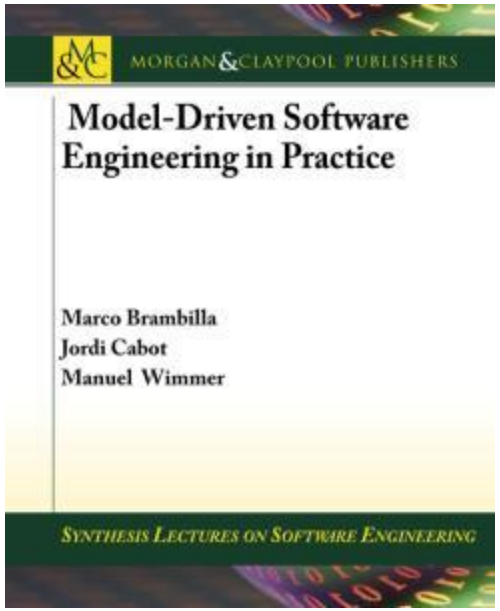
MORGAN & CLAYPOOL PUBLISHERS

Chapter #3

MDSE CASOS DE USO(USE CASES)

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



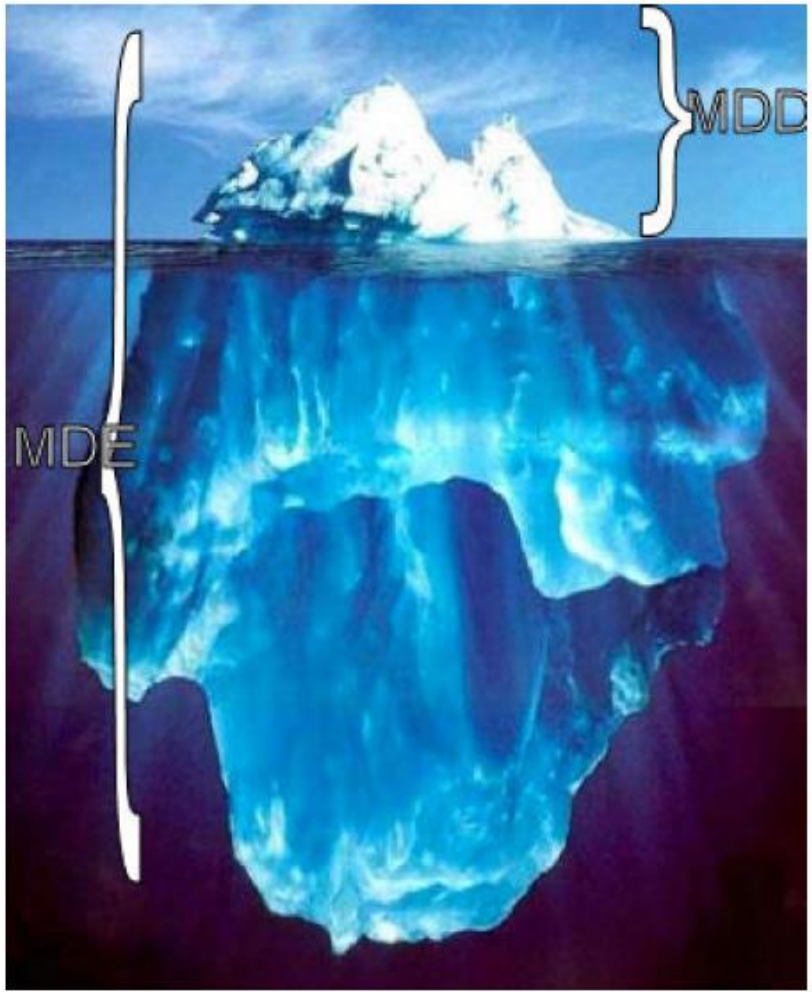
MDSE VAI ALÉM DA SIMPLES GERAÇÃO DE CÓDIGO

MDSE possui muitas aplicações

A primeira e mais bem conhecida aplicação do MDSE é o MDD.

- No contexto de MDSE, MDD é como se fosse a ponta do iceberg.
- A idéia do MDD é utilizar técnicas de desenvolvimento dirigido por modelos para automatizar o máximos possível o processo do desenvolvimento de software.

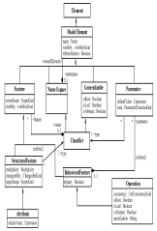
- A MDA é uma implementação do conceito de MDD utilizando os padrões da OMG.



