22CSC51 - AGILE METHODOLOGIES

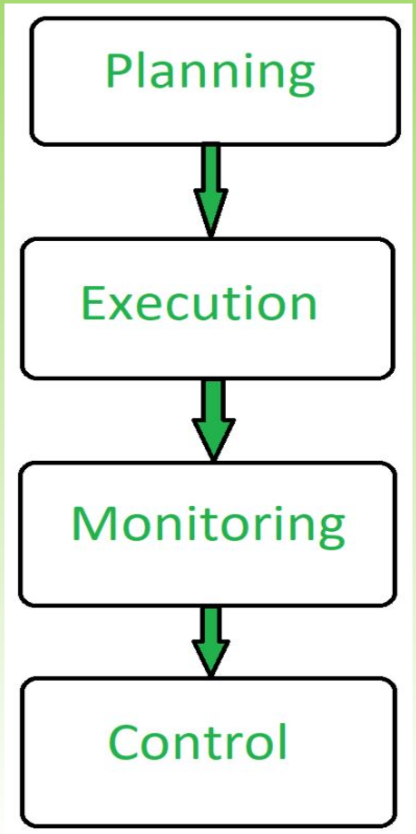
Unit - V

Software Project Management

Muthuraja M, AP/CSE, KEC

Definition

- Software Project Management (SPM) is a proper way of planning and leading software projects.
- It is a part of project management in which software projects are planned, implemented, monitored, and controlled.
- Project Management begins before any technical activity and continues throughout the Definition,development and support of computer software.



THE MANAGEMENT SPECTRUM

- Effective software project management focuses on the four P’s:
   - People,
   - Product,
   - Process, and
   - Project.

The People

- ***The people capability maturity model(People-CMM)*** defines the following key practice areas for software people:

  staffing, communication and coordination, work environment, performance management, training, compensation, competency analysis and development, career development, workgroup development, team/culture development, and others.

- Organizations that achieve high levels of People-CMM maturity have a higher likelihood of implementing effective software project management practices.
- The different types of people involved in project management are:
  - The Stakeholders
  - Team Leaders
  - The Software Team
  - Agile Teams

The People

The Stakeholders:

The software process (and every software project) is populated by stakeholders who can be categorized into one of five constituencies:

Senior managers: who define the business issues

Project (technical) managers: who must plan, motivate, organize, and control the practitioners who do software work.

Practitioners: who deliver the technical skills that are necessary to engineer

Customers: who specify the requirements for the software to be engineered

End users: who interact with the software once it is released for production use.

The People

Team Leaders:

Jerry Weinberg MOI model of leadership:

- ✔ Motivation
- ✔ Organization
- ✔ Ideas or innovation

- The effective project manager emphasizes four key traits:
  - Problem solving
  - Managerial identity
  - Achievement
  - Influence and team building

The People

The Software Team:

seven project factors that should be considered when planning the structure of software engineering teams are:

- ✔ Difficulty of the problem to be solved
- ✔ “Size” of the resultant program(s) in lines of code or function points
- ✔ Time that the team will stay together (team lifetime)
- ✔ Degree to which the problem can be modularized
- ✔ Required quality and reliability of the system to be built
- ✔ Rigidity of the delivery date
- ✔ Degree of sociability (communication) required for the project

The People

Agile Teams:

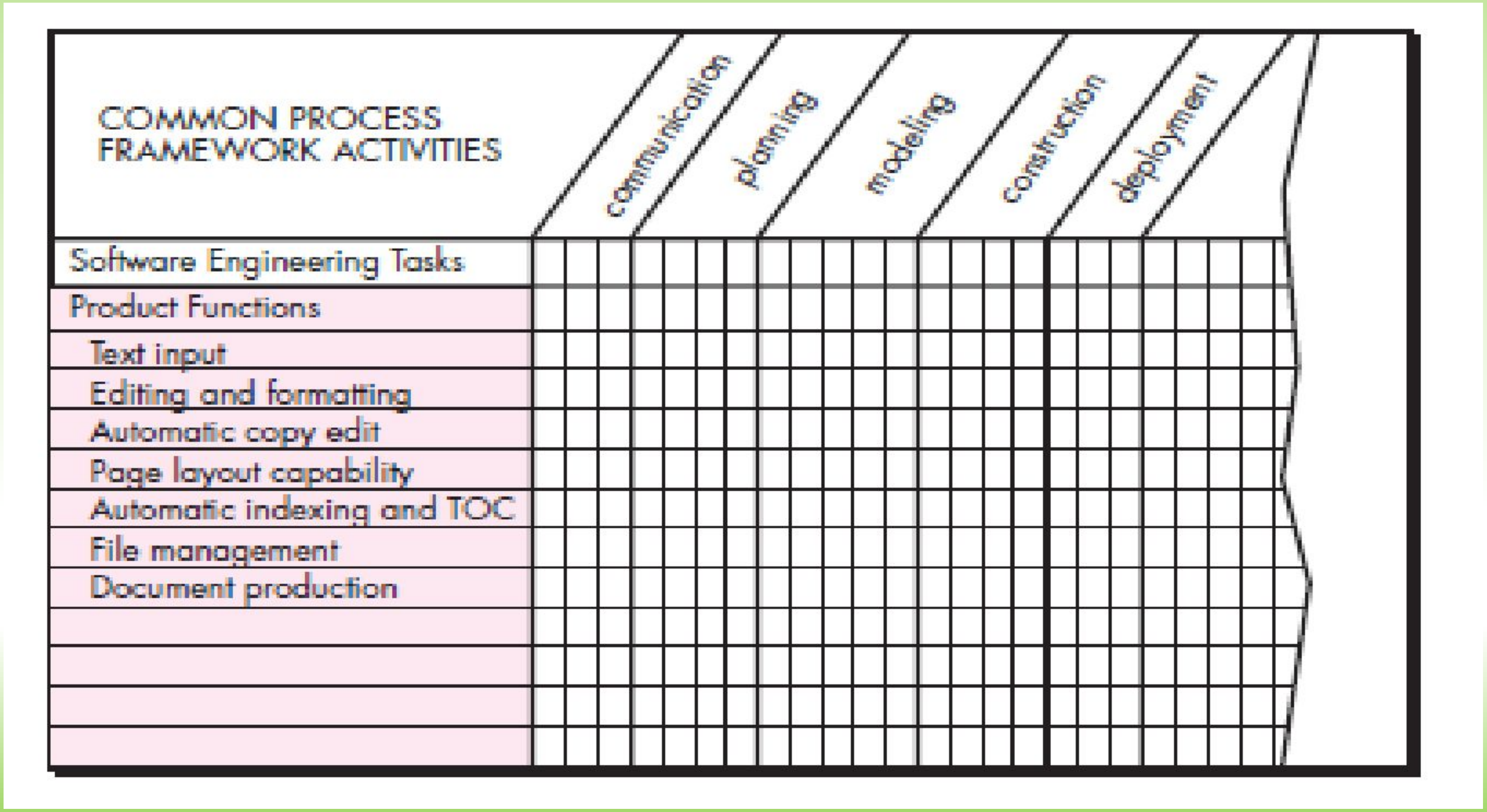
- To review, the agile philosophy encourages customer satisfaction and early incremental delivery of software, small highly motivated project teams, informal methods, minimal software engineering work products, and overall development simplicity.
- Many agile process models (e.g., Scrum) give the agile team significant autonomy to make the project management and technical decisions required to get the job done.

The Product

- Before a **project can be planned**, **product objectives and scope should be established,** alternative solutions should be considered, and technical and management constraints should be identified.
- The **first software project management activity** is the determination of **software scope.**
- Software project scope must be **unambiguous and understandable at the management and technical levels**.
- Scope is defined by answering the following questions:
  - ✔ **Context: How does the software to be built** fit into a larger system.
  - ✔ **Information objectives: What customer-visible data objects** are produced as output from the software? What data objects are **required for input?**
  - ✔ **Function and performance: What function does the software**e perform to transform input data into output? Are any special performance characteristics to be addressed?

THE PROCESS

- A software **process provides the framework** from which a comprehensive plan for software development can be established.



COMMON PROCESS FRAMEWORK ACTIVITIES	communication	planning	modeling	construction	deployment
Software Engineering Tasks					
Product Functions					
Text input					
Editing and formatting					
Automatic copy edit					
Page layout capability					
Automatic indexing and TOC					
File management					
Document production					

THE PROJECT

- In order to manage a successful software project, we have to **understand what can go wrong** so that problems can be avoided.
- Five-part common sense approach to software projects are as follows:
  - **Start on the right foot.**
  - **Maintain momentum**
  - **Track progress**
  - **Make smart decisions**
  - **Conduct a postmortem analysis**

THE W^{5}HH PRINCIPLE

Boehm suggest an approach that addresses project objectives , milestones and schedules, responsibilities, management and technical approaches, and required resources. He call it as ***THE W^{5}HH PRINCIPLE***

- **Why is the system being developed?**
- **What will be done?**
- **When it will be done?**
- **Who is responsible for a function?**
- **Where are they located organizationally?**
- **How will the job be done technically and managerially?**
- **How much of each resource is needed?**

PROCESS AND PROJECT METRICS

- Introduction
- Metrics in the Process Domain
- Metrics in the Project Domain
- Software Measurement Metrics for Software Quality
- Integrating Metrics within the Software Process
- Establishing A Software Metrics Program

PROCESS AND PROJECT METRICS

- ***Process metrics*** are collected across all projects and over long periods of time. Their intent is to provide a set of process indicators that lead to long-term software process improvement.
- ***Project metrics*** enable a software project manager to

  (1) assess the status of an ongoing project,

  (2) track potential risks,

  (3) uncover problem areas before they go "critical,"

  (4) adjust work flow or tasks, and

  (5) evaluate the project team's ability to control quality of software work products.

What are Metrics?

- Software process and project metrics are quantitative measures
- They are a management tool
- They offer insight into the effectiveness of the software process and the projects that are conducted using the process as a framework
- Basic quality and productivity data are collected
- These data are analyzed, compared against past averages, and assessed
- The goal is to determine whether quality and productivity improvements have occurred
- The data can also be used to pinpoint problem areas
- Remedies can then be developed and the software process can be improved

Metrics in the Process Domain

- Process metrics are collected across all projects and over long periods of time
- They are used for making strategic decisions
- The intent is to provide a set of process indicators that lead to long-term software process improvement
- The only way to know how/where to improve any process is to

  -Measure specific attributes of the process

  -Develop a set of meaningful metrics based on these attributes

  -Use the metrics to provide indicators that will lead to a strategy for improvement

Metrics in the Process Domain(Contd..)

