Module 3

Aniket Mishra

Software Process Metrics

- Software metrics refers to a broad range of measurements for computer software.
- Software *process* and *product* metrics are quantitative measures that enable software people to gain insight into the effectiveness of the software process and the projects that are developed using the process as a framework.
- Metrics can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control.
- Metrics are also used to pinpoint problem areas so that remedies can be developed and the software process can be improved.
- Software metrics are analyzed and assessed by software managers.

What are the steps?

- We begin by defining a limited set of process, project, and product measures that are easy to collect.
- These measures are often normalized using either size- or function-oriented metrics.
- The result is analyzed and compared to past averages for similar projects performed within the organization.
- Thus, trends are assessed and conclusions are generated.

PROCESS METRICS AND PROJECT DOMAINS

- Metrics should be collected so that process and product indicators can be determined.
- *Process indicators* enable a software engineering organization to gain insight into the effectiveness of an existing process. They enable managers and practitioners to assess what works and what doesn't.
- Process metrics are collected across all projects and over long periods of time. Their intent is to provide indicators that lead to long-term software process improvement.
- *Project indicators* 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

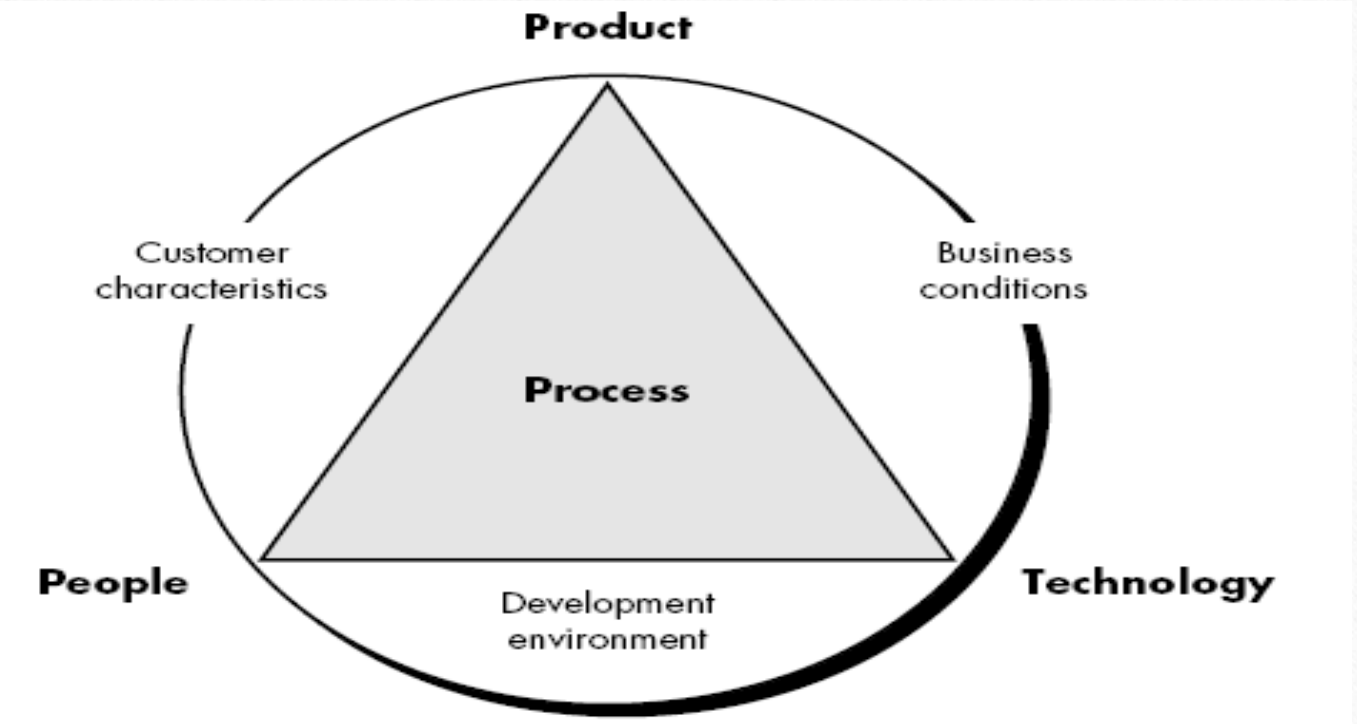
(5) Evaluate the project team's ability to control quality of software work products

I. Process Metrics and Software Process Improvement

- The only rational way to improve any process is
- Develop a set of meaningful metrics based on these attributes, and
- Then use the metrics to provide indicators that will lead to a strategy for improvement.

