Outline of The Course

- **Module 1:** Measurement theory
- **Module 2:** Software product and process measurements
- **Module 3:** Measurement management
1: Measurement Theory

- Overview of software metrics
- The basics of measurement
- Framework for software measurement
- Empirical investigation
2: SE Measurement

- Software metrics data collection
- Analyzing software measurement data
- Measuring internal product attributes: size and structure (complexity)
- Measuring external product attributes: quality and reliability
- Making process predictions: estimate effort, size, release date, etc.
3: Measurement Management

- Planning measurement
- Resource management: productivity, teams, and tools
- Benefits of metrics
- Support for measurement
At the End …

- What is software measurement about?
- Why software measurement is important?
- What does empirical investigation mean in the SE context?
- Is software measurement equivalent to software metrics? What makes them different?
- What are common software metrics that you already know?
- What attributes of the software you suggest to be measured?
- What is software measurement process?
- How to implement a software measurement plan?
- What are challenges and difficulties of applying software metrics?
Reference Books


Moral

- Industrial oriented course.
- Attend lectures: there is more to lectures than to the notes.
- Sit close-up: some of the material is hard to see from the back.
- Feel free to ask questions.
Part 1: Overview of Software Metrics

Introduction (Chapter 1)

Department of Electrical & Computer Engineering, University of Calgary

B.H. Far (far@ucalgary.ca)

http://www.enel.ucalgary.ca/People/far/Lectures/SENG421/01/
Contents

► What is measurement?
► What are software metrics?
► Scope of software engineering metrics: a chronological review.
What is Measurement?
Measurement: Definition /1

- **Measurement** is the process by which numbers or symbols are assigned to attributes of entities (objects) in the real world in such a way as to ascribe them according to defined rules.

\[
Object = \begin{cases} 
\text{attribute}_1 & (\text{value}_{11}, \text{value}_{12}, \ldots) \\
\text{attribute}_2 & (\text{value}_{21}, \text{value}_{22}, \ldots) \\
\vdots & \vdots \\
\text{attribute}_n & (\text{value}_{n1}, \text{value}_{n2}, \ldots) 
\end{cases}
\]
Measurement: Definition /2

- **Metrics** are standards (i.e., commonly accepted scales) that define measurable attributes of entities, their units and their scopes.

- **Measure** is a relation between an attribute and a measurement scale.
An entity in software measurement can be any of the following:

- **Processes:** any activity related to software development and/or maintenance (e.g., requirements engineering, testing) – these can be at different levels of granularity
- **Products:** any artifact produced or changed during software development and/or maintenance (e.g., source code, software design documents)
- **Resources:** people, hardware or software needed for the processes
An attribute is a feature or property of an entity
- e.g., blood pressure of a person, cost of a journey, duration of the software specification process

There are two general types of attributes:
- **Internal attributes** can be measured only based on the entity itself,
  - e.g., entity: code, internal attribute: size, modularity, coupling
- **External attributes** can be measured only with respect to how the entity relates to its environment
  - e.g., entity: code, external attribute: reliability, maintainability
### Measurement Example

<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Size, Reuse, Redundancy</td>
</tr>
<tr>
<td>Specification</td>
<td>Size, Reuse, Redundancy</td>
</tr>
<tr>
<td>Design</td>
<td>Size, Reuse, Modularity, Cohesion, Coupling</td>
</tr>
<tr>
<td>Code</td>
<td>Size, Reuse, Modularity, Cohesion, Coupling, Complexity</td>
</tr>
<tr>
<td>Test Cases</td>
<td>Size, Coverage</td>
</tr>
</tbody>
</table>

![Diagram](image)
Measurement: Types

- Measurements are needed as:
  - **Descriptors** of entities already in existence.
  - **Prescriptors** (standards, norms, failure intensity objectives, benchmarks) which entities of certain class or category should satisfy.
  - **Predictors** to estimate properties of entities yet to be designed or implemented.
Measurement: How to

- In order to make entities measurable:
  - What *entities (objects)* should be selected?
  - What *attributes* should be selected?
  - What *values* should be assigned to the attributes?
  - What shall be the *rules (relationships)* ascribed to the attributes and their entities?