- We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as

  -Errors uncovered before release of the software

  -Defects delivered to and reported by the end users

  -Work products delivered

  -Human effort expended

  -Calendar time expended

  -Conformance to the schedule

  -Time and effort to complete each generic activity

Etiquette of Process Metrics

- Use common sense and organizational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who collect measures and metrics
- Don’t use metrics to evaluate individuals
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Never use metrics to threaten individuals or teams
- Metrics data that indicate a problem should not be considered “negative”

  -Such data are merely an indicator for process improvement
- Don’t obsess on a single metric to the exclusion of other important metrics

Metrics in the Project Domain

- Project metrics enable a software project manager to Assess the status of an ongoing project
- Track potential risks
- Uncover problem areas before their status becomes critical
- Adjust work flow or tasks
- Evaluate the project team's ability to control quality of software work products
- Many of the same metrics are used in both the process and project domain
- Project metrics are used for making tactical decisions
- They are used to adapt project workflow and technical activities

Use of Project Metrics

- The first application of project metrics occurs during estimation
- Metrics from past projects are used as a basis for estimating time and effort
- As a project proceeds, the amount of time and effort expended are compared to original estimates
- As technical work commences, other project metrics become important
- Production rates are measured (represented in terms of models created, review hours, function points, and delivered source lines of code)
- Error uncovered during each generic framework activity (i.e, communication, planning, modeling, construction, deployment) are measured
- Project metrics are used to
- Minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- Assess product quality on an ongoing basis and, when necessary, to modify the technical approach to improve quality

Software Measurement

Two categories of software measurement

- ***Direct measures*** of the

  -Software process (cost, effort, etc.)

  -Software product (lines of code produced, execution speed, defects reported over time, etc.)

- ***Indirect measures*** of the

  -Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)

- Software metrics domains are partitioned into process , project and product
- Product metrics are private to an individual are often combined to develop project metrics
- Project metrics can be consolidated to create process metrics for an organization

Software Measurement

- **Size-Oriented Metrics**
- **Function-Oriented Metrics**
- **Reconciling LOC and FP Metrics**
- **Object-Oriented Metrics**
- **Use Case –Oriented Metrics**
- **Web App Project Metrics**

Size-Oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the size of the software produced
- Thousand lines of code (KLOC) are often chosen as the normalization value
- Metrics include

  -Errors per KLOC

  -Defects per KLOC

  -Dollars per KLOC

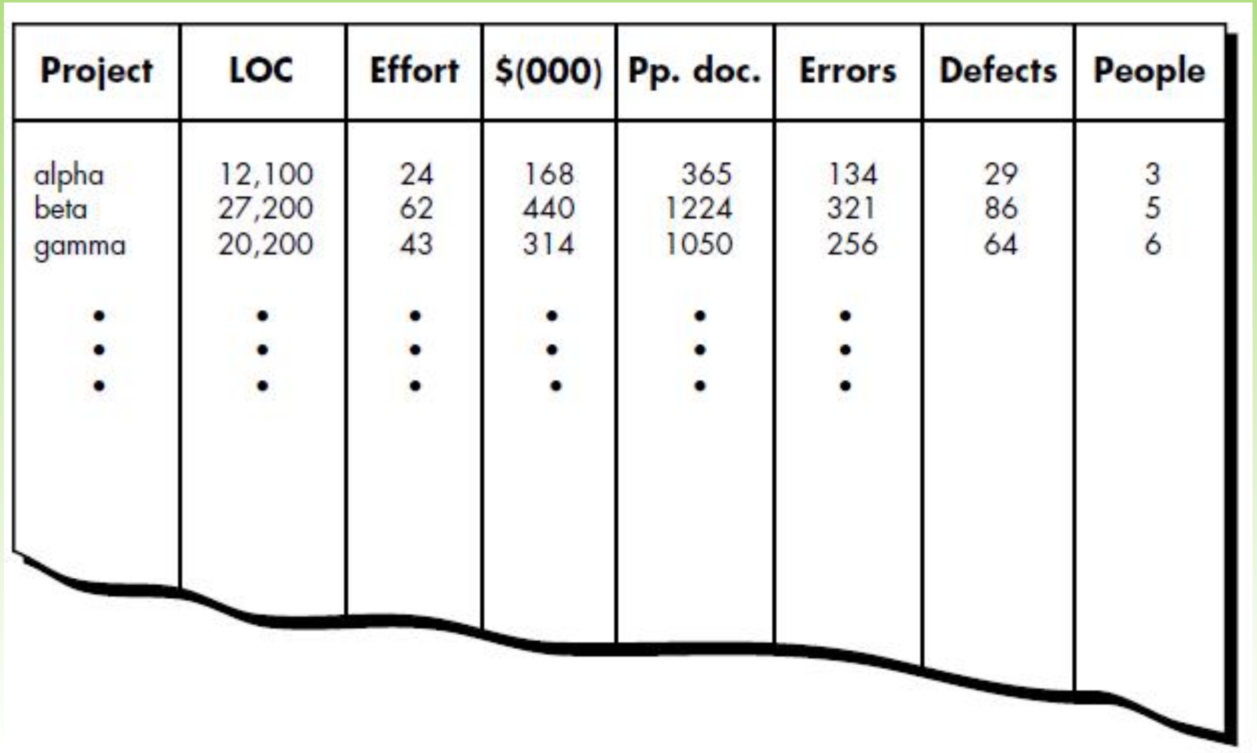
  -Errors per person-month

  -KLOC per person-month

  -Dollars per page of documentation

  -Pages of documentation per KLOC

Size-Oriented Metrics



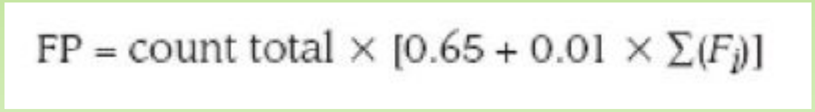
Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

Size-Oriented Metrics

- Size-oriented metrics are not universally accepted as the best way to measure the software process
- Opponents argue that KLOC measurements
- Are dependent on the programming language
- Penalize well-designed but short programs
- Cannot easily accommodate nonprocedural languages
- Require a level of detail that may be difficult to achieve

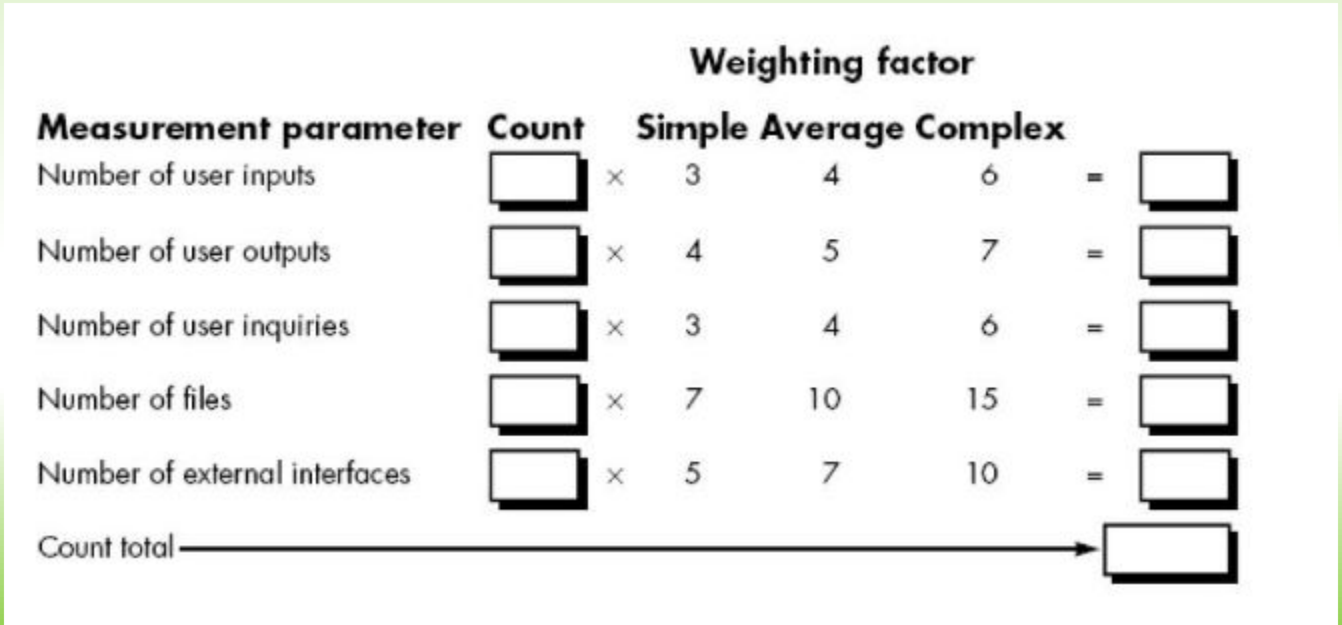
Function Oriented Metrics

- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value
- Most widely used metric of this type is the function point



