SOFTWARE QUALITY METICS

What should be measured?

Product Quality
- Software Quality

Process Quality

Product Quality

efficiency

- Product Quality ?
  code size

maintenance

- Recent studies have reported that something like 70% of total software cost is devoted to maintenance
  - This is therefore a major quality consideration

Maintenance and Product Quality Factors

- Two types of Maintenance each reflecting software quality factors:

1. Changes required because customer not satisfied with the system delivered.

- Quality Factors

CORRECTNESS:

Ability of software to exactly perform its tasks, as defined by the Requirements Specification

ROBUSTNESS:

Even if the system performed exactly to the Requirements Specification, given the lack of formal specification techniques, there will always be situations that the specification does not cover.

Ability of software to function in abnormal conditions.

e.g. In the case of certain Hardware Failures, the

software should report the failure and proceed with reduced functionality.

2. Changes required because World changes

- Quality Factors

|

EXTENDIBILITY:

The ease with which software products may be adapted to changes of

specification.

Process Quality - Development

- Studies have revealed software costs to be running in the region of 60- 70% of total production cost.

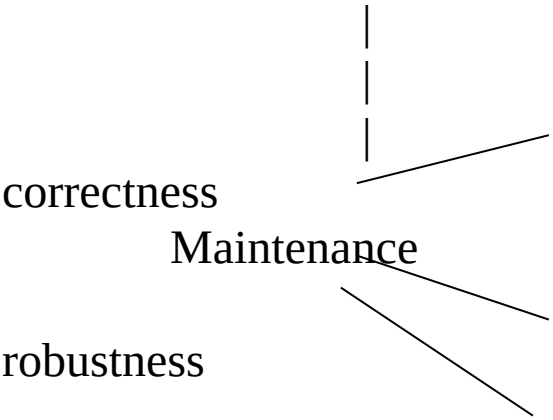
- In general software lacks **REUSABILITY**

- Ability to be reused, in whole or in part, for new applications.

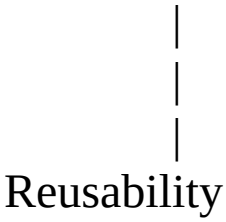
Conclusion

- With the dramatic reduction in hardware costs the following issues now dominate the software development process.

1. Quality of product



2. Quality of process



- These quality factors are not well understood at the moment.

- We need to consider the nature of design to enable at least the classification of features that may affect these factors.

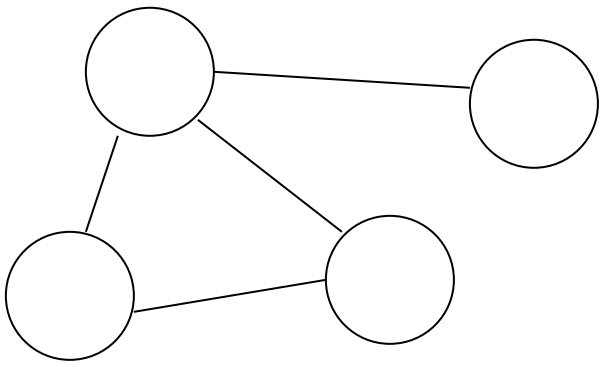
NATURE OF DESIGN

Modularity

- It was recognised early on that large monolithic designs rarely produced quality systems.

- Complexity needed to be managed

- 'Divide and Conquer' strategy
  - Design systems in terms of modules which cooperate to perform system functions.



Architecture of Design

Key Question:

- What characteristics of the architecture should be used to relate to quality of product and process?

- Can we identify characteristics of an architecture that we can use to give some measure of a quality factor.

- Example:

  - In a building, does the average distance to the nearest stairs say anything about the quality of safety?

  - In a software architecture does the level of communication between modules say anything about robustness?

ARCHITECTURAL CONSIDERATIONS

- Various researchers have suggested that the following are important characterisitcs of an architecture

Mapping of Problem Domain into Implementation

- Changes are requested at the User level in terms of domain entities. But change actually happens at the code level.

    - Example: In a library system
      " a **book** should now be marked as late if the **Time on Loan** is greater than 3 days "

- Relationship or mapping between code and domain should therefore be clear.

- So we require clear
  - Mapping of domain to design modules of architecture
  - Mapping of design modules to code

- Example: Change to user interface required.

- Mapping of domain to design modules of architecture

    - Change to input/output module of design architecture.

- Mapping of design modules to code

    - Easily find input/output processing in code?

                                                |

YES -if have input/output processing is clearly written into a defined coding unit e.g procedure or object

NO -if input/output processing has been spread over several coding units destroying the modularity of the processing.