Foco do capítulo

= **Geração de Código/MDD**

=



= **Interoperabilidade entre os sistemas**

=

Evolução do Software/(Re)Engenharia

USE CASE1 – MDD

Contribuições do MDD: Comunicação

- Modelos captam melhor e organizam o entendimento do problema.
  - Foco em tarefas conceituais;
  - Modelos facilitam a comunicação entre os diversos interessados no sistema(especialistas no domínio e especialistas na solução).

Contribuições do MDD: Comunicação

- São utilizados em diversas outras engenharias.

Como ficará a ocupação urbana da capital Xangai em 2020?

Pedro tem dois irmãos

Daqui a cinco anos, o mais novo terá a metade da idade de Pedro

Daqui a dez anos, o mais velho terá o triplo da idade do mais novo

Quantos anos tem Pedro e os irmãos?

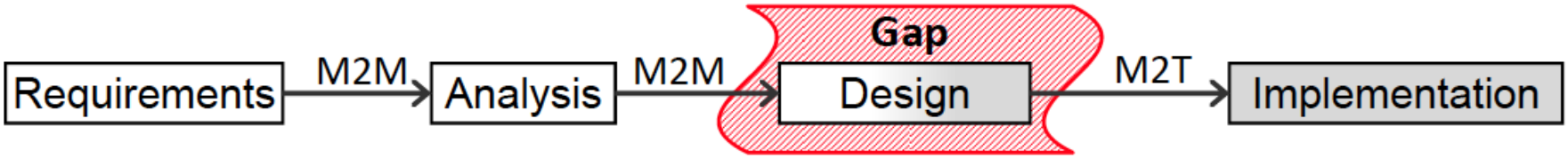
Idade de Pedro hoje = x
Idade do irmão mais novo hoje = y
Idade do irmão mais velho hoje = z
Sistema

$$\begin{cases} y+5 = (x+5)/2 \\ z+10 = (y+10)*3 \end{cases}$$

Contribuições do MDD: Comunicação

- Normalmente, em uma abordagem MDSE, a aplicação é obtida por meio uma ou mais transformações de modelos.

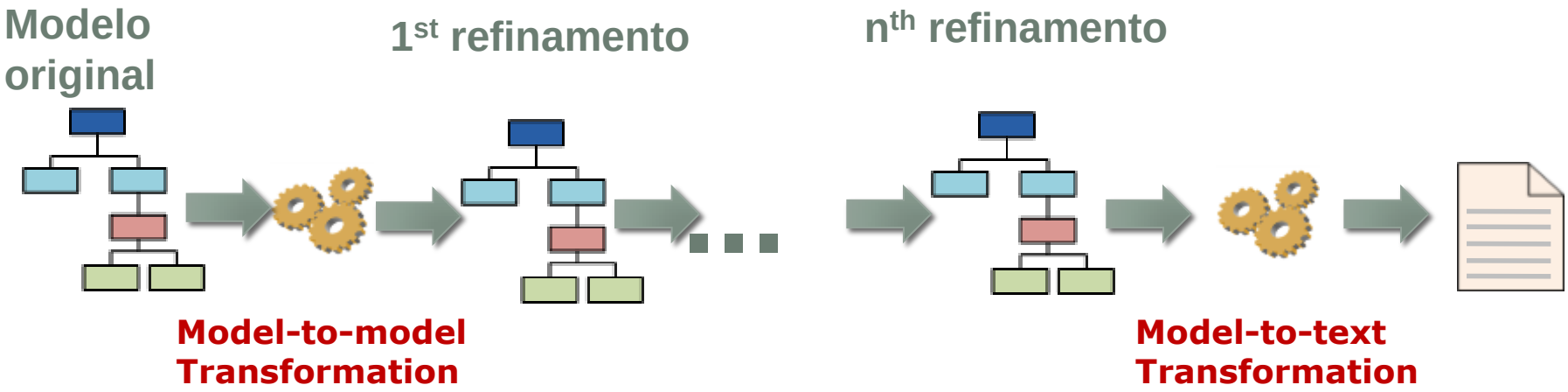
- Um processo Baseado em MDSE:



Contribuições do MDD: Produtividade

- MDD (Semi)automatiza tarefas do desenvolvimento de software;
- Um elemento conceitual pode corresponder a vários elementos de código;

