The Power of Natural Language

From ATDD to Lean Modelling

Who knows what **Test Driven Development** is?

Who actually uses **TDD**?

Who knows what
Acceptance Test Driven Development is?

Who actually uses **ATDD**?

Who knows what **Model Driven Software Development** is?

Who actually uses **MDSD**?

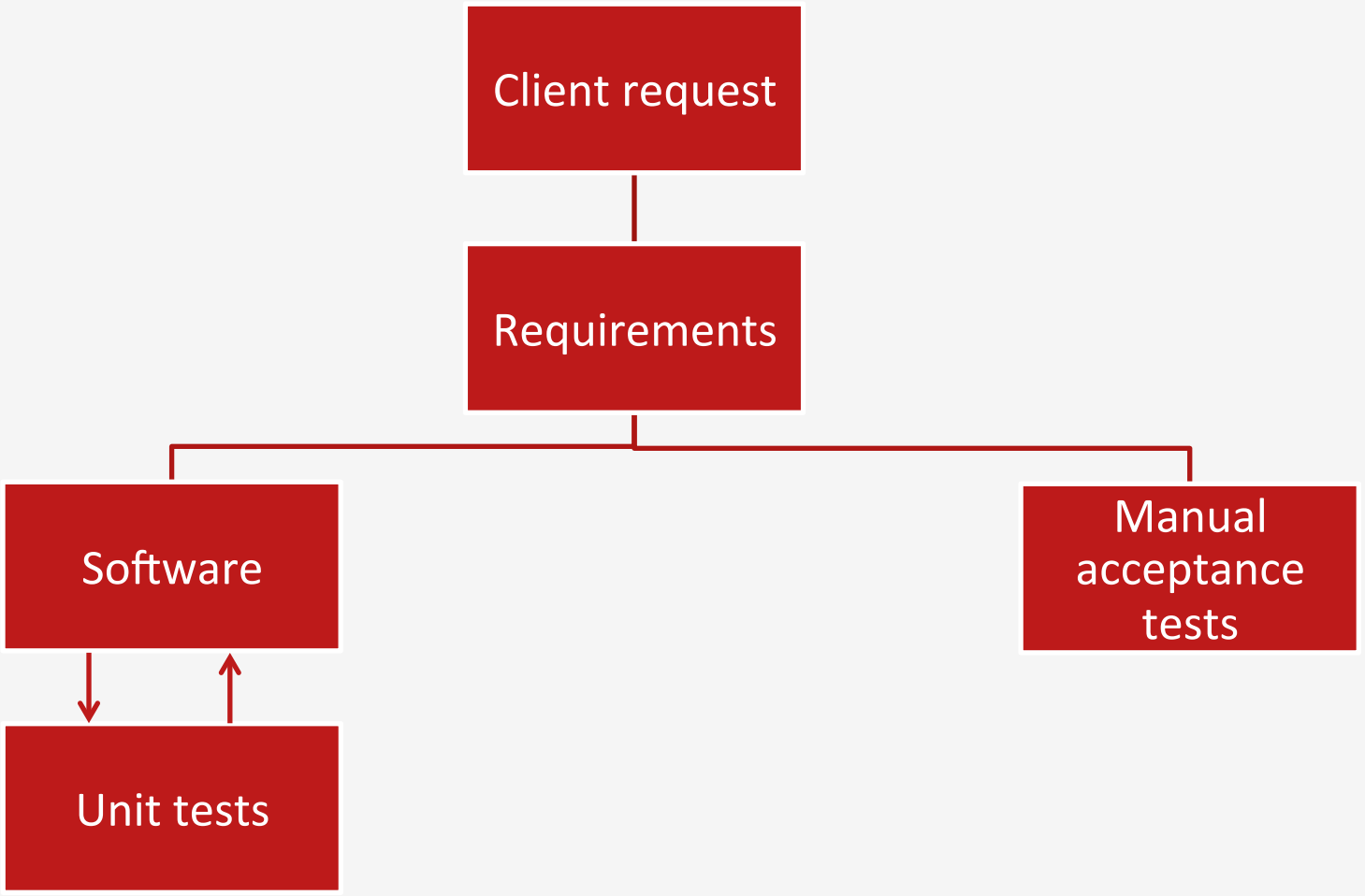
Who actually uses **MDSD with Python**?

