POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

LOG8430E : Design Quality

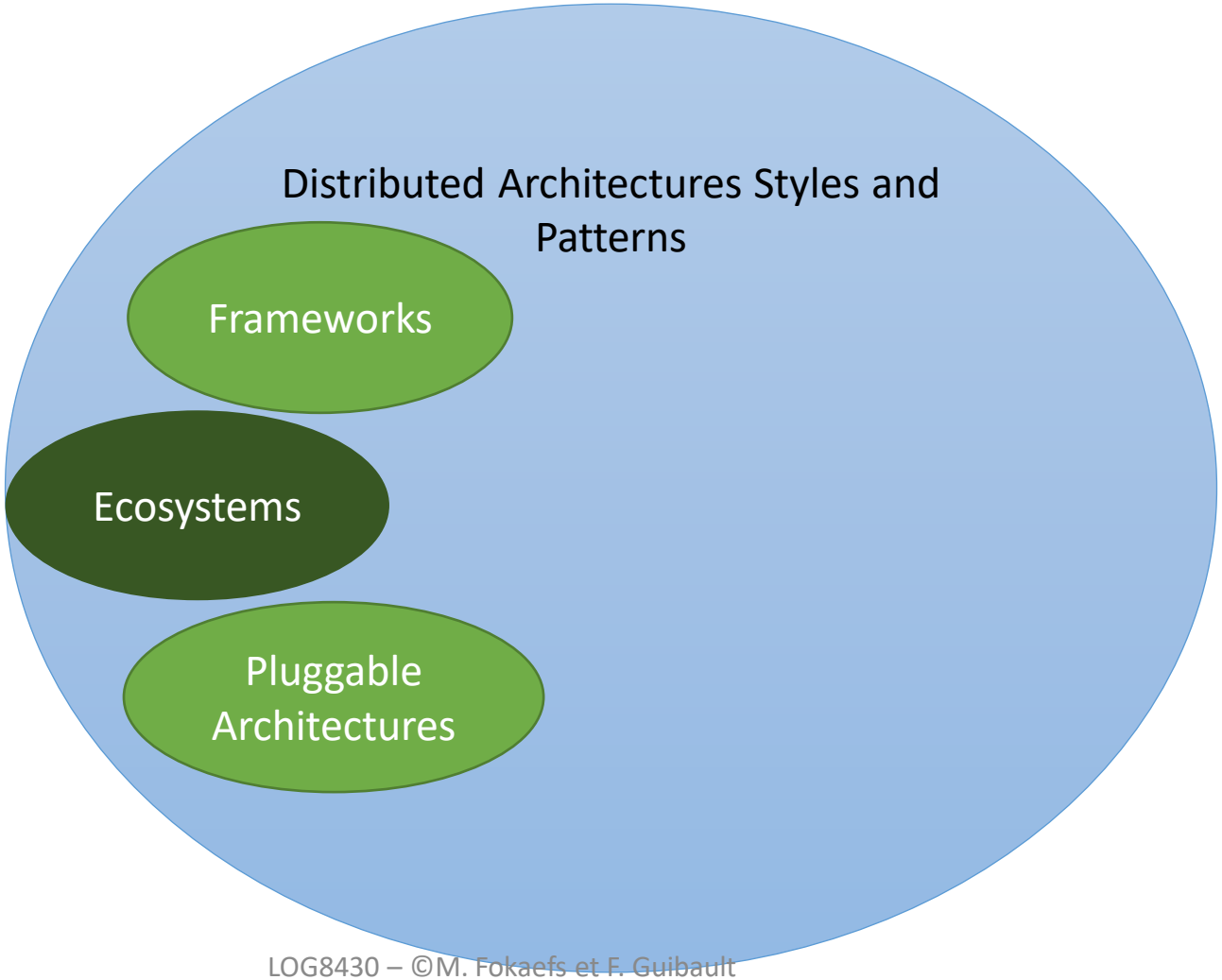
Fall 2020

So far

LOG2410

OO Design (SOLID)

Design Patterns



Today

LOG2410

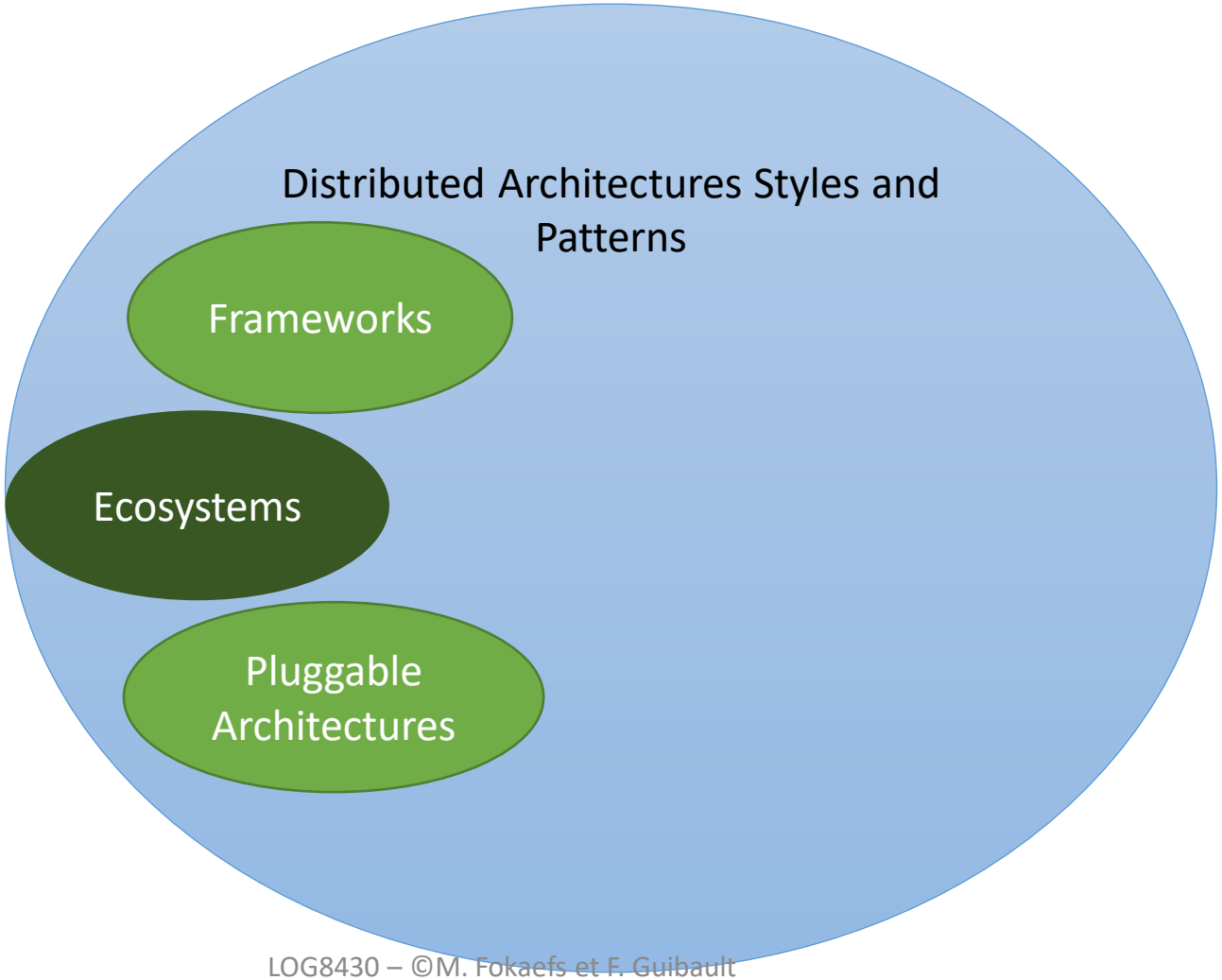
OO Design (SOLID)

Design Patterns

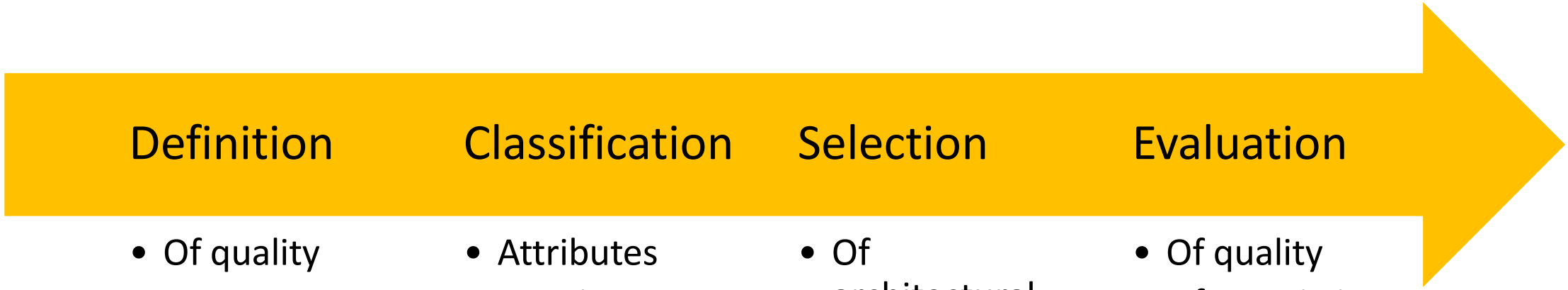
LOG8371

LOG3430

Design Quality



Agenda



- Of quality

- Attributes
- Metrics
- ISO 9126/25010

- Of architectural type
- According to quality criteria

- Of quality
- Of an existing system

After this class…

- You will be able to recognize and prioritize the quality attributes according to the system requirements.
- You will appreciate the importance of high quality and of maintaining this level of quality.
- You will be able to measure the quality to evaluate software and ensure their quality, and to identify cases of bad design (in combination with the next class).
- You will acquire a part of the competences necessary for quality assurance (QA).

What is design quality?

…lots of things!

Definition

- In fact, quality includes many notions and concepts, which are employed together to determine and confirm that the software functions are designed properly and as desired.
- We will concentrate on the internal structural quality.
- **Quality is a fundamental criterion for the selection of architecture and for software design.**

Different perspectives of quality

- Quality for the user: The software is useful and it works as it should in the context of use.
- Value-oriented quality: The software is evaluated based on the value it creates.
- Quality of product: The quality is measured according to the characteristics of the product.
- Quality for the manufacturer: The software is evaluated according to the requirements.

Different perspectives of quality

- Quality for the user: The software is useful and it works as it should in the context of use.
- Value-oriented quality: The software is evaluated based on the value it creates.
- Quality of product: The quality is measured according to the characteristics of the product.
- Quality for the manufacturer: The software is evaluated according to the requirements.

Which one is better?

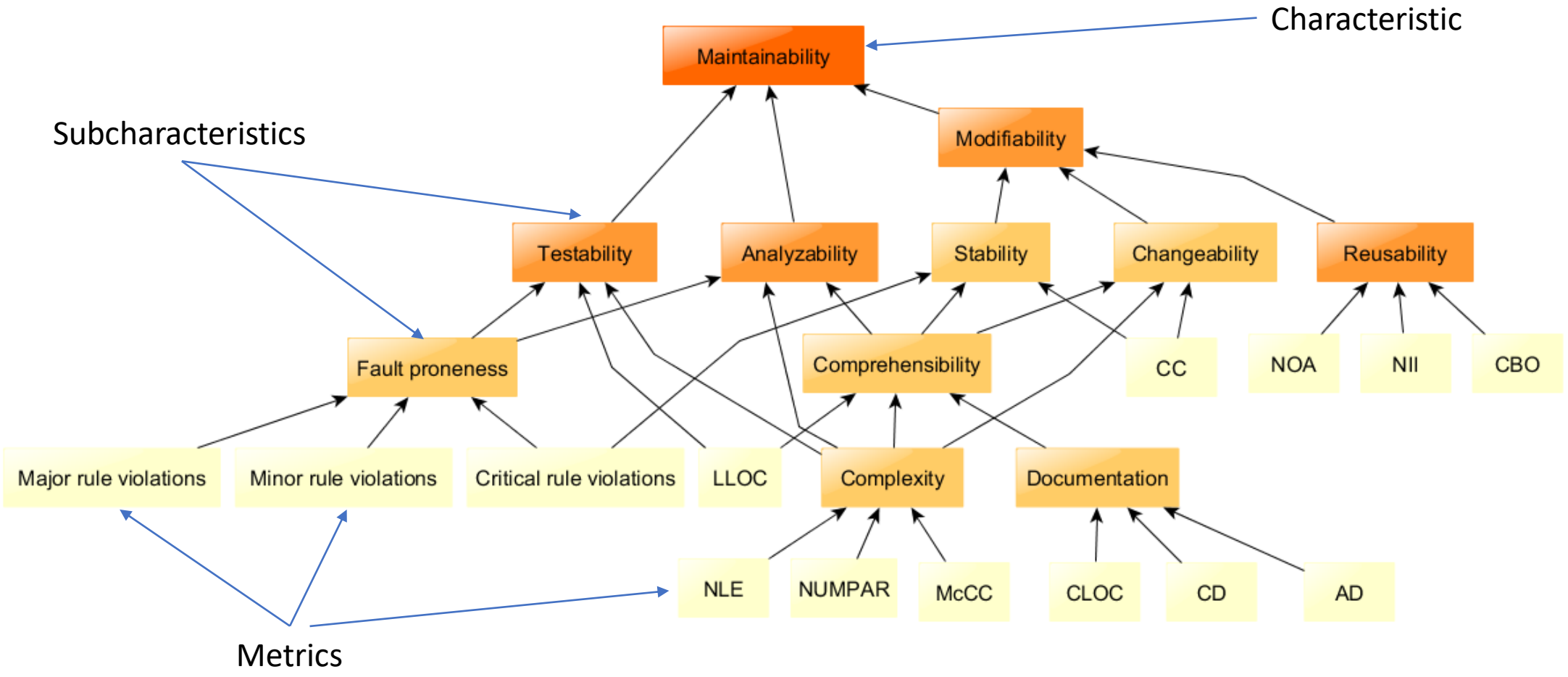
Before comparing/evaluating the quality

- We need:
- **A model**
  - What are the criteria according to which we will evaluate the software?
  - What are our priorities for the design, implementation and use of the software?
- **Metrics**
  - Internal, during development (measured statically).
  - External, during execution (measured dynamically).
  - In-use, during usage (measured by humans).
- **A process**
  - Design, plan, execute the evaluation.
  - Define the framework for the interpretation of the results.

The ISO/IEC 9126 model (the old standard)

Functionality	Reliability	Usability	Efficiency	Maintainability	Portability
• Suitability • Accuracy • Interoperability • Security • Compliance	• Maturity • Fault tolerance • Recoverability	• Understandability • Learnability • Operability • Attractiveness	• Time behaviour • Resource utilisation	• Analyzability • Changeability • Stability • Testability	• Adaptability • Installability • Coexistence • Replaceability

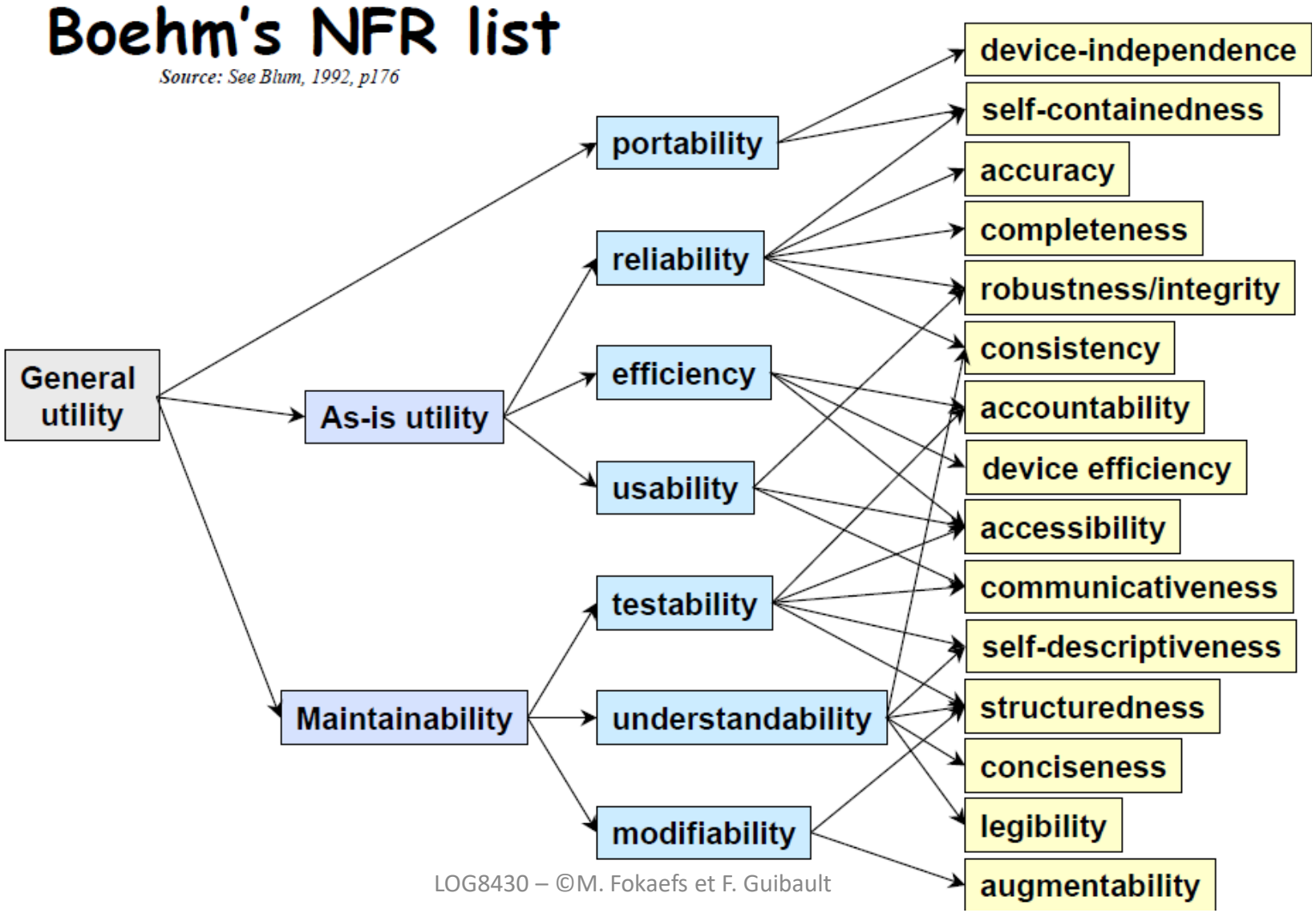
(Semi-)Evolution



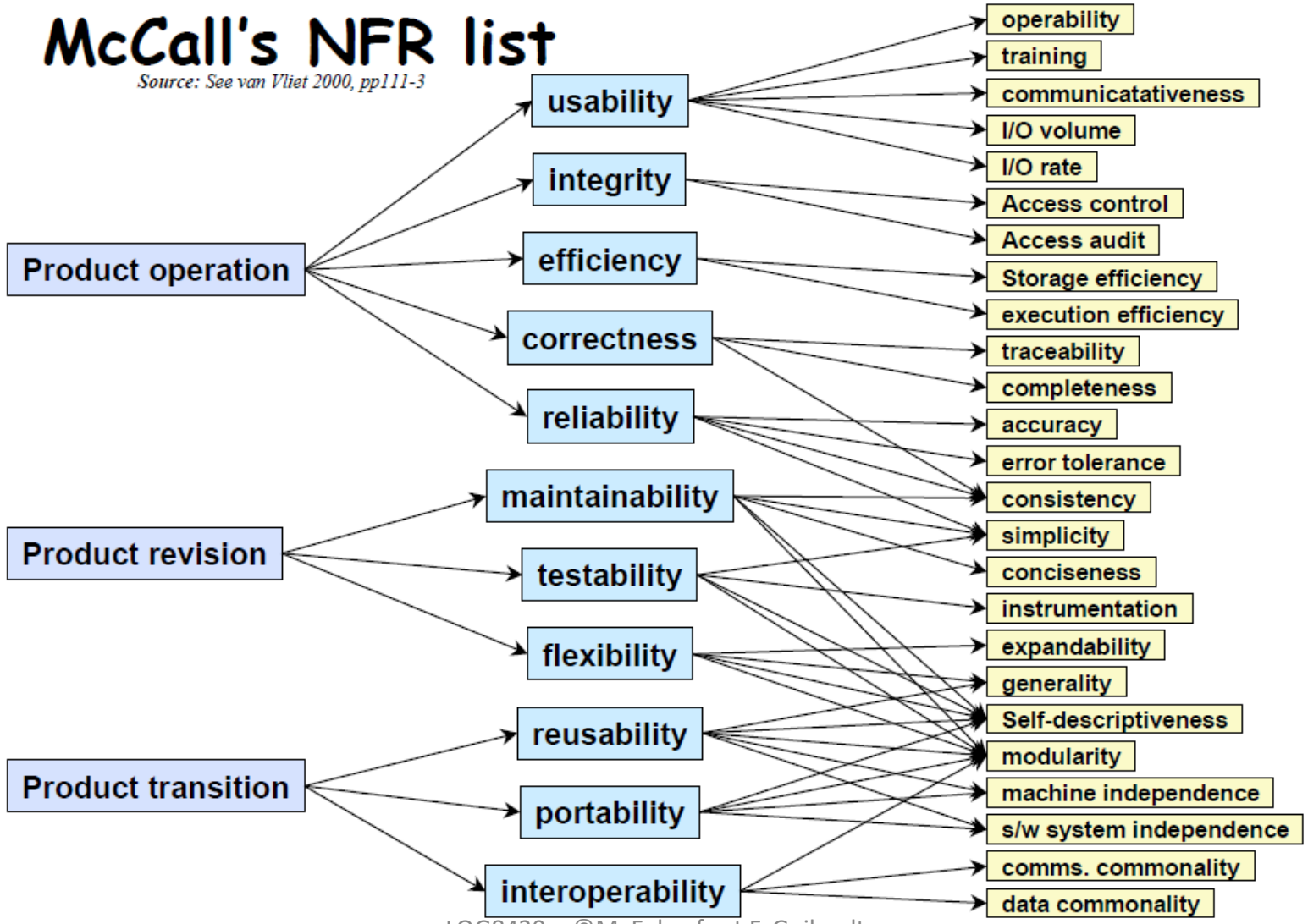
The ISO/IEC 25010:2011 model

Functional Suitability	Performance Efficiency	Compatibility	Usability	Reliability	Security	Maintainability	Portability
• Completeness • Correctness • Appropriateness	• Time Behaviour • Resource Utilization • Capacity	• Coexistence • Interoperability	• Recognizability • Learnability • Operability • User Error Protection • User Interface Aesthetics • Accessibility	• Maturity • Availability • Fault Tolerance • Recoverability	• Confidentiality • Integrity • Non-repudiation • Authenticity • Accountability	• Modularity • Reusability • Analysability • Modifiability • Testability	• Adaptability • Installability • Replaceability

Other models: Boehm



Other models: McCall



Can we measure the criteria?

The Metrics

- Size
  - Lines of code (LOC)
- Complexity of interface
  - Number of attributes and methods ($SIZE2$)
  - Number of local methods (NOM)
- Structural complexity
  - McCabe cyclomatic complexity (CC)
  - Weighted methods count (WMC)
  - Response for class (RFC)
- Inheritance
  - Depth of inheritance tree (DIT)
  - Number of children (NOC)

The Metrics (cont'd)

- Coupling
  - Afferent coupling (Ca)
  - Coupling between objects (CBO)
  - Others…
- Cohesion
  - Lack of cohesion between methods ($LCOM$)
  - Tight class cohesion (TCC)
  - Others…
- Documentation
  - Lack of documentation (LOD)

Size

- LOC: It counts the number of return characters (e.g., \r) in a method, a class, a compilation unit.
  - There are numerous variants: lines of source code (SLOC), lines of code without documentation and white lines.
- SIZE2: It counts the number of attributes and methods of a class.
- NOM: It counts the number of methods declared in a class, not the derived methods.
- Absolute values.
- When LOC, SIZE2 and NOM increase:
  - Understandability, Attractiveness, Analyzability, Modifiability, Testability, Adaptability, Learnability, Stability, Replaceability, Interoperability, Security, Fault tolerance, Recoverability

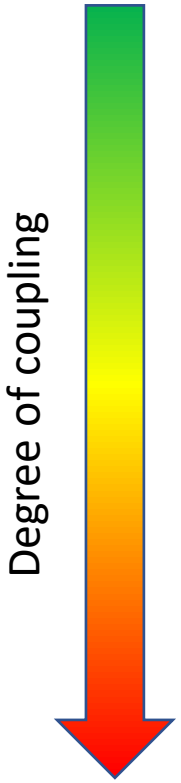
Complexity

- CC: It counts the number of independent paths in a slice of code.
  - Definition1: $CC = E - V + 2*P$, E=edges, V=nodes, P=connected components
  - Definition2: count the number of branching instructions (if, while, for, do, switch), the logical operations (OR, AND) plus 1 (definition of Eclipse Metrics and STAN).
- WMC: It counts the sum of a metric over all methods of a class. The metric could be the complexity, the LOC or none (unweighted)
  - Definition: $WMC = \sum_{i=1}^{n} CC_i$
- RFC: It counts the number of public methods of a class and all methods accessed directly by these public methods.
- Absolute values.
- When CC, WMC and RFC increase:
  - Understandability, Operability, Attractiveness, Analyzability, Modifiability, Testability, Adaptability, Learnability, Stability, Replaceability, Interoperability, Security, Fault Tolerance, Recoverability

Inheritance

- DIT: It calculates the maximal length from a derived class to a basic class in the inheritance structure of the system.
- NOC: It counts the number of classes immediately derived from a basic class.
- Absolute values.
- When DIT increases:
  - Understandability, Operability, Attractiveness, Analyzability, Modifiability, Testability, Adaptability, Learnability, Stability, Replaceability, Interoperability, Security, Fault tolerance, Recoverability
- When NOC increases:
  - Understandability, Operability, Attractiveness, Analyzability, Modifiability, Testability, Adaptability, Learnability, Stability, Replaceability

Coupling (between two modules A and B)



Type	Characteristics	Desirability
Data	A and B use shared data.	High
Data structure	A and B use shared data structures.	OK
Control	A uses methods from B.	Necessary
External	A and B use a common externally imposed “tool”. (data format, device, communication protocol)	Bad
Common	A and B use the same global variables	Bad
Content	A uses the code of B directly or changes the state of B directly.	Very Bad!

Coupling Metrics

- CBO: It counts the number of classes that are “coupled” with another class. Two classes A and B are coupled, if class B uses methods of class A.
- Ca: It counts the number of external classes that are coupled (as per the definition of CBO) with classes from another package with respect to incoming coupling (i.e., external classes use methods of internal classes).
- Absolute values.
- When CBO increases:
  - Understandability, Operability, Attractiveness, Analyzability, Modifiability, Testability, Adaptability, Learnability, Interoperability, Security, Fault tolerance, Recoverability
- When Ca increases:
  - Operability, Attractiveness, Learnability, Stability, Replaceability, Interoperability, Security, Fault tolerance

Cohesion (between members of a module)

Type	Characteristics	Desirability
Data	All are part of a data abstraction.	Very High
Functional	All solve the same problem.	High
Sequential	The output of one is the input of another.	OK
Communication	Operations that use the same input or the same output.	Moderate
Procedural	A series of operations that need to be executed in order.	Bad
Temporal	Elements that function at the same time.	Bad
Logic	Elements that perform logically similar tasks.	Very Bad!
By coincidence	Elements that have no conceptual relation, but they have replicated code.	Very Bad!

Cohesion metrics

- LCOM : More than 6 definitions!
- We will use the one by Handerson-Sellers :

$$LCOM96a = \frac{(\frac{1}{a}\sum_{j=1}^{a}\mu(Aj)) - m}{1 - m}$$

  - α is the number of attributes in a class, m is the number of methods, $\mu(Aj)$ is the number of methods that access attribute Aj.
- But why is it so hard to calculate LCOM?
  - There classes without attributes, or with a single method, or with a single attribute.
  - Inheritance complicates the situation with respect to derived attributes and methods.
  - There are setters and getters.
- TCC = NDP/NP, NDP = number of pairs of methods that access attributes directly, NP = number of pairs of methods (n(n-1)/2)
- Percentages (normalized values) LCOM=0 ➔ maximum cohesion, TTC=1 ➔ maximum cohesion
- Two of the most reliable metrics.
- When LCOM decreases or TCC increases:
  - Maturity, Understandability, Attractiveness, Modifiability, Testability, Adaptability, Learnability, Stability, Replaceability

Documentation

- LOD: Assuming one comment per class and one comment per method, it calculates the percentage of elements that are not commented.
  - LOD= 1- #comm/(m+c+1), #comm = number of comments, m = number of methods, c = number of internal classes.
- Percentage (normalized value)
- When LOD increases:
  - Understandability, Operability, Attractiveness, Maturity, Analyzability, Modifiability, Testability, Adaptability, Learnability, Stability, Replaceability

Matrix of Metrics

Software Quality Matrix

		Main property	Functionality					Reliability				Re-Usability					Efficiency			Maintainability					Portability					
		Sub Property	Suitability	Accuracy	Interoperability	Security	Functionality Compliance	Maturity	Fault-tolerance	Recoverability	Reliability Compliance	Understandability for Reuse	Learnability for Reuse	Operability for Reuse - Programmabilit	Attractiveness for Reuse	Re-Usability Compliance	Time Behavior	Resource Utilization	Efficiency Compliance	Analyzability	Changeability	Stability	Testability	Maintainability Compliance	Adaptability	Installability	Co-existence	Conformance	Replaceability	Portability Compliance
Category	Sub-Category	Metric																												
Complexity	size	LOC																												
	interface C.	SIZE 2																												
		NOM																												
	structural C.	CC																												
		WMC																												
		RFC																												
Architecture & Structure	Inheritance	DIT																												
		NOC																												
	Coupling	Ca																												
		CBO																												
		CDBC																												
		CDOC																												
		Ce																												
		CF																												
		DAC																												
		I																												
		LD																												
		MPC																												
		PDAC																												
	Cohesion	LCOM																												
		ILCOM																												
		TCC																												
Design	Documentatio	LOD																												

Legend

not evaluated	
highly directly,	directly related
highly inversly,	inversly related
not related	

Bonus: Entity Placement

- The entity set of a class consists of all attributes and all methods in the class.
- The entity set of a method consists of all the attributes and all the methods used by this method.
- The entity set of an attribute consists of all the methods that access this attribute.
- The distance between an entity, *e* (method or attribute), and a class, *C*:

$$distance(e, C) = 1 - \frac{|S_e \cap S_C|}{|S_e \cup S_C|}, where\ S_C = \bigcup \{e_i\}.$$

- The degree of Entity Placement in a class:

$$EntityPlacement_C = \frac{\frac{\sum_{e_i \in C} distance(e_i, C)}{|entities \in C|}}{\frac{\sum_{e_j \notin C} distance(e_j, C)}{|entities \notin C|}}$$

Bonus: Entity Placement

- EP is a metric that combines the concepts of cohesion and coupling.
- Violations of this metric are resolved by moving class members or by extracting members in a new class.
- It is used by the JDeodorant tool (for the next class!) to evaluate the impact of refactorings on software quality.

Selecting the type of architeture according to quality criteria

Haoues, M., Sellami, A., Ben-Abdallah, H., & Cheikhi, L. (2017). A guideline for software architecture selection based on ISO 25010 quality related characteristics. International Journal of System Assurance Engineering and Management, 8(2), 886-909.

Comparison of software architectures with respect to quality characteristics and subcharacteristics (1)

ISO 25010 characteristics	ISO 25010 sub-characteristics	SOA	MDA	CBA	EDA	AOA
Functional Suitability	Functional Completeness	0	0	+	0	0
	Functional Correctness	+	0	+	0	+
	Functional appropriateness	0	0	0	0	0
Reliability	Maturity	–	0	0	–	0
	Availability	0	+	+	–	+
	Fault Tolerance	–	+	+	+	+
	Recoverability	0	0	+	0	+
Usability	Appropriateness Recognizability	+	0	0	0	0
	User interface aesthetics	–	+	0	0	0
	Operability	0	+	0	0	0
	User error protection	–	0	0	0	0
	Learnability	–	0	0	0	0
	Accessibility	+	+	0	+	0
Compatibility	Coexistence	+	+	0	0	0
	Interoperability	+	+	+	+	+

- SOA = Service-oriented architecture
- MDA = Model-driven architecture
- CBA = Component-based architecture
- EDA = Event-driven architecture
- AOA = Aspect-oriented architecture

Comparison of software architectures with respect to quality characteristics and subcharacteristics (2)

Performance efficiency	Time Behaviour	−	0	+	+	0
	Resource Utilization	+	+	0	+	+
	Capacity	−	−	−	+	+
Portability	Adaptability	+	+	+	+	+
	Instability	−	−	−	0	+
	Replaceability	+	+	+	0	0
Security	Confidentiality	−	−	0	−	0
	Integrity	−	+	+	−	0
	Non-repudiation	0	−	0	0	0
	Accountability	0	0	0	0	0
	Authenticity	−	−	0	0	0
Maintainability	Modularity	+	+	+	+	+
	Reusability	+	+	+	+	−
	Analyzability	+	0	+	0	0
	Modifiability	+	+	+	−	+
	Testability	−	0	−	0	0

- SOA = Service-oriented architecture
- MDA = Model-driven architecture
- CBA = Component-based architecture
- EDA = Event-driven architecture
- AOA = Aspect-oriented architecture

Relationships between quality characteristics

	Functional suitability	Performance efficiency	Usability	Compatibility	Reliability	Security	Maintainability	Portability
Functional suitability	+	−	+	0	+	−	+	0
Performance efficiency	0	+	−	−	0	−	−	−
Usability	+	±	+	0	+	0	±	0
Compatibility	0	0	0	+	0	−	±	+
Reliability	+	0	+	0	+	0	+	0
Security	0	−	−	−	+	+	0	0
Maintainability	+	−	0	+	±	±	+	+
Portability	0	−	0	+	0	0	+	+

Architecture selection guide

	Functional suitability	Performance efficiency	Usability	Compatibility
1st choice	CBA	EDA	MDA	SOA|MDA
2nd choice	SOA|AOA	AOA	EDA	CBA|EDA|AOA
3rd choice	MDA|EDA	MDA|CBA	CBA|AOA	
	Reliability	Security	Maintainability	Portability
1st choice	CBA|AOA	CBA	MDA	AOA
2nd choice	MDA	AOA	SOA|CBA	EDA
3rd choice	EDA	EDA	EDA|AOA	SOA|MDA|CBA

Next time

LOG2410

OO Design (SOLID)

Design Patterns

LOG8371

Bad Design

LOG3430

Design Quality