CARL VON OSSIETZKY universität OLDENBURG

Software engineering and Key aspect of the field

Dilshod Kuryazov, MSc.

Software Engineering Group
Department of Computing Science
Carl von Ossietzky University Oldenburg

August 10, 2012

Carl von Ossietzky University

Statistics

Students: 11325

- Female: 6354
- Male: 4971

Professors: 182

- Female: 57
- Male: 125

Researcher: 999

- Female: 436
- Male: 563

Software Engineering Group

Head

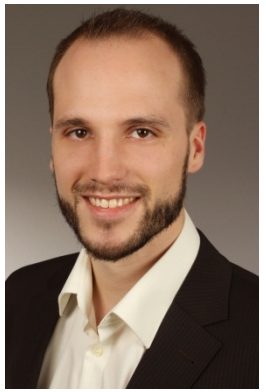
- Andreas Winter

Secretary

- Marion Bramkamp

PhD Students

- Jan Jelschen
- Maxat Kulmanov
- Dilshodbek Kuryazov
- Yvette Teiken (OFFIS)



Student Assistants

- Marion Gottschalk

Software Engineering Group

Topics in Research and Teaching

- Software-Engineering
- Modeling and Metamodeling
- Graph-Technology
  - Graph based modeling and implementation
- Process-Models in Software Development
- Software Evolution

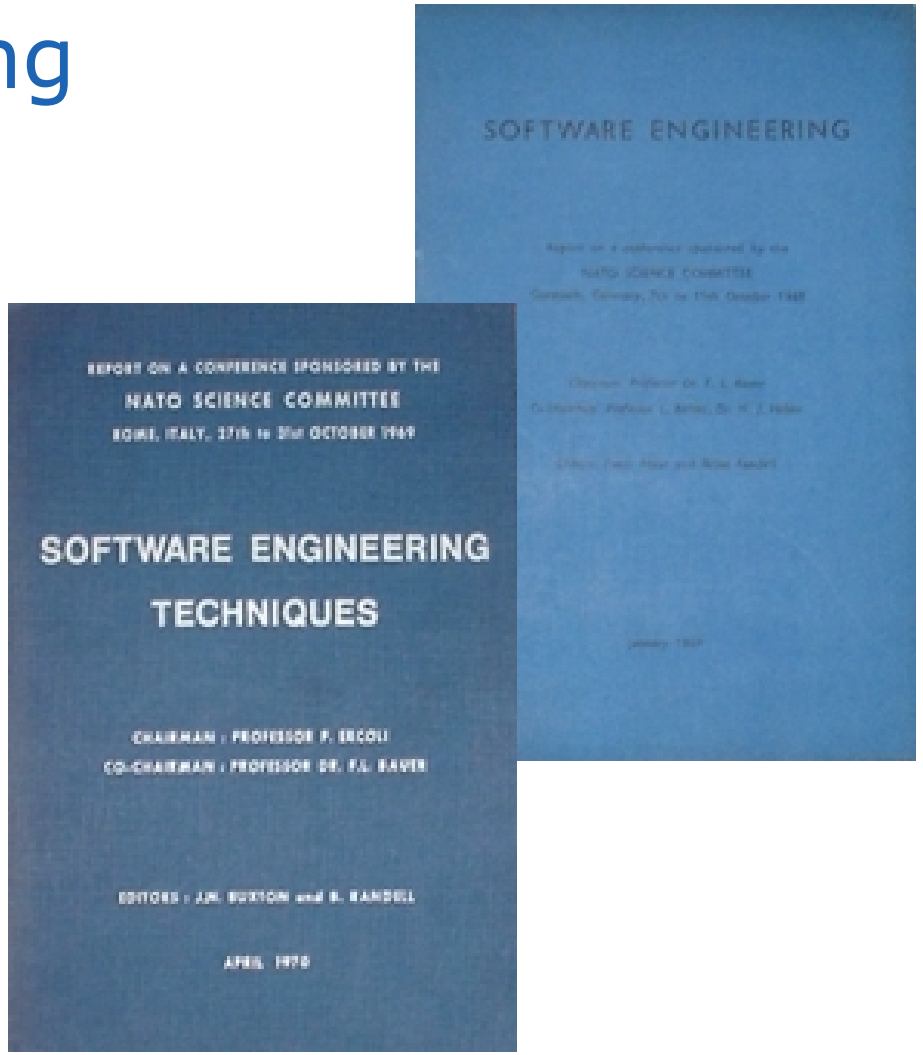
Mission

- Development and Application of Graph-Technology
  to improve Software Evolution

Software Engineering and Software Evolution

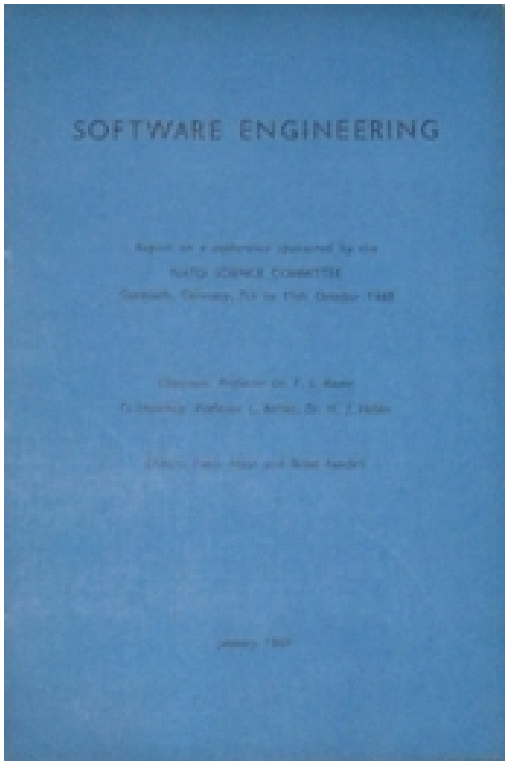
Software Engineering
Software Crisis

- Software development in the sixties
  - Increase of software complexity
  - missing suitable programming languages
  - missing suitable methods and techniques for engineering software systems
  - No mail, internet, Java, .net, eclipse, Google, sourceforge, twitter, facebook, ...



[http://homepages.cs.ncl.ac.uk/brian.randell/NATO/]

Software Engineering



[F. L. Bauer]

- *"[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."*
  (Software Engineering, Garmisch, October 7-11, 1968)

[IEEE Std. 601.12-1990, 1993]

- Software Engineering:
  (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  (2) The study of approaches as in (1).

Software Engineering

http://www.mobileslead-just-needs-high-efficiency/

Engineering

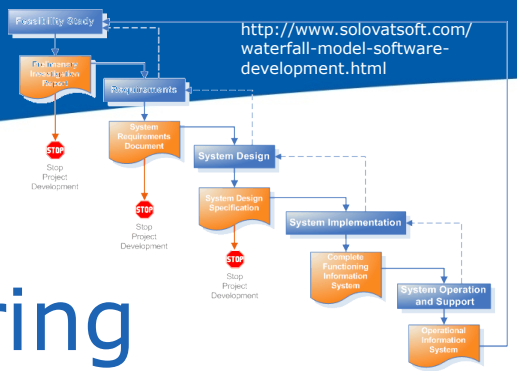
- follows established principles
- applies methods and techniques purposefully
- looks for technically and costly efficient solutions
- rejects blindly and imprudently ad hoc problem solving

Software Engineering

- elicits and clearly defined system requirements
- constructs (models) alternative solutions (software architecture)
- evaluates solutions
- realizes solutions (i.e. programming)
- reviews solutions according their requirements

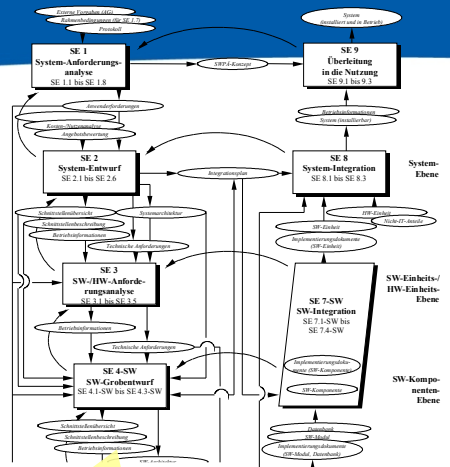
Conclusion

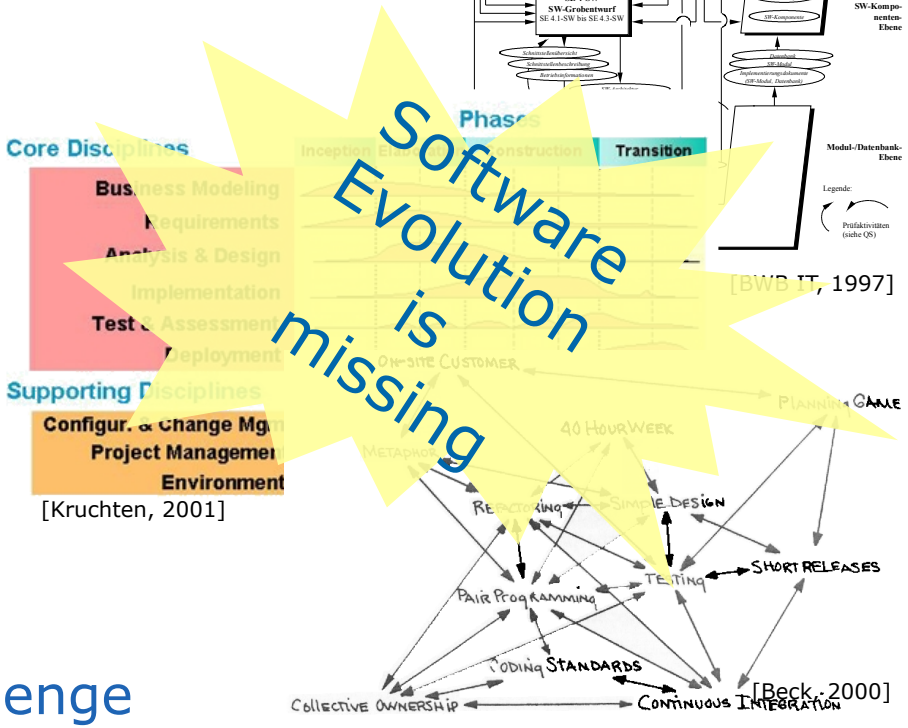
- **`software engineering != programming`**



http://www.solovatsoft.com/waterfall-model-software-development.html

Software Engineering





[Kruchten, 2001]

[BWB IT, 1997]

[Beck, 2000]

Software development activities

- plan and organize projects
- elicit requirements
- define software architecture
- construct software systems
- test software systems
- run software systems

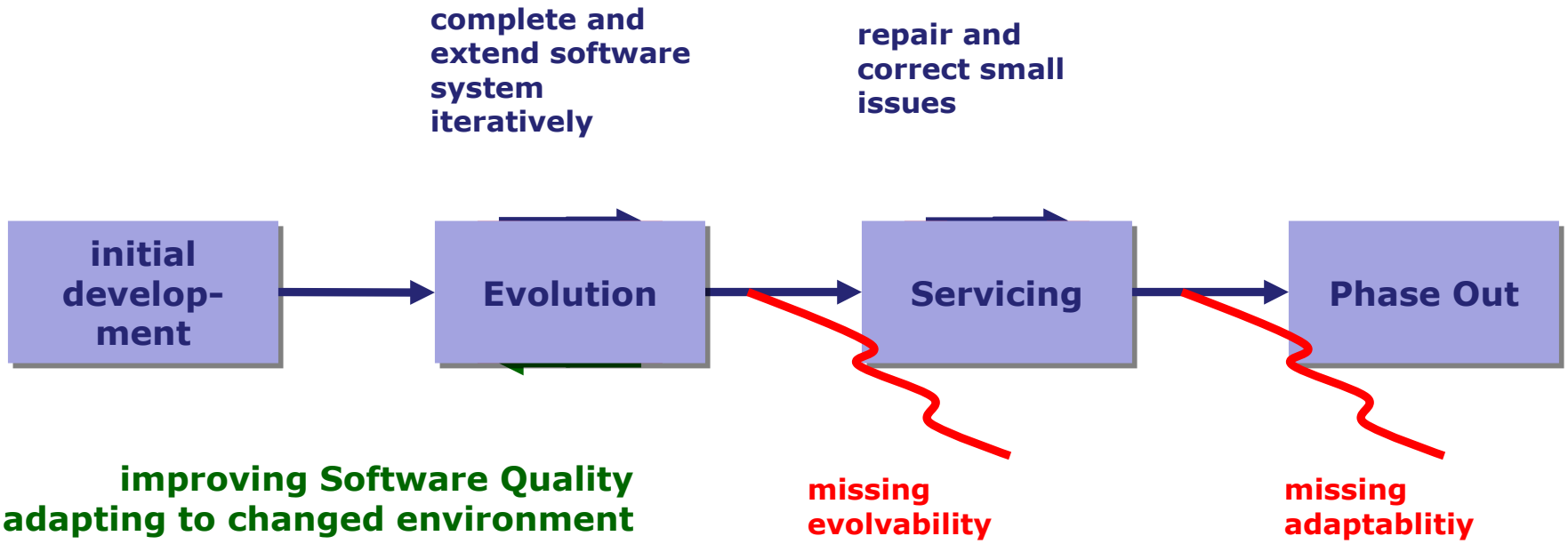
Today´s software development challenge

- improve software development methods and technologies
- keep existing software systems fulfilling user´s needs