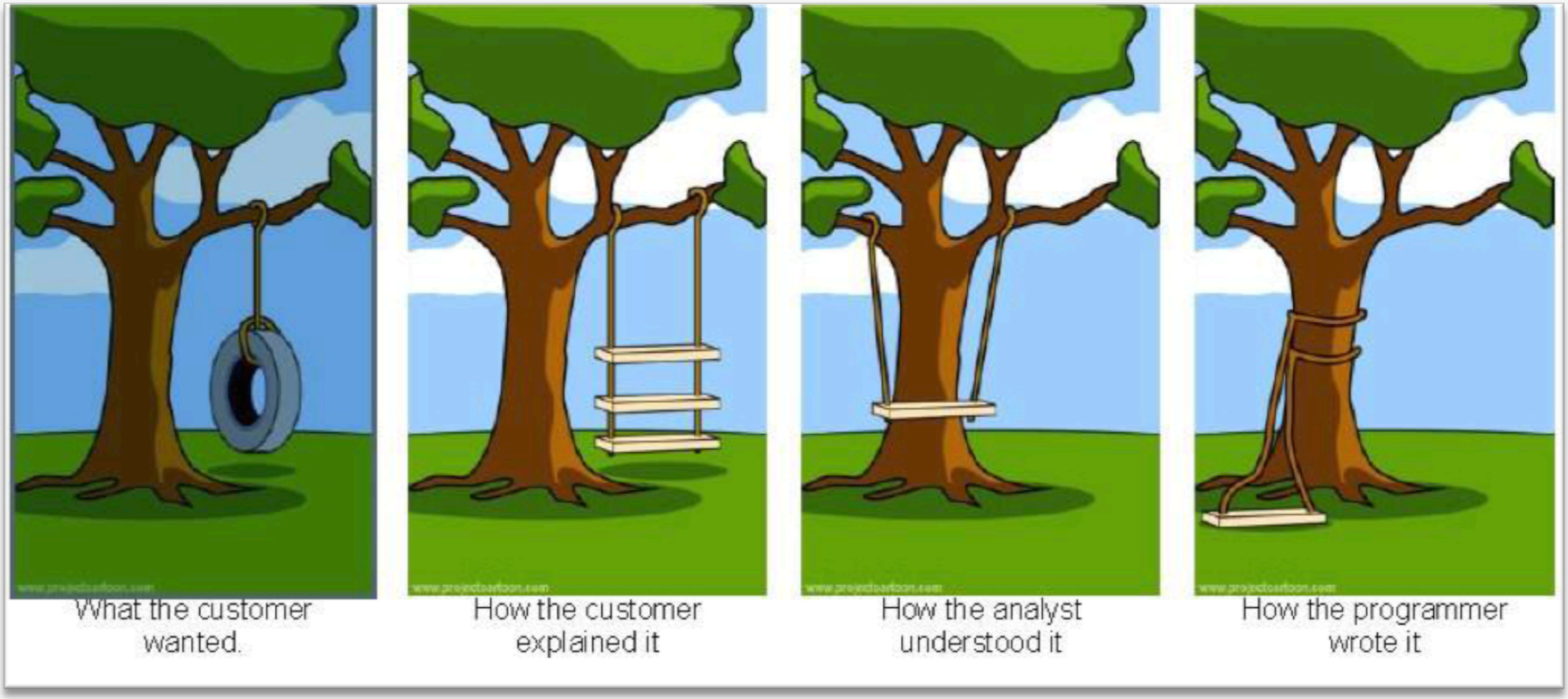
Who knows what **Lean Modelling** is?

Who actually uses **Lean Modelling**?

