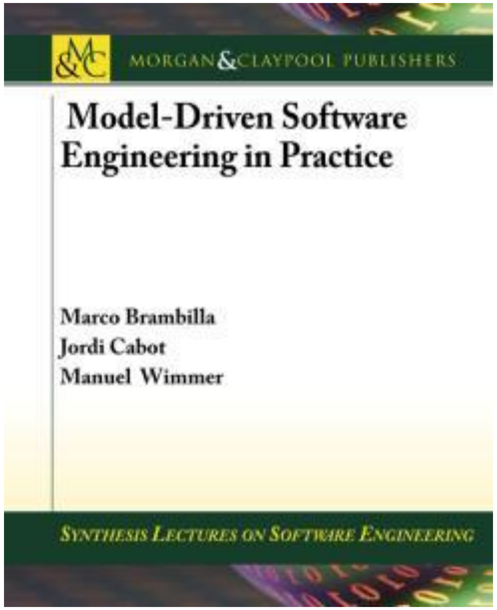
MORGAN & CLAYPOOL PUBLISHERS

Chapter #3

MDSE USE CASES

Teaching material for the book
Model-Driven Software Engineering in Practice
by Marco Brambilla, Jordi Cabot, Manuel Wimmer.
Morgan & Claypool, USA, 2012.



MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

MDSE GOES FAR BEYOND CODE-GENERATION

Morgan & Claypool Publishers

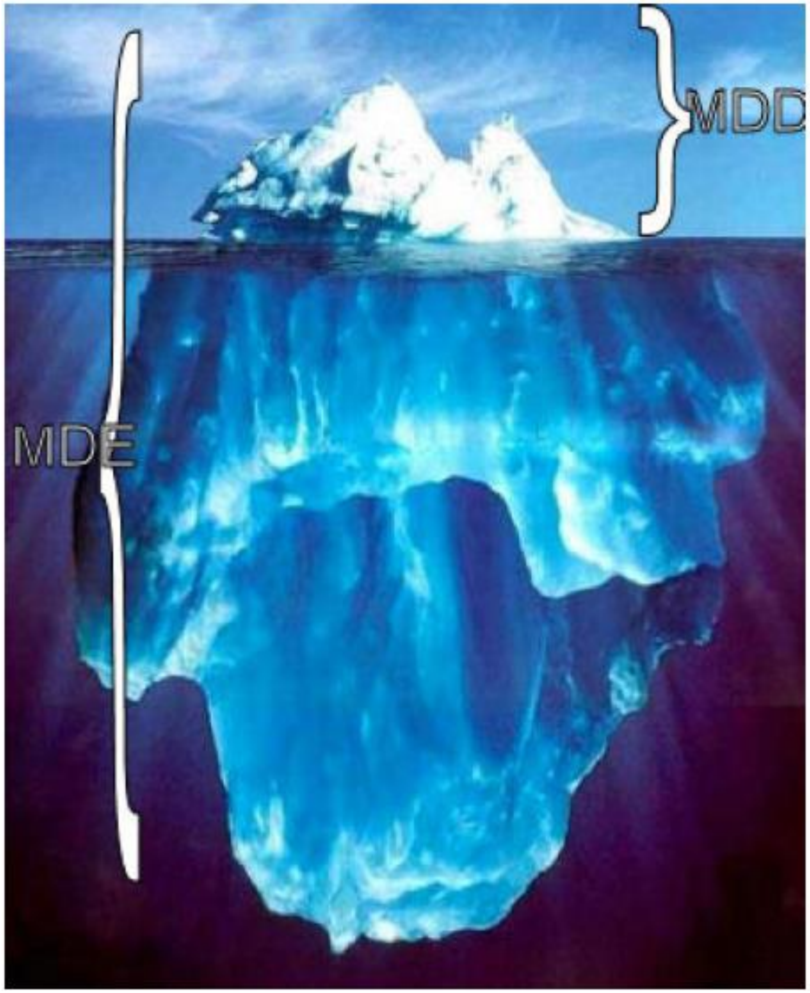
Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

