Lecture 12: Software Design Quality

What is software quality?

How can it be measured?

- How can it be measured before the software is delivered?

Some key quality factors

Some measurable indicators of software quality

Quality

Think of an everyday object

- e.g. a chair
- How would you measure it's "quality"?
  - construction quality? (e.g. strength of the joints,…)
  - aesthetic value? (e.g. elegance,…)
  - fit for purpose? (e.g. comfortable,…)

All quality measures are relative

- there is no absolute scale
- we can say A is better than B but it is usually hard to say how much better

For software:

- construction quality?
  - software is not manufactured
- aesthetic value?
  - but most of the software is invisible
  - aesthetic value matters for the user interface, but is only a marginal concern
- fit for purpose?
  - Need to understand the purpose

Source: Budgen, 1994, pp58-9

Fitness

Design quality is all about fitness to purpose

- does it do what is needed?
- does it do it in the way that its users need it to?
- does it do it reliably enough? fast enough? safely enough? securely enough?
- will it be affordable? will it be ready when its users need it?
- can it be changed as the needs change?

But this means quality is not a measure of software in isolation

- it is a measure of the relationship between software and its application domain
  - might not be able to measure this until you place the software into its environment…
  - …and the quality will be different in different environments!
- during design, we need to be able to *predict* how well the software will fit its purpose
  - we need to understand that purpose (requirements analysis)
  - we need to look for quality predictors

Can you measure quality from the representation?

image courtesy of www.jsbach.net

Measuring Quality

Source: *Budgen, 1994, pp60-1*

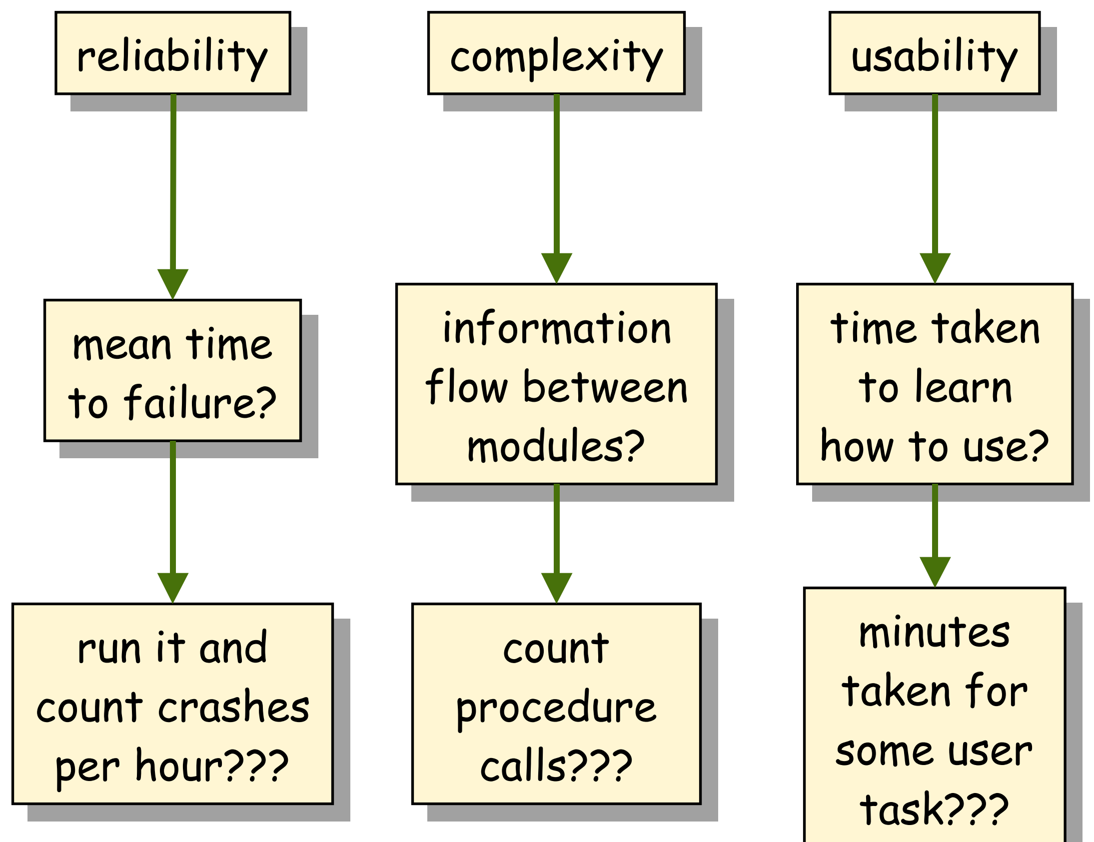
We have to turn our vague ideas about quality into measurables