Coupling and Cohesion

- Coupling - what goes on between modules
- Cohesion - what goes on within modules, how focused is the module.
- These are obviously related
  - Good coupling/cohesion means
    - Few Interfaces/ Small Interfaces
    - Every module minimises communication.

Information Hiding

- Access to information must be controlled.

- Modules must not be allowed to directly gain access to the internals of other modules.

- Access must be through an explicitely defined interface.
  - Explicitely Interfaces.

Information Hiding Example

- Module A requires the input/output module to change the size of window on the screen. This is controlled by a variable SIZE in the input/output module.

- Module A must not be allowed direct access to variable SIZE.

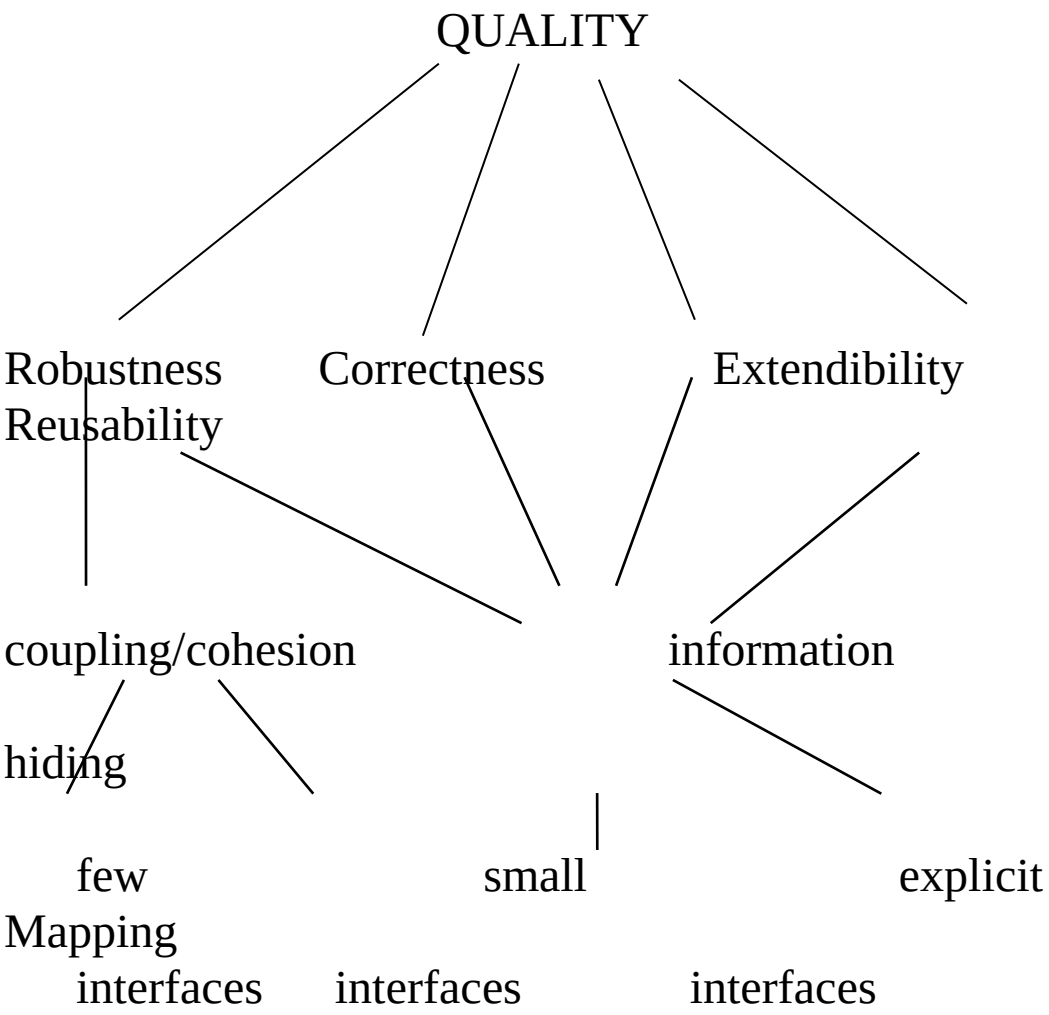
- If many modules did this, each one could change the variable in its own way
  - maybe to a size that is not possible.

- The input/output module has all the knowledge about the screen
  - so it should be the only modulecapable of directly accessing SIZE.

- Other modules must **REQUEST** a change, by accessing part of an **INTERFACE** that the input/output module provides.

- The SIZE information is **HIDDEN** and protected behind **explicit interface.**

Relationship between quality factors



- The above relationships are just a simple example of the process to map high level quality featues to measuarable quantities.

- Most Software Engineering books will describe more complex relations – see Pressman.