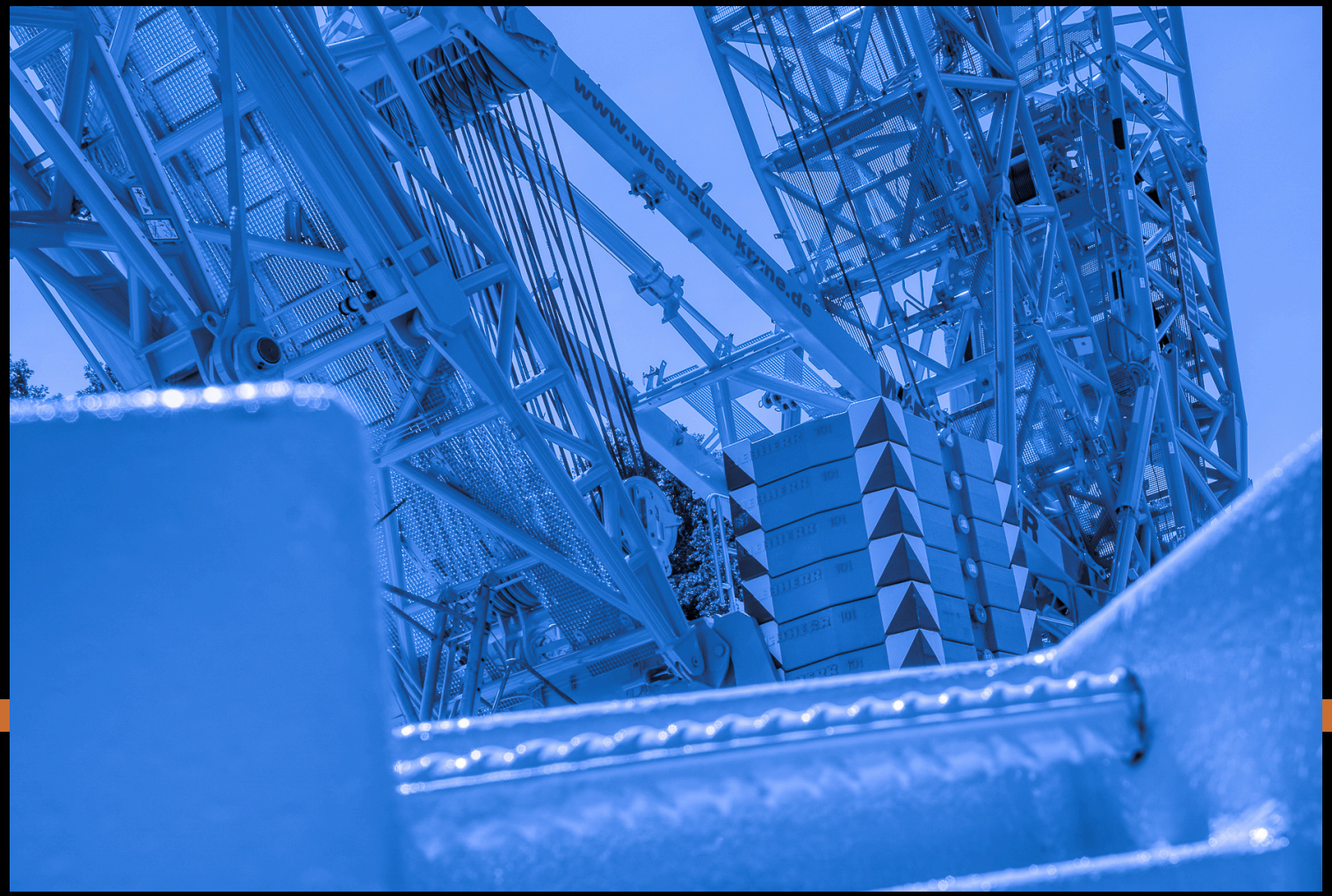
How Domain-Specific Languages Improve Software Quality



Dr. Markus Völter

HEISENBUG 2021

voelter.de
voelter@acm.org
@markusvoelter

The Problem

Subject Matter Experts are 2nd class

Subject matter experts, or SMEs, own the knowledge and expertise that is the backbone of software.

But too often this rich expertise is not captured in a structured way and gets lost when translating it for software developers who then analyze, interpret and understand it before writing code.

With the rate of change increasing, time-to-market shortening and product variability blooming, this approach is increasingly untenable. It causes delays, **quality problems** and frustration for everybody involved.

We advocate for adopting a mindset that puts subject SMEs directly in control of "their" part of the software and lets developers focus on their core skill, software engineering.

Here is how we achieve it:

DSLs for SMEs

Automate DSL to code transformation

Let devs build DSLs, IDEs, trafos and robust platforms

SUBJECT MATTER FIRST!

Here is how we achieve it:

DSLs for SMEs

Automate DSL to code transformation

Let devs build DSLs, IDEs, trafos and robust platforms

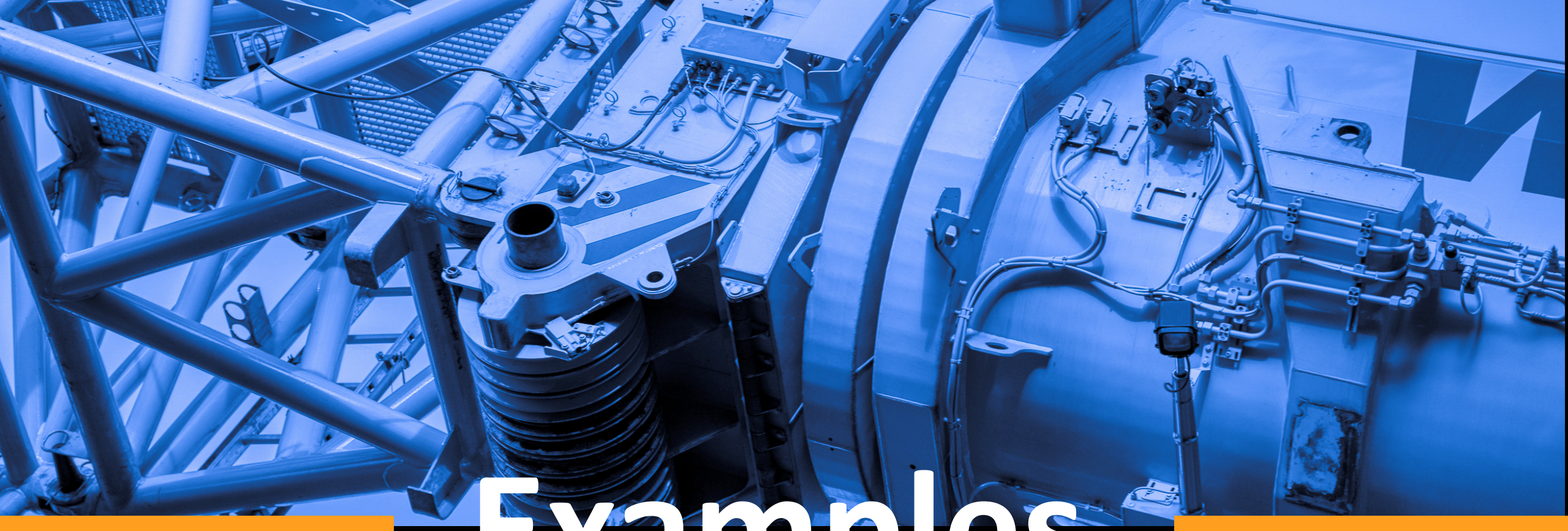
Here is how we achieve it:

DSLs for SMEs

Automate DSL to code transformation

Let devs build DSLs, IDEs, trafos and robust platforms

What does this mean for SOFTWARE QUALITY



Examples

Tax, Healthcare, Systems Engineering

Tax Calculation

10,000 fields and formulas
1,000 validation rules
100 SMEs
10 years back
significant yearly changes

DATEV

Structure oriented
along the legal text

```
Gruppe für Bierdeckelsteuer

  private tax SteuerAbsetzbar = min(Steuer, 2000)

    private tax Steuer = EssenSteuer + GetränkeSteuer

      private tax EssenSteuer = EssenNetto * 15%

        private tax EssenNetto : real

      private tax GetränkeSteuer = GetränkeNetto * 7%

        private tax GetränkeNetto : real
```

+

Iterate over lists, count, sum
Monthly and yearly data structures
Time series and operations on them (Kf TT)
Queries in order to construct derived data
Data tables for parameter sets

Video von der OOP 2021
https://youtu.be/q56wzLQkEho

Salary Calculation


```
val beitragProzentsatzArbeitnehmer: %%% = 1.50%
val beitragProzentsatzArbeitgeber: %%% = 1.50%
```

Percent Types

```
daten ArbeitslosenversicherungStamm {
  beitragsgruppe            : arbeitslosenversicherungBeitragsgruppe
  unternehmenRechtskreisOst : boolean
}
```

```
ergebnis [monatlich] ArbeitslosenversicherungErgebnis {
  arbeitgeberBeitrag  : €€€
  arbeitnehmerBeitrag : €€€
}
```

Currenty Types

```
fun getSvBruttoGekürzt(rechtskreisOst: boolean, svBrutto: €€€): €€€
  =
```

rechtskreisOst	svBrutto > bbgOst	svBrutto > bbgWest	wert: €€€
true	true		bbgOst
false		true	bbgWest
			svBrutto

Decision Tables

http://voelter.de/data/pub/PayrollDSL.pdf

Buchkapitel Case Study

Digital Therapeutics

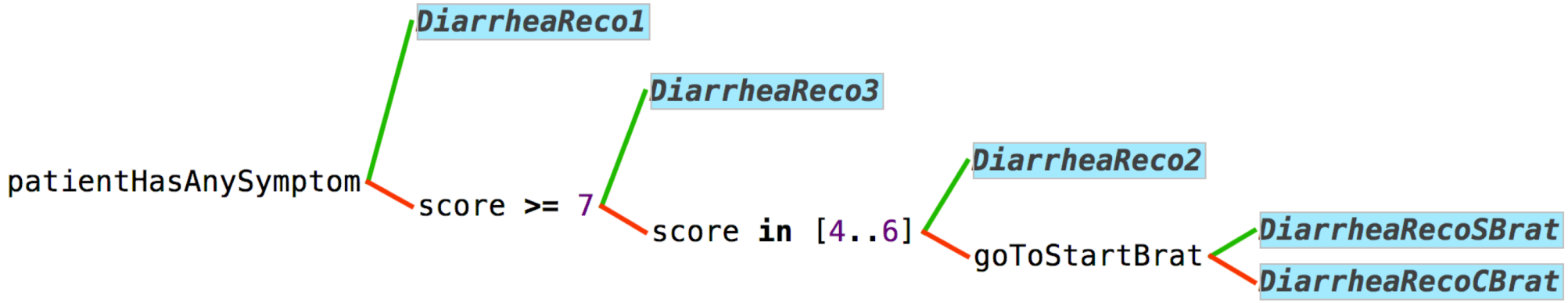
VOLUNTIS

dozens of treatmets apps per year

```
decision table BpScoreDecisionTable(sys: bpRange, dia: bpRange) =
```

		dia					
		<= 50	[51..90]	[91..95]	[96..100]	[101..109]	>= 110
sys	<= 90	1	1	3	4	5	6
	[91..140]	2	2	3	4	5	6
	[141..150]	3	3	3	4	5	6
	[151..160]	4	4	4	4	5	6
	[161..179]	5	5	5	5	5	6
	>= 180	6	6	6	6	6	6

```
decision tree DiarrheaStoolsDecisionTree(score: DiarrheaStoolsOverBaseline,
                                         patientHasAnySymptom: boolean, goToStartBrat: boolean)
```



http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

Paper in SoSym Journal

Social Insurance

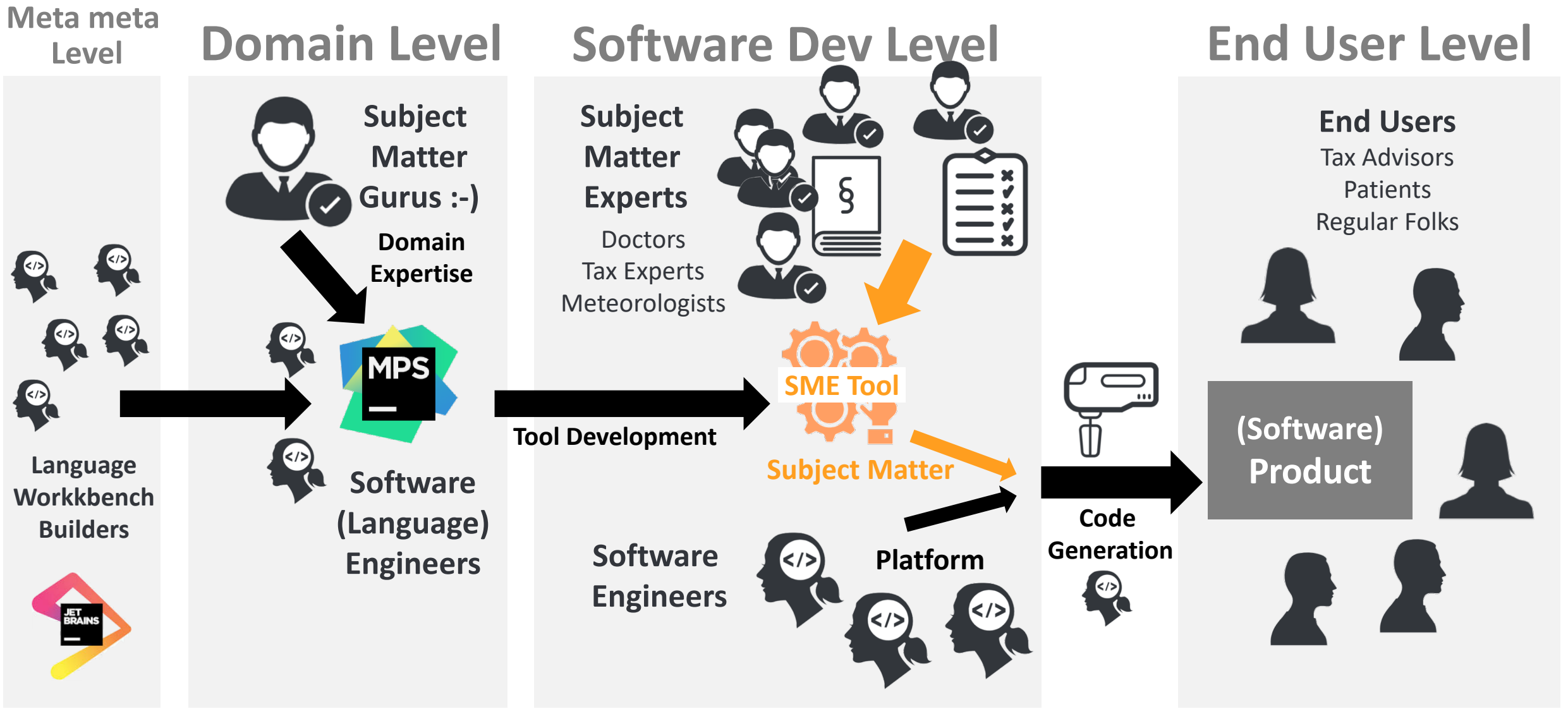
Mix between form style and „real“ language.