The Quality Concepts
(abstract notions of quality properties)

Measurable Quantities
(define some metrics)

Counts taken from Design Representations
(realization of the metrics)

examples...



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

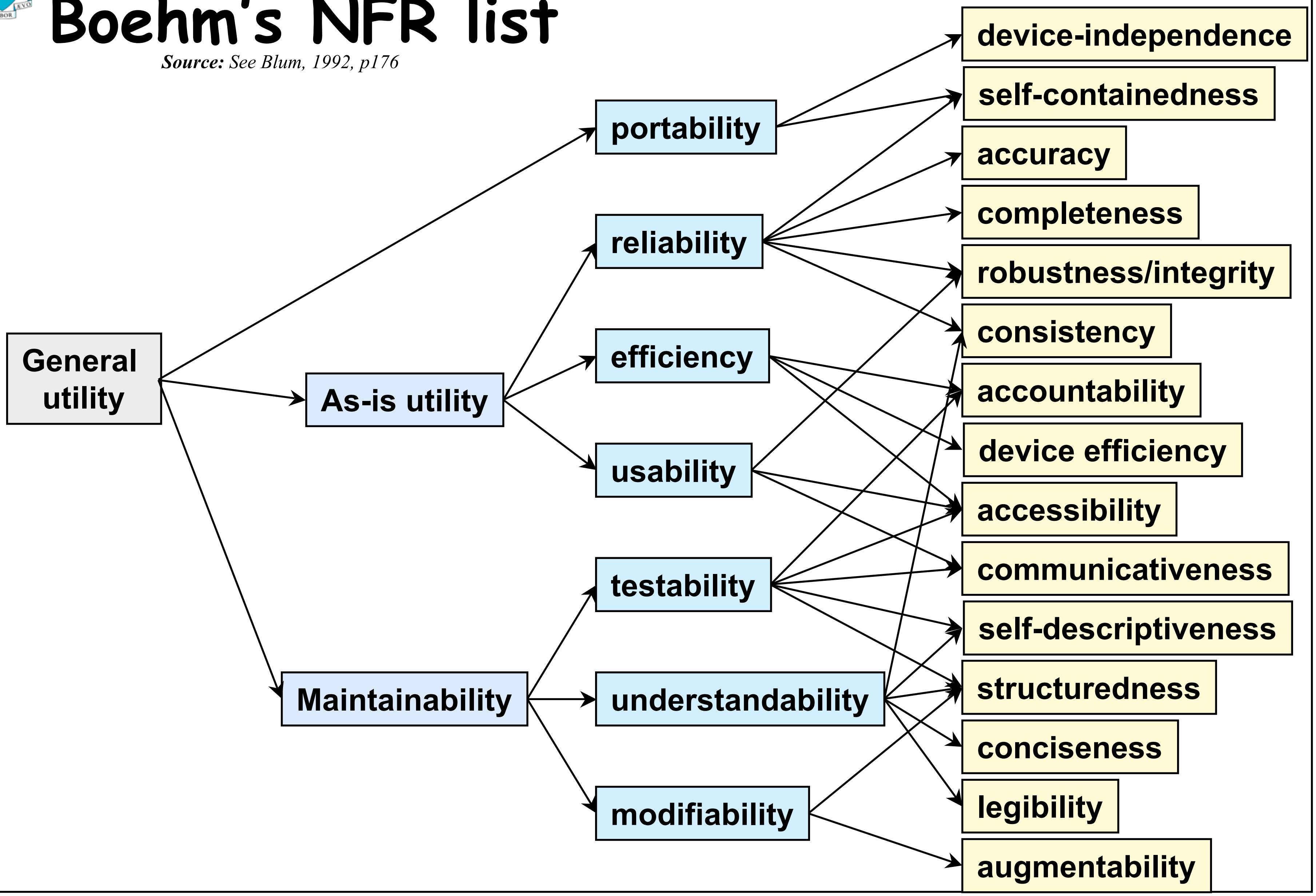
- How easy will it be to modify in the future?
  - perfective, adaptive, corrective

Usability

- How easy is it to use?