- **Note:** assigned values and/or ascribed rules can be either quantitative or qualitative.
Example 1: Pressure Tank
Example 1: Pressure Tank /2

- **Entities (Objects):**
  - Tanks (T₁, T₄), Valves (CV₁, CV₆), Pipes

- **Attributes:**
  - Level of liquid, pressure of liquids, flow of material

- **Values:**
  - Tank full, Tank empty, etc.

- **Rules:**
  - Relations among level, pressure and flow of material.
Example 2: Code

- Entity: Code
- Attribute: Size
- Possible measures:
  - NCSLOC (Not Commented Source Lines of Code)
  - #Statements
  - #Modules
  - #Procedures
  - #Classes
  - #Methods
  - …
Example 3: Availability

- **Entity:** Availability
- **Attributes:** system uptime, system downtime
- **Values:** time in seconds
- **Relations:**
  \[
  \text{Availability} = \frac{\text{uptime}}{\text{uptime} + \text{downtime}}
  \]
Software Metrics Challenges

- Measuring physical entities:
  
<table>
<thead>
<tr>
<th>entity</th>
<th>attribute</th>
<th>unit</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>Height</td>
<td>cm</td>
<td>178</td>
</tr>
</tbody>
</table>

- Measuring non-physical entities:

<table>
<thead>
<tr>
<th>entity</th>
<th>attribute</th>
<th>unit</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>IQ</td>
<td>IQ index</td>
<td>89</td>
</tr>
</tbody>
</table>

- SE metrics are mostly non-physical
  
  - Reliability, maturity, portability, flexibility, maintainability, etc. and relations are unknown
Misleading Metrics!

- Fact (1): Knowledge is power
- Fact (2): Time is money
- Relation (rule): power = work / time

Substituting “power” and “time”
- Knowledge = work / money
- As knowledge approaches zero money approaches infinity regardless of the amount of work done!

Conclusion:
- The less you know, the more you make!

What went wrong here?
- Counter intuitive
- Needs validation
What is Software Measurement?
What Is Software Measurement?

- Software metrics are measures that are used to quantify software, software development resources, and/or the software development process.
- This includes items which are directly measurable, such as *lines of code*, as well as items which are calculated from measurements, such as *software quality*. 
Measurement in SE

- Measurement in SE is selecting, measuring and putting together many different attributes of the software, and adding our subjective interpretations in order to get a whole picture of the software.
- This is not a trivial task!
- 300+ metrics have been defined.
Before a measurement project can be planned

- Objectives and scope should be established
- Alternative solutions should be considered
- Technical and management constraints should be identified.

This information is required to estimate costs, project tasks, and a project schedule.
In order to manage software measurement project one must understand and plan:

- The goal and scope of work
- Risks
- Resources required
- Tasks to be accomplished
- Milestones to be tracked
- Total costs of the project
- Schedule to be followed
Software metrics help us understand the technical process that is used to develop a software product.
- The process is measured to be improved.
- The product is measured to increase its quality.

But …
- Measuring software projects is controversial.
- It is not yet clear which are the appropriate metrics for a software project or whether people, processes, or products can be compared using metrics.
Scope of Software Metrics

- Cost and effort estimation
- Productivity measures and models
- Data collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Structural and complexity metrics
- Capability maturity assessment
- Management by metrics
- Evaluation of methods and tools

More about scope in the next session
Example 1

- We are going to buy a new colour laser printer for our department. We have borrowed the printer for the test.
- Maker’s data shows that we need to change the toner every 10,000 pages. We would like to have only one failure during the period.
- During test period, we observe that failures occur at 4,000 pages, 6,000 pages, 10,000 pages, 11,000 pages, 12,000 pages and 15,000 pages of output.
- What can we conclude about this printer?
Example 1 (cont’d)

- Failure intensity objective:
  \[ \lambda_F = 1/10000 \text{ pages} \]

- Using reliability demonstration chart we can conclude that the printer must be rejected.
Example 2

- We are going to initiate a new game and video on-demand download service. The service is provided to the customers who own PCs and register with the service.