Yellow parts are scaffolding and cannot be removed.

```
Name: UVG-Leistungen
Unterhaltsvorschuss
Zeitangabe:          laufend
Häufigkeit:          monatlich einmal
Leistungskontext:
Leistungsart:        Leer
Zählart: uvg
Anspruch Beginn: Anfang – Unbegrenzt: junger Mensch.geburtsdatum
Anspruch Ende: 01.01.1800     – 31.12.9999     : min(junger Mensch.geburtsda
                                                        12 Jahre
Zeitraum für Berechnung: Anfang – Unbegrenzt: {standardzeitraum, standardz
zweckgebundene Leistung: ☐
dem Grunde nach:         ☐
Zeitraumbezogene Daten
nullwerte Anzeigen : boolean = 01.01.1800   – 31.05.2016   : true
                               01.06.2016   – Unbegrenzt: false
berechnungsart : berechnungsarttyp = 01.01.1800   – 31.12.9999   :
Bezugsobjekte: << ... >>
Attribute:
bemerkung     : string wird validiert
antragsdatum : Datum
Nebenberechnungen
Name: Kindergeld für vollen Monat
(01.01.1800   – 31.12.9999   )
Rechnungsart:  wenn: wird geboren mit junger Mensch als person dann voller
               sonst: taggenau
               Begünstigtenprinzip: ☐
  monatswert = Kindergeld 1. Kind

zwischenergebnisse = [<< ... >>]
endergebnis = monatswert
```

Big Picture: How does knowledge get into software



Big Picture: How does knowledge get into software

Responsibilities (daker shade = care more)	Subject Matter Experts	Software Engineers
Caring about the intricacies of each subject matter instance		
Deternining what is "correct" in terms of subject matter		
Writing and executing tests, the notion of coverage		
Use Arithmetic and conditional operators, case distinction, etc.		
Understand the core conceptual abstractions of a domain		
Complexity, Dependencies, Modularity, Cohesion		
Conceptual consistency (if needed)		
Finding and then building new abstractions		
Develop Languages, Generators and Tools		
Scalability, Performance, Security, Robustness, Availability		
Develop and run Build-, Test- and Deployment Pipelines		

Useful for the following Domains

Useful for the following Domains

Large and complicated subject matter	Experts that understand the subject matter	High rate of change within the domain	Long-lived domain or large variety within domain

Insurance [Product Definition]

Healthcare [Treatment algorithms]

Public Administration [Tax, Public Benefits]

Law and Legal [Contract Modeling]

A CAD program for the knowledge worker

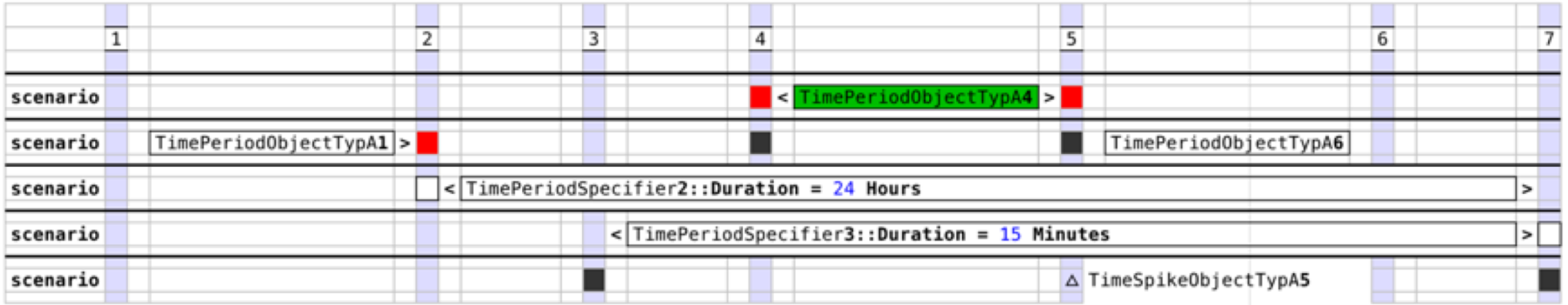
A compiler for requirements

Tachographs

```
isore rest rule TimingPattern_07_ObjectsAcrossRows_Modified_TandA_897_1Iteration {
  description: 1381R3.ARCHD.0609.TimingPatternLanguage_TestScenarioMap.xlsx

  parameters: TimePeriodObjectTypA4

  patterns:
```




```
}
```

database databaseOneElementAcrossRows

Type	Begin	End	Duration	Occurence
eTimePeriodObjectTypA	500	550	50	

database databaseOneAndMoreIterationsHappy

Type	Begin	End	Duration	Occurence
eTimePeriodObjectTypA	50	100	50	
eTimeSpikeObjectTypA				86000
eTimePeriodObjectTypA	86020	86030	10	

Testing

Testing

```
Gruppe für Bierdeckelsteuer

  private tax SteuerAbsetzbar = min(Steuer, 2000)

    private tax Steuer = EssenSteuer + GetränkeSteuer

      private tax EssenSteuer = EssenNetto * 15%

        private tax EssenNetto : real

      private tax GetränkeSteuer = GetränkeNetto * 7%

        private tax GetränkeNetto : real
```

Automatic derivation of test structure from calculation schema

\+ Automated coverage measurement for models and languages

Direct Execution in the MPS IDE

multi test case TestBierdeckelsteuer [fail] {
  <no fiscalYear>
  Gruppe: Bierdeckelsteuer
  Knoten: <no taxUnderTest>
  ☐ Integrationstest

	case unter2000 [ok]	case über2000 [failed]
Bierdeckelsteuer		
SteuerAbsetzbar	[ok] ? 8,90	[ok] ? 2000
Steuer	[ok] ? 8,90	[ok] ? 2200
EssenSteuer	[ok] ? 7,50	[ok] ? 1500
EssenNetto	50,00	10000
GetränkeSteuer	[ok] ? 1,40	[was: 700.00] ? 1,40
GetränkeNetto	20,00	10000

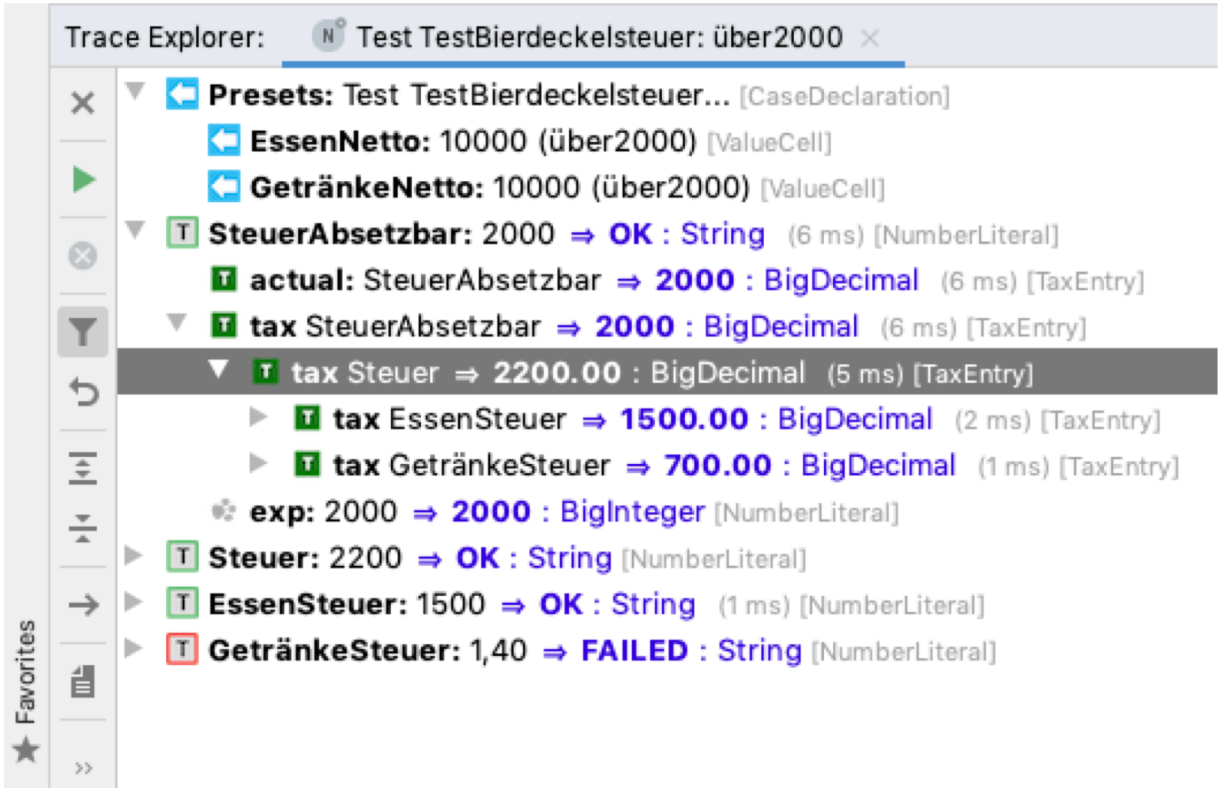
}

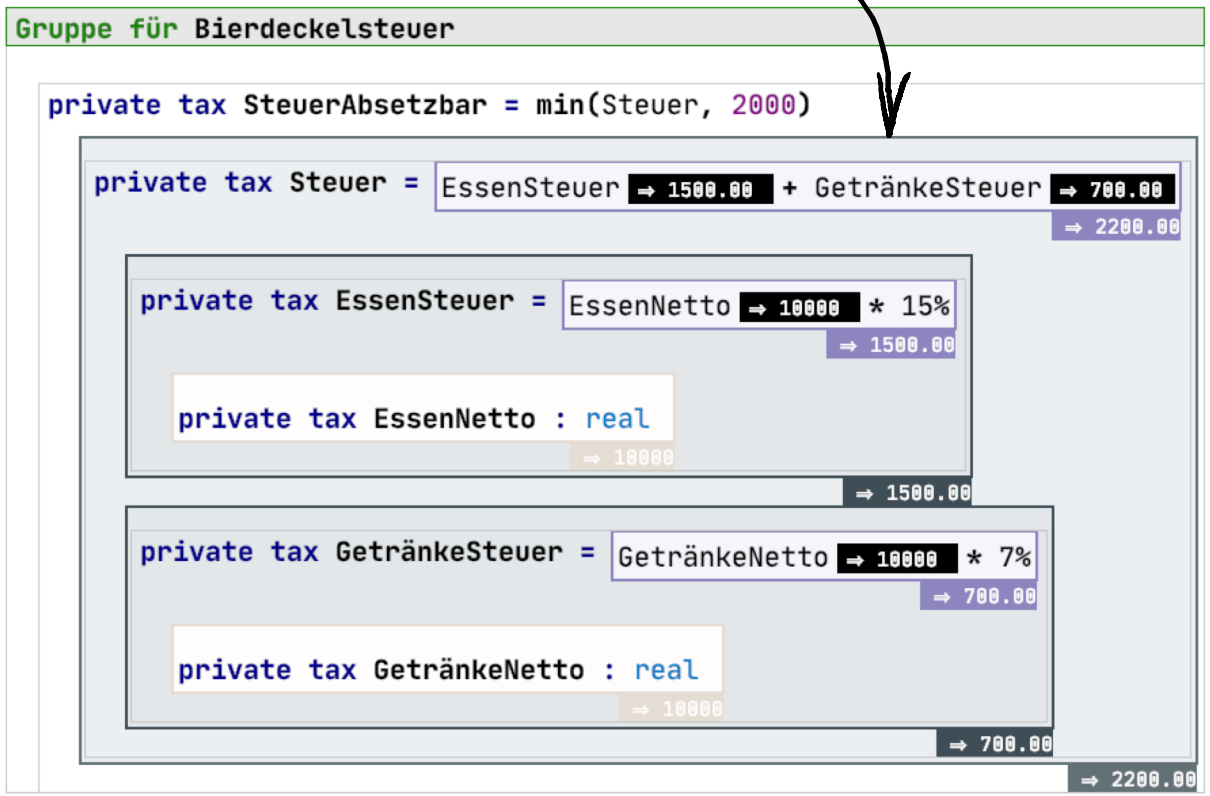
Intuitive capture of relevant data constellations and test expectations