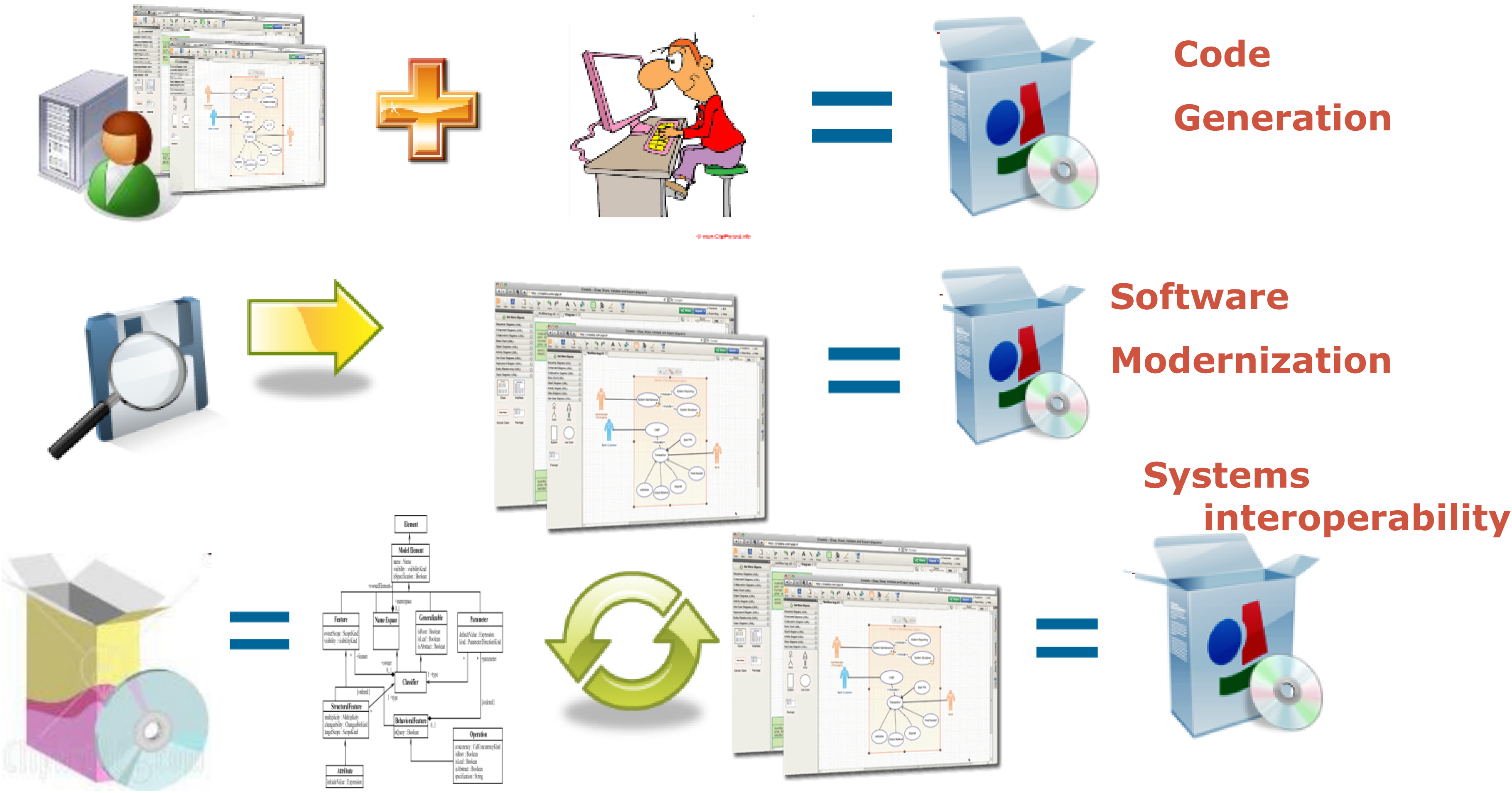
Synthesis Lectures on Software Engineering

MDSE has many applications

- MDD is just the tip of the iceberg
  - And MDA a specific “realization” of MDD when using OMG standards



Three killer MDSE applications



Code Generation

Software Modernization

Systems interoperability

USE CASE1 – MODEL DRIVEN DEVEOPMENT

MORGAN & CLAYPOOL PUBLISHERS

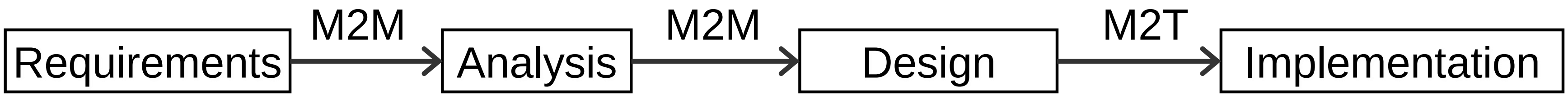
Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

