Introduction to

Domain-Specific Languages

Dr. Markus Völter

voelter.de
voelter@acm.org
@markusvoelter

Introduction to
Me :-)

DSL Design and Implementation
Domain Analysis
Software Architecture
Mitdenker

Consulting, Development,
Speaking, Writing

Dipl. Ing. Technical Physics
PhD Computer Science

- Data Transformation, Diffing, Migration
- Laser machining programming, hardware description, capability mapping
- Wage and tax calculation
- Realtime data processing in medical systems
- Code pattern detection and transformation
- Clinical trial protocols, plus testing
- Document description and diffing
- Data processing in Big Science
- Tax calculation
- Data modeling and validation
- User interface modeling
- An embeddable functional programming language (KernelF)
- Treatments in digital therapeutics
- Social insurance
- Insurance product modeling
- Functional architecture in automotive systems
- Variability modeling

Dr. Markus Völter

voelter.de/

voelter.de
voelter@acm.org
@markusvoelter

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

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

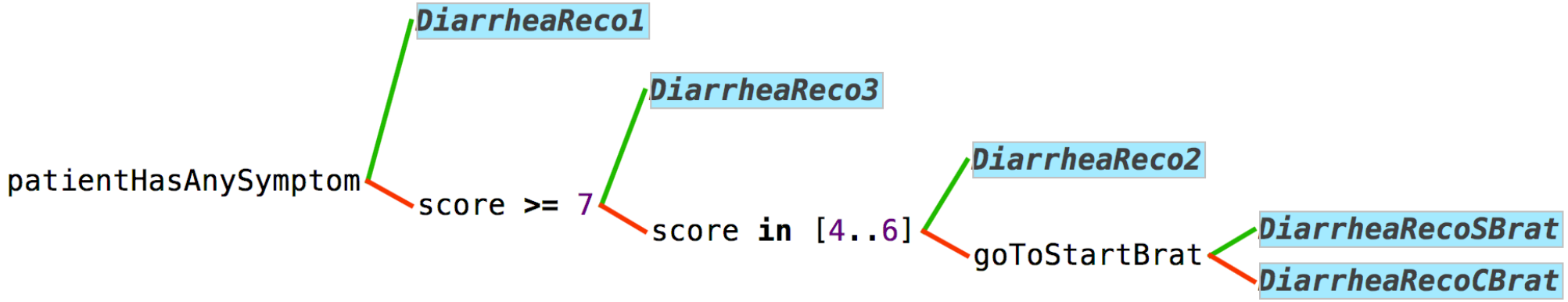
VOLUNTIS

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.

```
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
```

Social Insurance

Mix between form style and „real“ language.

Yellow parts are scaffolding and cannot be removed.

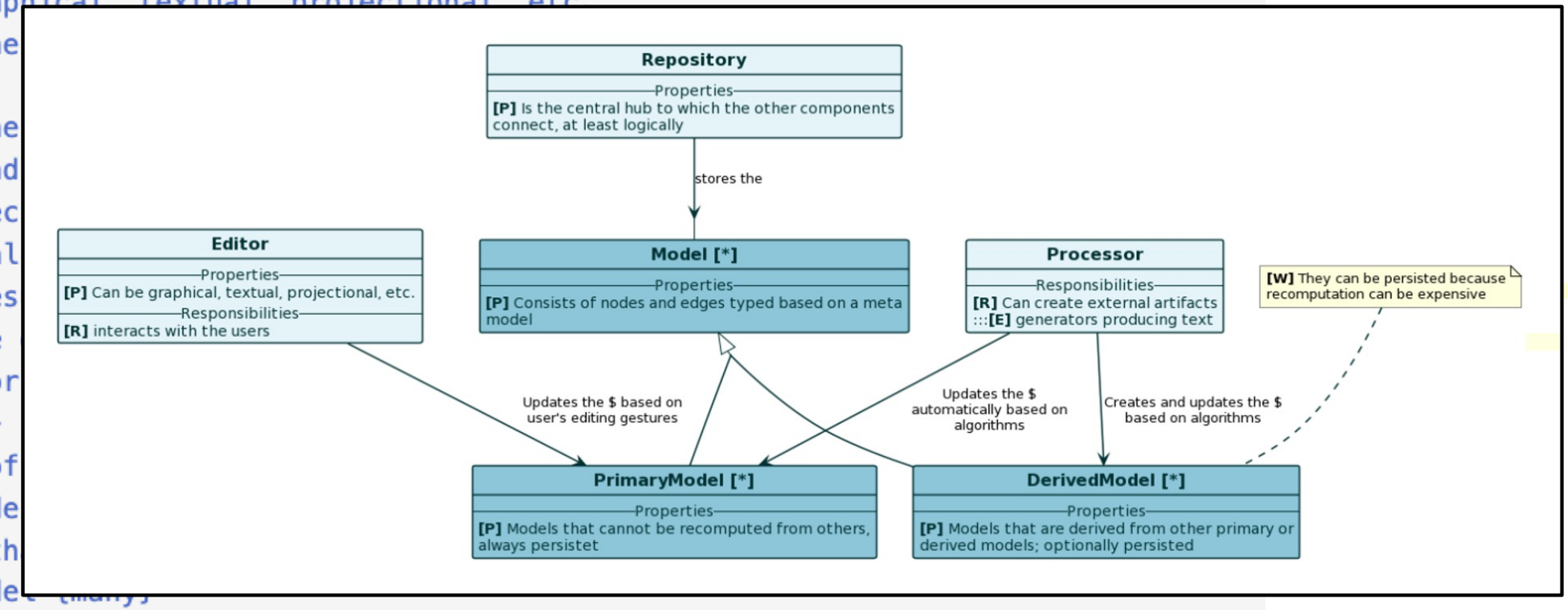
```
Name: UVG-Leistungen
Unterhaltsvorschuss
Zeitangabe:          laufend
Häufigkeit:          monatlich einmal
Leistungskontext:
Leistungsart:        Leer
Zählart: uvg
Anspruch Beginn: Anfang – Unbegrenzt: junger Mensch.geburtsdatum
Anspruch Ende: 01.01.1800     – 31.12.9999     : min(junger Mensch.geburtsda
                                                        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
```

CRC++

* T: Repository
  * C: stores the [Model]
  * P: Is the central hub to which the other components connect, at least logically
* T: Editor
  * R: interacts with the users
  * P: Can be graphical, textual, projectional, etc.
  * C: Updates the [PrimaryModel] based on user's editing gestures
* T: Processor
  * C: Updates the [PrimaryModel] automatically based on algorithms
  * C: Creates and updates the [DerivedModel] based on algorithms
    * E: Type checkers
    * E: Scope calculators
    * E: Model desugarers
  * R: Can create external artifacts
    * E: generators producing text
* D: Model {many}
  * P: Consists of nodes and edges typed based on a meta model
  * V: PrimaryModel {many}
    * P: Models that cannot be recomputed from others, always persistet
  * V: DerivedModel {many}
    * P: Models that are derived from other primary or derived models; optionally persisted
    * W: They can be persisted because recomputation can be expensive

CRC++

* T: Repository
  * C: stores the [Model]
  * P: Is the central hub to which the other components connect, at least logically
* T: Editor
  * R: interacts with the users
  * P: Can be graphical, textual, projectional, etc
  * C: Updates the
* T: Processor
  * C: Updates the
  * C: Creates and
    * E: Type chec
    * E: Scope cal
    * E: Model des
  * R: Can create
    * E: generator
* D: Model {many}
  * P: Consists of
  * V: PrimaryMode
    * P: Models th
  * V: DerivedMode
    * P: Models that are derived from other primary or derived models; optionally persisted
    * W: They can be persisted because recomputation can be expensive



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

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]