- Referring to Figure 4.1, process sits at the center of a triangle connecting other three factors.

I. Process Metrics and Software Process Improvement contd..



I. Process Metrics and Software Process Improvement contd..

- The skill and motivation of people has been shown to be the single most influential factor in quality and performance.
- The complexity of the product can have a substantial impact on quality and team performance.
- The technology (i.e., the software engineering methods) that populates the process also has an impact.
- In addition,
- The development environment (e.g., CASE tools),
- Business conditions (e.g., deadlines, business rules), and
- Customer characteristics (e.g., ease of communication).

Personal Software Process (PSP)

- The personal software process (PSP) is a structured set of process descriptions, measurements, and methods that can help engineers to improve their personal performance.
- It provides the forms, scripts, and standards that help them estimate and plan their work.
- It shows them how to define processes and how to measure their quality and productivity.
- A fundamental PSP principle is that everyone is different and that a method that is effective for one engineer may not be suitable for another.
- The PSP thus helps engineers to measure and track their own work so they can find the methods that are best for them.

Types of Process Metrics:-

1. Private

Private Metrics should be private to the individual and serve as an indicator for the individual only.

- Examples of *private metrics* include defect rates (by individual), defect rates (by module), and errors found during development.
- Some process metrics are private to the software project team but public to all team members.
- Examples include defects reported for major software functions (that have been developed by a number of practitioners), errors found during formal technical review

2. Public

Public metrics **-generally take in information that originally was private to individuals and**