MDD contribution: Communication

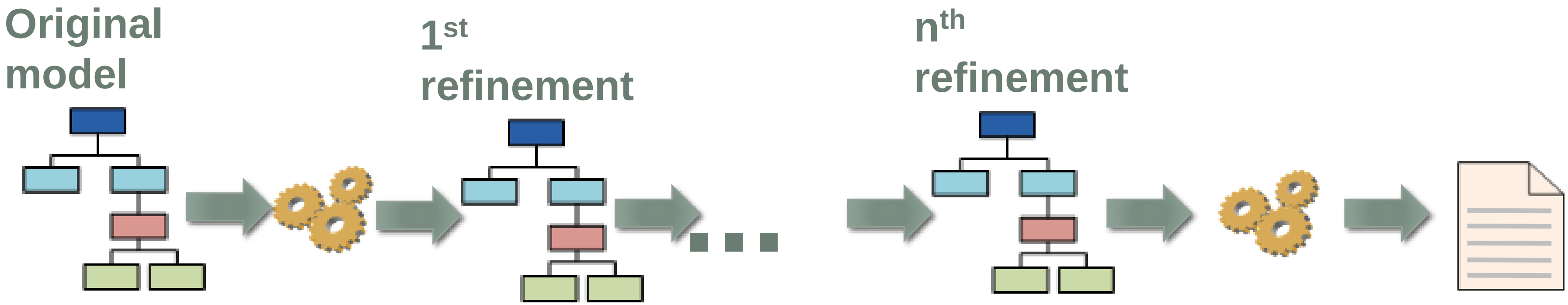
- Models capture and organize the understanding of the system within a group of people
- Models as *lingua franca* between actors from business and IT divisions



Requirements \xrightarrow{M2M} Analysis \xrightarrow{M2M} Design \xrightarrow{M2T} Implementation

MDD contribution: Productivity

- MDD (semi)automates software development
- In MDD, software is derived through a series of model-to-model transformations (possibly) ending with a model-to-text transformations that produces the final code



Original model

1st refinement

...

nth refinement

Model-to-model Transformation

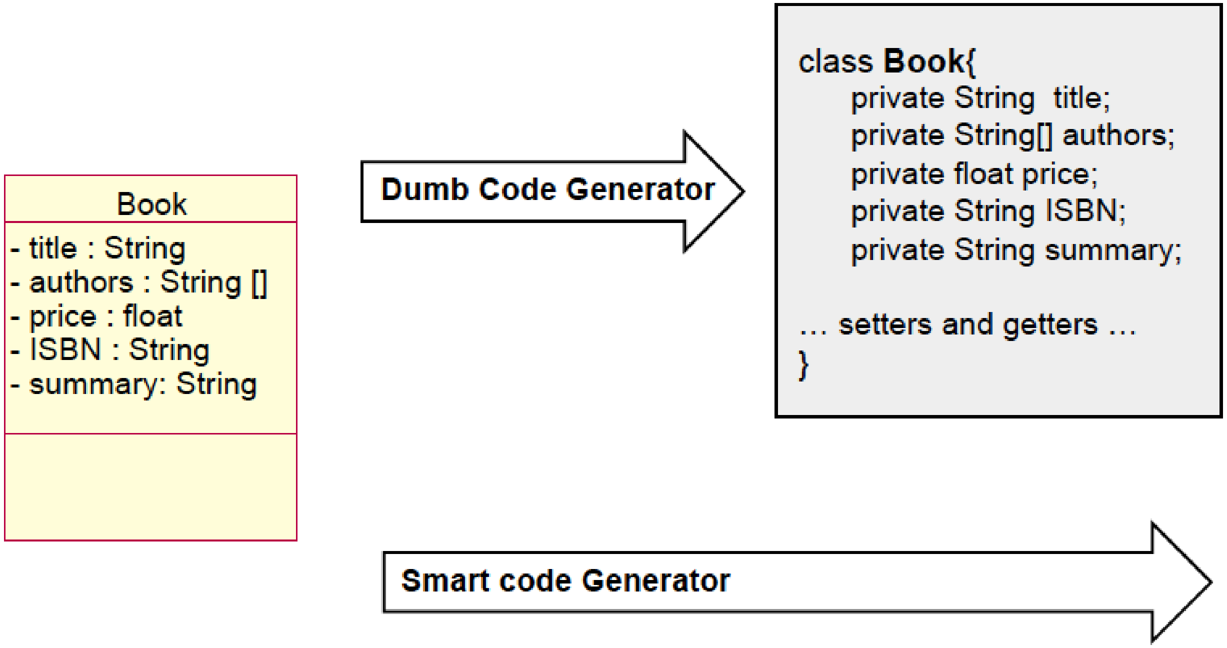
Model-to-text Transformation

Executable models

- An executable model is a model complete enough to be executable
- From a theoretical point of view, a model is executable when **its operational semantics are fully specified**
- In practice, the executability of a model may depend on the adopted execution engine
  - models which are not entirely specified but that can be executed by some advanced tools that are able to fill the gaps
  - Completely formalized models that cannot be executed because an appropriate execution engine is missing.