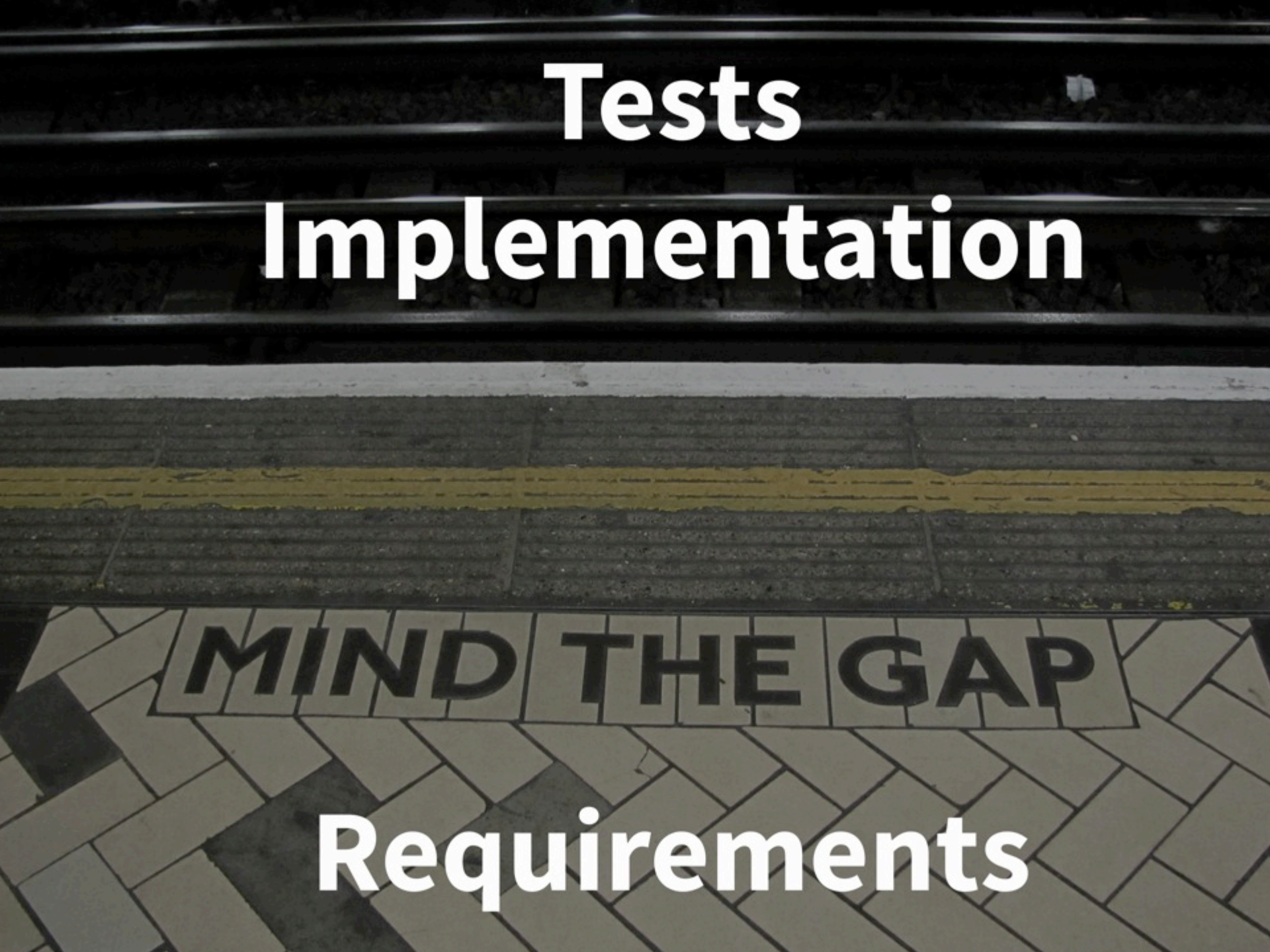
Constantine the African

Classic Approach




What the customer wanted.
How the customer explained it
How the analyst understood it
How the programmer wrote it

Software development is
a **comunication problem!**


Tests
Implementation
MIND THE GAP
Requirements

Solution:

ATDD

Specify **together!**

Client:

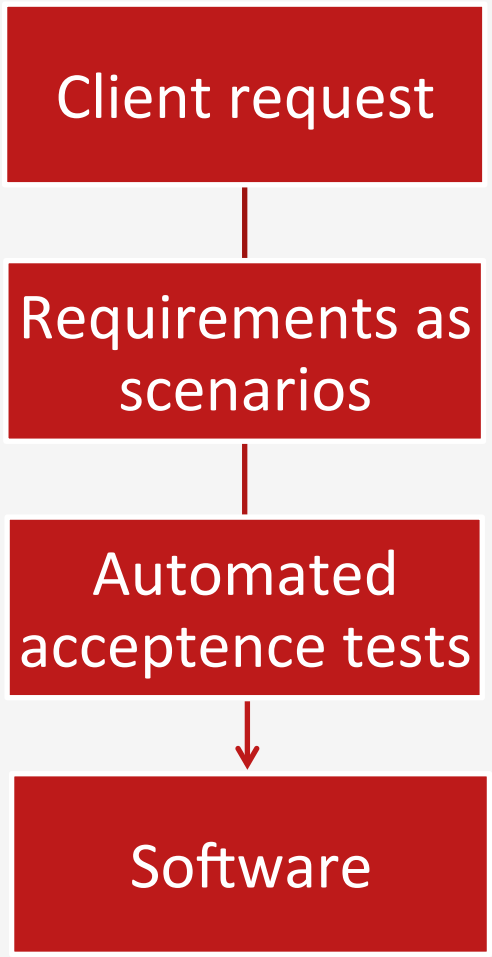
What?

Why?

Developer:
How?

