COMP354

Software Engineering

Lecture 22–23

Quality, Measurement, Metrics

Outline

Quality Factor, Criteria, Metric

- internal vs external
- utility vs life-cycle view
- of product, resources, process

Measurement Theory

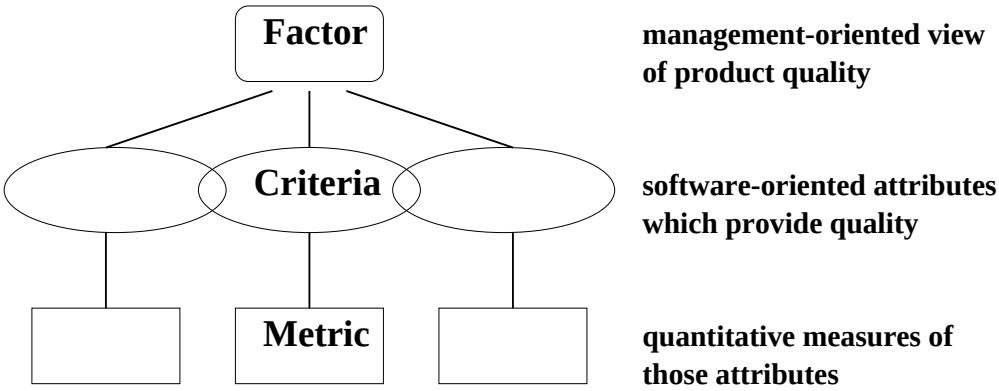
Common Metrics

- lines of code (LOC)
- Halstead
- McCabe
- function points

Do's and Don't's for Metrics

A Software Quality Program

Quality Factor, Criteria, Metric



Quality Factor is a property of a system that impacts its "quality" from a management point of view

eg, reliability, usability, maintainability, ...

These are *external quality characteristics* of the system.

Quality Criteria is a property of the software that impacts a quality factor

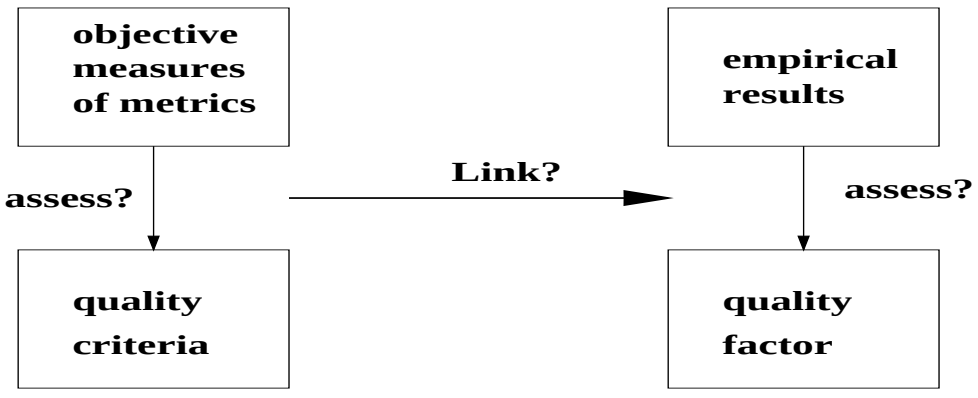
eg, modularity, simplicity, traceability

These are *internal quality characteristics* of the system.

Quality Metric is a quantitative measure of an attribute of the software that relates to a quality criteria

eg, lines of code, number of tests passed, score on a checklist

The Problem: Are they related?



Utility vs Life-cycle View

Some qualities concern the final product,
others the maintenance of the product

Life-Cycle View

Maintainability
Can I fix it?