- EM MDD, o software é derivado de uma série de tranformações model-to-model (possivelmente) e por fim é feita uma transformação model-to-text para produzir o código



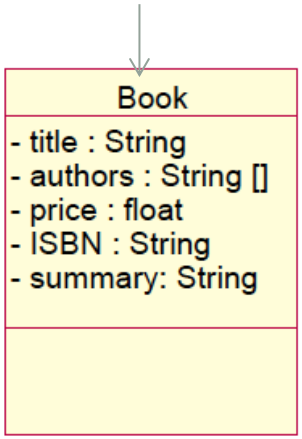
Modelos Executáveis

- Um modelo executável é um modelo completo o suficiente para ser executável, i.e, um modelo é executável quando sua semântica operacional é completamente especificada;

- Na prática a executabilidade de um modelo pode depender do mecanismo de execução adotado;
  - É possível encontrar modelos que não são inteiramente especificados, mas que podem ser executados por algumas ferramentas que o reconhecem.

Mecanismos de geração espertos vs burros

Definição apenas do modelo estático da classe



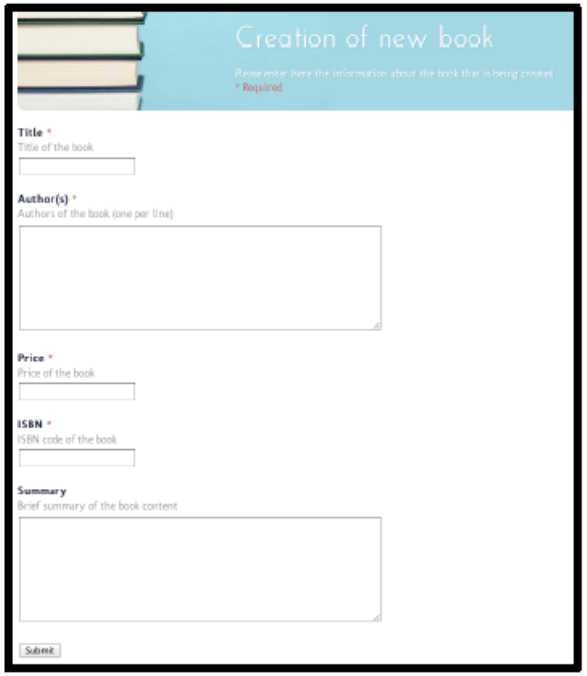
Dumb Code Generator

Geração apenas do esqueleto da classe

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

… setters and getters …
}
```

Smart code Generator



Um gerador esperto pode inferir que para cada classe será gerado um CRUD

Modelos executáveis

- Mais populares: modelos executáveis UML;

- Método UML executável de desenvolvimento (xUML) inicialmente proposto por Steve Mellor;

