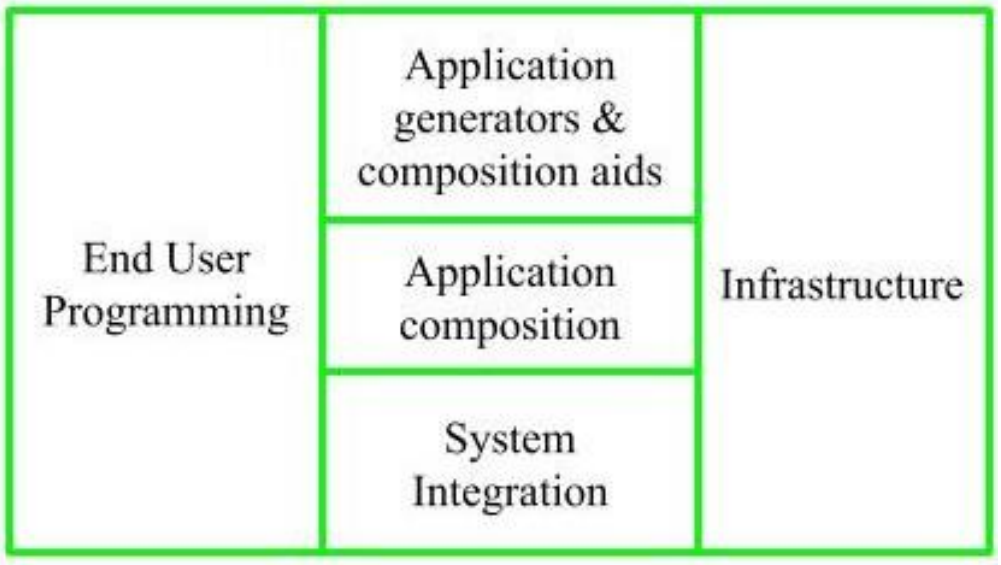
- Incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.
- The complete COCOMO model considers the differences in characteristics of all the subsystems and estimates the effort and development time as sum of the estimates for the individual sub systems.
- Uses different effort multipliers for each cost driver attribute. These Phase Sensitive effort multipliers are each to determine the amount of effort required to complete each phase. In complete COCOMO, the whole software is divided in different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort
- The effort is calculated as function of program size and a set of cost drivers given according to each phase of software life cycle
- A complete project schedule is never static

# COCOMO II Model

- **COCOMO-II** is the revised version of the original Cocomo (Constructive Cost Model) and was developed at the University of Southern California.
- It is the model that allows one to estimate the cost, effort, and schedule when planning a new software development activity.
- COCOMO II model gives estimates that indicate a standard deviation near the most common estimate.
- Its primary goal is to offer methodologies, quantitative analytic structure, and tools. It computes the total development time and effort that is based on the estimates of all individual subsystems.
- It is based on the non-linear reuse formula. This approach assists in offering estimates that represent one standard deviation of the most common estimate.
- COCOMO-II model is useful in non-sequential, rapid development, reengineering and reuse models of software development cycle.

# COCOMO II Model

## *Sub-Models of COCOMO Model*



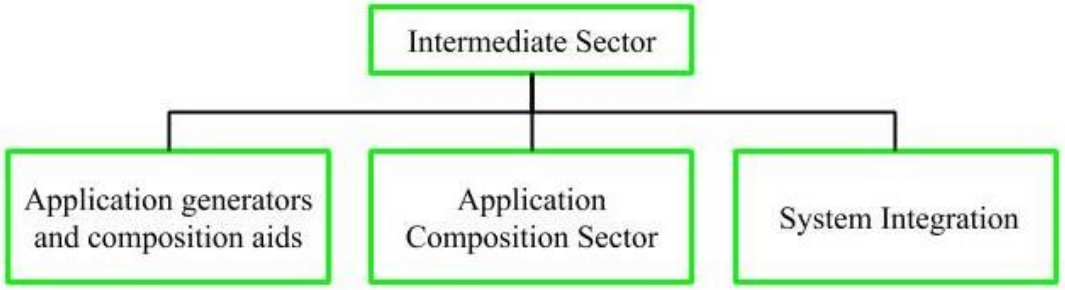

**1. End User Programming**

Application generators are used in this sub-model. End user write the code by using these application generators.

For Example, Spreadsheets, report generator, etc.

# COCOMO II Model

## 2. Intermediate Sector




• **Application Generators and Composition Aids:** This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus, Oracle, IBM, Novell.

• **Application Composition Sector:** This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.

