COMP 3700: Software Modeling and Design

(Design Quality Evaluation)

IEEE

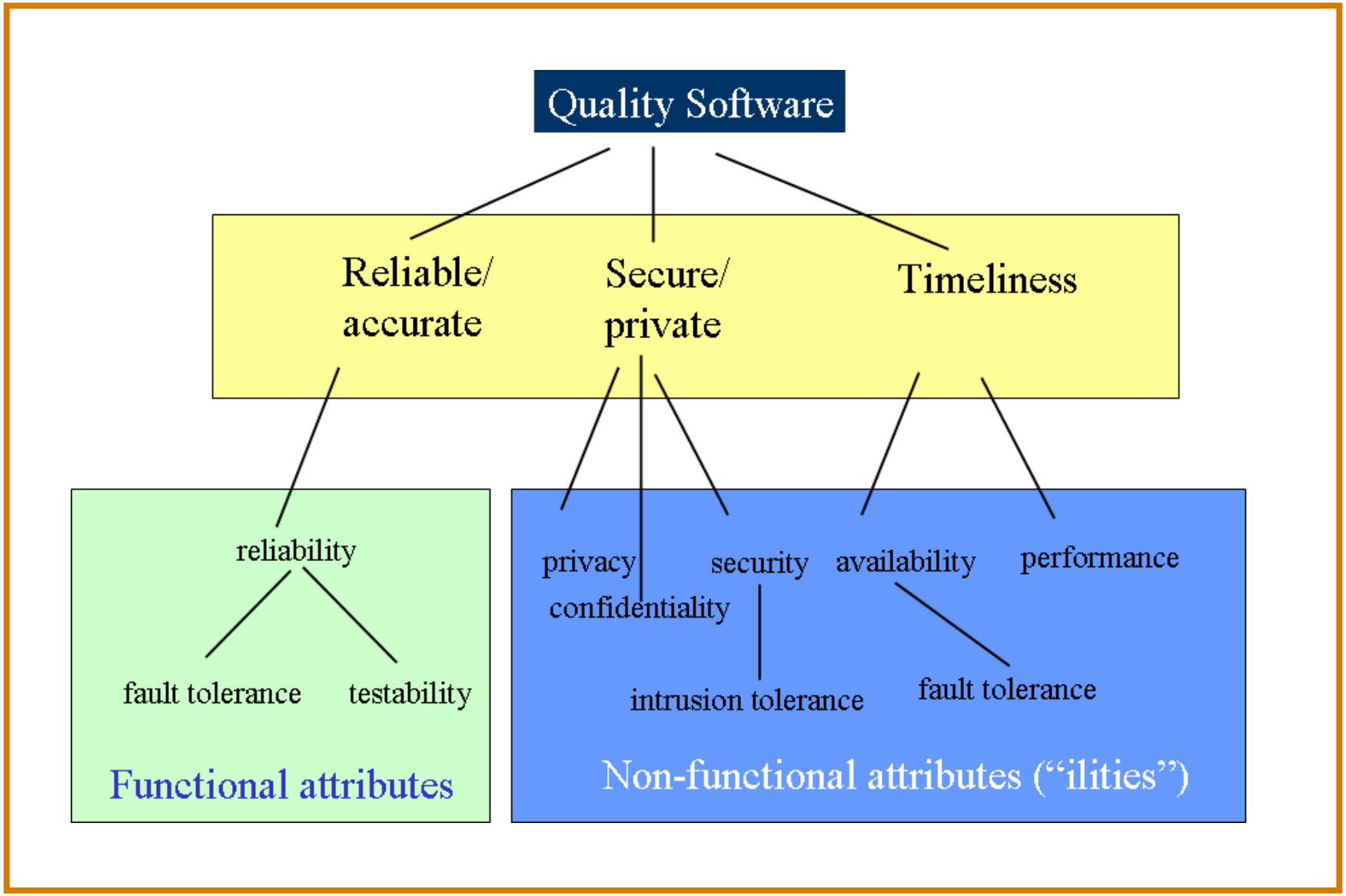
- *"Totality of features of a software product that bears on its ability to satisfy given needs." [Source: IEEE-STD-729]*
- *"Composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer." [Source: IEEE-STD-729]*

Design quality is also 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**


Quality Software
Reliable/
accurate
Secure/
private
Timeliness
reliability
fault tolerance
testability
Functional attributes
privacy
confidentiality
security
availability
performance
intrusion tolerance
fault tolerance
Non-functional attributes ("ilities")