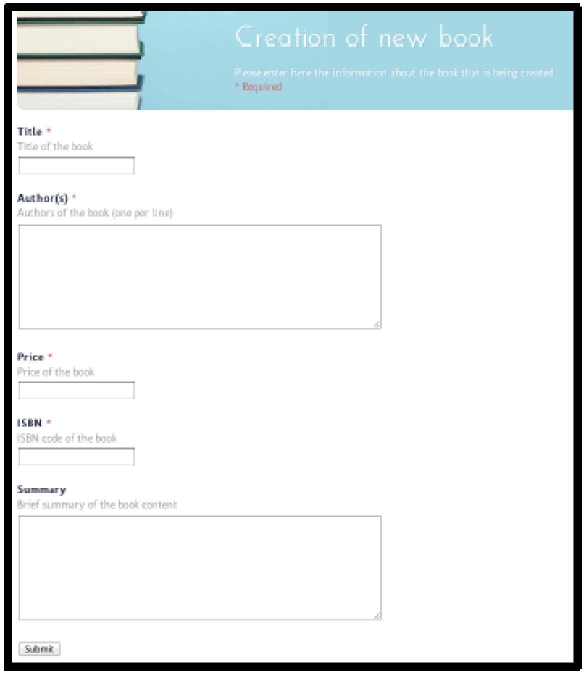
Smart vs dumb execution engines

- CRUD operation typically account for 80% of the overall software functionality
- Huge spared effort through simple generation rules



```
class Book{
     private String  title;
     private String[] authors;
     private float price;
     private String ISBN;
     private String summary;

… setters and getters …
}
```



Creation of new book

Please enter here the information about the book that is being created

* Required

Title *
Title of the book

Author(s) *
Authors of the book (one per line)

Price *
Price of the book

ISBN *
ISBN code of the book

Summary
Brief summary of the book content

Executable models

- Most popular: Executable UML models
- Executable UML development method (xUML) initially proposed by Steve Mellor
- Based on an action language (kind of imperative pseudocode)

- Current standards
  - Foundational Subset for Executable UML Models (fUML)
  - Action language is the Action Language for fUML (Alf)
    - basically a textual notation for UML behaviors that can be attached to a UML model

Executable models: 2 main approaches

- **Code generation**: generating running code from a higher level model in order to create a working application
  - by means of a rule-based template engine
  - common IDE tools can be used to render the source code produced

- **Model interpretation**: interpreting the models and making them run

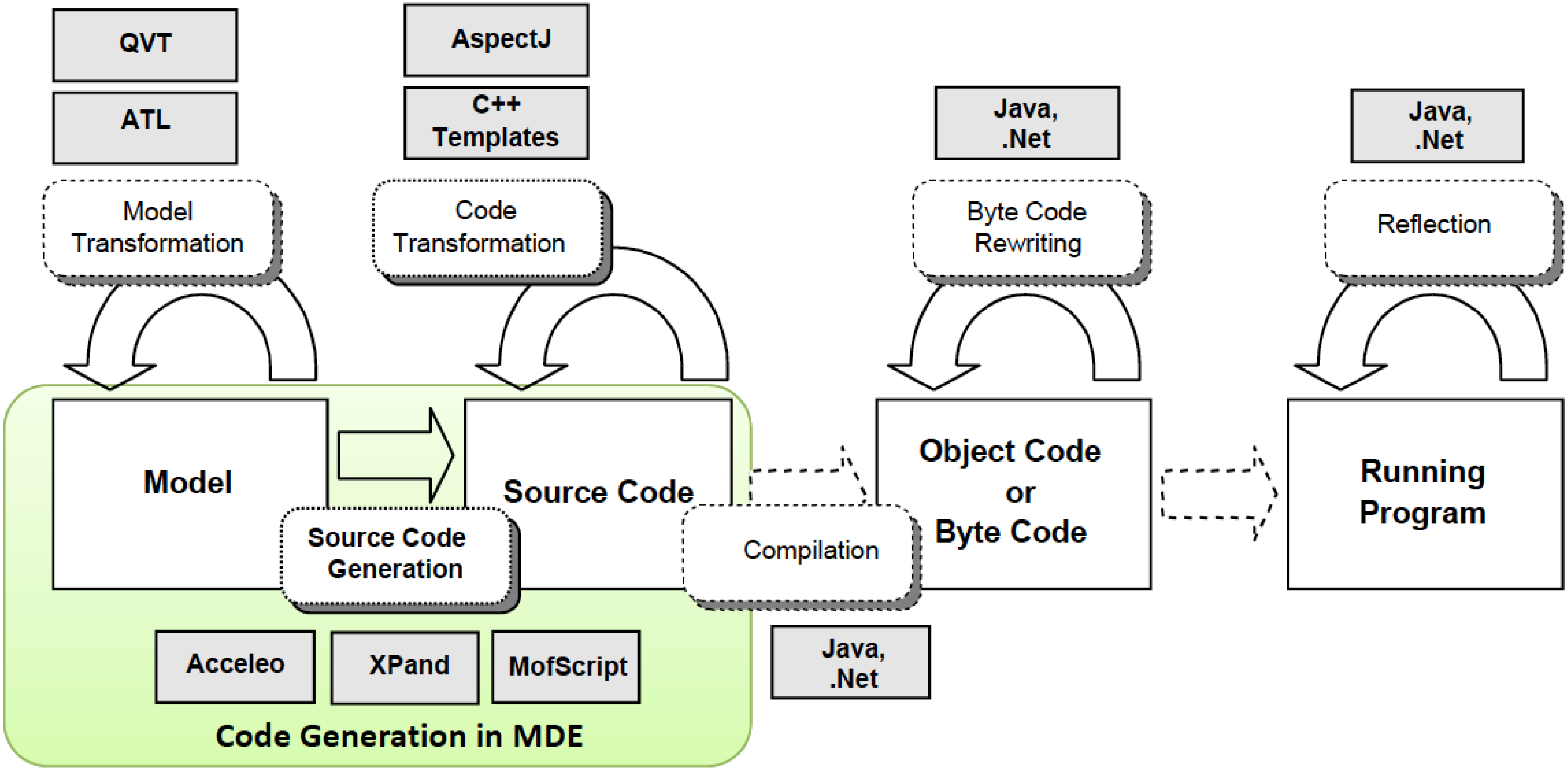
- Non-empty intersection between the two options