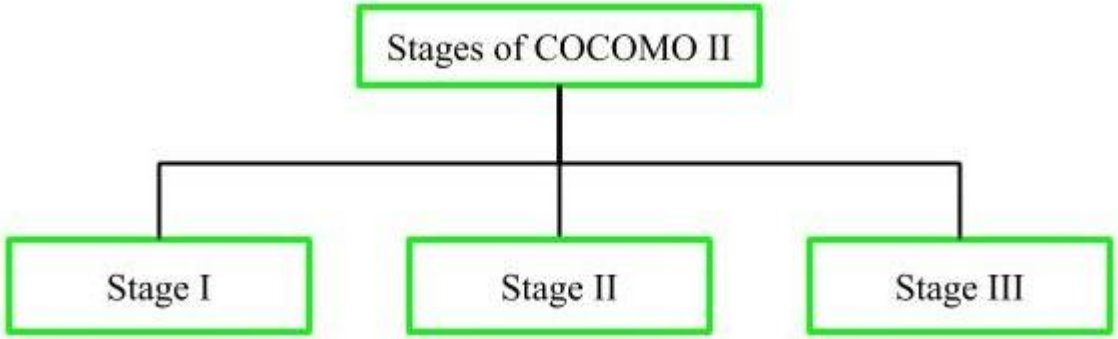
• **System Integration:** This category deals with large scale and highly embedded systems.

## 3. Infrastructure Sector

This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

# COCOMO II Model

## Stages of COCOMO II Model




**Stage-I**

It supports estimation of prototyping. For this it uses Application Composition Estimation Model. This model is used for the prototyping stage of application generator and system integration.

**Stage-II**

It supports estimation in the early design stage of the project, when we less know about it. For this it uses Early Design Estimation Model. This model is used in early design stage of application generators, infrastructure, system integration.

**Stage-III**

It supports estimation in the post architecture stage of a project. For this it uses Post Architecture Estimation Model. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.

# COCOMO Model

## Difference Between COCOMO I & COCOMO II Model

| Features | COCOMO I | COCOMO II |
|---|---|---|
| **Utilization Models** | It is utilized in the waterfall model of SDLC. | It is useful in quick development, non-sequential, and reuses software models. |
| **Basic** | It is based on the linear reuse formula. | It is based on the non-linear formula. |
| **Estimation Precision** | It offers estimates of the effort and timeline. | It offers estimates that are one standard deviation from the most common estimate. |
| **Size of program statements** | The program's size is indicated in code lines. | It has extra factors for expressing software size, including lines of code, object points, and function points. |
| **Model framework** | The requirements given to the software come first in the development process. | It utilizes the spiral type of development. |
| **Effort equation's exponent** | The 3 development methods define the exponent of the effort equation. | The 5 scale methods define the exponent of the effort equation. |
| **Sub models** | It has 3 submodels. | It has 4 submodels. |
| **Cost Drivers** | It utilizes 15 cost drivers. | It utilizes 17 cost drivers. |

# *Agile Estimation*

- Agile estimation gauges the effort needed to achieve a prioritized task in the product backlog.

- Usually measured concerning the time needed for task completion, this effort leads to accurate sprint planning. Agile estimates are also made with reference to story points– a number that enables evaluation of the difficulty of successful completion of a user story successfully.

- Despite accurately estimating the effort, one must keep room for impediments and not strive to achieve perfect accuracy. Changing requirements, Agile anti-patterns and other realities alter the course of development.

- Estimation is the process of finding an estimate, or approximation. It is an Art of guessing. Estimation includes four main factors – money, effort, resources, and time needed to build a specific system or product.

- There are many techniques available in today's world for doing estimations in an Agile Project.

# Agile Estimation

## *Principles of Agile Estimation*

Some of the basic principles of Agile estimation techniques are as follows:

• **Collaborative:** Involving everyone on the Agile development team is one of the best practices because collaborative efforts lead to better estimates. Collaborative techniques put an end to the blame game for an incorrect estimate.

• **They are designed to be fast:** Faster than any traditional techniques, Agile estimation is not about predicting the future. Rather it recognizes that estimations are a non-value-added activity and tries to minimize them.

• **Use of relative units:** Estimation is not in terms of dollars or days; instead, “points” or qualitative labels are employed for estimating or comparing tasks.

# *Agile Estimation - Planning Poker*

- **Planning Poker is a widely used agile estimation technique** for estimating the size or effort required to complete tasks, features, or user stories during the software development process.
- **Planning Poker is a consensus-based technique for estimating**, mostly used to estimate effort or relative size of user stories in Scrum.
- *Planning Poker combines three estimation techniques – Wideband Delphi Technique, Analogous Estimation, and Estimation using WBS.*
- Planning Poker was first defined and named by James Grenning in 2002 and later popularized by Mike Cohn in his book "Agile Estimating and Planning", whose company trade marked the term.
- *During a planning session, the product owner or customer reads the user story or describes the features to the team.*

# Agile Estimation - Planning Poker

The entire Scrum team is involved and it results in quick & reliable estimates. Steps involved are as below -

**1.Preparation**: The team gathers together, either physically or virtually, for an estimation session. Each team member should have a deck of Planning Poker cards or access to a digital tool that facilitates the process.