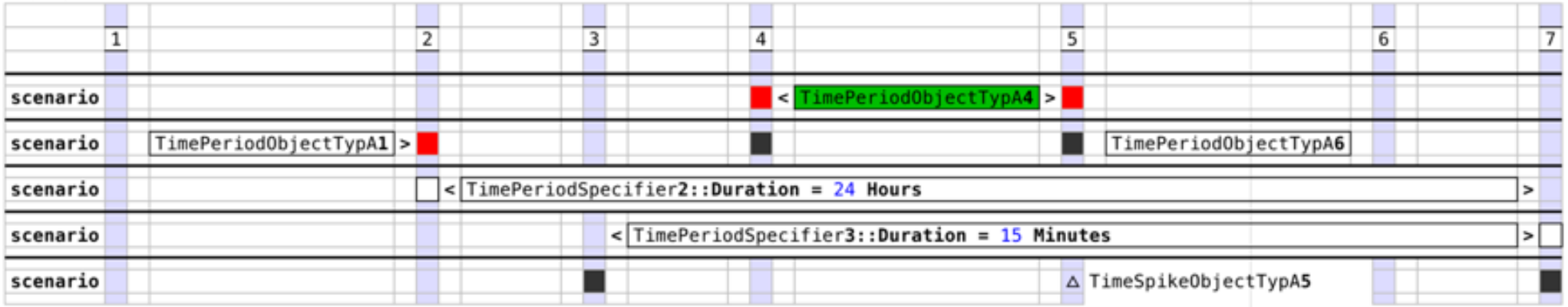
Law and Legal [Contract Modeling]

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

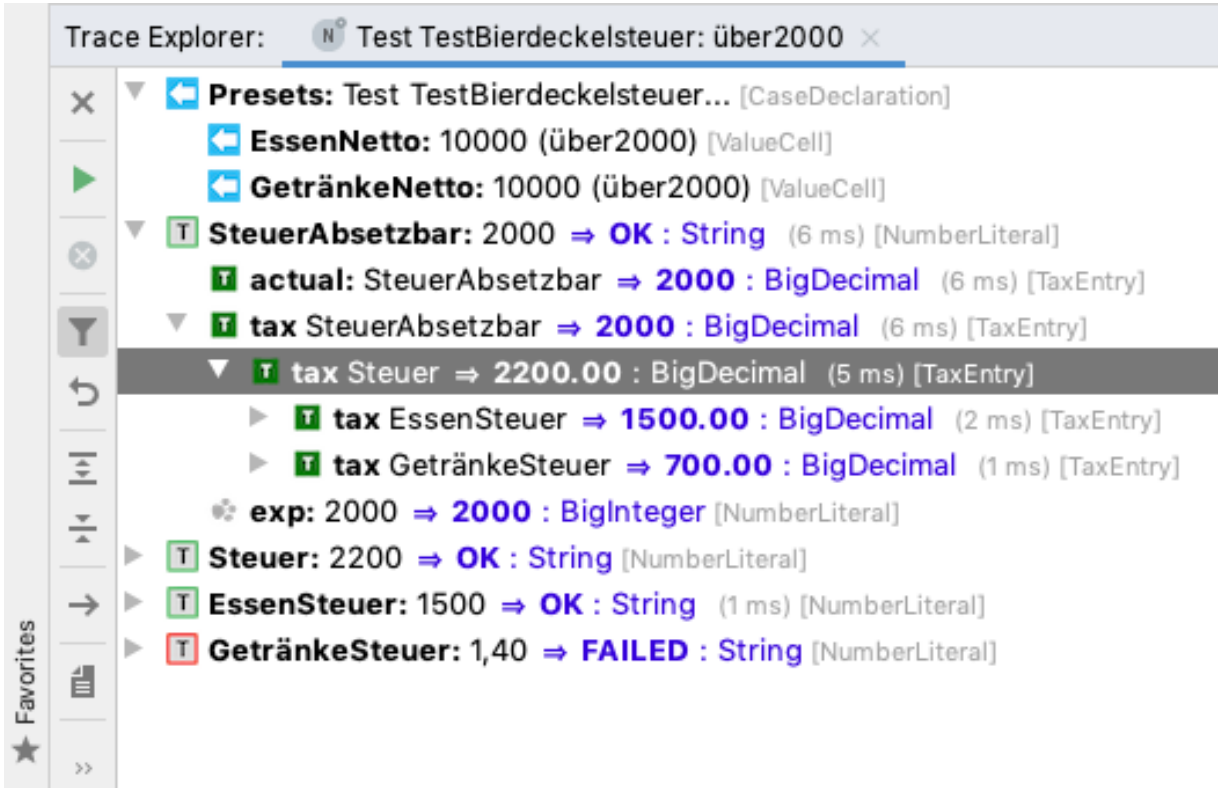
}

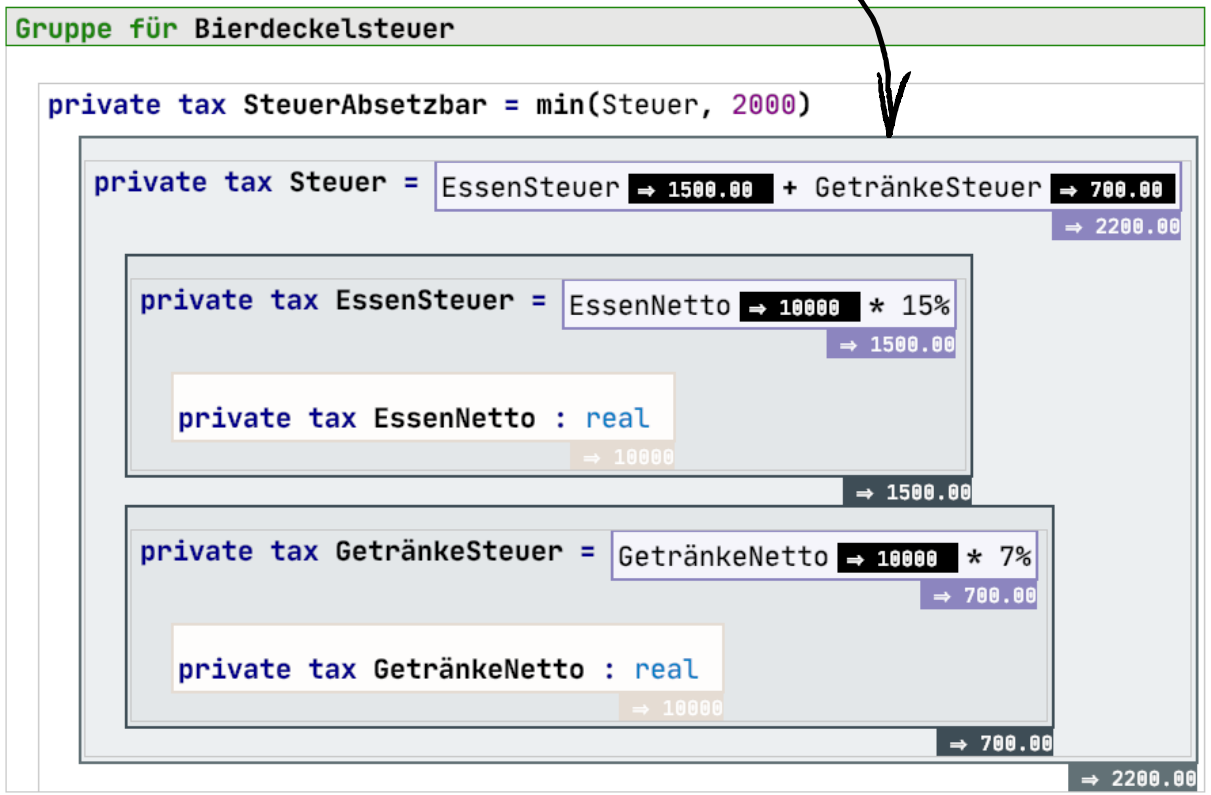
Intuitive capture of relevant data constellations and test expectations