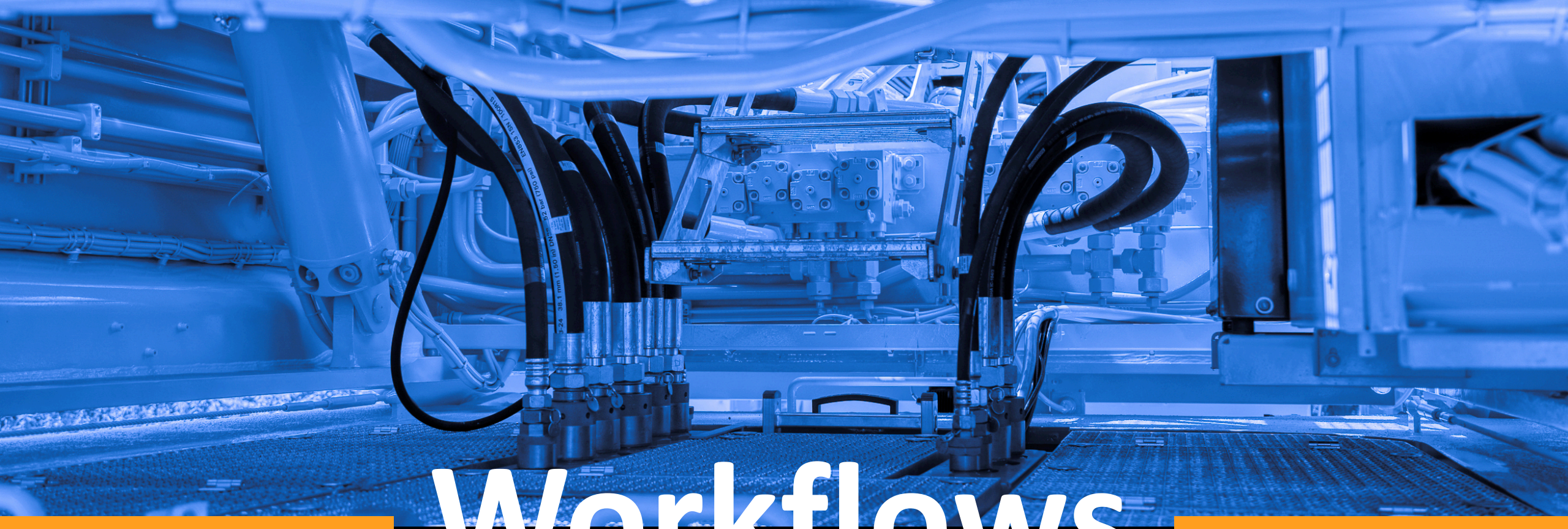
Testing

Tracing in the IDE

Overlay of values over the calcualtion schema





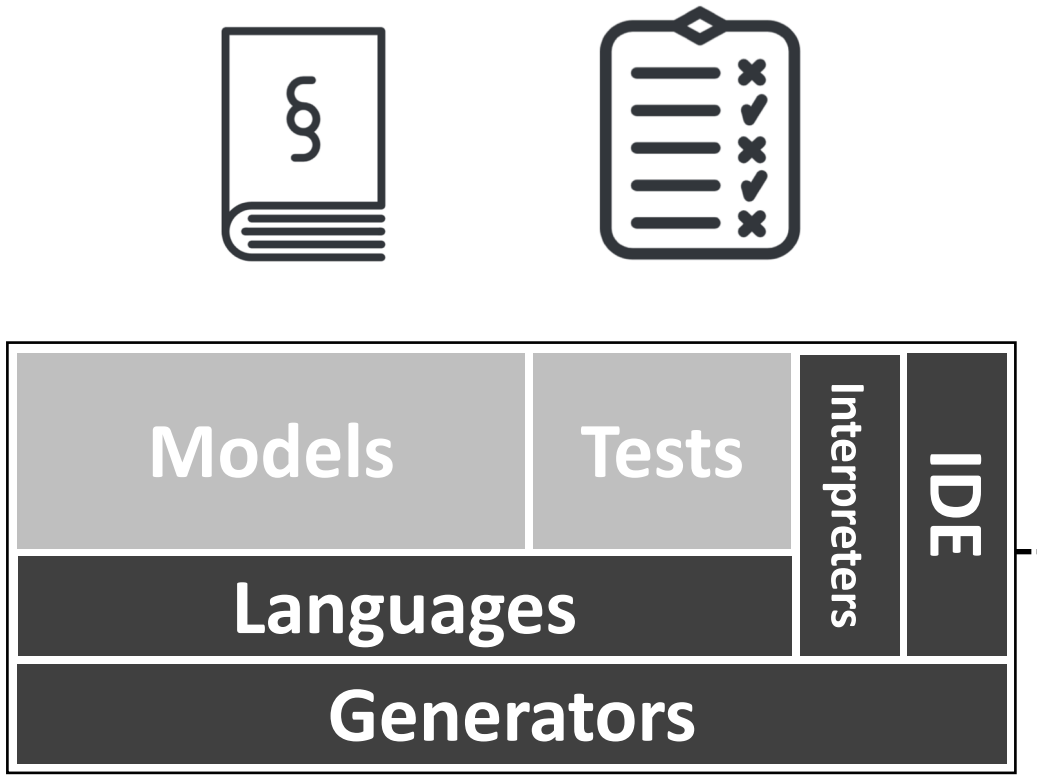


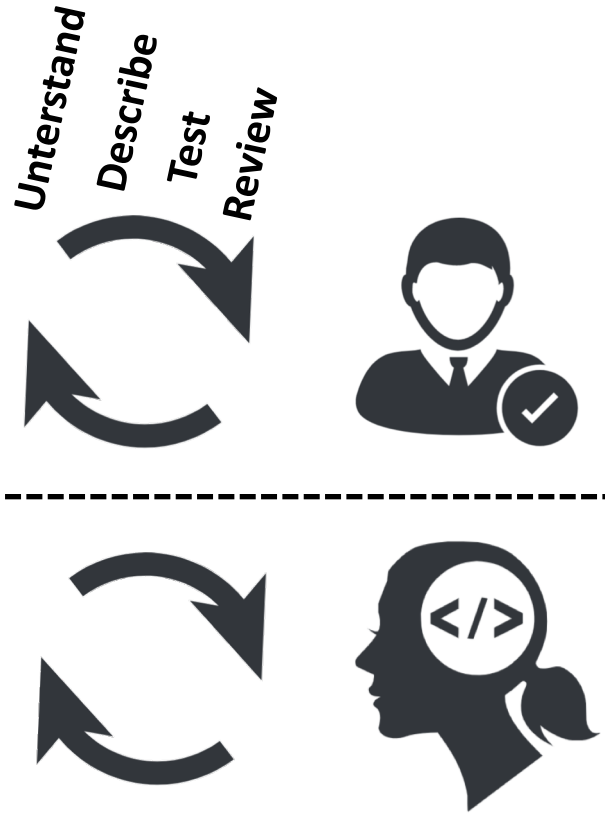
Workflows

Teams, Generation and DevOps

Subject Matter Workflow

Specification and test
of calculation rules



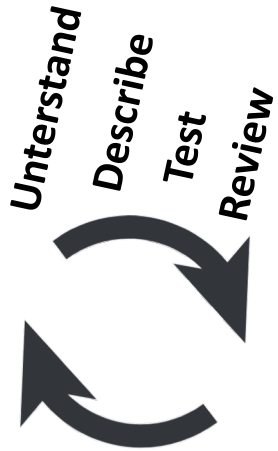


Technical Workflow

Efficient and high-quality
implementations for data center
and on-premise apps

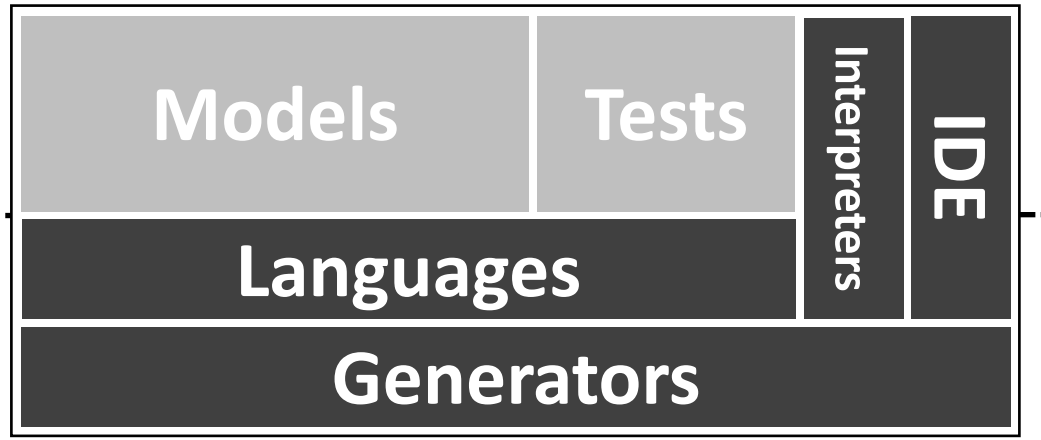
Subject Matter Workflow

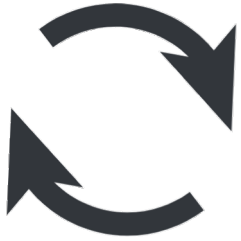
Specification and test
of calculation rules