Example driven

ATDD approach



ATDD approach

Client request

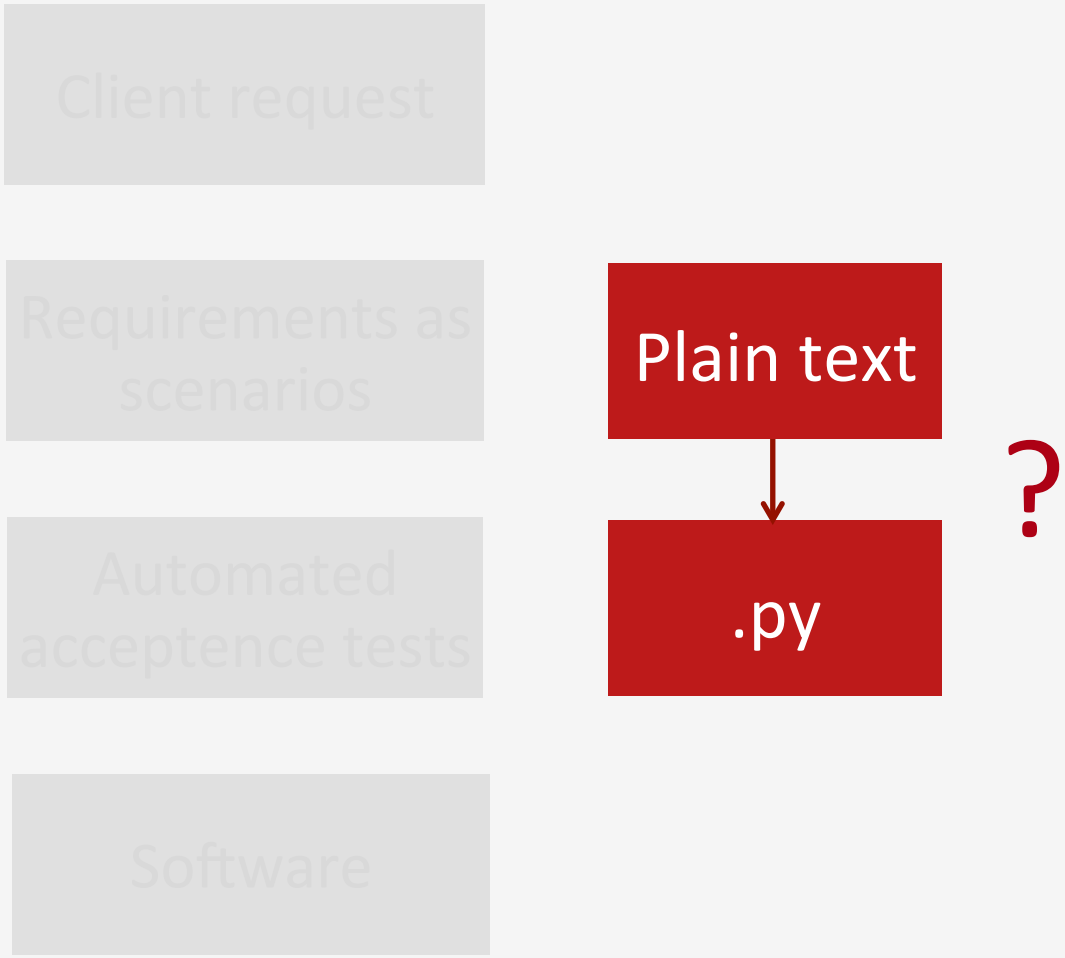
Requirements as scenarios

Plain text

Automated acceptence tests

Software

ATDD approach



NatSpec

Free for opensource projects

Eclipse and **PyDev**

Example

simple shop application

```
Create product cuescience Scoreboard for 299.00
Add cuescience Scoreboard to cart

Assert that we were redirected
Assert total cart item count: 1
Assert total cart price: 299.00

Assert 1 cuescience Scoreboard in cart for 299.00
```

```
Create product cuescience Scoreboard for 299.00
Add cuescience Scoreboard to cart

Assert that we were redirected
Assert total cart item count: 1
Assert total cart price: 299.00

Assert 1 cuescience Scoreboard in cart for 299.00
```

```
class TestSupport():
 #...
 @TextSyntax(
   "Create product #1 for #2“,
   types=["list<str>", "float"]
 )
 def create_product(self, title_words, price):
   title = " ".join(title_words)
   product = Product(title=title, price=price)
   product.save()
```

```
class TestSupport():
 #...
 @TextSyntax(
   "Create product #1 for #2“,
   types=["list<str>", "float"]
 )
 def create_product(self, title_words, price):
   title = " ".join(title_words)
   product = Product(title=title, price=price)
   product.save()
```