Testing

Tracing in the IDE

Overlay of values over the calcualtion schema



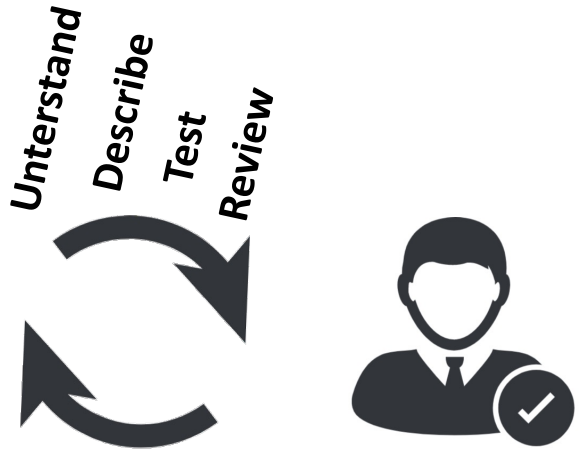


Workflows

Teams, Generation and DevOps

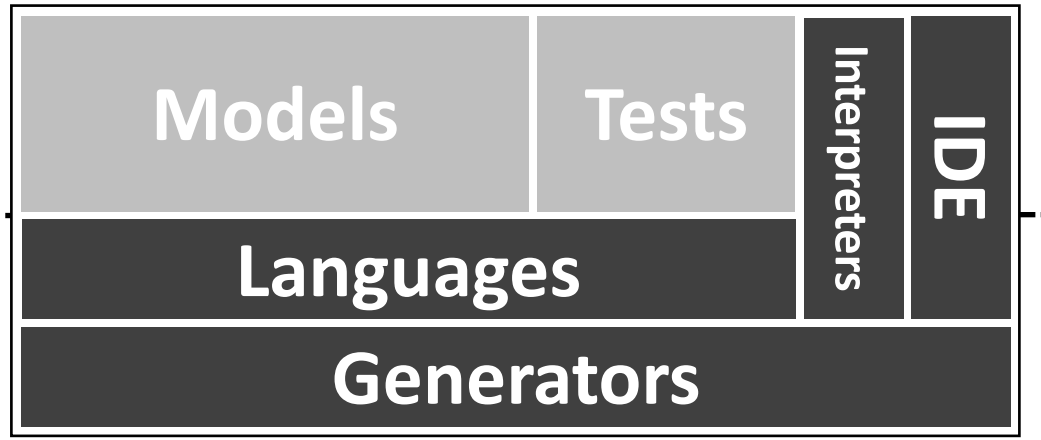
Subject Matter Workflow

Specification and test
of calculation rules



Collaboration

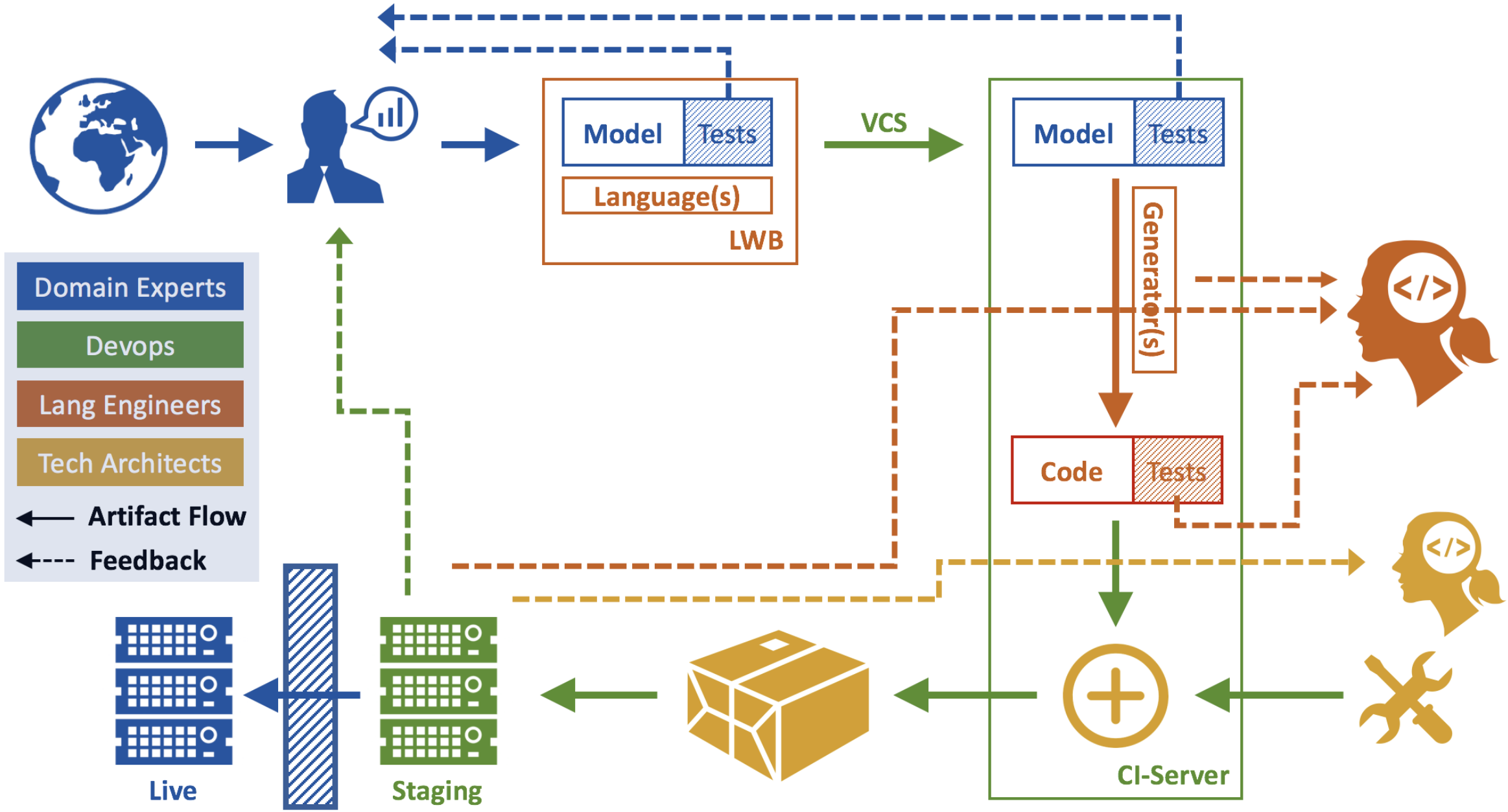
based on well-defined
and executable artifacts