Software Evolution Life Cycle

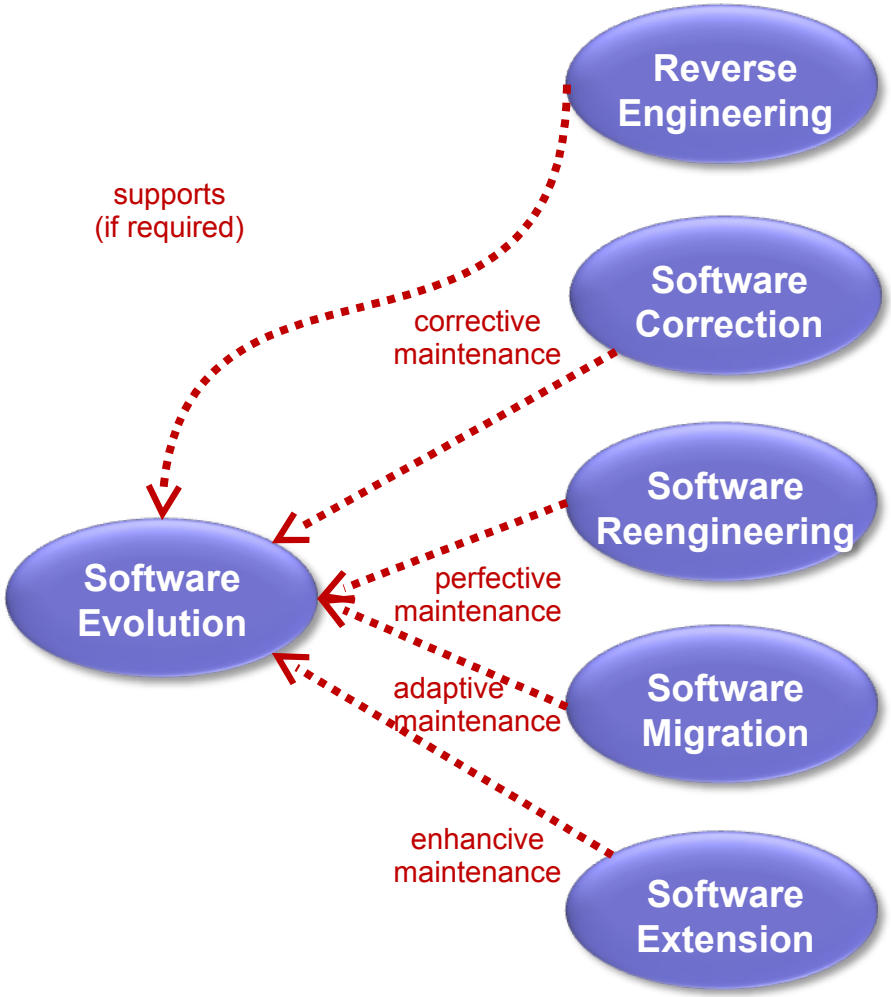
Software Evolution

- covers all activities to keep an existing software system running in its changing environment



[Rajlich/Bennett, 2000]

Activities in Software Evolution



Extracting a more abstract system description

software → documentation

eliminating software errors

software → more (?) correct software

improving software quality (not changing functionality)

software → better (?) software

transferring software to new environment (not changing functionality)

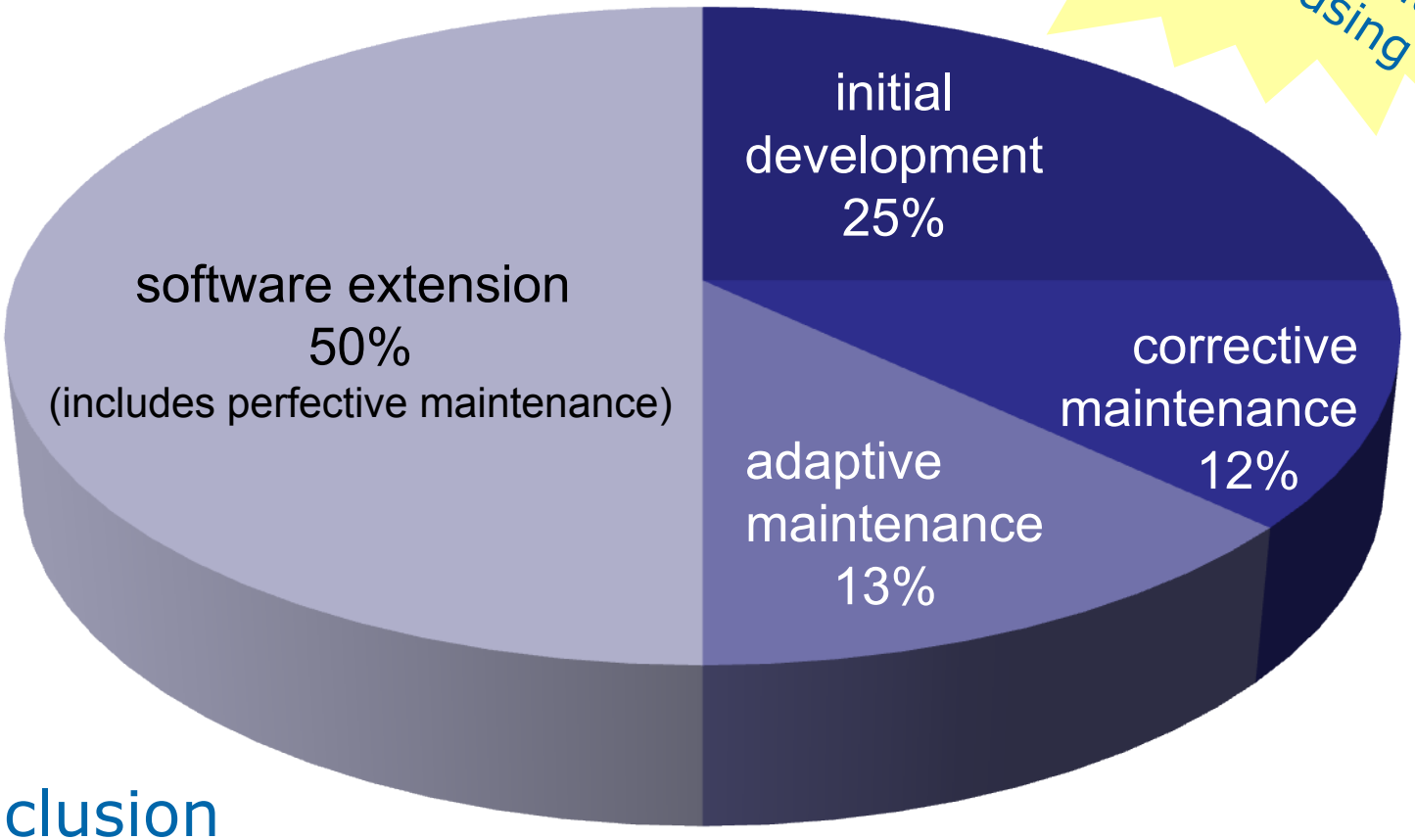
software → software in environment

extending software

software → software with new or changed functionality

Software Evolution

initial development is decreasing



Conclusion

most work in software development has to deal with existing systems

[Lientz/Swanson, 1980, McKee, 1984, Nosek/Palvia, 1990]

1.1. What is a Model?

A model is

- a simplification of reality;

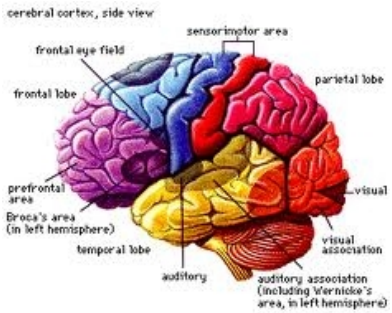
- a representation in a certain medium of something in the same or other medium;

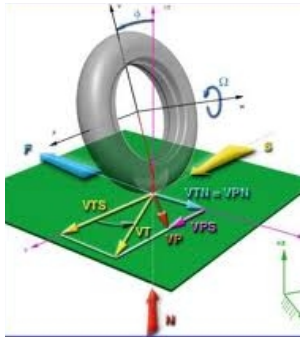
- a systematic description of an object or phenomenon that shares important characteristics with the object or phenomenon.

1.1. What is a Model?

Characteristics of models:

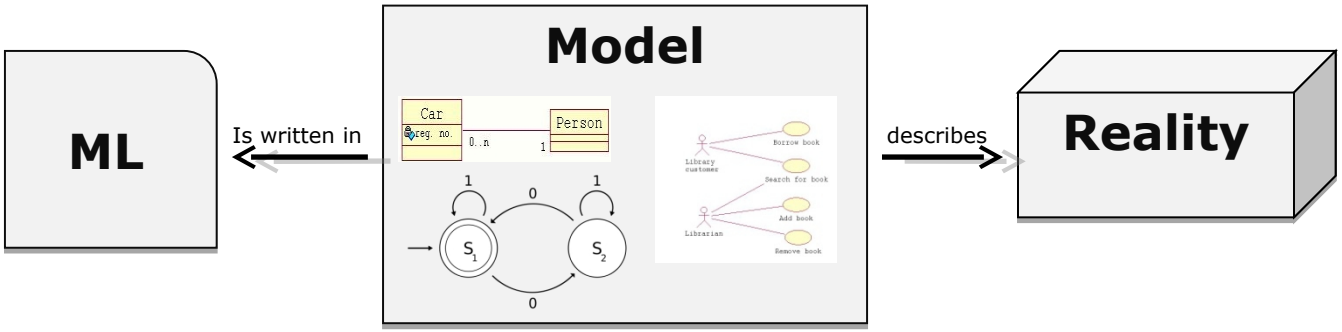
- a representation, on a smaller scale, of a device, structure, etc;
- a standard to be imitated;
- a representative form, style, or pattern;
- a person who wears clothes to display them;
- a design or style, designs of particular product;
- a simplified representation or description of a system or complex entity, designed to facilitate calculations and predictions;
- an interpretation of a formal system;





What is a Model in computer science?

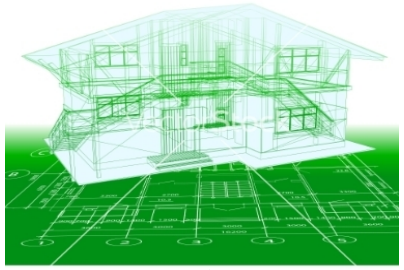
A **model** is a description of (part of) a system written in well-defined language.



A **well-defined language** is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer.

An **abstraction** helps system users to understand (part of) a system and interaction of subsystems in it.