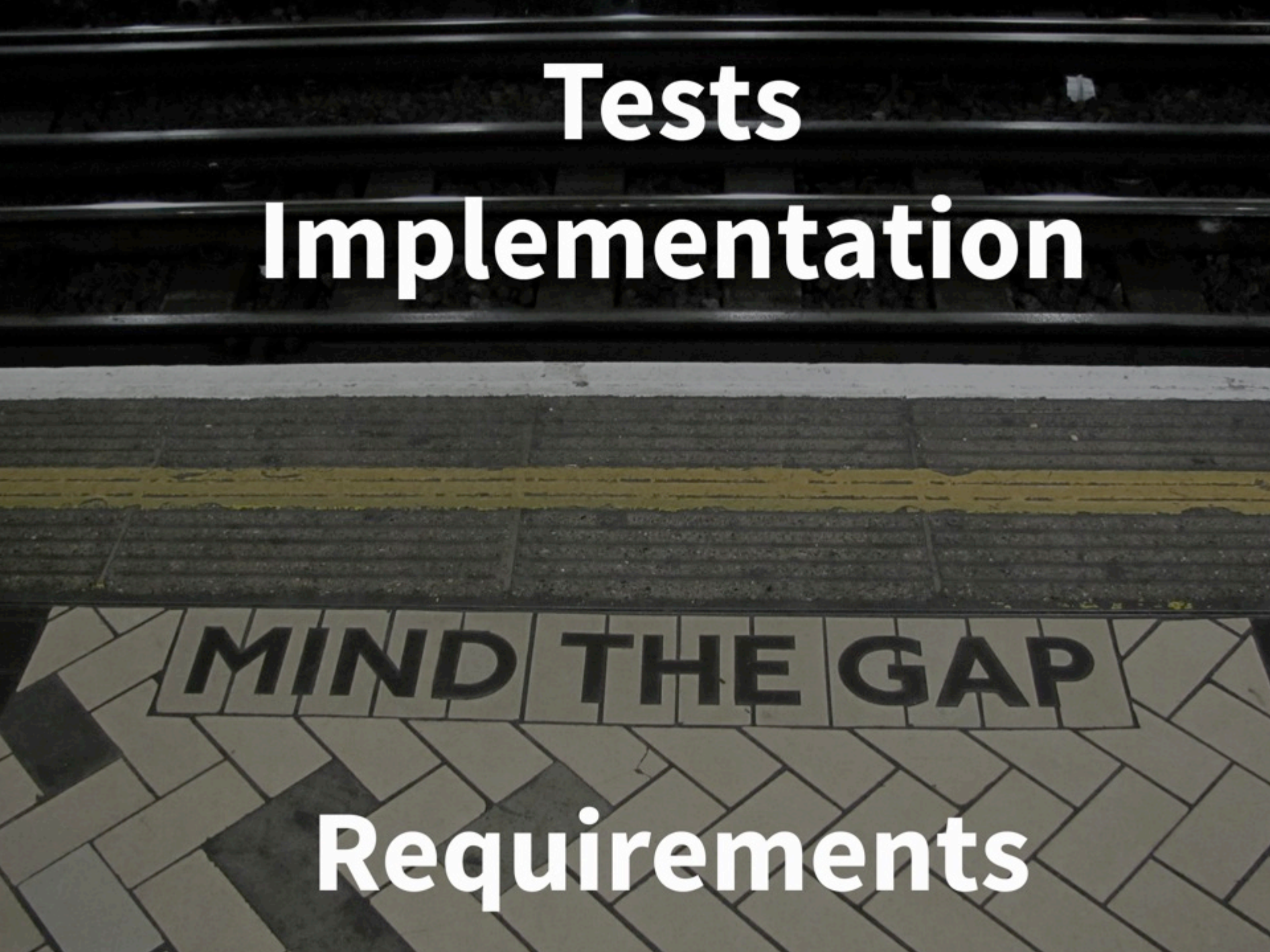
Demo

improved communication

Living artifact

focus on **important** features

progress **measurement**


Tests
Implementation
MIND THE GAP
Requirements

Why just generate **test code**?

Why don‘t generate:

