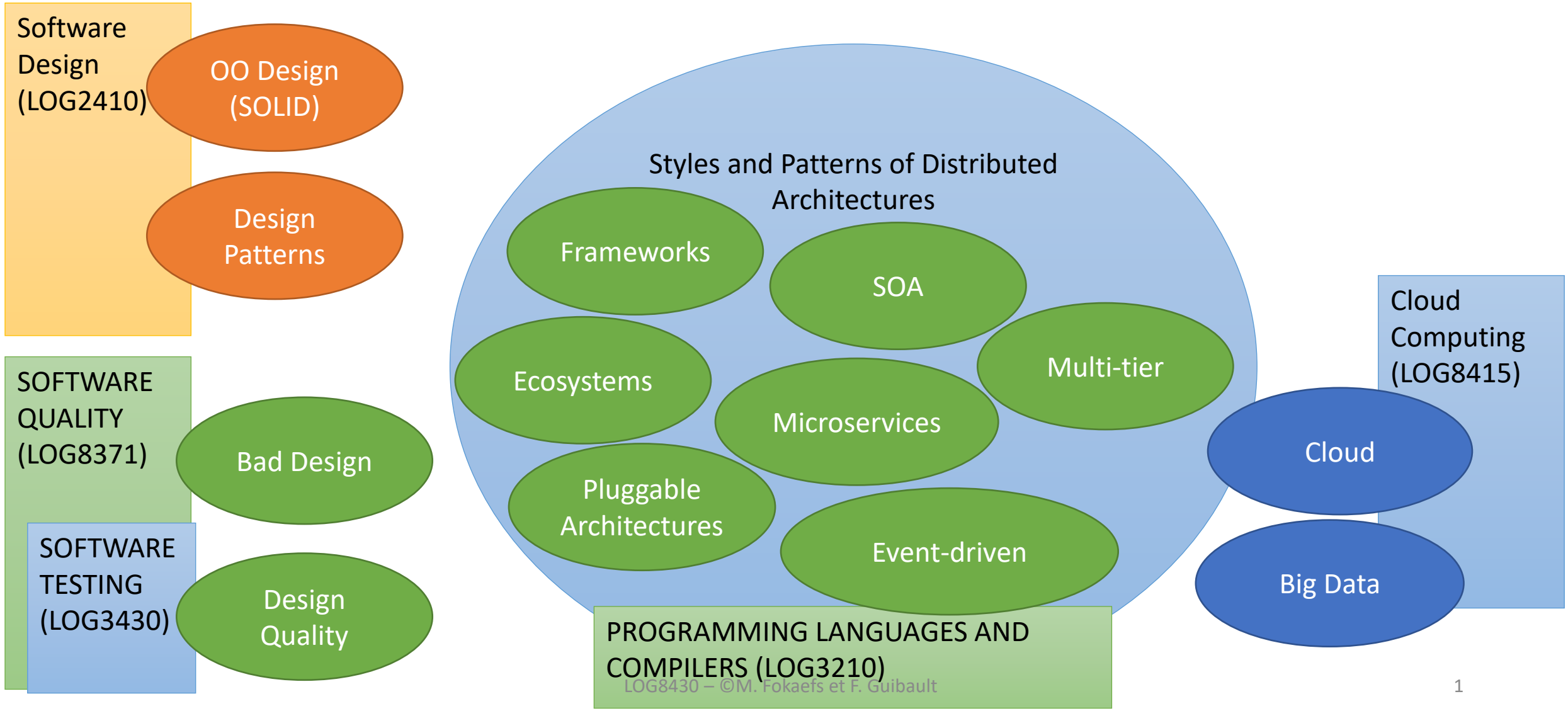
Course Map


Software Design (LOG2410)
OO Design (SOLID)
Design Patterns
SOFTWARE QUALITY (LOG8371)
Bad Design
SOFTWARE TESTING (LOG3430)
Design Quality
Styles and Patterns of Distributed Architectures
Frameworks
SOA
Ecosystems
Multi-tier
Microservices
Pluggable Architectures
Event-driven
PROGRAMMING LANGUAGES AND COMPILERS (LOG3210)
Cloud Computing (LOG8415)
Cloud
Big Data

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

LOG8430E: Bad Design

Fall 2020

Today

LOG2410

OO Design (SOLID)