Technical Workflow

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

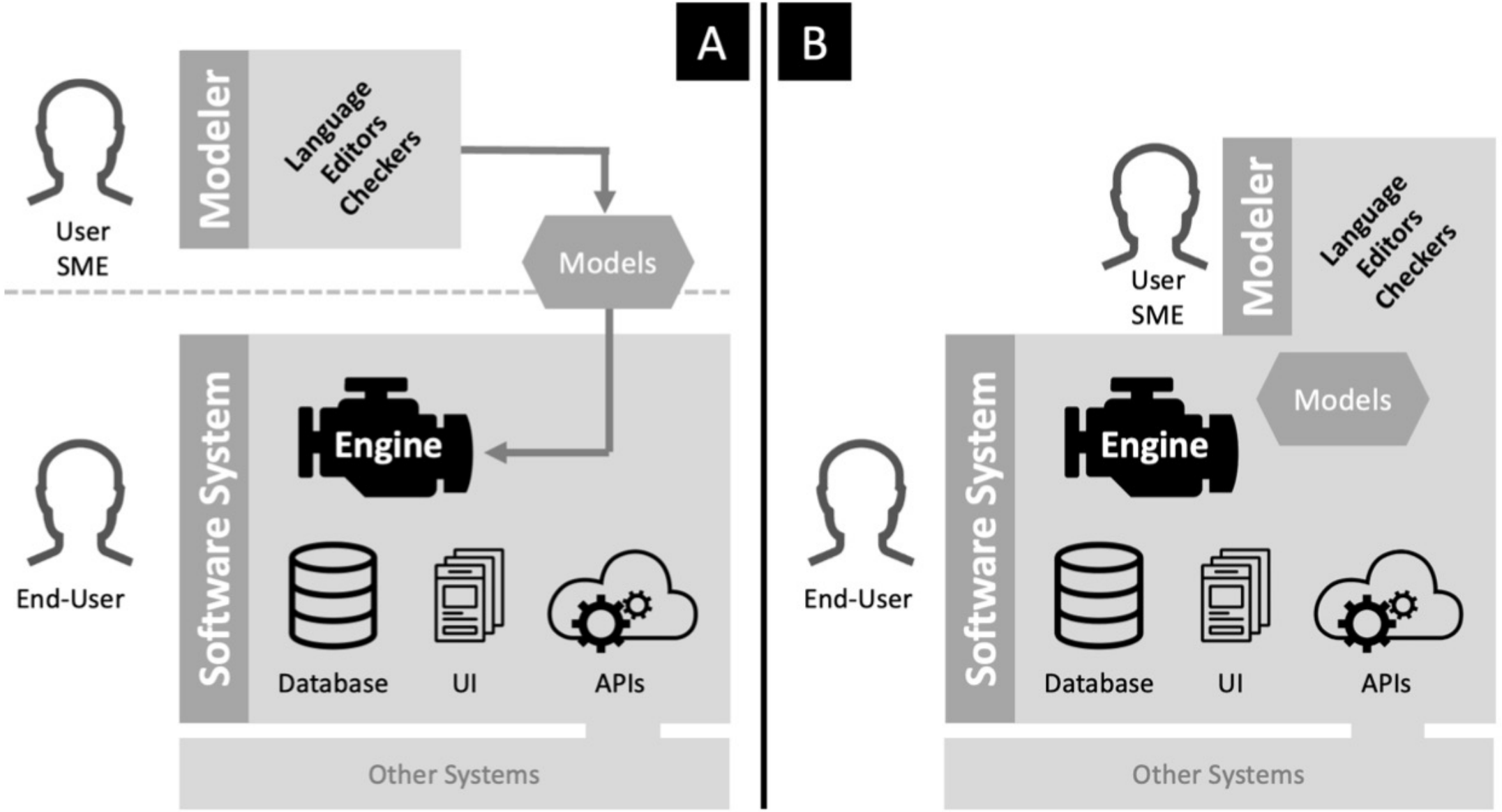
DevOps Perspective



Architecture

How to integrate DSL runtimes

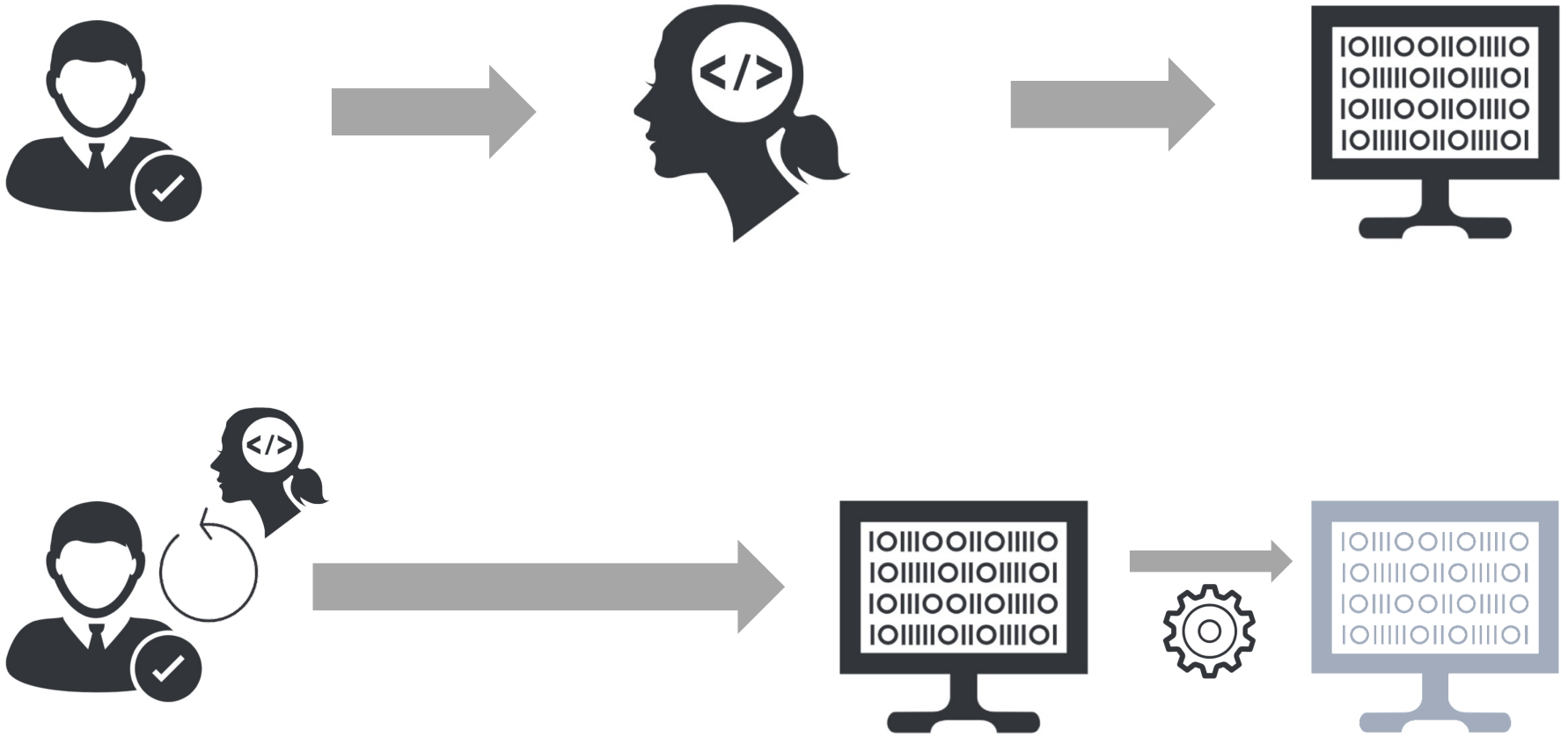
Runtime vs. Rest of the System



Quality

Why DSLs help with software quality

Direct “programming“ by SMEs avoids misunderstandings



Higher Level of Abstraction avoids low-level errors

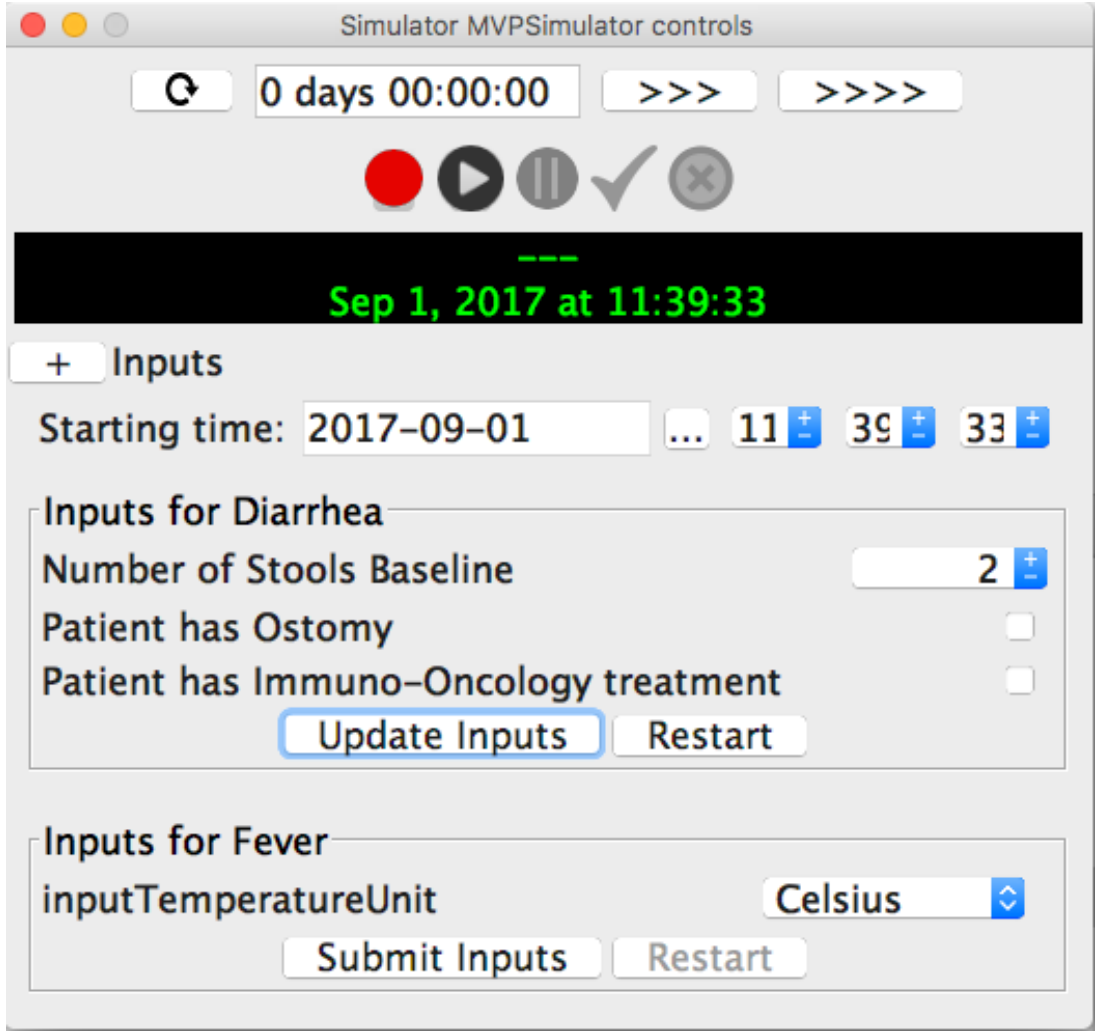
```
TaxGroup_t *G__1699215591_Einzeln_1_addGroupInstance_Aspekte_Erste_1_Einzeln_1
