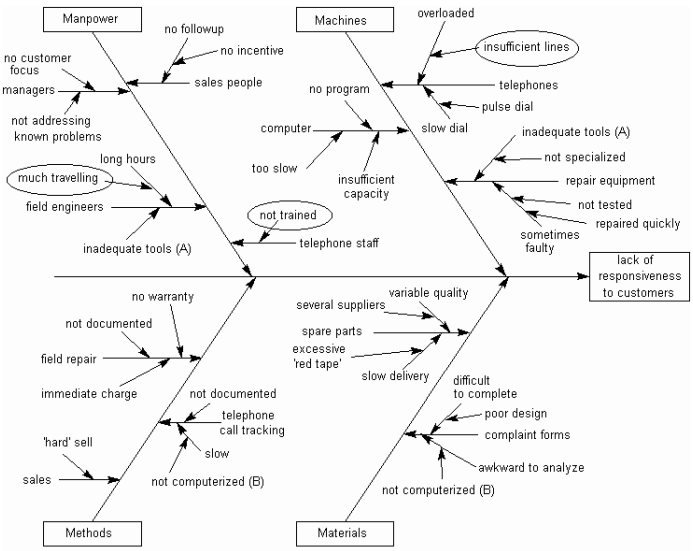
Lecture 21: Software Quality (part 2)

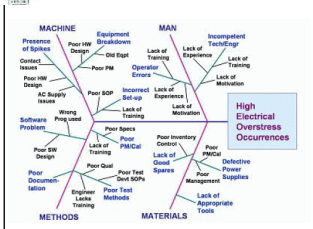
Tools for improving process quality

Software Quality Attributes

Ishikawa (Fishbone) Diagram



Pareto Chart



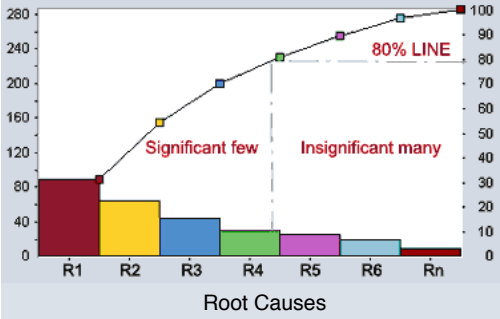
Measure frequency of each cause

"20% of the problem cause 80% of the defects"

Plot causes in order of frequency

Plot percentage contributions

Identify the top causes



How to assess software quality?

Source: Budgen, 1994, pp65-7

Reliability

- designer must be able to predict how the system will behave:
  - completeness - does it do everything it is supposed to do? (e.g. handle all possible inputs)
  - consistency - does it always behave as expected? (e.g. repeatability)
  - robustness - does it behave well under abnormal conditions? (e.g. resource failure)

Efficiency

- Use of resources such as processor time, memory, network bandwidth
  - This is less important than reliability in most cases

Maintainability

- How easy will it be to modify in the future?
  - perfective, adaptive, corrective

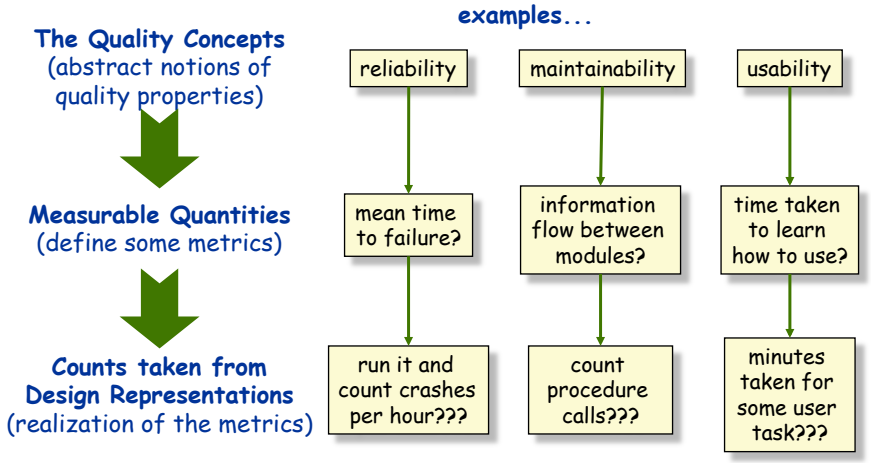
Usability

- How easy is it to use?