**2.Backlog Refinement**: The Product Owner presents the user stories or backlog items to be estimated. These are typically features, tasks, or user stories that need to be implemented during the upcoming sprint or iteration.

**3.Explanation**: The Product Owner provides a brief overview of the user story, including its requirements, acceptance criteria, and any relevant information necessary for the team to understand the scope of the work.

**4.Estimation**: Each team member independently selects a Planning Poker card representing their estimate of the effort required to complete the user story. The cards typically use Fibonacci sequence numbers (0, 1, 2, 3, 5, 8, 13, 21, etc.) to reflect the relative sizing rather than absolute time estimates. This helps to avoid the misconception of precise estimates and encourages focusing on relative complexity.

**5.Reveal**: Once everyone has selected a card, all team members reveal their cards simultaneously.

# Agile Estimation - Planning Poker

**6.Discussion**: If there's a significant difference in estimates, team members discuss their reasoning. This discussion helps clarify misunderstandings, uncover assumptions, and align everyone's understanding of the user story.

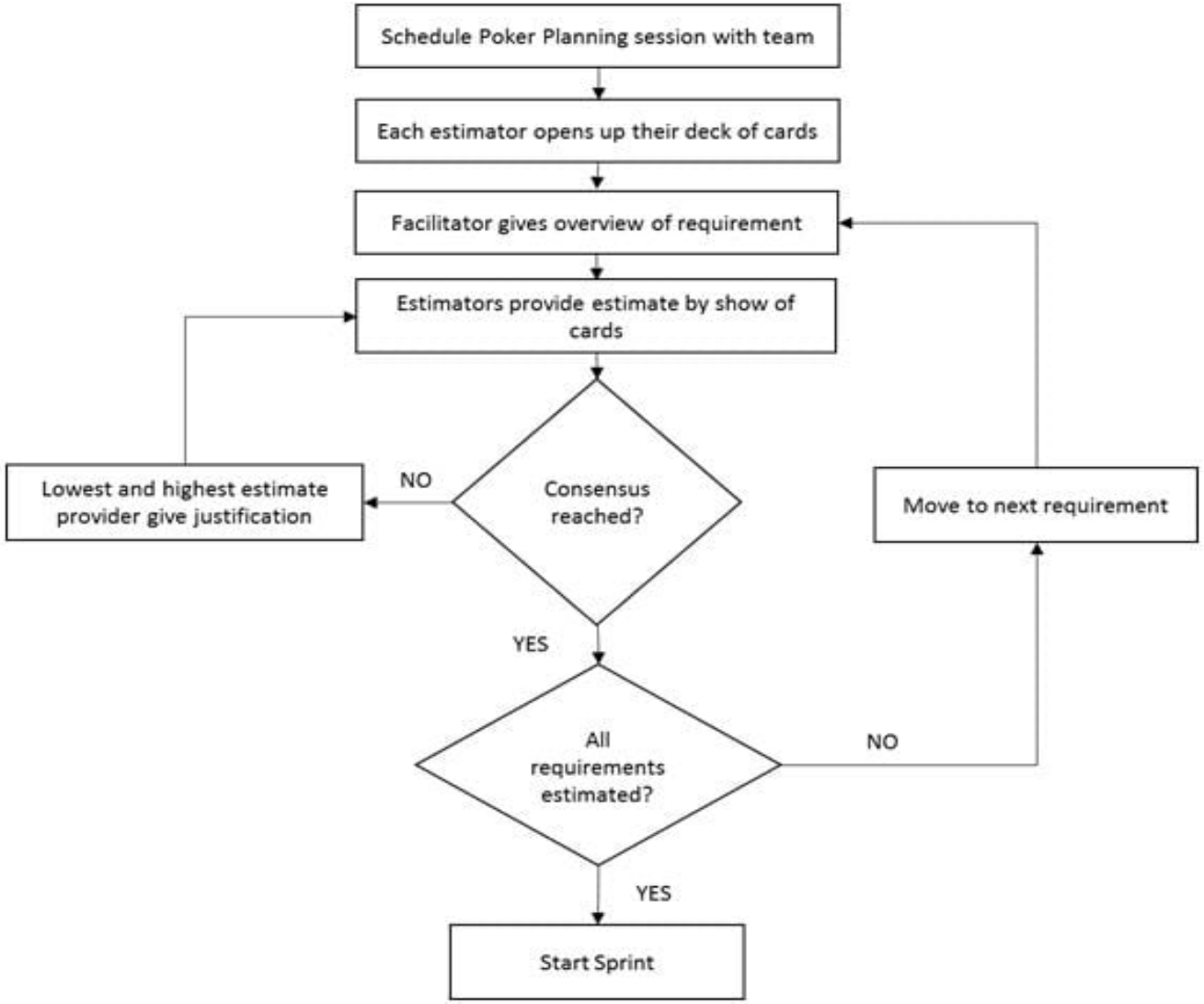
**7.Re-Estimation (Optional)**: After discussion, team members may choose to re-estimate the user story. This process continues until a rough consensus is reached.