          (TaxList_t *list, uint32_t index, bool tryFindInstanceFirst)
{
  TaxGroup_t *group = list->group;
  VALUE indexValue = NULL;
  TaxGroup_t *child = NULL;
  DLListElement_t *element = NULL;
  if (tryFindInstanceFirst)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

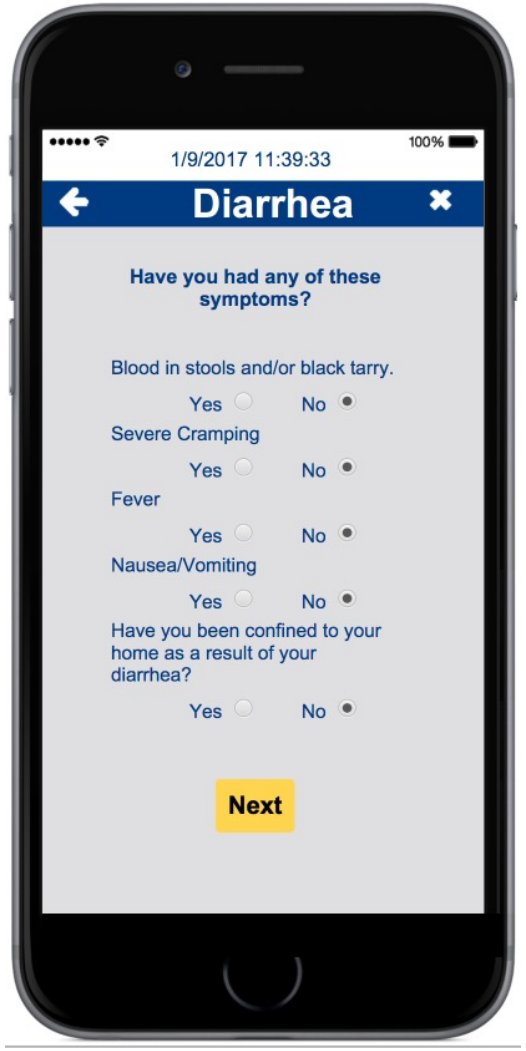
Abstraction and Notation helps with Reviews

```
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.

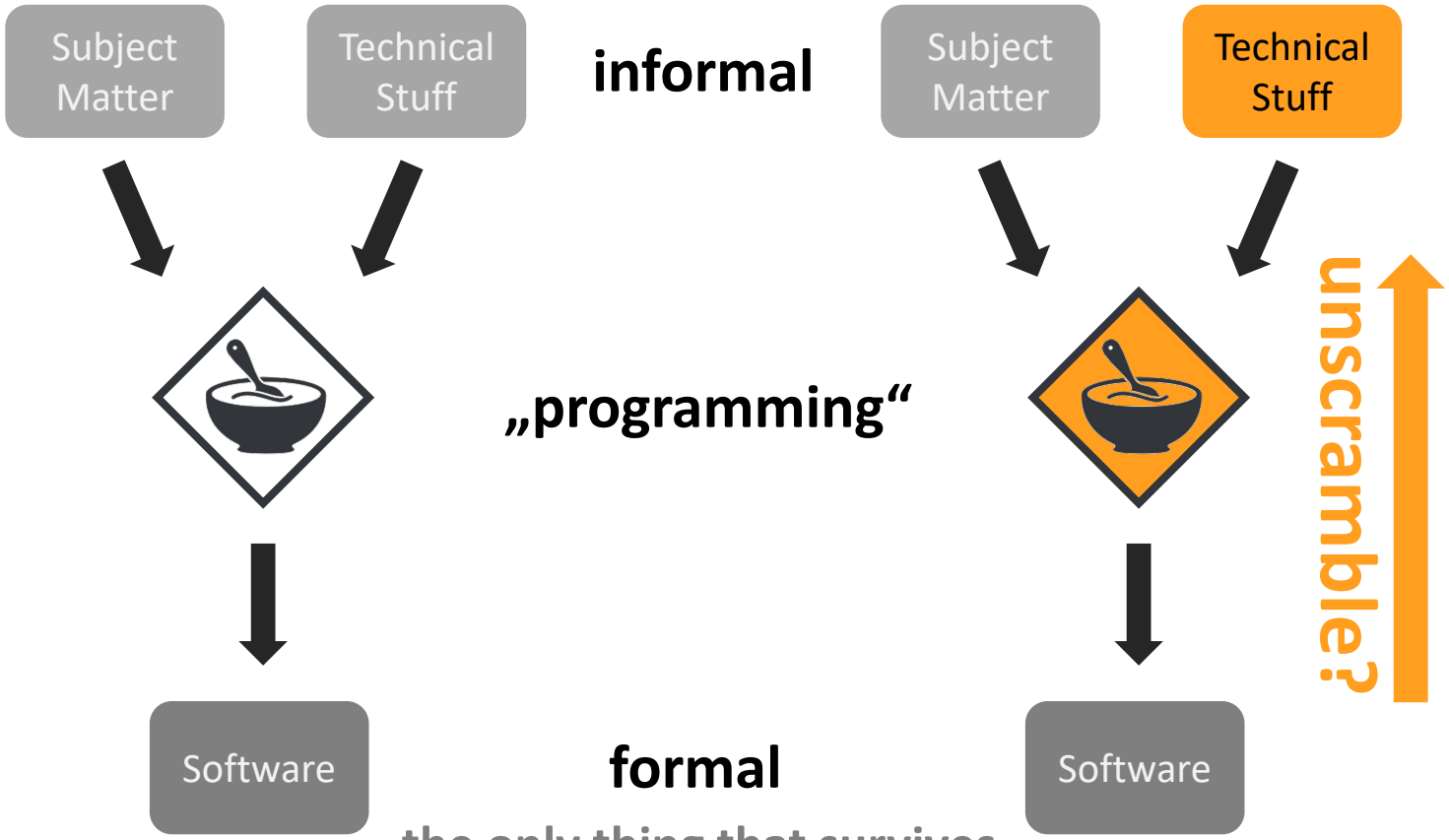
There’s a resource contention betw. resources X and Y in scenario Z.