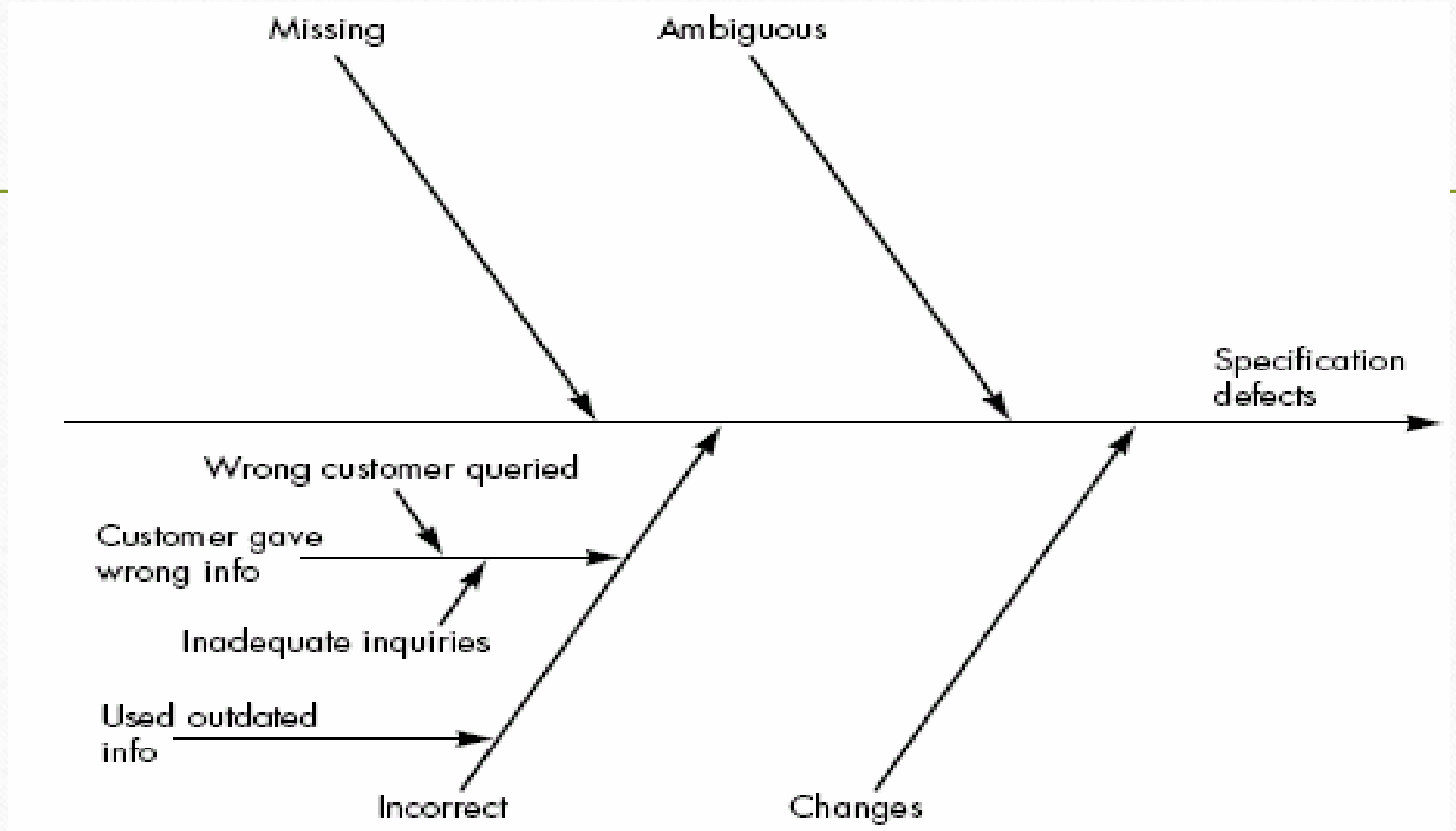
teams.

- Project level defect rates, effort, calendar

times, and related data are collected and evaluated.

A *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.


Missing
Ambiguous
Specification defects
Wrong customer queried
Customer gave wrong info
Inadequate inquiries
Used outdated info
Incorrect
Changes

The CMM levels

- To determine an organization's current state of process maturity, the SEI uses an assessment that results in a five point grading scheme. The grading scheme determines compliance with a *capability maturity model (CMM) that defines key activities required at different levels* of process maturity. The levels that are defined in the following manner

- **Level 1: Initial. The software process is characterized as ad hoc and occasionally** even chaotic. Few processes are defined, and success depends on individual effort.
- **Level 2: Repeatable. Basic project management processes are established** to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

- **Level 3: Defined. The software process for both management and engineering** activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software.

  This level includes all characteristics defined for level 2.
- **Level 4: Managed. Detailed measures of the software process and product** quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.
- **Level 5: Optimizing. Continuous process improvement is enabled by quantitative** feedback from the process and from testing innovative ideas and technologies.
- This level includes all characteristics defined for level 4.

II. Project Metrics

- Software process metrics are used for strategic purposes.
- Software project metrics are tactical. That is, project metrics and the indicators derived from them are used by a project manager and a software team to adapt project work flow and technical activities.
- *The first application of project metrics* on most software projects occurs during estimation.

- The intent of Project Metrics is given as:-

1.) These metrics are used to minimize delays & reduce the potential problems & risks.

2.) Project metrics are used to assess product quality on an ongoing basis and, when necessary modify the technical approach to improve quality.

- Every project should measure:

• *Inputs*—measures of the resources (e.g., people, environment) required to do the work.

• *Outputs*—measures of the deliverables or work products created during the software engineering process.

• *Results*—measures that indicate the effectiveness of the deliverables.

III. 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.

III. 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.
- 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
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

III. SOFTWARE MEASUREMENT contd..

- 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 (Figure 4.4) 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:***

  1. This method is Language dependent.
  2. 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
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

III. 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 4.5. Five information domain characteristics are determined and counts are provided in the appropriate table location.