Collaboration

based on well-defined
and executable artifacts

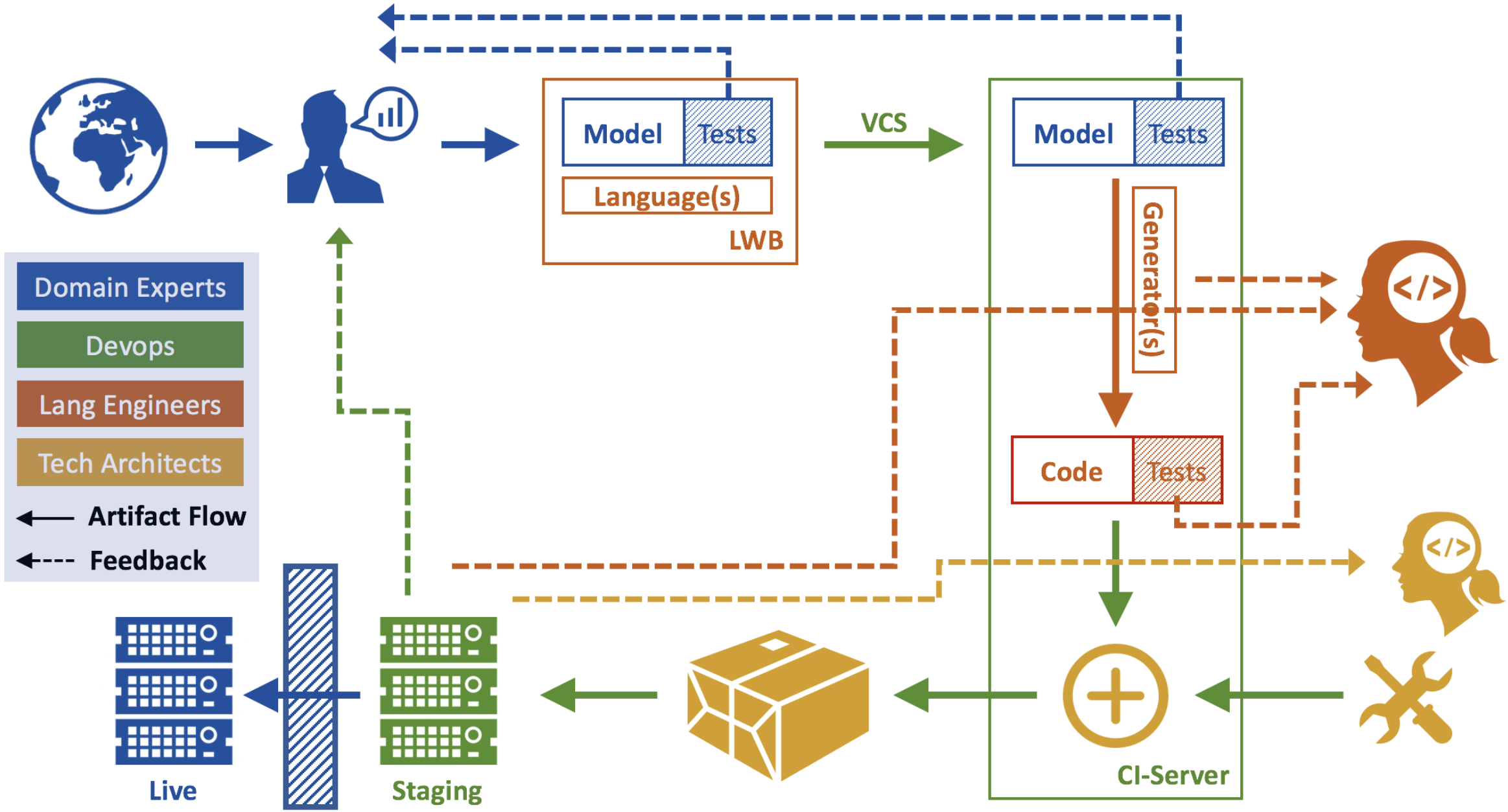




Technical Workflow

Efficient and high-quality
implementations for data center
and on-premise apps

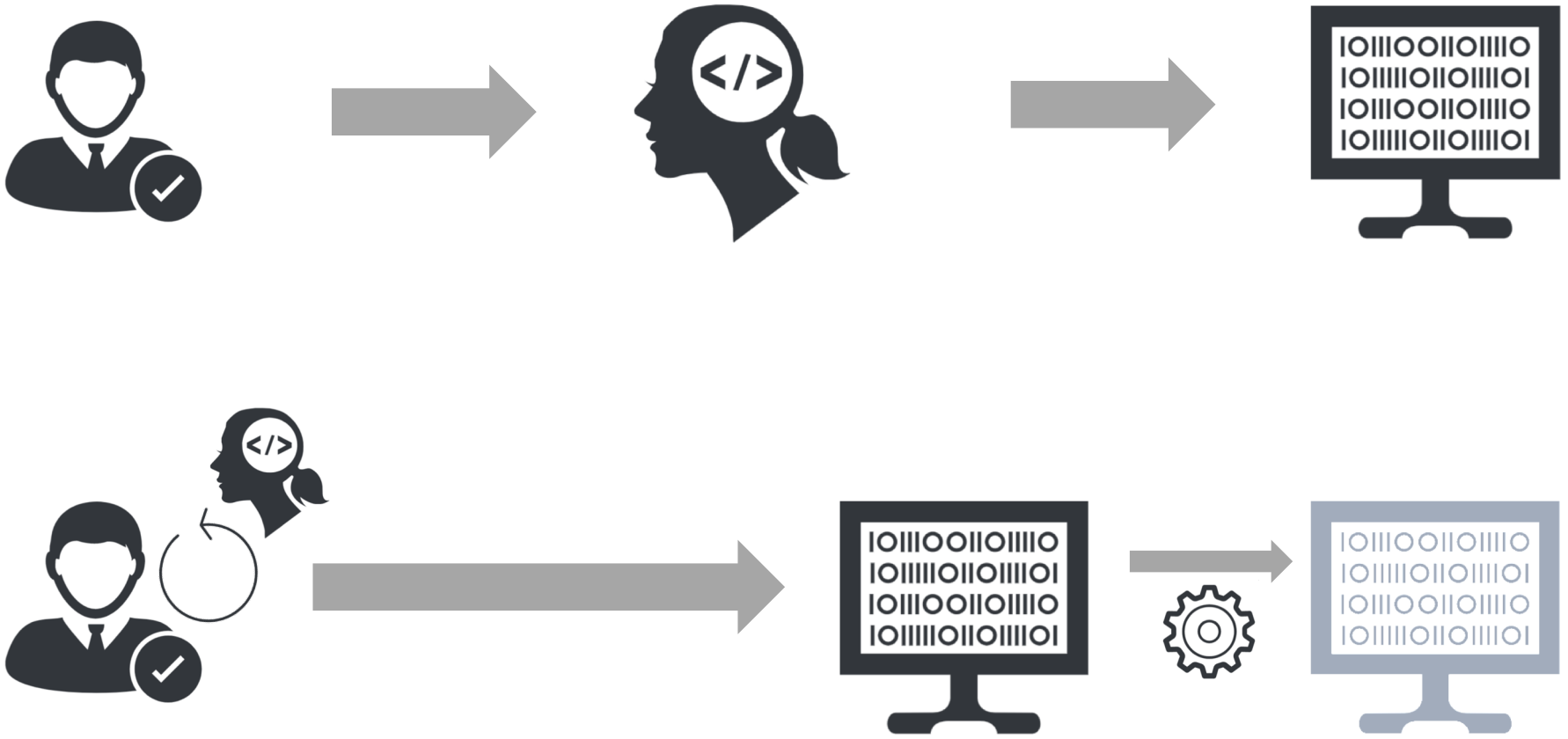
DevOps Perspective



Quality

Why DSLs help with software quality

Direct “programming“ by SMEs avoids misunderstandings



Higher Level of Abstraction avoids low-level errors

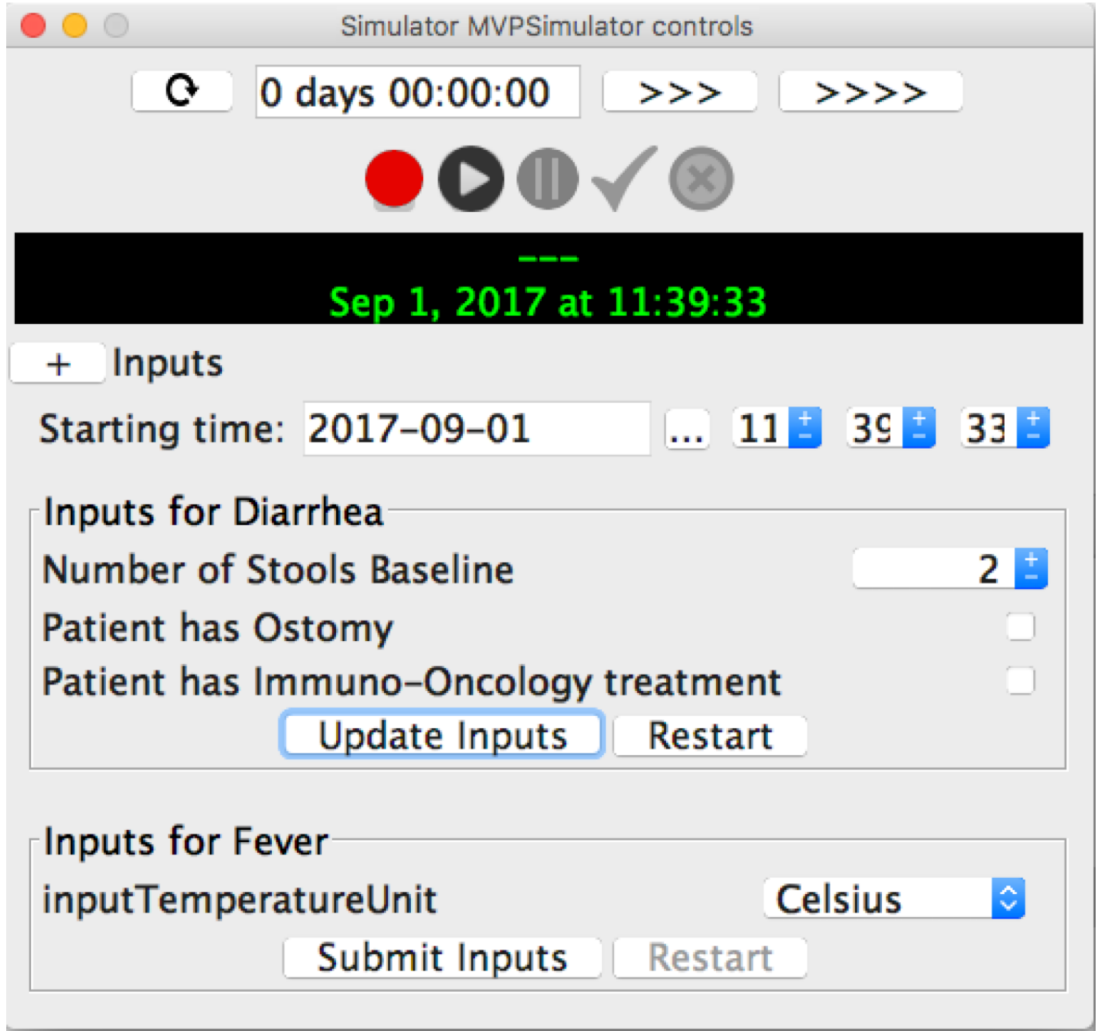
```
TaxGroup_t *G__1699215591_Einzeln_1_addGroupInstance_Aspekte_Erste_1_Einzeln_1
          (TaxList_t *list, uint32_t index, bool tryFindInstanceFirst)
{
  TaxGroup_t *group = list->group;
  VALUE indexValue = NULL;
  TaxGroup_t *child = NULL;
  DLListElement_t *element = NULL;
  if (tryFindInstanceFirst)
  {
    child = getInstanceForLiteralIndex(list, index);
    if (child != NULL)
    {
      return child;
    }
  }
```

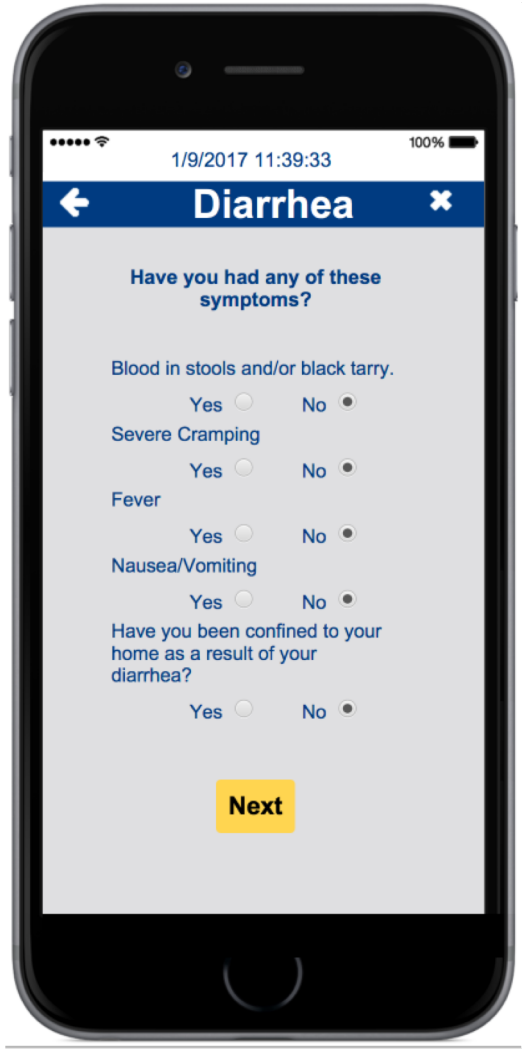
Abstraction and Notation helps with Reviews

```
decision table BpScoreDecisionTable(sys: bpRange, dia: bpRange) =
```

		dia					
		<= 50	[51..90]	[91..95]	[96..100]	[101..109]	>= 110
sys	<= 90	1	1	3	4	5	6
	[91..140]	2	2	3	4	5	6
	[141..150]	3	3	3	4	5	6
	[151..160]	4	4	4	4	5	6
	[161..179]	5	5	5	5	5	6
	>= 180	6	6	6	6	6	6

Simulators allow SMEs to “play“ with stuff





SM-level analyses are much easier to build

There are tax values declared as public, but they are never used.

You cannot add a temporal value and a scalar value.

Pre- and postconditions of function-like things are always met.

In your decision tree, the following alternative is not handled.

For all possible program executions, a dangerous state never occurs.

Not all security risks have been discharged through a mitigation.

The attack scenario X is classified HIGH RISK, but there‘s no mitigation.

The fault X is propagated from A to B but B does not handle it.

There’s a resource contention betw. resources X and Y in scenario Z.

Devs freed from SM details can focus on platforms

Automatic translations capture idioms and patterns

Devs freed from SM details can focus on platforms

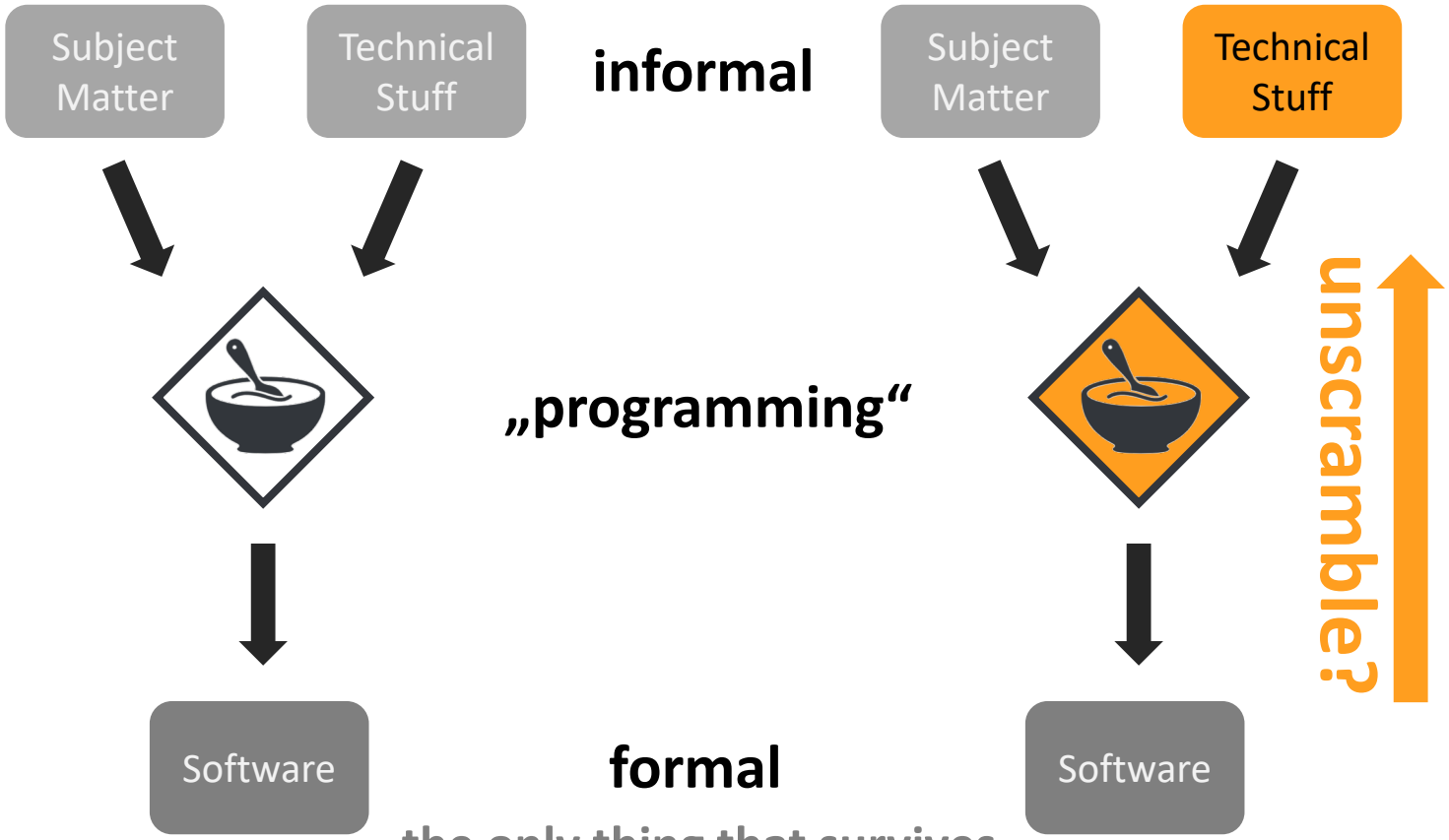
Automatic translations capture idioms and patterns

Devs freed from SM details can focus on platforms

Automatic translations capture idioms and patterns

Separation of SM and technology avoids legacy problem

so what do you do when you want to
run that subject with new technology?