Abstraction



Modelling

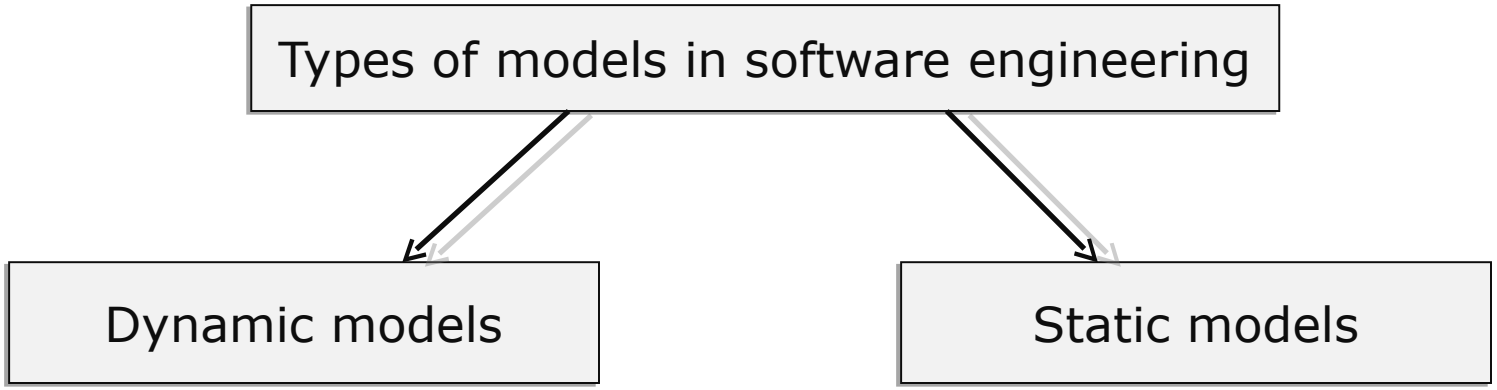
System

1.2. Why do we need models?

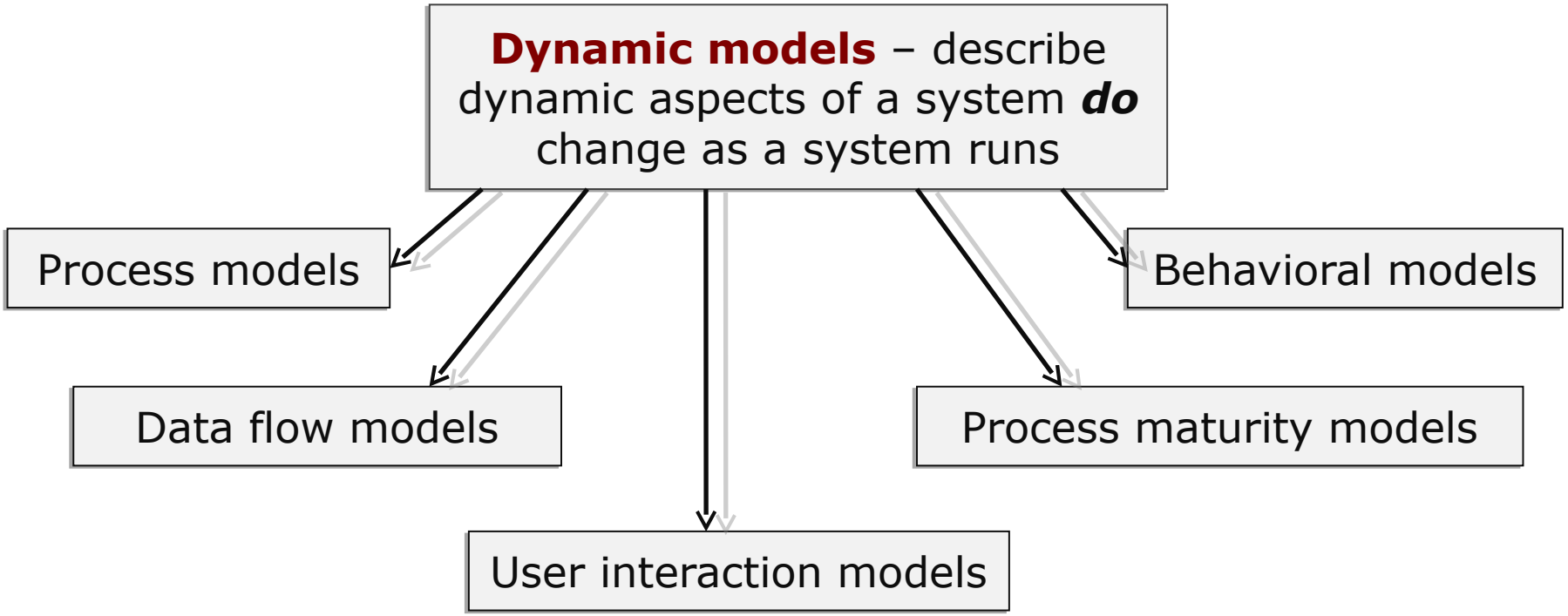
Models are to

- visualize a system;
- analyst to understand the functionality of the system;
- communicate with customers;
- document the decisions;
- show external and internal prespective of a system‘s context;

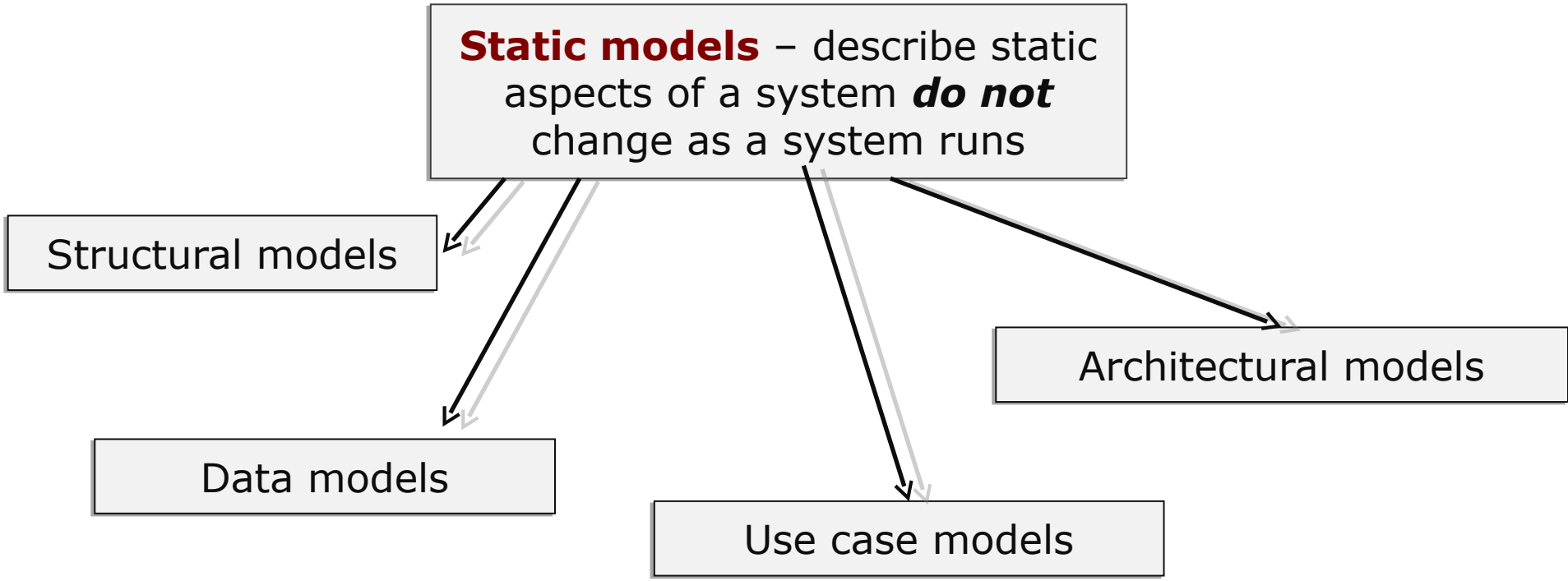
1.3. What kind of models are in software engineering?