**8.Recording**: The agreed-upon estimate is recorded. This could be the value chosen by the majority of the team or an average of the estimates.

**9.Repeat**: The team repeats the process for each user story or backlog item, continuing until all necessary items are estimated.

- Planning Poker encourages collaboration, engagement, and shared understanding among team members.
- It leverages the collective intelligence of the team while avoiding biases associated with individual estimation.
- Over time, teams typically improve their estimation accuracy and gain a better understanding of their velocity, enabling more predictable delivery of software increments.

# Agile Estimation - Planning Poker - Workflow



# *Agile Estimation - Planning Poker*

## Advantages of using Planning Poker

- The structured, game-like format ensures that the session keeps moving along.
- It provides an opportunity for experts to voice and share their opinion.
- The format can surface-up wrong assumptions when there are differences in estimates and thereby bring a consensus to the understanding of the user story.

## Disadvantages of using Planning Poker

- Coming to a consensus in estimates does not necessarily mean that the understanding of the expectations of the user story is accurate.

# Agile Estimation - User Story

- In this estimation technique, user stories are given story points based on complexities, effort, and risk by teams.
- Story points are a metric without units and can stand in for any comparable value decided by the team.
- This method is adaptable and frequently used alongside Planning Poker.
- Story points are a way to estimate the amount of effort required to complete a user story in your product backlog.
- Teams usually estimate story points before a sprint planning meeting, since that's when your team determines how much work they can carry out in an upcoming sprint.
- Typically, story points take into account three factors that can impact a task's scope and effort, and the story point's value increases accordingly.
    - **Risk -** is the amount of total risk or uncertainty associated with the task. For example if the task involves third parties, contractors, or project stakeholders, it can increase the amount of risk.
    - **Repetition -** is the team's experience with similar tasks.
    - **Complexity -** is the task's level of difficulty (and how clear the objectives of the task are).
- One important thing to know is that story points are relative—meaning that their relative value and ratios to each other are what matter, not their actual numerical value.

# Agile Estimation - User Story

***Process of Agile Estimation using User Story Planning***

**1.Product Backlog Creation**: Start by creating a product backlog, which is a prioritized list of all desired features, enhancements, and bug fixes for the software product. Each item in the backlog should be a user story.

**2.User Story Definition**: User stories should follow the INVEST criteria:

1. Independent: Each user story should be self-contained and not dependent on other stories.
2. Negotiable: User stories are open for discussion and can be modified based on feedback.
3. Valuable: Each user story should deliver value to the end-user or customer.
4. Estimable: User stories should be clear enough for the team to provide estimates.
5. Small: User stories should be small enough to be completed within a single iteration.
6. Testable: There should be clear acceptance criteria for each user story.

**3.Story Points Estimation**: Instead of estimating in hours, Agile teams often use story points for estimation.

- Story points are a relative measure of the effort required to implement a user story, considering factors like complexity, uncertainty, and effort.
- The Fibonacci sequence (1, 2, 3, 5, 8, 13, 21, etc.) is commonly used for story point estimation. During estimation, the team discusses each user story and collectively assigns story points based on their understanding.

# *Agile Estimation - User Story*

***Process of Agile Estimation using User Story Planning***

**4. Planning Poker**: Planning Poker is a popular technique used by Agile teams for story point estimation. In Planning Poker, each team member receives a deck of cards representing story points (e.g., Fibonacci sequence). The team discusses a user story, and then each member selects a card representing their estimate. If there's a wide discrepancy in estimates, the team discusses the reasons behind their estimates and **repeats the process until a consensus is reached.**

5. **Velocity Calculation**: *Velocity is the measure of the amount of work a team can complete* in a single iteration (sprint). After several iterations, the team can **calculate their average velocity**. This helps in predicting how many story points the team can complete in future sprints, aiding in release planning and prioritization.

6. **Refinement and Adjustments**: As the project progresses, *the team continues to refine user stories, update estimates, and adjust priorities based on new information, feedback, and changes in requirements.*

By following above steps, Agile teams can effectively plan and estimate user stories, leading to better project management and successful software delivery.

# Agile Estimation – User Story

***Advantages of using User Story Planning***

- **Drive faster planning -** Story points are relative, meaning you calculate the value of one story point by comparing it to similar, already estimated points. Using a relative scoring method leads to faster estimation over time.
- **Take unpredictability and risk into account -** Story points account for elements like unpredictability and risk. Using these factors in your planning takes the guesswork out of estimating, letting you more accurately scope effort.
- **Remove skills bias from your planning and get your team on the same page-** Relying on individual team member estimations isn't always best. After all, a senior team member will probably give a pretty different effort estimation than a junior team member. Story points prevent these issues by encouraging collaboration in the form of planning poker meetings.
- **Create meaningful deadlines-** No one likes arbitrary deadlines, but that's often what you get when you use other estimation techniques. Since story points are more nuanced, they result in meaningful due dates.
- **Build better estimations going forward-** One of the major perks of story points is they're adaptable and reusable. That means once you've created a story point matrix and held your first sprint, you can use your learnings to reevaluate your original story point values and develop more accurate estimations.

