SOFTWARE ENGINEERING

Unit-3
Managing Software Project

Outlines

- The Management Spectrum
- Software Metrics
  - Process, Product and Project Metrics
- Software Project Estimations
- Decomposition Techniques
- Integrating metrics within the SW process

Why Measure Software?

- To **determine quality** of a product or process.
- To **predict qualities** of a product or process.
- To **improve quality** of a product or process.

Management Spectrum

1. Process
   - Specifies **activities** related to production of software.
   - Specifies the **abstract set of activities** that should be performed to go from user needs to final product.
2. Project
   - Software **development work** in which a software process is used.
   - The **actual act of executing the activities** for some specific user needs.
3. Product
   - The **outcome** of a software project.
   - **All the outputs** that are produced while the activities are being executed.

Management Spectrum

4. People

- People in Software engineering process is an important part.
- In this section we examine the players who participate in the software process and the manner in which they are organized to perform effective software engineering.

1. Process Metrics

- Process Metrics are an **invaluable tool for companies to monitor, evaluate and improve their operational performance** across the enterprise.
- They are **used for making strategic decisions**.
- 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.
  - Ex., **Defect Removal Efficiency (DRE) metric**
    - Relationship between errors (E) and defects (D)
    - **DRE = E / (E + D)**

(Total defects found in development [A]/ (Total defects found in development[A] + Defects found in production [B])) x 100

1. Process Metrics (Cont...)

- 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

2. Project Metrics

- 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 their status becomes critical
  4. Adjust work flow or **tasks**
  5. 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** smart **decisions**.
- They are **used to adapt project workflow** and **technical activities**.

2. Project Metrics (Cont…)

- Project metrics are used to
  - **Minimize the development schedule** by making the adjustments necessary to **avoid delays and** to reduce **potential problems and risks**.
  - Evaluate **product quality** on an ongoing basis and **guide to modify the technical approach** to improve quality.

3. Product Metrics

- Product metrics help software engineers **to gain insight into the design and construction** of the software they build,
- By **focusing on specific, measurable attributes** of software engineering work products.
- Product metrics provide a basis from which analysis, design, coding and testing can be conducted **more objectively and assessed more quantitatively**.

4. People Metrics

- There are following area's for s/w people like recruiting, selection, performance, training, compassion, career development, work design etc…
- The Players:-
  - Senior managers, Project manager, practitioners, customers, end users.
- Team leader:-
  - Motivation, Organization, Ideas or Innovation.

The W5HH Principle for Project Management

- Boehm suggests an approach (W5HH) that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources.
  - Why is the system being developed?

    Enables all parties to assess the validity of business reasons for the software work.
  - What will be done?

    Establish the task set that will be required.
  - When will it be accomplished?

    Project schedule to achieve milestone.
  - Who is responsible?

    Role and responsibility of each member.
  - Where are they organizationally located?

    Customer, end user and other stakeholders also have responsibility.

The W5HH Principle for Project Management

- How will the job be done technically and managerially?

  Management and technical strategy must be defined.

- How much of each resource is needed?

  Develop estimation.

- It is applicable regardless of size or complexity of software project.

Functional and non-functional requirements

- **Functional Requirements**
  - Statements of services the system should provide.
  - How the system should react to particular inputs.
  - What system should do?
- **Non-functional Requirements**
  - Constraints on the services or functions offered by the system, such as timing constraints, constraints on the development process, standards, etc.
  - How system Works?
- **Domain Requirements**
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain.

Functional Requirements

- **Details of operations conducted in every screen.**
- **Complete information about the workflows performed by the system.**
- **ex:- display name, show available space, total size, add customer, make payment etc..**

Non-functional Requirements

- **Security**

  Ex– Log in, OTP, Authenticator etc.

- **Performance Requirements**

  Ex- Response Time, User interface, capacity

- **Maintainability**

  Ex- Back up, errors/crash reports

- **Reliability**

  Ex- **Availability**

Terminologies

- Measure
  - It **provides a quantitative indication** of the range, amount, dimension, capacity or size of some attributes of a product or process.
    - Ex., the **number of uncovered errors**.
- Metrics
  - Is a **quantitative measure of the degree** (limit) to which a system, component or process obtains a given attribute.
  - It **relates individual measures** in some way.
    - Ex., **number of errors found** per **review**.

Terminologies (Cont…)

- Indicators
  - Is a **metric** or **combination of metrics** that provides insight into the software process, project or the product itself.
  - It **enables the project manager** or software engineers **to adjust** the **process,** the **project** or **the product** to make things better.
    - Ex., **Product Size** is an indicator of increased coding, integration and testing effort
- Direct Metrics
  - **Immediately measurable** attributes.
    - Ex., Line of Code (LOC), Execution Speed, Defects Reported
- Indirect Metrics
  - Aspects that are **not immediately quantifiable**.
    - Ex., Functionality, Quantity, Reliability

Terminologies (Cont…)

- Faults
  - Errors
    - Faults **found by the practitioners** during software development.
  - Defects
    - Faults **found by the customers** after release.