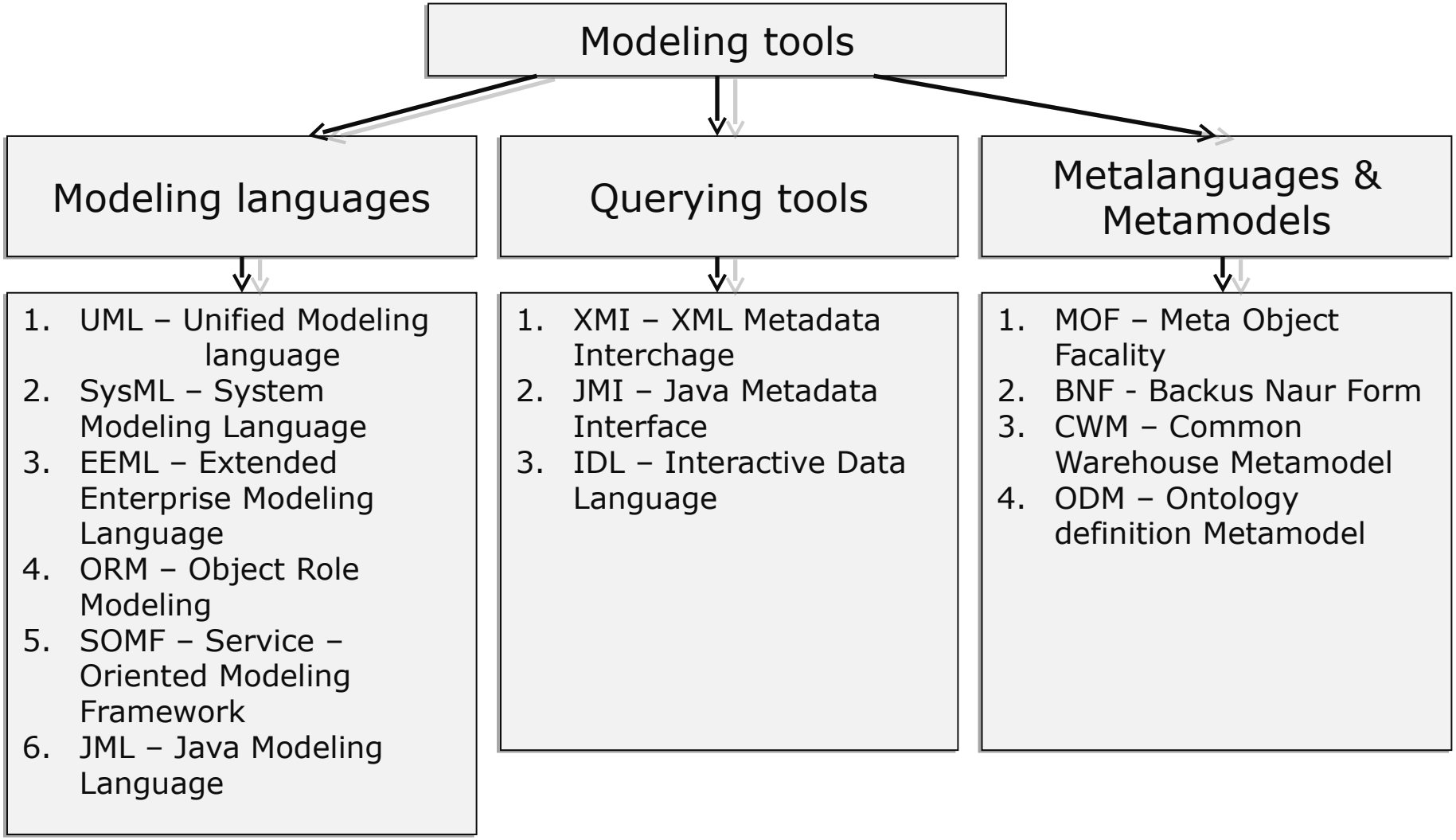


1.3. What kind of models are in software engineering?



1.3. What kind of models are in software engineering?

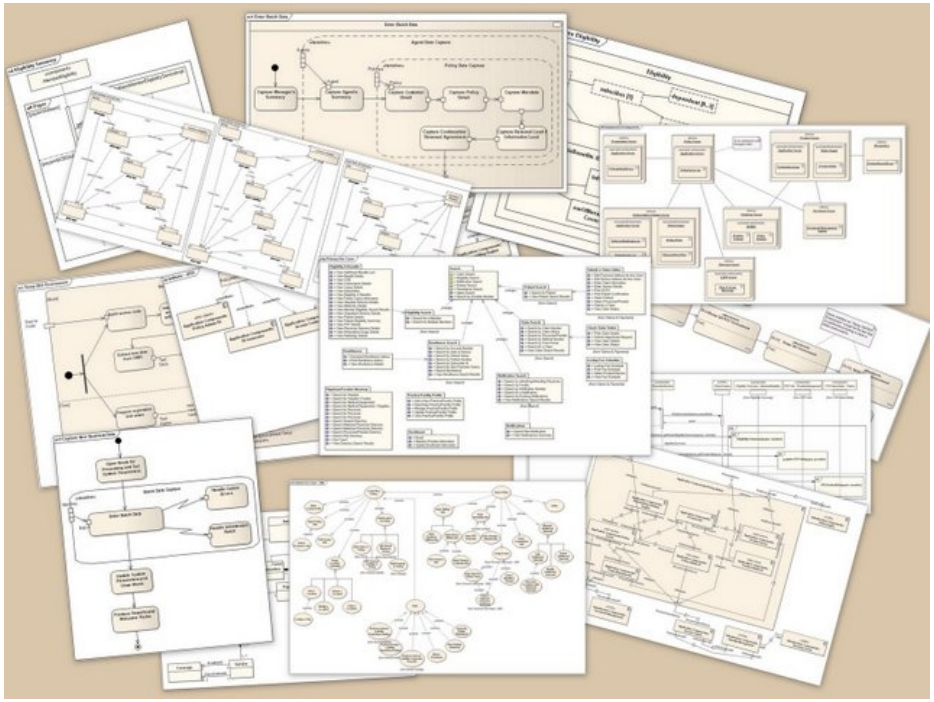



Modeling tools
Modeling languages
1. UML – Unified Modeling language
2. SysML – System Modeling Language
3. EEML – Extended Enterprise Modeling Language
4. ORM – Object Role Modeling
5. SOMF – Service – Oriented Modeling Framework
6. JML – Java Modeling Language
Querying tools
1. XMI – XML Metadata Interchage
2. JMI – Java Metadata Interface
3. IDL – Interactive Data Language
Metalanguages & Metamodels
1. MOF – Meta Object Facality
2. BNF - Backus Naur Form
3. CWM – Common Warehouse Metamodel
4. ODM – Ontology definition Metamodel

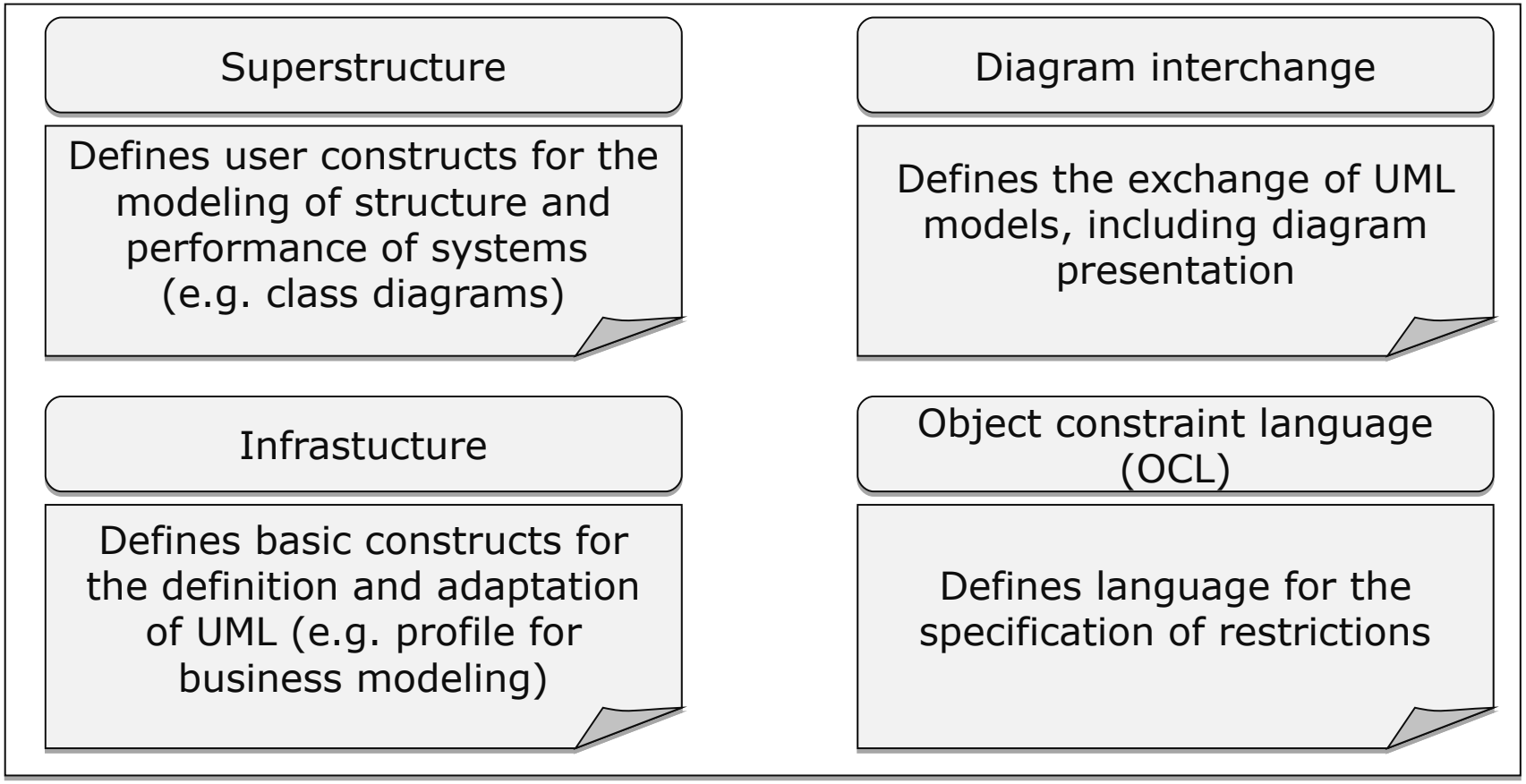
2. UML overview

The UML is a language for

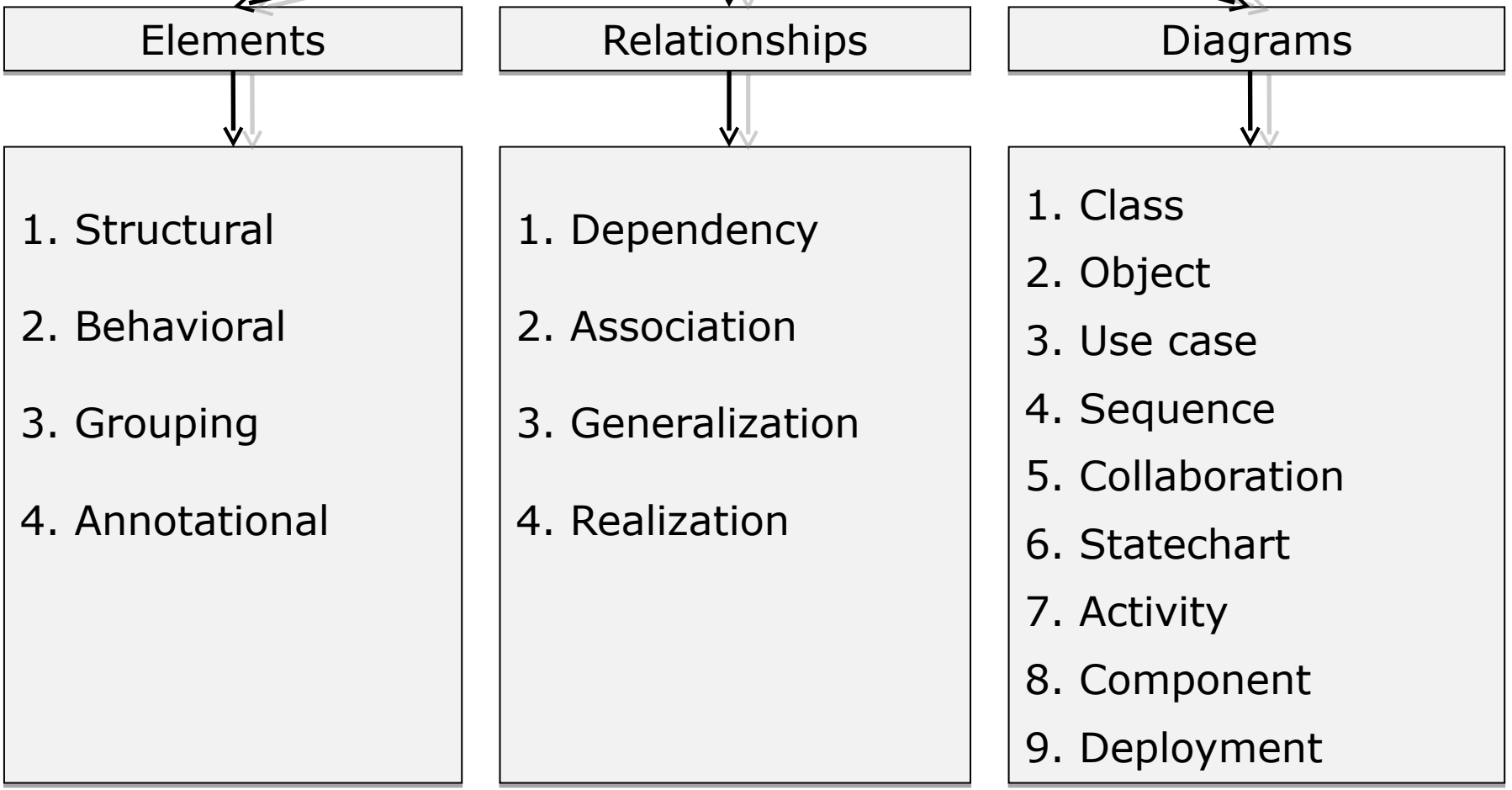
- Visualizing
- Specifying
- Constructing
- Documenting



2.1. Understanding concepts of UML.
The complete UML 2.0 package



2.2. Building blocks of the UML



Elements in the UML

Structural elements are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical.

- ***1. Class***
- ***2. Interface***
- ***3. Collaboration***
- ***4. Use case***
- ***5. Active class***
- ***6. Component***
- ***7. Node***

Behavioral elements are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space.

- ***1. Interaction***
- ***2. State machine***

Grouping elements are the organizational parts of UML models. These are the boxes into which a model can be decomposed.

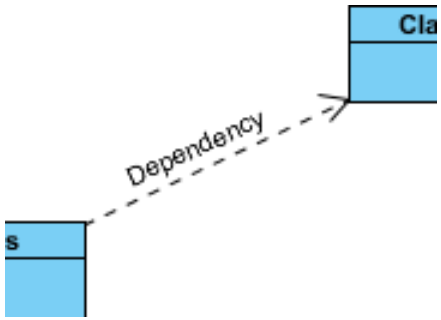
- ***1. Packages***

Annotational elements are the explanatory parts of UML models. These are the comments you may apply to describe, illuminate, and remark about any element in a model.

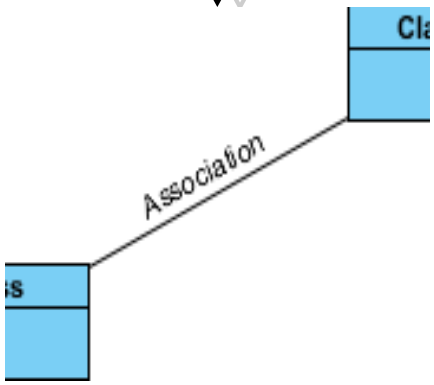
- ***1. Note***

Relationships in the UML

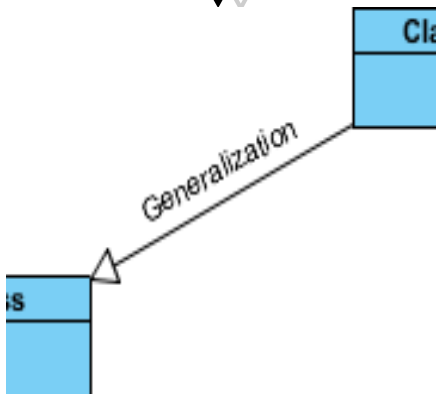
A ***dependency*** is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing.