Code Generation

- Goal: generating running code from higher level models
  - Like compilers producing executable binary files from source code
  - Also known as model compilers

- Once the source code is generated state-of-the-art IDEs can be used to manipulate the code

Code Generation: Scope



Code Generation: Benefits

- Intellectual property
- Separation of modeling and execution
- Multi-platform generation
- Generators simpler than interpreters
- Reuse of existing artefacts
- Adaptation to enterprise policies
- Better performances

Code Generation: Partial Generation

- Input models are not complete & code generator is not smart enough to derive or guess the missing information
- Programmers will need to complete the code manually
- **Caution!** Breaking the generation cycle is dangerous

Solutions:

- Defining protected areas in the code, which are the ones to be manually edited by the developer
- Using round-trip engineering tools (not many available)
- Better to do complete generation of parts of the system instead of partial generation of the full system

Code Generation: Turing test

- A human judge examines the code generated by one programmer and one code-generation tool for the same formal specification. If the judge cannot reliably tell the tool from the human, the tool is said to have passed the test

Model interpretation

- A generic engine parses and executes the model on-the-fly using an interpretation approach

- Benefits
  - Faster changes & Transparent (re)deployment
  - Better portability (if the vendor supports several platforms)
  - The model is the code. Easier model debugging
  - No deployment
  - Updates of the model at runtime
  - Higher level abstraction of the system (implemented by the interpreter)
  - Updates in the interpreter may result in automatic improvements of your software

- Danger of becoming dependent of the application vendor. Limited influence in the –ities of the SW

Generation and interpretation

- Can be used together in the same process
  - Interpretation at early prototyping / debugging time
  - Generation for production and deployment
- Hybrid solutions are possible:
  - Model interpretation based on internal code generation implementation
  - Code generation that relies on predefined, configurable components / framework at runtime. The generated code is e.g., XML descriptor / configurations of the components

USE CASE2 – SYSTEMS INTEROPERABILITY

MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

Interoperability

- Ability of two or more systems to exchange information (IEEE)

- Needed for collaborative work (e.g. using different tools), tool and language evolution, system integration…