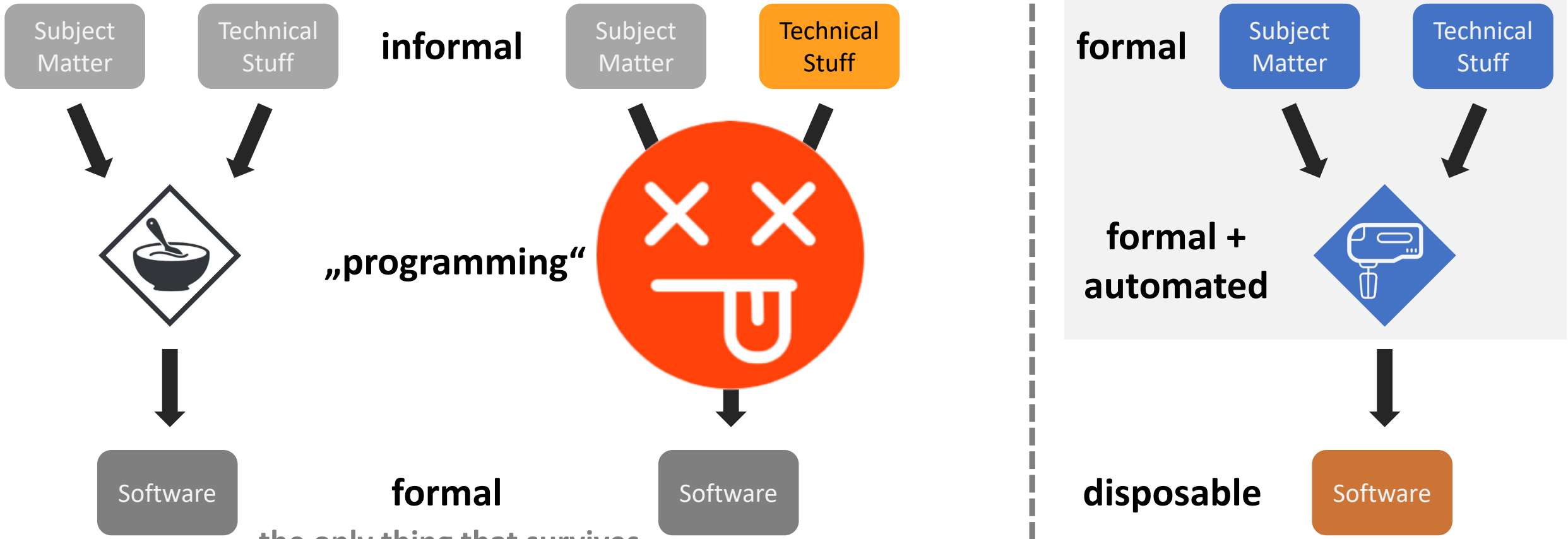


the only thing that survives
and is maintained and evolved

Separation of SM and technology avoids legacy problem

so what do you do when you want to run that subject with new technology?

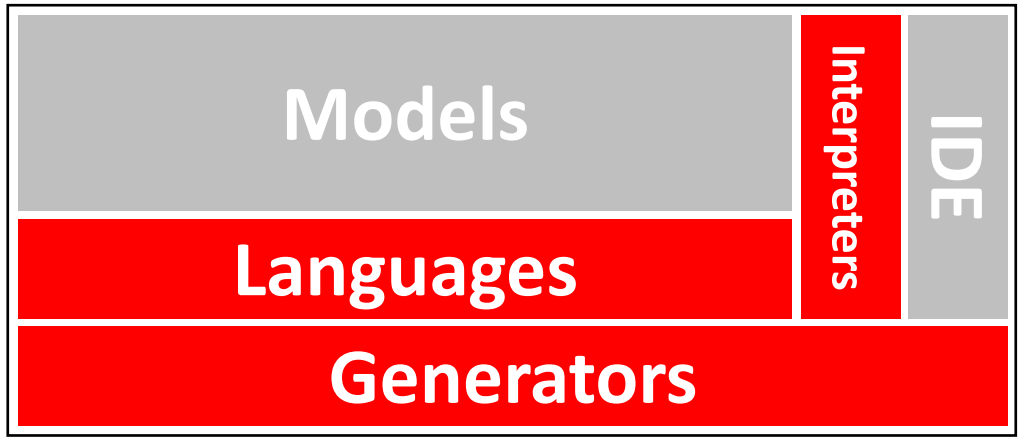
these now survive and are maintained and evolved



the only thing that survives and is maintained and evolved

Safety

How to build reliable generators



Semantic Redundancy for Assurance

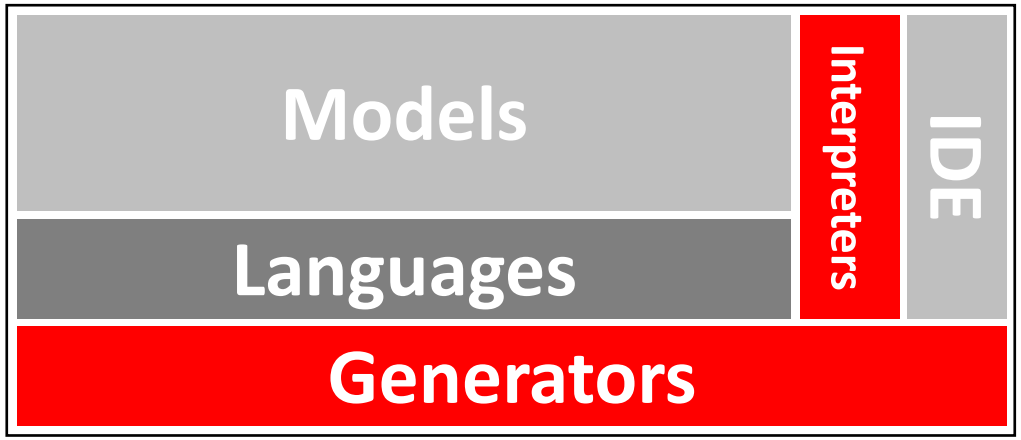
Technical Workflow

Efficient and high-quality implementations for data center and on-premise apps

Java Services

C DLLs

...



How do you test languages?

Expressivity Experimentation

(Grammar/Parser Testing)

Testing Static Semantics

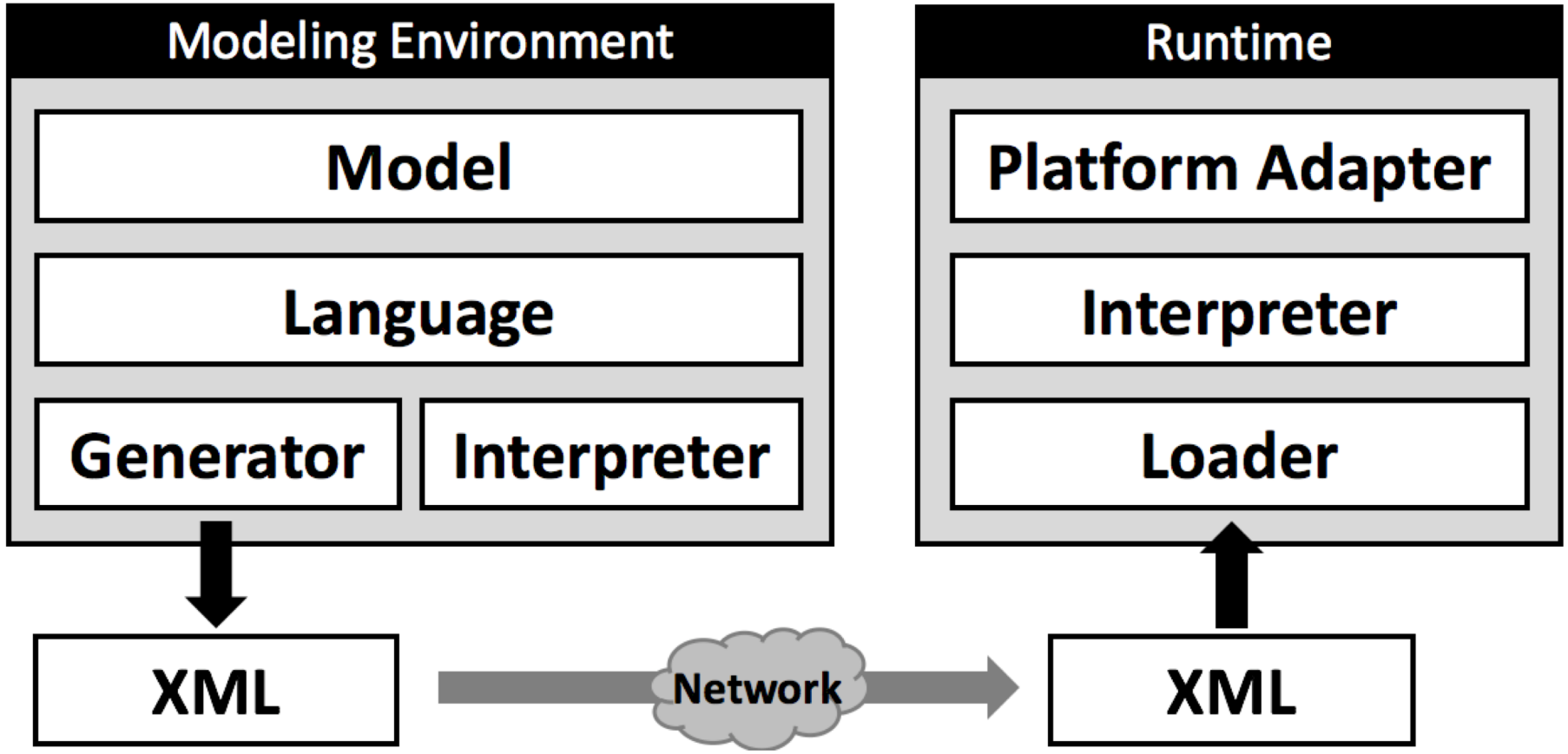
Testing Execution Semantics

http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

Paper in SoSym Journal

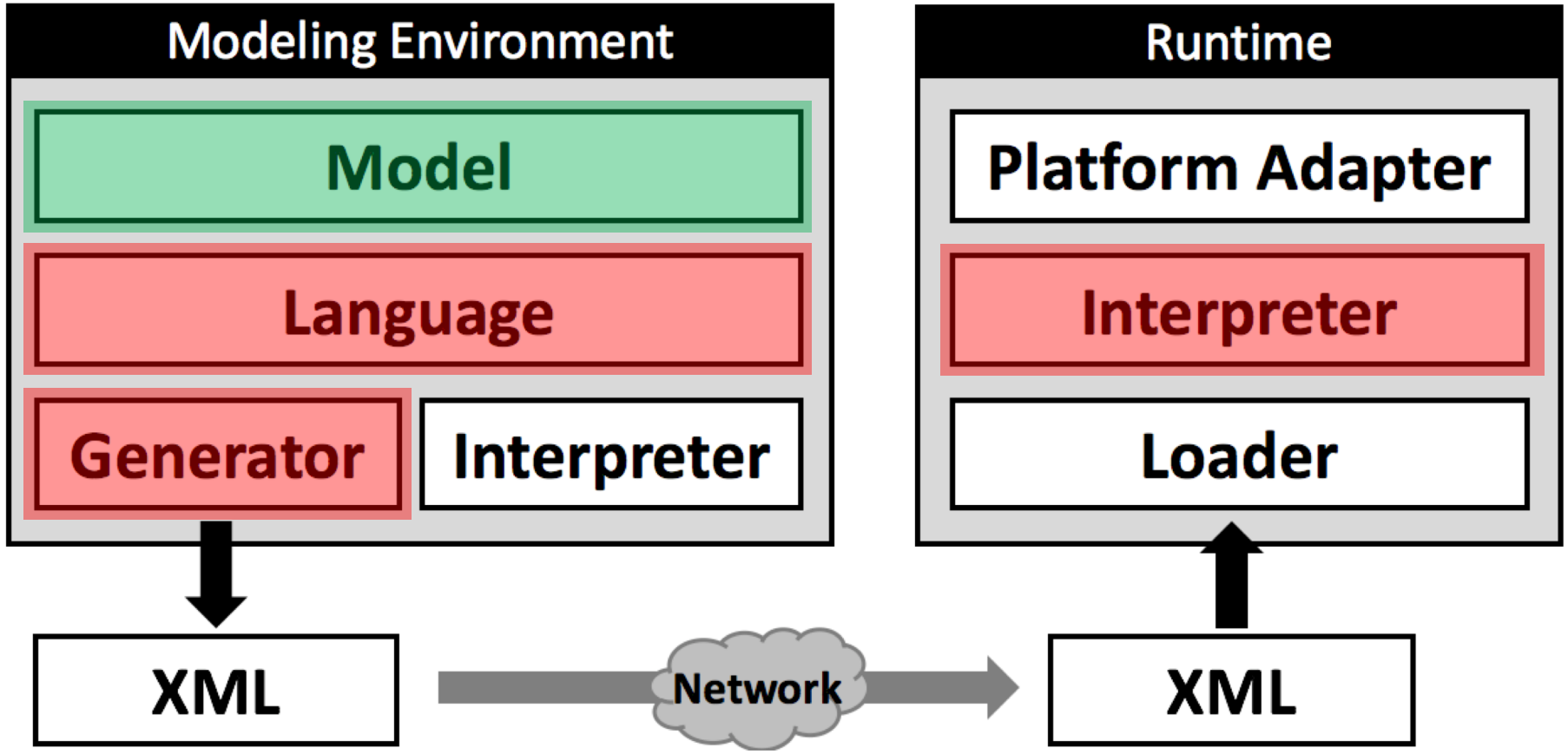
Healthcare Domain
Safety Critical

System Architecture



What good is all the abstraction if we cannot trust the translation to the implementation?