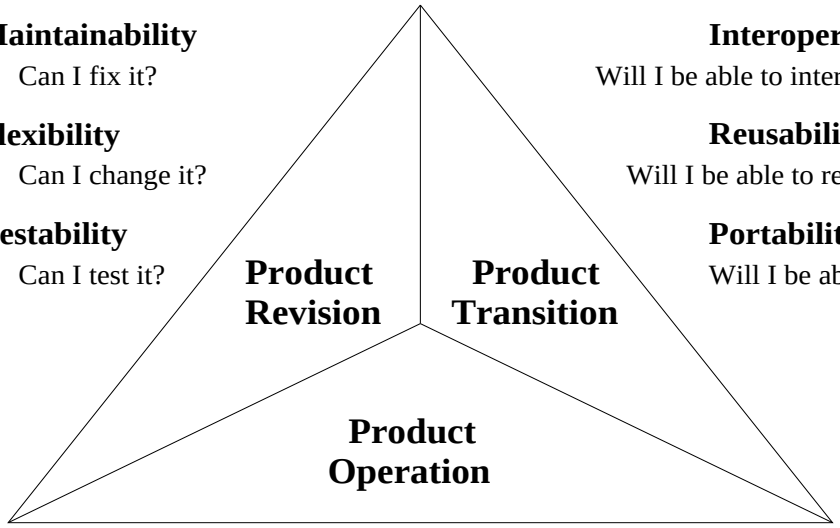
Flexibility
Can I change it?

Testability
Can I test it?

Interoperability
Will I be able to interface it with another system?

Reusability
Will I be able to reuse some of the software?

Portability
Will I be able to use it on another machine?



Correctness	Does it do what I want?
Reliability	Does it do it accurately all the time?
Efficiency	Will it run on my hardware as well as it can?
Integrity	Is it secure?
Usability	Can I run it?

Product Utility View

Product, Resource, Process

Measurements can be made of different entities.

- SE products: the code and documents

  code metrics: LOC, etc

  design metrics: coupling, cohesion?

  requirement metrics?: function points?

- SE resources: the inputs to the process

  measures of task: requirements for system

  measures of available computer resources

  measures of available human resources
  quantity, skill set

- SE process: the activities of the process

  measure time, cost, effort of activity

  measure effectiveness of activity in terms of measures on its products and efficiency

Measurement Theory

Fenton offers a scientific basis for software metrics.

Measurement theory provides framework for *Link?* between metrics and empiricism.

empirical relation system (C, R)
C is a set of entities and R is a set of relations

$$\begin{aligned} C &= \text{the set of people} \\ R &= \text{the relation ``is taller than''} \end{aligned}$$

numerical relation system (N, P)
N is a set of numbers and P is a relation on N

$$\begin{aligned} N &= \text{the set of real numbers in } [0, 120] \\ P &= \text{the relation } > \end{aligned}$$

A numerical relation system is a **representation** of an empirical representation system if the **representation condition** is satisfied.

$$x \text{ is taller than } y \quad \Leftrightarrow \quad h(x) > h(y)$$

where $h(x)$ is the height of person x in inches

A relation R is a **strict weak order** if:

- it is *asymmetric* (xRy implies $\neg\, yRx$), and
- it is *negatively transitive* (there are no values x, y, and z such that xRy, yRz, and zRx).

Cantor's Theorem The empirical relation system (C, R) has a representation in $\mathcal{R}, <$ if and only if R is a strict weak order. ($\mathcal{R}$ is the set of real numbers.)

Measurement Theory — Student Example

empirical system: Students, with relation
"X is smarter than Y"

numerical system: Students, with relation
"X has a higher GPA than Y"

But the representation condition is no longer so clear.

Student	*Grades*			
X	B	B	B	B
Y	A	C	A	C
Z	A	A	C	C

All three students have the same GPA,
but do they have the same "smartness"?

Common Metrics — Lines of Code

Simple, common metric

Illustrates difficulties of using metrics:

Clearly defined?

Do we count blank lines, lines with comments, lines that contain just "**begin**" or just "{"?

Jones identified eleven different ways of counting lines with 500% variation.

Easily interpreted?

What is being measured?

Is a long program better or worse than a short program?

Is the amount of effort required to write a program proportional to the LOCs in the program?

Standard for comparison across projects?

LOCs clearly depend on the programming language we use, but how?

Common Metrics — Halstead's Software Science

Count

$$
\begin{aligned}
a &= \text{the number of distinct operators in the program} \\
b &= \text{the number of distinct operands in the program} \\
X &= \text{the total number of operators in the program} \\
Y &= \text{the total number of operands in the program}
\end{aligned}
$$

Define

$$
\begin{aligned}
V &= a + b &&= \text{the vocabulary of the program} \\
S &= X + Y &&= \text{the size of the program}
\end{aligned}
$$

Halstead argues that the program must contain all of the 2^S ordered subsets of S elements. Thus

$$2^S = V \cdot a^a \cdot b^b.$$

Thus we can obtain the following estimate $\widehat{S}$ of S:

$$\widehat{S} = \log_2 V + a \log_2 a + b \log_2 b.$$

The reasoning seems dubious.

