SWE 530:
Software Design Process
Design Qualities

Dr. H. Birkan YILMAZ
Department of Computer Engineering
Boğaziçi University

(birkan.yilmaz@bogazici.edu.tr)

Adapted from slides of Dr. Albert Ali Salah

Recap

- We looked at:
  - design in the context of particular life-cycle models, most notably the linear (waterfall) and incremental models;
  - the associated role of requirements elicitation in setting the context for the various design activities;
  - the roles that software prototyping can play in supporting aspects of the design process;
  - the question of how well a design meets the users' needs, and the roles of:
    - verification: are we building the product right?
    - validation: are we building the right product?
  - the cost of fixing errors in design at a later stage of development;
  - the influence of the *maintenance* phase

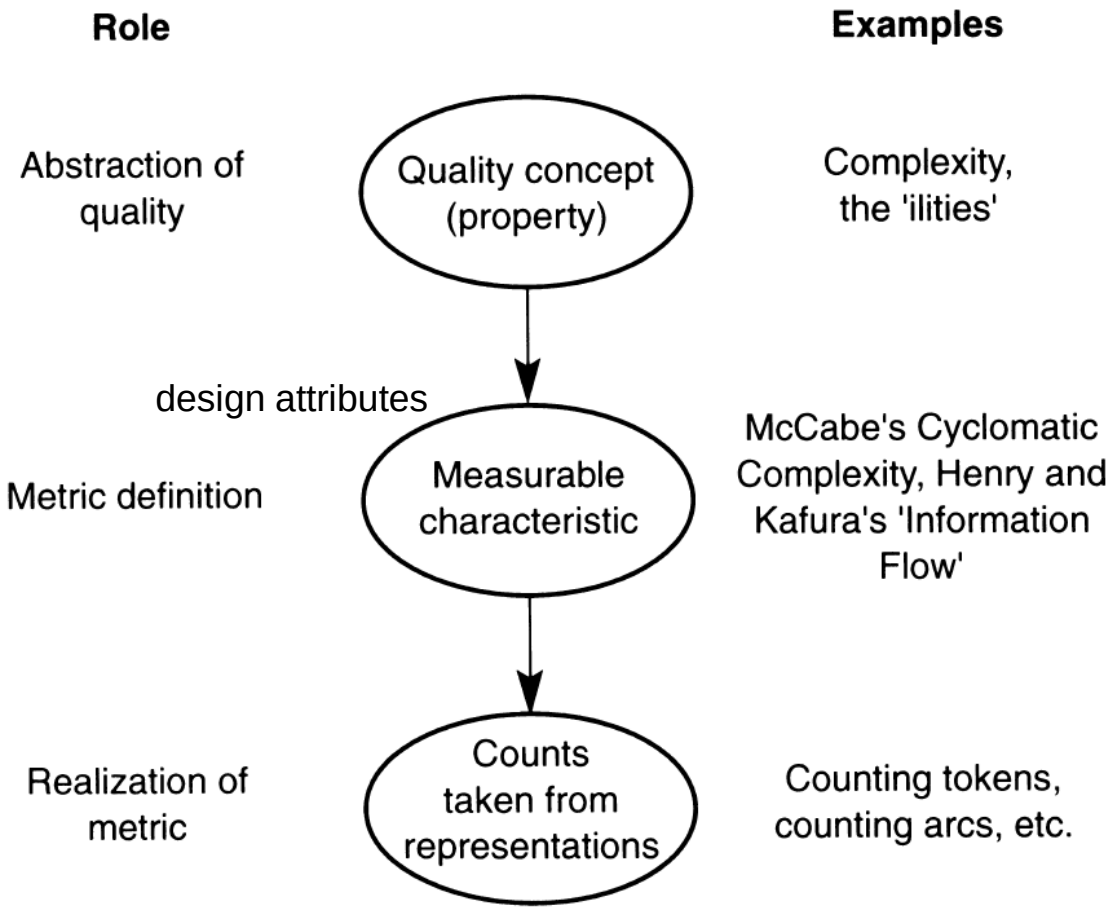
Overview

- The quality concept
- Assessing design quality
- Quality attributes of the design product
- Assessing the design process