Software Metrics

- Software Metrics for Software Cost and Effort Estimations
  1. Size-Oriented Metrics
  2. Function-Oriented Metrics
  3. Object-Oriented Metrics
  4. Use-Case–Oriented Metrics

Software Metrics

- Software Metrics for Software Cost and Effort Estimations
  1. Size-Oriented Metrics
  2. Function-Oriented Metrics
  3. Object-Oriented Metrics
  4. Use-Case–Oriented Metrics

1. Size-Oriented Metrics

- **Derived by** standardizing **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.
- A set of simple size-oriented metrics can be developed for each project
  - Errors per KLOC
  - Defects per KLOC
  - $ per KLOC
  - Pages of documentation per KLOC

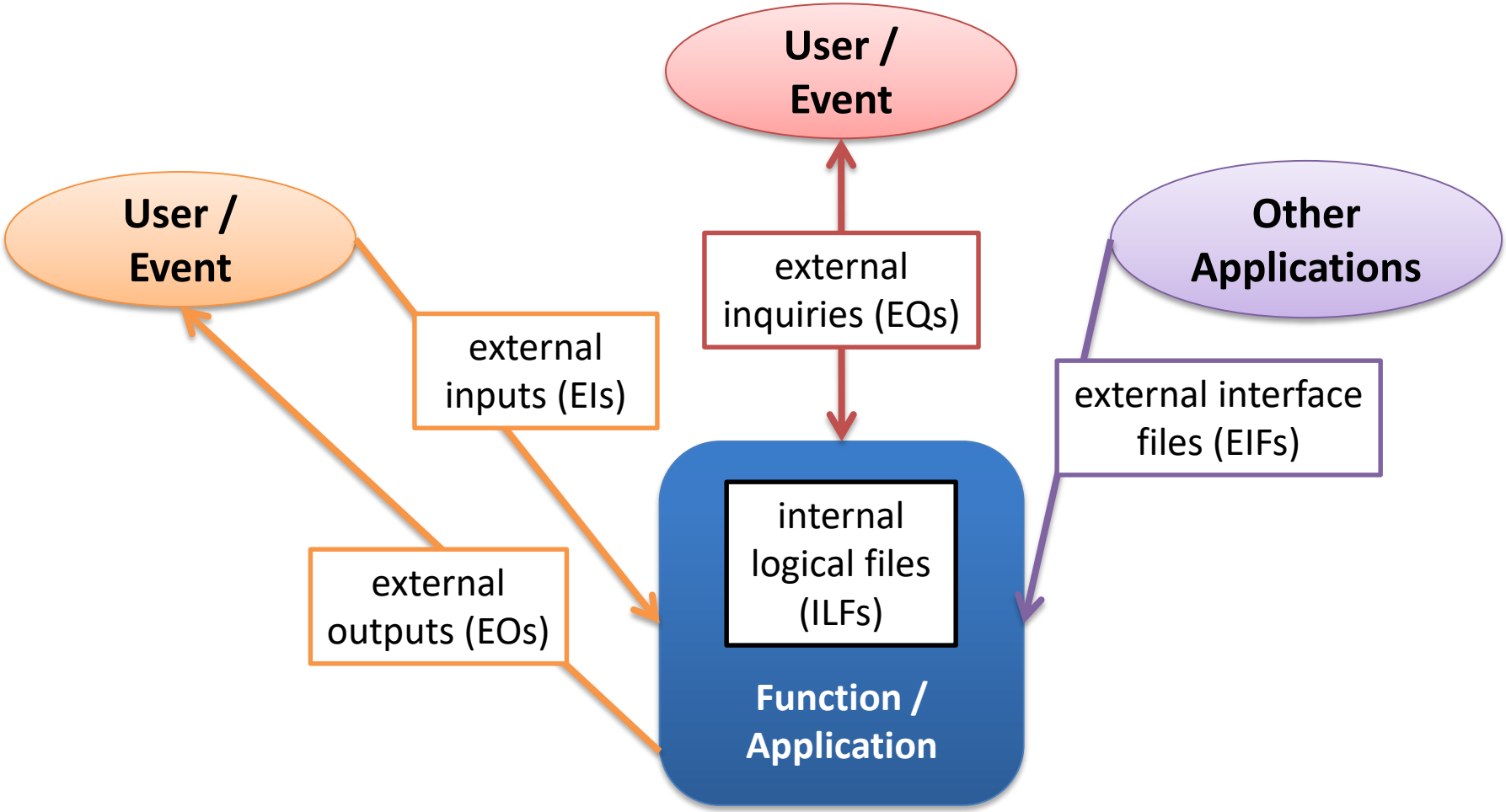
1. Size-Oriented Metrics (Cont...)

