Model Driven Development in industrial practice

Dr. Martin Girschick
February 2018

Thanks for attending my presentation about model-driven development, I hope you gained some insight into how MDD is applied successfully. Michael Eichberg will make the slides available to you. Feedback of any kind is highly appreciated, just send me an e-mail: martin.girschick@capgemini.com. You can also send me a photo/scan of the contact sheet or if you have any others questions concerning Capgemini.

As mentioned, I organize student events and workshops. Upcoming next is an “after work” event on 20th of March, where you can meet young colleagues and talk about their work at Capgemini. Details will be posted on https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=293 . If you are interested you can also just send me an e-mail.

Capgemini

- Study and PhD at TU Darmstadt
- Since 2008 working for Capgemini
- Projects in Public Sector, Telecommunications, Finance, Logistics
- Different Roles: Developer, Architect, Quality Assurance, Project lead, Consultant, …
- Lead of german Capgemini Community for model-driven development
- University Relations for TU Darmstadt

Capgemini in numbers

~200.000
people worldwide

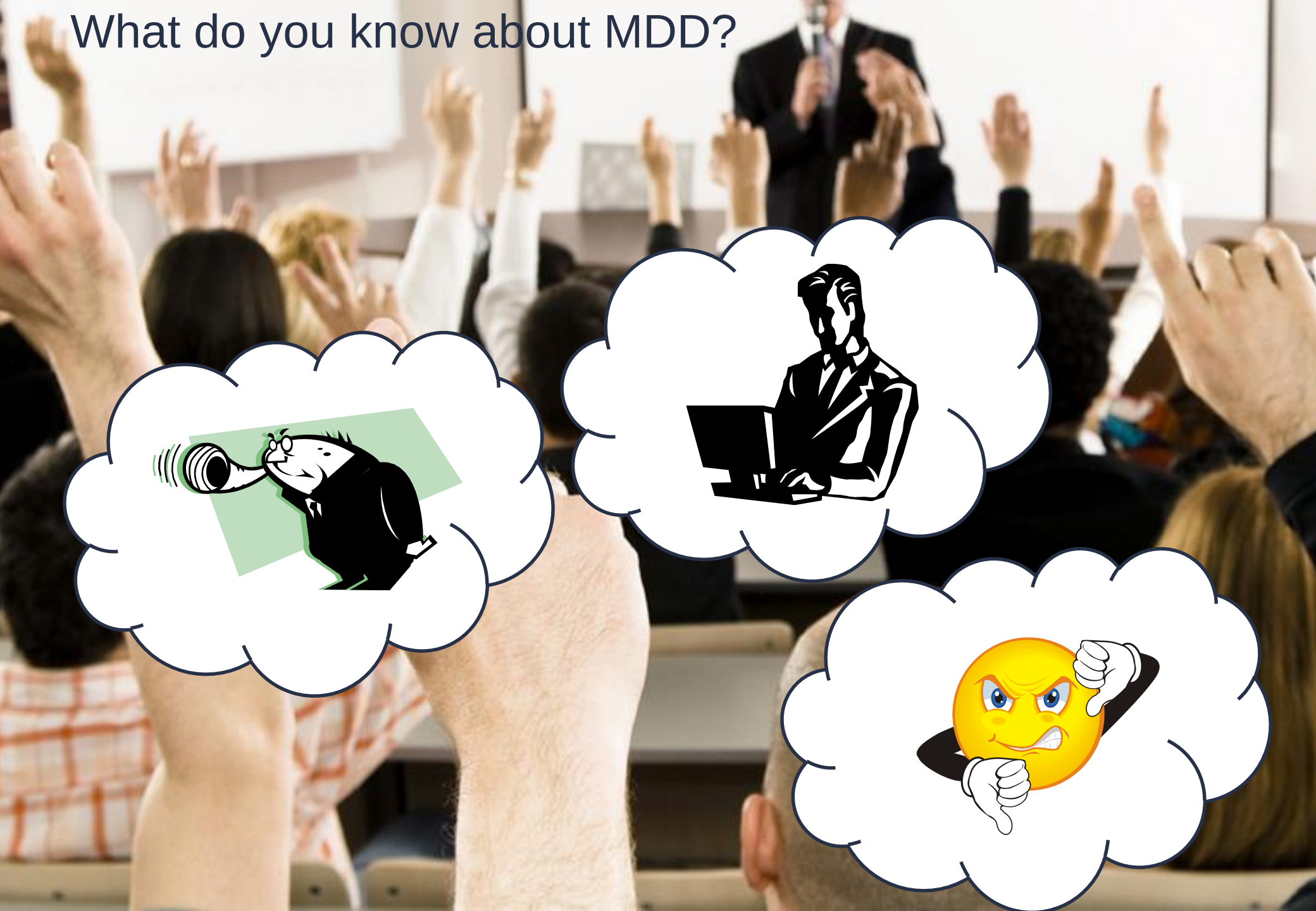
Revenue worldwide for 2016: **12,5 M€**

Over **120** different nationalities

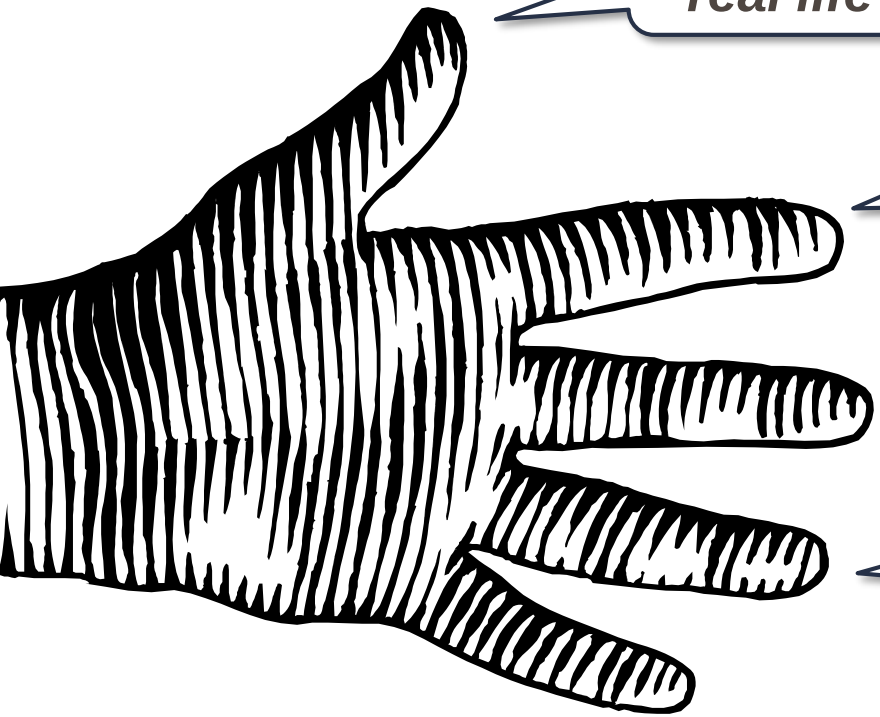
2017: **50th anniversary**

More than **40** countries

What do you know about MDD?



Five arguments against model driven development



Models are models, real life is different

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

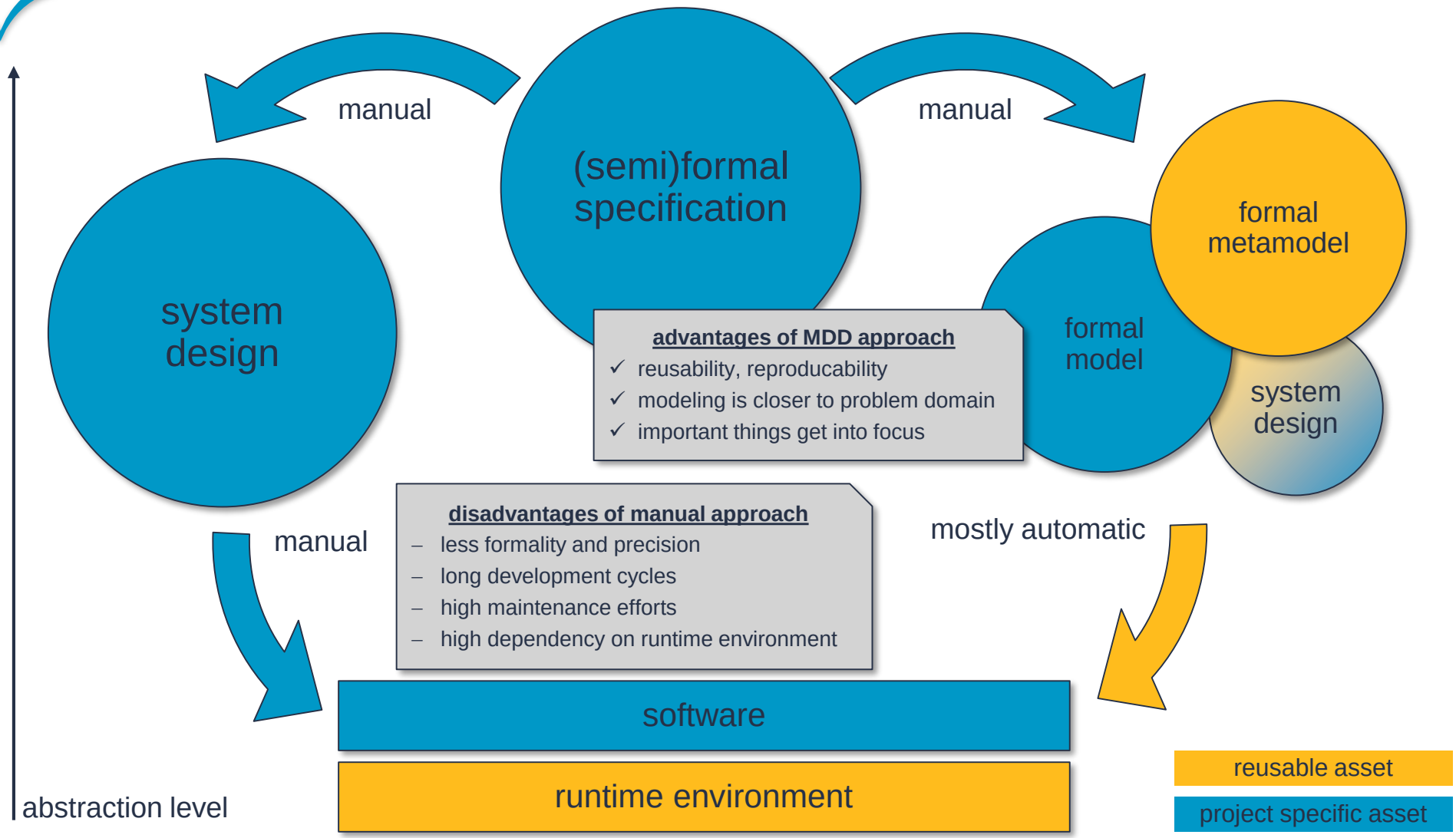
Performance woes

All in and no way out

High effort with low return

This presentation refutes these arguments and gives examples on successful use of MDD.