Devs freed from SM details can focus on platforms

Automatic translations capture idioms and patterns

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

Subject Matter

Technical Stuff

informal

Software

formal

the only thing that survives and is maintained and evolved

„programming“

Subject Matter

Technical Stuff

Software

formal

Subject Matter

Technical Stuff

formal + automated

disposable

Software

Language?

When is something a language?

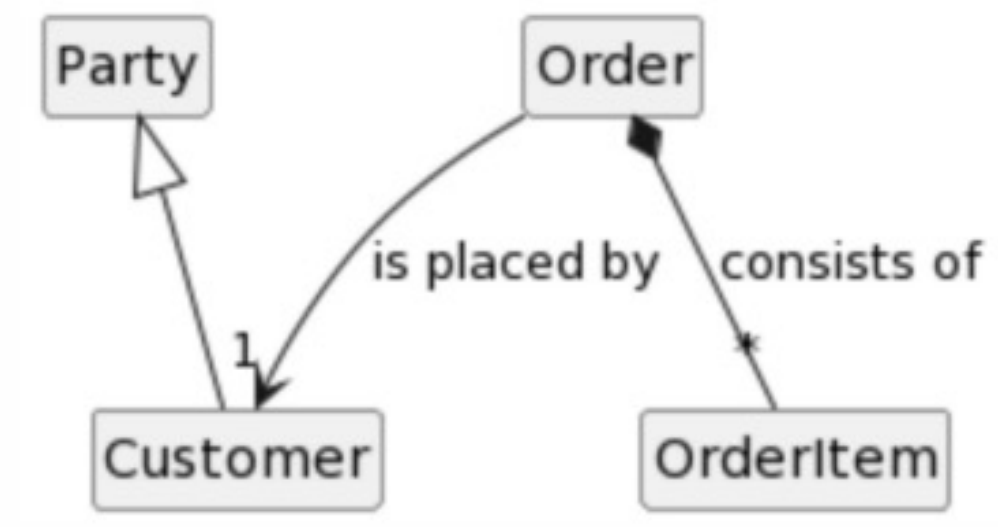
When is something a language?

Glossary

When is something a language?

Glossary

Structured Glossary

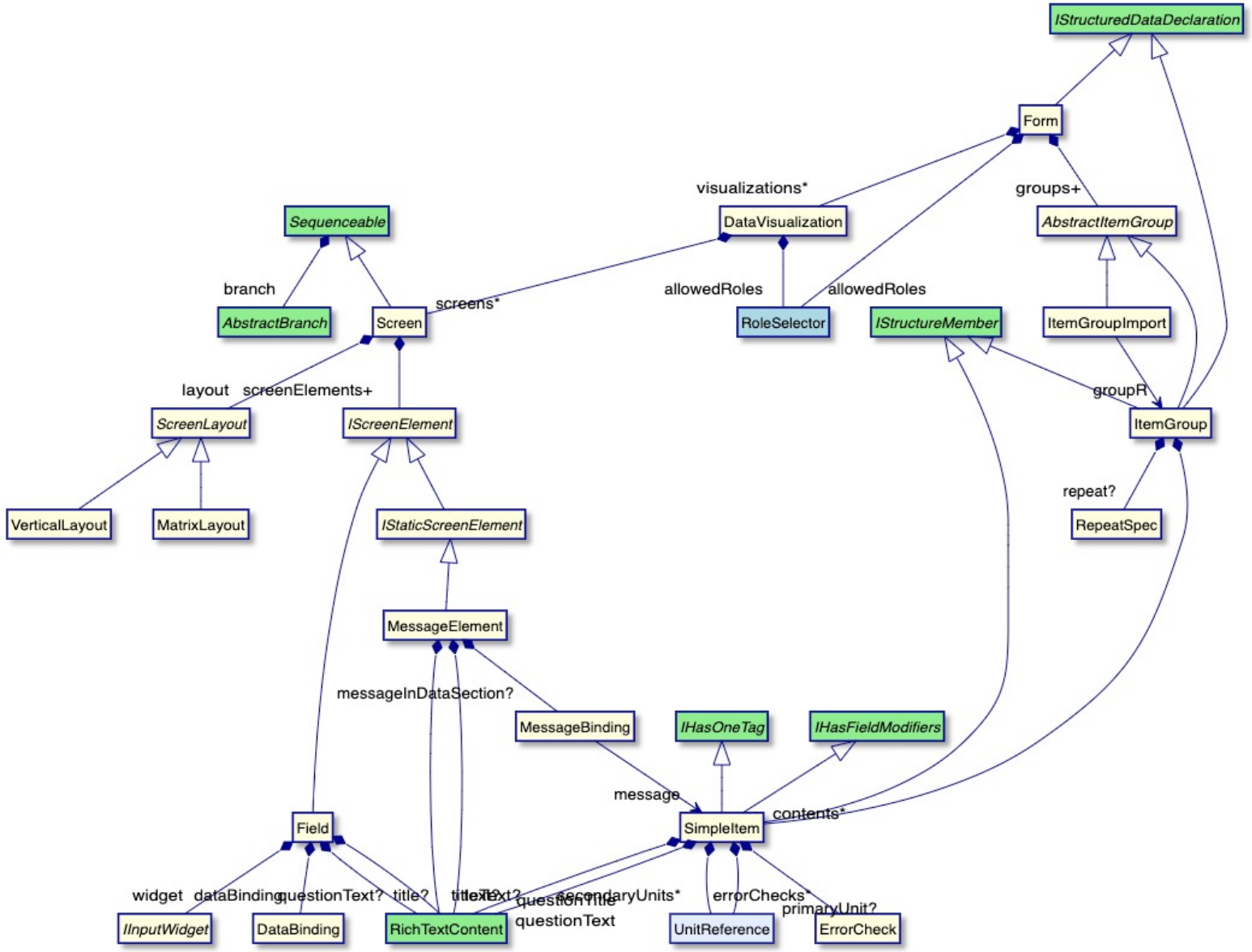


When is something a language?

Glossary

Structured Glossary

Metamodel



When is something a language?

- Glossary
- Structured Glossary
- Metamodel
- **Validations**

```
if
  the X contains a Y
then
  this A over there cannot have
  more than 2 children of type B.
```

When is something a language?

- Glossary
- Structured Glossary
- Metamodel
- Validations
- **Serialisation Format**

```
FunCall name="myFun"
  arg: NumLit value="10"
  arg: PlusOp
         arg: NumLit value="4"
         arg: NumLit value="5"
```

```
<FunCall name="myFun">
  <arg><NumLit value="10"/></arg>
  <arg>
    <PlusOp>
      <arg><NumLit value="4"/></arg>
      <arg><NumLit value="5"/></arg>
    </PlusOp>
  </Arg>
</FunCall>
```

When is something a language?