- Interoperability must be done at the syntactic and semantic levels

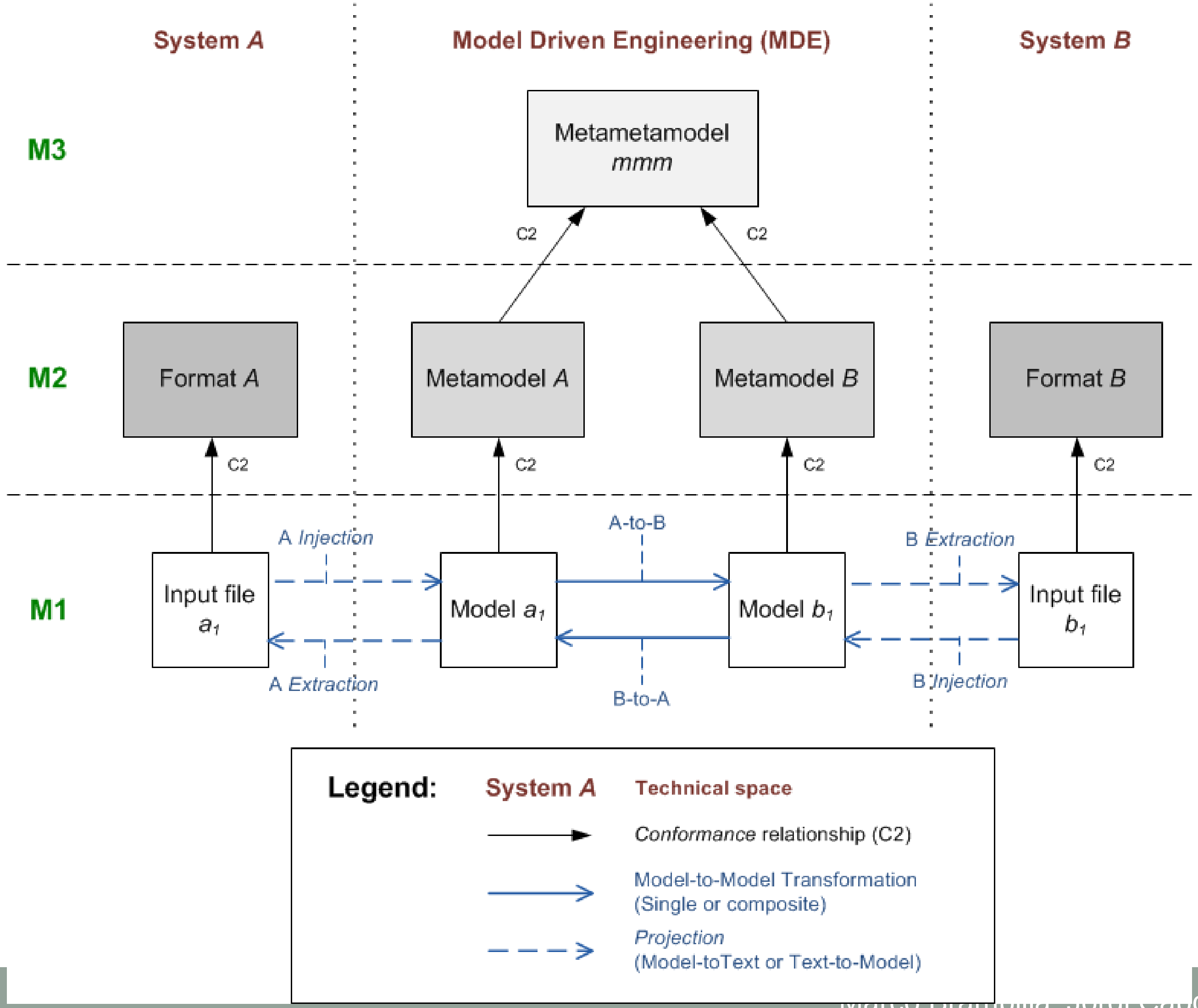
Model-Driven Interoperability

- MDSE techniques to bridge the interoperability gap

- The metamodels (i.e. “schemas”) of the two systems are made explicit and aligned

- Transformations follow the alignment to move information
  - Injectors (text-to-model) represent system A data as a model (syntactic transformation)