# Agile Estimation - User Story

## *Disadvantages of using User Story Planning*

**1.Subjectivity**: User stories rely heavily on the interpretation of the team members. Different team members may have varying perspectives on the complexity and effort required to implement a user story, leading to inconsistencies in estimation.

**2.Lack of Detail**: User stories are intentionally brief and high-level, which can sometimes lead to ambiguity or insufficient detail. Without clear acceptance criteria or a full understanding of the requirements, accurate estimation becomes challenging.

**3.Dependency on Expertise**: Estimating user stories often requires domain knowledge and technical expertise. Teams with diverse skill sets may struggle to reach a consensus on estimates, especially if certain team members are unfamiliar with the technology or domain.

**4.Time-consuming Estimation Process**: Planning Poker and other estimation techniques can be time-consuming, especially for large backlogs or complex user stories. Spending too much time on estimation can detract from actual development work.

**5.Overemphasis on Velocity**: Agile teams often use velocity as a key metric for planning and forecasting. However, solely relying on velocity for estimation can be misleading, as it doesn't account for factors such as team dynamics, changes in requirements, or unforeseen technical challenges.

# *Project Scheduling*

- Break down tasks and activities
- Estimate task durations
- Sequence activities based on dependencies
- Allocate resources considering availability and skills
- Create a timeline using scheduling techniques
- Monitor progress and track deviations
- Adjust schedule as needed to accommodate changes
- Communicate schedule updates to stakeholders
- Ensure adherence to budget and scope
- Continuously refine schedule for optimization

# *Timeline Charts*

- Project management is a balancing act, and project managers need project management skills and tools to keep every ball up in the air. There's the vision statement that clarifies the project's purpose, a cost-benefit analysis that compares the costs and benefits of a business action.

- A timeline chart orders events sequentially. Time can be measured in any unit, from mere minutes to months to years. Important dates or milestones are plotted chronologically along a vertical or horizontal time scale to provide a quick but comprehensive view of the sequence of past or future events.

# *Timeline Charts*

- **Timeline charts and visual project management**

- Visual project management, as its name suggests, is a practice that incorporates visual tools and data visualization methodologies with traditional project management techniques to make complex data points easier to communicate and understand.

- Timeline charts show project teams and stakeholders at a glance which milestones have been completed or yet to be achieved, and the due dates associated with upcoming milestones. Charting deliverables as visible milestones is an industry best practice and stresses their urgency.

# *Types of Timeline Charts*

There are several types of timeline charts commonly used in project management and software engineering. Here are some of the most popular ones:

1. Gantt Chart: A Gantt chart is a horizontal bar chart that illustrates a project schedule. It shows tasks or activities along the vertical axis and time along the horizontal axis. The length of the bars represents the duration of each task, and dependencies between tasks are shown using arrows.

2. PERT Chart: Program Evaluation Review Technique (PERT) charts are network diagrams used to depict project tasks and their dependencies. PERT charts typically include nodes representing tasks and arrows indicating the sequence of activities. They often incorporate estimated durations and critical paths.

3. Milestone Chart: A milestone chart focuses on significant events or milestones throughout the project lifecycle. It highlights key dates such as project kickoff, deliverable deadlines, or major milestones. Milestone charts often use symbols or markers to represent each milestone on a timeline.

