Software Engineering

Lecture-16

Process and Project Metrics

Topics Covered

- Introduction
Metrics in the Process Domain
Metrics in the Project Domain
Software Measurement
Integrating Metrics within the Software Process

Introduction

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

A Quote on Measurement

“When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.”

LORD WILLIAM KELVIN (1824 – 1907)

Uses of Measurement

- Can be applied to the software process with the intent of improving it on a continuous basis
- Can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control
- Can be used to help assess the quality of software work products and to assist in tactical decision making as a project proceeds

Reasons to Measure

- To characterize in order to
  - Gain an understanding of processes, products, resources, and environments
  - Establish baselines for comparisons with future assessments
- To evaluate in order to
  - Determine status with respect to plans
- To predict in order to
  - Gain understanding of relationships among processes and products
  - Build models of these relationships
- To improve in order to
  - Identify roadblocks, root causes, inefficiencies, and other opportunities for improving product quality and process performance

Metrics in the Process Domain

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

(More on next slide)

METRICS IN THE PROCESS DOMAIN (CONTINUED)

- 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

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

(More on next slide)

USE OF PROJECT METRICS (CONTINUED)

- 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
- In summary
  - As <u>quality improves</u>, defects are minimized
  - As <u>defects go down</u>, the amount of rework required during the project is also reduced
  - As <u>rework goes down</u>, the overall project <u>cost is reduced</u>

Software Measurement

Categories of 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.)
- Project metrics can be consolidated to create process metrics for an organization

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 - Errors per person-month
  - Defects per KLOC - KLOC per person-month
  - Dollars per KLOC - Dollars per page of documentation
  - Pages of documentation per KLOC

(More on next slide)

Size-oriented Metrics (continued)

- 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 = count total * [0.65 + 0.01 * sum (value adj. factors)]

- Material in Chapter 15 covered this in more detail
- Function point values on past projects can be used to compute, for example, the average number of lines of code per function point (e.g., 60)

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 fact
  - FP 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
- The table on the next slide provides a rough estimate of the average LOC to one FP in various programming languages

LOC Per Function Point

Language	Average	Median	Low	High
Ada	154	--	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	55	53	9	214
PL/1	78	67	22	263
Visual Basic	47	42	16	158

Object-oriented Metrics

- Number of scenario scripts (i.e., 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

(More on next slide)

Object-oriented Metrics (continued)

- 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

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

Defect Removal Efficiency

- Defect removal efficiency provides benefits at both the project and process level
- It is a measure of the filtering ability of QA 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 before delivery of the software to the end user
  - D is the number of defects found after delivery
- As D increases, DRE decreases (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 as many errors as possible before delivery