The quality concept

- Qualitative vs. Quantitative
- Fashion
- Actual implementation cannot be neglected
- Software has static and dynamic attributes
- Elegance of form vs. function
- Fitness for purpose

Mapping quality concepts to measurement



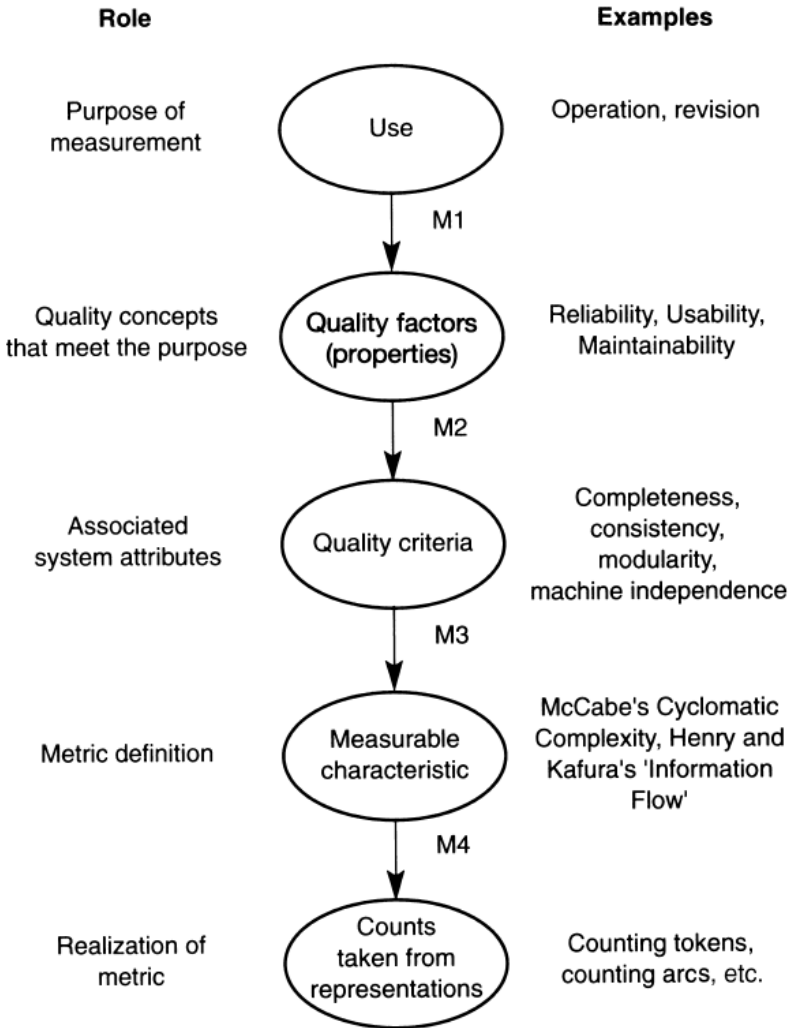
McCabe's Cyclomatic Complexity measure

- Counting lexical tokens such as IF and WHILE
- Numerical value does NOT imply a ratio scale!
- (six is not twice complex as three...)

