Lecture 6

Software Quality

Today’s Lecture

What is a good design?

- The general theory of quality
- Software quality
  - Software quality models
  - Effects of design on software quality
- Quality of software design

The Notion of Quality

- Garvin’s definitions of quality
  - **Transcendent view**: Quality is universally recognisable. It is related to a comparison of features and characteristics of products.
  - **Product-based view**: Quality is a precise and measurable variable. Differences in quality reflect the differences in quantities of some product attributes.
  - **User-based view**: Quality is the fitness of intended uses.
  - **Manufacturing-based view**: Quality is conformation to the specifications.
  - **Value-based view:** A quality product is one that provides performance at an acceptable price or conformance at an acceptable cost.

Quality, in particular software quality, is an elusive concept and varying from people to people.

Software Quality Models

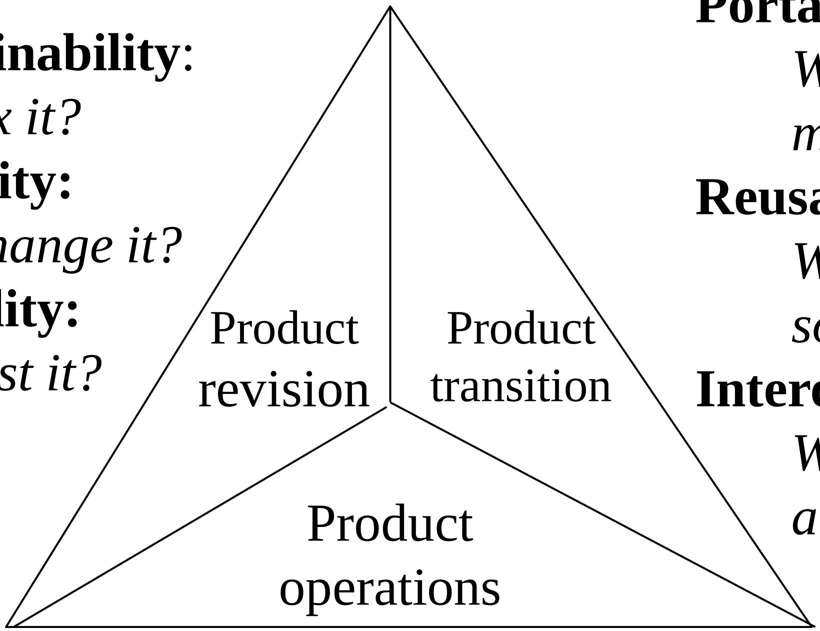
- Models about software quality in terms of
  - The factors that affect software quality
    - Attributes directly indicate the quality of he system, e.g. reliability, correctness, user friendliness, etc.
    - Attributes indirectly related to quality, e.g. internal complexity, etc.
  - The interrelations between the factors
    - Causality models:
      - Causal relationship, in terms of stereo-type relations
    - Quantitative models:
      - Quantitative relations expressed as numerical functions,
      - Numerical values of the basic attributes are given, e.g. through using metrics/measurements
      - The overall quality in an attribute on a more high level of abstraction is calculated numerically

Hierarchical Models of SQ

- A set of quality related properties organised into a hierarchical structure
  - E.g. decompose quality into a number of quality attributes and attributes into a number of factors, and then a number of measures, etc.
  - Positive causal relationship is represented
- Examples:
  - McCall model (1977)
  - Boehm model (1978)
  - ISO model (1992)
  - Bansiya-Davis model of OO designs (2002), etc.

