# Lecture 22:
# Software Quality

## Understanding Quality

## Importance of Process Quality

- tools for improving process quality

## Software Quality Attributes






# Challenge Problem

## Context

- You built some software
- You tested it
- You shipped it

## But:

- Is it any good?
- How would you know?
- Can you do a better job next time?

**“Quality is value to some person”**

**“Quality is fitness to purpose”**

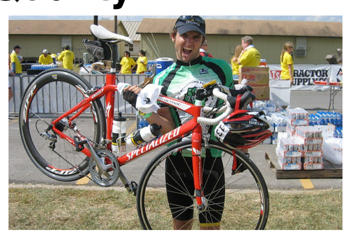
**“Quality is exceeding the customer’s expectations”**



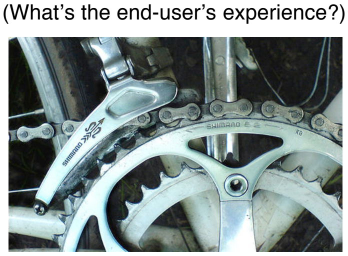



# 4 Views of Quality

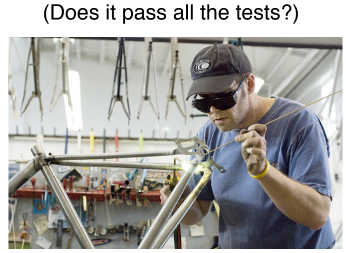
**Quality in Use**
(What’s the end-user’s experience?)



**External Quality Attributes**
(Does it pass all the tests?)



**Internal Quality Attributes**
(Is it well-designed?)



**Process Quality**
(Is it assembled correctly?)

# Importance of Process Quality

## Cannot test-in software quality

- testing or inspection cannot improve the quality of a software product
  - (by that stage it is too late)

## Defect removal

- Two ways to remove defects:
  - fix the defects in each product (i.e patch the product)
  - fix the process that leads to defects (i.e. prevent them occurring)
- The latter is cost effective as it affects all subsequent projects

## Defect prevention (from Humphrey)

- Programmers must evaluate their own errors
- feedback is essential for defect prevention
- there is no single cure-all for defects (must eliminate causes one by one)
- process improvement must be an integral part of the process
- process improvement takes time to learn






# Managing Quality (Background)

***Source:** Adapted from Blum, 1992, p473-479. See also van Vliet, 1999, sections 6.3 and 6.6*

## Industrial Engineering

- Product Inspection (1920s)
  - examine intermediate and final products to detect defects
- Process Control (1960s)
  - monitor defect rates to identify defective process elements & control the process
- Design Improvement (1980s)
  - engineering the process and the product to minimize the potential for defects

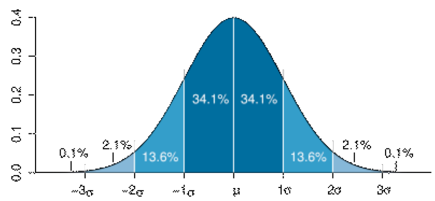
## Deming and TQM

- Use statistical methods to analyze industrial production processes
- Identify causes of defects and eliminate them
- Basic principles are counter-intuitive:
  - in the event of a defect (sample product out of bounds)…
  - …don't adjust the controller or you'll make things worse.
  - Instead, analyze the process and improve it

## Adapted to Software

- No variability among individual product instances
- All defects are design errors (no manufacturing errors)
- Process improvement principles still apply (to the design process!)

# Six Sigma




## Key ideas:
- Use statistics to measure defects
- Design the process to reduce defects

## Origin of the term
- 99.999999% of all items are with ±6σ of the mean on a normal curve
- So a target of 6σ mean no more than 1 defective part per million
- In practice, must allow for ±1.5σ drift in the mean
- So we really only get ±4.5σ = 3.4 defective parts per million

## For complex devices
- 100 parts: probability of a defective device is 0.0013
- 10,000 parts: probability of a defective device is 0.04 (I.e. 96% are okay….)
- ⇒ Design things to have fewer components
- ⇒ Control the manufacturing variability of the components