III. SOFTWARE MEASUREMENT contd..

FIGURE 4.5
Computing function points

III. SOFTWARE MEASUREMENT contd..

- Information domain values are defined in the following manner:
- 1. **Number of user inputs.** Each user input that provides distinct application-oriented data to the software is counted.
- 2. **Number of user outputs.** Each user output that provides application-oriented information (i.e. reports, screens, error messages) to the user is counted.
- 3. **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.
- 4. **Number of files.** Each logical master file is counted.
- 5. **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.

III. 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.

III. 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?

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?

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.

- 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.

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.

METRICS FOR SOFTWARE QUALITY

- The overriding goal of software engineering is to produce a high-quality system,application, or product.
- To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process.
- In addition, a good software engineer (and good software engineering managers) must measure if high quality is to be realized.

- **Measures for software Quality????**
- There are many measures of software quality, correctness; maintainability, integrity, and
- usability provide useful indicators for the project team.

- A. **Correctness.**
- Correctness is the degree to which the software performs its required function.
- The most common measure for correctness is defects per KLOC, where a defect is
- defined as a verified lack of conformance to requirements.
- When considering the overall quality of a software product, defects are those
- problems reported by a user of the program after the program has been released for
- general use.
- For quality assessment purposes, defects are counted over a standard period of time,
- typically one year.

- **B. Maintainability.**
- Maintainability is the ease with which a program can be
- corrected if an error is encountered,
- adapted if its environment changes, or
- enhanced if the customer desires a change in requirements

- There is no way to measure maintainability directly; therefore, we must use indirect
- measures.
- A simple time-oriented metric is *mean-time-to change* (MTTC), the time it takes to
- analyze the change request,
- design an appropriate modification,
- implement the change, test it, and
- Distribute the change to all users.
- On average, programs that are maintainable will have a lower MTTC (for equivalent
- types of changes) than programs that are not maintainable.

- **C. Integrity.**
- This attribute 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:
- • *Threat*
- • *Security*.
- *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.
- *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) * (1 – security)]**
- Where threat and security are summed over each type of attack

- D. **Usability.**
- |If a program is not user-friendly, it is often doomed to failure, even if the functions that
- it performs are valuable.
- 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 measured when the system is used by someone
- who is moderately efficient, and
- (4) a subjective assessment of users attitudes toward the system.

- **Defect Removal Efficiency:-**
- A quality metric that provides benefit at both the project and process level is defect
- removal efficiency (DRE).
- In essence, DRE is a measure of the filtering ability of quality assurance and control
- activities as they are applied throughout all process framework activities.
- When considered for a project as a whole, DRE is defined in the following manner:
- **DRE = *E*/ (*E* + *D*)**
- Where *E* is the number of errors found before delivery of the software to the end-user
- and *D* is the number of defects found after delivery.

- The ideal value for DRE is 1, That is, no defects are found in the software.
- Realistically will be greater than 0, but the value of DRE can still approach 1. As *E*
- *i*ncreases (for a given value of *D*), the overall value of DRE begins to approach 1.
- DRE can also be used within the project to assess a team's ability to find errors before
- they are passed to the next framework activity or software engineering task.
- When used in this context, we redefine DRE as, **DRE*i* = *Ei* / (*Ei* + *Ei+1)***
- Where *Ei* is the number of errors found during software engineering activity *i* and
- *Ei+1* is the number of errors found during software engineering activity *i+1* that are
- traceable to errors that were not discovered in software engineering activity *i*.