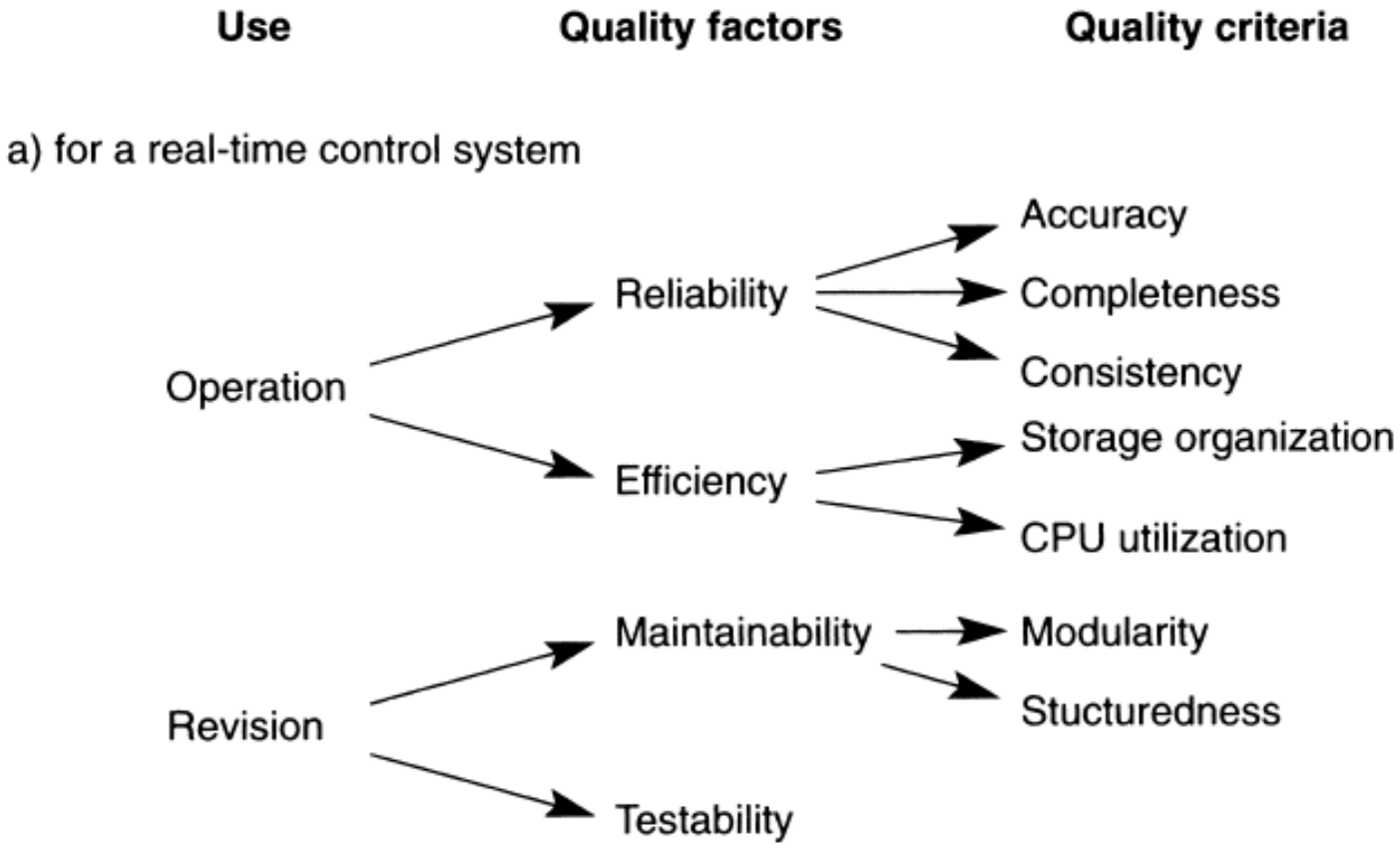
Activity

- Find five more measures of software complexity

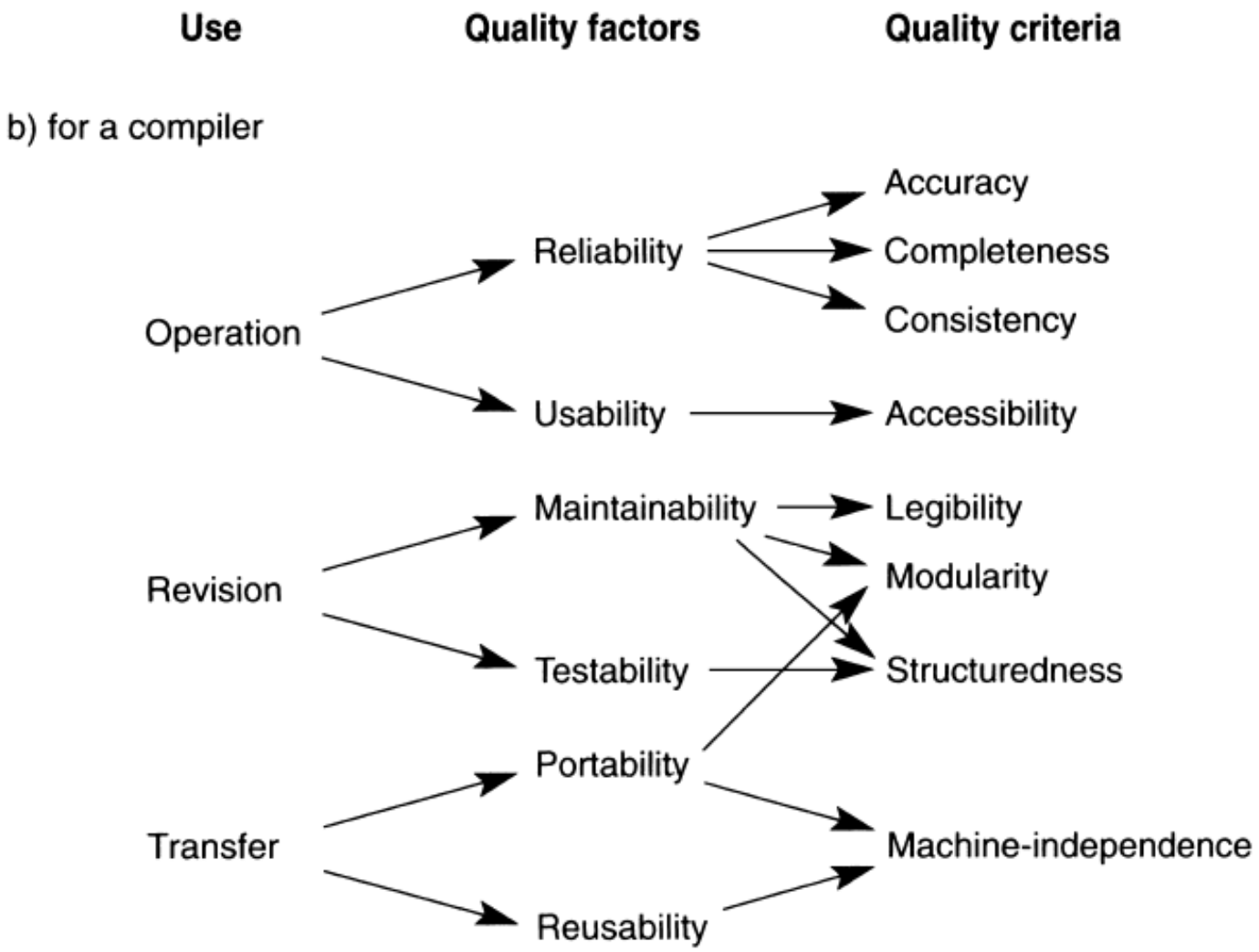
Mapping quality concepts to measurement II



Example: Real-time control system



Example: Compiler



Considerations

- The needs of the a system of measurement may run counter to the needs of the designer

- It is difficult to measure abstractions
- Abstract design notations have poor syntax
- Dynamic attributes are harder to measure at the design stage

The ‘ilities’

- reliability
- efficiency
- maintainability
- usability

Reliability

- ***complete*****,** in the sense of being able to handle all combinations of events and system states;
- ***consistent*****,** in that its behaviour will be as expected, and will be repeatable, regardless of the overall system loading at any time;
- ***robust*** when faced with component failure or some similar conflict (for example, if the printer used for logging data in a chemical process-control plant fails for some reason, this should not be able to ‘hang’ the whole system, but should be handled according to the philosophy summed up in the term ‘graceful degradation’).