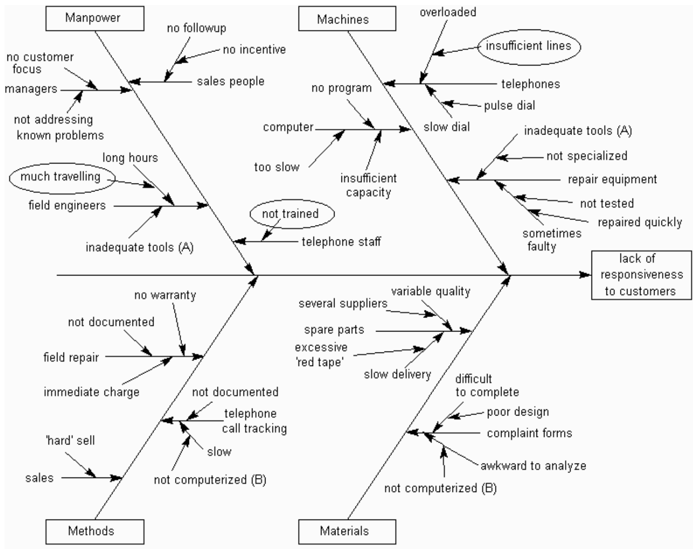

# Counterpoint: 6 Sigma for Software?

## Software processes are fuzzy
- Depend on human behaviour, not predictable

## Software Characteristics are not ordinal
- Cannot measure degree of conformance for software
- Mapping between software faults and failures is many-to-many
- Not all software anomalies are faults
- Not all failure result from the software itself
- Cannot accurately measure the number of faults in software

## Typical defect rates
- NASA Space shuttle: 0.1 failures/KLOC (but it cost $1000 per line)
- Best military systems: 5 faults/KLOC
- Worst military systems: 55 faults/KLOC
- Six Sigma would demand 0.0034 faults/KLOC

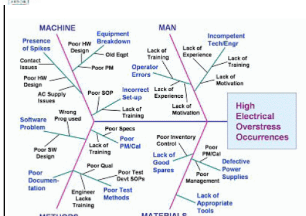
# Ishikawa (Fishbone) Diagram



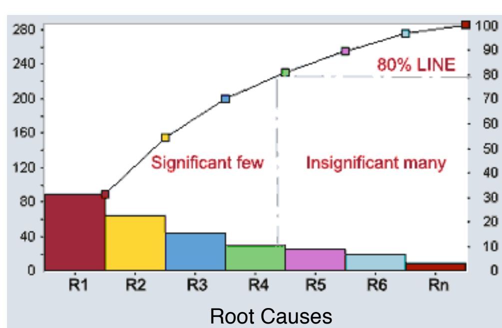

 



# Pareto Chart




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

# Capability Maturity Model

**Source:** *Adapted from Humphrey, 1989, chapter 1. See also van Vliet, 1999, section 6.6.*

| Level | Characteristic | Key Challenges |
|---|---|---|
| 5. Optimizing | Improvement fed back into process | Identify process indicators<br>"Empower" individuals |
| 4. Managed | (Quantitative) measured process | Automatic collection of process data<br>Use process data to analyze and modify the process |
| 3. Defined | (Qualitative) process defined and institutionalized | Process measurement<br>Process analysis<br>Quantitative Quality Plans |
| 2. Repeatable | (Intuitive) process dependent on individuals | Establish a process group<br>Identify a process architecture<br>Introduce SE methods and tools |
| 1. Initial | Ad hoc / Chaotic<br>No cost estimation, planning, management. | Project Management<br>Project Planning<br>Configuration Mgmnt, Change Control<br>Software Quality Assurance |






# Four Key Quality Concepts

**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

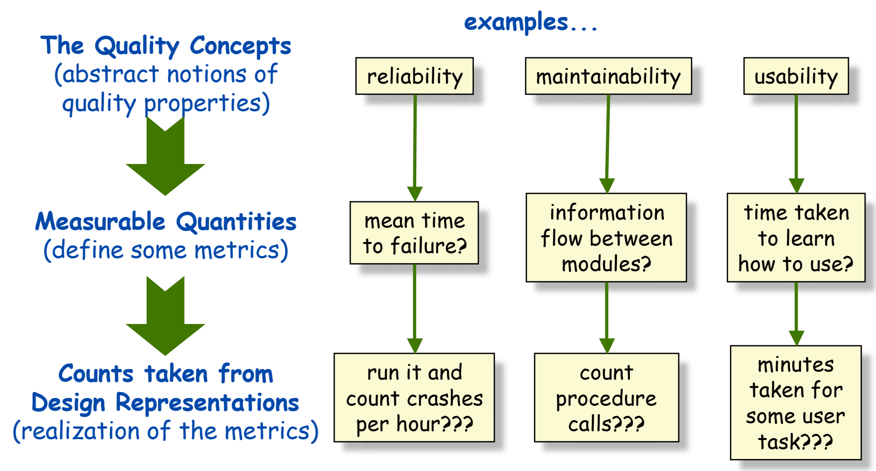
## Usability

- How easy is it to use?

# Measuring Quality

***Source:*** *Budgen, 1994, pp60-1*

**We have to turn our vague ideas about quality into measurables**



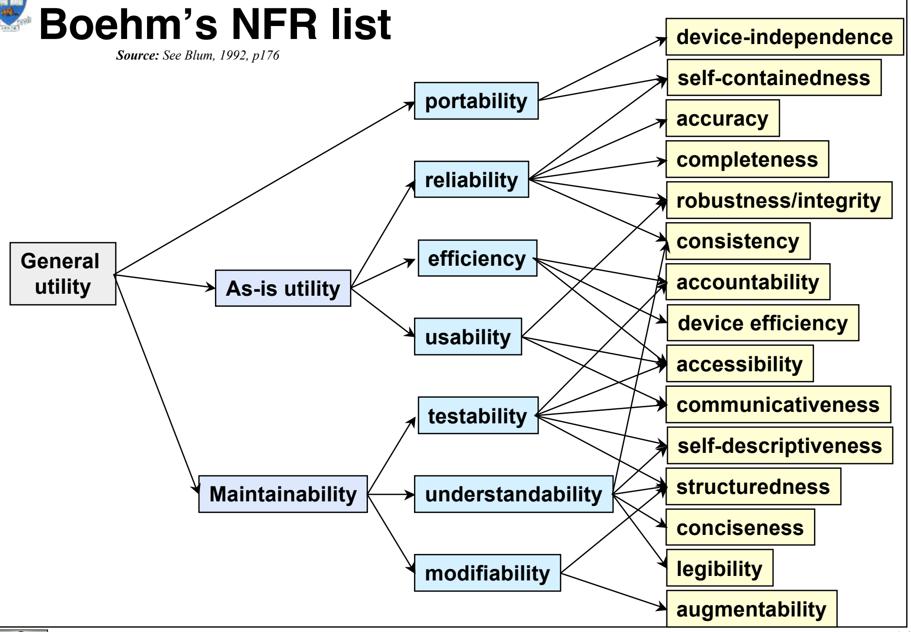






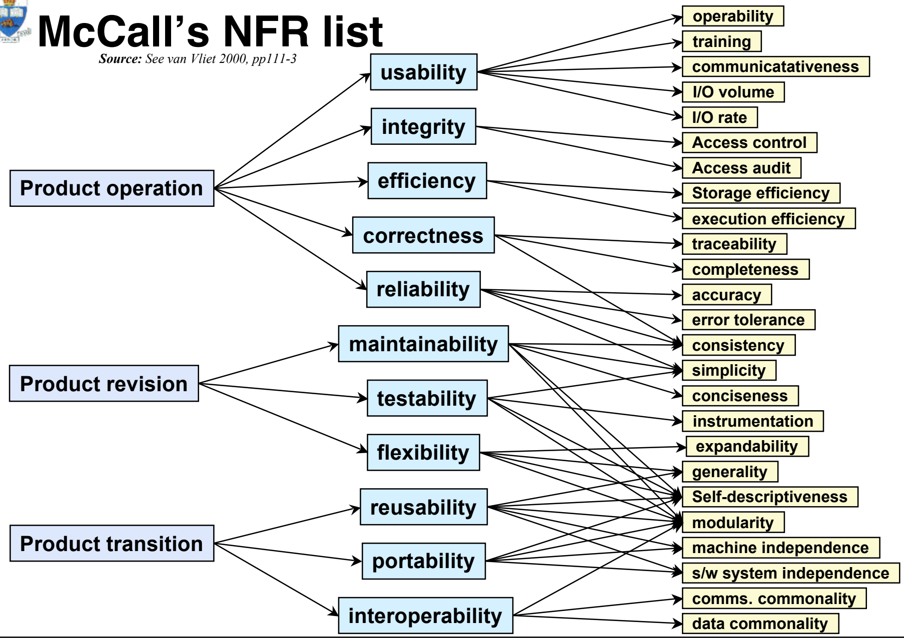
# Boehm's NFR list

***Source:*** *See Blum, 1992, p176*



# McCall's NFR list

***Source:*** *See van Vliet 2000, pp111-3*

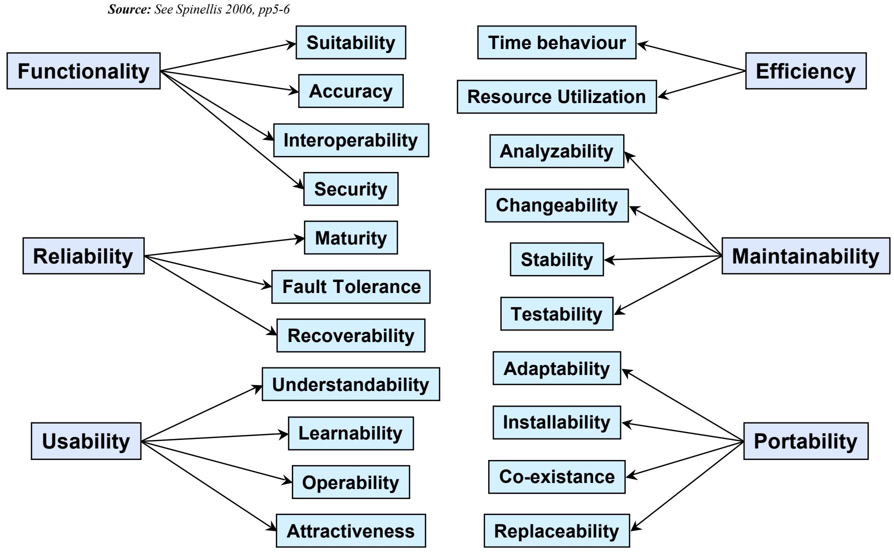


University of Toronto Department of Computer Science

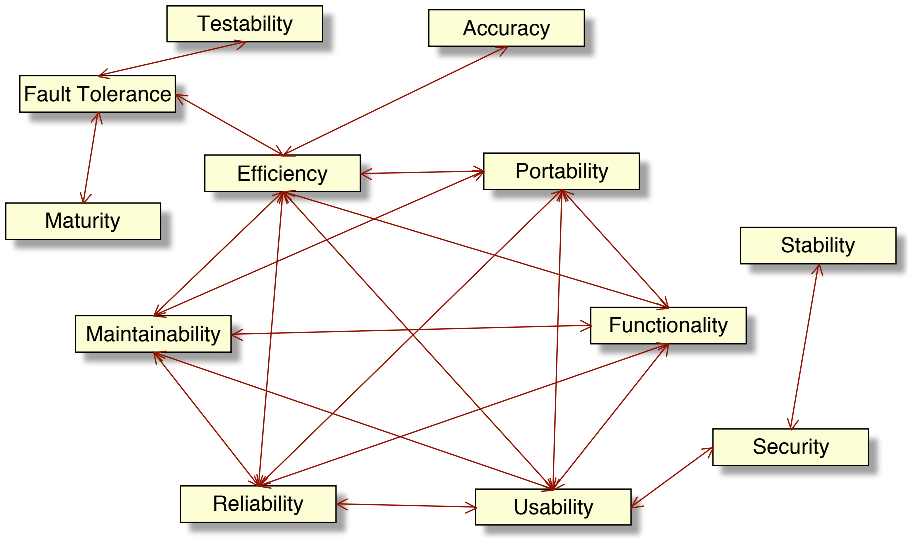
# ISO/IEC 9126

***Source:*** *See Spinellis 2006, pp5-6*



# Conflicts between Quality factors

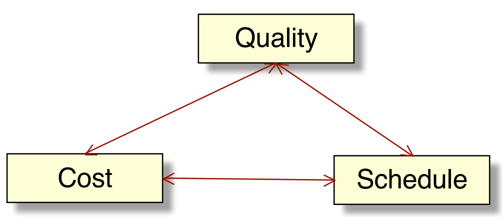








# More abstractly…




"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