Integrating Metrics within the Software Process

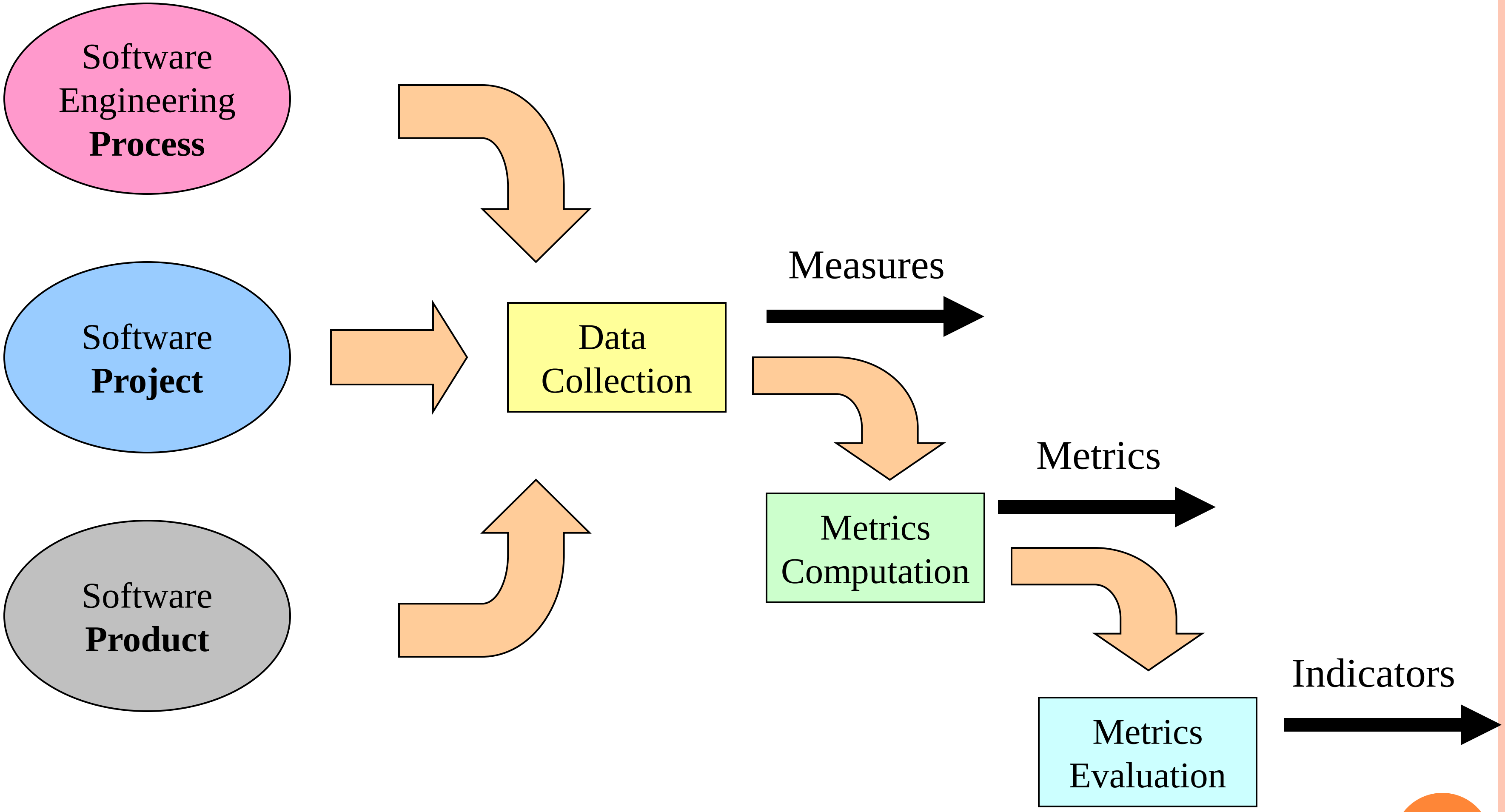
Arguments for Software Metrics

- Most software developers do not measure, and most have little desire to begin
- Establishing a successful company-wide software metrics program can be a multi-year effort
- But if we do not measure, there is no real way of determining whether we are improving
- Measurement is used to establish a process baseline from which improvements can be assessed
- Software metrics help people to develop better project estimates, produce higher-quality systems, and get products out the door on time

ESTABLISHING A METRICS BASELINE

- By establishing a metrics baseline, benefits can be obtained at the software process, product, and project levels
- The same metrics can serve many masters
- The baseline consists of data collected from past projects
- Baseline data must have the following attributes
  - Data must be reasonably accurate (guesses should be avoided)
  - Data should be collected for as many projects as possible
  - Measures must be consistent (e.g., a line of code must be interpreted consistently across all projects)
  - Past applications should be similar to the work that is to be estimated
- After data is collected and metrics are computed, the metrics should be evaluated and applied during estimation, technical work, project control, and process improvement

Software Metrics Baseline Process



Software Engineering **Process**

Software **Project**

Software **Product**

Data Collection

Measures

Metrics Computation

Metrics

Metrics Evaluation

Indicators

Getting Started with Metrics

1) Understand your existing process
2) Define the goals to be achieved by establishing a metrics program
3) Identify metrics to achieve those goals
    - Keep the metrics simple
    - Be sure the metrics add value to your process and product
4) Identify the measures to be collected to support those metrics

(More on next slide)

Getting Started with Metrics (continued)

5) Establish a measurement collection process
   - a) What is the source of the data?
   - b) Can tools be used to collect the data?
   - c) Who is responsible for collecting the data?
   - d) When are the data collected and recorded?
   - e) How are the data stored?
   - f) What validation mechanisms are used to ensure the data are correct?
6) Acquire appropriate tools to assist in collection and assessment
7) Establish a metrics database
8) Define appropriate feedback mechanisms on what the metrics indicate about your process so that the process and the metrics program can be improved

☺