IV. INTEGRATING METRICS WITHIN THE SOFTWARE PROCESS

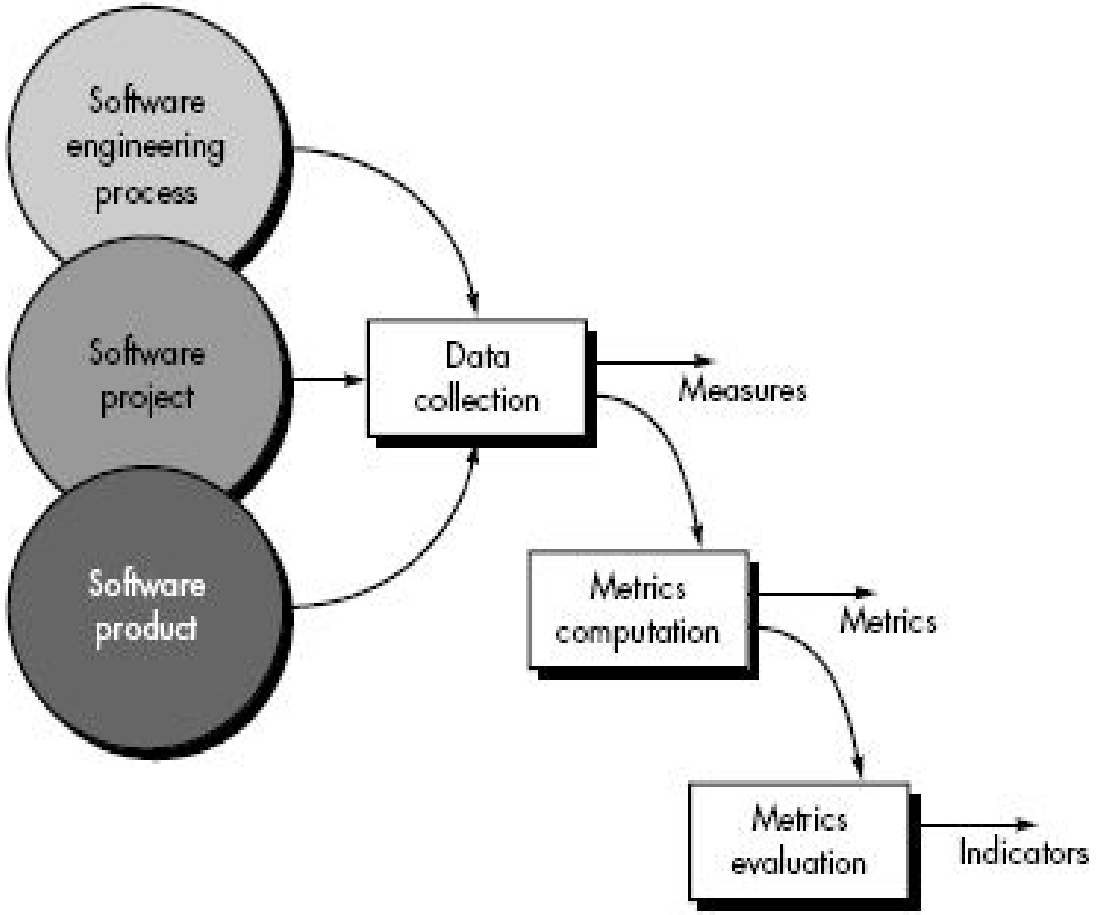
- (*NOT IMPORTANT TOPIC*)
- In this section, we consider some arguments for software metrics and present an
- approach for instituting a metrics collection program within a software engineering.
- **A. Arguments supporting for Software Metrics** (*NOT IMPORTANT TOPIC*)
- Why is it so important to measure the process of software engineering and the product
- (software) that it produces?
- If we do not measure, there no real way of determining whether we are improving.
- And if we are not improving, we are lost.
- By requesting and evaluating productivity and quality measures, senior management
- can establish meaningful goals for improvement of the software engineering process.
- If the process through which software is developed can be improved, a direct
- Impact on the bottom line can result.
- But to establish goals for improvement, the current status of software development
- must be understood. Hence, measurement is used to establish a process baseline from

- The day-to-day rigors of software project work leave little time for strategic thinking.
- Software project managers are concerned with more mundane (but equally important) issues: developing meaningful project estimates, producing higher-quality systems, getting product out the door on time.
- By using measurement to establish a project baseline, each of these issues becomes more manageable.
- We have already noted that the baseline serves as a basis for estimation. Additionally, the collection of quality metrics enables an organization to "tune" its software process to remove the "vital few" causes of defects that have the greatest impact on software development.
- At the project and technical levels (in the trenches), software metrics provide immediate benefit. As the software design is completed, most developers would be anxious to obtain answers to the questions such as
- Which user requirements are most likely to change?
- Which components in this system are most error prone?
- How much testing should be planned for each component?
- How many errors (of specific types) can I expect when testing commences?