Standardization and formal specification helps to solve complex problems.



Models are models, real life is different.

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, …

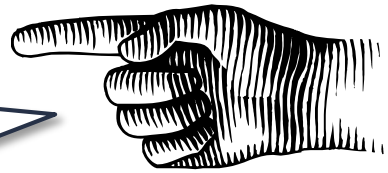
Bottom-Up

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

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

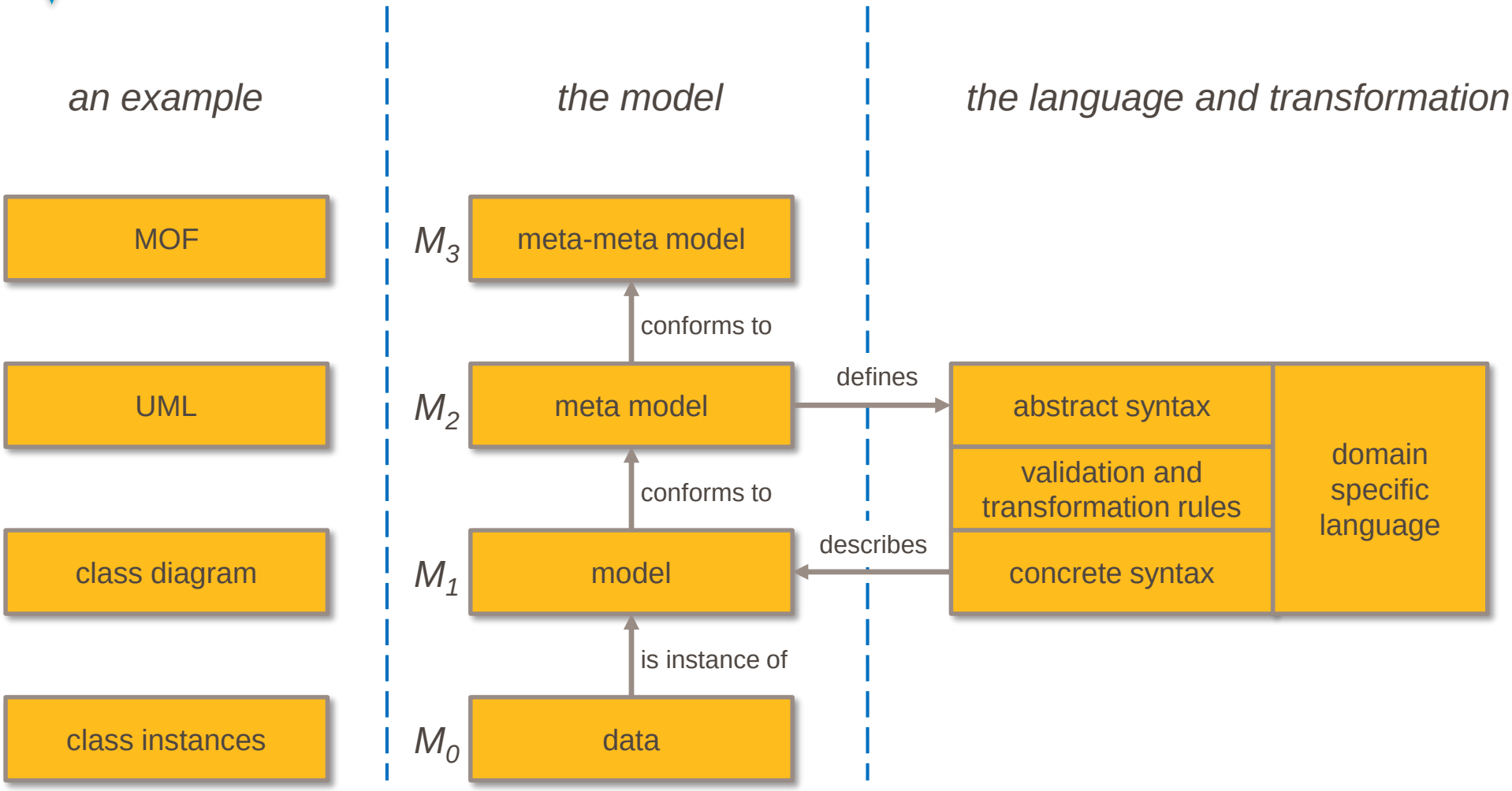
The **generated artifacts** can be models or source code	or simply data in the same or another format as the input model	so documents, XML or images can be created as well.
The **model** is a primary development artefact	but it is not the only one	because not everything can be put into the model.
A **formal 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.