  - M2M transformation adapts the data to system B metamodel (semantic transformation)
  - Extractors (model-to-text) generate the final System B output data (syntactic transformation).

MDI: Global schema



USE CASE3 – MODEL DRIVEN REVERSE ENGINEERING

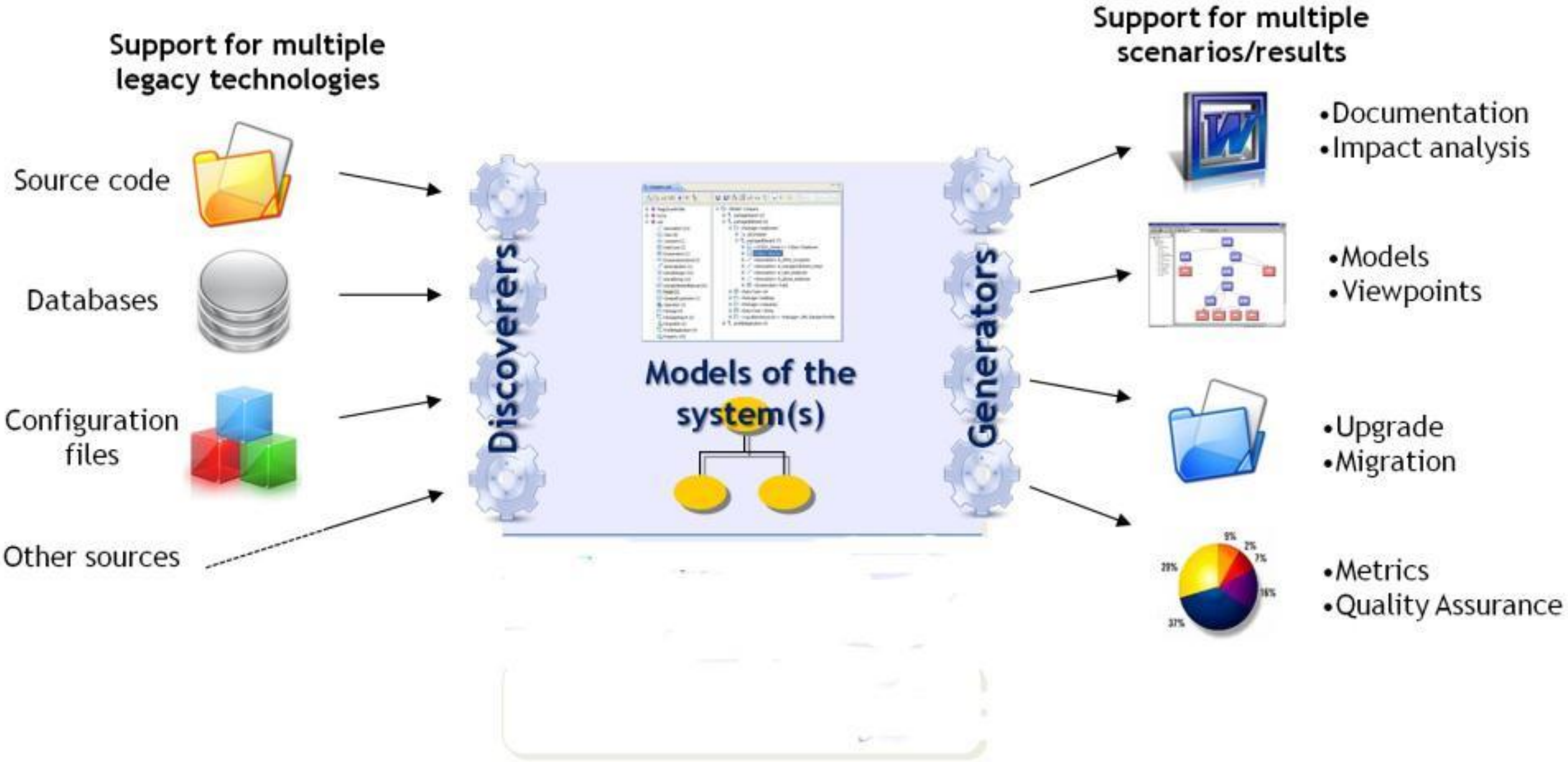
MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