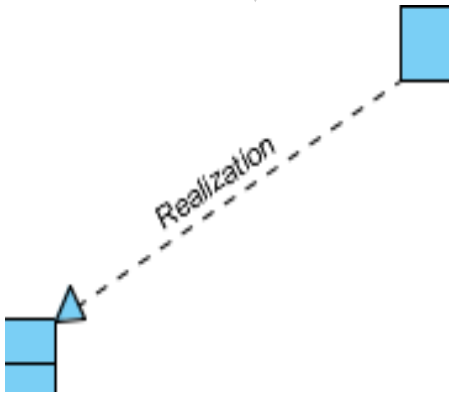
An ***association*** is a structural relationship that describes a set of links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.



A ***generalization*** is a specialization/generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent). In this way, the child shares the structure and the behavior of the parent.



A ***realization*** is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. In two places: between interfaces and the classes or components that realize them, and between use cases and the collaborations that realize them.



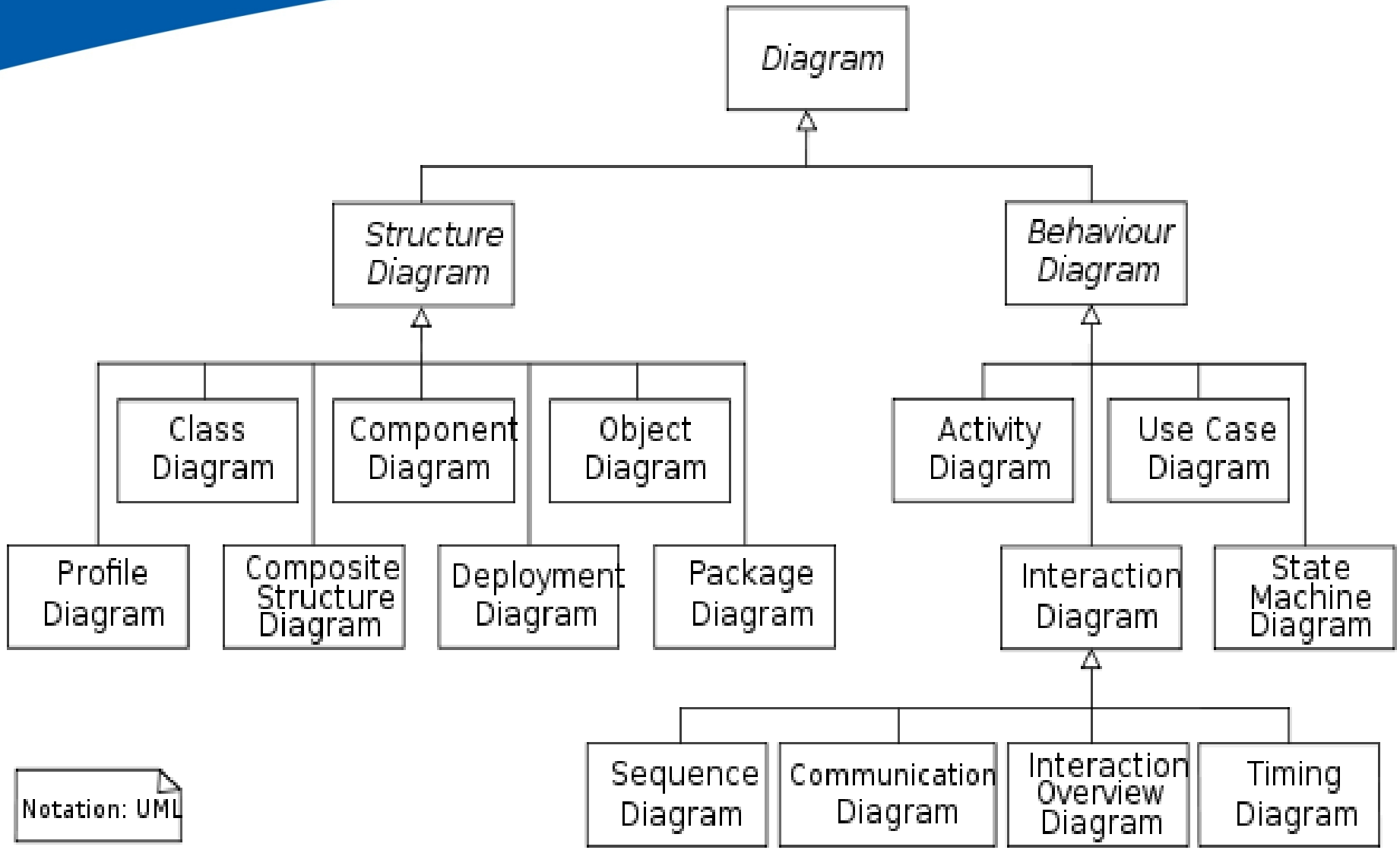
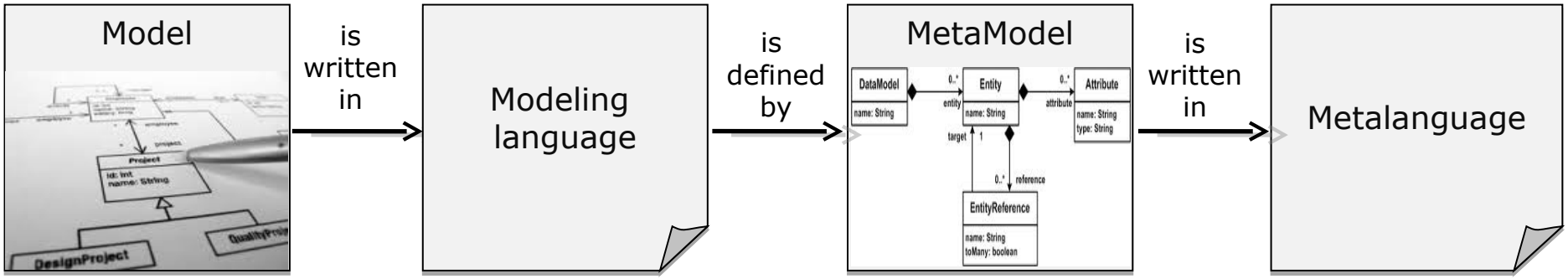


Diagram
Structure Diagram
Behaviour Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

2.3. UML Metamodeling

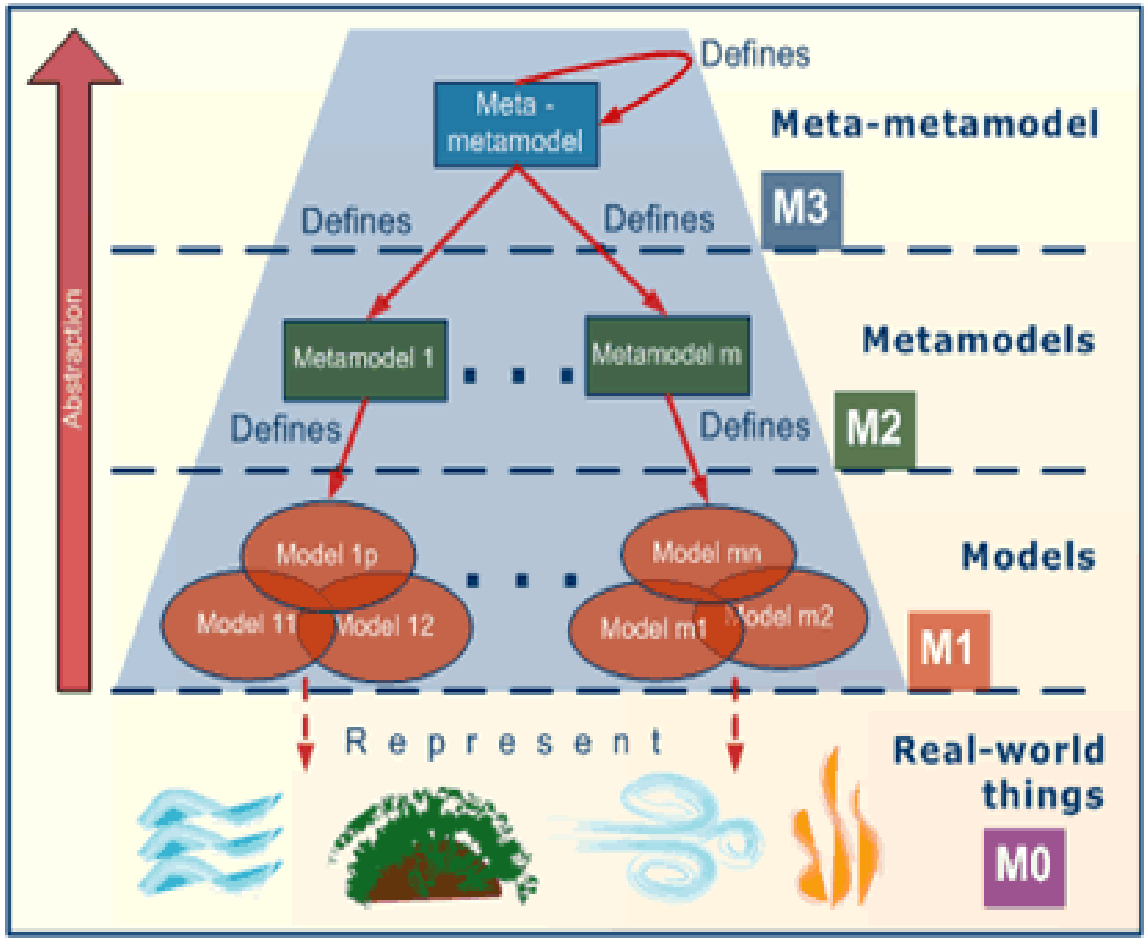
A ***metamodel***

- is at a higher level of abstraction than a model;
- is called "a model of a model";
- is the rules/grammar for the modelling language (ML) itself;
- describes the rules and constraints of metatypes and metarelationships;
- is a description of a modelling language;
- defines all the concepts that can be used within a language.

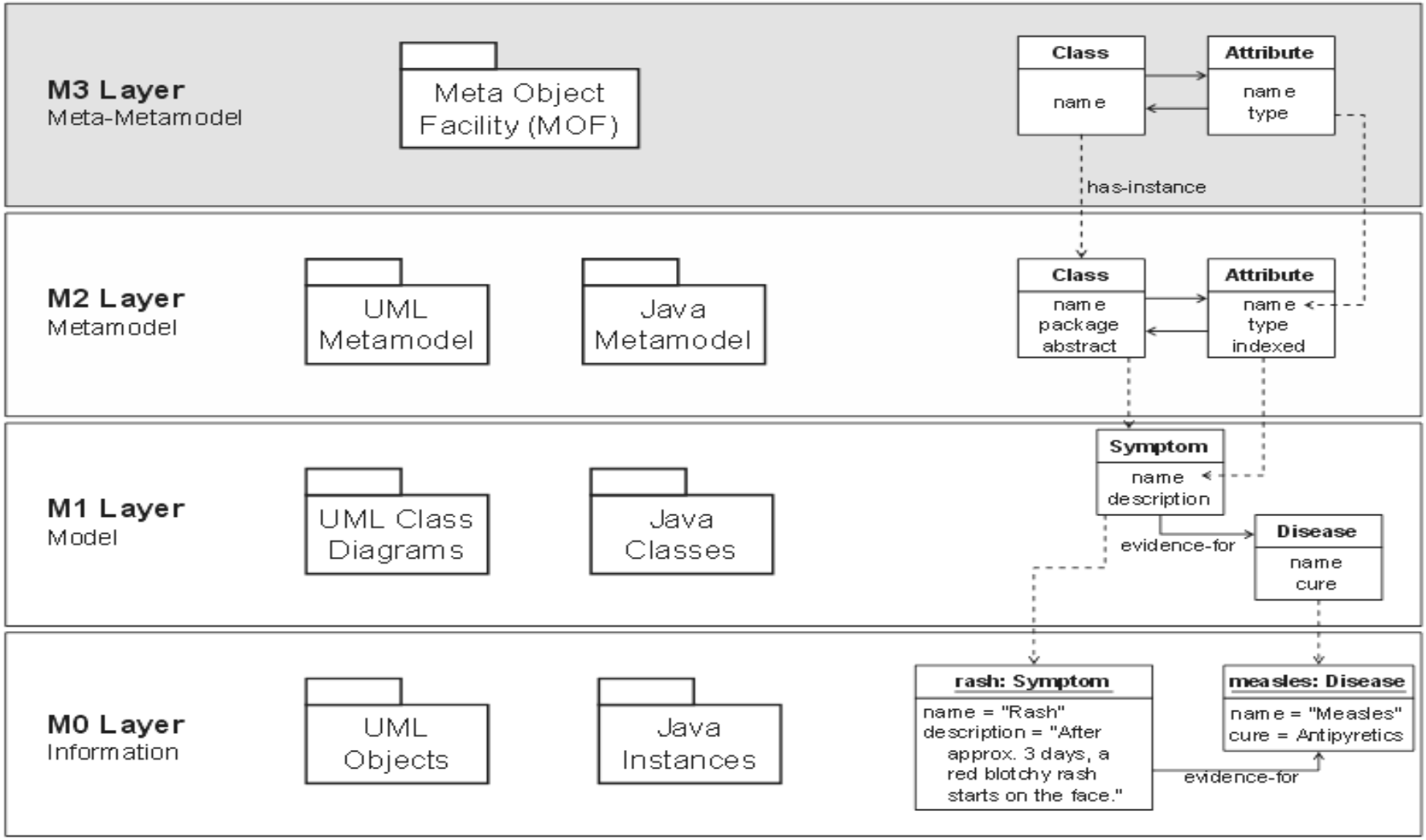


Architecture of metamodel layers

Layer	Description
M3	The infrastructure for a metamodeling architecture. Defines the language for specifying metamodels.
M2	An instance of a meta-metamodel. Defines the language for specifying a model
M1	An instance of a metamodel. Defines a language to describe an information domain.
M0	An instance of a model. Defines a specific information domain.