- Glossary
- Structured Glossary
- Metamodel
- Validations
- Serialisation Format
- **Syntax**

```
myFun(10, 4 + 5)
```

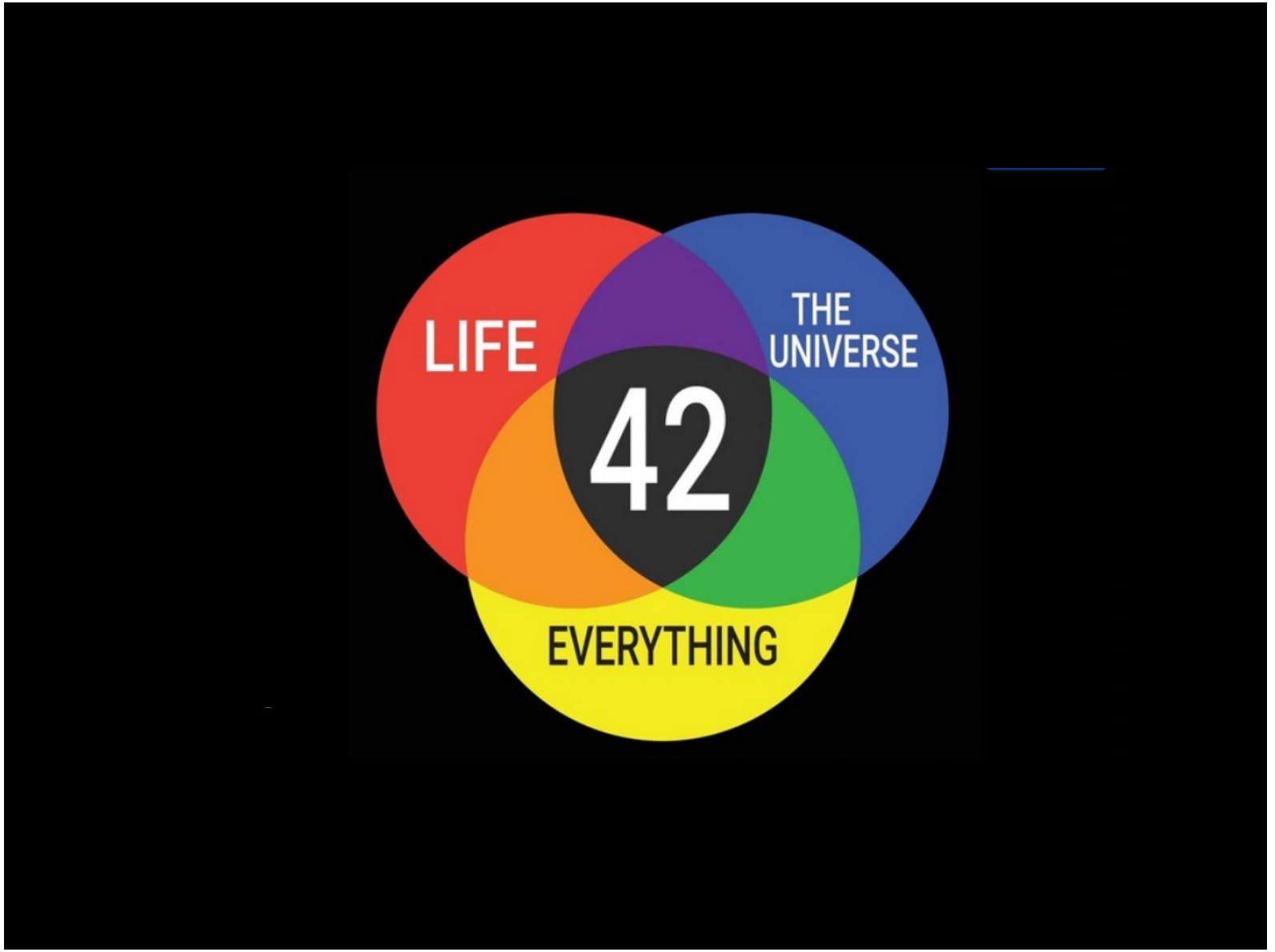
When is something a language?

- Glossary
- Structured Glossary
- Metamodel
- Validations
- Serialisation Format
- Syntax
- **Type System**

```
+(int, int)   → int
+(int, real)  → real
+(real, int)  → real
+(real, real) → real
+(string, *)  → string
+(*, string)  → string

val(<name>, <type>, <init>) → typeof(type)
  # typeof(type) > typeof(init)
```

When is something a language?

- Glossary
- Structured Glossary
- Metamodel
- Validations
- Serialisation Format
- Syntax
- Type System
- **Semantics**



When is something a language?

- **Glossary**
- Structured Glossary
- Metamodel
- Validations
- Serialisation Format
- Syntax
- Type System
- Semantics

Too informal.

When is something a language?

- Glossary
- **Structured Glossary**
- Metamodel
- Validations
- Serialisation Format
- Syntax
- Type System
- Semantics

Too informal.

When is something a language?

- Glossary
- Structured Glossary
- **Metamodel**
- Validations
- Serialisation Format
- Syntax
- Type System
- Semantics

That's just a data model.
Or a domain model.
Or an OO structure.
Or a schema.

When is something a language?

- Glossary
- Structured Glossary
- **Metamodel**
- **Validations**
- Serialisation Format
- Syntax
- Type System
- Semantics

That's just a data model.
Or a domain model.
Or an OO structure.
Or a schema.

With Validations.

When is something a language?

- Glossary
- Structured Glossary
- **Metamodel**
- **Validations**
- **Serialisation Format**
- Syntax
- Type System
- Semantics

That's just a data model.
Or a domain model.
Or an OO structure.
Or a schema.

With Validations.
And a way to store.

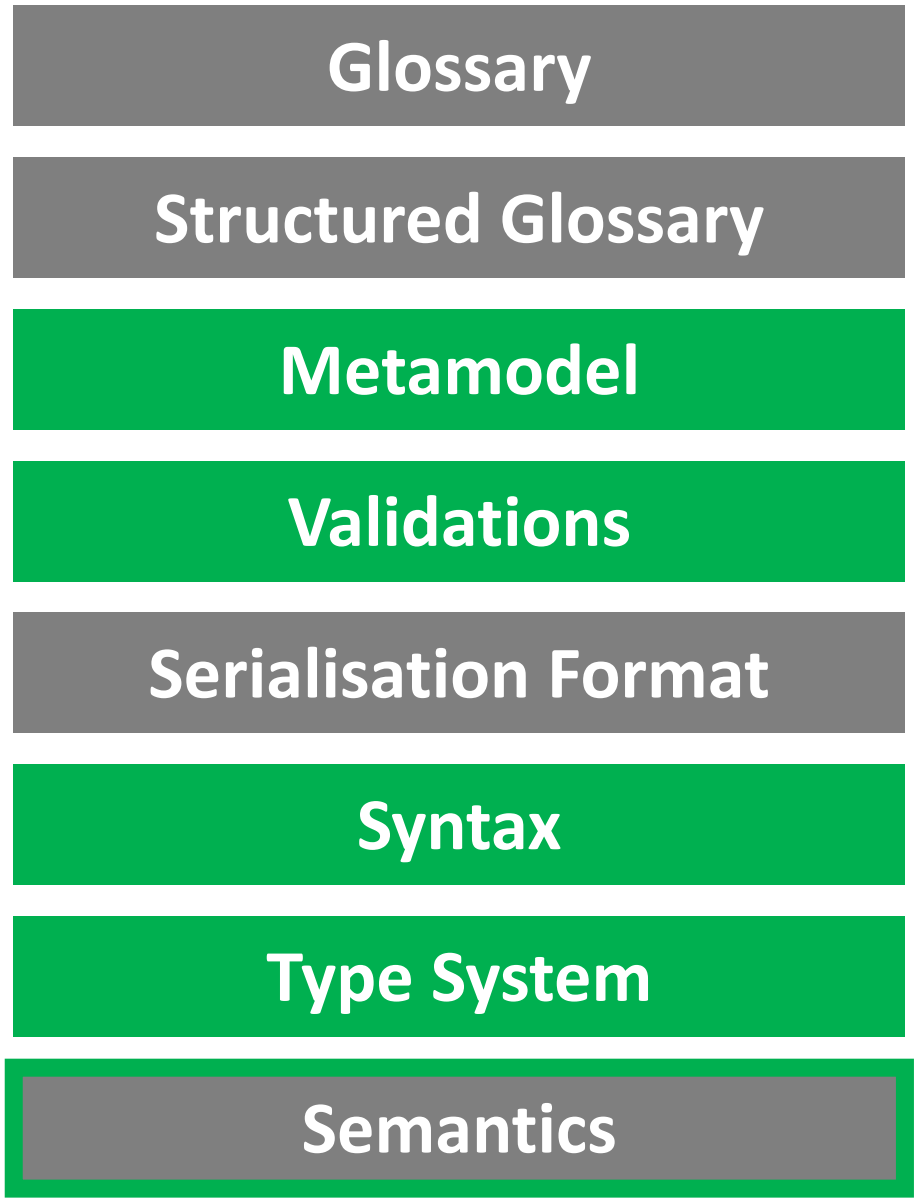
When is something a language?