Efficiency

- The efficiency of a system can be measured through its use of resources such as processor time, memory, network access, system facilities, disk space and so on.

Maintainability

- Allowing future modifications
- Related to implementation, and detailed design decisions:
  - choice of identifiers,
  - comment structuring practices,
  - documentation standards

Usability

- Designing for ‘ease of use’
- User experience
- Interaction through computers

Cognitive dimensions

Dimension	Interpretation
Abstraction	Types and availability of abstraction mechanisms
Hidden dependencies	Important links between entities are not visible
Premature commitment	Constraints on the order of doing things
Secondary notation	Extra information provided via means other than formal syntax
Viscosity	Resistance to change
Visibility	Ability to view components easily
Closeness of mapping	Closeness of representation to domain
Consistency	Similar semantics are expressed in similar syntactic forms
Diffuseness	Verbosity of language
Error-proneness	Notation invites mistakes
Hard mental operations	High demand on cognitive resources
Progressive evaluation	Work-to-date can be checked at any time
Provisionality	Degree of commitment to actions or marks
Role-expressiveness	The purpose of a component is readily inferred

Premature commitment (and enforced lookahead)

- Happens when an early design decision which was made before proper information was available has the effect of constraining later ones
- Hierarchical menus in a spoken telephone response system is an example
- This concept can provide a ‘measure’ of the quality of the design *process*, rather than of the design *product*

Hidden dependencies

- This concept describes a relationship between two components, such that while one is dependent upon the other the relationship is not fully visible.
- When components share representations, for instance...
- If design is modified during maintenance, the problem is aggravated.
- Relevant for websites and their maintenance

Viscosity

- This describes the concept of ‘resistance to change’.
- Can arise as a result of premature commitment
- Example: using embedded numeric constants (instead of symbolic constants)
- Not easily quantified

Quality attributes of the design product

- What measures of quality we actually have available to us at the design stage?
- Two problems of measurement:
  - to identify a set of design attributes that are related to these properties;
  - to find ways of extracting information about these attributes from the available forms of design document.

Design attributes

- There are some attributes and criteria that are widely used for design assessment
  - Simplicity
  - Modularity
  - Information-hiding
- There are many more such attributes

Simplicity

- Characteristic of almost all good designs
- ‘a solution should be as simple as possible, but no simpler’ (Einstein?)
- Usually measured by its converse, *complexity*
  - Complexity of control flow (number of possible paths of control during execution)
  - Complexity of structure in terms of information flow
  - Complexity of comprehension (number of identifiers and operators)
  - Complexity of structure (e.g. reln. between elements)

Simplicity

- There are no ready measures that can currently be used to help assess the architectural complexity of a design (other than for object-oriented forms)
- In terms of the design quality concepts, simplicity is clearly related to maintainability and testability as well as reliability and possibly efficiency.

Design attributes

- There are some attributes and criteria that are widely used for design assessment
  - Simplicity
  - Modularity
  - Information-hiding
- There are many more such attributes

Modularity

- The «analysis» approach: divide the problem into meaningful (or well-defined) units.
- APIs, libraries, standardized interfaces
- Adopt a design practice based on a **separation of concerns:** group functions within modules in such a way that their interdependence is minimized

Modularity

- Benefits:
  - modules are easy to replace;
  - each module captures one feature of a problem, so aiding comprehension (and hence maintenance), as well as providing a framework for designing as a team;
  - a well-structured module can easily be reused for another problem.
- Related to maintainability, testability and (possibly) to usability and reliability

Modularity

- Two useful quality measures for modularity:
  - **Coupling** is a measure of intermodule connectivity, and is concerned with identifying the forms of connection that exist between modules and the ‘strength’ of these connections.
  - **Cohesion**, in its turn, provides a measure of the extent to which the components of a module can be considered to be ‘functionally related’. The ideal module is one in which all the components can be considered as being solely present for one purpose.