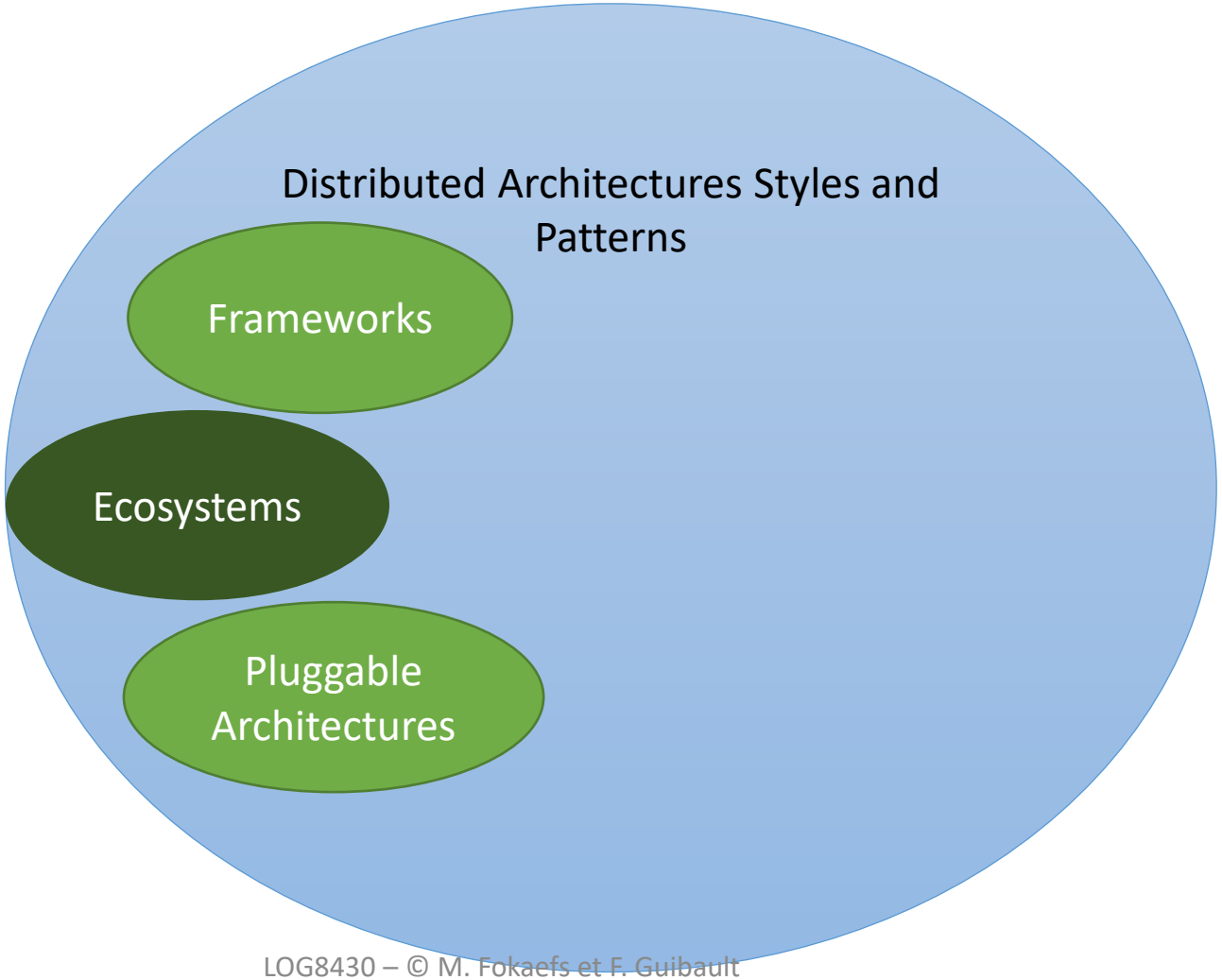
Design Patterns

LOG8371

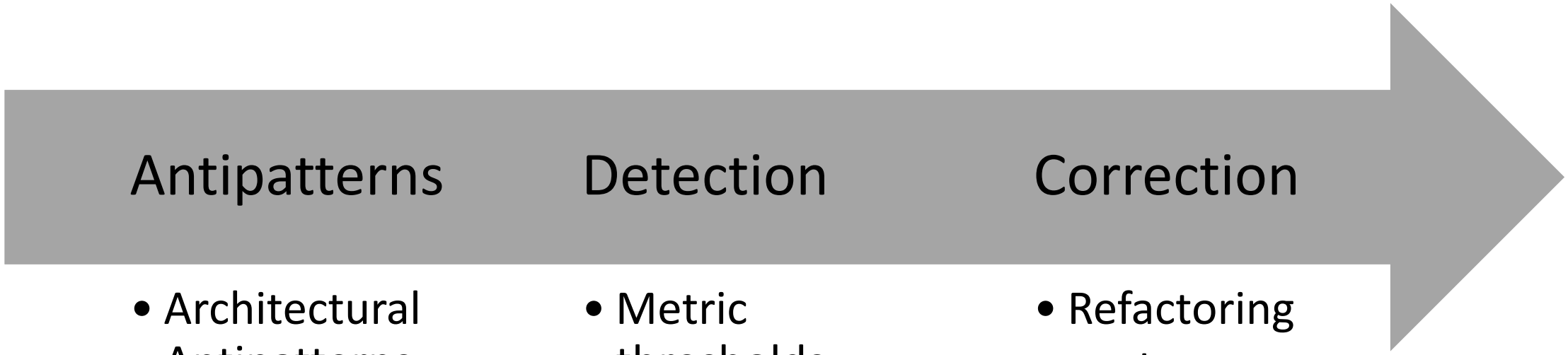
Bad Design

LOG3430

Design Quality



Agenda



- Architectural Antipatterns
- Design Antipatterns
- Code Antipatterns

- Metric thresholds
- Detection Strategies

- Refactoring
- Tools (JDeodorant, Eclipse

What is an antipattern?

- A pattern is a solution to a recurring problem and which facilitates the activities that are peripheral to the development, like maintenance, quality assurance and others.
  - It is a good practice!
- An antipattern is a solution to a recurring problem, but which is often insufficient and counterproductive.
  - It’s a bad practice!
- But why do antipatterns exist?
  - Inability or insufficient ability
  - Good short-term solutions but with harmful long-term consequences .
  - Use of good patterns in an inappropriate context.

Architectural antipatterns

- Architecture by implication
- Autogenerated Stovepipe
- Cover your Assets
- Design by Committee
- Jumble
- Reinvent the Wheel
- Spaghetti Code
- Stovepipe System
- Swiss Army Knife
- Vendor Lock-in

Architecture by Implication

- Definition
  - The architects have a lot of experience in developing similar system and they do not prepare sufficient architectural specifications.
- Symptoms and Consequences
  - A lack of planning and specification of architecture.
  - Ignorance concerning new technologies
  - Hidden risks that emerge during the project: lack of domain knowledge, technologies and unexpected complexity.
  - Imminent failure
- Causes
  - Absence of risk management
  - Excessive confidence from managers, architects and developers.
  - Dependence to previous experiences that differ under certain critical aspects.
  - Non-resolved architectural issues or implicit causes from deficiencies in engineering the system.

Architecture by Implication

- Refactoring
  - Modularize the architecture specification in views and in points of view.
  - Each view represents a stakeholder (architect, developer, client, etc.)
  - Express the architecture in terms of goals and questions (GQA): goals, questions, views, analysis of views, integration of views, matching between views and requirements, refinement of views, communication of the architecture, validation of the implementation.
- Exceptions
  - If the differences between two projects are small, few and local.
  - For an exploratory project.

Autogenerated Stovepipe

- Definition
  - The problem occurs when we migrate a system to a distributed version.
  - The interfaces of the original system may be inappropriate for the distributed system.
  - The local operations often make various assumptions over the placement, e.g., the access to the local file system.
- Refactoring
  - Define the distributed interfaces from scratch (using the façade pattern?).
  - The interoperability and the stability must be of the highest priority.

Cover your assets

- Definition
  - The documentation is very detailed to cover all options and all exceptions, which makes it very complicated.
- Refactoring
  - Use abstractions, like diagrams and tables, to communicate the architecture in a more efficient manner.

Design by committee

- Definition
  - All the team members can influence the decisions for the architecture and the design. The result is a very complex documentation with all the possible characteristics.
- Symptoms and consequences
  - Frequent meetings, but not organized and without objective.
  - A lot of conflicts among architects and developers.
  - A lot of effort to interpret and develop the specifications.
- Causes
  - The roles are not well-defined.
  - An effort to satisfy everyone.
  - Long but not organized meetings.

Design by committee

- Refactoring
  - Define clear roles for the team members.
  - Designate a principal architect for the project, who will make the final decisions.
  - Prioritize the requirements.
  - Organize the meetings and reduce their duration.
  - Possibly divide the teams to better organize the meetings (DevOps).
- Exceptions
  - When the teams are small.

Jumble

- Definition
  - When the horizontal elements merge with the vertical elements. The horizontal elements are the layers of a general architecture. The vertical elements are the architecture tiers specific to an application. (Like a DIP violation).
- Symptoms and consequences
  - Reduced reusability and stability.
- Causes
  - Lack of communication between the developers and the architects.
- Refactoring
  - Treat the horizontal elements as abstractions and the vertical elements as implementations.
  - Add vertical elements for special functionalities or for performance.

Reinvent the Wheel

- Definition
  - When a system is designed from scratch. The existing examples and architectures are not considered.
- Symptoms and consequences
  - The reusability and interoperability are reduced. The effort is increased.
  - Closed architectures, relevant only to one specific system (like a violation of OCP).
  - Useless replication of existing designs.
- Causes
  - “Greenfield Assumptions”: the system needs to be designed from scratch. We are the first to do something like this.
  - Insufficient knowledge of other projects.

Reinvent the Wheel

- Refactoring
  - Mine architectures to identify relevant examples.
  - Put in place a spiral, agile or iterative software process.
  - Refine the requirements and the design decisions, and identify the architecture prototypes in other systems.
- Exceptions
  - When the communication is difficult and we want to minimize the coordination costs.

Spaghetti Code

- Definition
  - The software lacks structure and coherence.
  - “The problem with visual programming is that spaghetti code really looks like spaghetti!”
- Symptoms and consequences
  - The understandability and the reusability are reduced.
  - The system is more procedural than object-oriented.
  - Maintenance effort is increased.
- Causes
  - Lack of experience.
- Refactoring
  - Refactoring
  - Use of patterns and good design practices.
- Exceptions
  - Spaghetti code can be tolerated when it exists in the implementation but not in the public interfaces.

Stovepipe System

- Definition
  - The antipattern occurs when we want to integrate subsystems. Each integration is special and there isn't a general integration plan for the entire system.
- Symptoms and consequences
  - Maintainability and interoperability are reduced.
  - The implementation does not correspond to the documentation.
  - The system becomes too complicated.
  - Portability is reduced.
- Causes
  - Lack of abstraction. The interface of each subsystem is unique.
  - High coupling.
  - Lack of a central architecture.
- Refactoring
  - Define a common interface (SOA).