- **B. Establishing a Baseline** (*NOT IMPORTANT TOPIC*)
- By establishing a metrics baseline, benefits can be obtained at the process, The metrics baseline consists of data collected from past software development projects and can be as simple as the table presented in Figure 4.4 or as complex as a comprehensive database containing dozens of project measures and the metrics derived from them.
- To be an effective aid in process improvement and/or cost and effort estimation, baseline data must have the following attributes:
- (1) Data must be reasonably accurate;
- (2) Data should be collected for as many projects as possible;
- (3) Measures must be consistent;
- (4) Applications should be similar to work that is to be estimated

- **C. Metrics Collection, Computation, and Evaluation** (*NOT IMPORTANT TOPIC*)
- The process for establishing a baseline is illustrated in Figure 4.7.
- Ideally, data needed to establish a baseline has been **collected** in an ongoing manner.
- Once measures have been collected (unquestionably the most difficult step), metrics **computatio**n is possible.
- Depending on the breadth of measures collected, metrics can span a broad range of LOC or FP metrics as well as other quality- and project-oriented metrics.
- Finally, metrics must be e**valuated** and applied during estimation, technical work, project control, and process improvement.
- Metrics evaluation focuses on the underlying reasons for the results obtained and produces a set of indicators that guide the project or process.

FIGURE 4.7
Software metrics collection process



V. MANAGING VARIATION: STATISTICAL PROCESS CONTROL

- Because the software process and the product it produces both are influenced by many
- parameters (e.g., the skill level of practitioners, technology that is to be implemented,
- the tools to be used in the development activity), metrics collected for one project or
- product will not be the same as similar metrics collected for another project.
- In fact, there is often significant variability in the metrics we collect as part of the
- software process.
- A graphical technique is available for determining whether changes and variation in
- metrics data are meaningful. Called the *control chart* and developed by Walter Shewart
- in the 1920s,

- This technique enables individuals interested in software process improvement to
- determine whether the dispersion (variability) and "location" (moving average) of
- process metrics are stable (i.e., the process exhibits only natural or controlled changes)
- or unstable (i.e., the process exhibits out-of-control changes and metrics cannot be used
- to predict performance).
- Two different types of control charts are used in the assessment of metrics data
- (1) The moving range control chart and
- (2) The individual control chart.