- In addition, other interesting metrics can be computed, like
  - Errors per person-month
  - KLOC per person-month
  - $ per page of documentation
- 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**
  - **well-designed** but short programs
  - Cannot easily accommodate **nonprocedural languages**

2. Function Point Metrics

- The function point (FP) metric can be used effectively as a means for **measuring the functionality** delivered by a system.
- FP metric can be used to
  1. **estimate the cost or effort required** to design, code, and test the software;
  2. **predict the number of errors** that will be encountered during testing; and
  3. **forecast the number of components and/or the number of projected source lines** in the implemented system.

Function Point Components



Function Point Components

- Information domain values (components) are defined in the following manner
  - Number of **external inputs** (EIs)
    - input data **originates from a user** or is transmitted from another application.
    - Ex- in Library Management System, entering Library card number of Student
  - Number of **external outputs** (EOs)
    - external output is **derived data within the application** that provides information to the user. output refers to reports, screens, error messages, etc.
    - Ex- List of books checked out by student

Function Point Components

- Number of **internal logical files** (ILFs)
  - internal logical file is a **logical grouping of data that resides within the application's boundary** and is maintained via external inputs.
  - Ex- File containing list of books in the library

- Number of **external interface files** (EIFs)
  - external interface file is a **logical grouping of data that resides external to the application** but provides information that may be of use to the another application.
  - Ex- file containing data of book borrow transactions of library

- Number of **external inquiries** (EQs)
  - Data retrieved from internal logical files and external interface files
  - Ex- number of books borrowed by a student

Compute Function Points

- **Formula**
  - **FP = Count Total * [0.65 + 0.01 * ∑(Fi)]**
    - ***Count Total*** = Information Domain Value Count * weighting factor
    - ***Weighting factor*** is determined for each organization via empirical (based on experience) data.
    - Fi (i=1 to 14) are ***complexity value adjustment factors (VAF).***

Compute Function Points (Cont…)

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total →							

Value Adjustment Factors

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).

Example-1

Study of requirement specification for a project has produced following results,

External Input: 3

External Output: 2

External Inquiry: 2

Internal Logical File: 1

External Interface File: 4

Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total	→						50

Used Adjustment Factors and assumed values are,

F03. Distributed processing = 5

F04. High performance = 4

F10. Complex internal processing = 3

F11. Code to be reusable = 2

F13. Multiple sites = 3

Value Adjustment Factor (VAF) = 17

Adjustment calculation

Adjusted FP = Count Total * [0.65 + 0.01 * ∑(Fi)]

= Unadjusted FP x [0.65 + (0.01 x Adjustment Factor)]

= 50 x [0.65 + (0.01 x 17)]

= 50 x [0.82]

= **41**

Function Point Controversy

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

3. Object-Oriented Metrics

- **Conventional software project metrics** (LOC or FP) **can be used** to estimate object-oriented software projects.
- However, these metrics **do not provide enough** detailing for the schedule and effort adjustments that are required as you iterate through an evolutionary or incremental process.
- Lorenz and Kidd suggest the following set of metrics for OO projects
  - Number of **scenario scripts**
  - Number of **key classes**
  - Number of **support classes**

4. Use Case Oriented Metrics

- Like FP, the use case is defined early in the software process, allowing it to be **used for estimation before significant modelling 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**, because use cases can be **created at vastly different levels of abstraction**, there is no standard “size” for a use case.

Barriers to implementing process improvement methods

1. getting started

orgs. need to conduct an assessment (e.g. Capability Maturity Model)

2. staff turnover

downsizing is difficult environment for process improvement

need champions to stick around

3. dedicated resources

need full-time dedicated resource(s) to implement process improvement methods

Barriers to implementing process improvement methods

4. management support
- it’s necessary
5. time restrictions
- you’ve got to make the time to institute it

Metrics for Software Quality.

- The overriding goal of software engineering is to produce a high-quality system, application or product.
- The quality of a system, application, or product is only as good as the requirements that describe the problem, the design that models the solution, the code that leads to an executable program, and the tests that exercise the software to uncover errors.
- Metrics such as work product (e.g., requirements or design) errors per function point, errors uncovered per review hour, and errors uncovered per testing hour provide insight into the efficacy of each of the activities implied by the metric.
- Error data can also be used to compute the *defect removal efficiency* (DRE) for each process framework activity.

Measuring Quality

- There are many measures of software quality, correctness, maintainability, integrity, and usability provide useful indicators for the project team.
- **Correctness.** A program must operate correctly or it provides little value to its users. Correctness is the degree to which the software performs its required function.
- **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-tochange* (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.

Measuring Quality

- **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 and security. *Threat* is the probability that an attack of a specific type will occur within a given time. *Security* is the probability that the attack of a specific type will be repelled.
- **Usability.** (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.