Modelos executáveis: 2 principais abordagens

- **Geração de código**: gera código a partir de um modelo de alto nível para criar uma aplicação funcional;

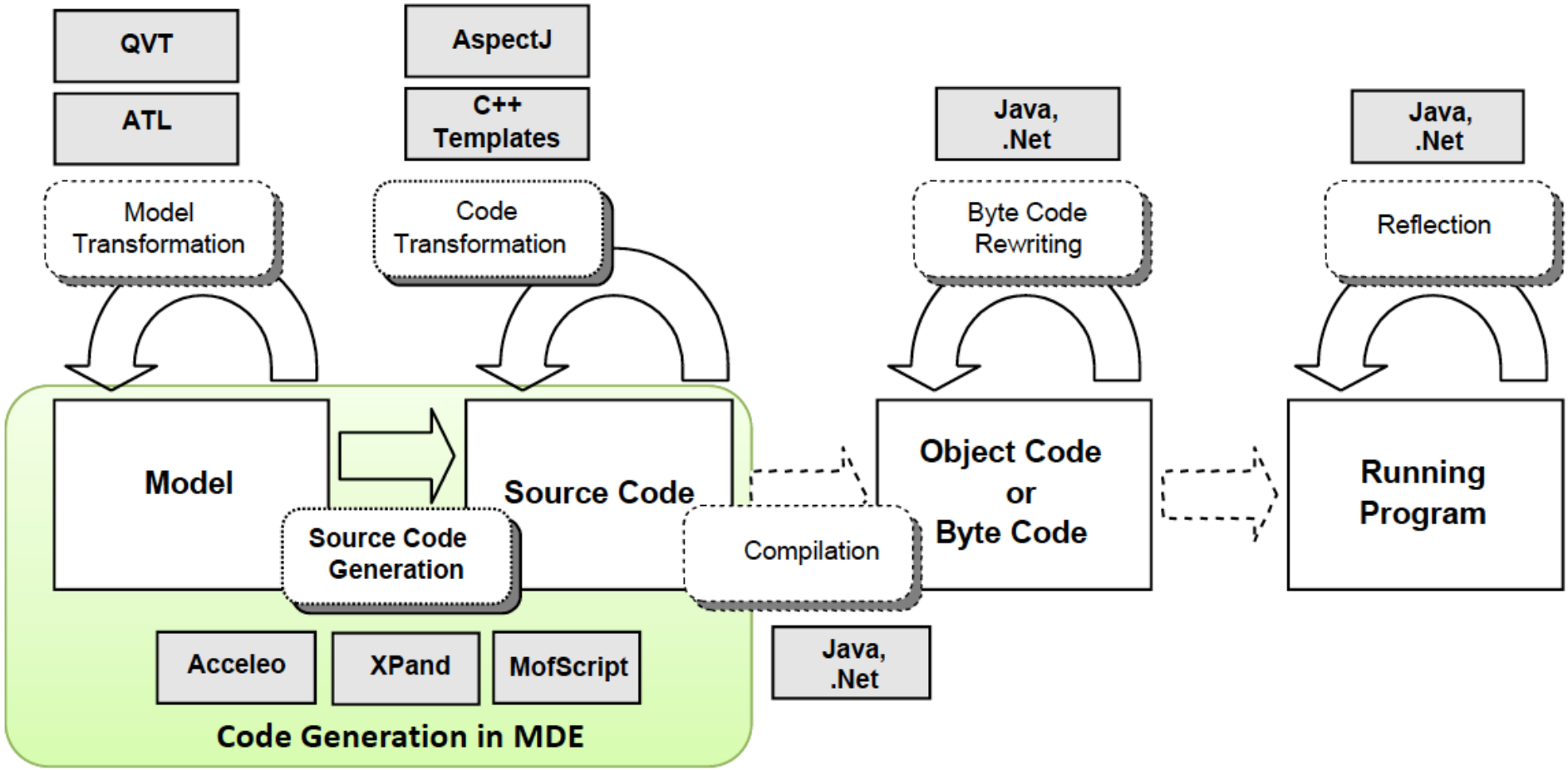
- **Interpretação de modelos**: interpreta os modelos e os torna executáveis;

Modelos Executáveis: geração de código

- Objetivo: gerar código funcional apartir de modelos alto nível.
  - Como compiladores produzem binários a partir de um código fonte;
  - Abordagem também conhecida como compiladores de modelos;

- Uma vez que o código é garado IDEs podem ser usados para manipular o código.

Escopo da geração de código



Benefícios da geração de código

- Protege a propriedade intelectual do modelador: modelos são reutilizados;
- Separação entre modelagem e execução: o executável pode ser gerado em uma linguagem padronizada;
- Geração Multi-platforma
- Geradores são mais simples que interpretadores:
- Reuso de artefatos existentes
- Melhor performance(compilação vs interpretação)

Geração de código:geração parcial

- As vezes os modelos não são completos e o gerador não é completo o suficiente para completar a informação perdida na abstração;
- Pode ser necessário que programadores completem manualmente o código;
- **Atenção!** Quebrar o ciclo de geração pode ser perigoso(inconsistencia)

Soluções:

- Definir áreas protegidas no código que serão manualmente editadas
- Utilizar engenharia de ida-e-volta(round-trip engineering) – ainda em pesquisa;
- É melhor completar a geração de partes do sistema que gerar o sistema em partes;

Geração de código: Teste de Turing

- Um avaliador humando examina o código escrito por um programador e um código gerado pelo gerador a partir de uma especificação formal. Se o avaliador não conseguir dizer precisamente qual o código feito pelo humano, o gerador passou no teste.

Interpretação de modelos

- Um mencanismo genérico faz um parser dos modelos e os executa como em uma abordagem de interpretação;
- Benefícios
  - Mudanças rápidas e (re)deployment transparente
  - Melhor portabilidade
  - O modelo é o código. Debugging a nível de modelos
  - Atualização de modelos em tempo de execução
  - Aplicação em alto nível de abstração(como em PaaS, o ambiente provê diversas outras funcionalidades)
  - Atualizações do interpretado podem resultado em melhorias automáticas do software.
- Perigoso ficar dependente do vendedor(como lock-in em PaaS)

Geração e interpretação

- Podem ser utilizadas juntas no mesmo processo
  - Interpretação em estágios iniciais de prototipação/debugging
  - Geração para produção e deployment
- Soluções Hibridas também são possíveis
  - Pensar no case de Java

Trabalhos correlatos disciplina

- **Lucrédio, D; A Model-Driven Software Reuse Approach, ICMC-USP, 2009.**

- **Cirilo, C, E;** Processo Dirigido a Modelos para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto, 2011.

- **CIRILO, CARLOS EDUARDO** ; PRADO, ANTONIO FRANCISCO DO ; SOUZA, WANDERLEY LOPES DE ; ZAINA, LUCIANA APARECIDA MARTINEZ . Experimentation of the Model Driven RichUbi Process in the Adaptive Rich Interfaces Development. In: 2011 25th Brazilian Symposium on Software Engineering (SBES), 2011, Sao Paulo. 2011 25th Brazilian Symposium on Software Engineering. v. 1. p. 182-191.

- Lucredio, D; Almeida, E, D; Fortes, R. P. M. An Investigation on the Impact of MDE on Software Reuse. SBCARS 2012.

USE CASE2 – INTEROPERABILIDADE DOS SISTEMAS

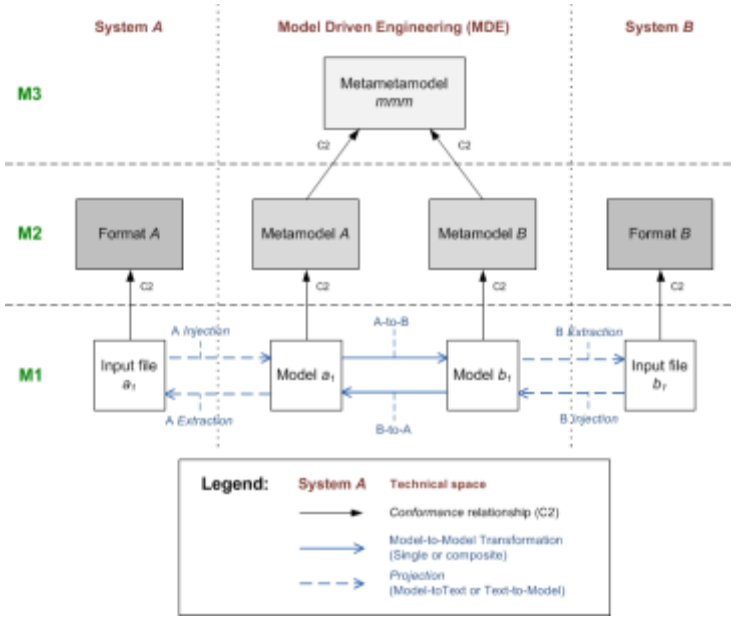
Interoperabilidade

- A habilidade de um ou mais sistemas trocarem informações(IEEE)
- Interoperabilidade é requerida em diferentes cenários: Ex. necessidade de trabalho colaborativo(usando diferente ferramentas, linguagens e integração de sistemas)
- Interoperability é ainda um desafio e deve ser obtida em níveis sintáticos e semânticos.
  - Niveis sintáticos: cada ferramenta/plataforma utiliza diferentes formatos sintáticos para armazenar;
  - Níveis semânticos: interpretar internamente as informações;

Model-Driven Interoperability

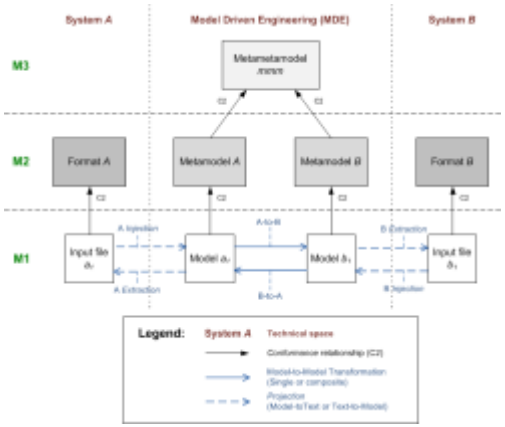
- Técnicas MDSE para interoperabilidade;
- Os metamodelos (i.e. “schemas”) dos dois sistemas são explicitos e alinhados;
- As Transformações seguem o alinhamento para mover as infeormações

MDI: Global schema



- Injectors(text-to-model) representam os dados do sistema A como um modelo(transformação sintática);
- Transformações M2M adaptma os dados do sistema B ao metamodelo(transformação semântica)
- Extractors (model-to-text) geram os dados de saida do sistema final(syntactic transformation).

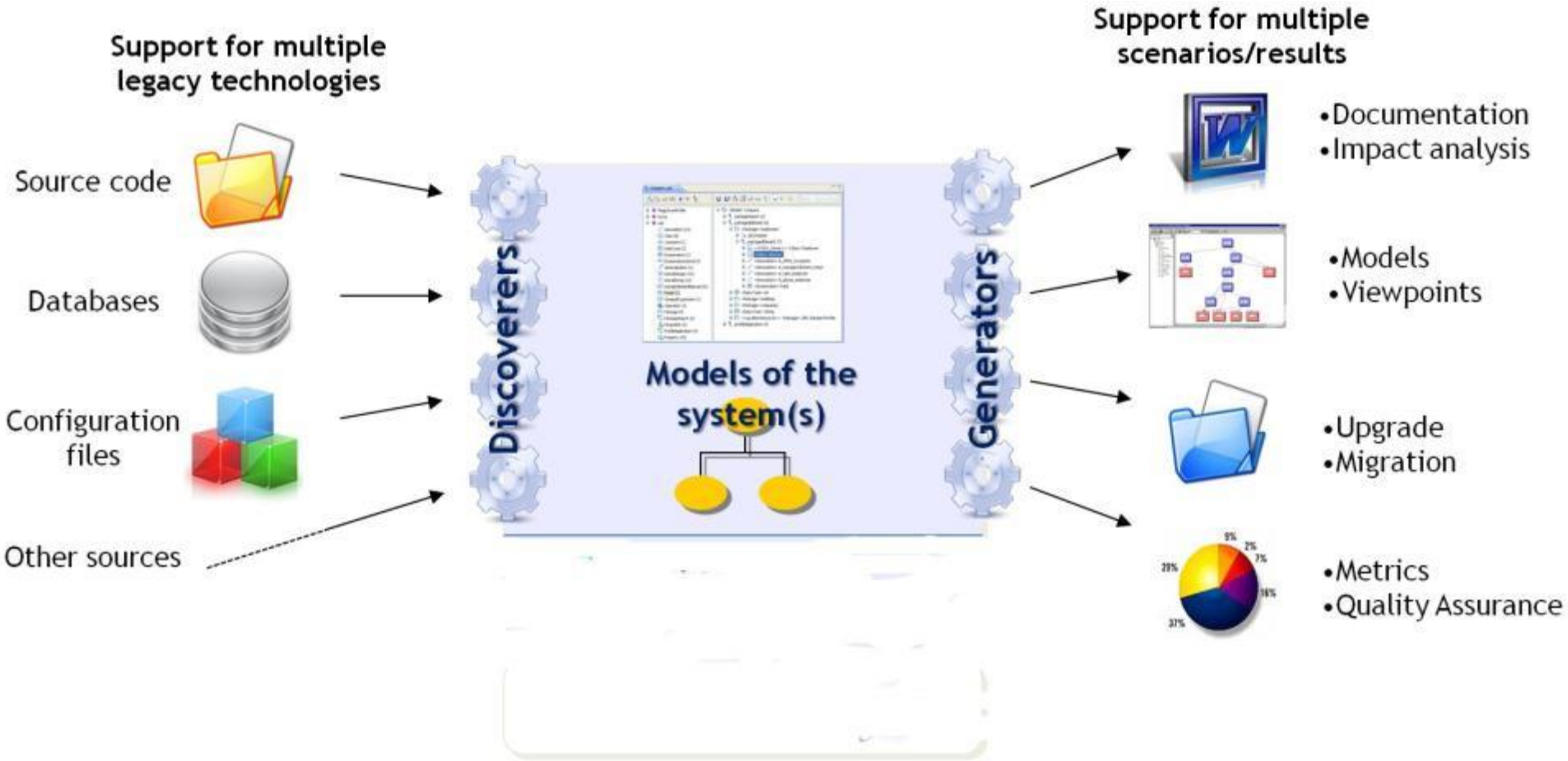
Model-Driven Interoperability



SHARMA, R.; SOOD, M. A Model Driven Approach to Cloud SaaS Interoperability. *International Journal of Computer Applications*, v. 30, n. 8, p. 1–8, 2011.