Measuring Quality

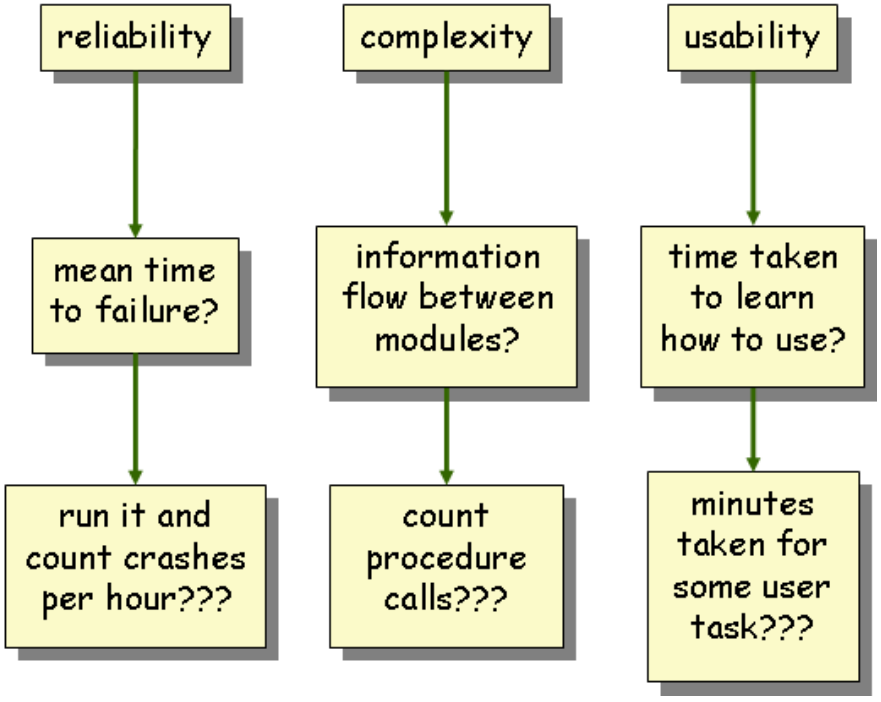
We have to turn our vague ideas about quality into measurables

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

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

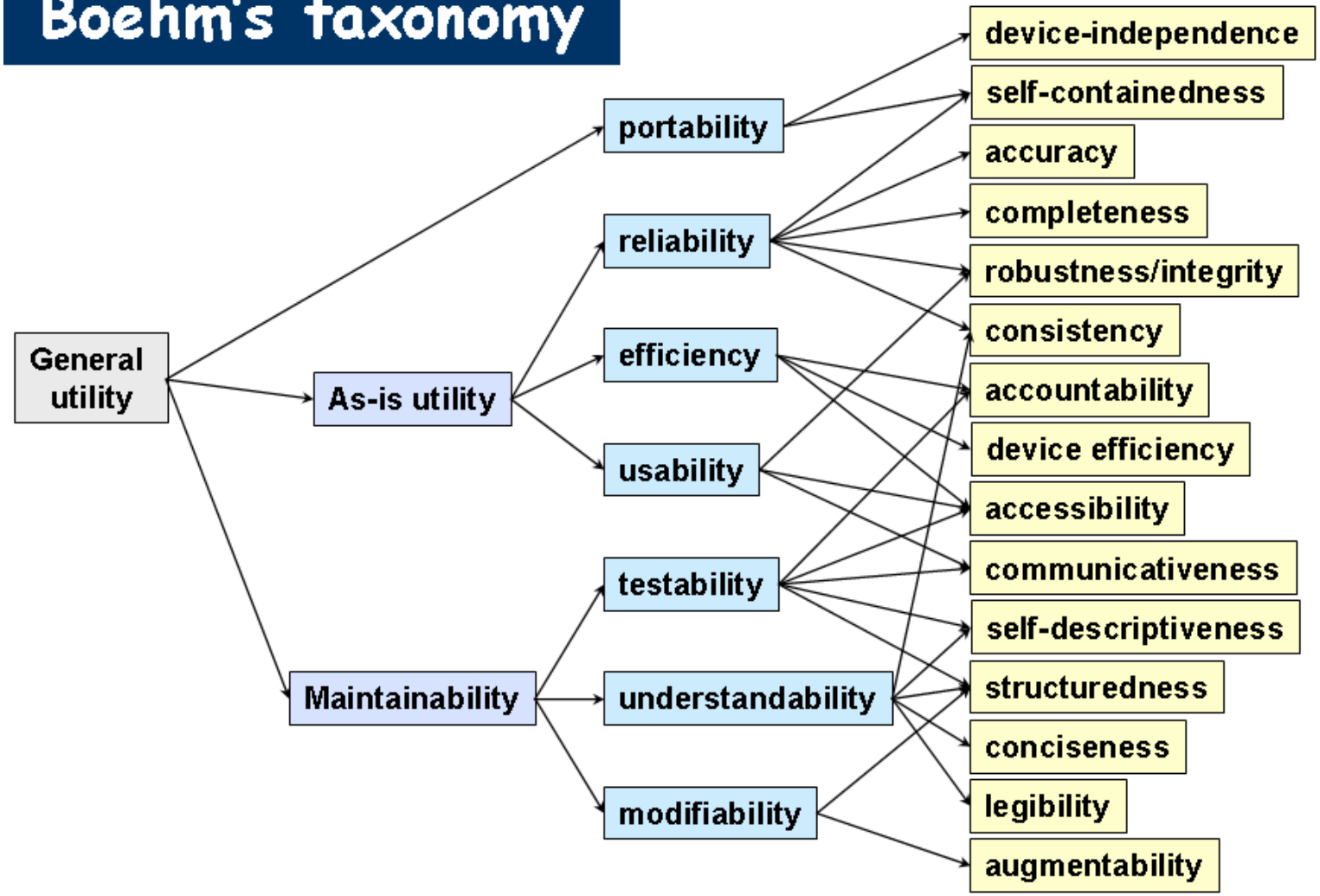
Maintainability

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

Usability

- How easy is it to use?

Boehm's taxonomy



Objectives Principles Attributes Framework

- **A set of** objectives **are defined that correspond to project level goals and objectives (maintainability, testability etc.)**
- **Achieving those objectives require adherence to certain principles that characterize the process by which software is developed.**
- **Adherence to a process governed by these principles should result in a product that possesses attributes considered to be desirable and beneficial.**

Measurable Predictors of Quality

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)

Measuring OO Design Quality

What is Metric: **The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products.**

Methods Per Class (MPC):

Definition: Average number of methods per object class.

Facts:

- The larger the number of methods per object class, the more complicated the testing becomes. This occurs because of the increased object size and complexity.
- Extensibility will be hard if the number of methods per object class becomes too big.
- More methods per object class may be favorable because of inheritance through subclasses will be greater and the amount of code reuse will be greater. But adhoc and random code reuse may be dangerous.

Inheritance Dependencies:

Definition: The depth of inheritance.

Facts:

- Depth is likely to be more favorable than width of an inheritance tree. Deeper trees would show a greater amount of method sharing than wider trees.
- A deep inheritance tree may be harder to test than a broad tree.
- It may be harder to understand with a larger number of layers.

Degree of Coupling Between Objects:

Definition: The average number of uses dependencies per object.

Facts:

- Application maintenance can be more complicated with a greater degree of coupling between objects. This is because object interactions and interconnections are more complex with a higher level of coupling.
- More objects will be suitable for reuse internally and externally to an application if there is a greater amount of uncoupled objects.
- It should be easier to augment uncoupled objects because they will have a lower degree of interaction compared to ones with a high degree of 'uses' dependencies.
- It will be harder to test a system of highly coupled objects.
- Increased amounts of errors can be caused by the complexity of the interactions of coupled objects.

Degree of Cohesion of Objects:

Definition: The degree of dependencies of parts within a single component.

Facts:

- A higher degree of errors are more likely to occur in the development process if there is low cohesion between objects, which raises the complexity of the application and can reduce the application's reliability.
- Objects that have fewer dependencies on other objects for data more likely to be reusable.

Average Method Complexity (AMC):

Definition: The average amount of complexity per method.

Facts:

- Complex methods are more difficult to maintain.
- Complex methods make it harder to understand the application.
- Complex methods are more likely to make an application less reliable.
- It is harder to test complex methods.

Depth of Inheritance Tree (DIT):

Definition: For a certain class, the length of the path from that class to the root of the class hierarchy.

Facts:

- A class will inherit more methods when it is deeper in the hierarchy.
- A class's behavior will be less predictable when it is deeper in the hierarchy.
- Deeper trees represent greater design complexity since they entail more classes and methods.
- When a class is deeper in a hierarchy it has greater possible reuse of inherited methods.

Number of Children (NOC):

Definition: The number of immediate children of a certain class.

Facts:

- There is a greater amount of reuse when there are a larger number of children, because inheritance is a form of reuse.
- Improper abstraction of a parent class is more likely when there are a greater number of children.
- A class's potential influence on the design can be seen through the number of children that parent class has.
- More testing is needed when a class has a large number of children.

Total Number of Methods per Class (TOM):

Definition: The total number of methods in a class, which includes all inherited methods.

Facts:

The larger number of methods, the larger amount of testing needed.

Percentage Public and Protected (PAP):

Definition: The percentage of public member variables in the related class.

Facts:

When the percentage is higher, the related classes need to be tested more.

Size Metrics:

Method Level Size Metrics:

Number of parameters required by a method

Number of operators and operands used in a method

Number of Instance Variables used by the method to perform the functionality

Class Level Size Metrics:

Number of Methods in a class

Number of Member Variables in a class

Size of Class Interface is the number of public member variables available to other classes

Total number of executable statements in all methods

System Level Size Metrics:

Number of Classes in the system

Total Number of Methods and Global Functions in the system

Total Number of Member Variables in all classes

Total Number of Global Variables in all classes