Forms of coupling

Form	Features	Desirability
Data coupling	Modules A and B communicate by parameters or data items that have no control element	High
Stamp coupling	Modules A and B make use of some common data type (although they might perform very different functions and have no other connections)	Moderate
Control coupling (i) Activating	A transfers control to B in a structured manner such as by means of a procedure call	Necessary
(ii) Coordinating	A passes a parameter to B that is used in B to determine the actions of B (typically a boolean 'flag')	Undesirable
Common-environment coupling	A and B contain references to some shared data area that incorporate knowledge about its structure and organization. Any change to the format of the block requires that all of the modules using it must also be modified	Undesirable

Coupling

- Biased towards structures like procedures and subprograms (as opposed to classes), but can be applied to the «uses» hierarchy
- Quantifying coupling (and cohesion) systematically is difficult
- Example of coordinating control coupling:
  - When a procedure can do one of two things, indicated by a «switch» parameter
  - Typically introduced to fix a design error
  - Example: Multiple passes in LaTeX compilation

Cohesion

- **Cohesion**, provides a measure of the extent to which the components of a module can be considered to be ‘functionally related’. The ideal module is one in which all the components can be considered as being solely present for one purpose.

Forms of cohesion

Form	Features	Desirability
Functional	All the elements contribute to the execution of a single problem-related task	High
Sequential	The outputs from one element of the module form the inputs to another element	Quite high
Communicational	All elements are concerned with operations that use the same input or output data	Fairly
Procedural	The elements of the module are related by the order in which their operations must occur	Not very
Temporal	The elements involve operations that are related by the time at which they occur, usually being linked to some event such as 'system initialization'	Not very
Logical	The elements perform operations that are logically similar, but which involve very different actions internally	Definitely not
Coincidental	The elements are not linked by any conceptual link (such modules may be created to avoid having to repeat coding tasks)	Definitely not