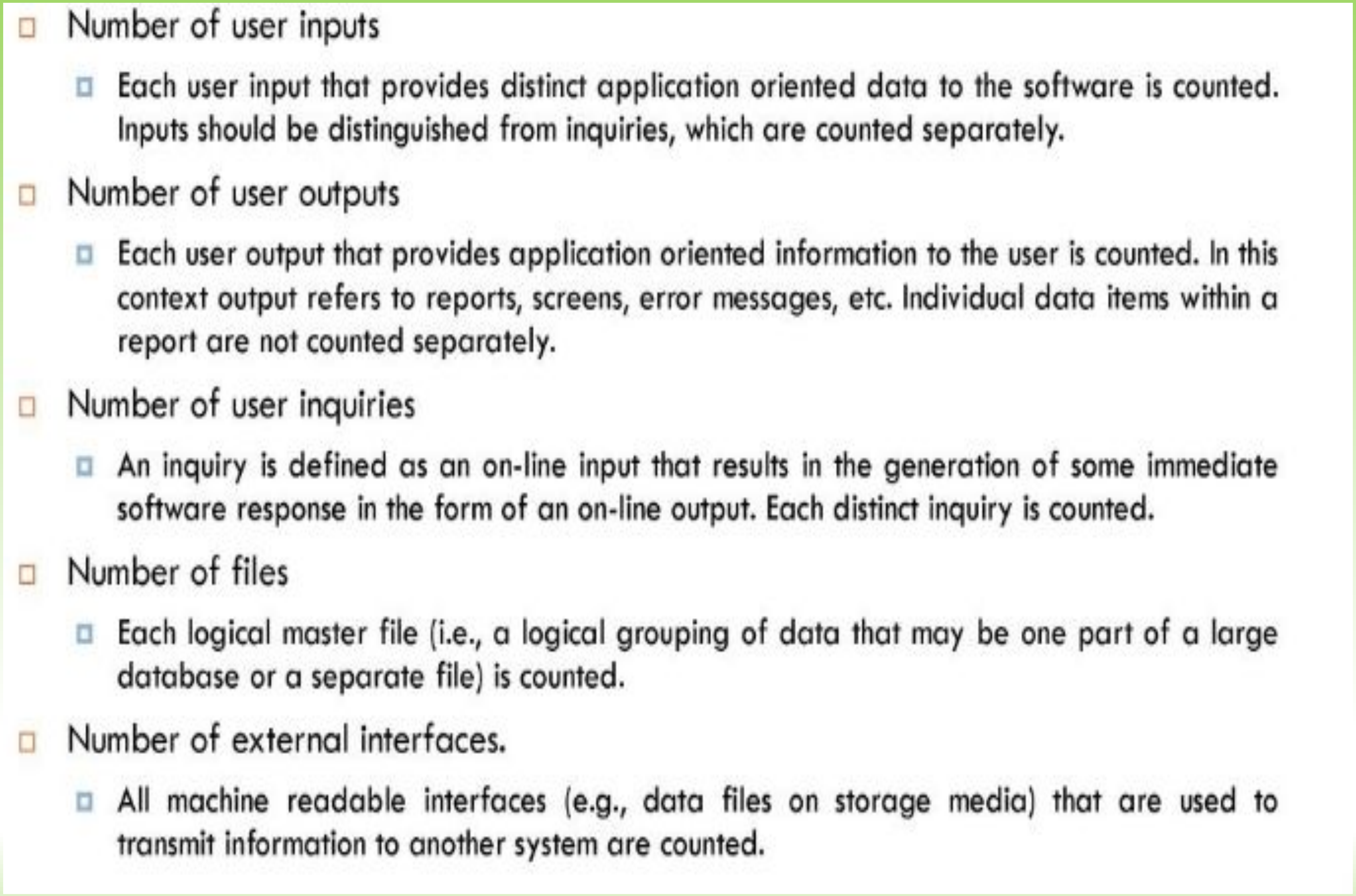
$$FP = \text{count total} \times [0.65 + 0.01 \times \sum(F_i)]$$

Function point values on past projects can be used to compute, for example, the average number of lines of code per function point



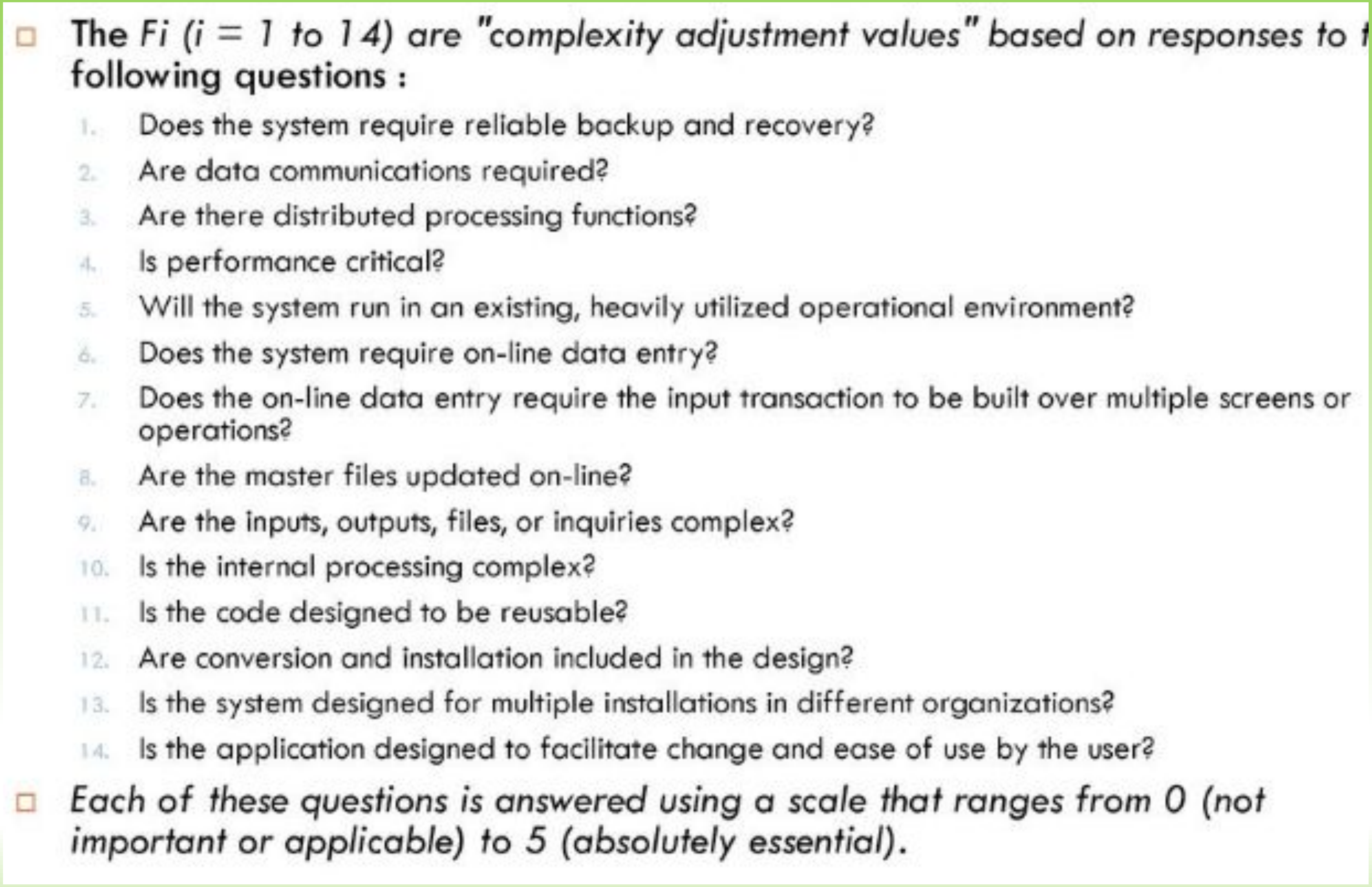
			Weighting factor				
Measurement parameter	**Count**		**Simple**	**Average**	**Complex**		
Number of user inputs	☐	×	3	4	6	=	☐
Number of user outputs	☐	×	4	5	7	=	☐
Number of user inquiries	☐	×	3	4	6	=	☐
Number of files	☐	×	7	10	15	=	☐
Number of external interfaces	☐	×	5	7	10	=	☐
Count total →							☐

Function Oriented Metrics



- Number of user inputs
  - Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.
- Number of user outputs
  - Each user output that provides application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
- 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 (i.e., a logical grouping of data that may be one part of a large database or a separate 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.

Function Oriented Metrics



- The *Fi* (*i* = 1 to 14) are "complexity adjustment values" based on responses to t
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?
  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?
- *Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).*

Function Oriented Metrics

Function Point Controversy

Like the KLOC measure, function point use also has proponents and opponents

Proponents claim that

FP is programming language independent

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

Opponents claim that

FP requires some “sleight of hand” because the computation is based on subjective data

Counts of the information domain can be difficult to collect after the factFP has no direct physical meaning…it’s just a number

Reconciling LOC and FP Metrics

Relationship between LOC and FP depends upon

-The programming language that is used to implement the software

-The quality of the design

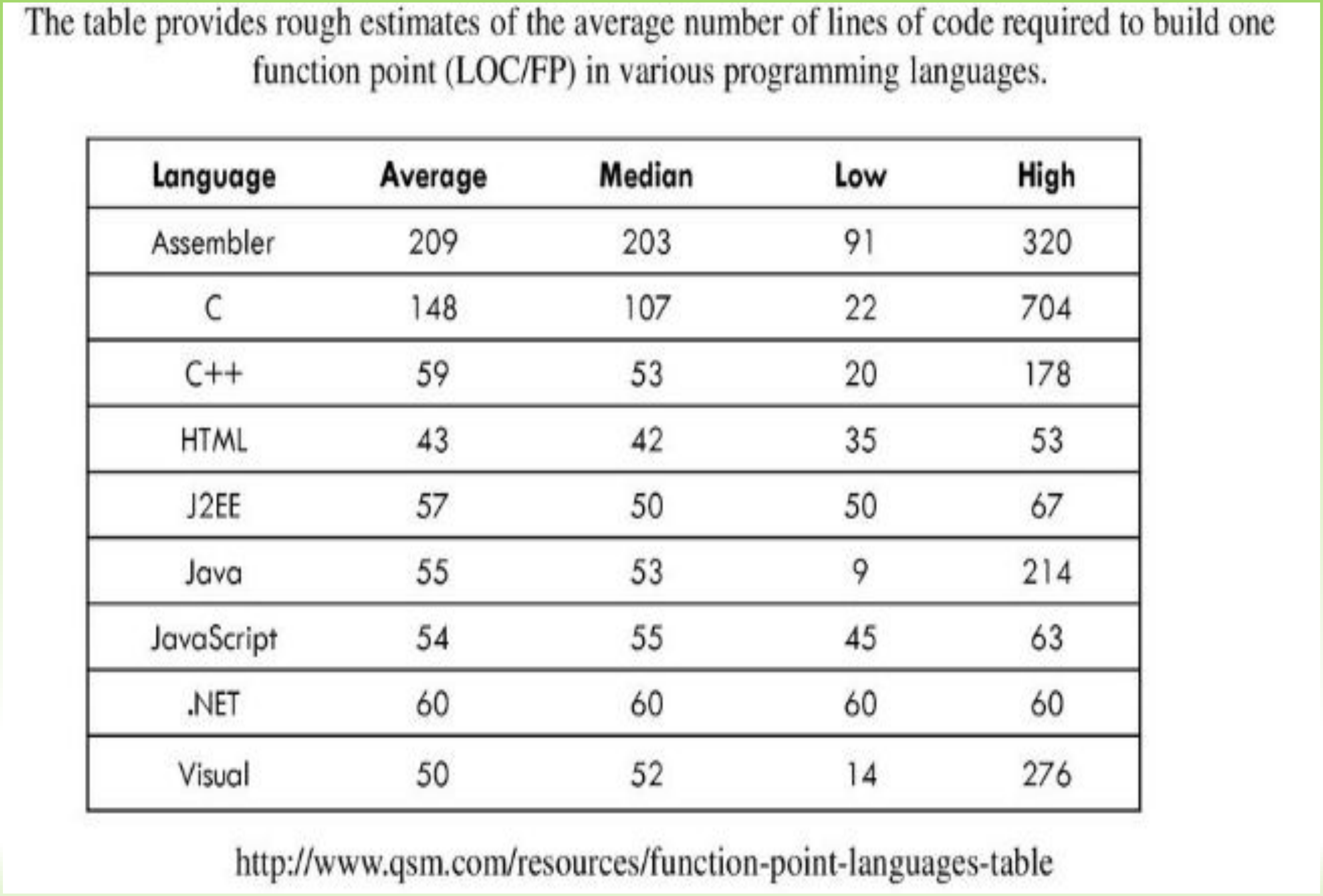
- FP and LOC have been found to be relatively accurate predictors of software development effort and cost

  -However, a historical baseline of information must first be established

- LOC and FP can be used to estimate object-oriented software projects

  -However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an evolutionary or incremental process

LOC per Function Point



The table provides rough estimates of the average number of lines of code required to build one function point (LOC/FP) in various programming languages.

