Software 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
Fitness
Source: Budgen, 1994, pp58-9

→ 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?

*Sonata a Violino solo senza Basso,*

J. S. Bach

*image courtesy of www.jsbach.net*
Measuring Quality

We have to turn our vague ideas about quality into measurables

The Quality Concepts
(abstract notions of quality properties)

Meanable Quantities
(define some metrics)

Counts taken from Design Representations
(realization of the metrics)

Examples...

- Reliability
  - Mean time to failure?
  - Run it and count crashes per hour??

- Complexity
  - Information flow between modules?
  - Count procedure calls??

- Usability
  - Time taken to learn how to use?
  - Minutes taken for some user task??
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?
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:

<table>
<thead>
<tr>
<th>Form</th>
<th>Features</th>
<th>Desirability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data coupling</td>
<td>A &amp; B communicate by simple data only</td>
<td>High (use parameter passing &amp; only pass necessary info)</td>
</tr>
<tr>
<td>Stamp coupling</td>
<td>A &amp; B use a common type of data</td>
<td>Okay (but should they be grouped in a data abstraction?)</td>
</tr>
<tr>
<td>Control coupling (activating)</td>
<td>A transfers control to B by procedure call</td>
<td>Necessary</td>
</tr>
<tr>
<td>Control coupling (switching)</td>
<td>A passes a flag to B to tell it how to behave</td>
<td>Undesirable (why should A interfere like this?)</td>
</tr>
<tr>
<td>Common environment coupling</td>
<td>A &amp; B make use of a shared data area (global variables)</td>
<td>Undesirable (if you change the shared data, you have to change both A and B)</td>
</tr>
<tr>
<td>Content coupling</td>
<td>A changes B’s data, or passes control to the middle of B</td>
<td>Extremely Foolish (almost impossible to debug!)</td>
</tr>
</tbody>
</table>
**Cohesion**

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

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

<table>
<thead>
<tr>
<th>Form</th>
<th>Features</th>
<th>Desirability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data cohesion</td>
<td>all part of a well defined data abstraction</td>
<td>Very High</td>
</tr>
<tr>
<td>Functional cohesion</td>
<td>all part of a single problem solving task</td>
<td>High</td>
</tr>
<tr>
<td>Sequential cohesion</td>
<td>outputs of one part form inputs to the next</td>
<td>Okay</td>
</tr>
<tr>
<td>Communicational</td>
<td>operations that use the same input or output data</td>
<td>Moderate</td>
</tr>
<tr>
<td>cohesion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedural cohesion</td>
<td>a set of operations that must be executed in a particular order</td>
<td>Low</td>
</tr>
<tr>
<td>Temporal cohesion</td>
<td>elements must be active around the same time (e.g. at startup)</td>
<td>Low</td>
</tr>
<tr>
<td>Logical cohesion</td>
<td>elements perform logically similar operations (e.g. printing things)</td>
<td>No way!!</td>
</tr>
<tr>
<td>Coincidental cohesion</td>
<td>elements have no conceptual link other than repeated code</td>
<td>No way!!</td>
</tr>
</tbody>
</table>
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


→ 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)