Swiss Army Knife

- Definition
  - A very complicated interface that exposes many functionalities (a violation of SRP and of ISP).
- Refactoring
  - Refactoring! (Extract an interface or a class).
  - Use the adapter or façade pattern.
  - Document the use and the implementation of an interface. Create a profile.

Vendor Lock-in

- Definition
  - When the system depends directly on proprietary software or hardware.
- Symptoms and consequences
  - The upgrades of commercial products drive the maintenance cycle of applications.
  - Failures in the applications occur due to delays induced by commercial products.
  - Violations of DIP and of OCP.
  - Portability is reduced.
- Causes
  - There is no universal standard.
  - The commercial product does not conform to open standards.
  - Choice of products for economic but not technical reasons.
- Refactoring
  - Multitier architecture
  - Use of abstractions

Design antipatterns

- Missing abstraction
- Multifaceted abstraction
- Duplicate abstraction
- Deficient encapsulation
- Unexploited encapsulation
- Broken modularization
- Insufficient modularization
- Cyclically-dependent modularization
- Unfactored hierarchy
- Broken hierarchy
- Cyclic hierarchy

Missing abstraction

- Definition
  - When groups of cohesive data often exist together in the context of a larger module.
  - E.g., `telephoneNumber, areaCode, countryCode`
- Symptoms and consequences
  - Violation of SRP
  - Increased complexity
  - Reduced maintainability and understandability.
- Refactoring
  - Extract a class.

Multifaceted abstraction

- Definition
  - An abstraction has more than one responsibility
- Symptoms and consequences
  - Violation of SRP.
- Refactoring
  - Extract a class or a superclass.

Duplicate abstraction

- Definition
  - Two abstractions have the same name or the same implementation (cloning).
  - Note: The same name is allowed between two classes as long as they belong in different packages.
- Causes
  - Copy-paste
  - Lack of communication
- Refactoring
  - Extract class or superclass

Deficient encapsulation

- Definition
  - When an abstraction gives more permissions than necessary, e.g., when all the attributes are declared as public.
- Symptoms and consequences
  - Maintainability and security are reduced.
- Causes
  - Procedural thinking in an object-oriented context.
  - Negligent maintenance.
  - To facilitate tests.
- Refactoring
  - Encapsulate data and provide access methods.

Unexploited encapsulation

- Definition
  - The use of explicit type verification when polymorphism is available.
- Symptoms and consequences
  - Increased complexity.
  - Reduced maintainability and understandability.
- Causes
  - Procedural thinking in an object-oriented context.
  - Failure of applying object-oriented principles.
- Refactoring
  - Replace type checking with polymorphism.

Broken modularization

- Definition
  - Data or methods that should be together, with respect to semantic similarity or usage, belong to many abstractions.
- Symptoms and consequences
  - High coupling, low cohesion.
  - Reduced maintainability.
  - The performance deteriorates.
- Causes
  - Procedural thinking in an object-oriented context.
  - Lack of knowledge of the existing design.
- Refactoring
  - Move methods or attributes
  - Inline class

Insufficient modularization

- Definition
  - An abstraction that has many public attributes and too complex methods.
- Symptoms and consequences
  - Violation of SRP and of ISP.
- Causes
  - Centralized control.
- Refactoring
  - Extract class or interface

Cyclically-dependent modularization

- Definition
  - Two abstractions use many members of each other.
- Symptoms and consequences
  - High coupling.
- Causes
  - Passing “this” pointer.
  - The abstraction is not designed correctly.
- Refactoring
  - Move methods or attributes.
  - Inline class.

Unfactored hierarchy

- Definition
  - In a hierarchy, there is duplication between the derived classes or between the derived classes and the base class.
- Symptoms and consequences
  - Maintainability is reduced.
- Causes
  - Code duplication (cloning)
  - Copy-paste
- Refactoring
  - Extract superclass
  - Pull up methods or attributes

Broken hierarchy

- Definition
  - The base class and the derived class do not have a “is-a” relationship.
- Symptoms and consequences
  - Violation of LSP.
- Causes
  - The inheritance is introduced for implementation purposes and not for design.
- Refactoring
  - Replace inheritance with delegation.
  - Invert the inheritance relationship.

Cyclic hierarchy

- Definition
  - A base class has an association with one or more of the derived classes.
- Symptoms and consequences
  - High coupling.
  - Reduced understandability
- Causes
  - Inheritance is not correctly designed.
- Refactoring
  - Extract class.
  - Move methods
  - Inline class
  - Implement a State or a Strategy pattern.

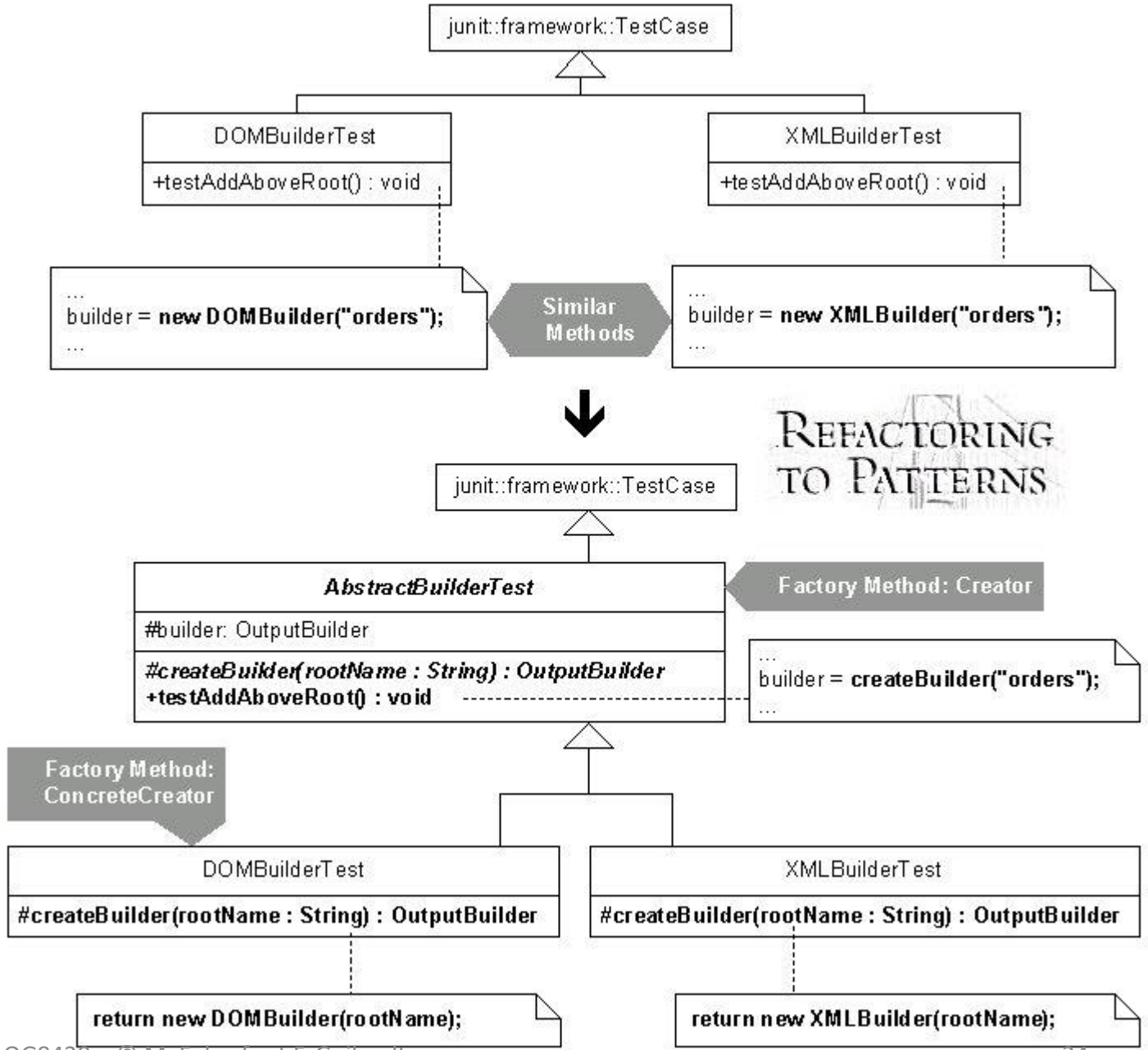
Code antipatterns

- Duplicated Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Switch Statements
- Message Chains
- Inappropriate Intimacy
- Refused Bequest

Duplicated Code

- Definition
  - The same code exists in multiple places in the system.
- Refactoring
  - Extract method
  - Construct a generic class
  - Extract class
  - Chain constructors
  - Introduce a polymorphic object creation method using the Factory Method design pattern.

Introduce Polymorphic Creation with Factory Method



Long Method

- Definition
  - A method has a long implementation that does a lot of things
- Refactoring
  - Extract method
  - Decompose conditional statements
  - Compose method

Compose Method

```
public void add(Object element) {
  if (readOnly) {
    int newSize = size + 1;
    if (newSize > elements.length) {
      Object[] newElements =
        new Object[elements.length + 10];
      for (int i = 0; i < size; i++)
        newElements[i] = elements[i];
      elements = newElements;
    }
    elements[size++] = element;
  }
}
```

```
public void add(Object element) {
  if (readOnly)
    return;
  if (atCapacity())
    grow();
  addElement(element);
}
```

Large Class

- Definition
  - A class that has a lot of members or a lot of responsibilities
- Refactoring
  - Extract class
  - Extract superclass
  - Extract subclass

Long Parameter List

- Definition
  - A method has a long list of parameters.
- Refactoring
  - Replace parameters with method
  - Introduce parameter object

Divergent Change

- Definition
  - A class changes a lot, many times, or every time there is a change in the requirements.
- Refactoring
  - Extract class

Shotgun Surgery

- Definition
  - When there is a change, we need to change the code in many places.
- Refactoring
  - Move methods
  - Inline class

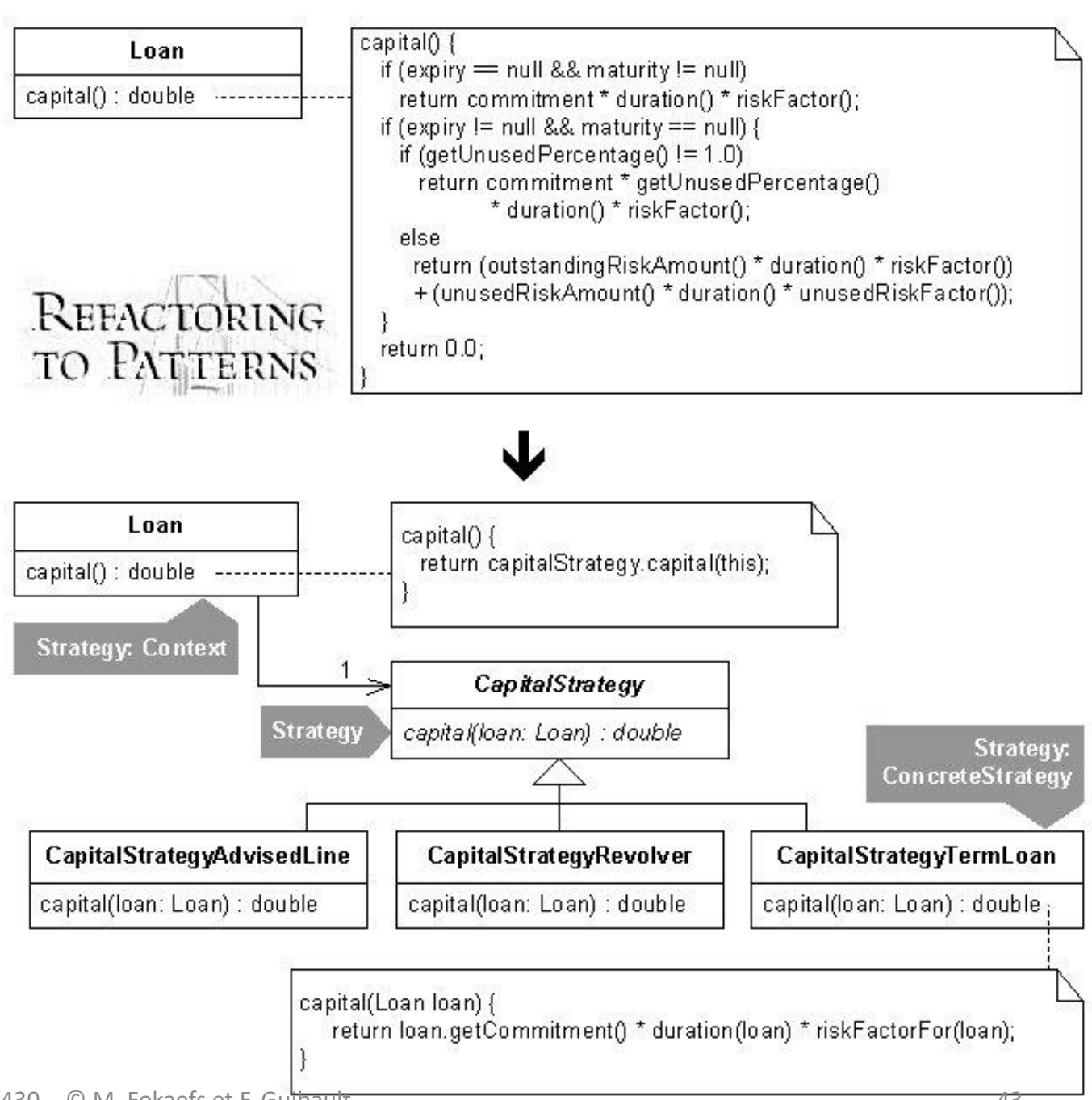
Feature Envy

- Definition
  - A method uses more members from other classes than from the class it belongs to.
- Refactoring
  - Move method
  - Extract or move methods or attributes.

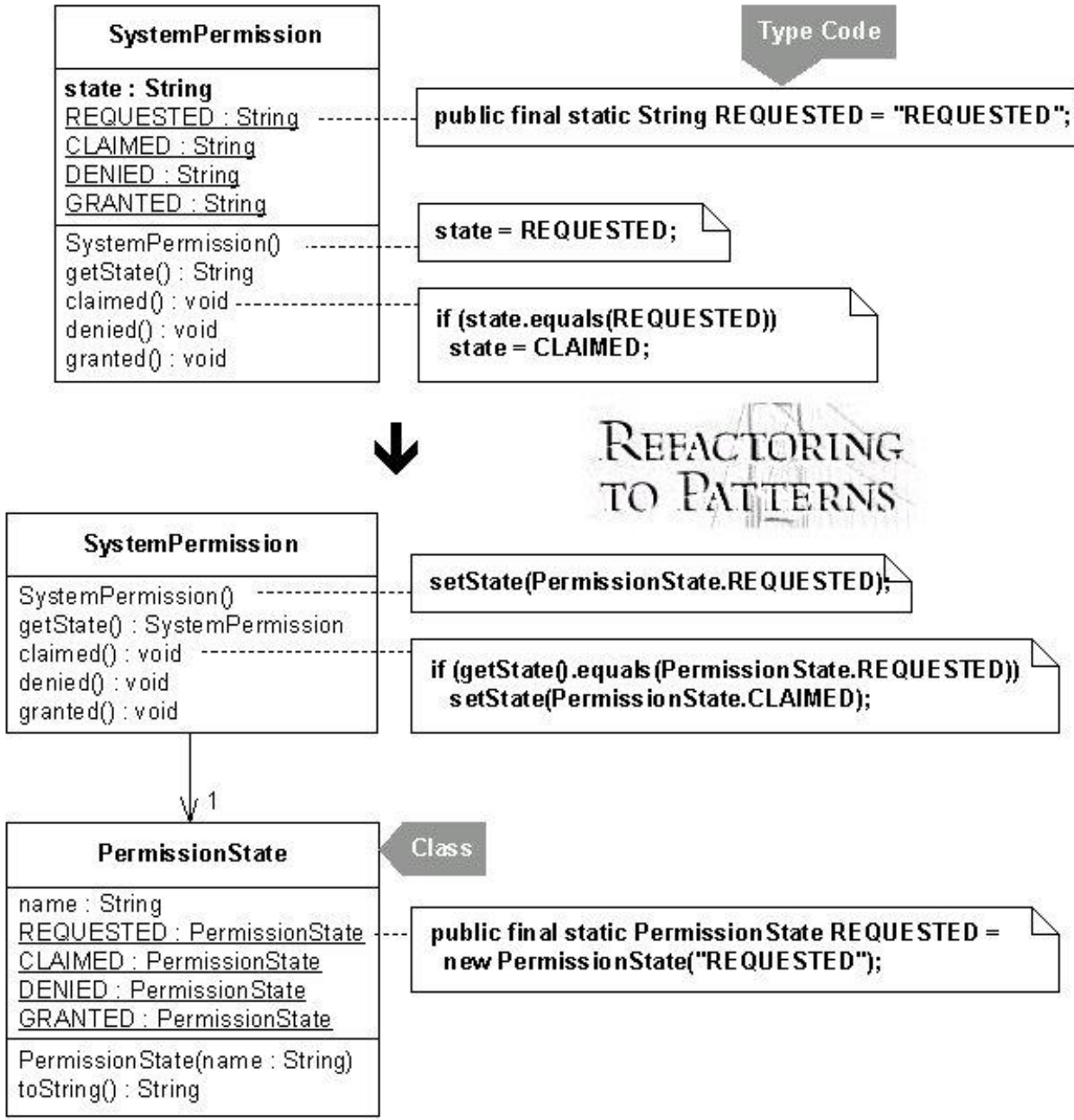
Switch statements

- Definition
  - A method is long and complex due to containing many “switch” or conditional statements.
- Refactoring
  - Replace type checking with subclasses.
  - Replace type checking with State/Strategy design pattern.
  - Replace conditional statements with polymorphism.

Replace Conditional Logic with Strategy



Replace Type Code with Class



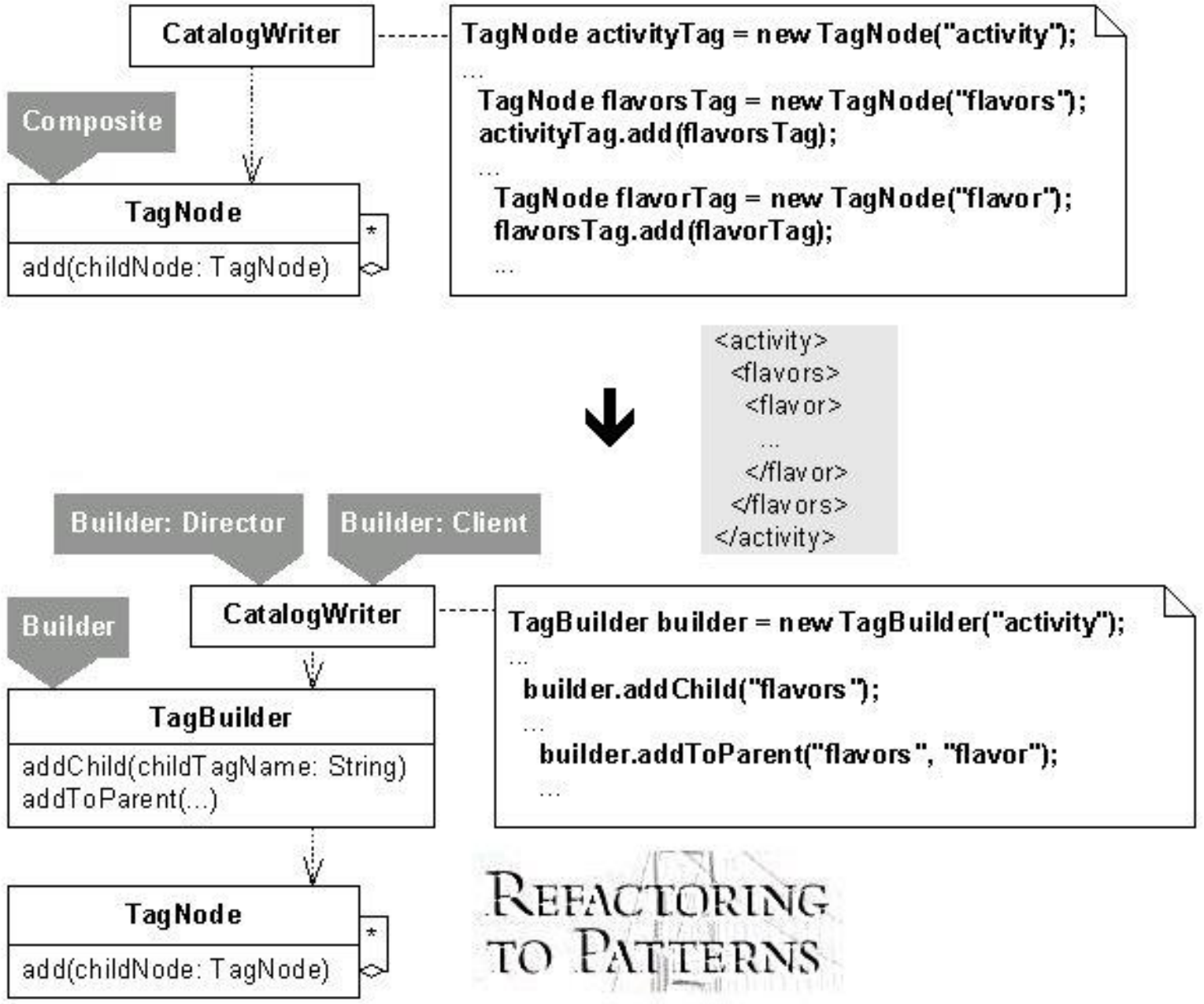
Message Chains

- Definition
  - There are long cascading method invocations.
- Refactoring
  - Extract and move method.

Inappropriate Intimacy

- Definition
  - A class has many associations with private members of another class.
- Refactoring
  - Move method/attribute
  - Extract class
  - Hide delegation
  - Encapsulate Composite with the Builder pattern.

Encapsulate Composite with Builder



Refused Bequest

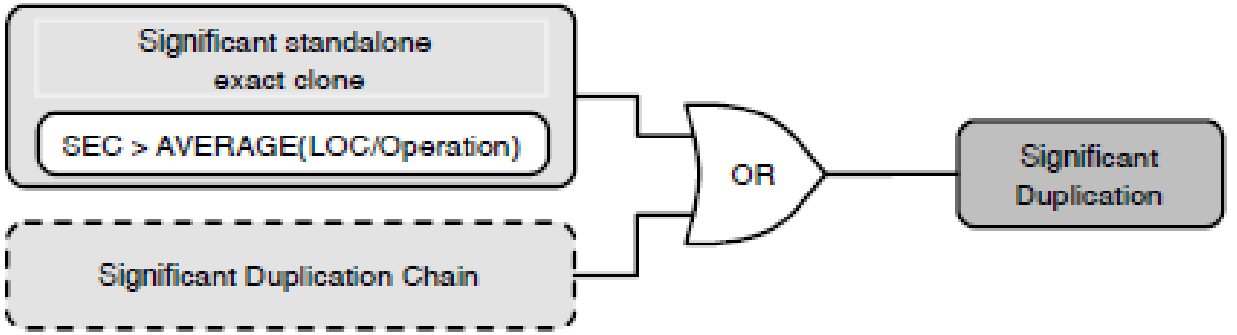
- Definition
  - A derived class does not need to inherit certain members of the base class.
- Refactoring
  - Push down method/attribute to the subclasses.

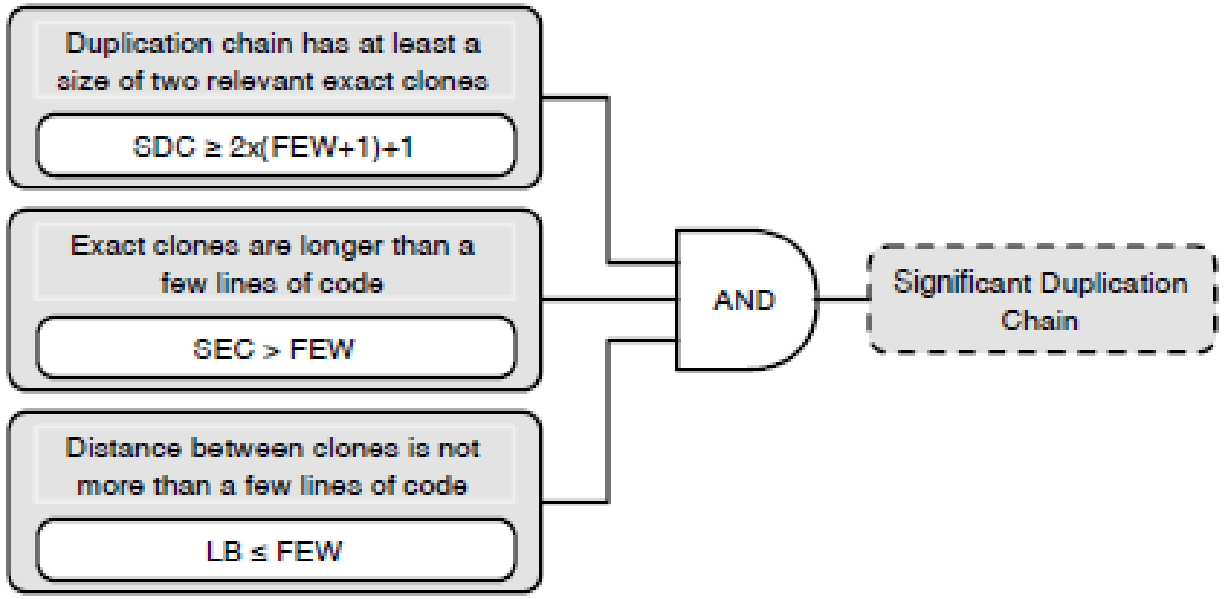
Detection of antipatterns

- It is possible to detect certain code antipatterns based on metrics.
- We can define thresholds on the metrics, combine the relevant conditions and detect the antipatterns.
- Definition of thresholds
  - Statistical: LOW = AVG - ST.DEV., HIGH = AVG + ST.DEV., VERY_HIGH = (AVG + ST.DEV.)*1.5
  - Semantic: a quarter, a third, half, two thirds, three quarters
  - General: 0 = NONE, 1 = ONE, 2-5 = FEW, 7 = Short Memory Cap, >7 = MANY

Duplicated Code

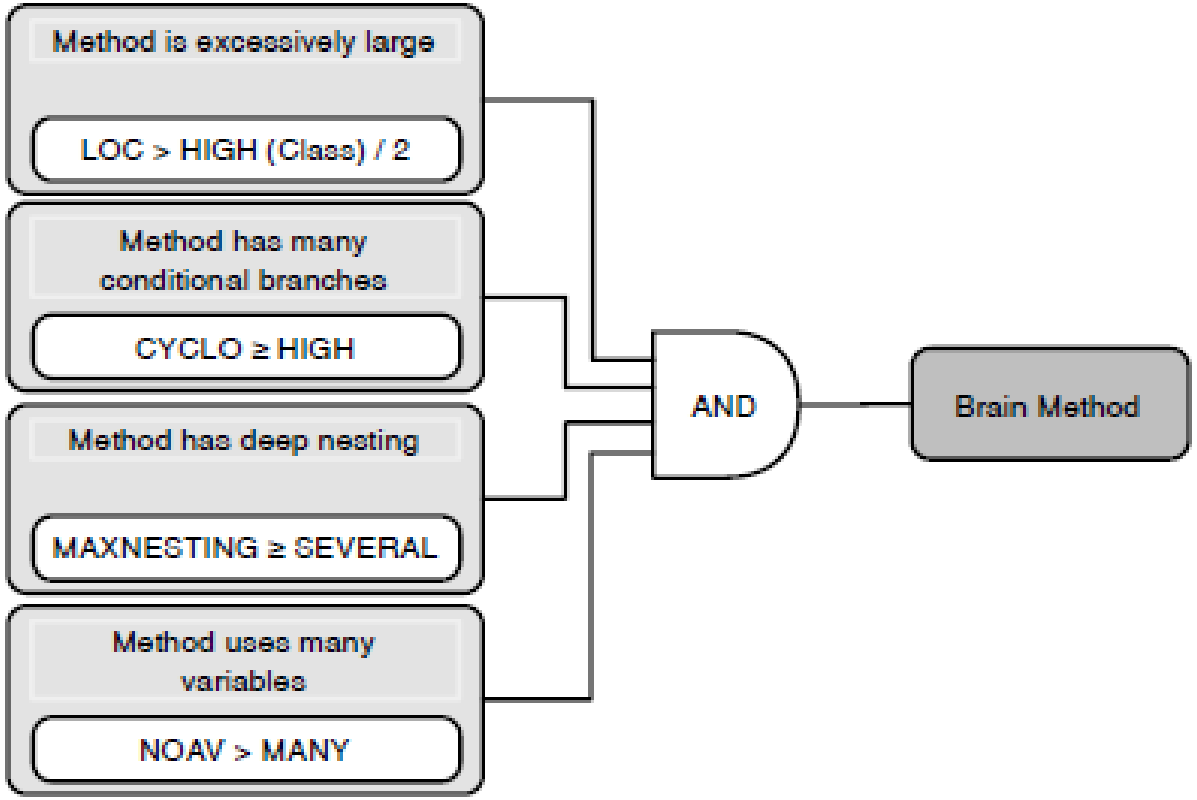
- SEC = Size of exact clone
  - Number of lines of exact clone.
- LB = Line Bias
  - Distance between two exact clones.
  - Number of equal lines between the two clones.
- SDC = Size of Duplication Chain
  - Number of exact clones in a duplication chain.
  - Duplication chain = exact clones that are close to each other in terms of LB.





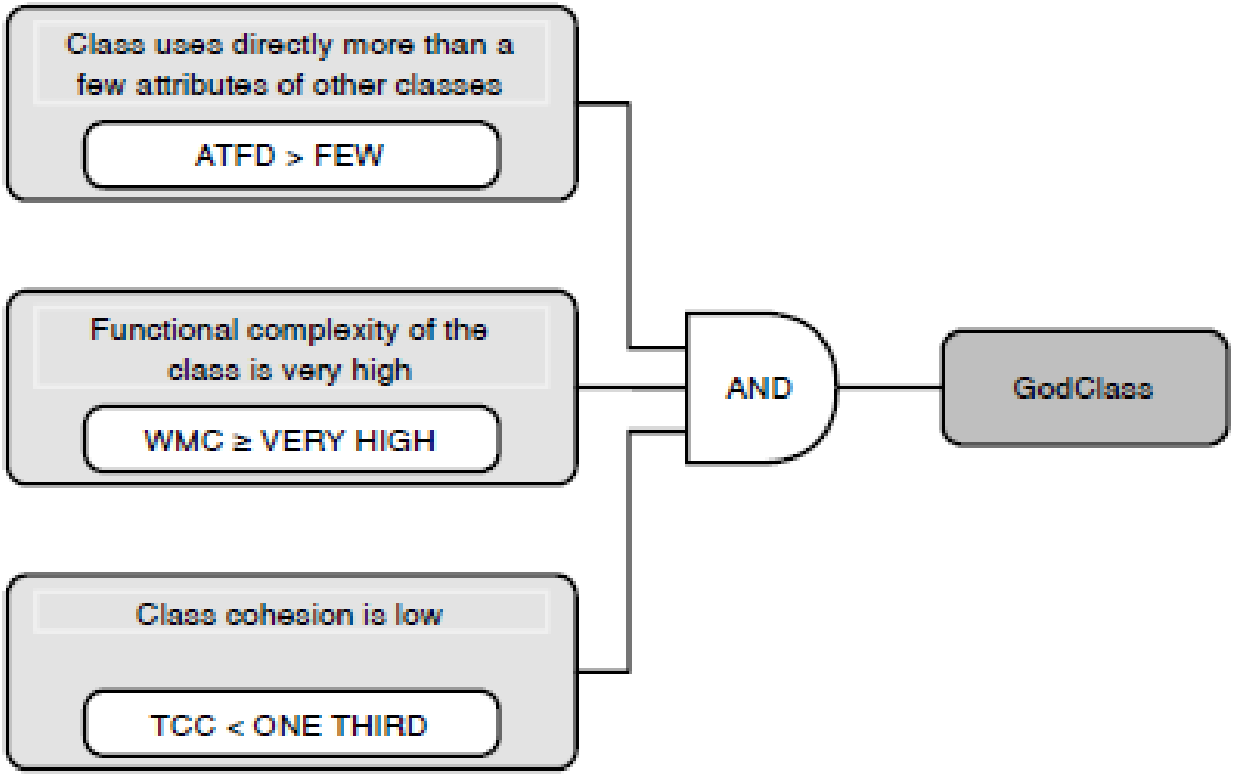
Long Method

- NOAV = Number of Accessed Variables
- MAXNESTING = Maximum Nesting Level



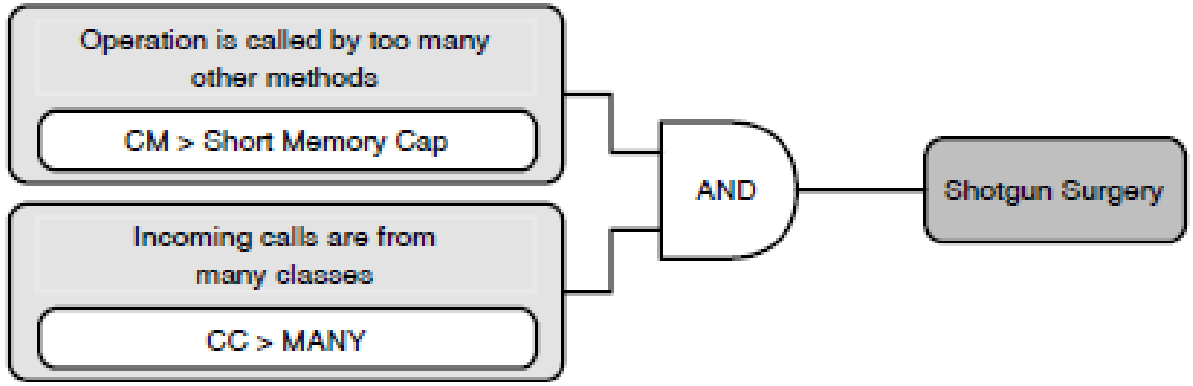
Large Class

- ATFD = Access to Foreign Data
  - Number of classes from which the class accesses attributes.



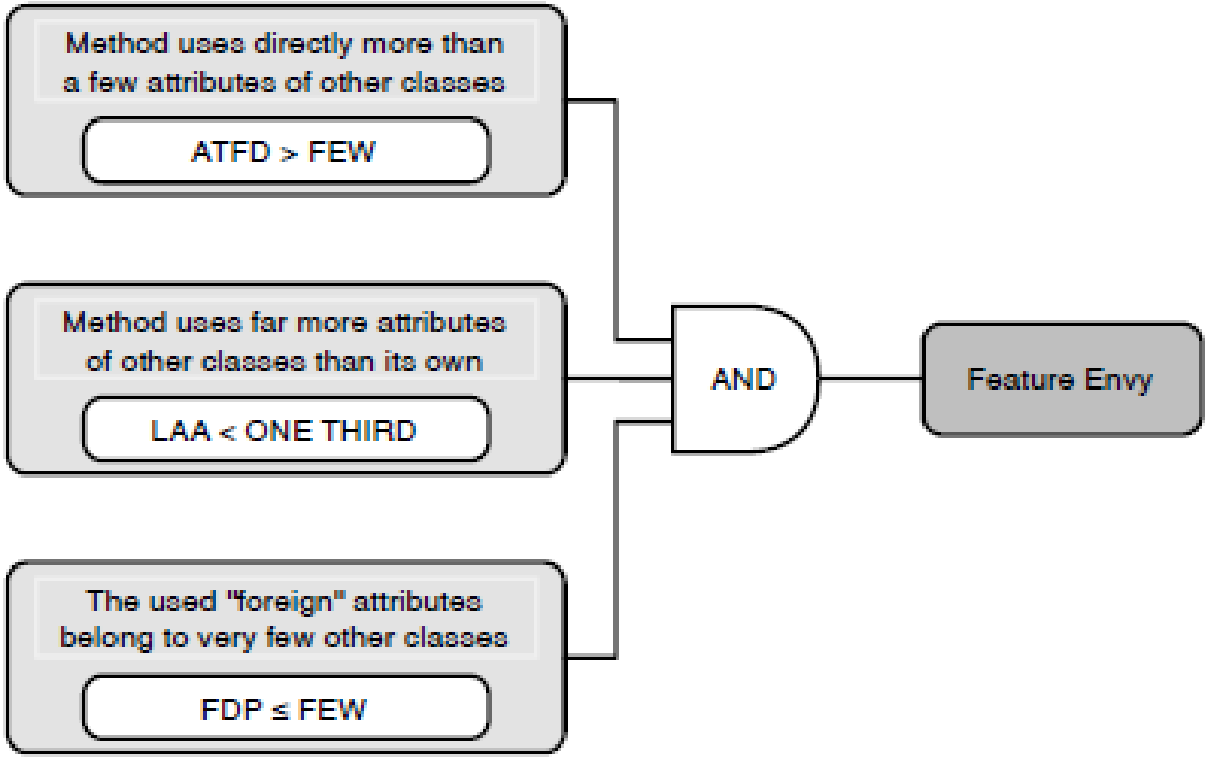
Shotgun Surgery

- CM = Changing Methods
  - Number of methods that call the measured method.
- CC = Changing Classes
  - Number of classes where the CM are defined.