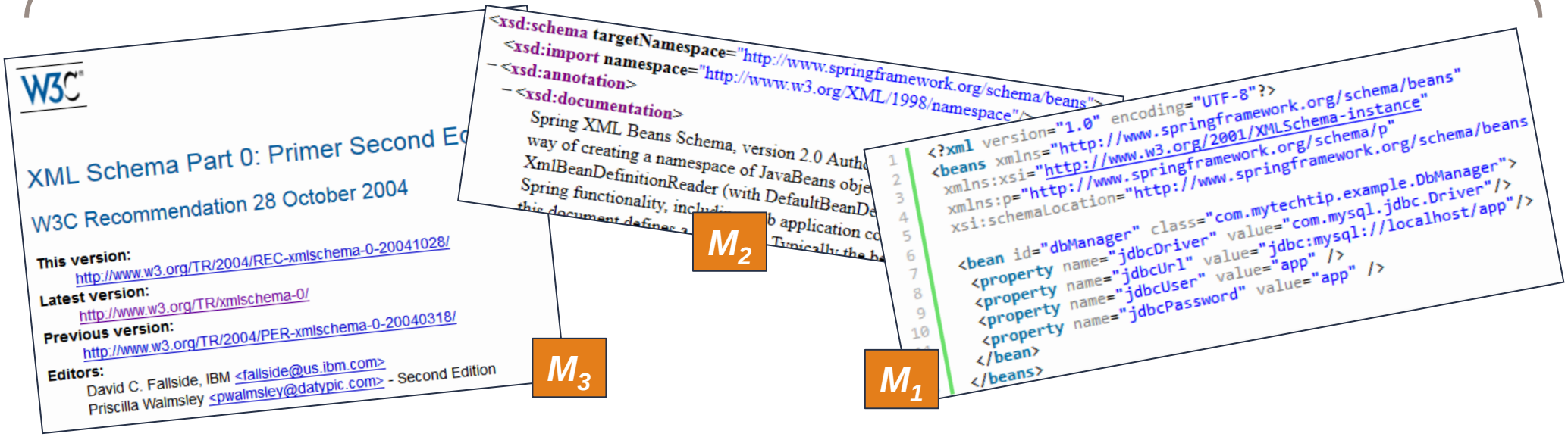
The concepts might sound strange, but they help to build a formal basis.



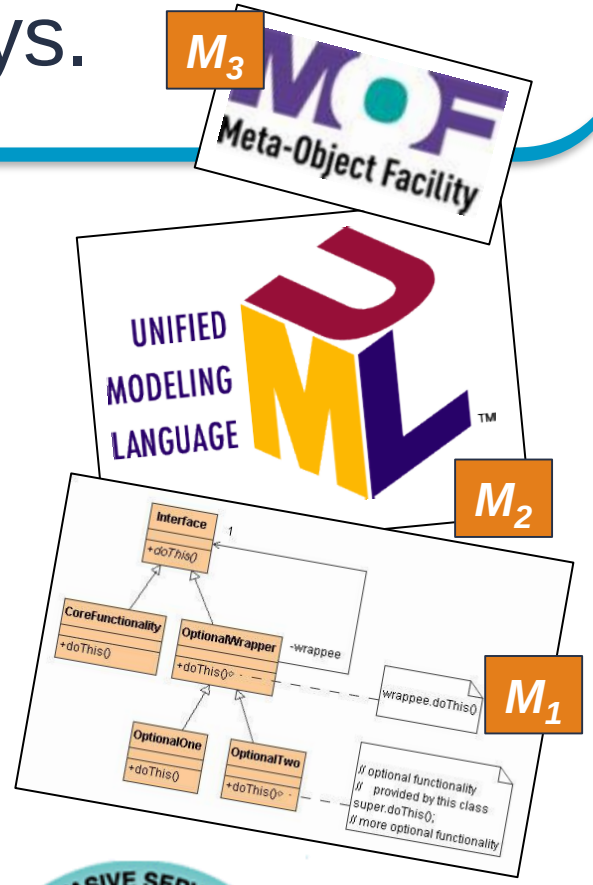
Let‘s take a look at a few example…

Domain specific languages are tailored towards specific applications.

technical ⟷ business



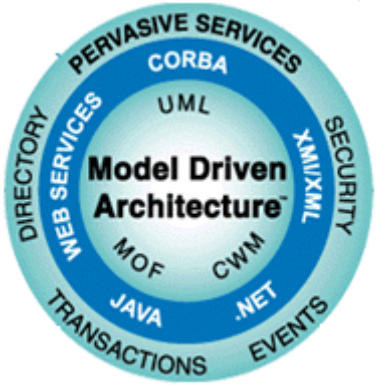
The UML can be extended in two ways.



- 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.

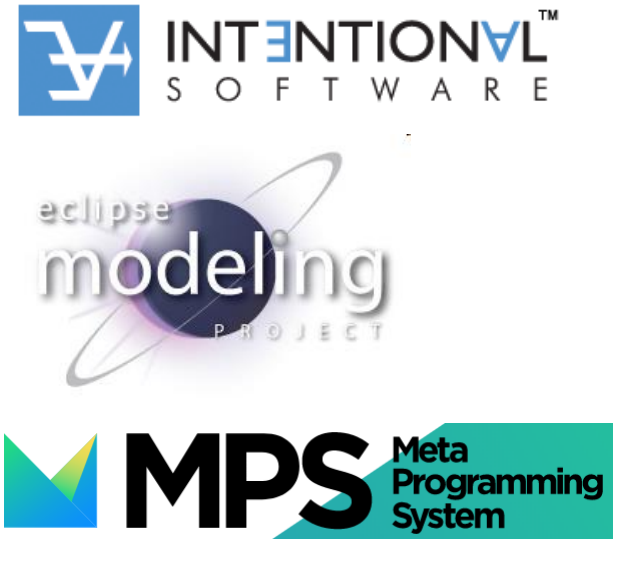
- The OMG propagates MDA as a paradigm for model driven development using UML profiles.



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 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.

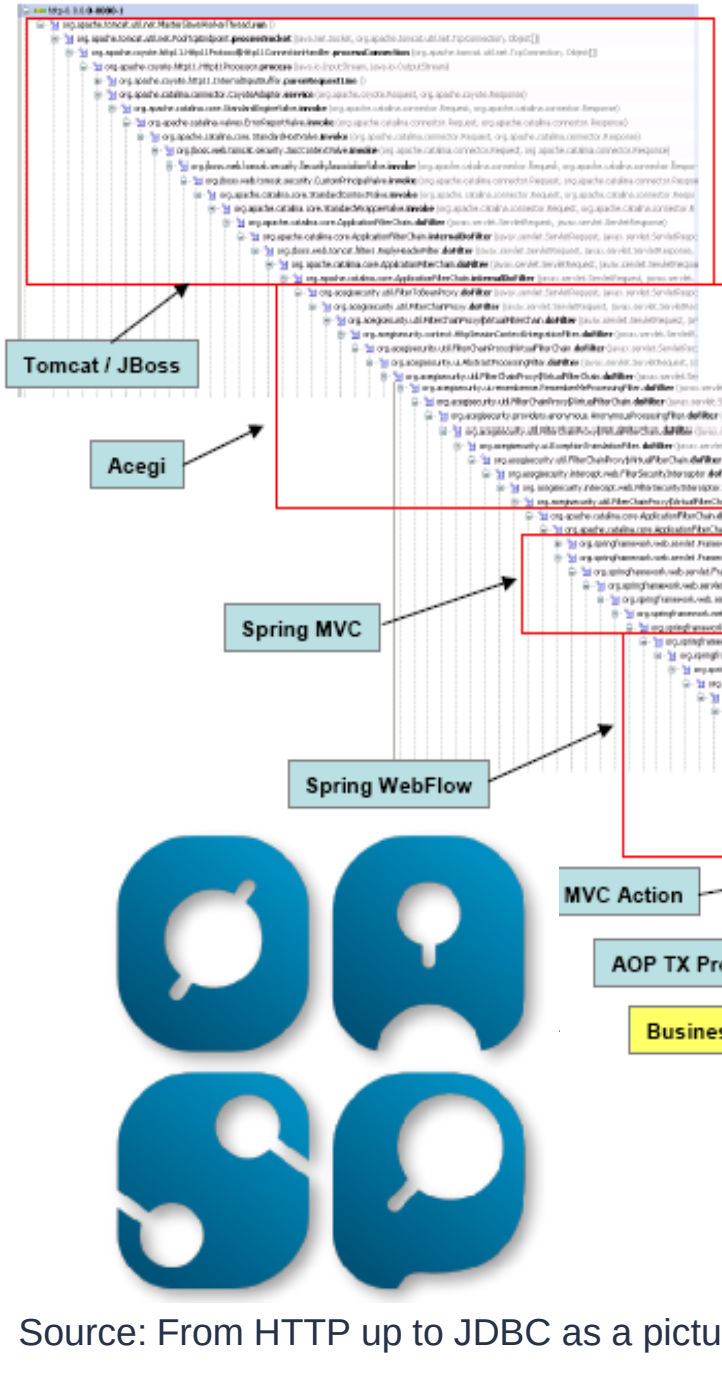
```
string address 30
```

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".

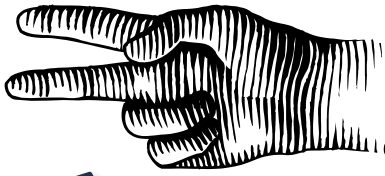
- The type "ID" serves as an identifier for the type system
- The type "INT" is used for integer type attributes.
- You can use "`|`" to separate expanding literals, e.g. `a: b | c;`
- **The rules not only define the abstract syntax (metamodel structure) but also the concrete syntax (how actual model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

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

Choosing the right platform is important.



- 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!