McCall’s Model

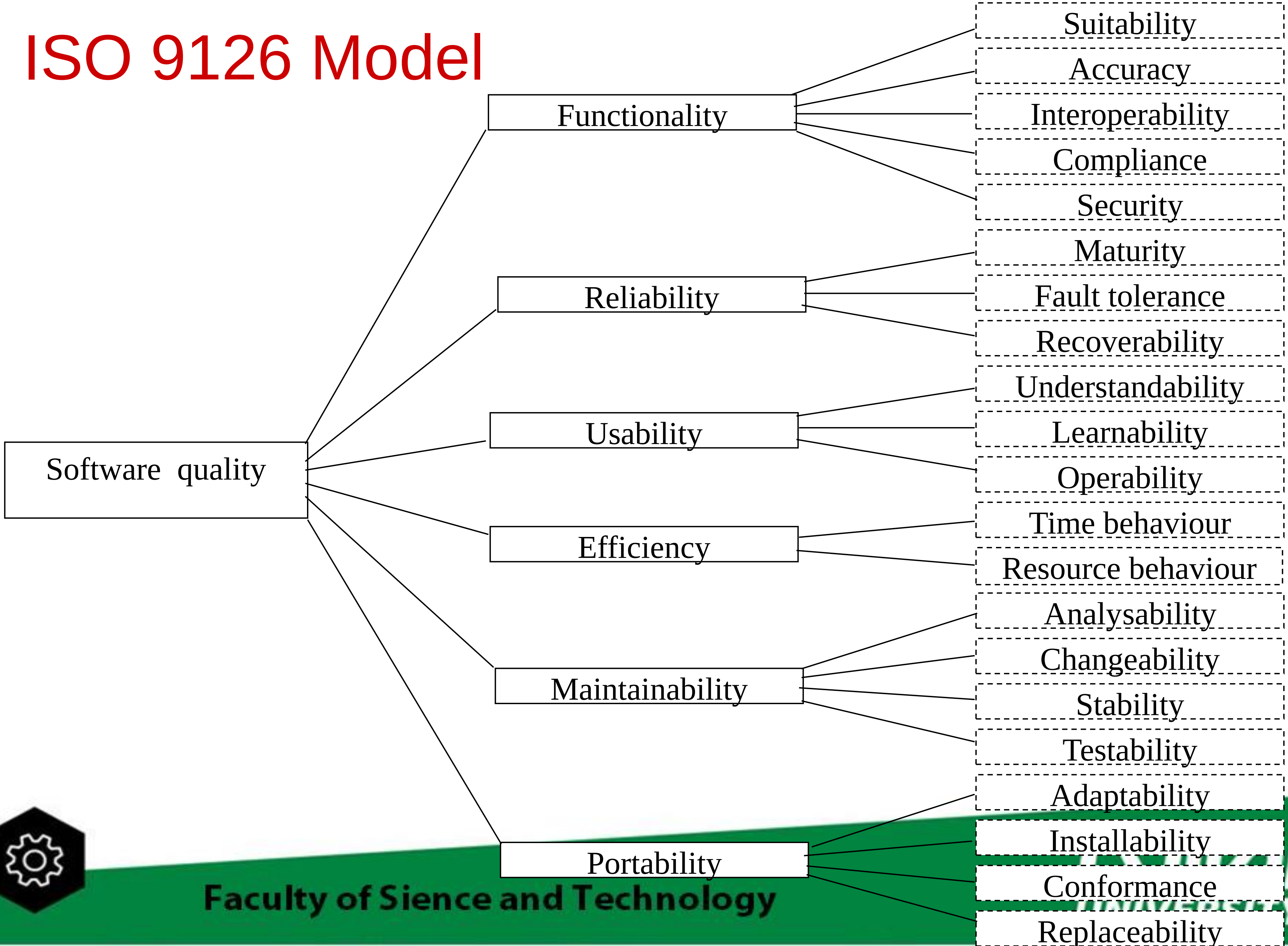
Maintainability:
Can I fix it?
Flexibility:
Can I change it?
Testability:
Can I test it?



Portability:
Will I be able to use it on another machine?
Reusability:
Will I be able to reuse some of the software?
Interoperatability:
Will I be able to interface it with another machine / software?

Correctness: *Does it do what I want?*
Reliability: *Does it do it accurately all the time?*
Efficiency: *Will it run on my machine as well as it can?*
Integrity: *Is it secure?*
Usability: *Can I run it?*

ISO 9126 Model



Relational Models of SQ

- A quality model consists of:
  - A number of quality attributes and factors, etc.
  - A set of stereo types of relationships between them, normally include
    - *Positive relation*:

      High on one aspect of quality implies inevitably high on the other
    - *Negative relation:*

      High on one aspect of quality implies inevitably low on the other
    - *Neutral relation:*

      High or low on one aspect of quality does not imply on the other aspect the quality is high or low.
- Examples:
  - Perry model (1991)
  - Gillies model (1992, 1997)

Perry's Model

	C	R	E	I	U	M	T	F	P	R	I
Correctness											
Reliability	○										
Efficiency											
Integrity			●								
Usability	○	○	●	○							
Maintainability	○	○	●		○						
Testability	○	○	●		○	○					
Flexibility	○	○	●		○	○	○				
Portability			●			○	○				
Reusability		●	●	●		○	○	○	○		
Interoperability			●	●					○		

- ○ Direct
- ● Inverse
- □ Neutral

Notes on Software Quality Attributes

- Not all attributes are of equal importance
- An attribute may have different importance in different software systems
- The importance of a quality attribute for a given software may change as time passes
- Relationships between attributes are very complicated; some are positive; some are negative
- Different people may have different views on quality

Alan Gillies, *Software Quality: Theory and Management*, Second Edition, International Thomson Computer Press, 1997.

Key Features of Existing SQ Models

Feature	Pros	Cons
Universal	Applicable to all software systems	Not address system specific issues
Simplicity	Stereo types of relationships between quality attributes	Incapable of dealing with complicated relationships between quality attributes
Black-box	Treat software systems as black-boxes	Provide little help to the designers to consider system’s structure
Empirical	Developed-based on empirical knowledge from experiences	Cannot be validated or verified of its correctness

Uses of Software Quality Models

- Understanding software quality
  - To measure an existing system’s quality,
  - To predict a system’s quality during development
  - To analyse quality problems during development and/or in operation of a systems
- Guidelines to development activities
  - To elicit users’ requirements on software quality
  - To perform quality assurance activities in software development and maintenance

Effects of Design on Quality

- Software quality must be considered at all phases of software development, including analysis, design, implementation, deployment and operation/maintenance.
- Design is the first stage in software system creation in which quality requirements can begin to be addressed.
- Different quality attributes manifest themselves differently during various phases.
- Design is critical to many of the quality attributes of software systems, and these qualities should be designed in and evaluated at the design phase.
- Some quality attributes are not structure sensitive, thus attempting to achieve these qualities or analyse for them through structural design will not be fruitful.