Need for reverse engineering



Support for multiple legacy technologies

Source code

Databases

Configuration files

Other sources

Discoverers

Models of the system(s)

Generators

Support for multiple scenarios/results

- Documentation
- Impact analysis

- Models
- Viewpoints

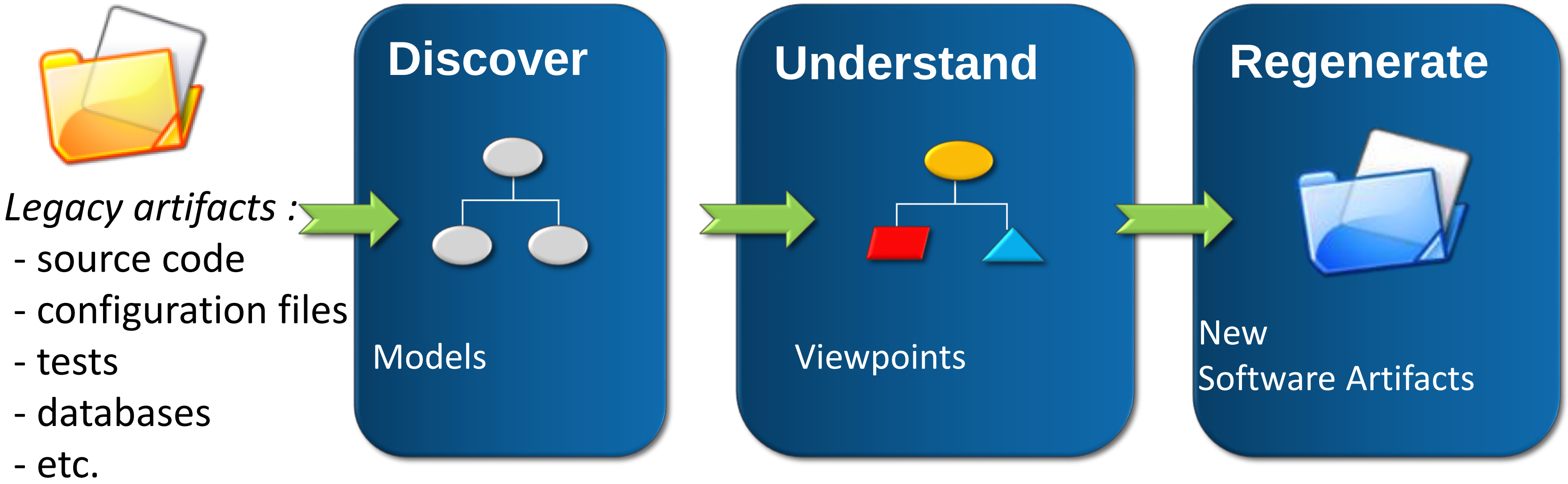
- Upgrade
- Migration

- Metrics
- Quality Assurance

Model-driven reverse engineering

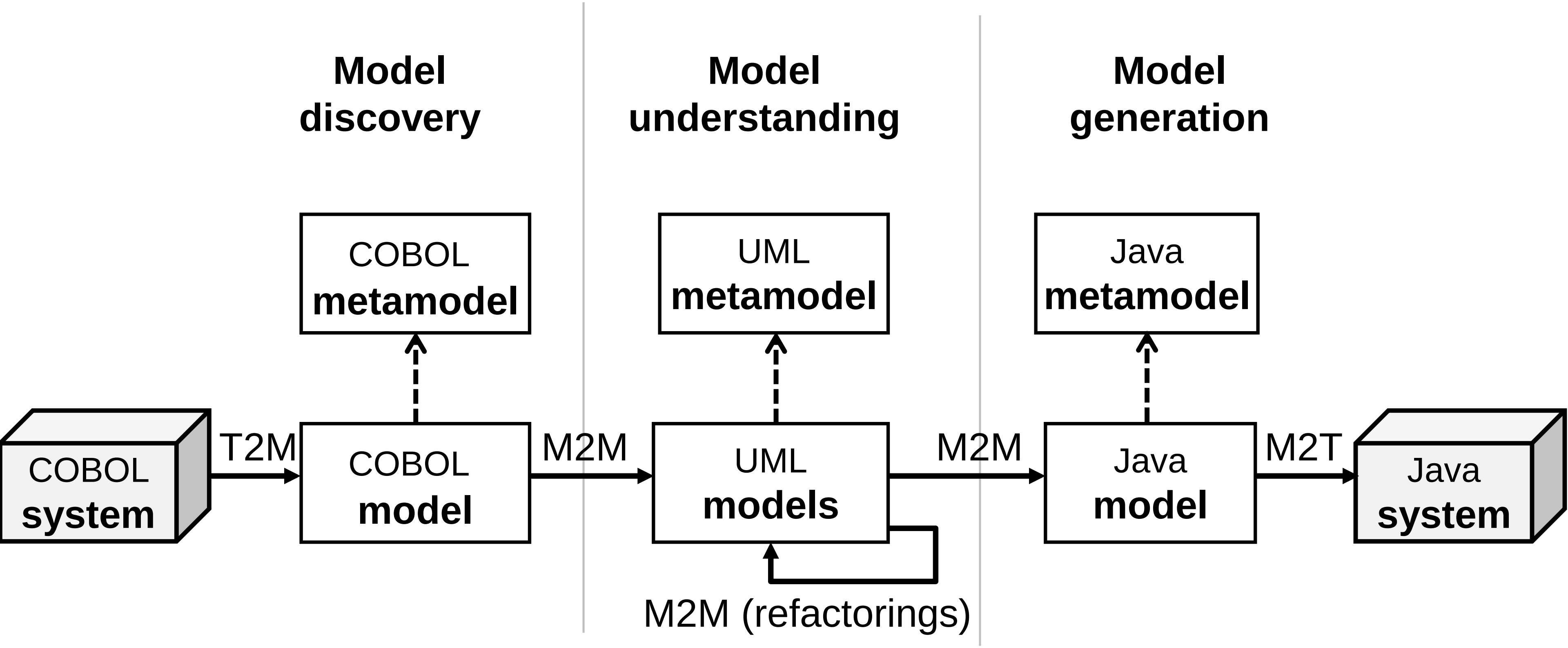
- **Why?** Models provide an homogeneous and interrelated representation of all legacy components.

  No information loss: initial models have a 1:1 correspondance with the code



Legacy artifacts :
- source code
- configuration files
- tests
- databases
- etc.

Model-Driven Interoperability: Example



Model discovery

Model understanding

Model generation

COBOL system

T2M

COBOL metamodel

COBOL model

M2M

UML metamodel

UML models

M2M (refactorings)

M2M

Java metamodel

Java model

M2T

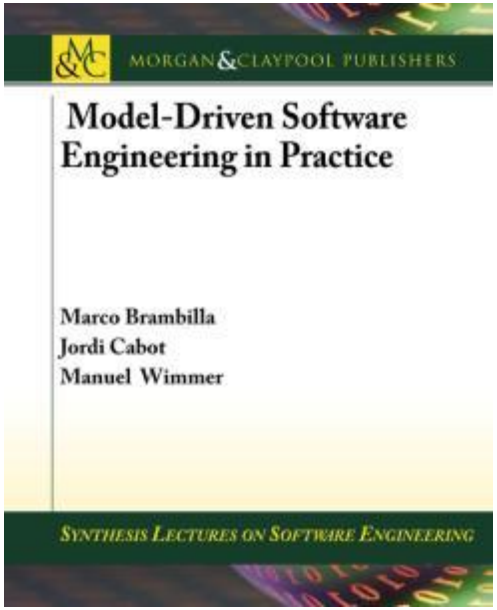
Java system

MORGAN & CLAYPOOL PUBLISHERS

MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE

Marco Brambilla,
Jordi Cabot,
Manuel Wimmer.
Morgan & Claypool, USA, 2012.

www.mdse-book.com
www.morganclaypool.com
or buy it at: www.amazon.com



MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING