Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

TECHNISCHE UNIVERSITÄT DARMSTADT

Software Engineering in Industrial Practice
Model-driven development

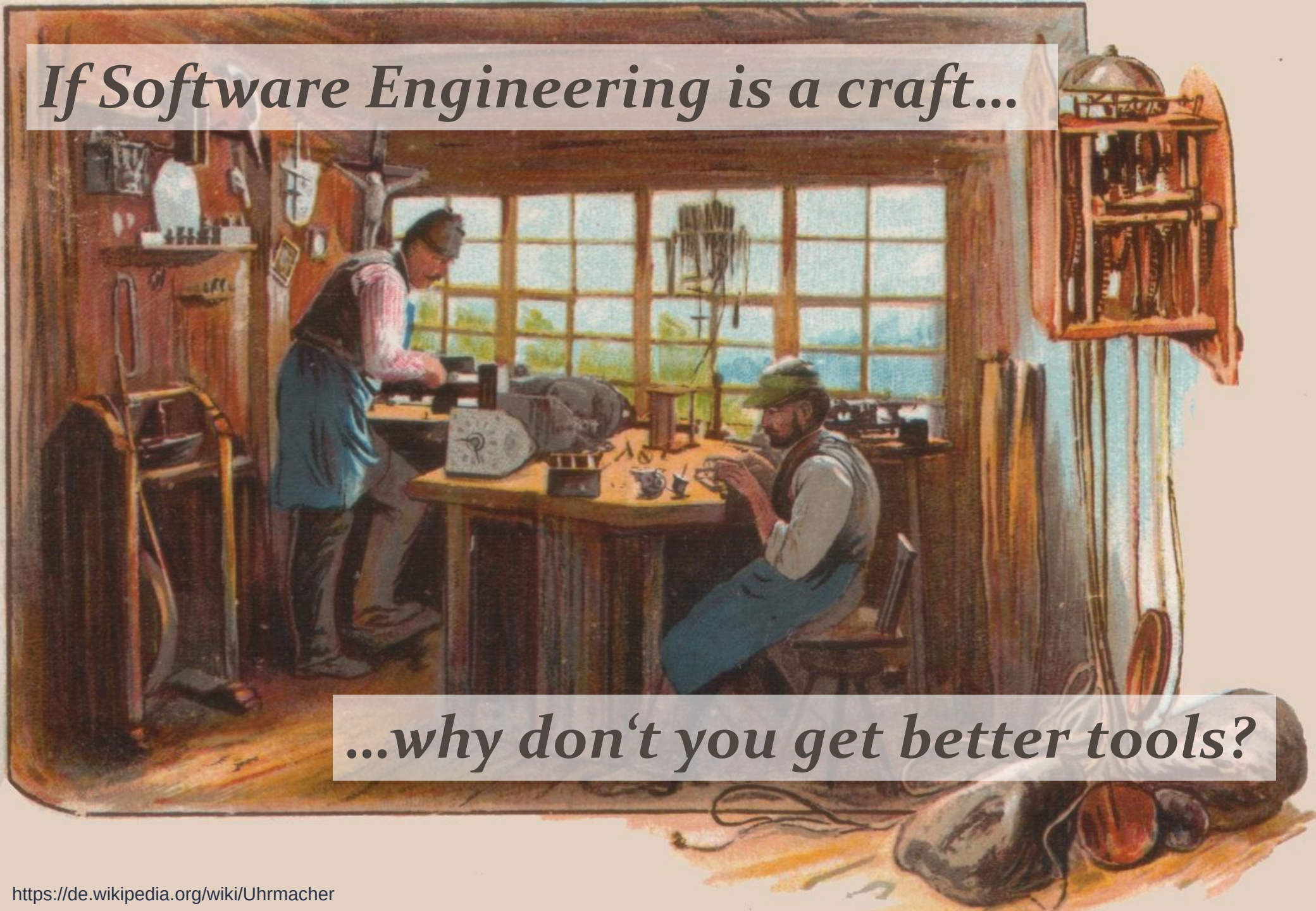
22nd January 2016

Dr. Martin Girschick
TU Darmstadt WS2015/2016

Questions/Comments?
http://www.yourpart.eu/p/seiip

Picture taken from http://notactuallybfg.deviantart.com/art/Low-Poly-Fantasy-Forest-Model-496769341

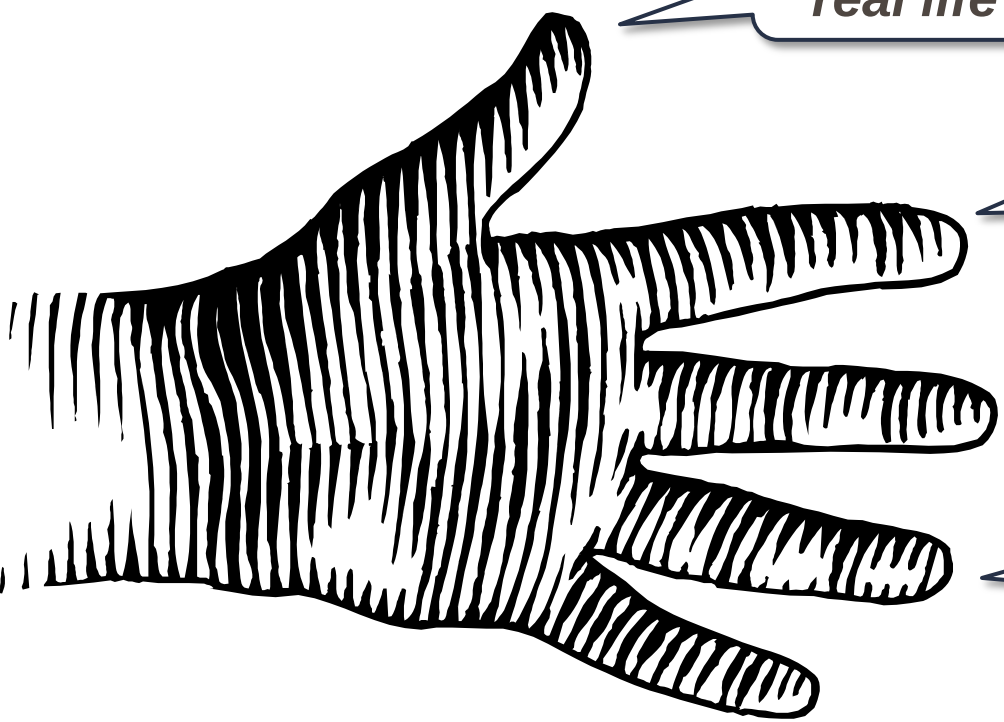
People matter, results count.



https://de.wikipedia.org/wiki/Uhrmacher

Questionnaire...

Five arguments against model driven development



Models are models, real life is different

No-one knows, how to do it (right)

Performance woes

All in and no way out

High effort with low return

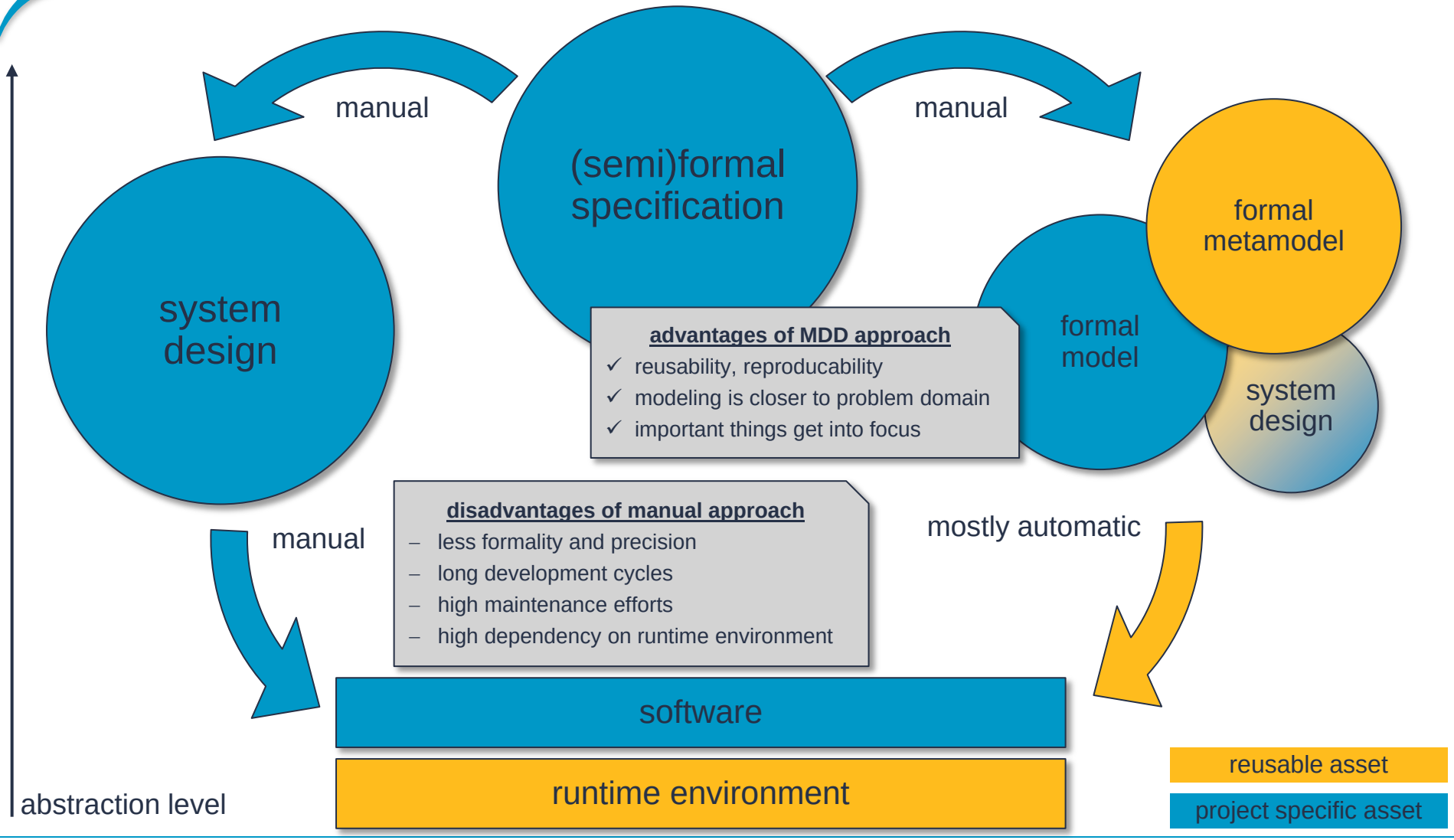
Agenda

- Classic vs. model-driven approaches
- Vocabulary: models and modelling languages
- Application: generators and runtime
- Process: success factors
- Summary

Agenda

- **Classic vs. model-driven approaches**
- Vocabulary: models and modelling languages
- Application: generators and runtime
- Process: success factors
- Summary

Standardization and formal specification helps to solve complex problems.



Different approaches for different use cases.

Top-Down

- “Full-scale” MDD project
- higher setup effort
- high customer involvement

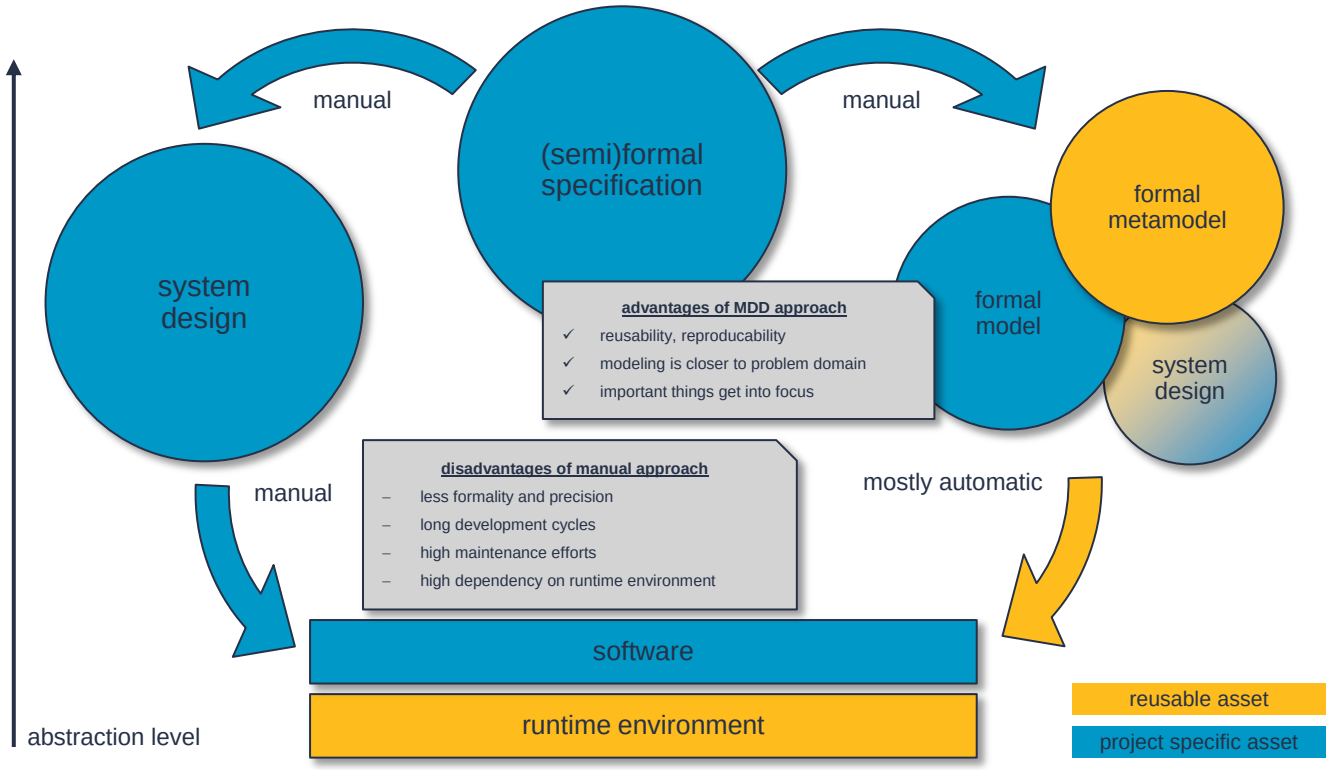
Closed System

- Vendor controlled runtime.
- Good tool support.
- Integration platform, often with analytical tools.
- Examples: SAP, BPM-Suites, …

Bottom-Up

- selected areas are modelled and generated
- often heterogeneous tool landscape

Let’s take a closer look…



M models

G generators

R runtime

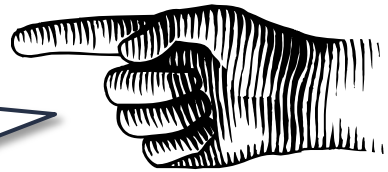
Agenda

- Classic vs. model-driven approaches
- **Vocabulary: models and modelling languages**
- Application: generators and runtime
- Process: success factors
- Summary

"Model driven development uses formal models to generate derived artefacts." – So what does that mean?

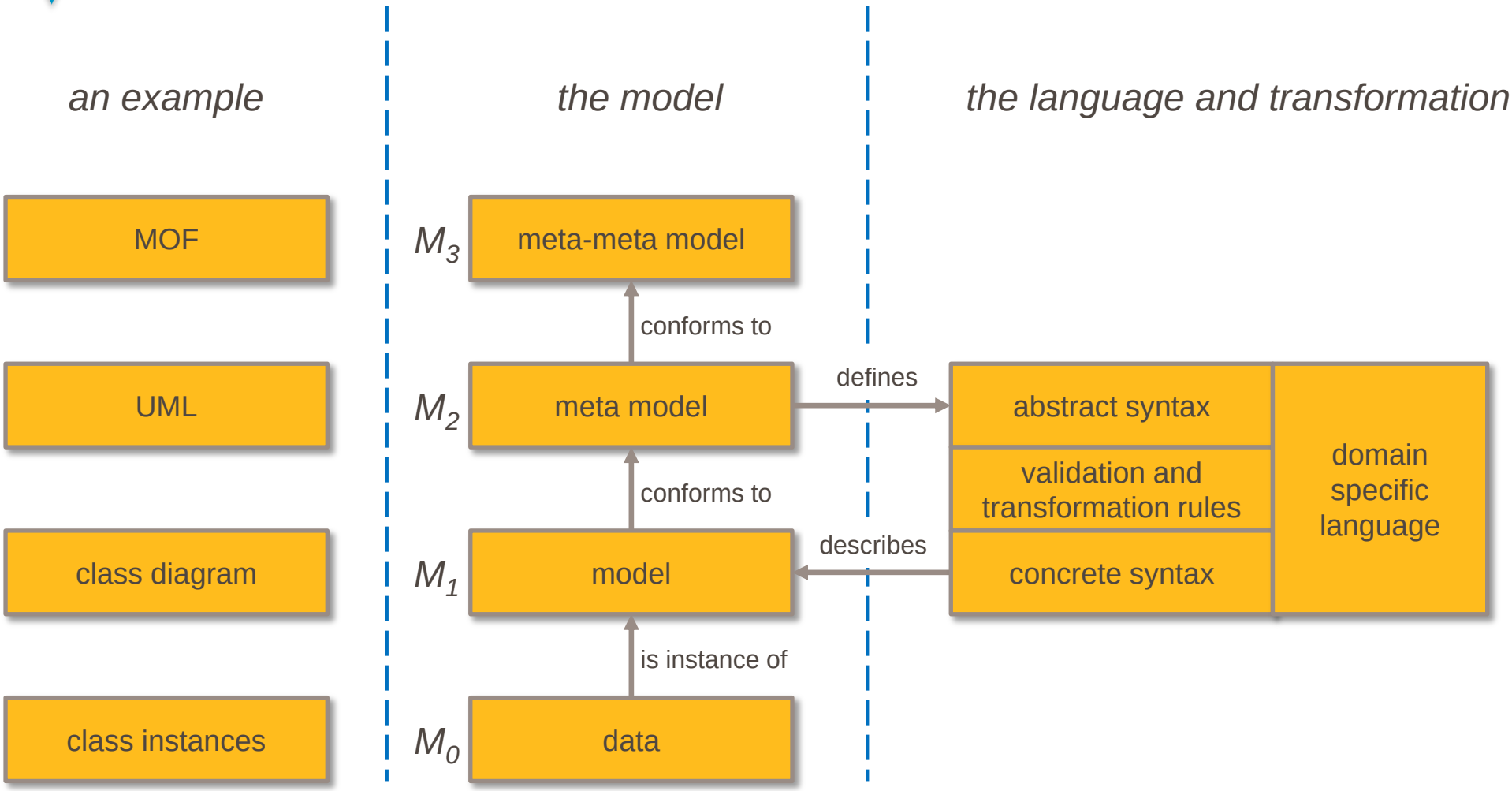
The **generated artifacts** can be models or source code	or simply data in the same or another format as the input model	so documents, XML or images can be created as well.
The **model** is a primary development artefact	but it is not the only one	because not everything can be put into the model.
A **formal metamodel** is required to generate artefacts	but the model is not limited to graphical representations	because text quite often allows for more concise representations.
The **modeling language** should be chosen carefully	and is not limited to UML	Because domain specific languages are often suited better.

Models are models, real life is different



Don’t be afraid of metamodelling.

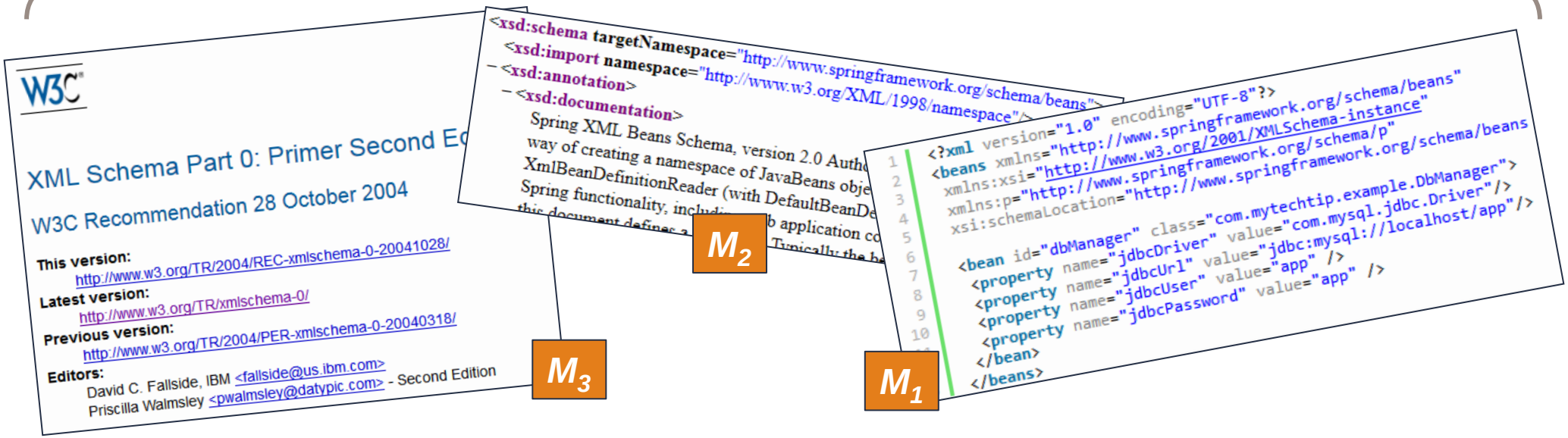
The concepts might sound strange, but they help to build a formal basis.



Let‘s take a look at a few example…

Domain specific languages are tailored towards specific applications.

technical ⟷ business

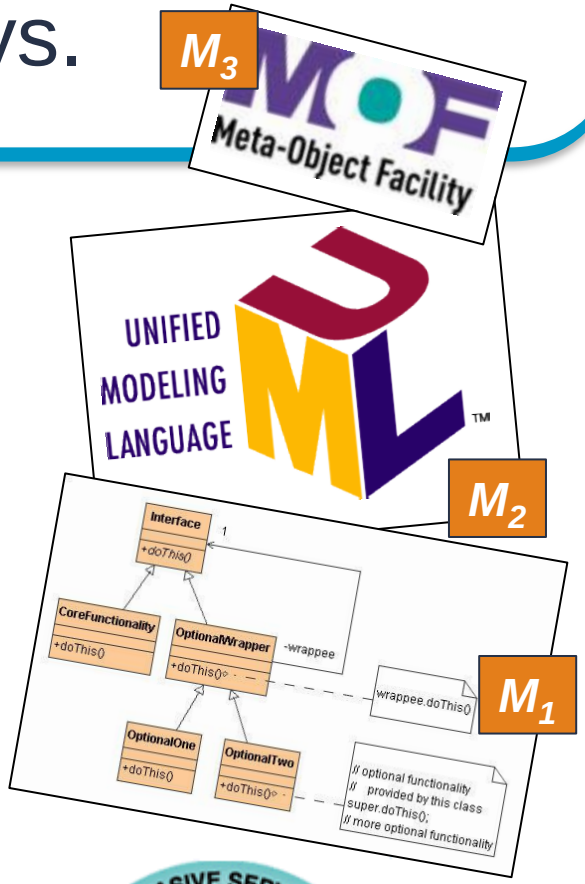


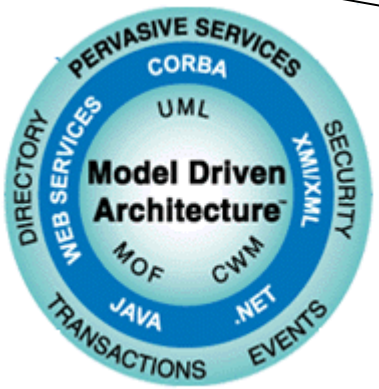
The UML can be extended in two ways.

- The MOF meta-metamodel is used to define the Unified Modeling Language.
- The UML consists of different viewpoints on software systems (e.g. class diagrams).

- UML profiles offer a lightweight extension of UML using stereotypes and tagged values.
- Heavyweight extensions, which add new graphical objects are possible as well, but there are nearly no tools available.

- The OMG propagates MDA as a paradigm for model driven development using UML profiles.





Defining the right domain specific language is the key to success with MDD.

- In some cases, existing languages are sufficient but often defining your own languages provides greater flexibility and can be tailored to the needs of the customer.

existing languages → custom made DSLs

- Extensive tool support for custom DSLs is already available:
  - Eclipse Modeling Platform, JetBrain MetaProgrammingSystem, Intentional Workbench
  - Languages with integrated, internal DSL support (e.g. Scala, .NET/LINQ)

Excerpt from MDD school at Capgemini

Replace “DataModel.xtext” with the following

```
grammar de.capgemini.mdd.DataModel with org.eclipse.xtext.common.Terminals
generate dataModel "http://www.capgemini.de/mdd/DataModel"
Model:
  (types+=StringType)*;

StringType:
  "string" name=ID length=INT;
```

Each rule starts with a literal (here “StringType”) following by a colon and then the production description. The rule is ended by a semicolon.

The production contains the parts to which the literal is expanded. They are separated by whitespace.

```
string address 30
```

This is a simple model (an instance of the metamodel). When it is parsed, a metamodel element of type “StringType” is created and its attributes name is set to “address” and length to “30”.

- The type “ID” serves as an identifier for the type system
- The type “INT” is used for integer type attributes.
- You can use “ | ” to separate expanding literals, e.g. `a: b | c;`
- **The rules not only define the abstract syntax (metamodel structure) but also the concrete syntax (how actual model instances look like).**

Some examples for Xtext DSLs can be found on http://blog.efftinge.de/

Business Process Modelling

Definition

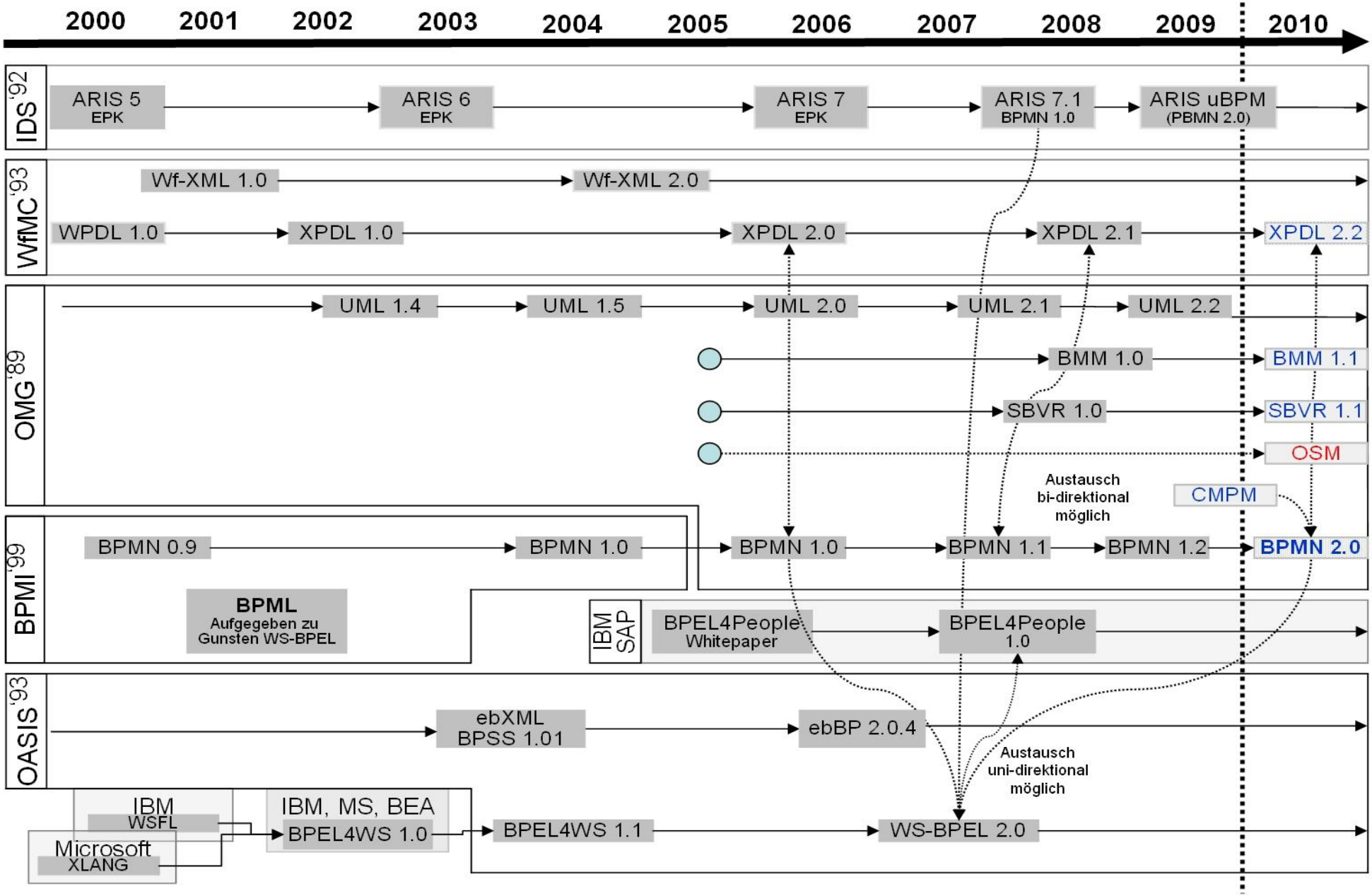
*BPM is a holistic management approach focused on **aligning** all aspects of an organization with the **wants and needs** of clients. It promotes business effectiveness and **efficiency** while striving for **innovation**, **flexibility**, and **integration** with technology. BPM attempts to **improve processes continuously**. [Wikipedia]*

Standard Notations

- Two main notations:
  - Event-driven Process Chain (EPC), part of the ARIS toolset (previously IDS Scheer, now Software AG Darmstadt)
  - Business Process Modelling Notation (BPMN)

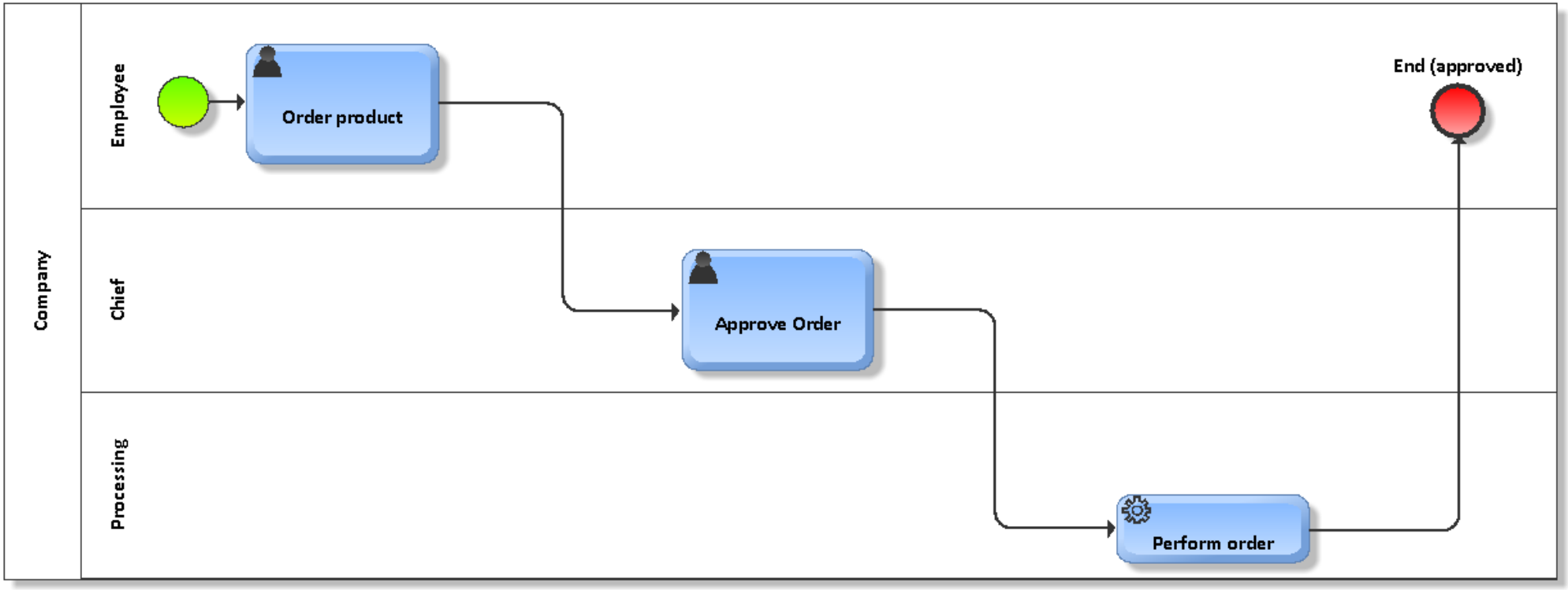
Tool support (mostly for BPMN)

- Visual editors
- Execution environment
- Monitoring and statistics
- Versioning and Deployment support
- Integration with other tools (SOA, Business rule engines)



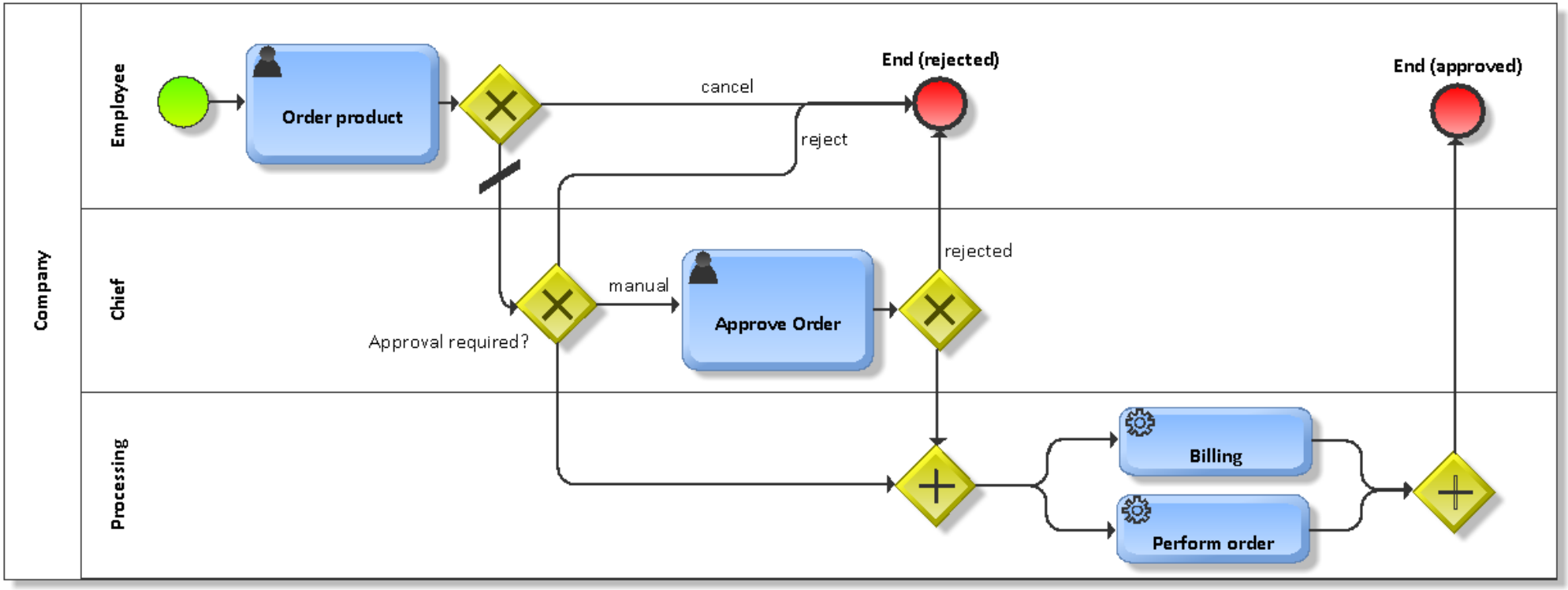
Sources:
History of BPM Standards: Dr. Martin Bartonitz, http://de.wikipedia.org/w/index.php?title=Datei:Standards_BPM_2009_11.jpg&filetimestamp=20100702012135

Modelling of Processes
BPMN – Start, End and Tasks

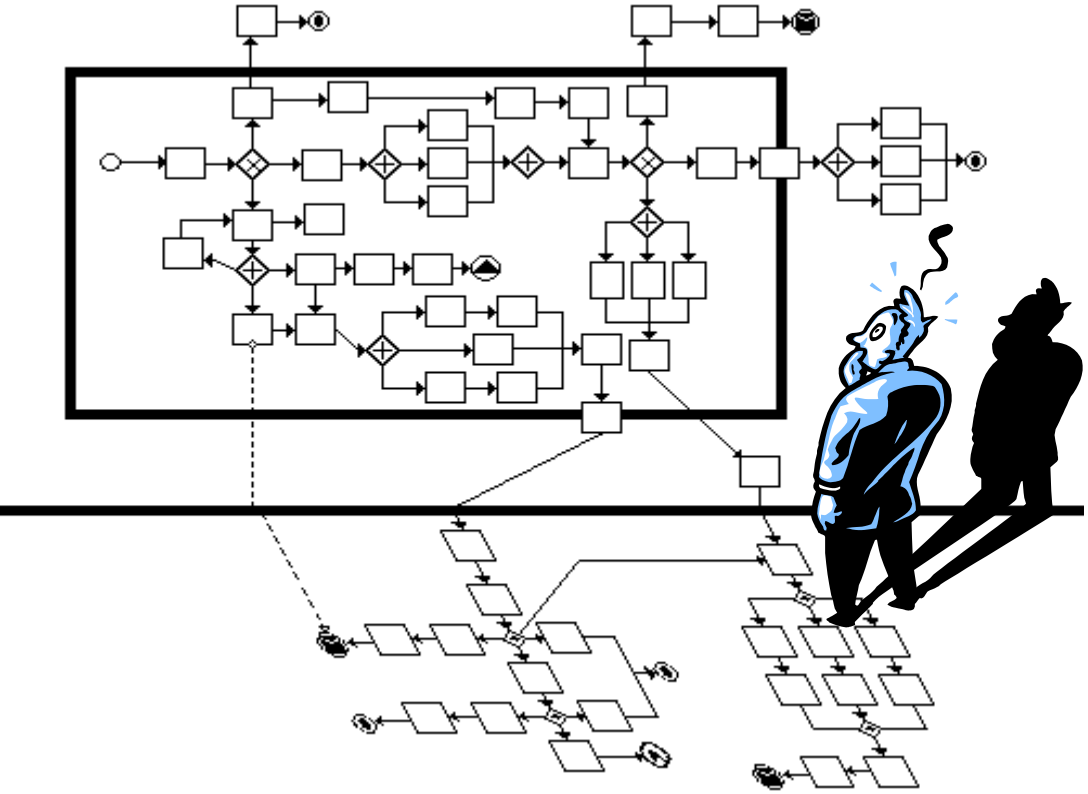


Modelling of Processes

BPMN – Exclusive Gateways (Decisions) and Parallel Gateways

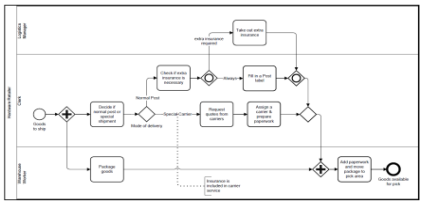


BPM – Modelling with different Levels of Abstraction

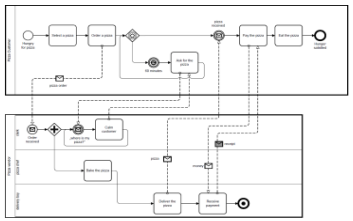


Highlevel Process

Happy Path

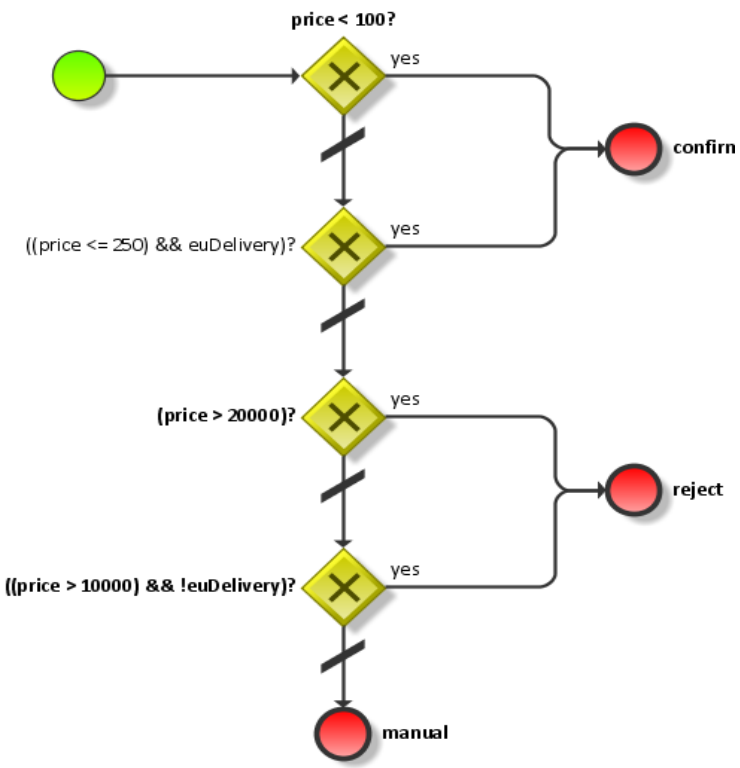


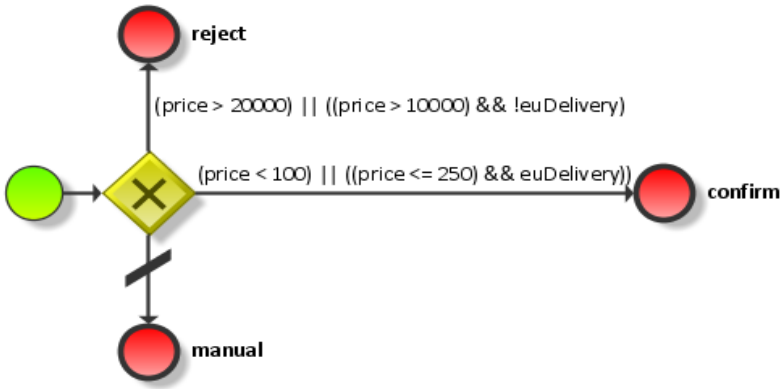
Real Life

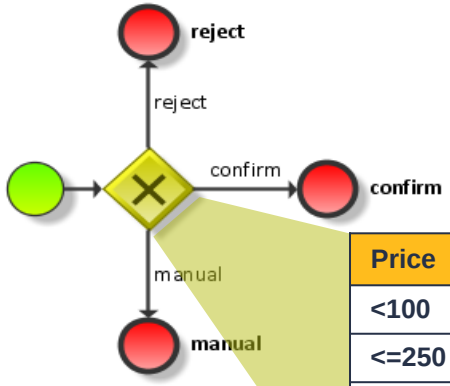


Fully Executabe

Combining Business Processes & Rules







Price	EU Delivery?	Result
<100		confirm
<=250	yes	confirm
>20000		reject
>10000	no	reject
Otherwise		manual

Business Rule Engine: JBoss Drools

- uses RETE algorithm to boost execution performance
- Runs on application server (e.g. Tomcat)
- Library approach
- Open source
- Homepage: http://www.jboss.org/drools
- Current Version: Drools 5
  - Drools Guvnor (BRMS/BPMS)
  - Drools Expert (rule engine)
  - Drools Flow (process/workflow)
  - Drools Fusion (event processing/temporal reasoning)
  - Drools Planner

The Rete algorithm is an efficient pattern matching algorithm for implementing production rule systems. …The word 'Rete' is Latin for 'net' or 'comb'. The same word is used in modern Italian to mean network. Charles Forgy has reportedly stated that he adopted the term 'Rete' because of its use in anatomy to describe a network of blood vessels and nerve fibers.

Performance woes

Using Excel as a DSL-Editor bridges the gap between business and technical architecture.

Microsoft Excel - beispiele.xls

B34

	A	B	C	D	E	F	G
1	**Regelbank fiktive Antragsprüfung**						
2	Sicht:	fachlich					
3	**Prüfung der Namen**						
21	**Prüfung des Geburtsdatums**						
22	In diesem Regelsatz wird die Korrektheit und Plausibilität des Geburtsdatums geprüft.						
23	1. Ist ein Geburtsdatum angegeben?						
24	2. Ist das Geburtsdatum mit dem Antragsdatum konsistent? (Wenn kein Antragsdatum angegeben ist, nimm das heutige Datum als Vergleichsbasis.)						
25							
26			**RuleTable Geburtsdatumpruefung**				
30	*Erläuterung*		*Regelname*	**Geburtsdatum**	**Antragsdatum**	**Geburtsdatum ist größer als ...**	**erzeuge Fehler**
31	*Wenn der Antragsteller kein Geburtsdatum angegeben hat, wird der Fehler 4311 erzeugt.*		Geburtsdatumpruefung.kein-Datum-vorhanden	leer			4311
32	*Wenn der Antragsteller ein Geburtsdatum angegeben hat, das nach dem Antragsdatum liegt, wird Fehler 8000 erzeugt.*		Geburtsdatumpruefung.Datum-nach-Antragsdatum	vorhanden	vorhanden	Antragsdatum	8000
33	*Wie Regel davor, jedoch hat der Antragsteller kein Antragsdatum angegeben, es wird mit*		Geburtsdatumpruefung.Datum-nach-heute	vorhanden	leer	heute	8000
34							
35	**Prüfung der Adresse**						
46							

fiktive Antragsprüfung

Bereit

Discussion on Business Process and Rule engines

Advantages

- declarative approach: “What” not “How”
- Separation of Logic from Data, Flow and processes
- High efficiency, especially for rule engines
- central repositories for processes and rules, “the model is the code”

When to use these approaches

- High involvement of business side necessary
- Changes have to be applied often
- complex relations, which are not easy to write in standard programming languages

Agenda

- Classic vs. model-driven approaches
- Vocabulary: models and modelling languages
- **Application: generators and runtime**
- Process: success factors
- Summary

Choosing the right platform is important.

R

- Today's software systems consist mostly of standard components, which have to be configured correctly.
- The actual business logic is only a minor part.
- The idea of MDD is not to generate the complete application but only parts, which are not part of the platform.
- In addition, configuration for the platform can be generated!

Tomcat / JBoss

Acegi

Spring MVC

Spring WebFlow

MVC Action

AOP TX Proxy

Business Logic

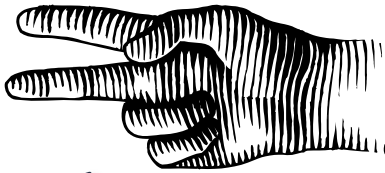
DAO

Spring-Hibernate

Hibernate

JDBC

contains generated executable code

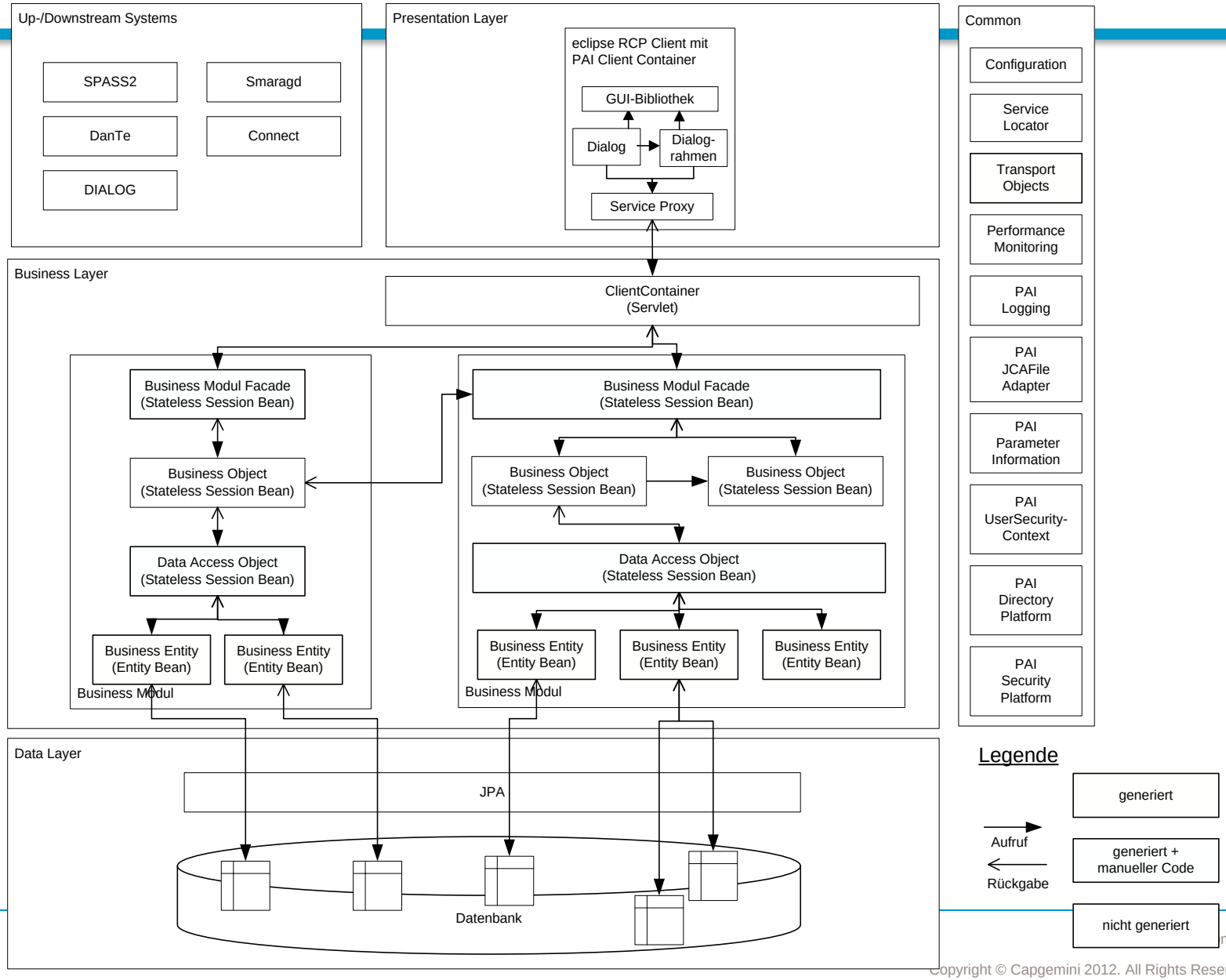


All in and no way out

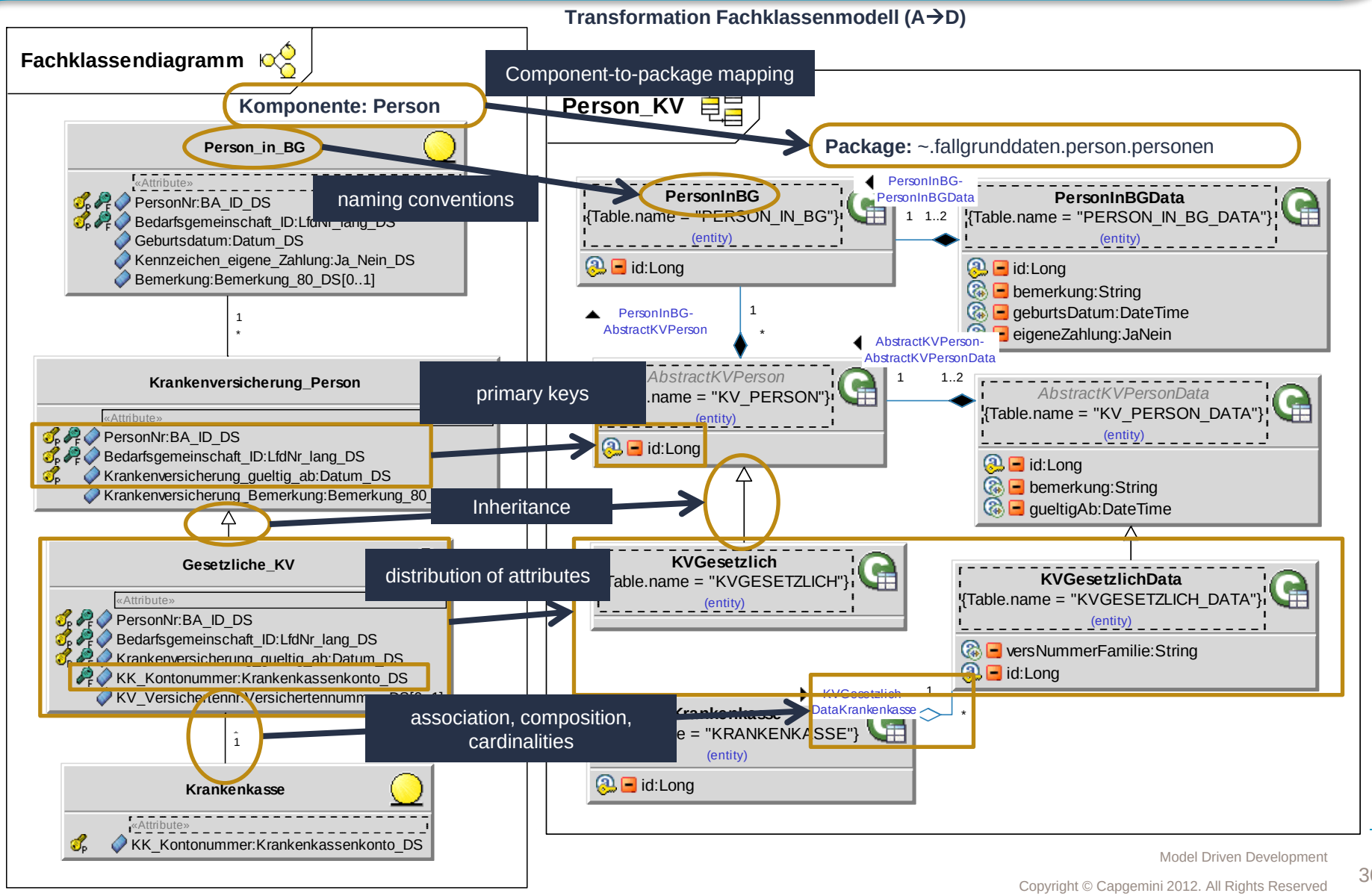
Source: From HTTP up to JDBC as a picture

http://ptrthomas.wordpress.com/2006/06/06/java-call-stack-from-http-upto-jdbc-as-a-picture/

Up to 50% can be generated on certain platforms.

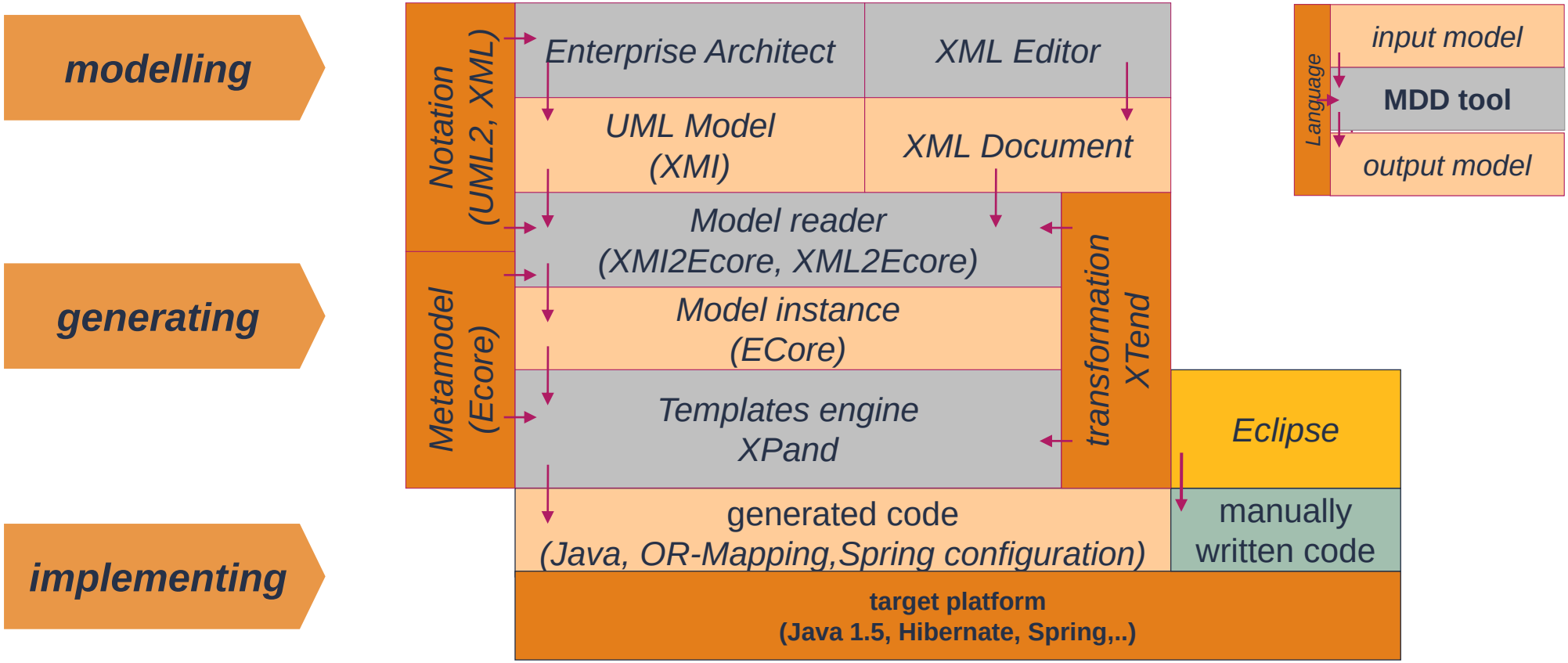


Example: Mapping from Specification to Design



An generic example illustrates the different roles.

G



“Software factory” for retirement provisioning (german: Altersversorgung)

High effort with low return

Generated code for three different instances

Zusatzversorgungs-Kasse

Beamten-Versorgung

Berufsständische Versorgungswerke

- Base- and business entitities
- In-/Exkasso
- Rules and Formulas for provisioning calculation
- Tariff- and Account-Management
- Printing
- Historation component
- Support for Multitenancy

Framework

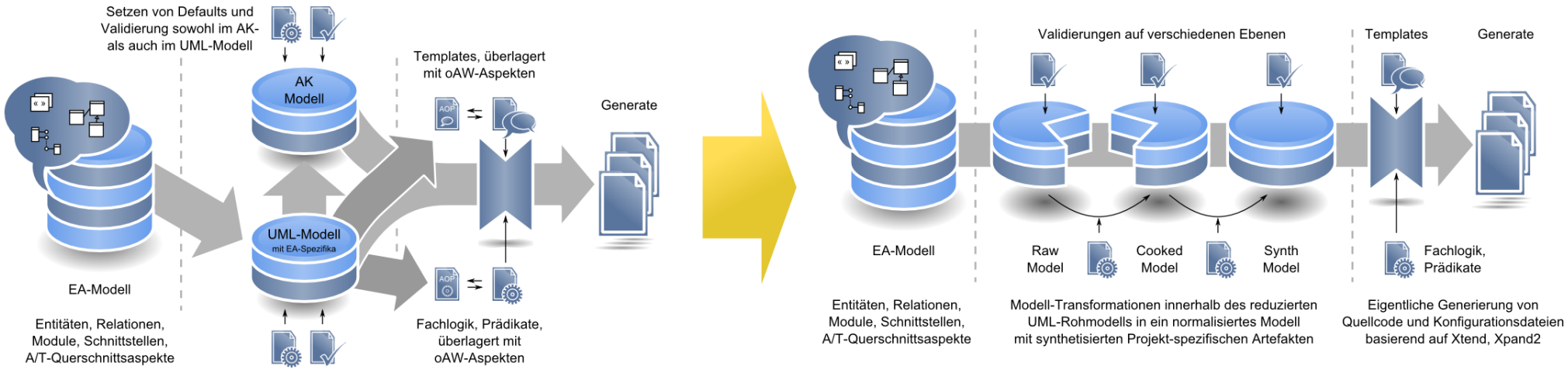
- Persistence (Hibernate)
- Transaction (Hibernate)
- DI-Framework(Spring)
- Middleware (Webservices, Axis)
- Language (Java 1.5)
- Quasar-Client
- Security (Acegi)

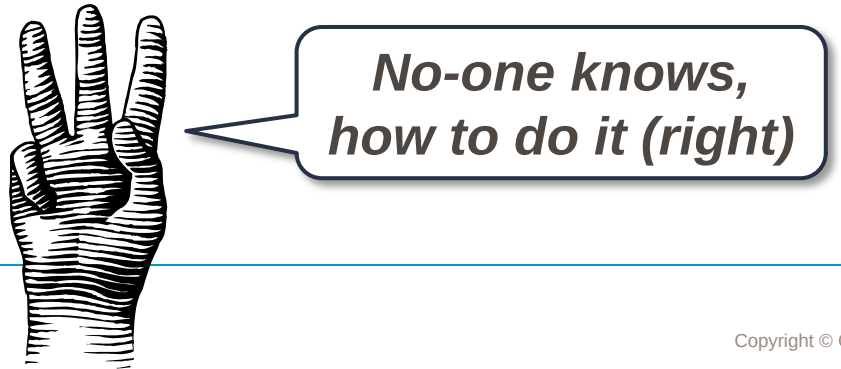
Platform

Developing and refining the generator chain is an important but time consuming task.

Example for evolution of a project specific generator chain:

- Effort: 1 month full time work (including new features)
- Result: Reduction of maintance work from several days to a few hours.





What we have seen so far…

Top-Down

- "Full-scale" MDD project
- higher setup effort
- high customer involvement

Closed System

- Vendor controlled runtime.
- Good tool support.
- Integration platform, often with analytical tools.
- Examples: SAP, BPM-Suites, …

More examples from a large project in the public sector.

Service Gateway Generator

- Technology: Groovy, Velocity, Ant
- Copies a parameterizable project template using ant.
- Generates code for authorisation, dispatching and error handling using wsimport and a groovy script, which parses a WSDL and control the velocity template engine.

Document Generator

- Technology: Enterprise Architect, .NET-Application
- The specification is modelled in the UML tool Enterprise Architect.
- Conventions for modelling include certain stereotypes and other aspects.
- A COM-based application reads the model from EA and controls Word to create a specification document.

Business Process and Rule Engine

- Technology: JBoss JBPM and Drools, MS Excel
- Validation rules for data are written using Excel.
- Macros and a converter creates native drools rules, which are parsed and startup of the application.
- Business processes are modelled using an Eclipse-based graphical editor, which creates XML.
- A JBPM tool creates SQL which persists the process definition into a database.

Model Transformation

- Technology: Enterprise Architect
- Code generation using proprietary EA template language
- Model transformation from specification to implementation model using so called “MDA style transformations” (also EA proprietary).
- Parts of the transformation script are generated using formulas and macros within an excel sheet.

Agenda

- Classic vs. model-driven approaches
- Vocabulary: models and modelling languages
- Application: generators and runtime
- **Process: success factors**
- Summary

Early project phases are vital to successful projects with high MDD usage.

Working Knowledge Management

- Consistent tool chain
- Community support

Customer acceptance

- Models are accepted artifacts
- Customer are actively involved in modelling

Distinct team roles

- Permanent team members with detailed knowledge of generator chain and modelling environment
- Capable offshore team

Early planning and project initialization

- Consider MDD during bid phase
- Early setup of tool chain with competent team
- MDD is not limited to the construction phase, consider all project phases
- Think about later: Migration, Merging, Lifecycle

So what are the ingredients for a complete MDD architecture?

Organisation (People & Processes)

- Process adaption to deal with formal models and generation processes
- A role model with skill definitions (who adapts the platform, who writes or configures transformation rules, etc.)
- Interaction concept with the customer (who is responsible for models, etc.)

Platform (Reusable assets)

- Target architecture: components, runtime, application servers, etc.
- Modelling architecture: meta-meta-language, transformation language
- Concepts for advanced topics: metamodel migration, reverse engineering, debugging

Infrastructure (Specific assets)

- Modelling language
- Validation and transformation rules
- Integration concept: What parts are generated, how do they integrate with manually written code, versioning
- Frameworks which adapt between generated and manually written code and the target architecture

Tools (Software & Configuration)

- Modelling tools
- Development environment
- Transformation engine
- Tool chain setup

Agenda

- Classic vs. model-driven approaches
- Vocabulary: models and modelling languages
- Application: generators and runtime
- Process: success factors
- **Summary**

Let's revisit the five arguments against model driven development:

Models are models, real life is different

No-one knows, how to do it (right)

Performance woes

All in and no way out

High effort with low return

*With **organizational structures** in place, an **experienced team** and **early setup** of a project **tailored tool chain** MDD provides several advantages over "classical" development.*

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.

About Capgemini

With more than 120,000 people in 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2011 global revenues of EUR 9.7 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini

www.capgemini.com

Research and science live on the exchange of ideas, the clear arrangements are thereby useful

Disclaimer:
Although this presentation and the related results were created carefully and to the best of author's knowledge, neither Capgemini nor the author will accept any liability for it's usage.

If you have any questions, please contact:
Capgemini | Offenbach
Dr. Martin Girschick
Berliner Straße 76, 63065 Offenbach, Germany

martin.girschick@capgemini.com

Appendix: MDD best practises

The abstract syntax – defining the right metamodel

Distilled from Markus Völter: “MD*/DSL Best Practices”

- **Understand** the business and the language they use. Take a look at the documents they write.
- Ensure that it can **properly be translated to code** (or whatever derived artefact you want to create)
- Think of **modularisation** and **viewpoints** (or even annotation concepts) to cover certain aspects of the complete model. Find well defined connection points between them, make sure those “interfaces” are unidirectional and simple.
- **Limit expressiveness**
  - Stick to declarative languages.
  - Often, DSLs can be categorized in two types:
    - **customization DSLs** provide a vocabulary to express facts
    - **configuration DSLs** provide values to parameters, they are often simpler to design but less expressive
  - The languages is the “**what**”, the generator creates the “**how**”. Domain experts often only know the “what” but not necessarily the “how”.
  - If the language needs to be turing complete, a DSL might not be a good idea. Define a proper API instead or provide hooks in the generated code to add expressiveness in a standard programming language. Internal DSLs or languages which can be properly extended might be an alternative as well.

http://voelter.de/data/pub/DSLBestPractices-2011Update.pdf

The concrete syntax – Notation matters!

Partly distilled from Markus Völter's paper.

- Stating the obvious (or maybe not)
  - Stick to **existing notations**, if possible.
  - Make sure, that **appropriate tooling** is available.
  - **Textual or graphical** - choose carefully! Sometimes mixed forms or separate viewpoints (with the same or a different representation) help. Think of the different user groups.
  - Provide proper defaults, try to make models small.

- **Textual notations**
  - Appropriate tooling is often easier to find (e.g. proper editors, multiuser-support, build integration).
  - Not limited to structured text. Tables or forms are possible as well.
  - It's often easier to structure large models using text, beautifying can be automated.
- **Graphical notations**
  - Might be necessary, if relationships exist (e.g. dependencies, flows, sequencing).
  - Not all cases require a specialized editor – providing templates and convention might be enough.
  - Specialized tools often offer GUI prototyping to create an appropriate editor (e.g. Eclipse-based GEF-Tools).

Code Generation – make it nice and they'll like it!

- The semantics are encoded in the generator or interpreter.
- However, the language user needs a description as well!

- Keep generated code **separate** from manually written code.
- Some systems offer "**protected regions**", which are retained upon regeneration. Refrain from using them, uses appropriate design patterns and APIs instead.
- User **versioning** for primary artefacts, only (models, transformation rules, manually written code).
- Generate **beautified code** (higher acceptance, easier debugging).
- Generate **templates** as a basis for manually written code. Do that only once.