- The customers must use specialized software to download games or videos from the server. The failure intensity of the software is 1 failure per 100 CPU hr. On average, the specialized software system runs 20 CPU hr per week on each client machine and there are 800 customers to be serviced.

- We would like to provide the customers with an on-site repair service. Each serviceperson can make 4 service calls per day. Service personnel are available 5 working days per week.

- How many service personnel do we need to hire?
Example 2 (cont’d)

- How many service personnel do we need?

- Using the value for failure intensity, each system experiences 0.2 failure per 20 hours of operation or 0.2 failure per week, on average.

- The total failures for 800 customers is 160 per week or 32 per day.

- Each serviceperson can visit 4 sites per day, therefore, the number of required personnel is $32 / 4 = 8$. 

Example 3

Failure intensity of a system is usually expressed using FIT (Failure-In-Time) unit which is 1 failure per $10^{**}9$ device hours. The failure intensity of an electric pump system used for pumping crude oil in Northern Alberta’s oil field is constant and is 10,000 FITs and 100 such pumps are operational. If for continuous operation all failed units are to be replaced immediately, what would be the minimum inventory size of pump units for one year of operation?

Pump’s Mean-Time-To-Failure (MTTF)

$\lambda = 10,000 \text{ FITs} = \frac{10,000}{10^{**9} \text{ hour}} = 1 \times 10^{*-5} \text{ hour}$

$= 1 \text{ failure per 100,000 hours}$

The 12-month reliability is: (1 year = 8,760 hours)

$R(8,760 \text{ hours}) = \exp\{-8,760/100,000\} = 0.916$ and “unreliability” is,

$F(8,760) = 1 - 0.916 = 0.084$

Therefore, inventory size is 8.4% or minimum 9 pumps should be at stock in the first year.
Overview:
Scope of Software Metrics

Process, Product and Resources
Scope of Software Metrics

- Cost and effort estimation
- Productivity measures and models
- Data collection
- Quality models and measures
- Reliability models
- Performance evaluation and models
- Structural and complexity metrics
- Capability maturity assessment
- Management by metrics
- Evaluation of methods and tools

ITEMS IN RED ARE COVERED IN THIS COURSE
Cost and effort estimation

- Software cost estimation is the process of predicting the amount of effort required to build a software system.
- Estimates for project cost and time requirements are derived during the planning stage of a project.
- Models used to estimate cost can be categorized as either cost models (e.g., Constructive Cost Model COCOMO) or constraint models (e.g., SLIM).
- Experience is often the only guide used to derive these estimates, but it may be insufficient if the project breaks new ground.
- Many models are available as automated tools.
Scope of Software Metrics /2

Productivity models and measures

- **Definition**: The rate of output per unit of input.
  - Productivity = size / effort
  - Productivity = LOC / person-month
- Productivity model based on decomposition to measurable attributes:

![Diagram showing decomposition of productivity into value and cost components.](image-url)
Scope of Software Metrics /3

Data collection

- Very critical and very hard step.
  - What data should be collected?
  - How it should be collected?
  - Is collected data reproducible?

**Example:** software failure data collection

1) Time of failure
2) Time interval between failures
3) Cumulative failure up to a given time
4) Failures experienced in a time interval
Scope of Software Metrics /4

Quality models and measures

- Software quality measurement (Rubey and Hartwick 1968~)
- McCall’s quality factors (1977~), ISO 9126
Reliability models

- Plot the change of failure intensity ($\lambda$) against time.
- Many models are proposed. The most famous ones are basic exponential model and logarithmic Poisson model.
- The basic exponential model assumes finite failures in infinite time; the logarithmic Poisson model assumes infinite failures.
- Automated tools such as CASRE are available.
Reliability Models

- Failure intensity ($\lambda$) versus execution time ($\tau$)

\begin{align*}
(B) \quad \lambda(\tau) &= \lambda_0 e^{-\frac{\lambda_0}{\nu_0} \tau} \\
(P) \quad \lambda(\tau) &= \frac{\lambda_0}{\lambda_0 \theta \tau + 1}
\end{align*}
Performance evaluation and models