Language	Average	Median	Low	High
Assembler	209	203	91	320
C	148	107	22	704
C++	59	53	20	178
HTML	43	42	35	53
J2EE	57	50	50	67
Java	55	53	9	214
JavaScript	54	55	45	63
.NET	60	60	60	60
Visual	50	52	14	276

http://www.qsm.com/resources/function-point-languages-table

Object Oriented Metrics

Number of use cases

This number is directly related to the size of an application and to the number of test cases required to test the system

Number of key classes (the highly independent components)

-Key classes are defined early in object-oriented analysis and are central to the problem domain

-This number indicates the amount of effort required to develop the software

-It also indicates the potential amount of reuse to be applied during development

Number of support classes

-Support classes are required to implement the system but are not immediately related to the problem domain (e.g., user interface, database, computation)

-This number indicates the amount of effort and potential reuse

Object Oriented Metrics

Average number of support classes per key class

- Key classes are identified early in a project (e.g., at requirements analysis)
- Estimation of the number of support classes can be made from the number of key classes
- GUI applications have between two and three times more support classes as key classes
- Non-GUI applications have between one and two times more support classes as key classes

Number of subsystems

- A subsystem is an aggregation of classes that support a function that is visible to the end user of a system

Use Case Oriented Metrics

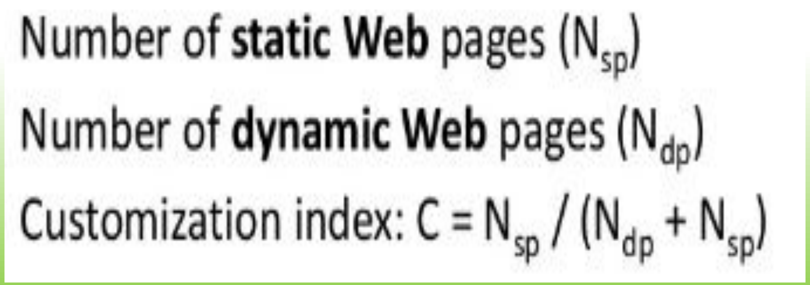
- Use Cases are widely used as a method for describing customer level or business domain requirements that imply software features and functions
- Use case is defined early in the software process, allowing it to be used for estimation before significant modeling and construction activities are initiated
- Use cases describe user visible functions and features that are basic requirements for a system
- The use case is independent of programming language
- The number of use cases is directly proportional to the size of the application in LOC and to the number of test cases
- ***Use Case Points(UCPs)*** as a mechanism for estimating project effor and other characteristics
- ***UCP is a function of the number of actors and transactions implied by the use case models***

WebApp Project Metrics

The objective of all web App projects is to deliver a combination of content and functionality to the end user. Various measures that can be collected are:

1. Number of static web pages
2. Number of dynamic web pages
3. Number of internal page links
4. Number of persistent data objects
5. Number of external system interfaced
6. Number of static content objects
7. Number of dynamic content objects
8. Number of executable functions

Metric *that reflect the degree of end user customization that is required for the WebApp and correlate it to the effort expended on the project and/or the errors uncovered as reviews and testing are conducted.*



Number of **static Web** pages (N_{sp})

Number of **dynamic Web** pages (N_{dp})

Customization index: $C = N_{sp} / (N_{dp} + N_{sp})$

Metrics for Software Quality

Correctness

- This is the number of defects per KLOC, where a defect is a verified lack of conformance to requirements
- Defects are those problems reported by a program user after the program is released for general use

Maintainability

- This describes the ease with which a program can be corrected if an error is found, adapted if the environment changes, or enhanced if the customer has changed requirements
- Mean time to change (MTTC) : the time to analyze, design, implement, test, and distribute a change to all users

  Maintainable programs on average have a lower MTTC

Metrics for Software Quality

Usability.

is an attempt to quantify user-friendliness and can be measured in terms of four characteristics:

1. The physical and or intellectual skill required to learn the system,
2. The time required to become moderately efficient in the use of the system,
3. The net increase in productivity (over the approach that the system replaces) measured when the system is used by someone who is moderately efficient, and
4. A subjective assessment (sometimes obtained through a questionnaire) of users attitudes toward the system.

Metrics for Software Quality

Integrity:

measures a system's ability to withstand attacks (both accidental and intentional) to its security.

Attacks can be made on all three components of software: programs, data, and documents.

To measure integrity, two additional attributes must be defined:

1. ***Threat*** is the probability (which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time.
2. ***Security*** is the probability (which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled.

The integrity of a system can then be defined as

integrity = summation [(1 – threat) X (1 – security)]where threat and security are summed over each type of attack.

Defect Removal Efficiency

- Defect removal efficiency provides benefits at both the project and process level
- It is a measure of the <u>filtering ability </u>of quality assurance activities as they are applied throughout all process framework activities

  --It indicates the percentage of software errors found before software release

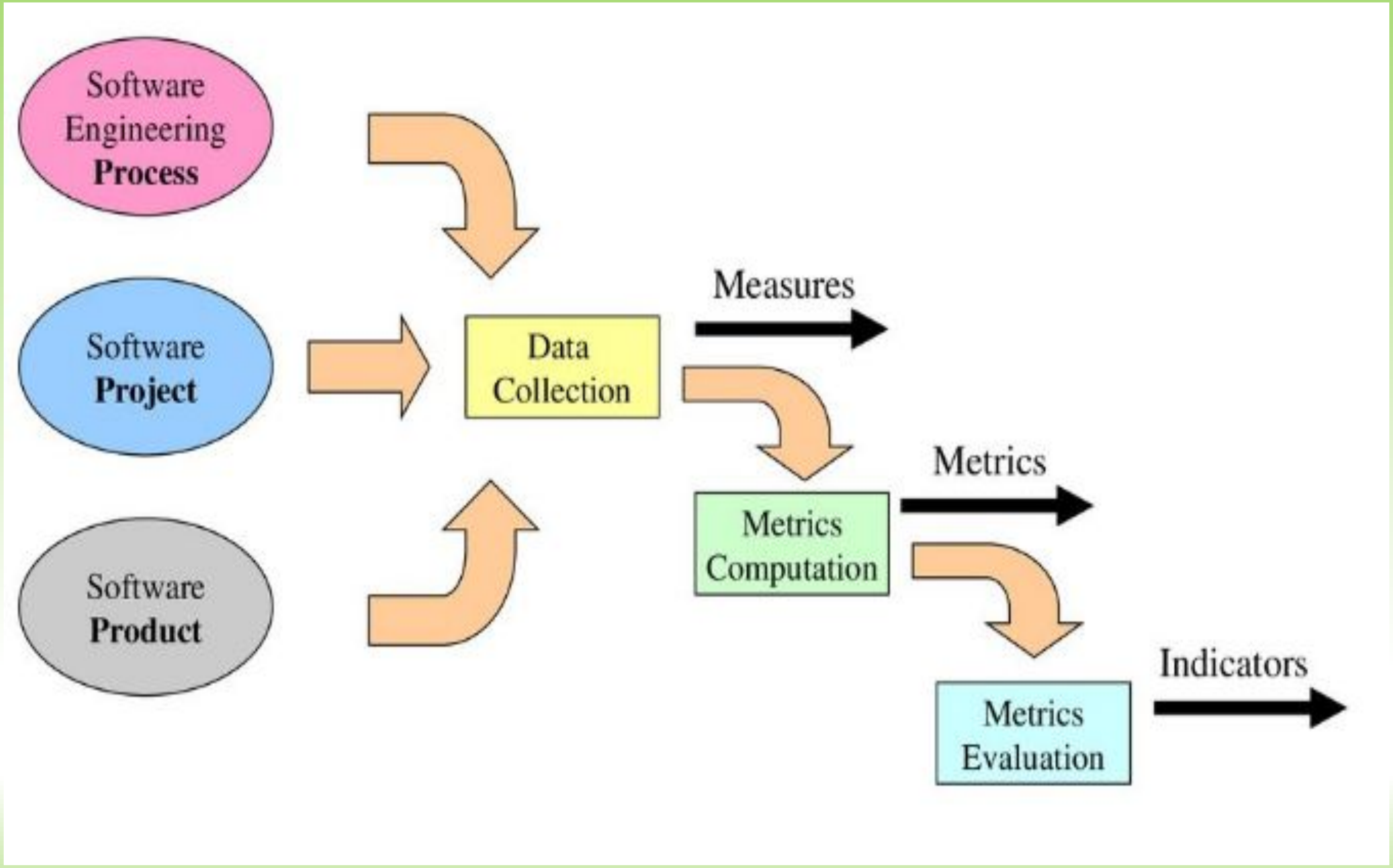
- It is defined as DRE = E / (E + D)

  --E is the number of errors found <u>before</u> delivery of the software to the end user

  --D is the number of defects found <u>after</u> delivery

- As D <u>increases</u>, DRE <u>decreases</u> (i.e., becomes a smaller and smaller fraction)
- The ideal value of DRE is 1, which means no defects are found after delivery
- DRE encourages a software team to institute techniques for finding <u>as many</u>

Software Metrics Baseline Process



Estimation For Software Projects

Estimation for Software Projects

- Project Planning
- Scope and feasibility
- Project Resources
- Estimation of project cost and effort
- Decomposition techniques
- Empirical estimation models

Software Project Planning

- Software Project Planning encompasses 5 major activities

  Estimation, Scheduling , risk analysis ,quality management planning , and change management planning

- Estimation determines how much money , effort, resources , and time it will take to build a specific system or product
- The software team first estimates