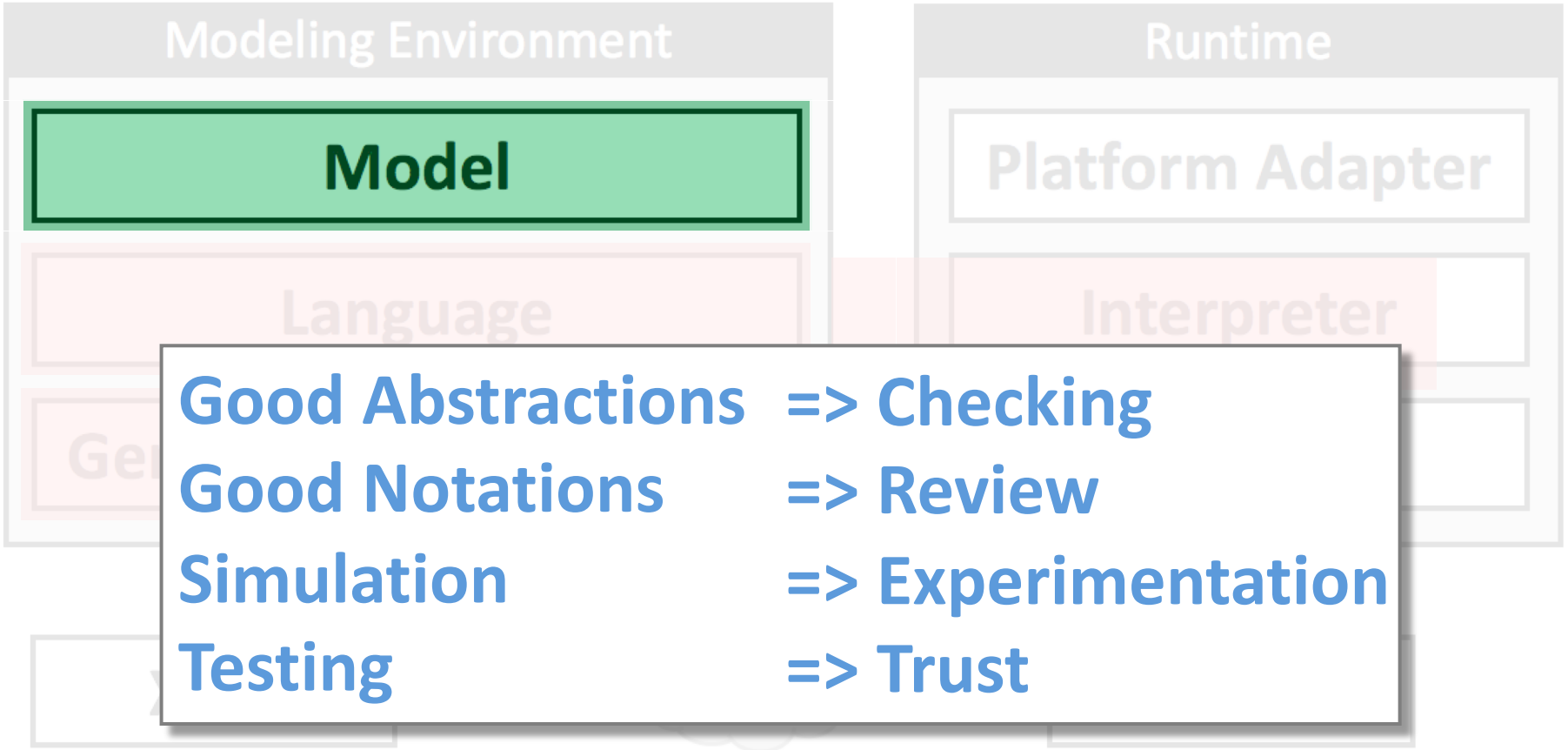
System Architecture & Safety Standards



Tools may introduce *additional systematic errors* if faulty.
Safety **standards** require reliable mitigation of such errors.

DO-178C EN50129 IEC62304 ISO26262

System Architecture & Safety Standards



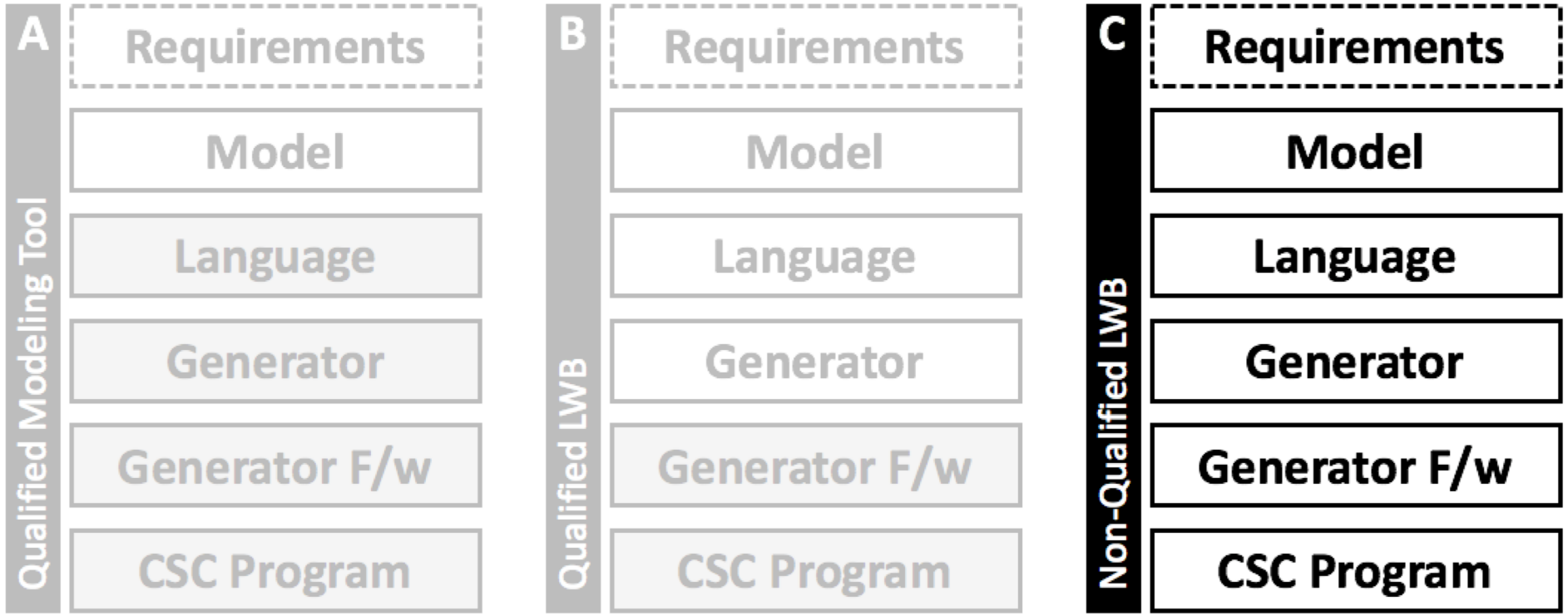
Good Abstractions => Checking
Good Notations => Review
Simulation => Experimentation
Testing => Trust

Tools may introduce *additional systematic errors* if faulty.
Safety standards require reliable mitigation of such errors.

DO-178C EN50129 IEC62304 ISO26262

Unqualified Tools!

What good is all the abstraction if we cannot trust the translation to the implementation?



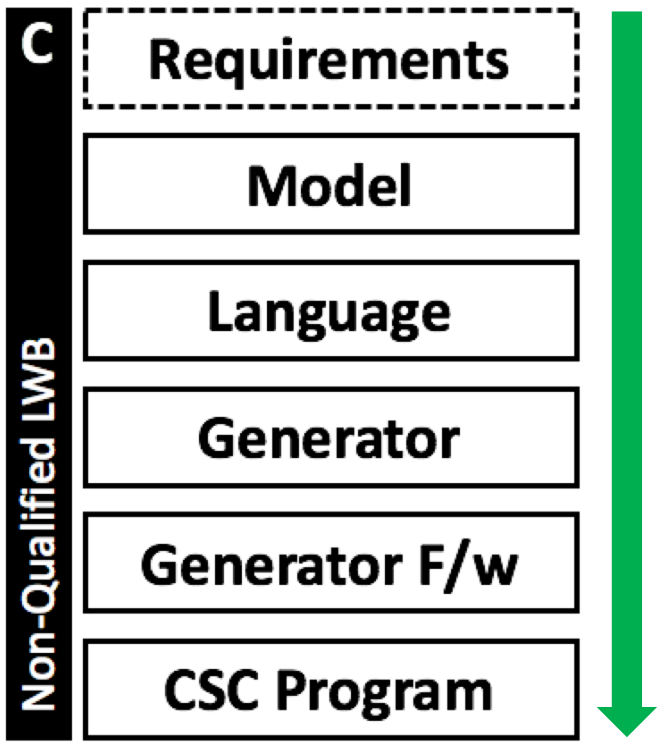
Unqualified Tools!

End-to-end testing required.
How to do this without exploding effort?

Automated Redundancy

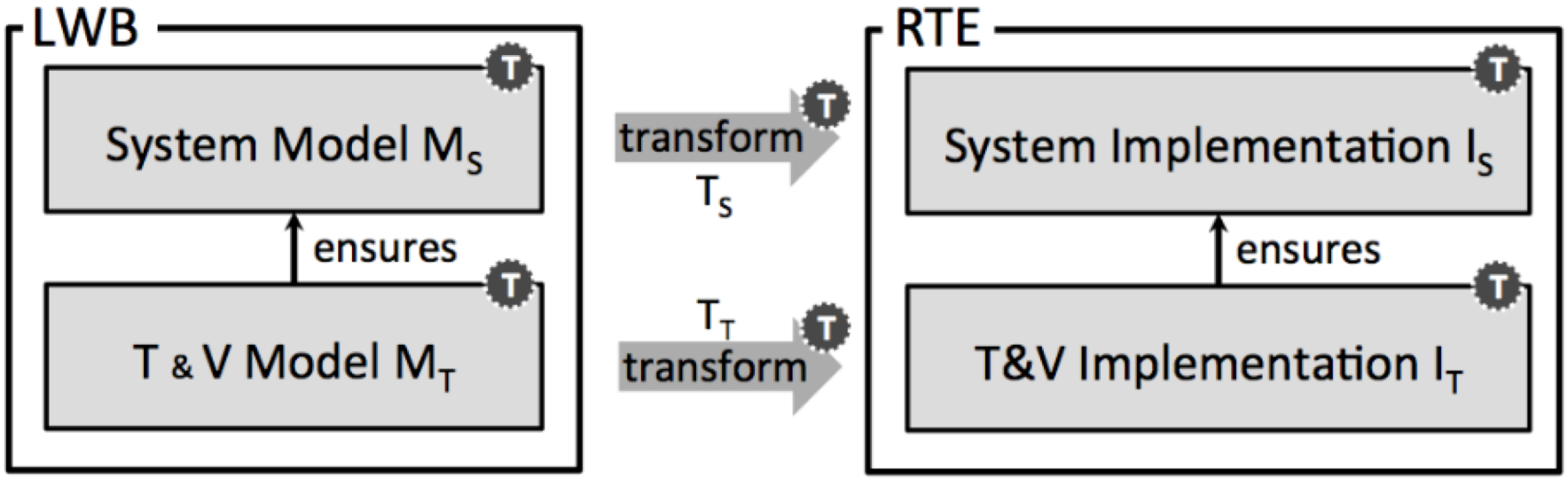
catch errors in redundant path
while reducing manual effort.

+ specific risk mitigations



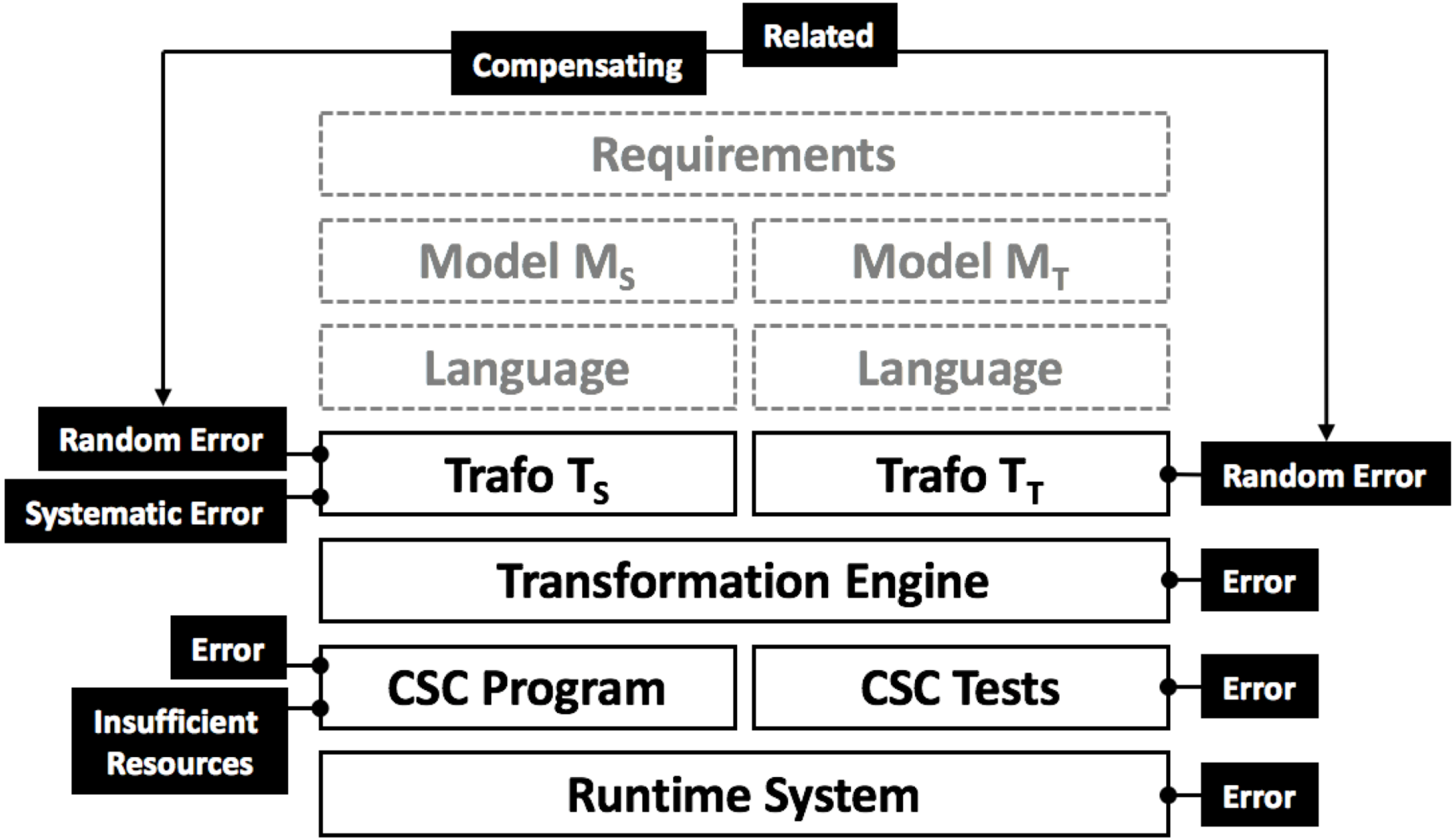
Modeling Architecture

Model the Algo/System with the DSL and also model the tests/verification. Then translate both and execute on the level of the implementation.