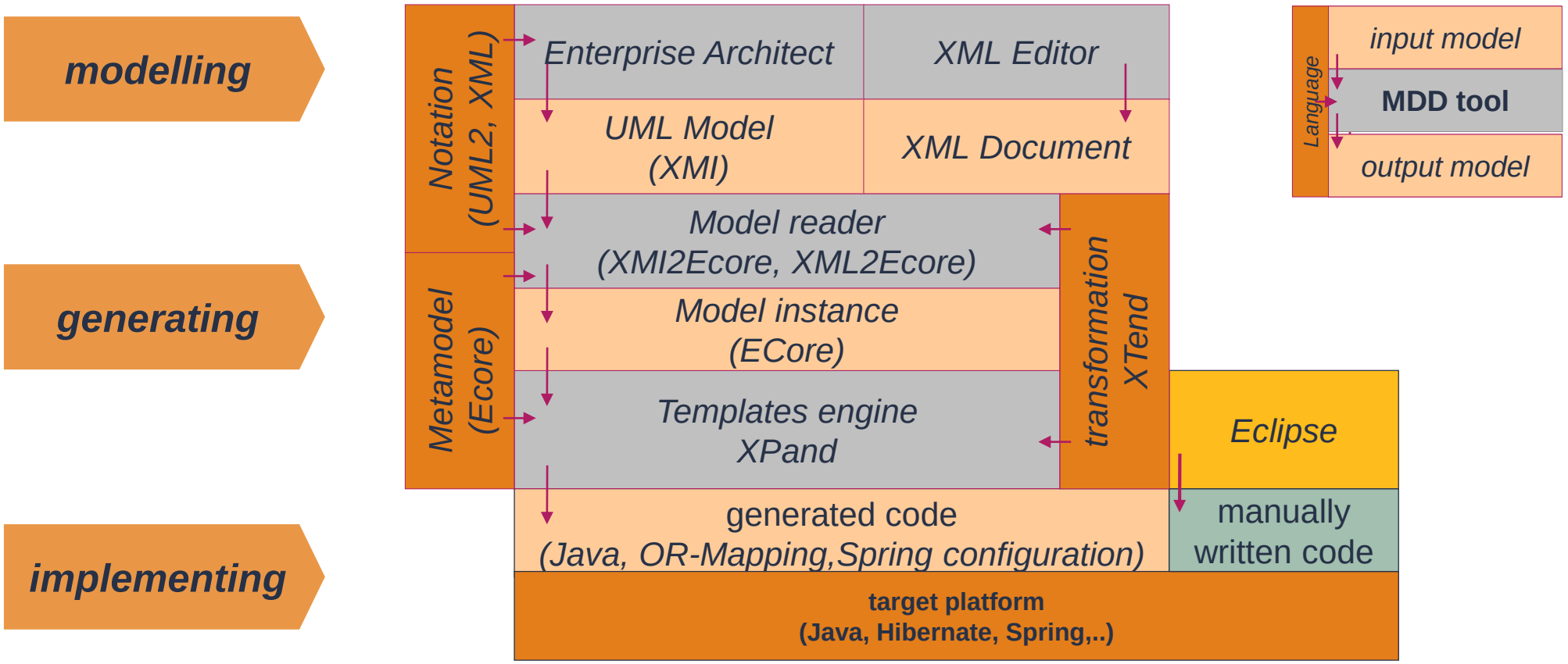


All in and no way out

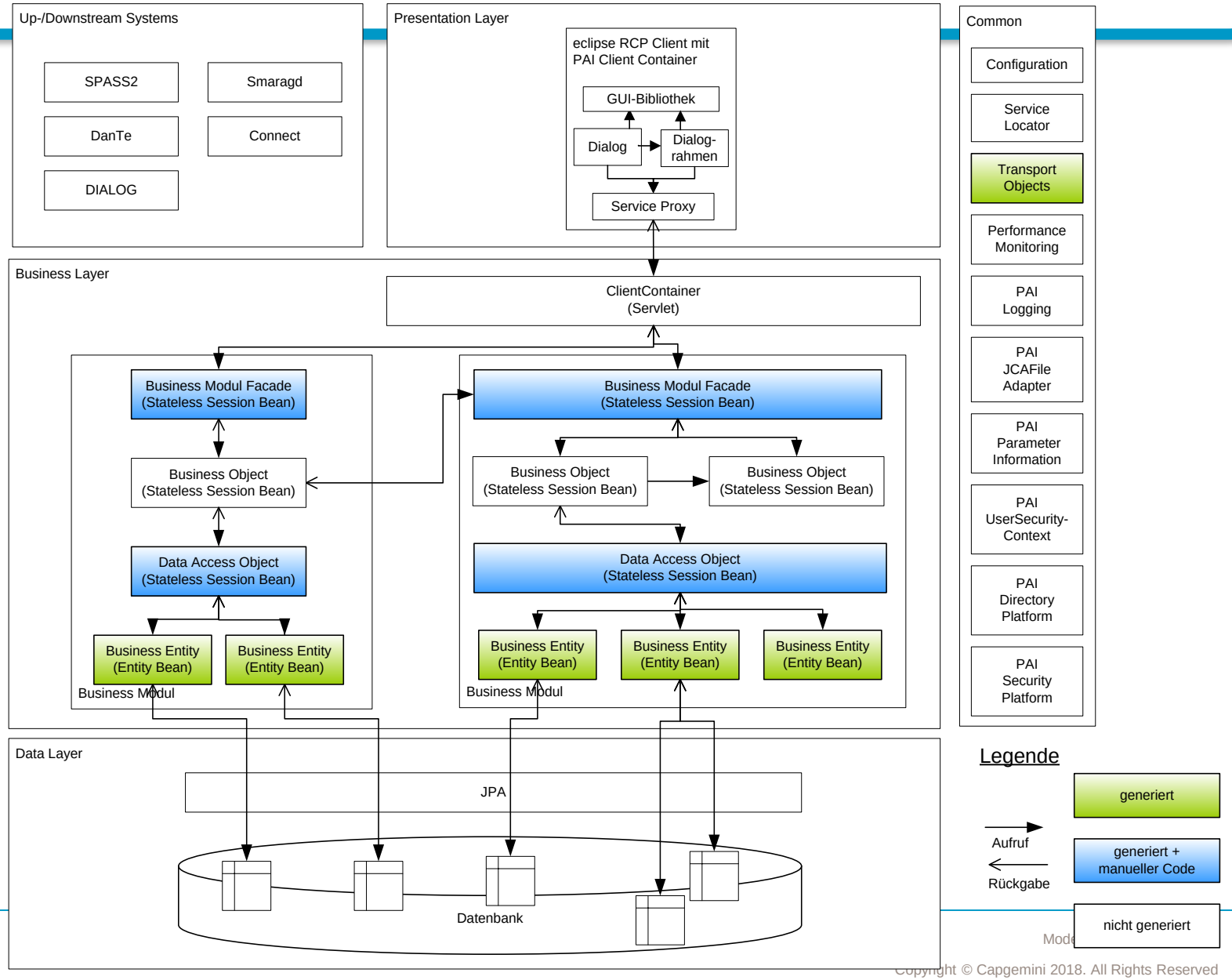
Source: From HTTP up to JDBC as a picture

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

The multistage process from model to code.