USE CASE3 – MODEL DRIVEN REVERSE ENGINEERING

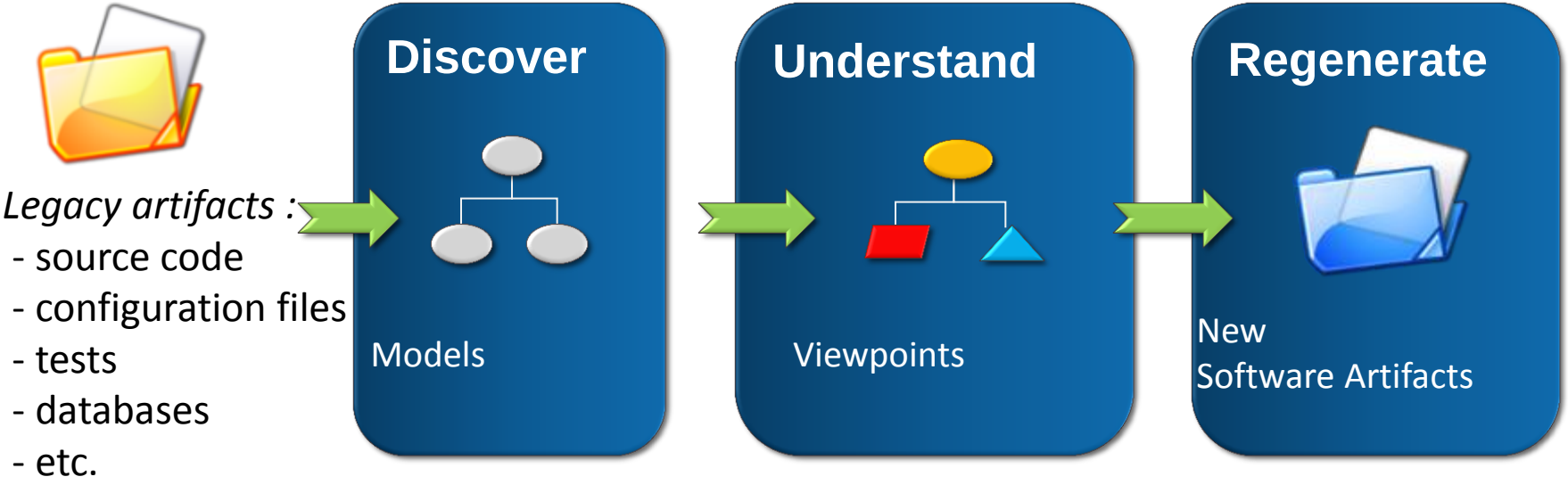
Necessidade de engenharia reversa



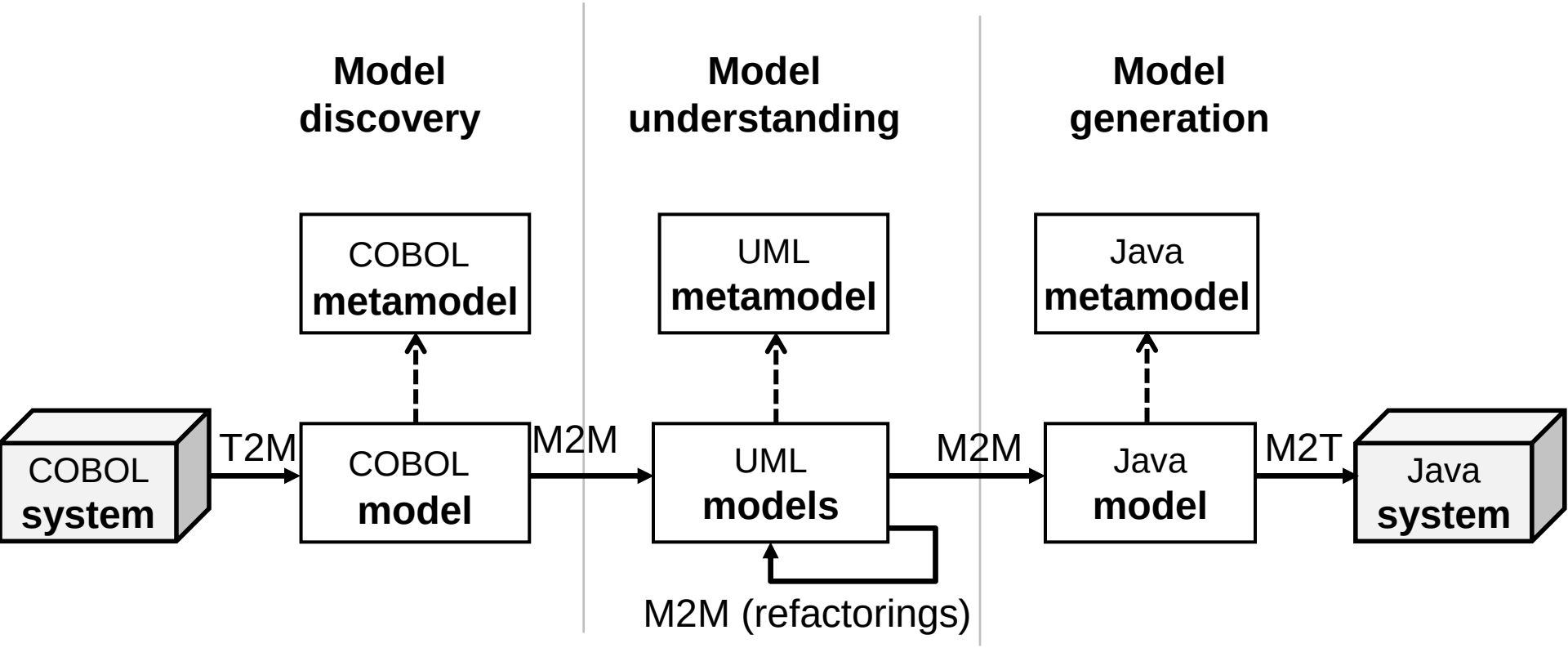
Model-driven reverse engineering

- Modelos apresentam uma representação homogenia dos componentes legados;
- 3 fases principais: discoberta, entendimento e regeração

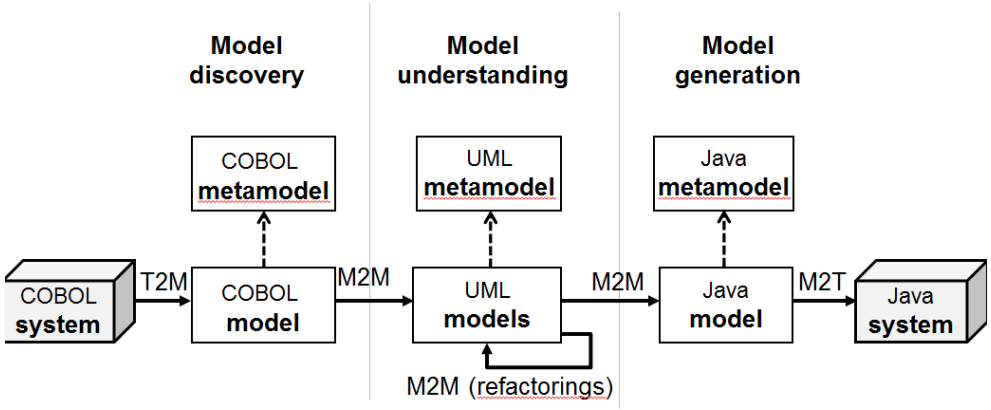
  Quando inclui a regeração -> reengenharia



Exemplo de uma Reengenharia do software



Exemplo de uma Reengenharia do software



PAPOTTI, P. E. ; PRADO, A. F. ; SOUZA, W. L. . Reducing Time and Effort in Legacy Systems Reengineering to MDD Using Metaprogramming. In: ACM Research in Applied Computation Symposium (ACM RACS 2012), 2012, San Antonio, TX, USA. Proceedings of the ACM Research in Applied Computation Symposium (ACM RACS 2012), 2012.

PAPOTTI, P. E. ; PRADO, A. F. ; SOUZA, W. L. ; CIRILO, C. E. ; PIRES, L. F. . A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation. In: International Conference on Advanced Information Systems Engineering (CAISE), 2013, Valência. Advanced Information Systems Engineering, 2013.

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