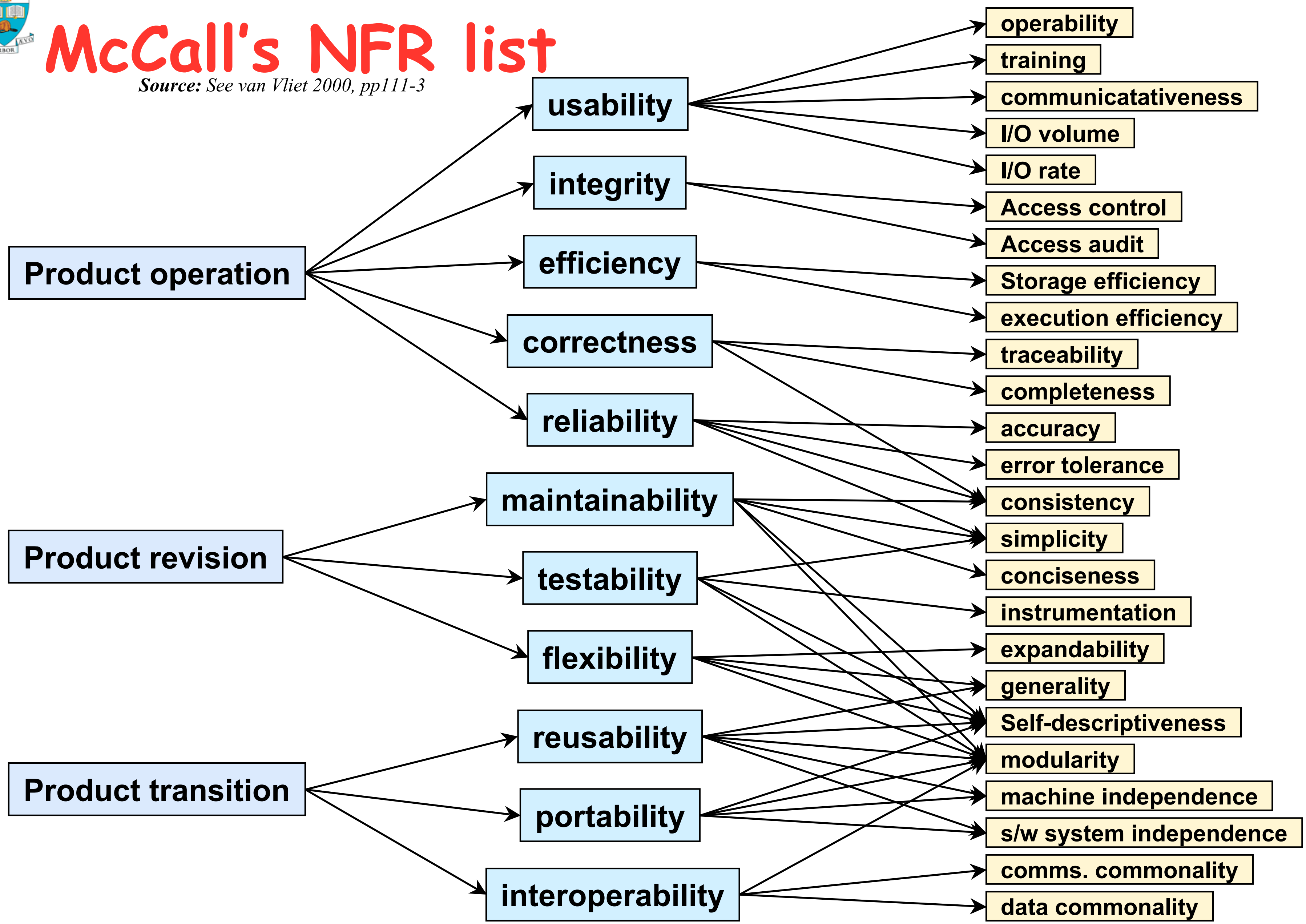
Boehm's NFR list

Source: *See Blum, 1992, p176*



McCall's NFR list

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



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)

Coupling

Source: *See van Vliet 2000, pp301-2*

Given two units *(e.g. methods, classes, modules, …)*, A and B:

form	features	desirability
data coupling	A & B communicate by simple data only	High (uses parameter passing, only pass necessary info)
stamp coupling	A & B use a common type of data	OK (but should they be grouped in a data abstraction?)
control coupling (activating)	A transfers control to B by procedure call	Necessary
control coupling (switching)	A passes a flag to B to tell it how to behave	Undesirable (why should A interfere like this?)
common environment coupling	A & B make use of a shared data area (global variables)	Undesirable (if you change the shared data, you have to change both A and B)
content coupling	A changes B's data, or passes control to the middle of B	Extremely foolish (almost impossible to debug!)

How well do the contents of a procedure (module, package,…) go together?

Cohesion

form	features	desirability
data cohesion	all part of a well-defined data abstraction	very high
functional cohesion	all part of a single problem-solving task	high
sequential cohesion	outputs of one part form inputs to the next	Okay
communicational cohesion	operations that use the same input or output data	moderate
procedural cohesion	a set of operations that must be executed in a particular order	low
temporal cohesion	elements must be active around the same time (e.g. start up)	low
logical cohesion	elements perform logically similar operations (e.g. printing things)	no way!!
coincidental cohesion	elements have no conceptual link other than repeated code	no way!!

Source: *van Vliet 1999, pp299-300 (after Yourdon & Constantine)*

Typical cohesion problems

Syntactic structure

- cohesion is all about program semantics
- if you use syntactic measures to decide how to design procedures…
  - e.g. length, no of loops, etc
- …your design will lack coherence

Hand optimization

- removing repeated code is often counter-productive
- it makes the program harder to modify
- *unless the repeated code represents an abstraction*

Complicated explanations

- if the only way to explain a procedure is to describe its internals…
  - …it is probably incoherent
- look for simple abstractions that can be described succinctly

Naming problems

- if it is hard to think of a simple descriptive name for a procedure…
  - …it is probably incoherent

How to spot incoherent designs

Source: *Liskov & Guttag 2000, chapter 14.*

An abstraction's effects clause is full of 'and's

e.g.

> effects: initialize the data structures and initialize the screen display and initialize the history stack and initialize the layout defaults and display an introductory text

Unless there is a strong functional link, use separate procedures

- temporal cohesion (bad)
- logical cohesion (very bad)

An effects clause contains 'or's, 'if…then…else's, etc.

e.g.

> effects: if x=0 then returns size(a[]) else if x=1 then returns sum(a[]) else if x=2 then returns mean(a[]) else if x=3 then returns median(a[])

These should be separate procedures

- control coupling by switching (bad)
- coincidental cohesion (very bad)
- logical cohesion (very bad)

Summary

- **Software quality generally means fitness for purpose**
  - need to know what that purpose is…
  - …what functions must it perform
  - …what other properties must it have (e.g. modifiability, reliability, usability…)
- **Not all quality attributes can be measured during design**
  - because quality is not an attribute of software in isolation
  - but we can look for predictors
- **Reliability, efficiency, maintainability, usability**
  - are usually the four most important quality factors
  - …although different authors give different lists
- **Modularity is often a good *predictor* of quality**
  - measure it by looking at cohesion and coupling

References

van Vliet, H. "Software Engineering: Principles and Practice (2nd Edition)" Wiley, 1999.

Chjapter 6 introduces the key ideas about software quality. Section 11.1 covers design considerations such as modularity, coupling and cohesion.

Budgen, D. "Software Design", 1994.

The neat book is one of the best introductions to the idea of "quality" software design that I've come across. Chapters 4 and 6 give a good overview of software design quality

Liskov, B. and Guttag, J., "Program Development in Java: Abstraction, Specification and Object-Oriented Design", 2000, Addison-Wesley.

chapter 14 is a nice summary of how to assess the quality of a piece of software.

Pirsig, R. M., "Zen and the Art of Motorcycle Maintenance : An Inquiry into Values", 1974, William Morrow & Company.

This is a novel about one man's quest to understand what "quality" is really all about. Great bedtime reading!