Up to 50% can be generated on certain platforms.



Four examples illustrate the potential for MDD.

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

Generated code for three different instances

Customer A

Customer B

Customer C

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

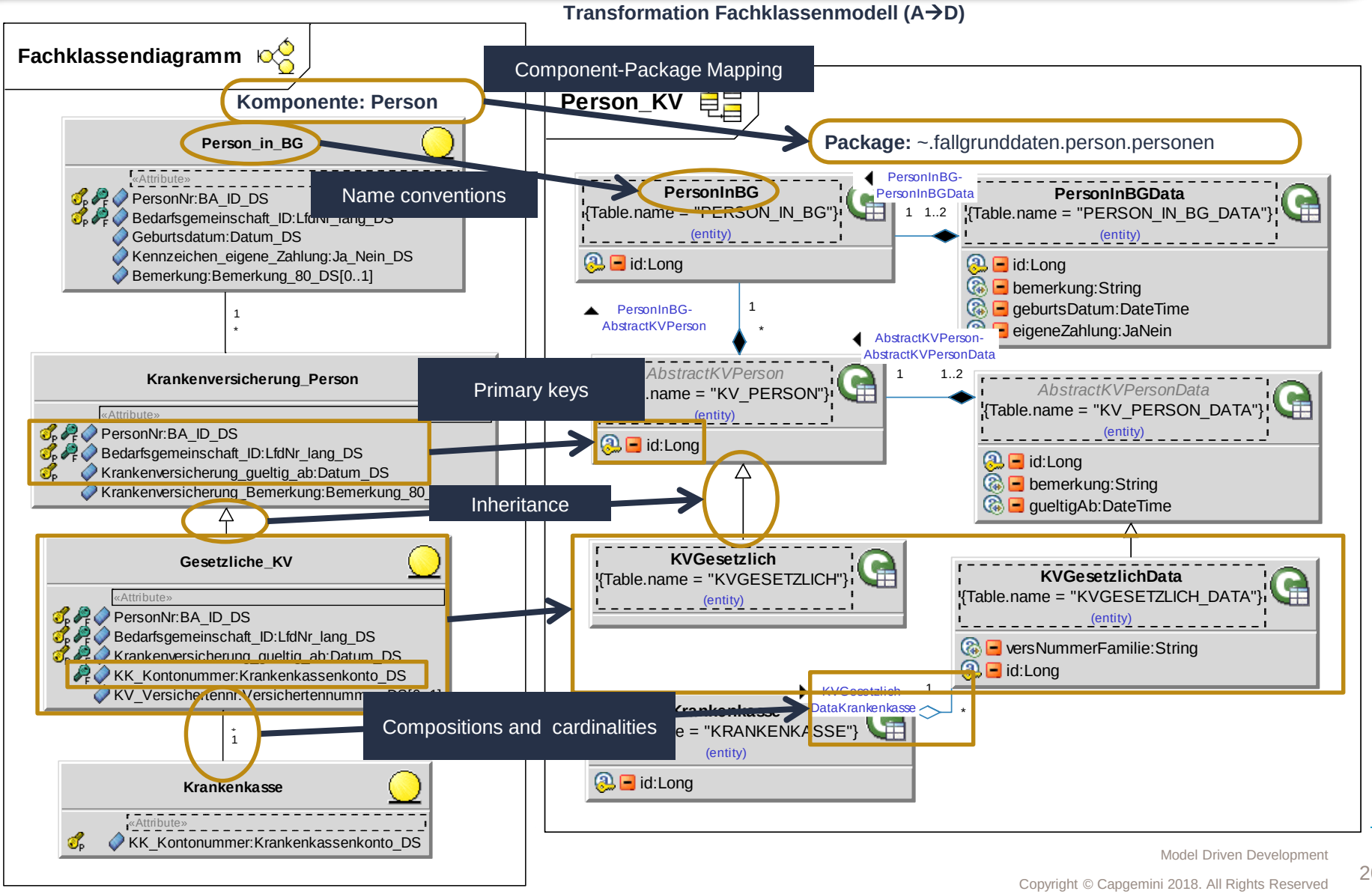
Framework

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

Platform

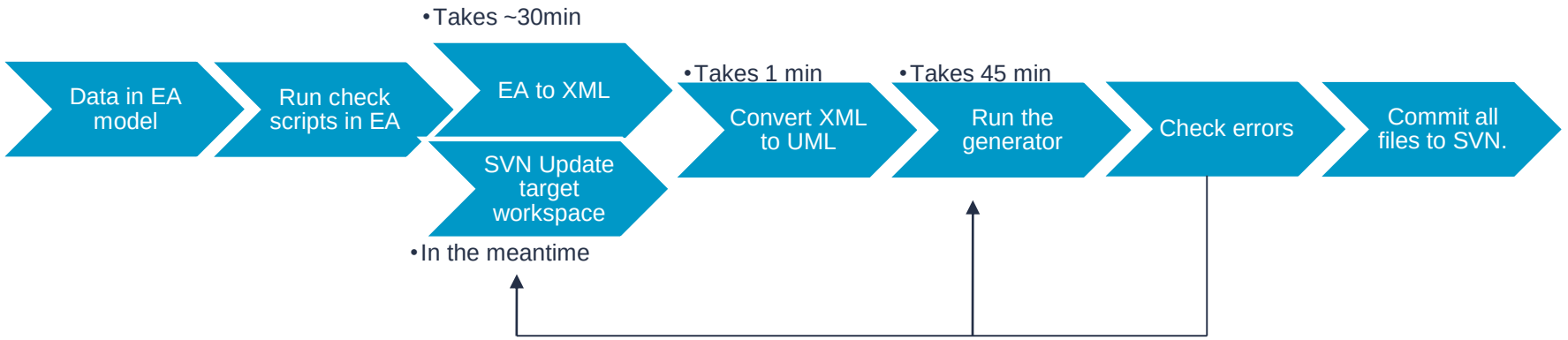
High effort with low return

Example: Mapping from Specification to Design

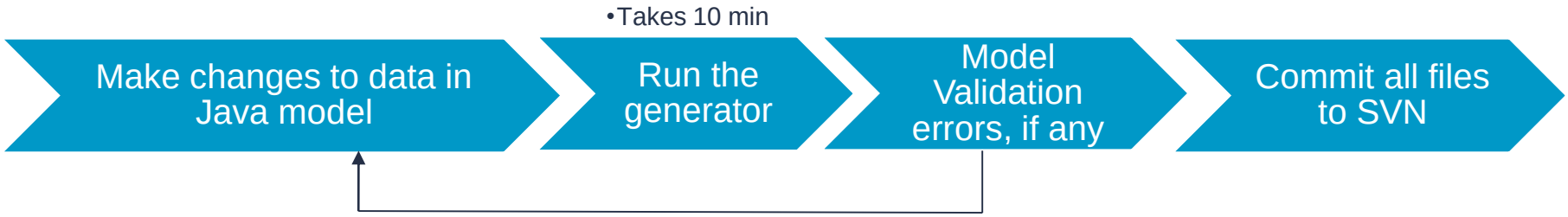


Simplification of the generation process

Old generation pipeline



Simplified new generation pipeline



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

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 an SQL script from it.