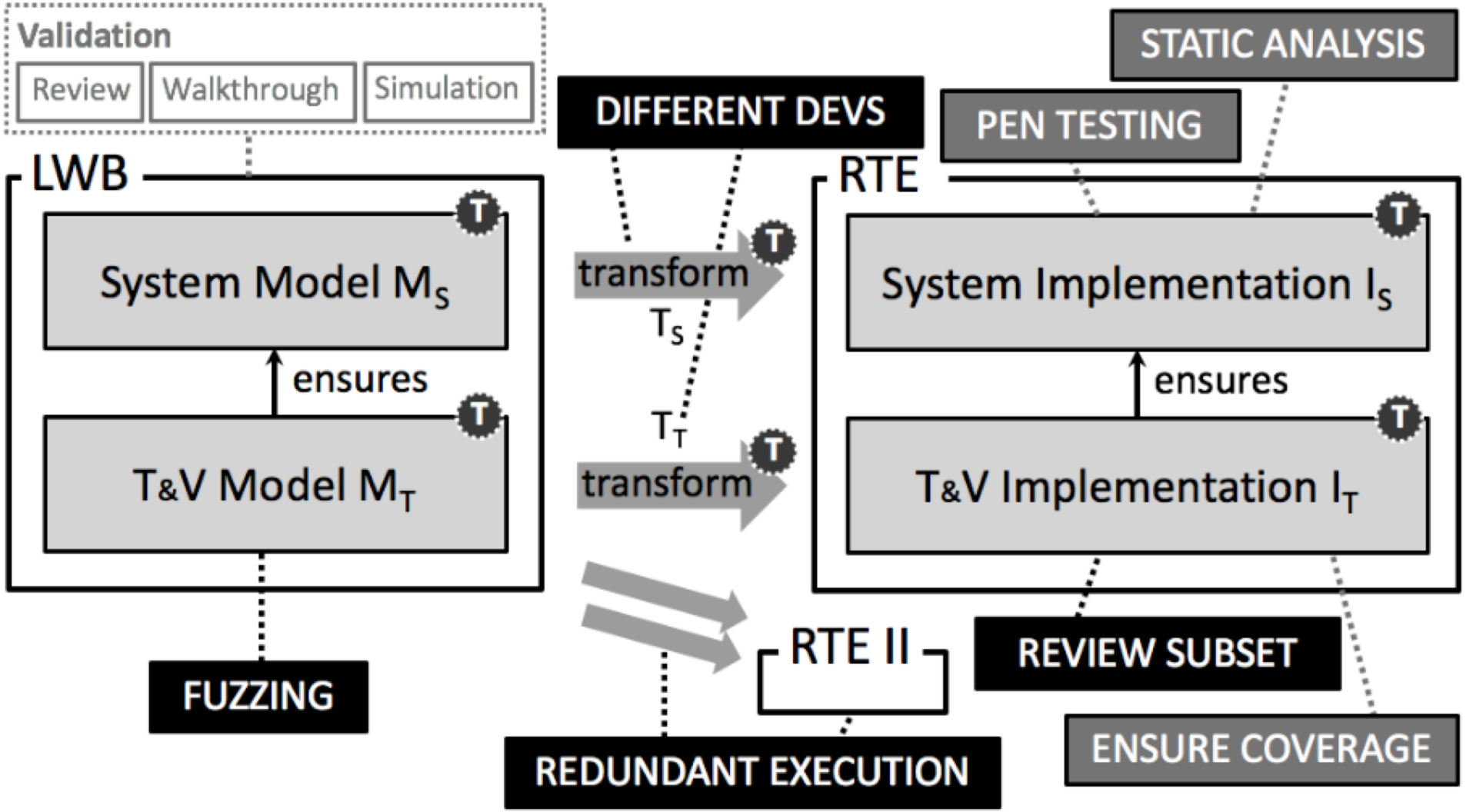


+ Risk Analysis + Mitigations

Risk Analysis



Mitigations – Safe Modeling Architecture



Mitigations – Safe Modeling Architecture

use redundant execution on two execution engines
use different developers for the two trafos
review a subset of the generated code
clearly define and QA the DSL
to use fuzzing on the tests
ensure high coverage for the tests
run the tests on the final device
perform static analysis on the generated code
perform penetration testing on the final system
and use architectural safety mechanisms.

Mitigations – Safe Modeling Architecture

use redundant execution on two execution engines
use different developers for the two trafos
review a subset of the generated code
clearly define and QA the DSL

---

only these specific to DSL use

to use fuzzing on the tests
ensure high coverage for the tests
run the tests on the final device
perform static analysis on the generated code
perform penetration testing on the final system
and use architectural safety mechanisms.

Mitigations – Safe Modeling Architecture

use **redundant** execution on two execution engines

- **C++ interpreter on device**
- **In-IDE Java Interpreter**

Lots of overhead? Not really.

Validation: the in-IDE interpreter is used for interactive testing, exploration, understanding, simulation. HCP‘s single-most appreciated use of the models!

Verification: addresses unrelated but compensating, as well as related errors in the transformations. Does not rely on trafo engine, so finds error in it. It‘s also simple (!fast), so acts as a specification.

Test Stats and other Numbers

Two reference Algos, 305 test cases for Bluejay, 297 for Greenjay, plus lower-level tests for decision tables and trees

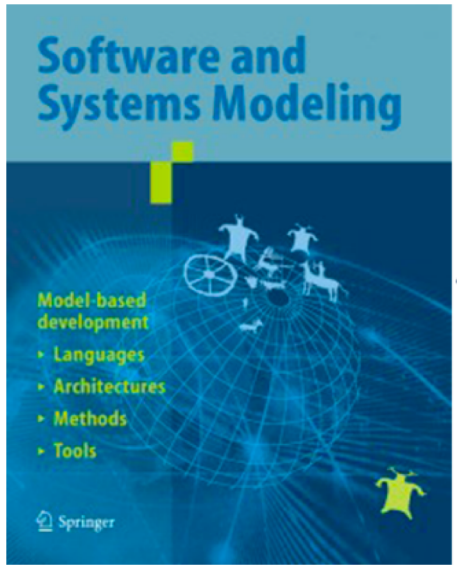
100% line coverage regarding language structure, Java interpreter and C++ interpreter

Validation Effort Reduction from **50 PD to 15 PD**

Test Setup Effort reduced by a **factor of 20**

Shortened Turnaround for req -> impl -> write tests -> execute tests b/c of much better tool integration

„Tremendous Speedup“ for changes to algo *after* it has been validated – automatic reexecution of everything.



Software & Systems Modeling
pp 1–24 | Cite as

Using language workbenches and domain-specific languages for safety-critical software development

Authors | Authors and affiliations

Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, Arne Nordmann

Regular Paper
First Online: 17 May 2018

Abstract

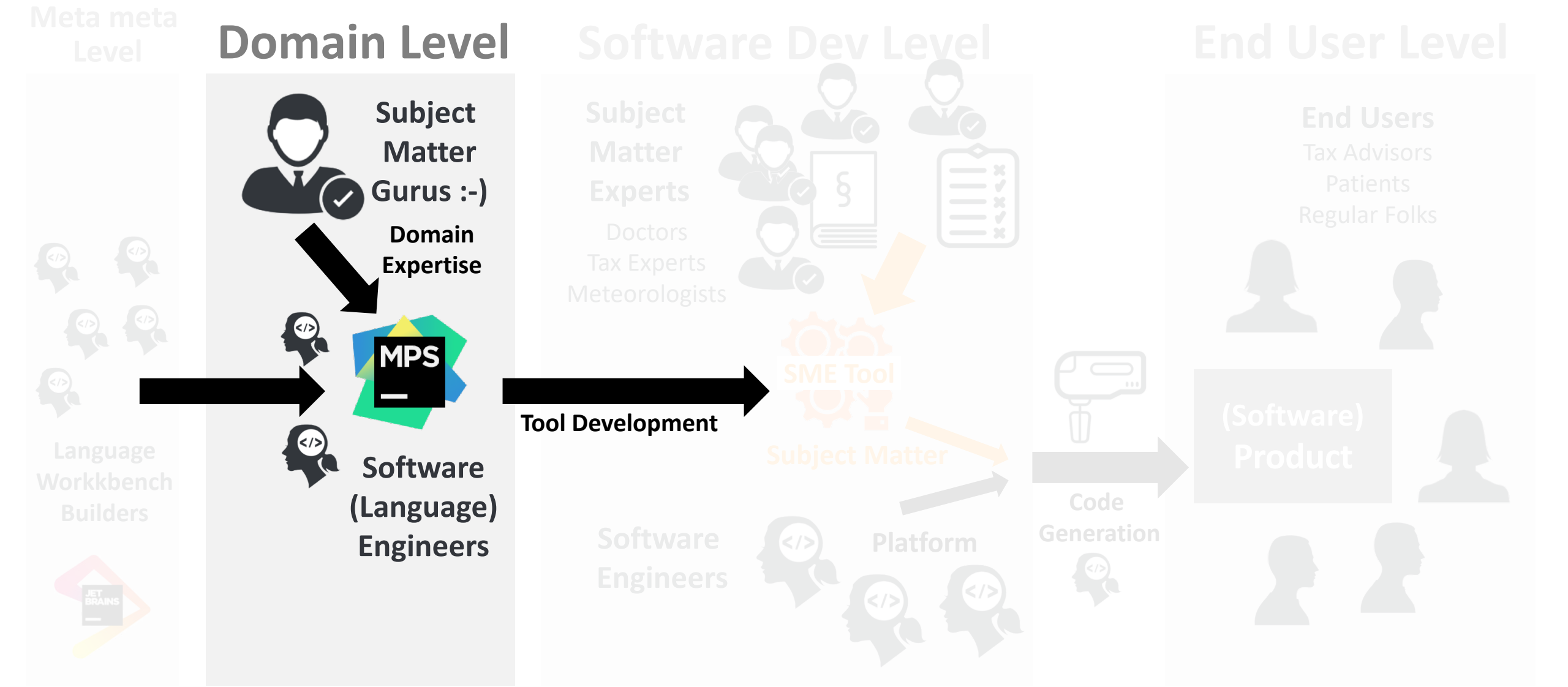
Language workbenches support the efficient creation, integration, and use of domain-specific languages. Typically, they execute models by code generation to programming language code. This can lead to increased productivity and higher quality. However, in safety-/mission-critical environments, generated code may not be considered trustworthy, because of the lack of trust in the generation mechanisms. This makes it harder to justify the use of language workbenches in such an environment. In this paper, we demonstrate an approach to use such tools in critical environments. We argue that models created with domain-specific languages are easier to validate and that the additional risk resulting from the transformation to code can be mitigated by a suitably designed transformation and verification architecture. We validate the approach with an industrial case study from the healthcare domain. We also discuss the degree to which the approach is appropriate for critical software in space, automotive, and robotics systems.

http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

Meta

How to build the languages, IDE and generators

Big Picture: How does knowledge get into software



Language Workbenches

Xtext

{S} spoofax

Rascal

Tools for building languages and their IDEs

Language Workbench

Xtext

{S} spoofax

Rascal

Open Source Language Workbench from JET BRAINS

Projectional Editor that supports a wide variety of notations

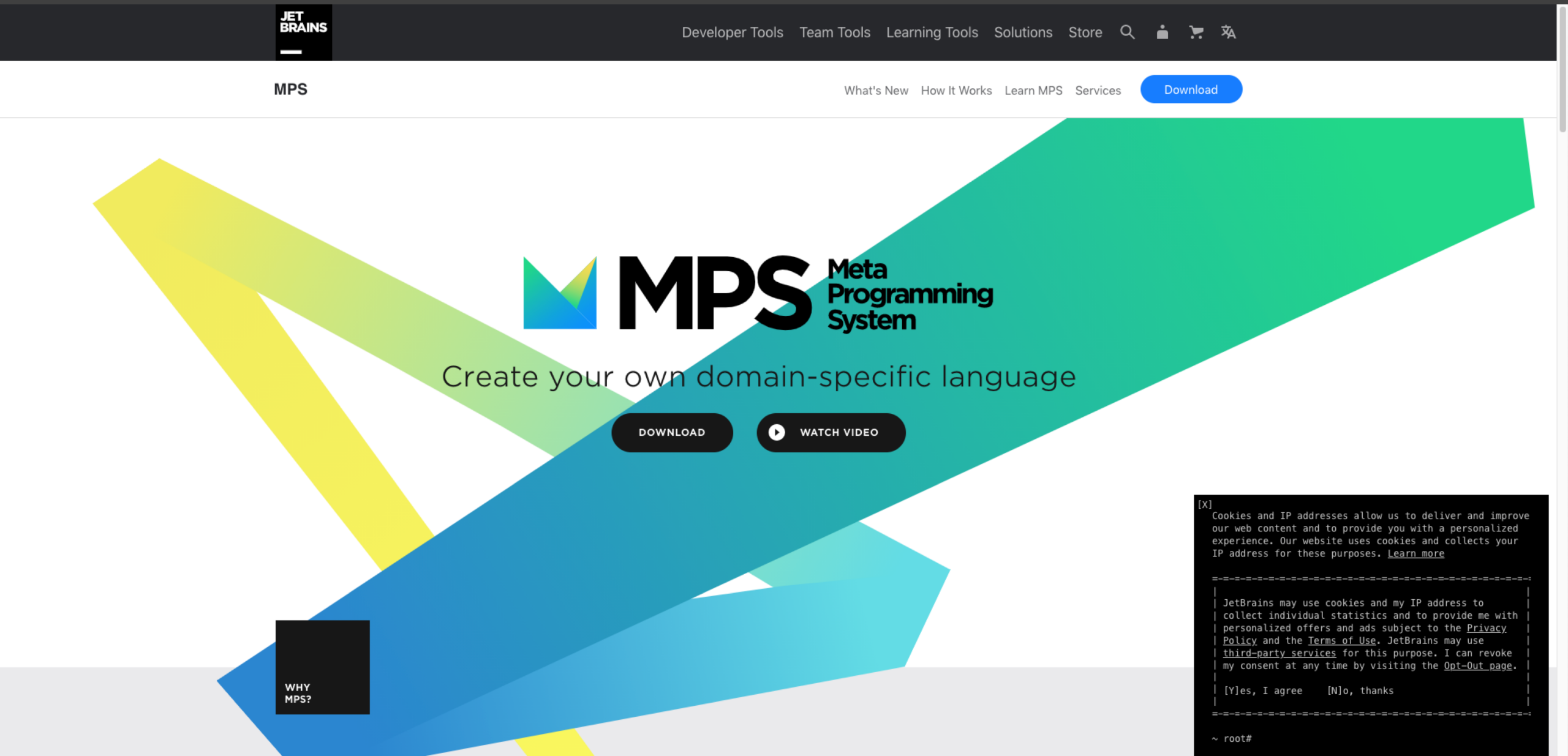
Robust support for language modularity and composition

Support for all relevant language aspects:

Structure • Editor • Type System • Constraints • Intentions
Refactorings • Interpretation • Code Generation
Code Completion • Find References • Goto Definition
Version Control • Diff/Merge …

Really not your Daddy‘s Parser Generator!