  --The work need to be done

  --The resources required

  --The time that will elapse from start to finish
- Then they establish a project schedule that

  --Defines task and milestones

  -Identify who is responsible for conducting each task

  --Specifies the inner task dependencies

Observations on Estimation

- Planning requires technical managers and the software team to make an initial commitment
- Process and project metrics can provide a historical perspective and valuable input for generation of quantitative estimates
- Past experience can aid greatly
- Estimation carries inherent risk, and this risk leads to uncertainty
- The availability of historical information has a strong influence on estimation risk
- When software metrics are available from past projects
  - Estimates can be made with greater assurance
  - Schedules can be established to avoid past difficulties
  - Overall risk is reduced
- Estimation risk is measured by the degree of uncertainty in the quantitative estimates for cost, schedule, and resources
- Nevertheless, a project manager should not become obsessive about estimation

  --Plans should be iterative and allow adjustments as time passes and more is made certain

Task Set for Project Planning

1) ***Establish project scope***
2) ***Determine feasibility***
3) ***Analyze risks***
4) ***Define required resources***
   - Determine human resources required
   - Define reusable software resources
   - Identify environmental resources
5) ***Estimate cost and effort***
   - Decompose the problem
   - Develop two or more estimates using different approaches
   - Reconcile the estimates
6) ***Develop a project schedule***
   - Establish a meaningful task set
   - Define a task network
   - Use scheduling tools to develop a timeline chart
   - Define schedule tracking mechanisms

Software Scope

- Software Scope defines
  - ✔ The functions and features that are to be delivered to end users
  - ✔ The data that are input to and output from the system
  - ✔ The "content" that is presented to users as a consequence of using the software
  - ✔ The performance, constraints, interfaces, and reliability that bound the system
- Scope can be define using two techniques
  - ✔ A narrative description of software scope is developed after communication with all stakeholders
  - ✔ A set of use cases is developed by end users
- After the scope has been identified, two questions are asked
  - ✔ Can we build software to meet this scope?
  - ✔ Is the project feasible?

Feasibility

- After the Scope is resolved, feasibility is addressed
- Software feasibility has four dimensions
  - **Technology** – Is the project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?
  - **Finance** – Is financially feasible? Can development be completed at a cost that the software organization, its client, or the market can afford?
  - **Time** – Will the project's time-to-market beat the competition?
  - **Resources** – Does the software organization have the resources needed to succeed in doing the project?

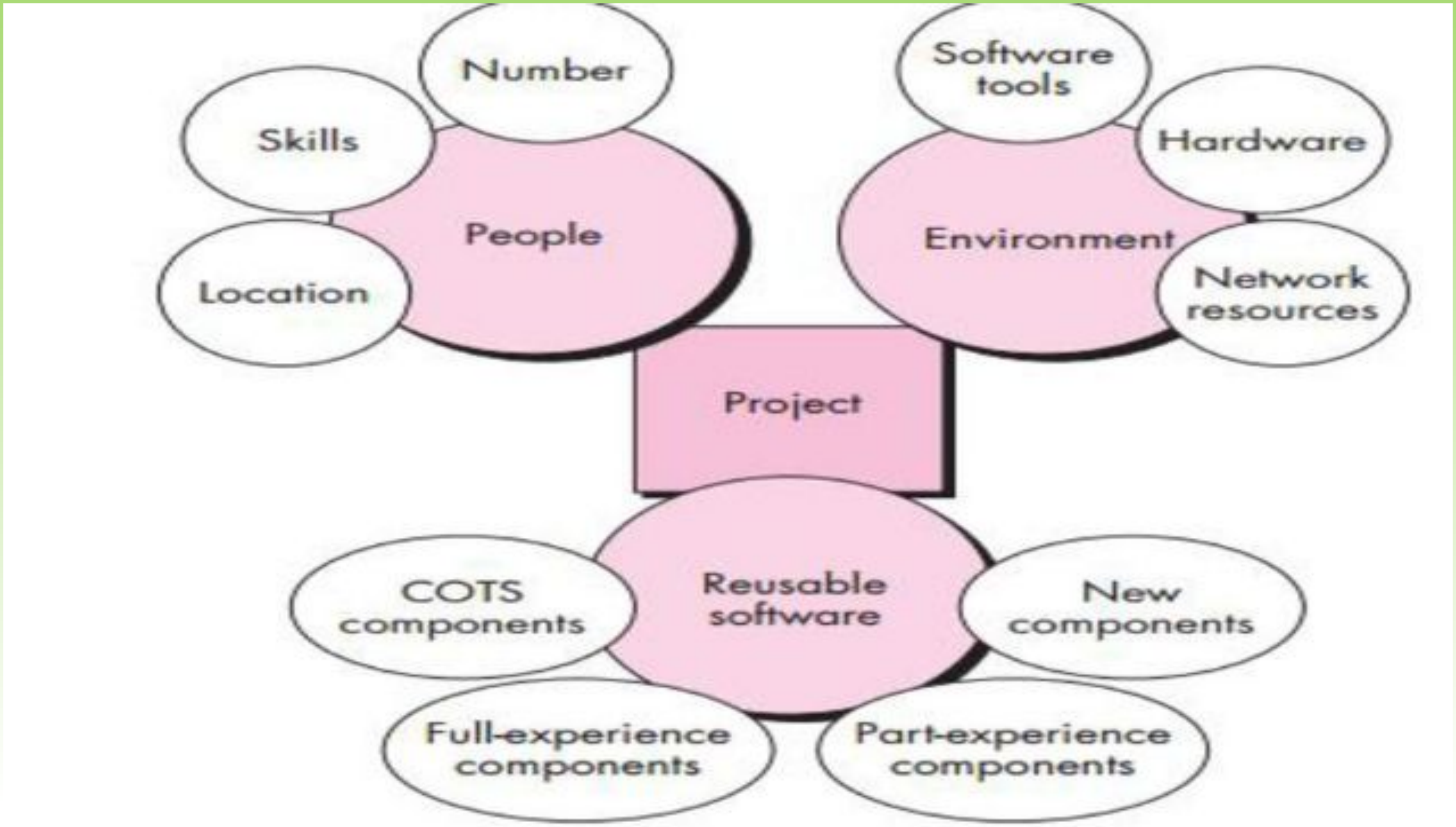
Resource Estimation

- Three major categories of software engineering resources
  - ✔ People
  - ✔ Development environment
  - ✔ Reusable software components

Often neglected during planning but become a paramount concern during the construction phase of the software process

- Each resource is specified with
  - ✔ A description of the resource
  - ✔ A statement of availability
  - ✔ The time when the resource will be required
  - ✔ The duration of time that the resource will be applied

Categories of Resource



Human Resources

- Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Small projects of a few person-months may only need one individual
- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

Development Environment Resources

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products
- Most software organizations have many projects that require access to the SEE provided by the organization
- Planners must identify the time window required for hardware and software and verify that these resources will be available

Reusable Software Resources

- ***Off-the-shelf components***

  -Components are from a third party or were developed for a previous project

  -Ready to use;

  -fully validated and documented;

  -virtually no risk

- ***On-the-shelf components***

  -Components are similar to the software that needs to be built

  -Software team has full experience in the application area of these components

  -Modification of components will incur relatively low risk

- ***Partial-experience components***

  -Components are related somehow to the software that needs to be built but will require substantial modification

  -Software team has only limited experience in the application area of these components

  -Modifications that are required have a fair degree of risk

- ***New components***

  -Components must be built from scratch by the software team specifically for the needs of the current project

  -Software team has no practical experience in the application area

  -Software development of components has a high degree of risk

Software Project Estimation

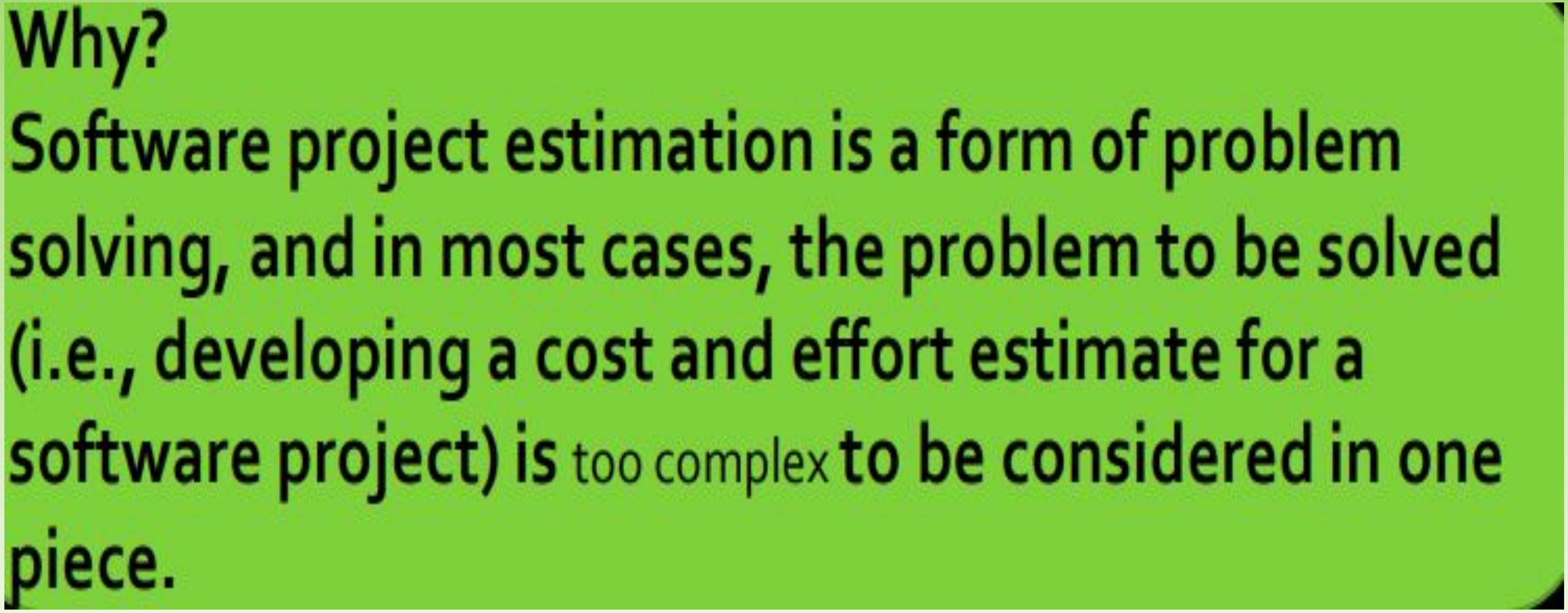
- To achieve reliable cost and effort estimates , a number of options arise

1. Delay estimation until late in the project (we should be able to achieve 100% accurate estimates after the project is complete)
2. Base estimates on similar projects that have already been completed
3. Use relatively simple decomposition techniques to generate project cost and effort estimates
4. Use one or more empirical estimation models for software cost and effort estimation

Project Estimation Approaches

- ***Decomposition techniques***
  - These take a "divide and conquer" approach
  - Cost and effort estimation are performed in *a stepwise fashion* by breaking down a project into major functions and related software engineering activities
- ***Empirical estimation models***
  - Offer a potentially valuable estimation approach if *the historical data used to seed the estimate* is good

Decomposition Techniques



Why?