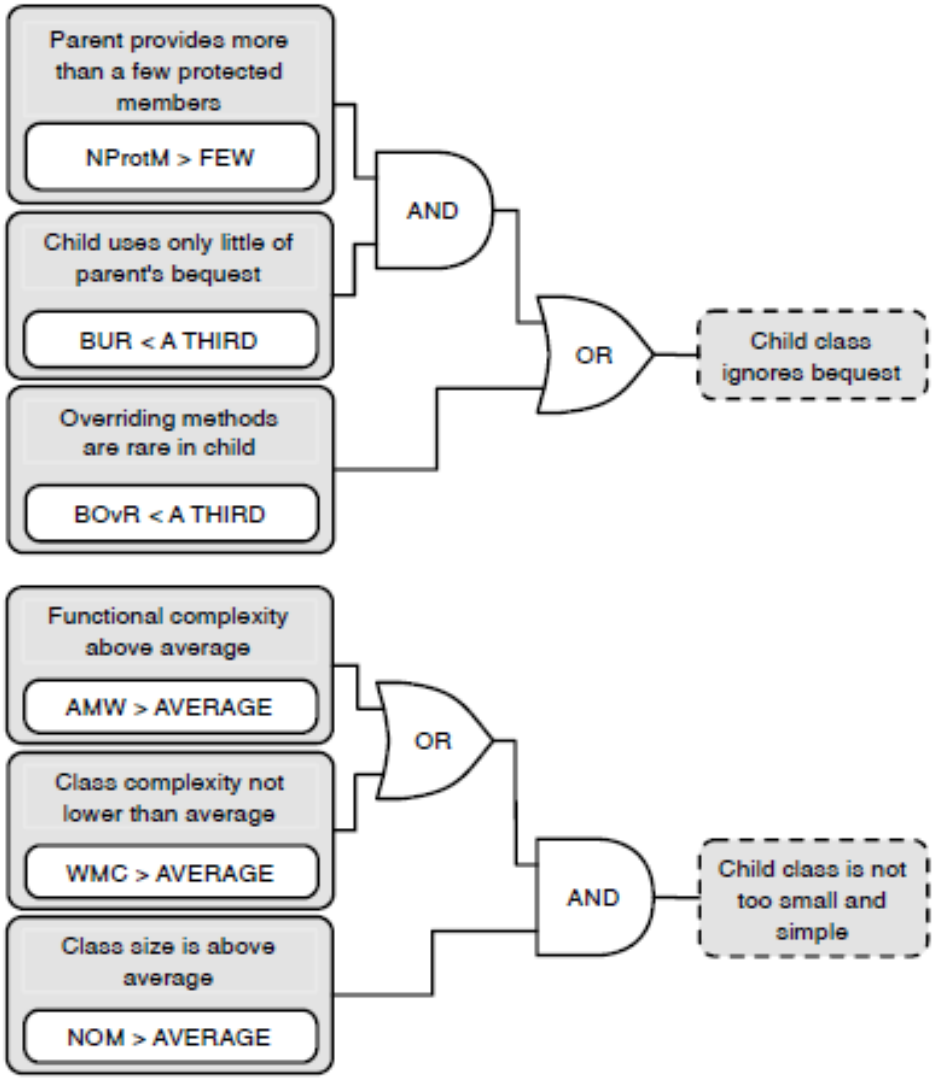
Feature Envy

- LAA = Locality of Attribute Accesses
  - Number of attributes belonging to the class of the method divided by the total number of attributes accessed by the method.
- FDP = Foreign Data Providers
  - Number of classes where the ATFD attributes are defined.

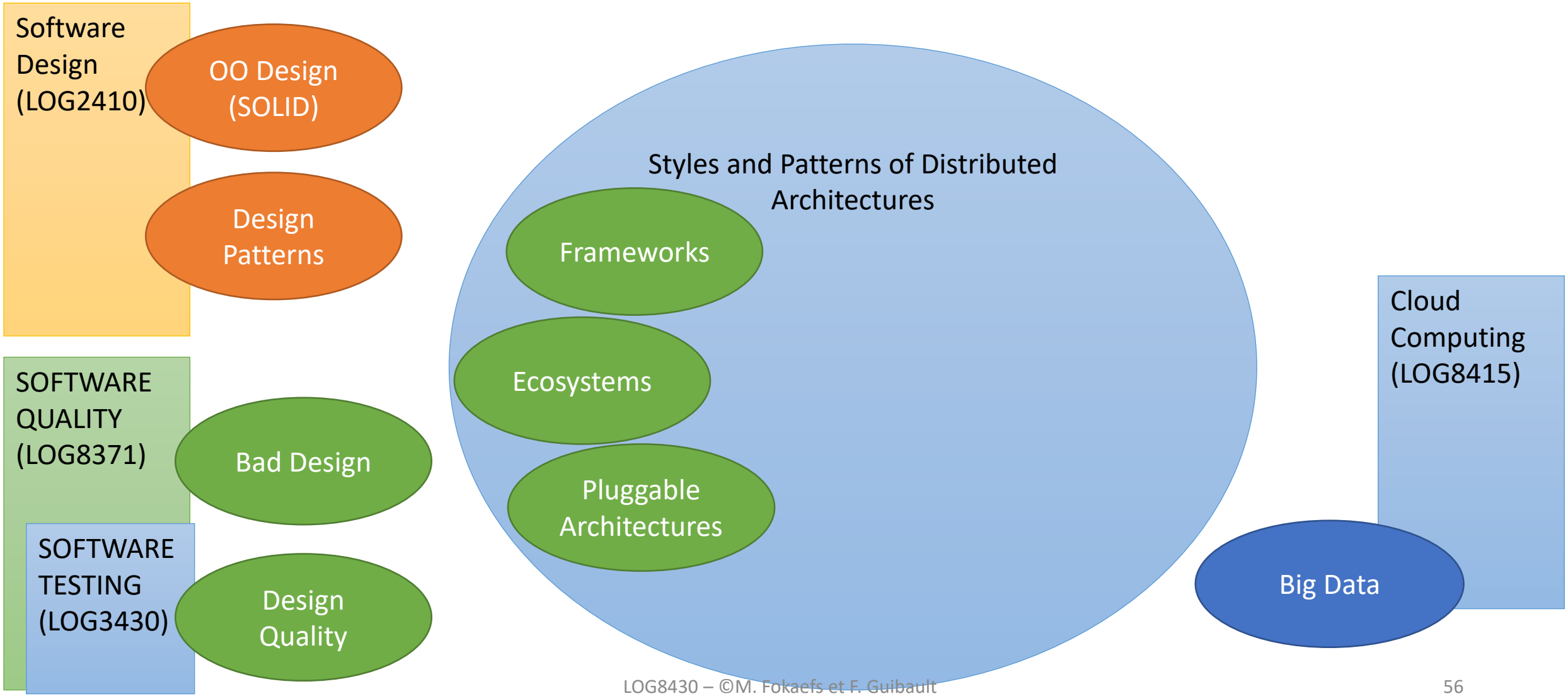


Refused Bequest

- NProtM = Number of Protected Members
- BUR = Base-class Usage Ratio
  - Usage ratio of protected members.
- BOvR = Base-class Overriding Ratio
  - Ratio of overridden members.
- AMW = Average Method Weight
  - Average complexity of all methods of a class.



Next time


Software Design (LOG2410)
OO Design (SOLID)
Design Patterns
SOFTWARE QUALITY (LOG8371)
Bad Design
SOFTWARE TESTING (LOG3430)
Design Quality
Styles and Patterns of Distributed Architectures
Frameworks
Ecosystems
Pluggable Architectures
Cloud Computing (LOG8415)
Big Data