- Using externally observable performance characteristics such as response time and completion rate (Ferrari 1978~)
- Efficiency of algorithms (Garey 1979~)
Scope of Software Metrics /7

Structural and complexity metrics

- Control-flow structure
- Data-flow structure
- Data structure
- Information flow attributes
- Complexity metrics (1979~)
  - Cyclomatic complexity (McCabe 1989) defining number of independent paths in execution of a program

\[ V(F) = 5 \]
Management by metrics

- Metrics for project control (1980~)
  - Metrics related to specification quality
  - Metrics for the design model
  - Metrics for documentation
  - Checking and testing metrics
  - Resource metrics
  - Change metrics
Scope of Software Metrics /9

Evaluation of methods and tools

- Efficiency of methods (1991~)
- Efficiency and reliability of tools
- Certification test of acquired tools and components
- Benchmarking
Capability maturity assessment

- ISO 9000-3: Guidelines for application of ISO 9001 to the development, supply and maintenance of software (1991)
How to Implement?

The eight steps required to implement a software measurement program are:

- Document the software development process
- State the goals
- Define metrics required to reach goals ← GQM
- Identify data to collect
- Define data collection procedures
- Assemble a metrics toolset
- Create a metrics database
- Define the feedback mechanism
Who Benefits From Measurement

- **Managers**
  - What does each process cost?
  - How productive is the staff?
  - How good is the code being developed?
  - Will the user be satisfied with the product?
  - How can we improve?

- **Engineers**
  - Are the requirements testable?
  - Have we found all the failures?
  - Have we met our product or process goals?
  - What can we predict about our software product in the future?
Exercise

Suppose that you are asked to study various software development tools and recommend the best three to your company. The following table shows a list of available development tools.
## Exercise (cont’d)

<table>
<thead>
<tr>
<th>Tool Name/Vendor</th>
<th>Languages Supported</th>
<th>Platforms</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bean Machine</td>
<td>Java</td>
<td>Windows, OS2, Unix</td>
<td>Best: Visual applet and JavaBean generation</td>
</tr>
<tr>
<td>IBM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodeWarrior Pro</td>
<td>Java, C, C++, Pascal</td>
<td>Unix, Windows, Mac</td>
<td>Best: if you need to support Unix, Windows, and Mac platforms</td>
</tr>
<tr>
<td>Metrowerks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java Workshop</td>
<td>Java</td>
<td>Solaris, Windows</td>
<td>Better: Written 100% in Java; tools based on a web browser metaphor</td>
</tr>
<tr>
<td>Sun Microsystems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JBuilder Imprise</td>
<td>Java</td>
<td>Windows, AS400</td>
<td>Better: database support</td>
</tr>
<tr>
<td>Symantec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual Cafe for Java</td>
<td>Java</td>
<td>Windows</td>
<td>Good: multithreaded debugger</td>
</tr>
<tr>
<td>Symantec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VisualAge</td>
<td>Java</td>
<td>Unix, Windows</td>
<td>Good: includes incremental compiler and automatic version control</td>
</tr>
<tr>
<td>IBM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual J++</td>
<td>Java</td>
<td>Windows</td>
<td>Fair: All the bells and whistles for Windows</td>
</tr>
<tr>
<td>Microsoft</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What are the entities, attributes and their values in your model?

<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Tool</td>
<td>Language supported</td>
<td>Java, C, C++, Pascal</td>
</tr>
<tr>
<td></td>
<td>Platform</td>
<td>Win, Unix, Mac, OS2, AS400</td>
</tr>
<tr>
<td></td>
<td>Feature</td>
<td>Fair, Good, Better, Best</td>
</tr>
</tbody>
</table>
Conclusion

- Without measurements there is no way to determine if the process/product are improving.
- Metrics allow the establishment of meaningful goals for improvement. **A baseline from which improvements can be measured can be established.**
- Metrics allow us to identify the causes of defects which have major effect on software development.
- When metrics are applied to a product they help identify:
  - which user requirements are likely to change
  - which modules are most error prone
  - how much testing should be planned for each module
References

References