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece.

Decomposition techniques

- ***Problem – Based Estimation***

  -LOC-Based Estimation

  -FP Based Estimation

Problem Based Estimation

LOC and FP are used in two ways during software project Estimation

1. As estimation variables to “size” each element of the software
2. As baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projection

- LOC and FP estimation are separate estimation technique
- LOC or FP(the estimation variable) is estimated for each function
- Yet both have a number of characteristics in common

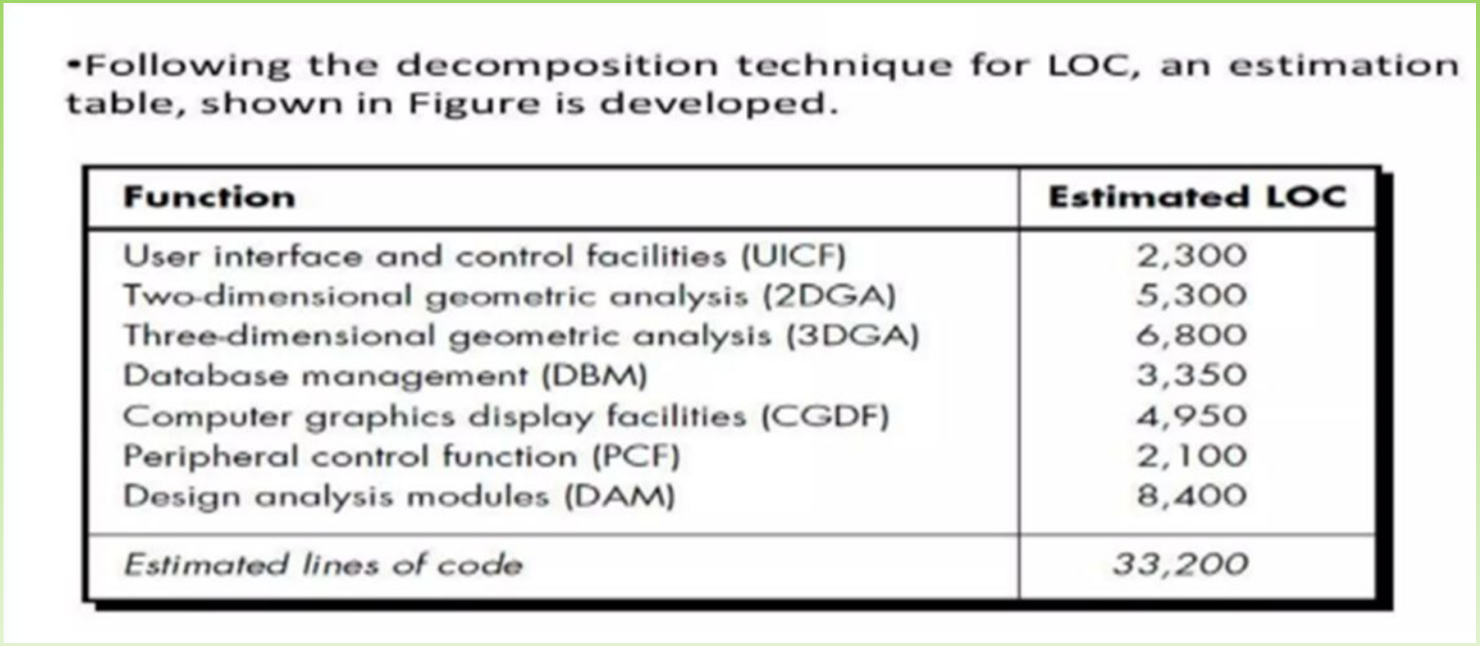
Problem Based Estimation

- When LOC is used as the estimation variable , decomposition is absolutely essential and is often taken to significant levels of details . The greater the degree of subdividing, the more like accurately estimates of LOC can be developed
- For FP estimates, decomposition works differently . Rather than focusing on function , each of the information variety characteristics inputs, outputs , data files , inquiries and external interfaces
- ***A three – point or expected value can then be computed***. The expected value for the estimation variable (size) Scan be computed as a weighted average of the optimistic (sopt),most likely (sm), and pessimistic (spess) estimates. For example,
- S=(sopt+4sm+spess)/6

An Example of LOC Based Estimation

- Example Scenario: Software package to be developed for a CAD application for mechanical components
- Following the decomposition technique for LOC, an estimation table is developed. A range of ***LOC estimates is developed for each function***.
- For example, the range of LOC estimates for the 3D geometric analysis is optimistic, 4600 LOC;most likely , 6900 LOC; and pessimistic , 8600 LOC
- By using the equation
- S=(sopt+4sm+spess)/6, the expected value for the 3D geometric analysis function is 6800 LOC.

An Example of LOC Based Estimation

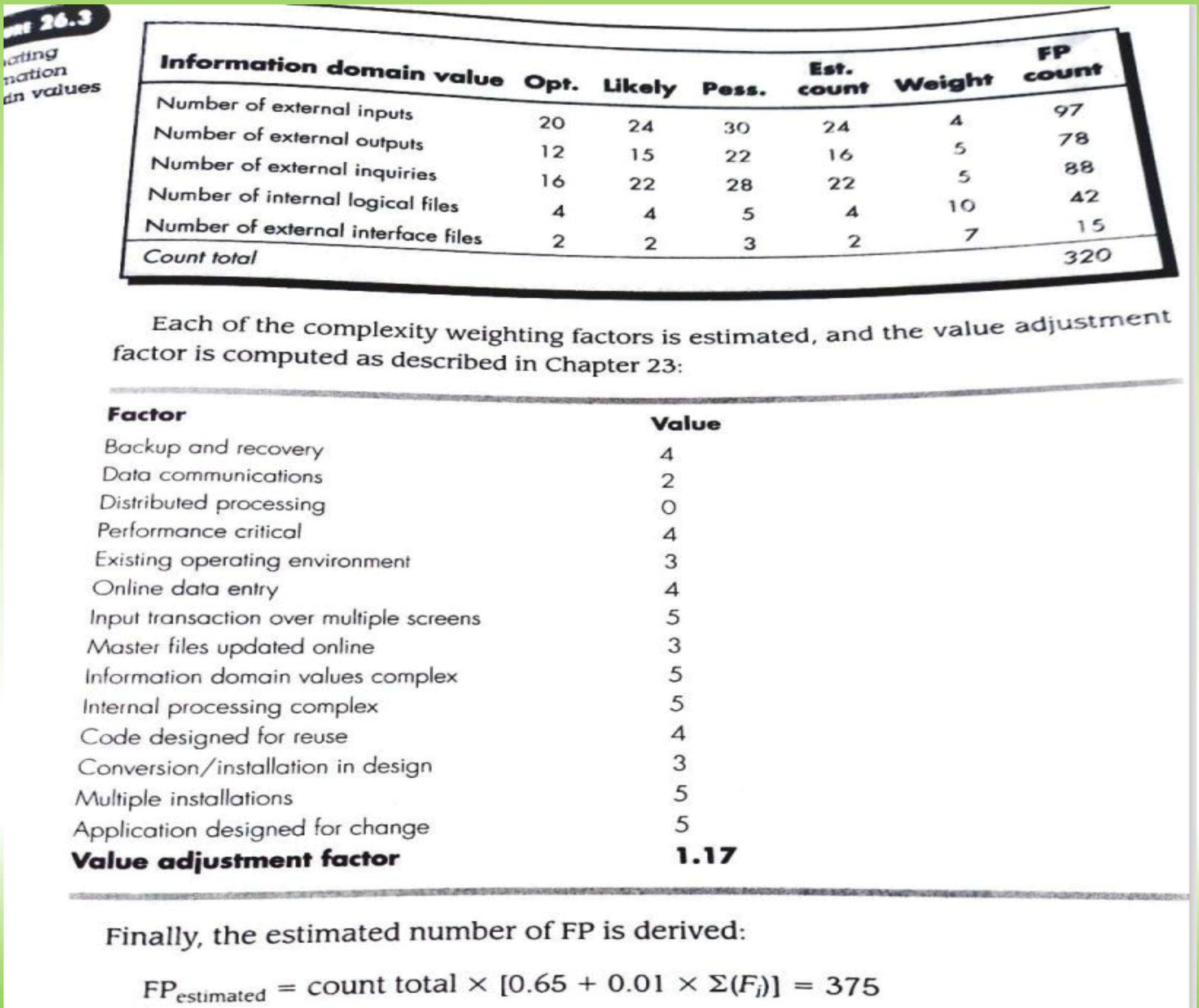


•Following the decomposition technique for LOC, an estimation table, shown in Figure is developed.

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

- Average Productivity for systems of this type is 620 LOC/pm
- Based on a burdened labor rate of $8000 per month, the cost per LOC is approximately $13.
- Based on the LOC estimates and the historical productivity data, the total estimated project cost is $431,000
- The estimated effort is 54 persons - months

An Example of FP Based Estimation



26.3
cting
nation
in values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

Each of the complexity weighting factors is estimated, and the value adjustment factor is computed as described in Chapter 23:

Factor	**Value**
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Value adjustment factor	**1.17**

Finally, the estimated number of FP is derived:

$$FP_{estimated} = \text{count total} \times [0.65 + 0.01 \times \Sigma(F_i)] = 375$$

An Example of FP Based Estimation

- Average Productivity for systems of this type is 6.5 FP/pm
- Based on a burdened labor rate of $8000 per month, the cost per FPis approximately $1230.
- Based on the FP estimates and the historical productivity data, the total estimated project cost is $461,000
- The estimated effort is 58 persons - months

The COCOMO model

- An empirical model based on project experience
- Well-documented, ‘independent’ model which is not tied to a specific software vendor
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 81

The basic COCOMO equations take the form

Effort Applied (E) = $a_b(\text{KLOC})^{b}{}_{b}$ [man-months]

Development Time (D) = $c_b(\text{Effort Applied})^{d}{}_{b}$ [months]

People required (P) = Effort Applied / Development Time [count]

Software project	a_b	b_b	c_b	d_b
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

Development Mode **Project Characteristics**

	Size	**Innovation**	**Deadline**	**Dev. Environment**
ORGANIC	Small	Little	Not Tight	Stable
SEMI-DITACHED	Medium	Medium	Medium	Medium
EMBEDDED	Large	Greater	Tight	Complex Hardware

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

- Solution:
  - Effort = 2.4 * (32)1.05 = 91 PM
  - Development time = 2.5 * (91)0.38 = 14 months
  - Cost required to develop the product = 14 * 15,000
  - = Rs. 210,000/-

COCOMO-II

- COCOMO II is actually a hierarchy of estimation models that address the following areas:
  - *Application composition model.* Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - *Early design stage model.* Used once requirements have been stabilized and basic software architecture has been established.
  - *Post-architecture-stage model.* Used during the construction of the software.