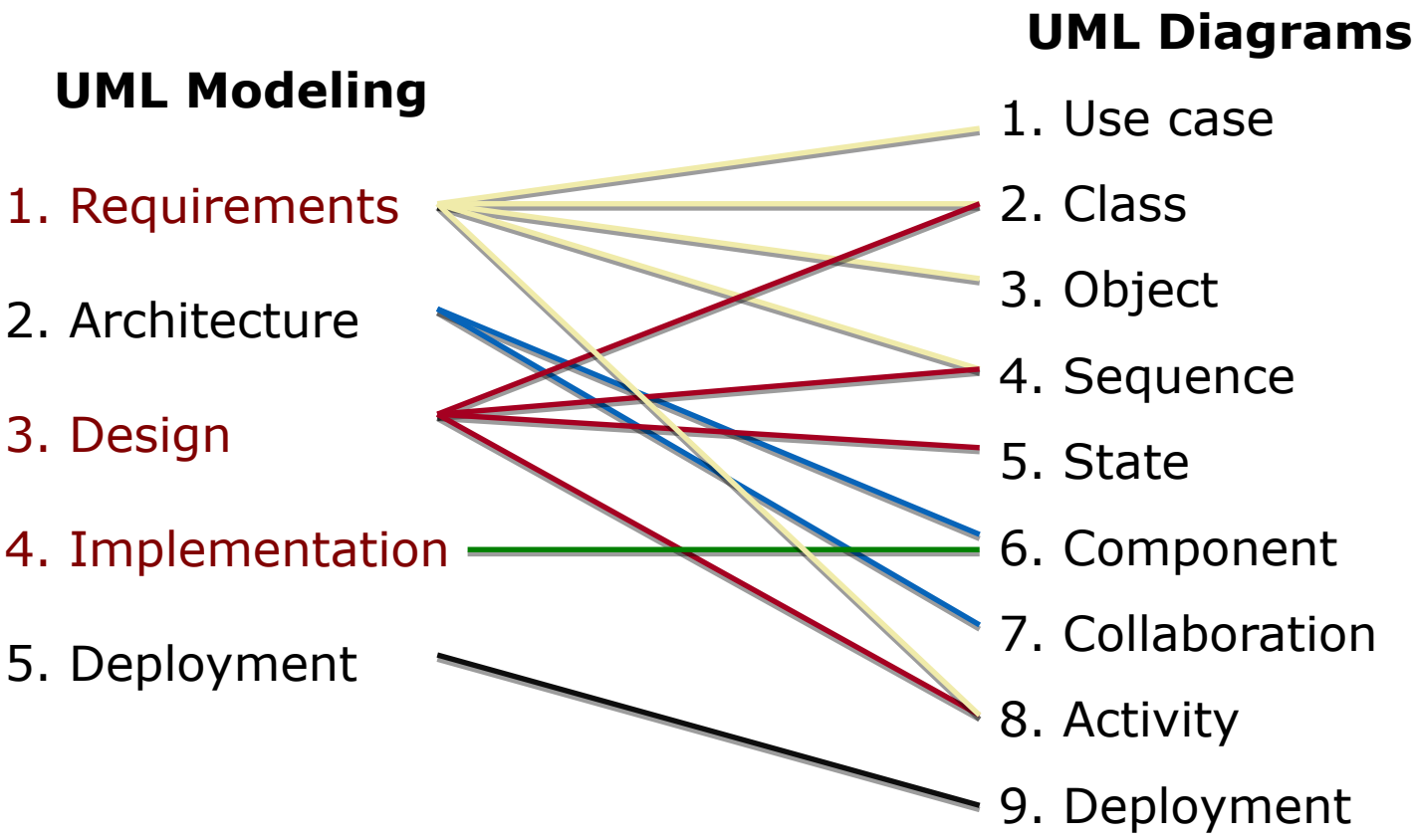
Metamodeling layers



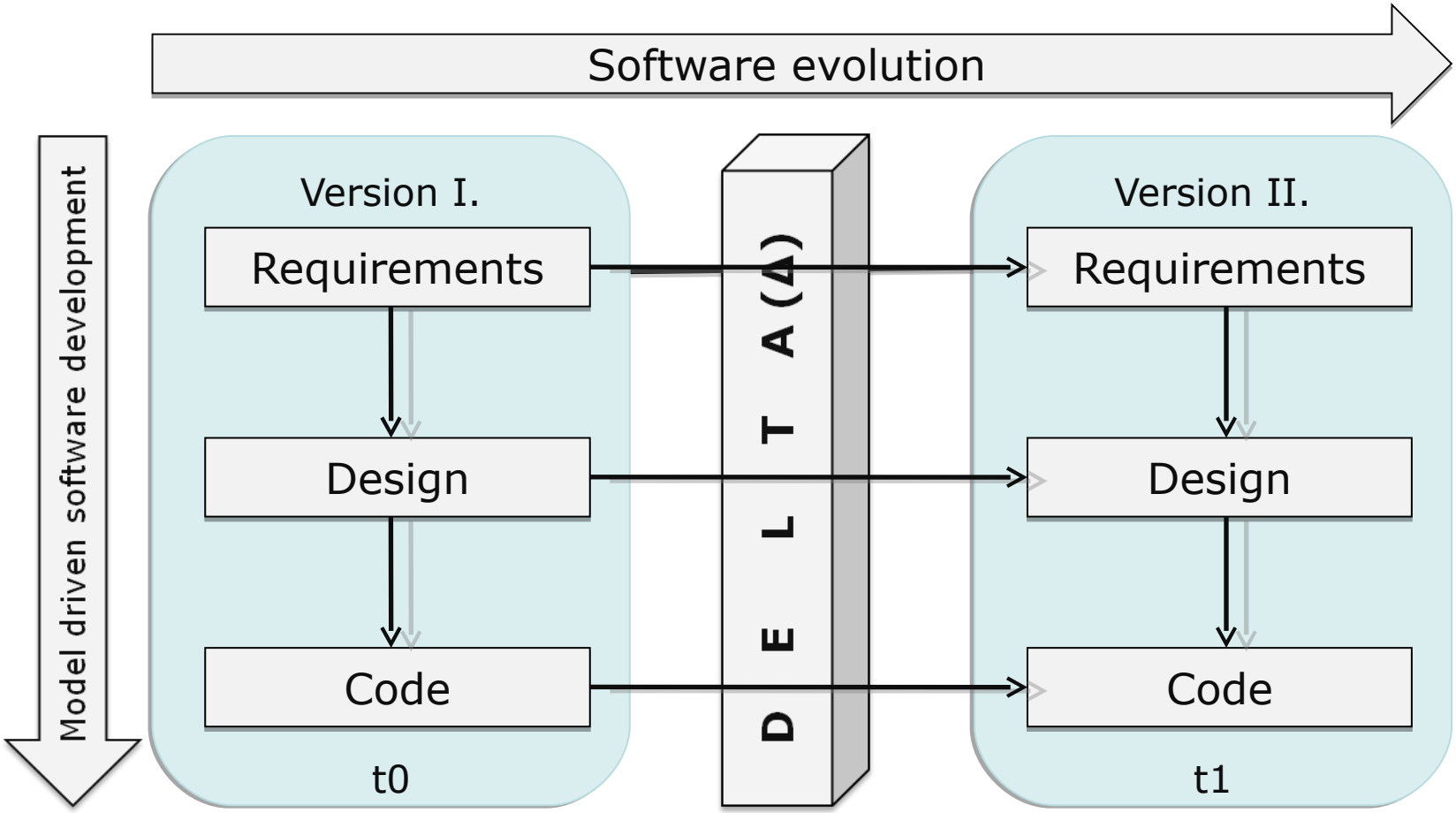
What is Meta Object Facility (MOF)?

- Metadata management framework and set of metadata services;
- Enable development and interoperability of model and metadata driven systems;
- MOF is used in most MDA-related technologies (UML, XMI, UML profiles, JMI);
- MOF has improved interoperability and productivity (all metamodels are based on the same metametamodel);
- MOF 2.0 Model is a framework for metamodelling and metadata representation and management;
- MOF 2.0 IDL and MOF 2.0 Java are the mappings from MOF 2.0 to IDL and Java respectively;
- MOF 2.0 Query/View/Transformation is a framework to define transformations based on MOF metamodels.

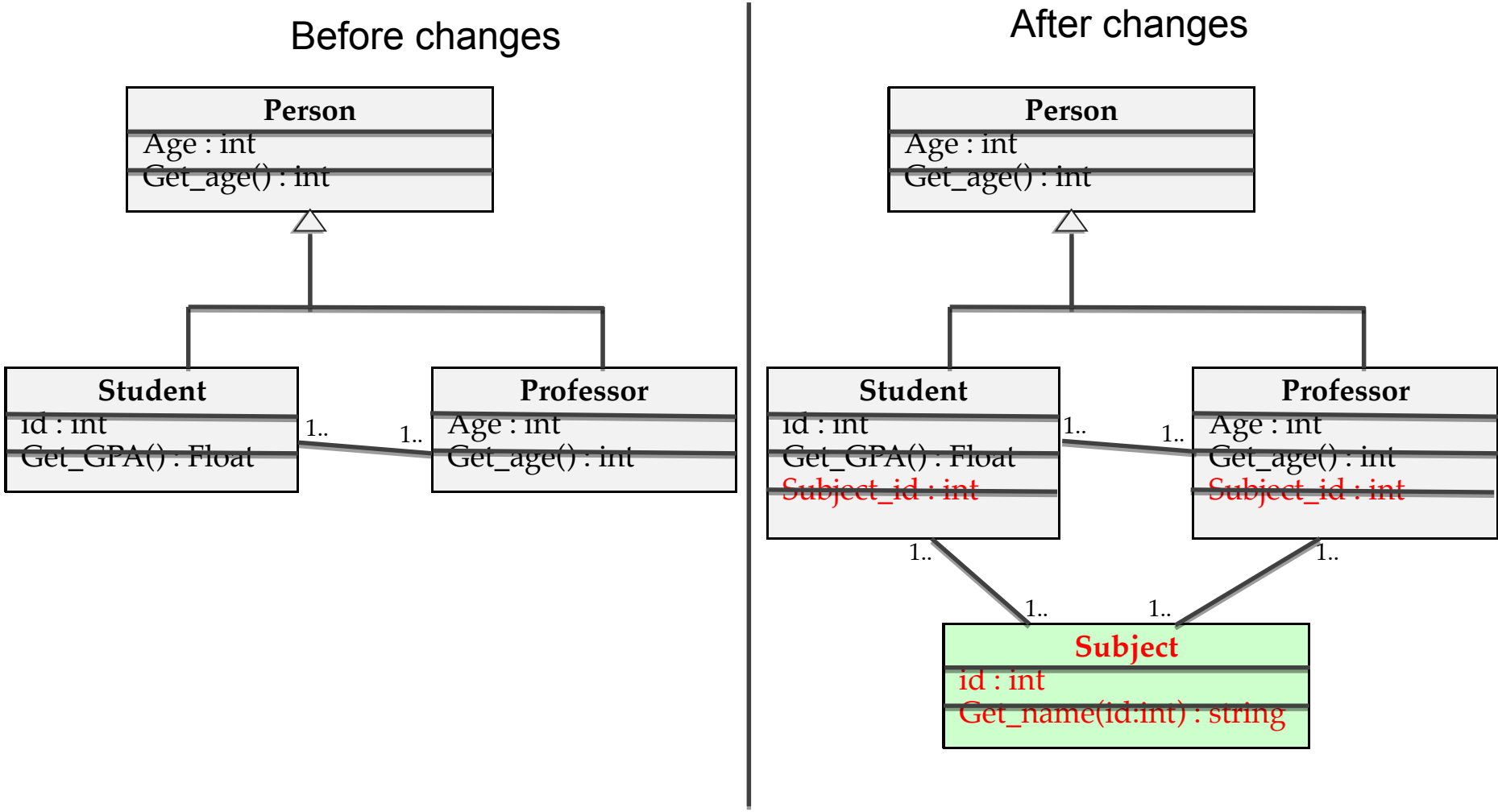
3. UML in Action



Understanding modeling deltas (Δs)