Nevertheless, experiments show the actual size and Halstead's estimate agree within 10%

Common Metrics — McCabe's Cyclomatic Complexity

Count entities of the control flow graph of the program:

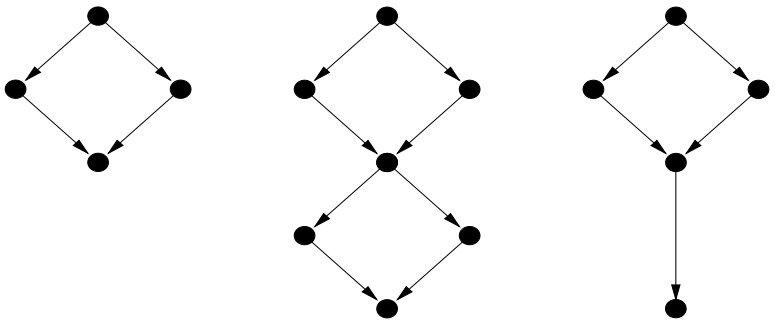
e = the number of edges
n = the number of nodes
p = the number of connected components (usually 1)

Define the **cyclomatic complexity** of the graph, C,

$$C = e - n + 2p.$$

McCabe says that well-structured modules should have a cyclomatic complexity between 3 and 7, with 10 as the upper limit.

Problem: if we ask programmers to rank flowgraphs, according to "simplicity", the programmers do not agree with cyclomatic complexity rankings.



C(x) = 2 C(y) = 3 C(z) = 2

Common Metrics — Function Points

Function points attempt to quantify the functionality of a system

metric has been empirically derived from IS applications

Calculation:

```
FP = c1 * w1  + c2 * w2  + c3 * w3  + c4 * w4 + c5 * w5

   c1 = number of inputs           weight w1 = 4
   c2 = number of outputs          weight w2 = 5
   c3 = number of inquiries        weight w3 = 4
   c4 = number of files            weight w4 = 10
   c5 = number of interfaces       weight w5 = 7
```

inputs: distinct number of items that the user provides to the system

outputs: distinct number of items that the system provides to the user
only count each screen or file of items as one item

inquiries: distinct number of interactive queries by the user that require action by the system

files: any group of information maintained by the system on behalf of the user
does not include temporary files used only for one run

interfaces: external interfaces to other systems
may include files — counted twice — if shared between applications

Do's and Don't's for Metrics

Measurement is likely to succeed if measurement results:

- are used to make decisions;
- are communicated and accepted outside the measuring department;
- are accumulated for at least two years.

Measurement is likely to fail if:

- the purpose is not clearly defined;
- measurement is perceived to be irrelevant;
- programmers perceive measurement as critical of their performance;
- overworked staff are given additional work;
- management ignores the results of measurement;

Summary

Big Question: Can we measure program complexity?

Halstead's theory: based on statistical expectations; some quite good predictions; pseudoscience.

McCabe's theory: based on flowgraph properties; a few predictions; pseudoscience.

Function points: based on detailed study of IS programs; quite useful; semiscience.

Big Question: How important is it to measure software?

According to Glass, the major problem — and the cause of many software "failures" — is the failure to estimate accurately

i.e. late delivery and budget excesses are not really a failure of software engineering but rather a failure to **estimate** the amount of work required to complete a project.

Are poor estimates due to inability to measure/estimate complexity?

A Software Quality Program

Training programming staff in new techniques, methods, and tool use

Evaluation of effectiveness of current development methods and tools

Project, quality program, and test program planning as policy or standards dictate

Use of reviews, analysis tools, and tests to find defects at the earliest possible time

Library control, change control, distribution, and storage as per project plan and relevant policies or standards

Recording of all defects found and follow-up to make certain they are corrected

Collection of defect data and subsequent analysis of defect, fault detection, and failure modality

Use of defect data to improve process

Generation and analysis of various data for early indication of adverse product or project control trends

Product qualification

Gathering, analysing, and evaluating user feedback

Survey of potential software vendors and surveillance of their performance

Objective evaluation of the fidelity with which plans and applicable standards are followed

Empowerment of staff to prevent defective code, artifacts of development, and user documentation from being entered into the system or delivered