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.

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.

Sucess factors for MDD projects

Working Knowledge Management

- Consistent tool chain
- Community support

Customer acceptance

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

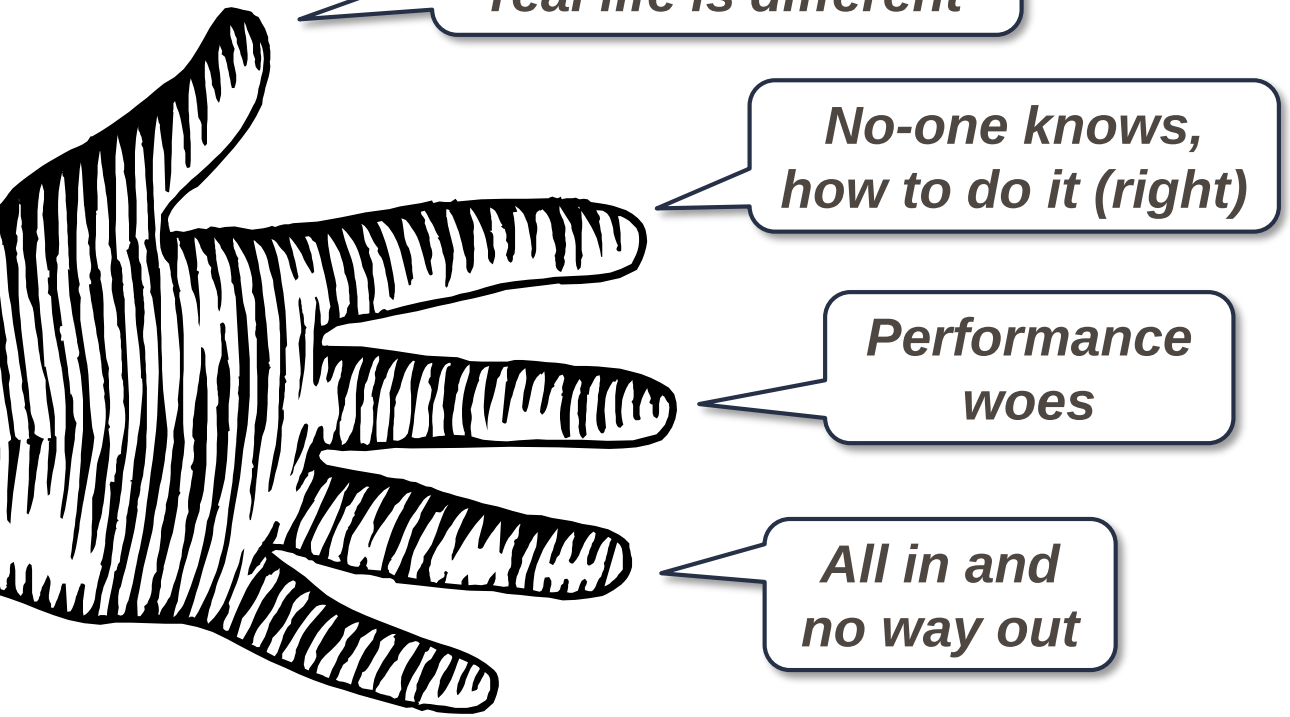
Distinct team roles

- 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

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.*

Dr. Martin Girschick
martin.girschick@capgemini.com

*„If you are interested in Capgemini,
don‘t hesitate to contact me
or hand in the contact form!“*

Appendix

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.

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.