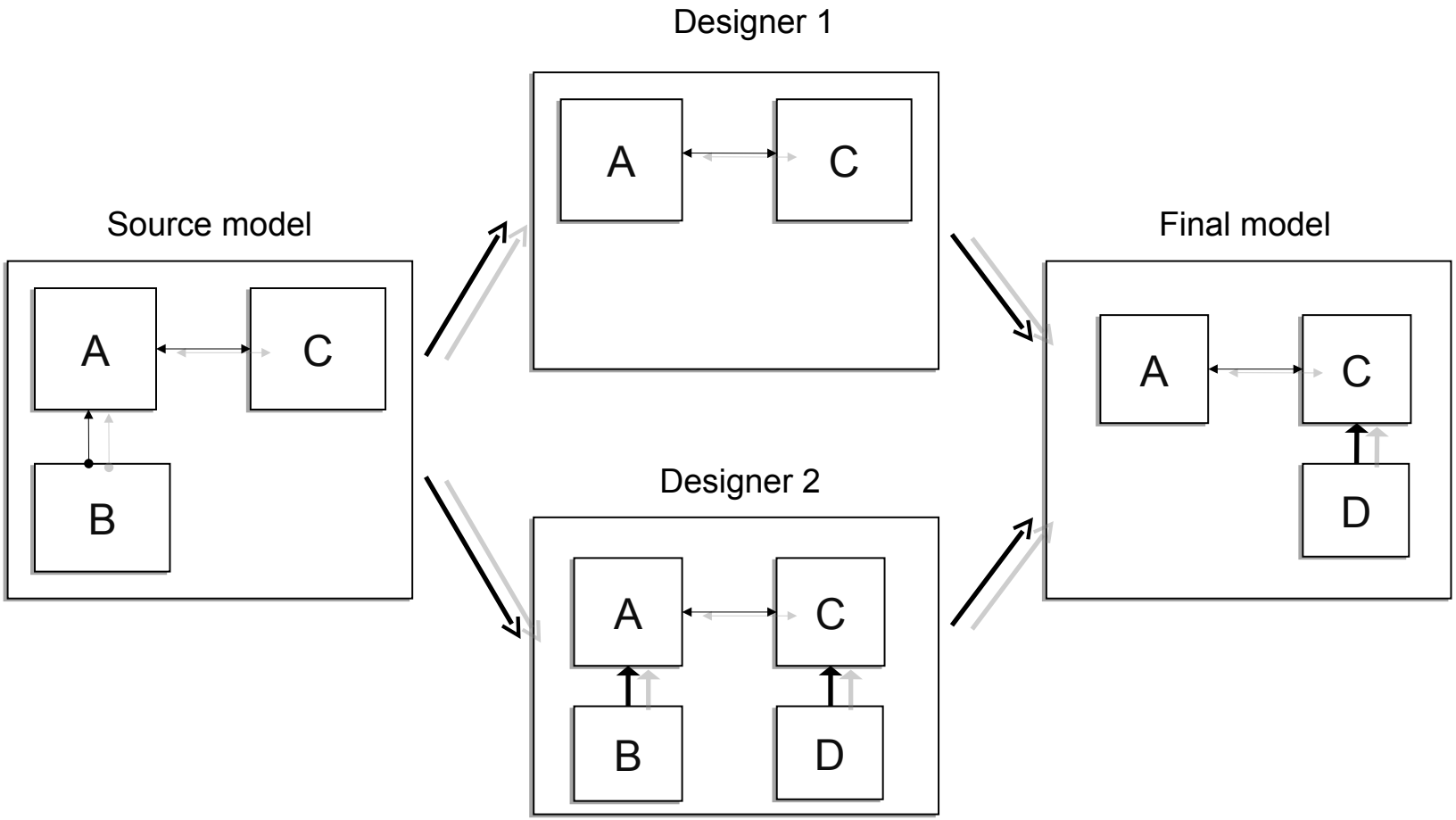


3.1. Modeling deltas

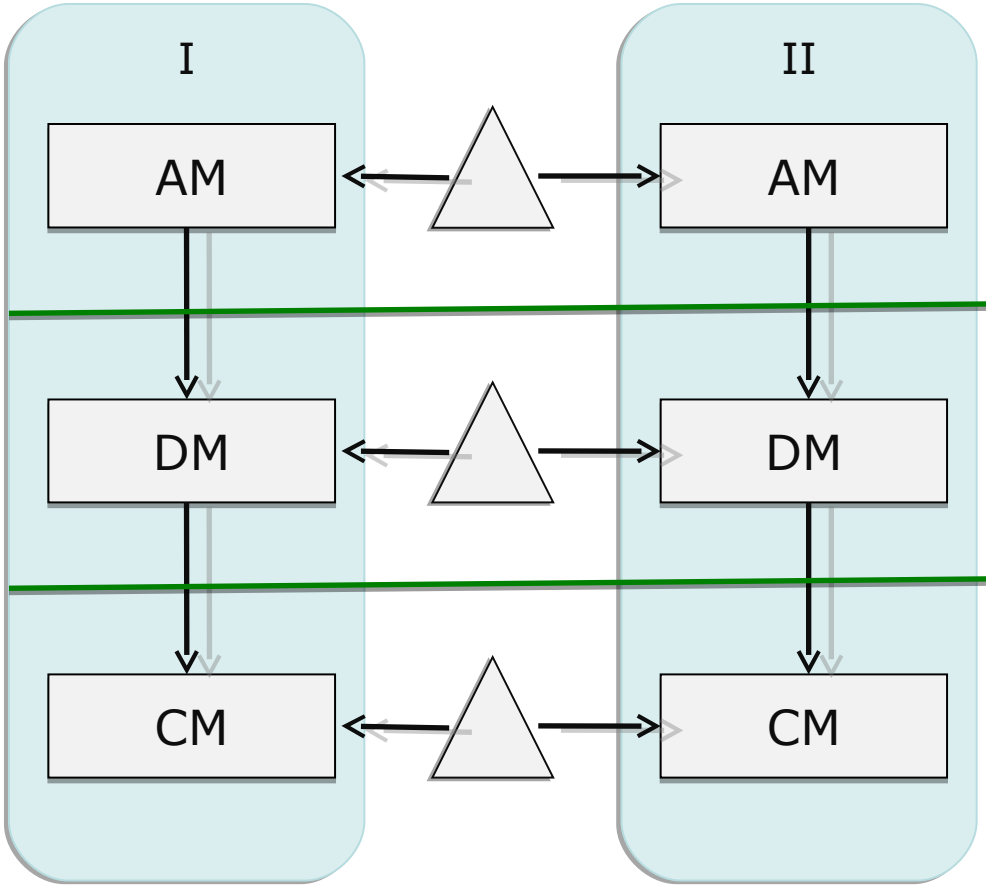


3.1. Modeling deltas

(presentation, merging, storing, adapting)



Difference Algorithms

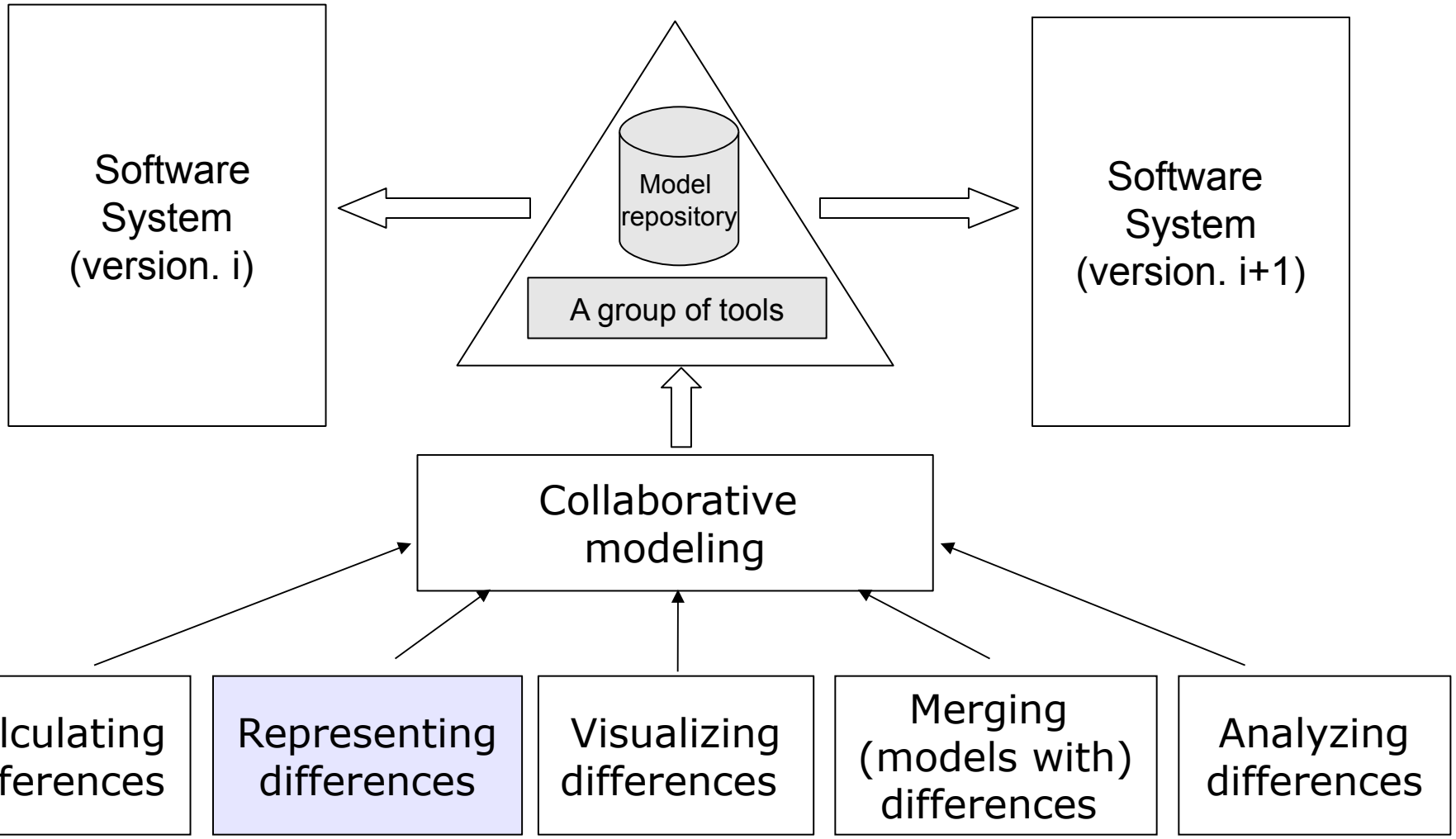


ReeDSedS requirements driven software development system (MOLA tools), SIDIFF (CDDiff, ADDiff).

SIDIFF (CDDiff, ADDiff), ReeDSedS

A text based versioning tools RCS, CVS, SCCS and subversion, ReeDSedS.