Performance

The notion of performance:

The responsiveness of a system, i.e. the time required to respond to stimuli (events), or the number of events processed in certain interval of time.

- Effects of architectural design :
  - determines how components are distributed on network and whether they are executed in parallel
  - determines how much communication and interaction between the components of the system
- Effects of detailed design :
  - Affected by the choice of algorithms to implement selected functionality assigned to the components.
- Effects of implementation :
  - Affected by how the algorithms are coded.

Correctness and Reliability (I)

- The notion of correctness and reliability:

  The properties that relate to how well the software implements the specified users' requirements.

- Architectural design:
  - A good architectural design provides an understandable solution of the problem, hence helps to reduce the probability of errors made at lower level design and implementation.
  - Well-structured design helps at testability and maintainability so that errors in design and implementation can be tested and fixed.
  - Software fault-tolerant features can be introduced in architecture design to detect failures and to recover from failures at run-time, hence improve the reliability of the system.

Correctness and Reliability (II)

➢ Detailed design:

- ◆ Detailed designs have great impact on the probability that the programmer correctly implement the algorithms.

➢ Interface design:

- ◆ Good HCI design can prevent invalid input and misinterpretation of output, hence significantly reduce the probability of system failures caused by human operation errors.

Portability

The notion of portability:

The easiness of a software system to be transported from one hardware / software platform to another.

- Architectural design:
  - A well structured design should group environment dependent code in a small number of components so that the change on the code to move to another environment can be achieved by replacing such components with new ones rather than rewrite the whole systems.
- Detail design:
  - Algorithms and data structures should not heavily depend on the platform specific feature so that portability can be obtained.

Maintainability

The notion of maintainability:

- The easiness of maintaining a software system in both corrective and adaptive maintenances.
- Both corrective and adaptive types of maintenance operations require software engineers understand how the software system works so that bugs can be fixed and changes in the environment can be adapted.

Architectural design:

- Well-structured design helps software engineers to understand the system.

Reusability

The notion of reusability:

- ◆ The property that the components of a software system can be easily reused in the development of other software systems.
- ◆ Reusability depends on the generality of the components in a given application domain and the extent to which the components are parameterised and configurable.

➢Architectural design:

- ◆ It determines how the functionality of a software system is decomposed into components and how they are inter-connected.
- ◆ It play important role in component-based software developments.

➢Detailed design:

- ◆ Algorithm and data structure design determine how easily the components can be parameterised and the way to configure the components.

Quality Attributes of Designs

- Design has two facets:
  - The product:
    - The description of the solution to the problem and/or the model of the product
  - The process:
    - The description and/or model of the plan about how to make the product
- Design quality attributes:
  - The product-oriented quality attributes
    - The quality attributes associated to the description of the product and defined in terms of the model of the solution
  - The process-oriented quality attributes
    - The quality attribute associated to the process facet of the design and defined in terms of the process

Parnas & Weiss’ Requirements

- ***Well structured***: consistent with chosen properties such as information hiding;
- ***Simple*****:** to the extend of being ‘as simple as possible, but no simpler’;
- ***Efficient*****:** providing functions that can be computed using the available resources;
- ***Adequate*****:** meeting the states 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.

Process Facet of Design Quality

The process facet of software design:

- project schedule and milestones
- development team organisation
- test plans, etc.

These are essential parts of a software design.

The process facet of software design has been studied under separate subject titles in computer science, e.g.

- Software process models
- Software project management
- Software risk management
- Configuration management
- Economics of software development

Process-Oriented Quality Attributes (I)

- *Feasibility*: Can the development plan bring about the required product?
- *Simplicity*: How complex is the development process to produce the required product? Is the development process manageable?
- *Reliability*: Can the development plan fail? And how probable will it fail?
- *Productivity*: How productive would it be if the development follows the development plan?
- *Time to market*: What is the time to market if the product is developed following the development plan?
- *Risk*: What are the consequences if the development process fails?

Process-Oriented Quality Attributes (I)

- *Resourcefulness*: does the development process use resources economically?
- *Cost*: what is the cost of the development following the plan?
- *Technical requirements*: does it require intensive training to obtain required technical skills for the people to develop the system? Does it rely on expensive equipment and tools?
- *Material requirements*: does it use expensive materials to produce the product?
- *Legitimacy*: is the development process and the product allowable according to law and conformant to applicable standards and regulations?

Further Readings

[1] Zhu, H., *Software Design Methodology - From Principles to Architectural Styles*, Chapter 1 and 2.

[2] Budgen, D, *Software Design*, Second Edition, Addison-Wesley, 2003. Chapter 4, pp63~85.