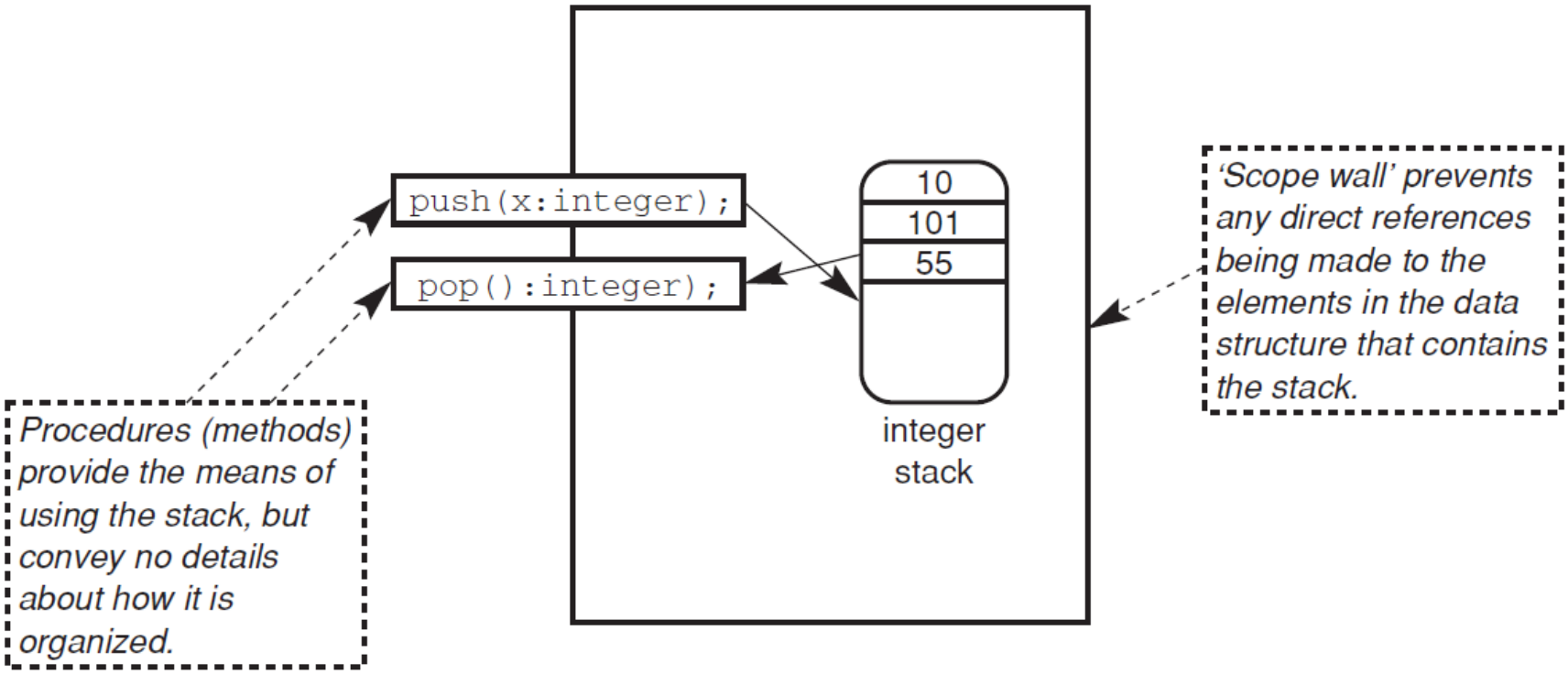
Design attributes

- There are some attributes and criteria that are widely used for design assessment
  - Simplicity
  - Modularity
  - Information-hiding

Information-hiding

- Related to modularity, but it also incorporates additional notions about managing information in a system.
- Keep information about the detailed forms of such objects as data structures and device interfaces local to a module, or unit, and to ensure that such information should not be made ‘visible’ outside that unit
- Prevent programmer shortcuts

Example of information-hiding



Design attributes

- There are some attributes and criteria that are widely used for design assessment
  - Simplicity
  - Modularity
  - Information-hiding
- What about undesirable design attributes?

Undesirable design attributes

- Having many copies of ‘state’ information spread around the system
- Using interfaces that are too complex
- The use of excessively complex control structures
- Needless replication of information
- Using modules that lack ‘functional strength’

Assessing design quality

- Use the «review» (or walkthrough, or inspection)
  - Technical review: ‘mental execution’ of the design model, assessing both static and dynamic aspects
  - Management review: assessing deadlines, deliverables, schedules
  - Prototyping: gaining domain knowledge and experience

Technical reviews

‘inspection is a very bad form of error removal – but all the others are much worse’ – Glass’99

- Code inspection vs. design inspection
- You need people with technical knowledge + domain knowledge!

Eight requirements for good design

- **well structured:** consistent with chosen properties such as information-hiding;
- **simple:** to the extent of being ‘as simple as possible, but no simpler’;
- **efficient:** providing functions that can be computed using the available resources;
- **adequate**: meeting the stated requirements;
- **flexible**: able to accommodate likely changes in the requirements, however these might arise;
- **practical**: module interfaces should provide the required facilities, neither more nor less;
- **implementable**: using current and available software and hardware technology;
- **standardized**: using well-defined and familiar notation for any documentation.

Parnas and Weiss, 1987

Summary

- Software quality concepts are for assessing the static structure and the dynamic behaviour of the system.
- The ultimate goal of quality must be that of **fitness for purpose**; the criteria will be both problem-dependent and domain-dependent.
- Abstraction is an important tool, but it makes it difficult to make any direct product measurements during design.
- Technical design reviews can provide a valuable means of obtaining and using domain knowledge (for both product **and** process)