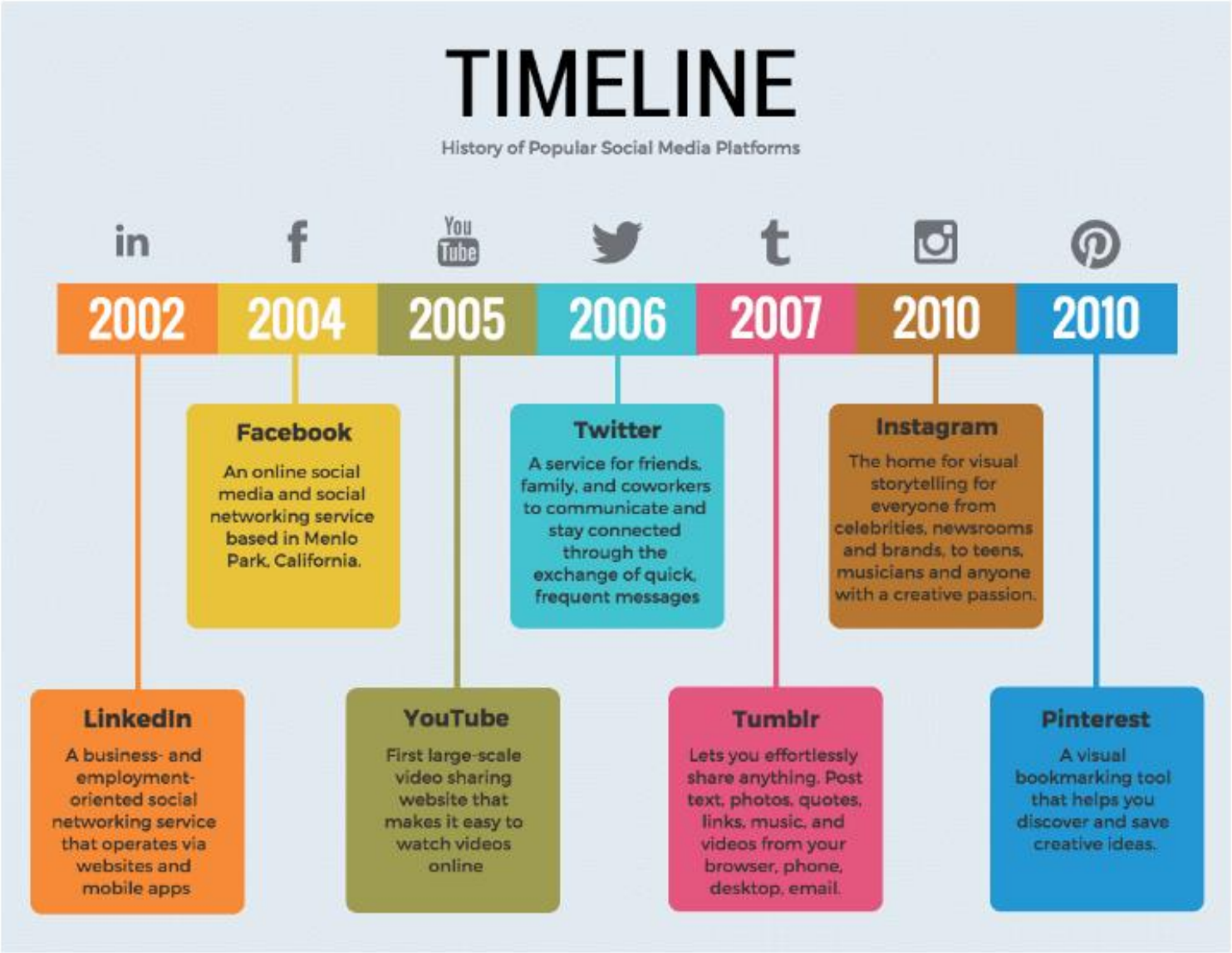
4. Burndown Chart : Burndown charts are commonly used in Agile project management to track progress over time. They plot the remaining work (typically in story points or tasks) against time. Burndown charts help visualize whether the team is on track to complete the work by the end of the iteration or release.

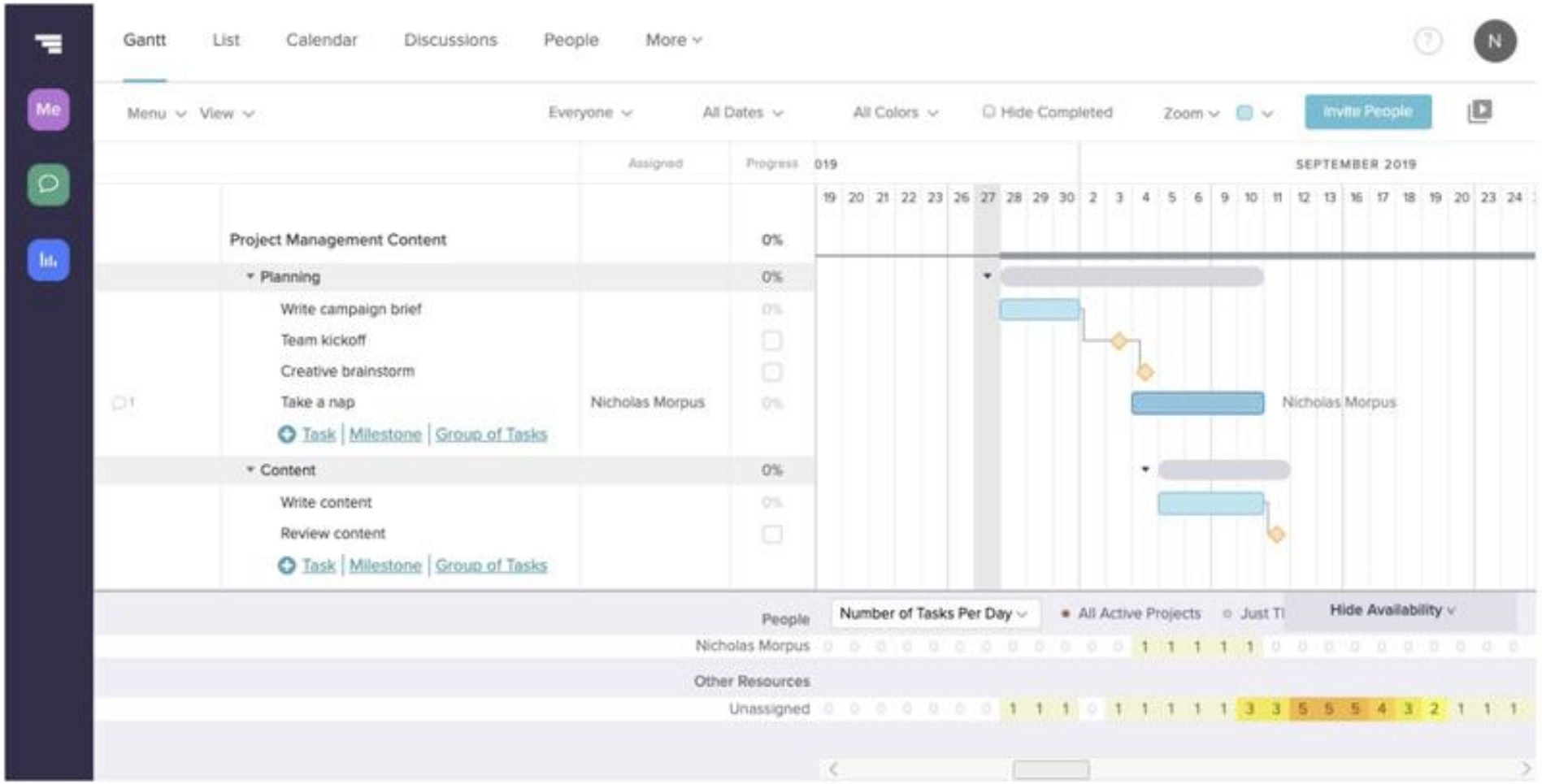
# *Types of Timeline Charts*

5. Burnup Chart: Burnup charts also track progress over time but display both completed work and total work (scope). They show the cumulative completed work alongside the total planned work, providing insight into how much work has been completed relative to the project's scope.

6. Histogram: While not exclusively a timeline chart, histograms can be used to visualize the distribution of task durations or resource utilization over time. They provide a graphical representation of data distribution and help identify patterns or trends in project schedules.

Each type of timeline chart serves a specific purpose and can be used individually or in combination to effectively communicate project schedules and progress to stakeholders.


TIMELINE
History of Popular Social Media Platforms
2002
2004
2005
2006
2007
2010
2010
LinkedIn
A business- and employment-oriented social networking service that operates via websites and mobile apps
Facebook
An online social media and social networking service based in Menlo Park, California.
YouTube
First large-scale video sharing website that makes it easy to watch videos online
Twitter
A service for friends, family, and coworkers to communicate and stay connected through the exchange of quick, frequent messages
Tumblr
Lets you effortlessly share anything. Post text, photos, quotes, links, music, and videos from your browser, phone, desktop, email.
Instagram
The home for visual storytelling for everyone from celebrities, newsrooms and brands, to teens, musicians and anyone with a creative passion.
Pinterest
A visual bookmarking tool that helps you discover and save creative ideas.


Gantt
List
Calendar
Discussions
People
More
Menu
View
Everyone
All Dates
All Colors
Hide Completed
Zoom
Invite People
Assigned
Progress
SEPTEMBER 2019
Project Management Content
0%
Planning
Write campaign brief
Team kickoff
Creative brainstorm
Take a nap
Nicholas Morpus
Task | Milestone | Group of Tasks
Content
Write content
Review content
Task | Milestone | Group of Tasks
People
Number of Tasks Per Day
All Active Projects
Hide Availability
Nicholas Morpus
Other Resources
Unassigned

# *Critical Path Method (CPM)*

- The critical path method (CPM) is a step-by-step methodology, technique or algorithm for planning projects that have complex, interdependent interactions.
- Technically, the critical path will have the least amount of float.
- It plays an important role in helping project managers understand which missed milestones could prevent the entire project from being successfully completed on time -- as well as where extra resources could help.
- Finding the critical path is very important for project managers because it allows them to:
- Accurately estimate the total project duration
- Identify task dependencies, resource constraints and project risks
- Prioritize tasks and create realistic project schedules

# *Critical Path Method (CPM)*

The basic steps in CPM are:

1.Determine required tasks.

2.List required tasks in sequence.

3.Create a flowchart including each required task.

4.Identify all critical and non-critical relationships (paths) among required tasks.

5.Assign an expected completion/execution time for each required task.

6.Study all critical relationships to determine possible alternatives for as many as possible.

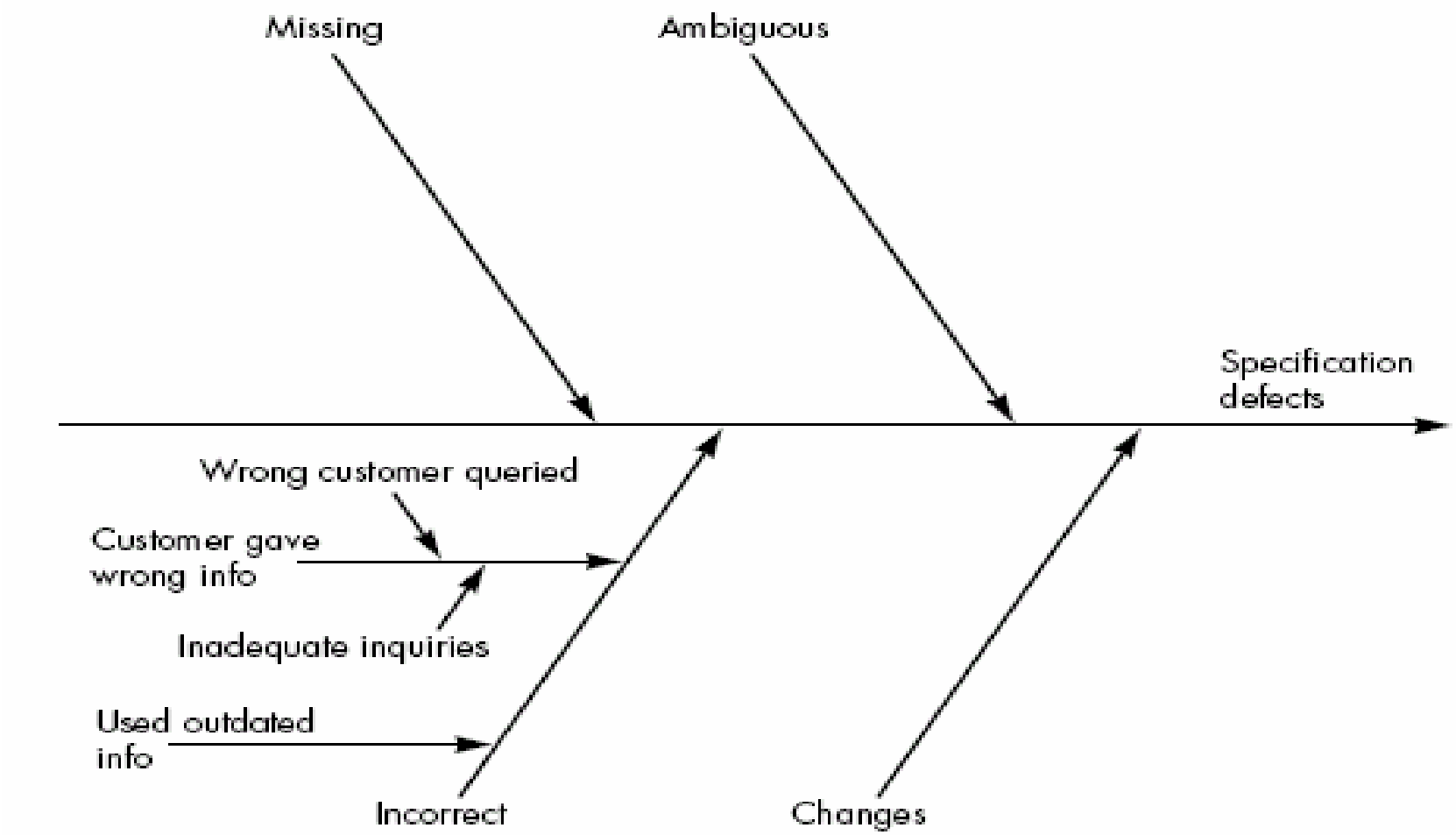
# *Key Elements of CPM*

- Earliest start time (ES): This is simply the earliest time that a task can be started in your project. You cannot determine this without first knowing if there are any task dependencies
- Latest start time (LS): This is the very last minute in which you can start a task before it threatens to delay your project schedule
- Earliest finish time (EF): The earliest an activity can be completed, based on its duration and its earliest start time
- Latest finish time (LF): The latest an activity can be completed, based on its duration and its latest start time
- Float: Also known as slack, float is a term that describes how long you can delay a task before it impacts its task sequence and the project schedule. The tasks on the critical path have zero float, because they can’t be delayed

# *Fishbone Diagram*

- Grady suggests the development of a *fishbone diagram* to help in diagnosing the data represented in the frequency diagram.
- the spine of the diagram (the central line) represents the quality factor under consideration
- Each of the ribs (diagonal lines) connecting to the spine indicate potential causes for the quality problem (e.g., missing requirements, ambiguous specification, incorrect requirements, changed requirements).
- Expansion is shown only for the *incorrect* cause .
- The collection of process metrics is the driver for the creation of the fishbone diagram.
- A completed fishbone diagram can be analyzed to derive indicators that will enable a software organization to modify its process to reduce the frequency of errors and defects.

# *A fishbone diagram:*




THANK
YOU!