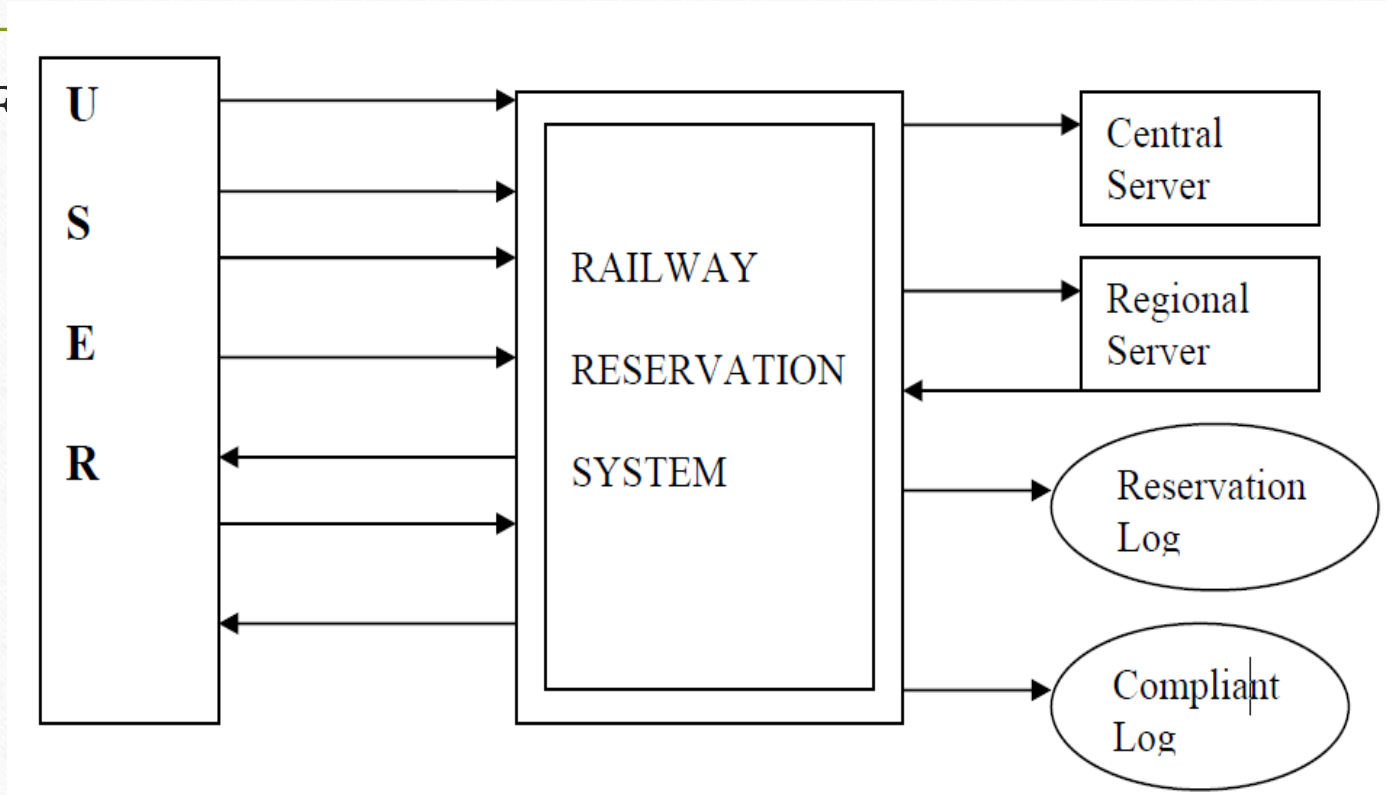
VI. METRICS FOR SMALL ORGANIZATIONS

- The vast majority of software development organizations have fewer than 20 software people.
- It is unreasonable to expect that such organizations will develop comprehensive software metrics programs.
- However, it is reasonable to suggest that software organizations of all sizes measure and then use the resultant metrics to help improve their local software process and the quality and timeliness of the products they produce.
- The software group is polled to define a single objective that requires improvement.
- For example, objective can be to "reduce the time to evaluate and implement change requests."

- A small organization might select the following set of easily collected measures:
- Time (hours or days) elapsed from the time a request is made until evaluation is complete, *tqueue*.
- Effort (person-hours) to perform the evaluation, *Weval*.
- Time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel, *teval*.
- Effort (person-hours) required to make the change, *Wchange*.
- Time required (hours or days) to make the change, *tchange*.
- Errors uncovered during work to make change, *Echange*.
- Defects uncovered after change is released to the customer base, *Dchange*.
- Once these measures have been collected for a number of change requests, it is possible to compute the total elapsed time from change request to implementation of the change and the percentage of elapsed time absorbed by initial queuing, evaluation and change assignment, and change implementation.
- Similarly, the percentage of effort required for evaluation and implementation can be determined.
- These metrics can be assessed in the context of quality data, *Echange* and *Dchange*.
- The percentages provide insight into where the change request process slows down and may lead to process improvement steps to reduce *tqueue*, *Weval*, *teval*, *Wchange*, and/or *Echange*.
- In addition, the defect removal efficiency can be computed as DRE = *Echange* /(*Echange* + *Dchange*).