The Software Equation

A dynamic multivariable model

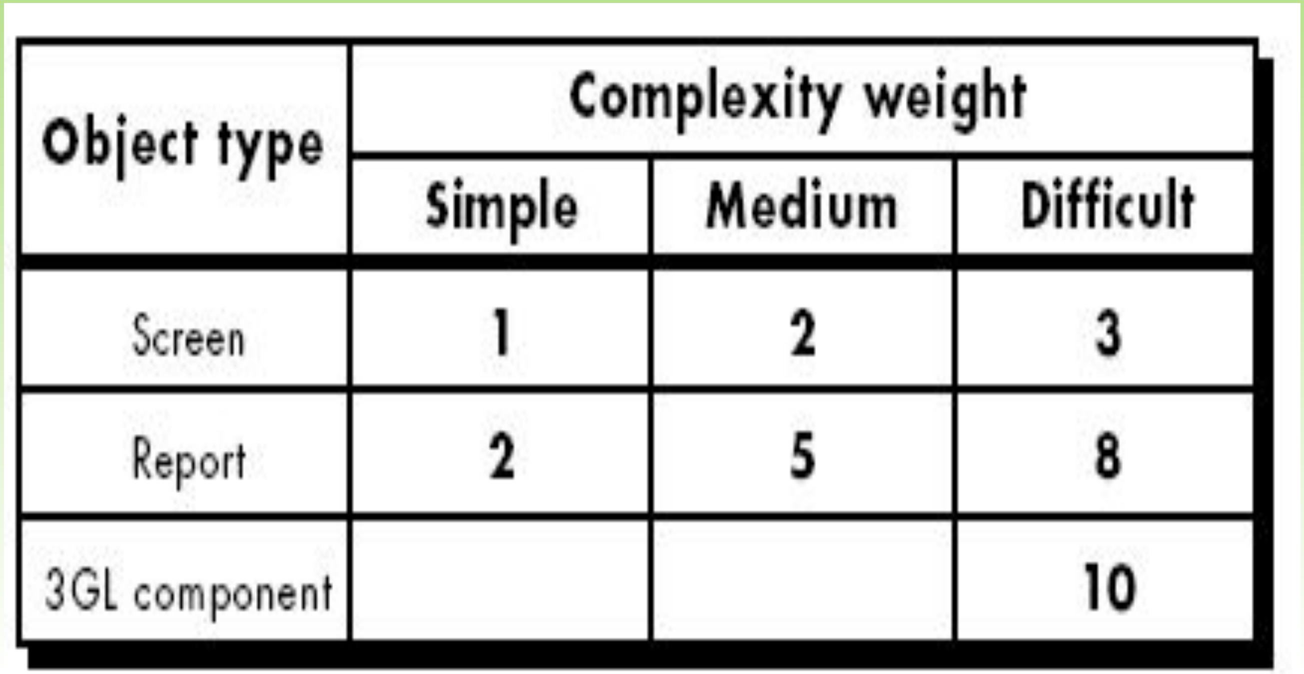
$$\mathbf{E = [LOC \times B^{0.333} / P]^3 \times (1 / t^4)}$$

where

- E = effort in person-months or person-years
- t = project duration in months or years
- B = “special skills factor”
- P = “productivity parameter”

- Three different sizing options are available as part of the model hierarchy:
  - Object points
  - Function points
  - Lines of source code.
- The COCOMO II application composition model ***uses object points***. Estimation models (using FP and KLOC) are also available as part of COCOMO II.
- Like function points, the *object point* is an indirect software measure that is computed using counts of the number of
    - Screens (at the user interface),
    - Reports, and
    - Components likely to be required to build the application.
- Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e., simple, medium, or difficult)

Object Points Weights



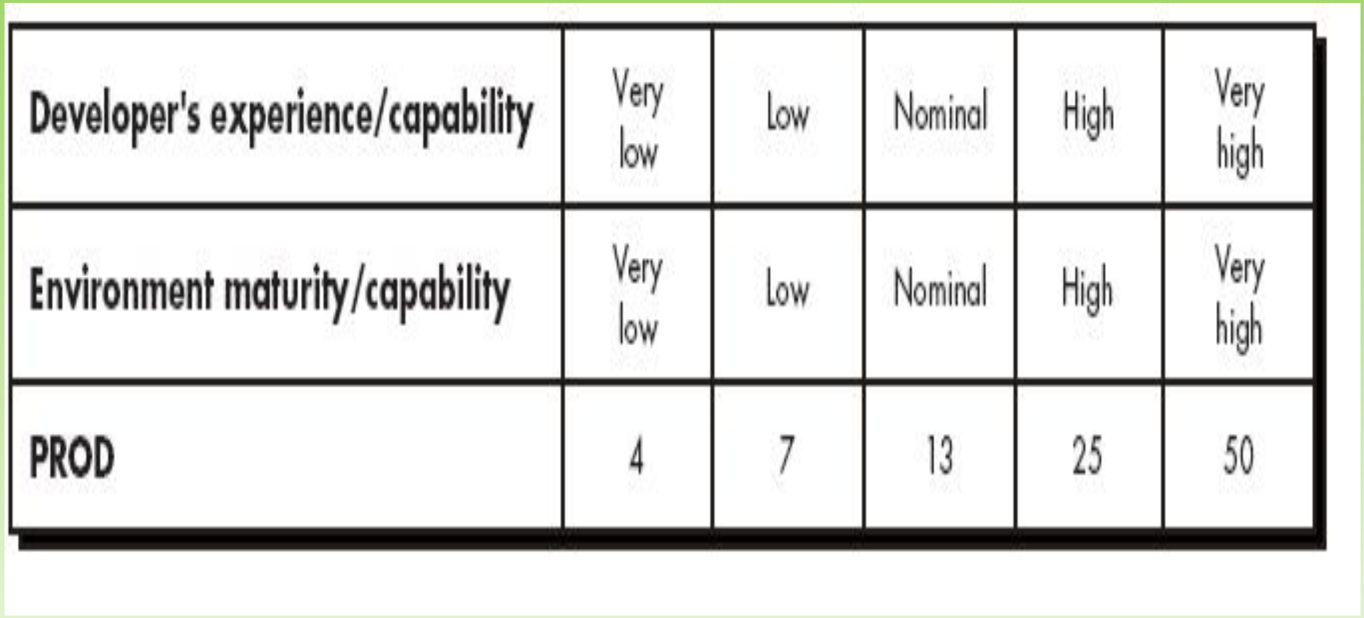
Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

- Complexity weight can be measure on source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report.
- Once complexity is determined, the number of screens, reports, and components are weighted according to Table.
- Object point count is then determined by multiplying the original number of object instances by the weighting factor in Table and summing to obtain a total object point count.
- When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

  NOP = (object points) x [(100 - %reuse)/100]

  NOP is defined as new object points.
- To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived.

  PROD = NOP/person-month



Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

- Table represents the productivity rate for different levels of developer experience and development environment maturity.
- Once the productivity rate has been determined, an estimate of project effort can be derived as

Estimated effort = NOP/PROD

PROBLEMS

- Estimate the effort required for the software development for ATM which has 2 screens, 8 reports and 15 software components. Assume average complexity and average developer maturity.
  - **OBJ POINTS = 2*2+8*5+15*10 =194**
  - **Effort = 194/13 = 14.92 ~ 15 PM**