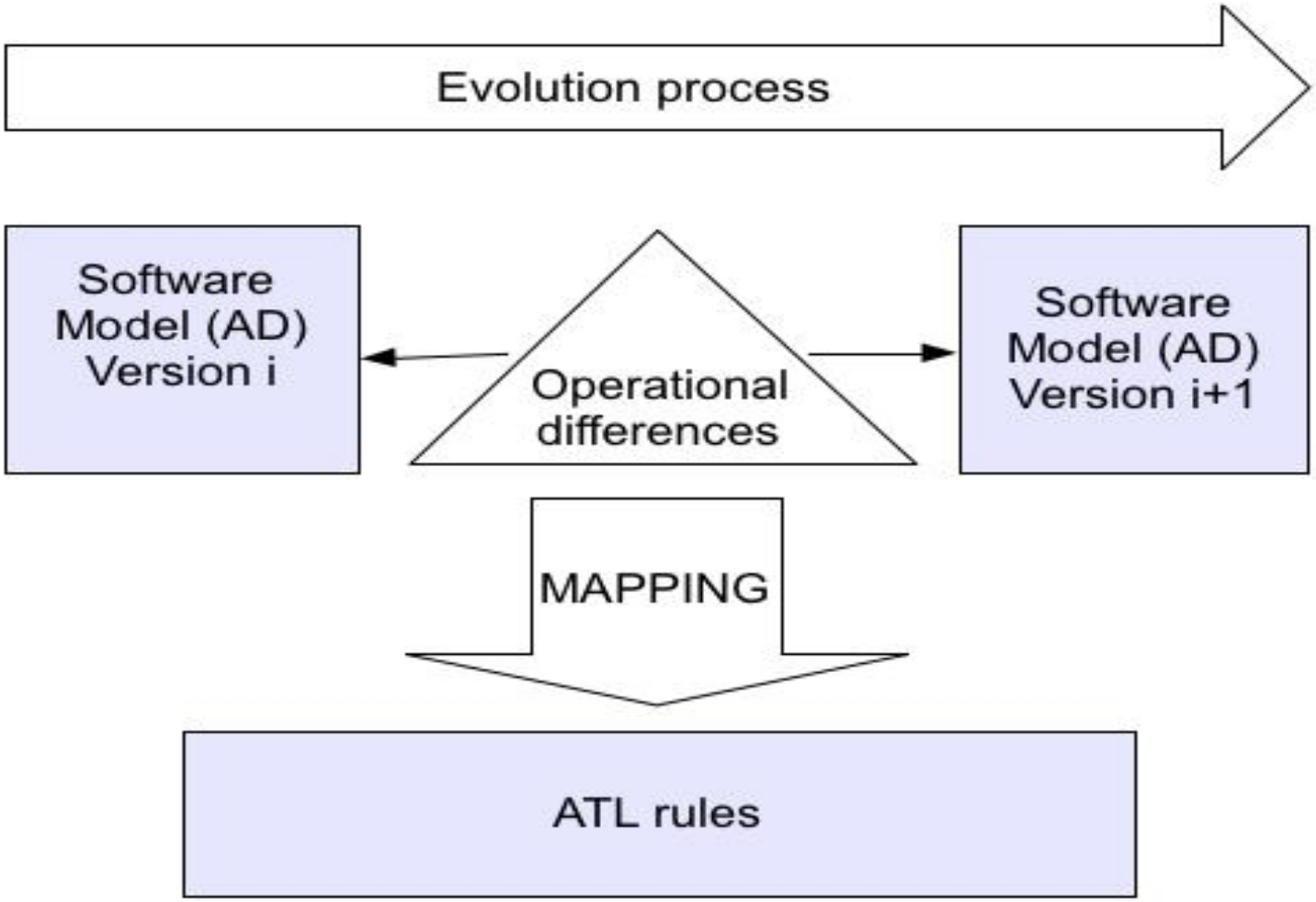
Motivation


Software System (version. i)
Model repository
A group of tools
Software System (version. i+1)
Collaborative modeling
Calculating differences
Representing differences
Visualizing differences
Merging (models with) differences
Analyzing differences

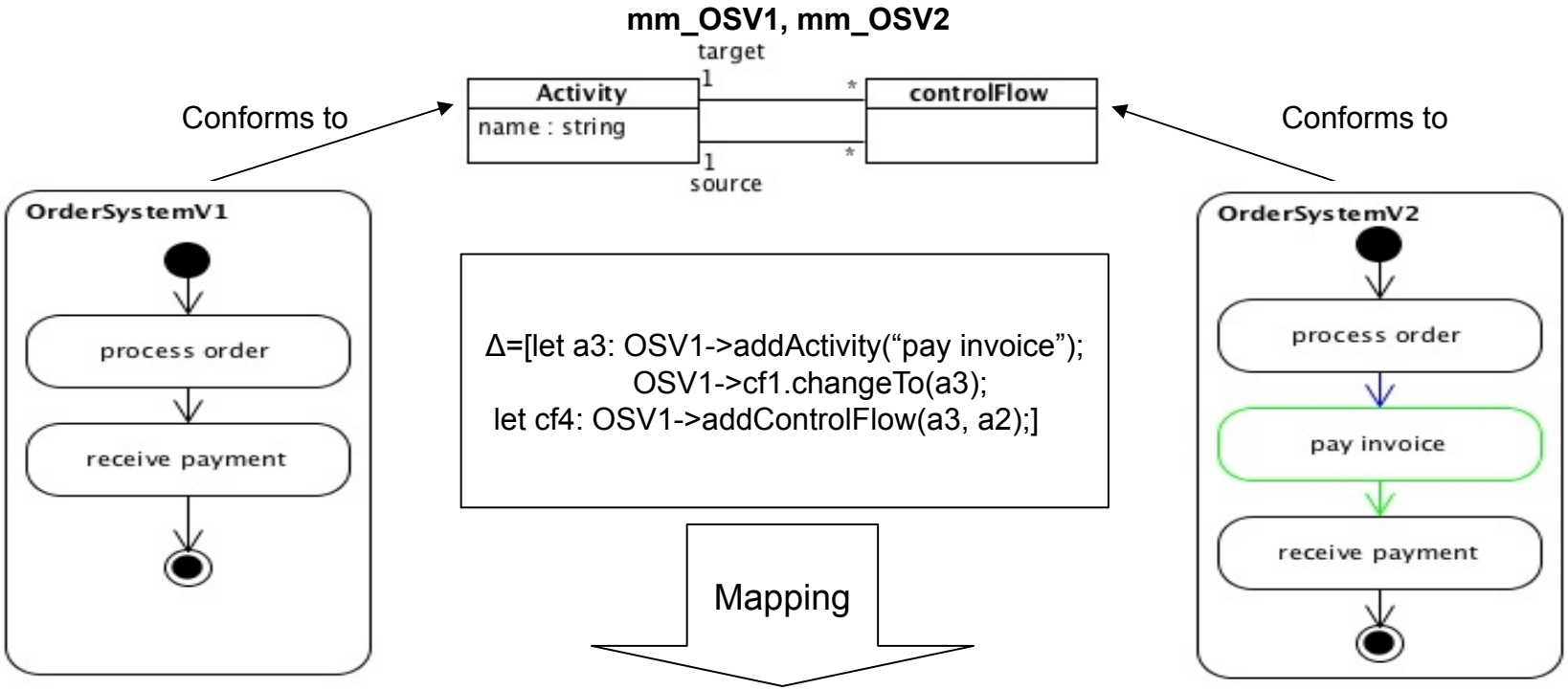
State of the Art

- **There is need for more sophisticated approach** which addressed to represent model differences
- **Integrability of an approach** brings huge amount of possiblities
- **Tool support** is still in its infancy
- **Mixture of design and code levels.** Recently, round-trip engineering raises the challenge of synchronizing models and code

Proposed approach



Rule representation of operations

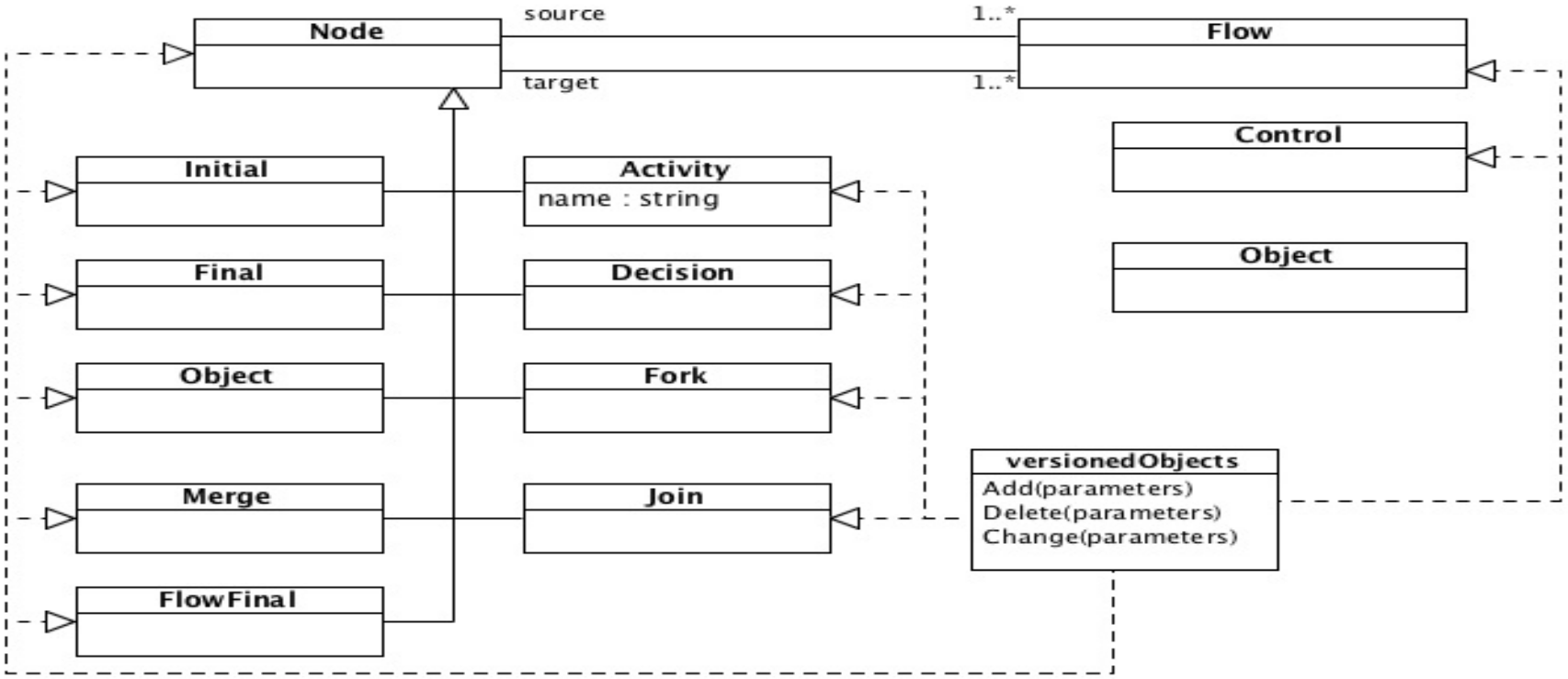



```
module in2out;
create OUT:OrderSystemV2 from IN:OrderSystemV1;
uses string, integer;
rule addActivity(name : String){
    to
        b : mm_OSV2!Activity(
            b.name <- name
        )
}
```

```
rule changeTarget(target : String){
    from a : mm_OSV1!controlFlow
    to   b : mm_OSV2!controlFlow(
        b.source <- a.source,
        b.target <- target
    )
}
```

```
rule addControlFlow(source, target : String){
    to b : mm_OSV2!controlFlow(
        b.source <- source,
        b.target <- target
    )
}
```

Meta-model for UML Activity diagram



DELTA △	Node									Flow			
	Activity (name)	Start	End	Decision	Object	Fork	Merge	Join	FlowFinal	Control		Object	
										source	target	source	target
Add	■									■	■		
Change											■		
Delete													

Benefits

- helps to make decision
- focus on the problem with polymetric view
- structural and behavioural representation
- detect and resolve of conflicts
- share model artefacts among the team members
- speeds up development process, and traces evolution process
- store differences instead of complete models

Thank you for attention!