Language Workbench



Growing a DSL on top of KernelF

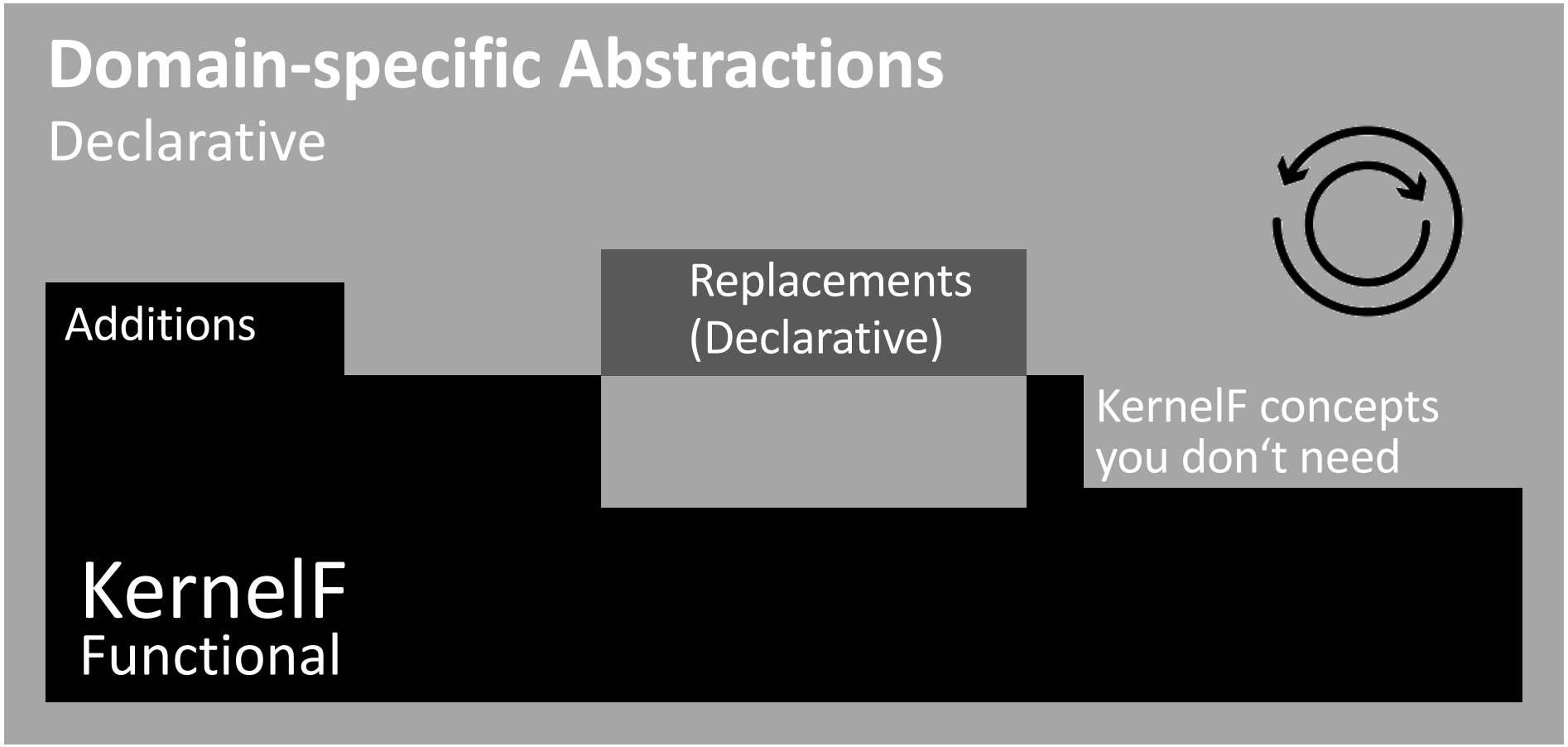
- Robust existing language and interpreter
- Initial "Demoware" very quick
- Good foundation for wow-features (Tables, Visualization)
- „Trap door“ for complex exceptions
- Step-wise DSL-ification

Primitive Types and Literals • Basic Operators • Conditionals • Decision Tables and Trees • Lists • Records • Dates • Temporale Types • Functions • Constants • Test Cases • Interpreter • Coverage Analyzer • etc.

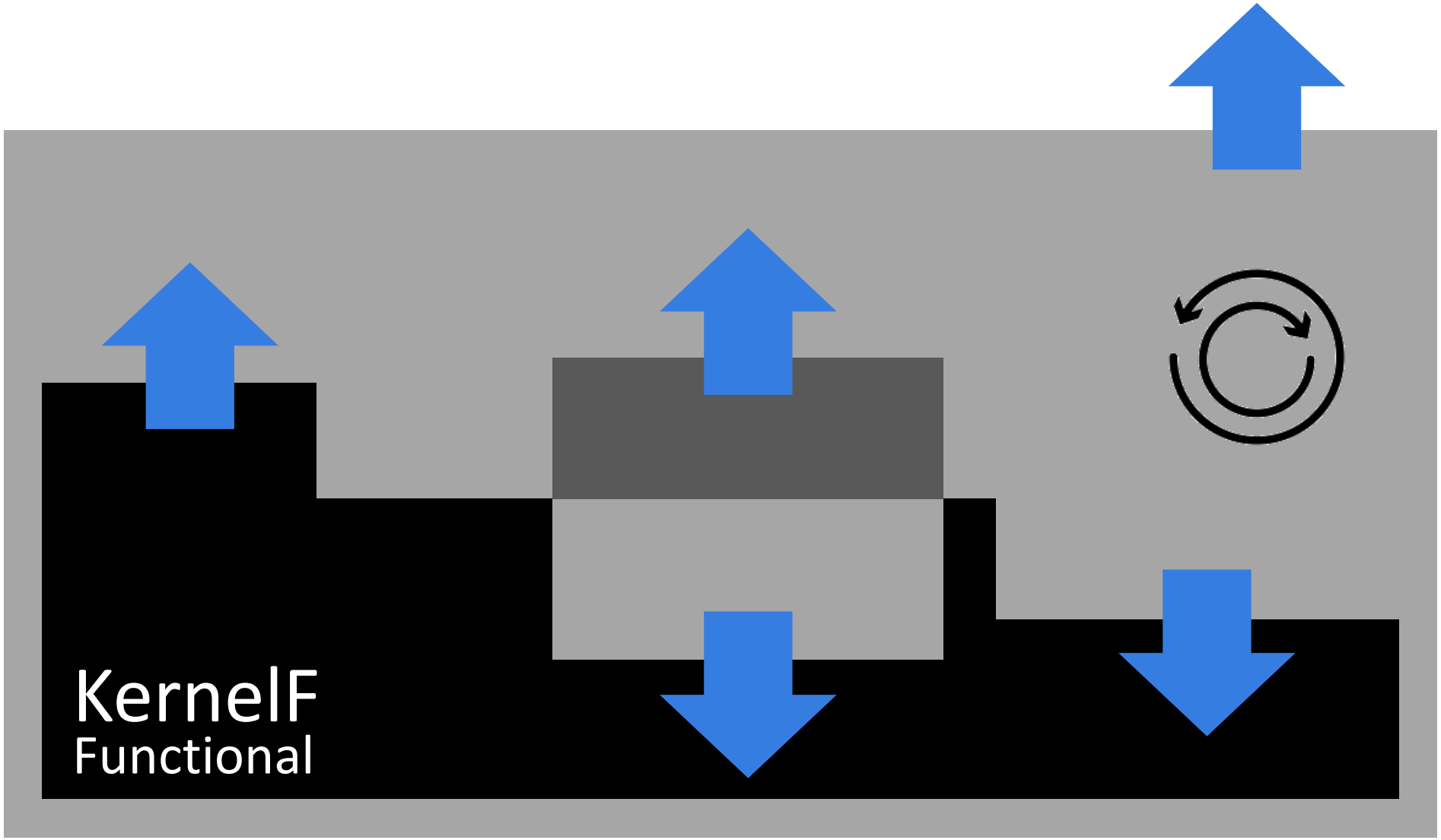
KernelF
Functional

https://github.com/IETS3/iets3.opensource
https://build.mbeddr.com/overview.html

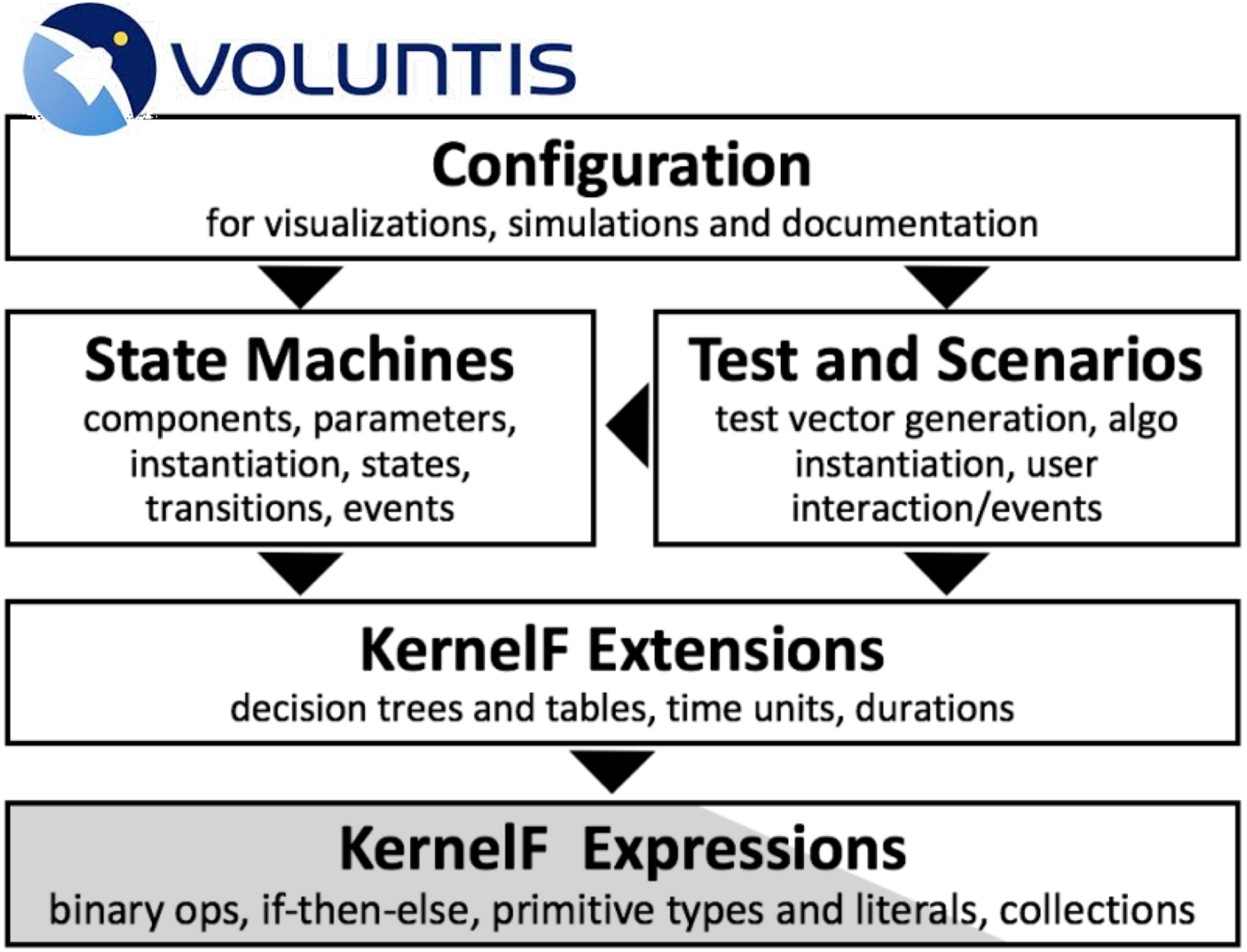
Growing a DSL on top of KernelF



Growing a DSL on top of KernelF



Language Architecture and Sizes



Language Part	# of concepts	percentage of total
Expressions (KernelF)	83	31%
Expressions (Extended)	63	23%
State Machines	29	11%
Testing, Scenarios	41	15%
Configuration	54	20%
Total	270	100%

DATEV

Kind	Language	# of concepts
DATEV-proprietary	`payroll.dsl.core`	166
	`payroll.dsl.test`	32
	`payroll.dsl.institution`	26
	`payroll.dsl.migration`	7
Developed for DATEV and then open sourced	`kernelf.datetime`	27
	`kernelf.temporal`	24
Use as-is from KernelF	KernelF	120
	Total	**402**

Wrap Up

Things to read and watch

MPS
http://jetbrains.com/mps

KernelF
https://github.com/IETS3/iets3.opensource

Artikel: Why DSLs? A collection of anecdotes
https://www.infoq.com/articles/why-dsl-collection-anecdotes

Paper: Fusing Modeling and Programming into Language-Oriented Programming
http://voelter.de/data/pub/markusvoelter-ISOLA2018-final.pdf

Paper: The Design, Evolution and Use of KernelF
http://voelter.de/data/pub/kernelf-icmt.pdf

Video/Presentation: Build your own Language: Why & How?
https://www.youtube.com/watch?v=9BvpBLzzprA

Video/Presentation: Language-oriented Business Applications
https://voelter.de/data/presentations/voelter-splash-i-LOBA.pdf

Things to remember

Use DSLs to allow SMEs to contribute directly.
Translate DSL models to code on top of platforms.

Direct SME input and easier validation will improve SM quality.
Platforms + Transformations will reduce/avoid low-level errors.

Software engineers build languages, IDEs, platforms and trafos.
Maintain these artifacts instead of the final software product.

Use language workbenches like MPS or Xtext for meta tooling.

Enjoy work (more) :-)

Dr. Markus Völter
voelter.de
voelter@acm.org
@markusvoelter