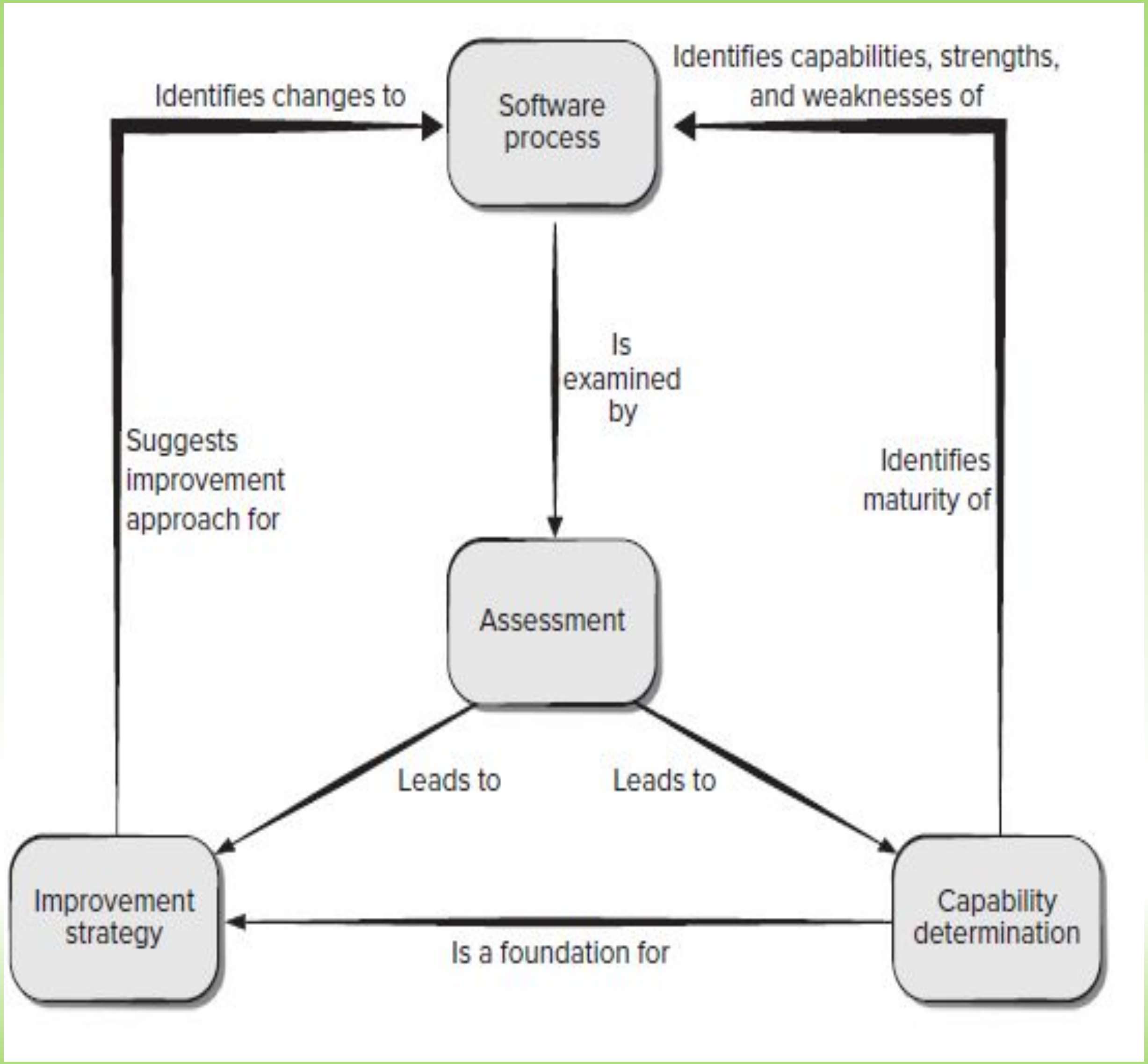
Software Process Improvement

An *SPI framework* defines

- a set of characteristics that must be present if an effective software process is to be achieved,
- a method for assessing whether those characteristics are present,
- a mechanism for summarizing the results of any assessment, and
- a strategy for assisting a software organization in implementing those process characteristics that have been found to be weak or missing.

Software Process Improvement

An *SPI framework*



Software Process Improvement

The SPI Process

- **Assessment and Gap Analysis**
- **Education and Training**
- **Selection and Justification**
- **Installation/Migration**
- **Evaluation**
- **Risk Management for SPI**

CMMI

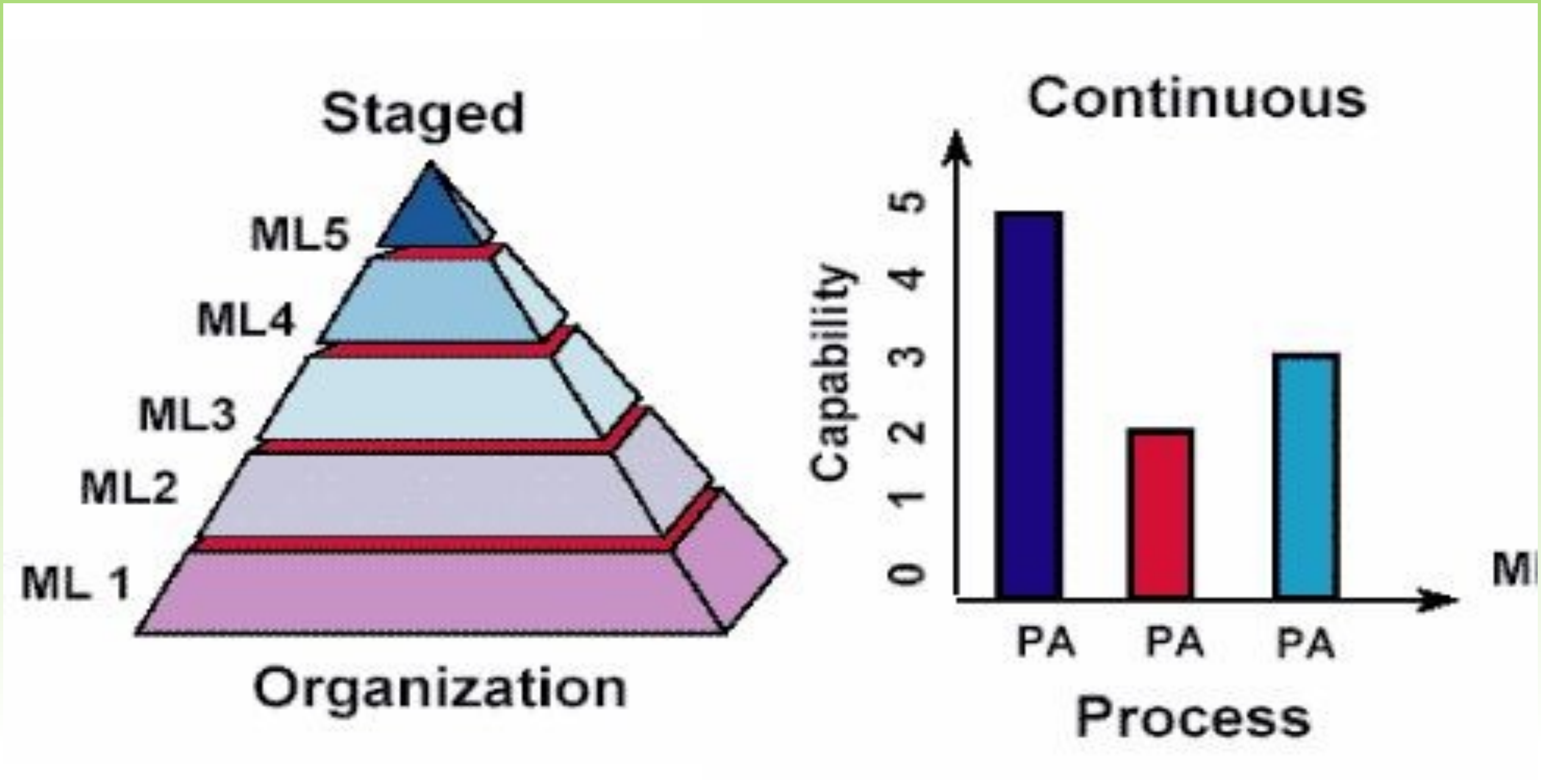
What is CMMI?

The Capability Maturity Model Integration (CMMI) is a process and behavioral model that helps organizations streamline process improvement and encourage productive, efficient behaviors that decrease risks in software, product, and service development.

Objectives of CMMI :

1. Fulfilling customer needs and expectations.
2. Value creation for investors/stockholders.
3. Market growth is increased.
4. Improved quality of products and services.
5. Enhanced reputation in Industry.

CMMI Representation



Staged Representation :

- uses a pre-defined set of process areas to define improvement path.
- provides a sequence of improvements, where each part in the sequence serves as a foundation for the next.
- an improved path is defined by maturity level.
- maturity level describes the maturity of processes in organization.
- Staged CMMI representation allows comparison between different organizations for multiple maturity levels.

Continuous Representation :

- allows selection of specific process areas.
- uses capability levels that measures improvement of an individual process area.
- Continuous CMMI representation allows comparison between different organizations on a process-area-by-process-area basis.
- allows organizations to select processes which require more improvement.
- In this representation, order of improvement of various processes can be selected which allows the organizations to meet their objectives and eliminate risks.

CMMI Model – Maturity Levels :

1. **Maturity level 1 : Initial**
   1. processes are poorly managed or controlled.
   2. unpredictable outcomes of processes involved.
   3. ad hoc and chaotic approach used.
   4. No KPAs (Key Process Areas) defined.
   5. Lowest quality and highest risk.

2. **Maturity level 2 : Managed**
   1. requirements are managed.
   2. processes are planned and controlled.
   3. projects are managed and implemented according to their documented plans.
   4. This risk involved is lower than Initial level, but still exists.
   5. Quality is better than Initial level.

3.Maturity level 3 : Defined

1. processes are well characterized and described using standards, proper procedures, and methods, tools, etc.
2. Medium quality and medium risk involved.
3. Focus is process standardization.

4. **Maturity level 4 : Quantitatively managed**
5. quantitative objectives for process performance and quality are set.
6. quantitative objectives are based on customer requirements, organization needs, etc.
7. process performance measures are analyzed quantitatively.
8. higher quality of processes is achieved.
9. lower risk

5.Maturity level 5 : Optimizing

10. continuous improvement in processes and their performance.
11. improvement has to be both incremental and innovative.
12. highest quality of processes.
13. lowest risk in processes and their performance.

CMMI Model – Capability Levels

A capability level includes relevant specific and generic practices for a specific process area that can improve the organization’s processes associated with that process area. For CMMI models with continuous representation, there are six capability levels as described below :

1. **Capability level 0 : Incomplete**
    1. incomplete process – partially or not performed.
    2. one or more specific goals of process area are not met.
    3. No generic goals are specified for this level.
    4. this capability level is same as maturity level 1.

2. **Capability level 1 : Performed**
    1. process performance may not be stable.
    2. objectives of quality, cost and schedule may not be met.
    3. a capability level 1 process is expected to perform all specific and generic practices for this level.
    4. only a start-step for process improvement.

3. Capability level 2 : Managed

1. process is planned, monitored and controlled.
2. managing the process by ensuring that objectives are achieved.
3. objectives are both model and other including cost, quality, schedule.
4. actively managing processing with the help of metrics.

4. Capability level 3 : Defined

5. a defined process is managed and meets the organization’s set of guidelines and standards.
6. focus is process standardization.

5. Capability level 4 : Quantitatively Managed

7. process is controlled using statistical and quantitative techniques.
8. process performance and quality is understood in statistical terms and metrics.
9. quantitative objectives for process quality and performance are established.

6. Capability level 5 : Optimizing

10. focuses on continually improving process performance.
11. performance is improved in both ways – incremental and innovation.
12. emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

Advantages and Disadvantages

Advantages of CMMI

1. Culture for maintaining Quality in projects starts in the mind of the junior programmers to the senior programmers and project managers
2. Centralized QMS for implementation in projects to ensure uniformity in the documentation which means less learning cycle for new resources, better management of project status and health
3. Incorporation of Software Engineering Best Practices in the Organizations as described in CMMI Model
4. Cost saving in terms of lesser effort due to less defects and less rework
5. This also results in increased Productivity
6. On-Time Deliveries
7. Increased Customer Satisfaction
8. Overall increased Return on Investment

Disadvantages of CMMI

1. CMMI-DEV is may not be suitable for every organization.
2. It may add overhead in terms of documentation.
3. May require additional resources and knowledge required in smaller organizations to initiate CMMI-based process improvement.
4. May require a considerable amount of time and effort for implementation.
5. Require a major shift in organizational culture and attitude.