Question

- **Compute the F**
- **Ans:**



- **Measurement Parameter Count weighting factor**
- Sim. Average. Complex
- Number of user inputs 50 x (3 + 4 + 6) = 650
- Number of user outputs 50 x (4 5 7) = 800
- Number of user inquiries 100 x (3 4 6) = 1300
- Number of files 20 x (7 10 15) = 640
- Number of external interfaces' 10 x (5 7 10) = 220
- **Count Total 3610**

- Complexity adjustment values (1-5 range) :
- 1. Does the system require reliable backup and recovery? - 4
- 2. Are data communications required? - 3
- 3. Are there distributed processing functions? - 3
- 4. Is performance critical? - 2
- 5. Will the system run in an existing, heavily utilized operational environment? - 1
- 6. Does the system require on-line data entry? - 5
- 7. Does the on-line data entry require the input transaction to be built over multiple screens
- or operations? - 5
- 8. Are the masters file updated on-tine? - 2
- 9. Are the inputs, outputs, files or inquiries complex? - 1
- 10. Is internal processing complex? - 1
- 11. Is the code designed to be reusable ? - 2
- 12. Are conversion and installations included in the design? - 2
- 13. Is the design designed for multiple installations in different organizations? - 2
- 14. Is the application designed to facilitate change and ease of use by the user? – 5
- Σ Fi => 38
- FP = count total X [0.65 + 0.01 X (ΣFi)]
- = 3610 X [0.65 + (0.01 X 38)]
- = 3610 X (1.03)
- FP = 3718.3

- **Compute the LOC and function point for an E-commerce book ordering system.**
- **(CMPN May-04 10-M)**
- **Ans.:**
- For function point refer above questions answer.
- For LOC,
- count number of lines of code and then find out
- Errors per KLOC
- Defects per KLOC
- Page of documentation per KLOC
- Errors per person-month
- LOC per person-month
- $ Per page of documentation.
- All above fields are to be calculated with respect to FP.

Q. Compute function point value for a project with following information domain

- **characteristics:** (Q 2 (b) CMPN Dec-05 10-M)
- No. of user inputs : 32
- No. of user outputs : 60
- No. of user inquiries : 24
- No. of files : 8
- No. of external inputs : 2
- Assume that all complexity adjustment values are average.

- Ans:
- Assume number of algorithm counted = 14. .
- **Function point FP = Count- total x [0.65+ 0.01x SUM (Fi)]**

- As it is stated that the values are all average, thus we, make the weight factors of 4,5,4,10,7
- respectively for the inputs mentioned, in a, b, c, d, and e.
- **∴ Total count = Number input x Wr+ Number of user *O/p's* x Wf+ Number of inquiries**
- **x Wr.+ Number of files x Wf+ Number of external interfaces x Wr**
- Wr => respective weights
- **Total Count =32 x 4 + 60 x 5 + 24 x 4 + g x 10'+ 2 x 7 = 618**
- F(i) are the complexity adjustment values based on the responses to the questions.

- F(i)
- System requires backup ?
- Data communication required etc.

- As stated all 14 algorithms are counted,
- **∴SUM F(i) = (1 + 2 + 3 ... + 14) * scale factor**
- **= (((14 * 15)/2)) * 3) = 315**
- **∴FP = 618 x [0.65 + 0.01 x 315] = 2348.4**
- Feature points are calculated on the basis of same equation, but with single set of weights
- *4,5,4,7,7,3 (i/p, olp,* inquires, files, interfaces, algorithm). .
- **∴Count total = 32x 4 + 60 x 5 + 24 x 4 + 8 x 7 + 2 x 7 + 14 x 3 = 636**
- **∴Feature points = 636 * [0.65 + 0.01 * 315] = 2416.8**

Empirical estimation models

- An *estimation model for computer software uses empirically derived formulas to predict* effort as a function of LOC or FP.
- The empirical data that support most estimation models are derived from a limited sample of projects.

The COCOMO Model

- Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for *COnstructive COst MOdel*.
- Gives the estimate of number of man-months to develop a software project.
-