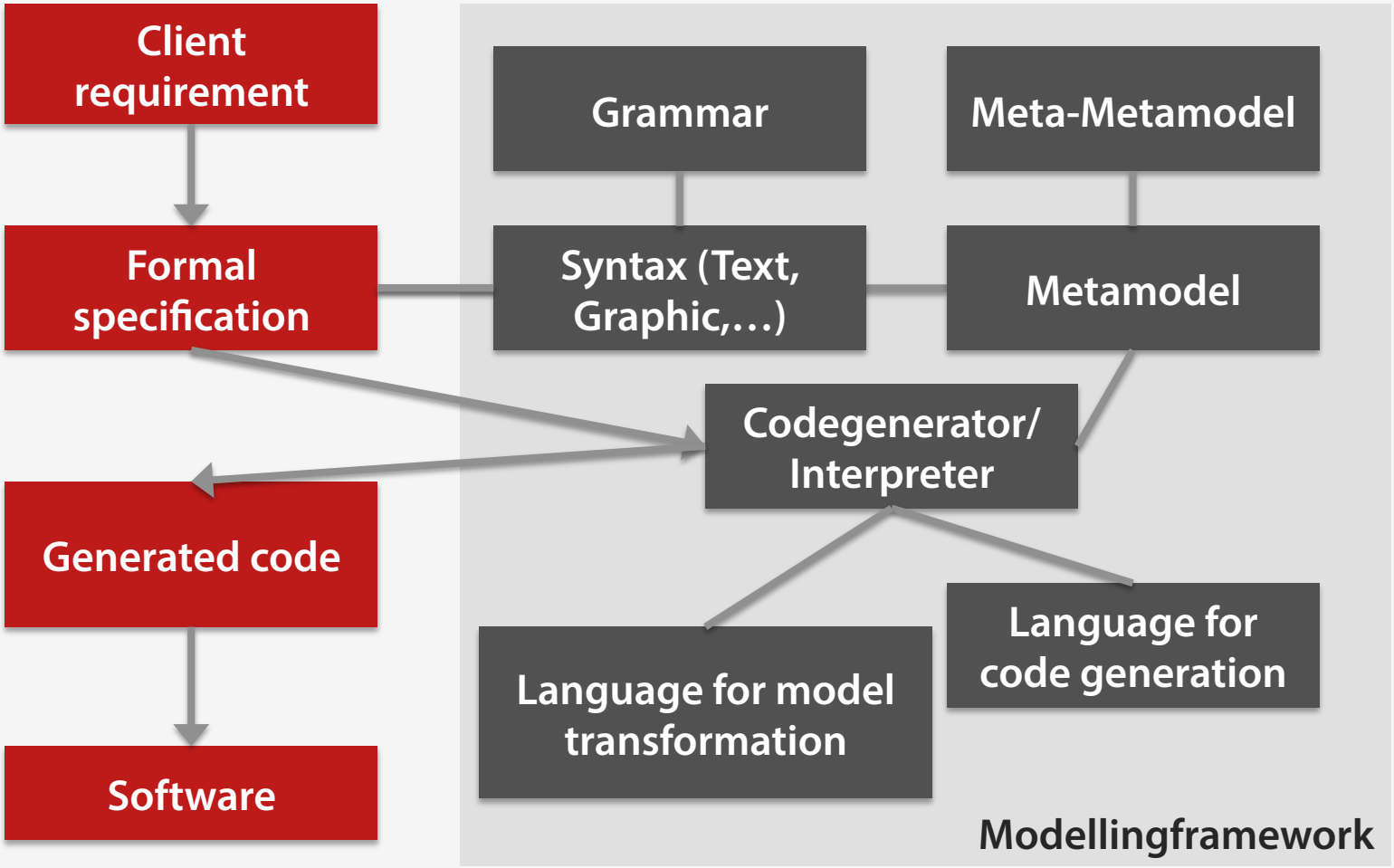
models

forms

wizards

Model Driven Software Development

Classic approach



high **complexity**

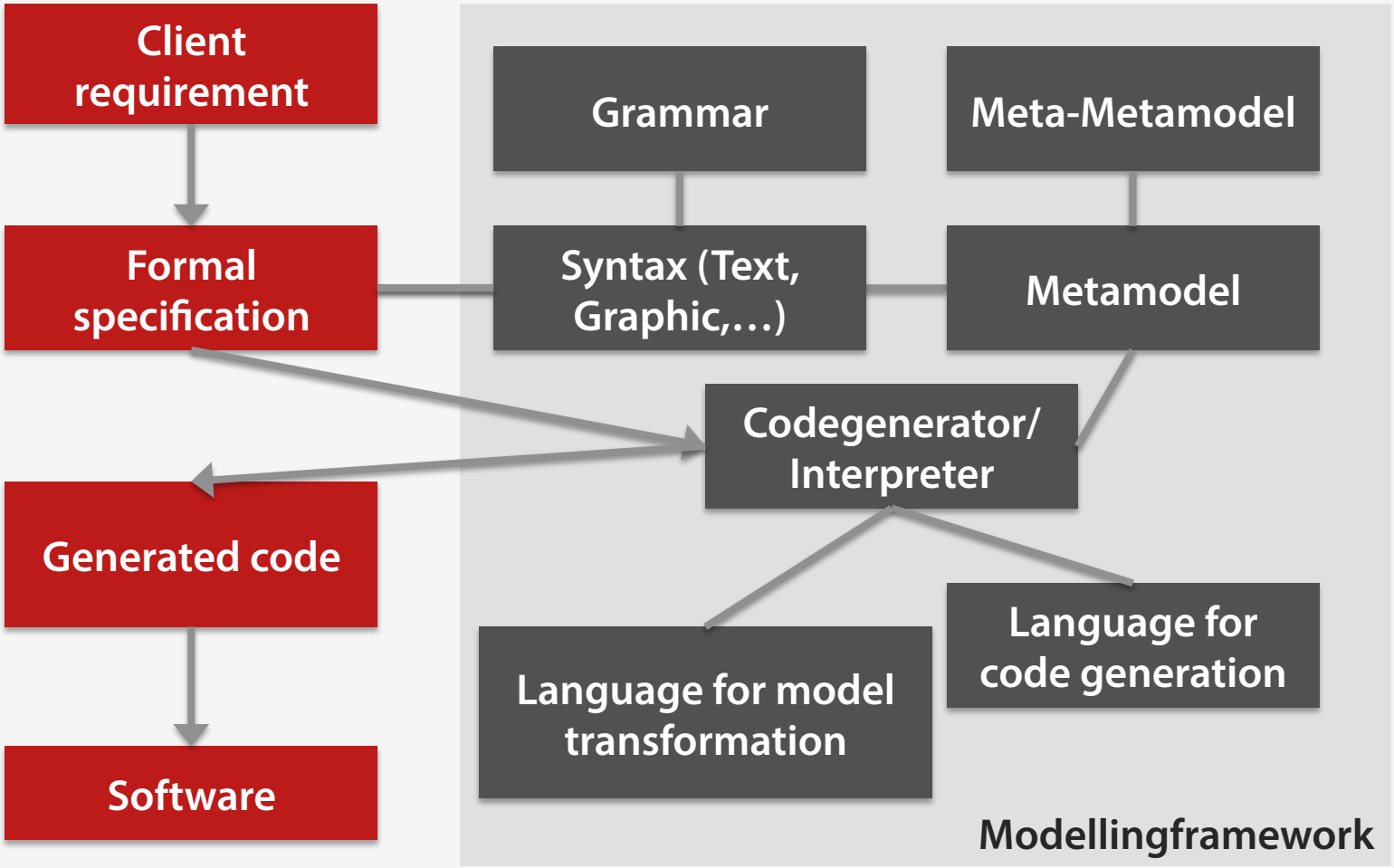
frameworks are **rigid**

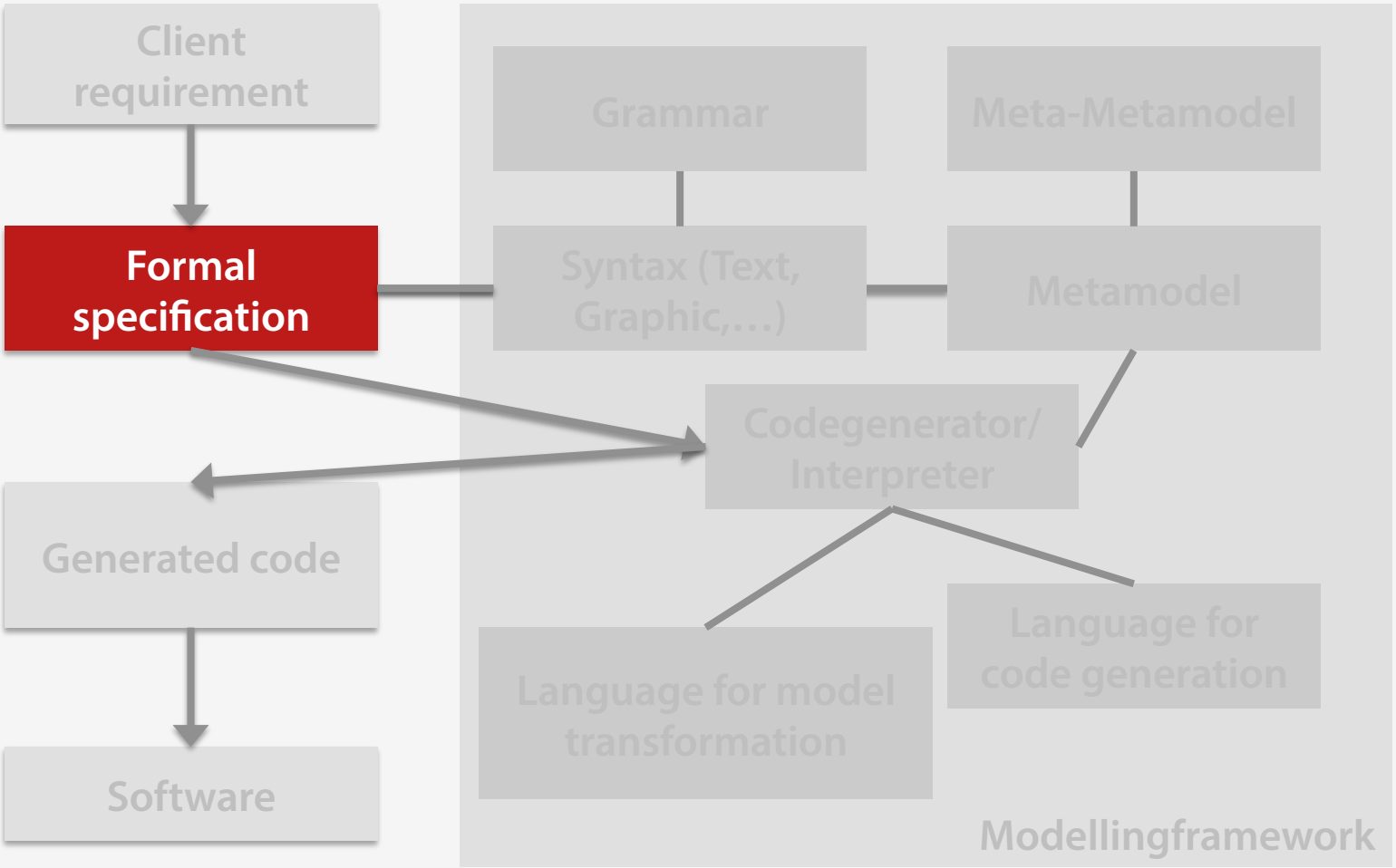

Too **many different**
languages

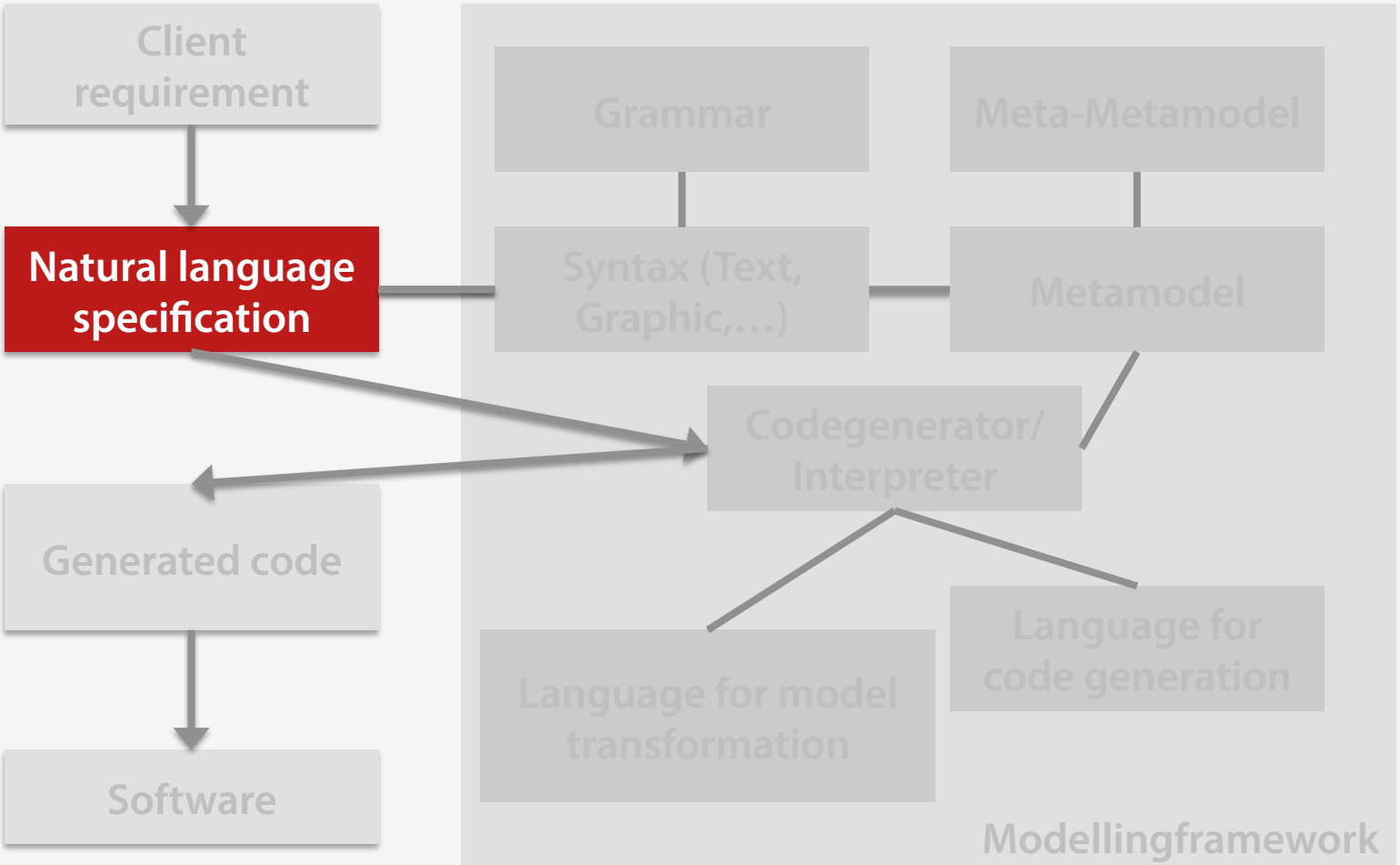
from __future__ import **braces**

SyntaxError: **not a chance**

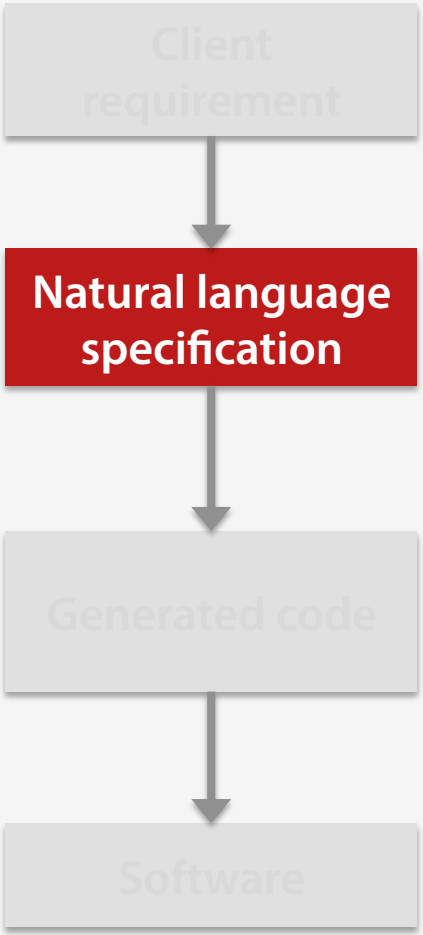
Classic approach




Client requirement
Formal specification
Generated code
Software
Grammar
Meta-Metamodel
Syntax (Text, Graphic,…)
Metamodel
Codegenerator/ Interpreter
Language for model transformation
Language for code generation
Modellingframework


Client requirement
Natural language specification
Generated code
Software
Grammar
Meta-Metamodel
Syntax (Text, Graphic,…)
Metamodel
Codegenerator/ Interpreter
Language for model transformation
Language for code generation
Modellingframework

Lean modelling



NatSpec

Demo

generate **n** artifacts
from **one** spec

e.g.:

models

rest api

javascript model

Next steps & Questions?

- Try out **NatSpec4Python** (Update site)
  - *http://www.devboost.de/natspec4python/update_trunk/*
- Try out **django-leanmodelling**
  - *https://github.com/iljabauer/django-leanmodelling*
- Try out **python-example-project**
  - *https://github.com/iljabauer/natpsec-example-python*
- **Need help?**
  - *github.com/iljabauer/*
  - i.bauer@cuescience.de
  - christan.wende@devboost.de (General NatSpec questions)

cuescience
innovations in cuesports