Measuring Quality

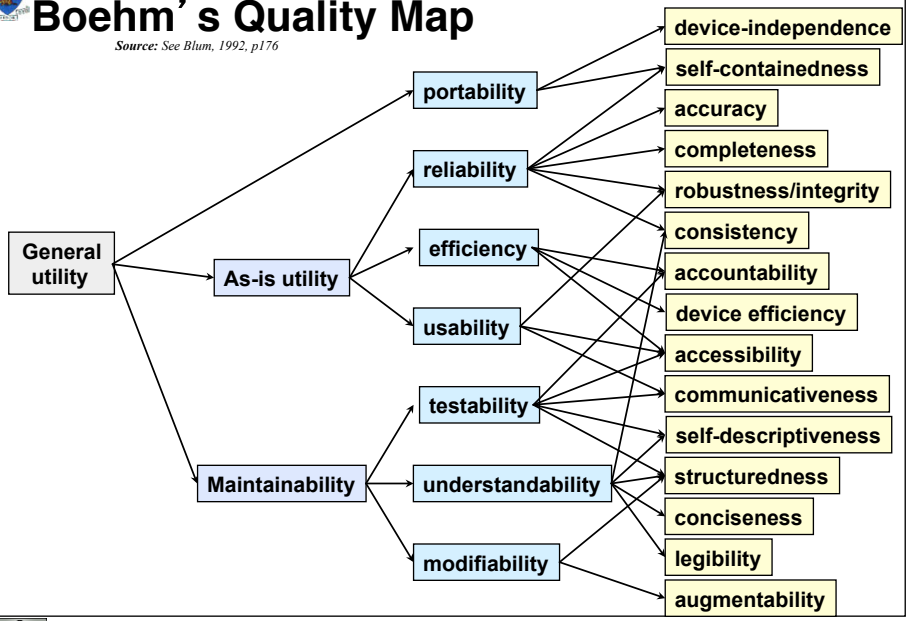
Source: Budgen, 1994, pp60-1

We have to turn our vague ideas about quality into measurables



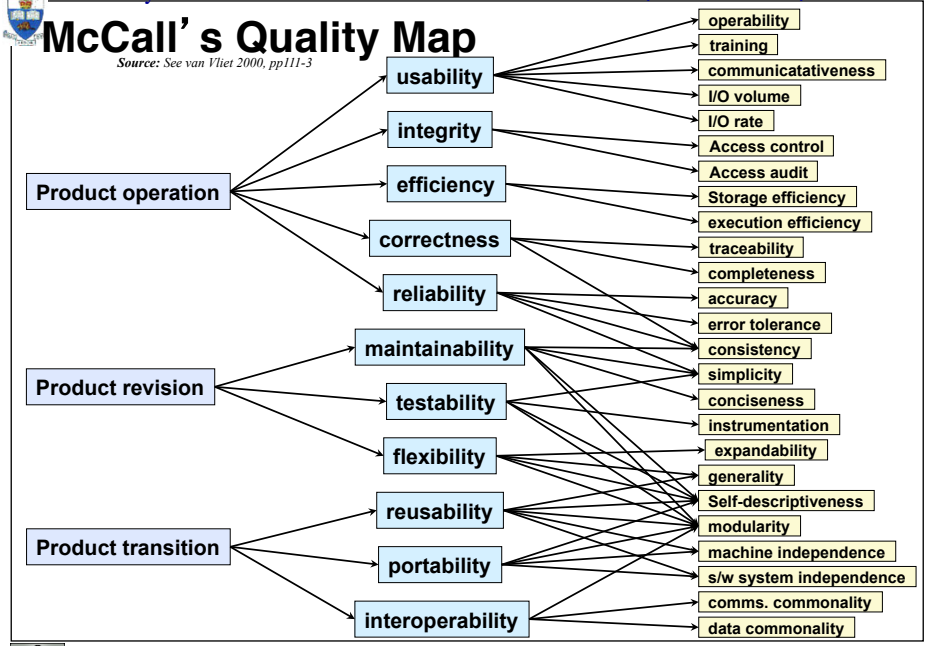
Boehm's Quality Map

Source: See Blum, 1992, p176



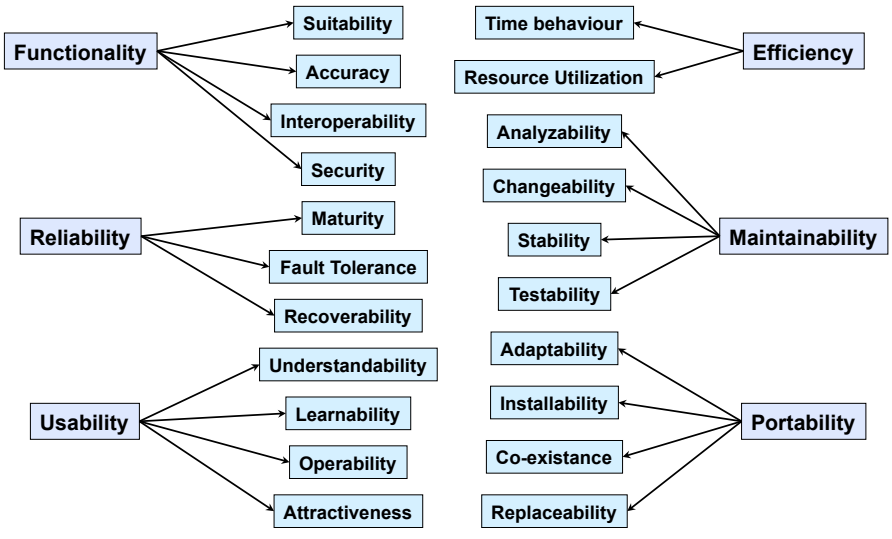
McCall's Quality Map

Source: See van Vliet 2000, pp111-3

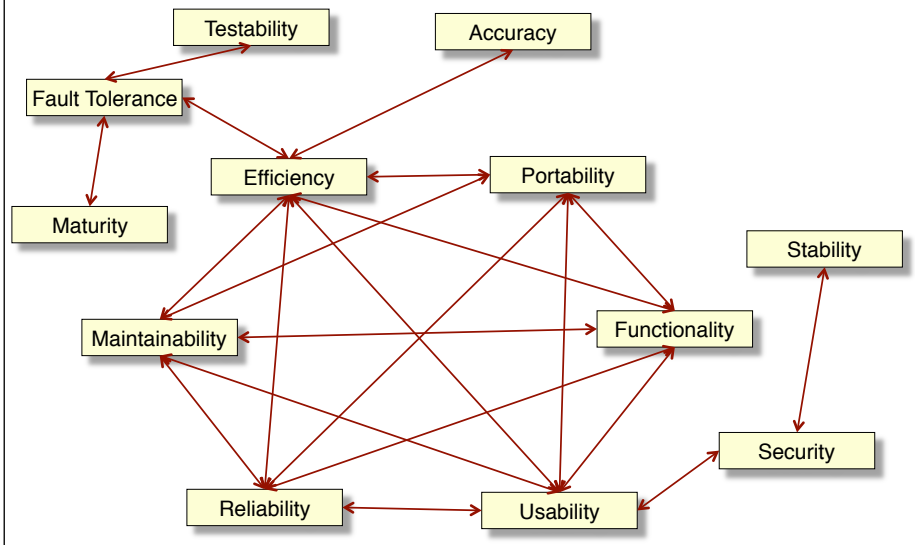


ISO/IEC 9126

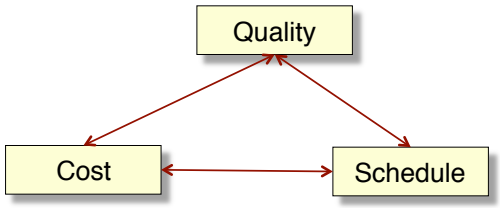
Source: See Spinellis 2006, pp5-6



Conflicts between Quality factors



More abstractly…



"Better, Faster, Cheaper - pick any two"

Measurable Predictors of Quality

Source: Budgen, 1994, pp68-74

Simplicity

the design meets its objectives and has no extra embellishments

can be measured by looking for its converse, complexity:

- control flow complexity (number of paths through the program)
- information flow complexity (number of data items shared)
- name space complexity (number of different identifiers and operators)

Modularity

different concerns within the design have been separated

can be measured by looking at:

- cohesion (how well components of a module go together)
- coupling (how much different modules have to communicate)

Wasserman's Steps to Maturity

Abstraction

Allows you to focus on the essence of a problem

Analysis and Design methods and notations

A shared language for expressing ideas about software

User Interface Prototyping

Understand the user and evaluate the user's experience

Software Architecture

Identify architectural styles and patterns

Software Process

Identify appropriate processes and assess their effectiveness

Reuse

Systematic ways to reuse past experience and products

Measurement

Better metrics to understand and manage software development

Tools and Integrated Environments

Automate mundane tasks, keep track of what we have done