- Glossary
- Structured Glossary
- Metamodel
- Validations
- Serialisation Format
- Syntax
- Type System
- Semantics

Finally, a language!

It‘s about syntax, stupid!

When is something a language?



A serious language :-)

Find more details at:

https://medium.com/@markusvoelter/when-is-something-a-domain-specific-language-83b7eff79ed4

What about frameworks and libraries?

Glossary

Structured Glossary

Metamodel

Validations

Serialisation Format

Syntax

Type System

Semantics

Only what the programming language supports.

Only those that relate to the language, not the domain.

Interpreters are simple, generators are hard

What about internal DSLs?

The DSL is defined with the means of the host language.
as opposed to using external, specialised language definition tools.

not whether the DSL code and GPL code are syntactically mixed or not.

```
test Data Access <no polarion id> tags: <none> screenshot << ... >>

Initial Setup: <none>

check timeline [must be run]
  requirement Data collected by forms must be accessible
  description
  day 1   A  assert expression events<V1|<no stateFilter>>.last.stateIs(ready)
          U  complete task Task1 with  Form DataAccess_Form
                                       IG1 | f1 = 10
          A  assert expression data<IG1>.last.owningEvent.isEvent<V1> is true
          A  event is completed V1
```

```
checking rule check_Step {
  applicable for concept = Step as step
  overrides <none>
  do not apply on the fly false

  do {
    node<AbstractBranch> b = step.branch;
    boolean usesSkip = b.descendants<concept = StepSkipExpr, +>.isNotEmpty;
    if (step.isSkippable()) {
      if (step.branch.isInstanceOf(BranchExpression)) {
        if (!usesSkip) {
          error "can't use skippable here" -> step.branch;
        }
      }
    }
  }
}
```

What about internal DSLs?

Glossary

Structured Glossary

Metamodel

Validations

Serialisation Format

Syntax

Type System

Semantics

```scala
"20.seconds ++ 1.minutes" should "be equal to 80.seconds" in {
  20.seconds ++ 1.minutes shouldBe 80.seconds
}
```

Scala

```kotlin
val myHtml = html {
  table {
    th {
      td { /* .. */ }
      td { /* .. */ }
    }
    for (i in 1..10) {
      tr {
        td { /* .. */ }
        td { /* .. */ }
} } } }
```

Kotlin

```ruby
describe UsersController, type: :controller do
  before do
    allow(controller).to receive(:current_user).and_return(nil)
  end

  describe "GET #new" do
    subject { get :new }

    it "returns success" do
      expect(subject).to be_success
    end
  end
end
```

Ruby

Depending on the language. Often just trees w/o parens. or “exploits” of syntactic freedom. Metaprogramming.

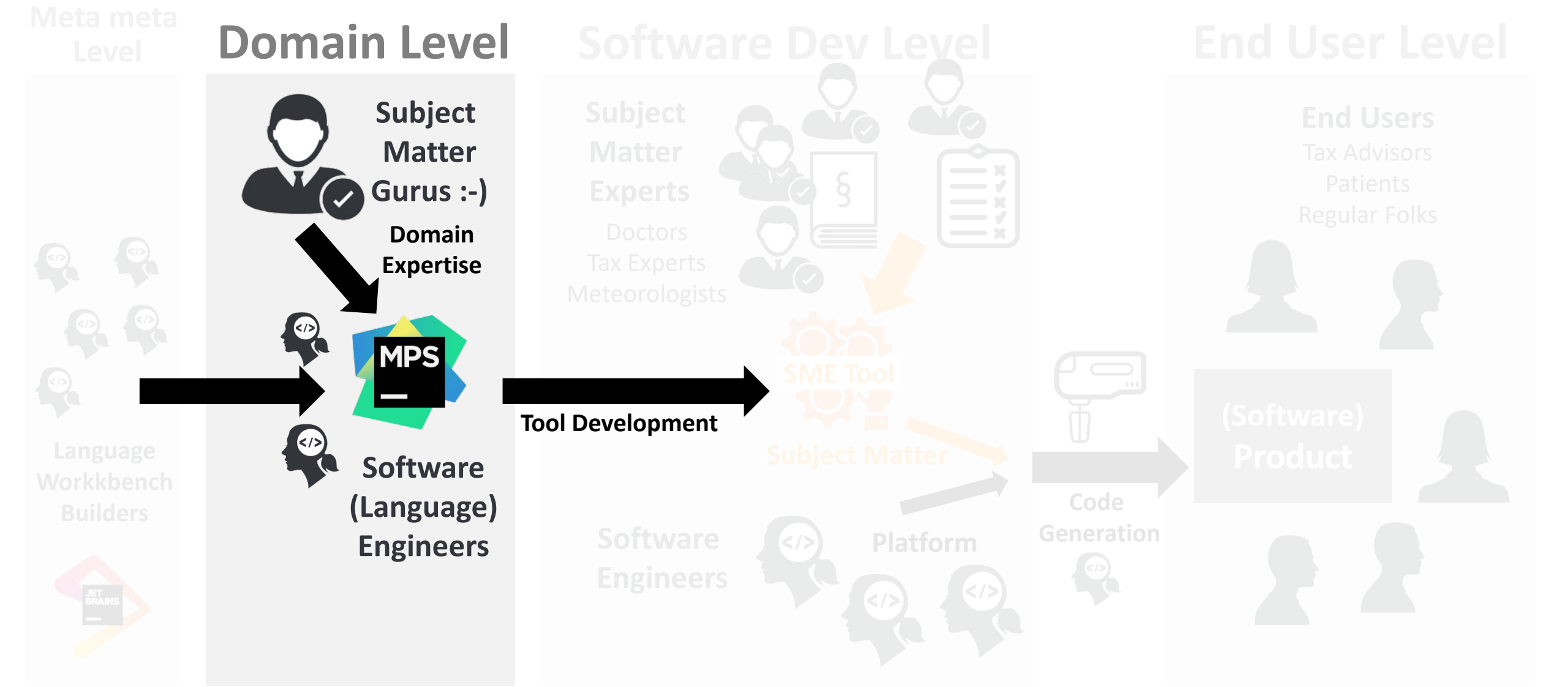
Often none in dynamically typed langs, some of it in Scala.

Interpreters are simple, generators are hard

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

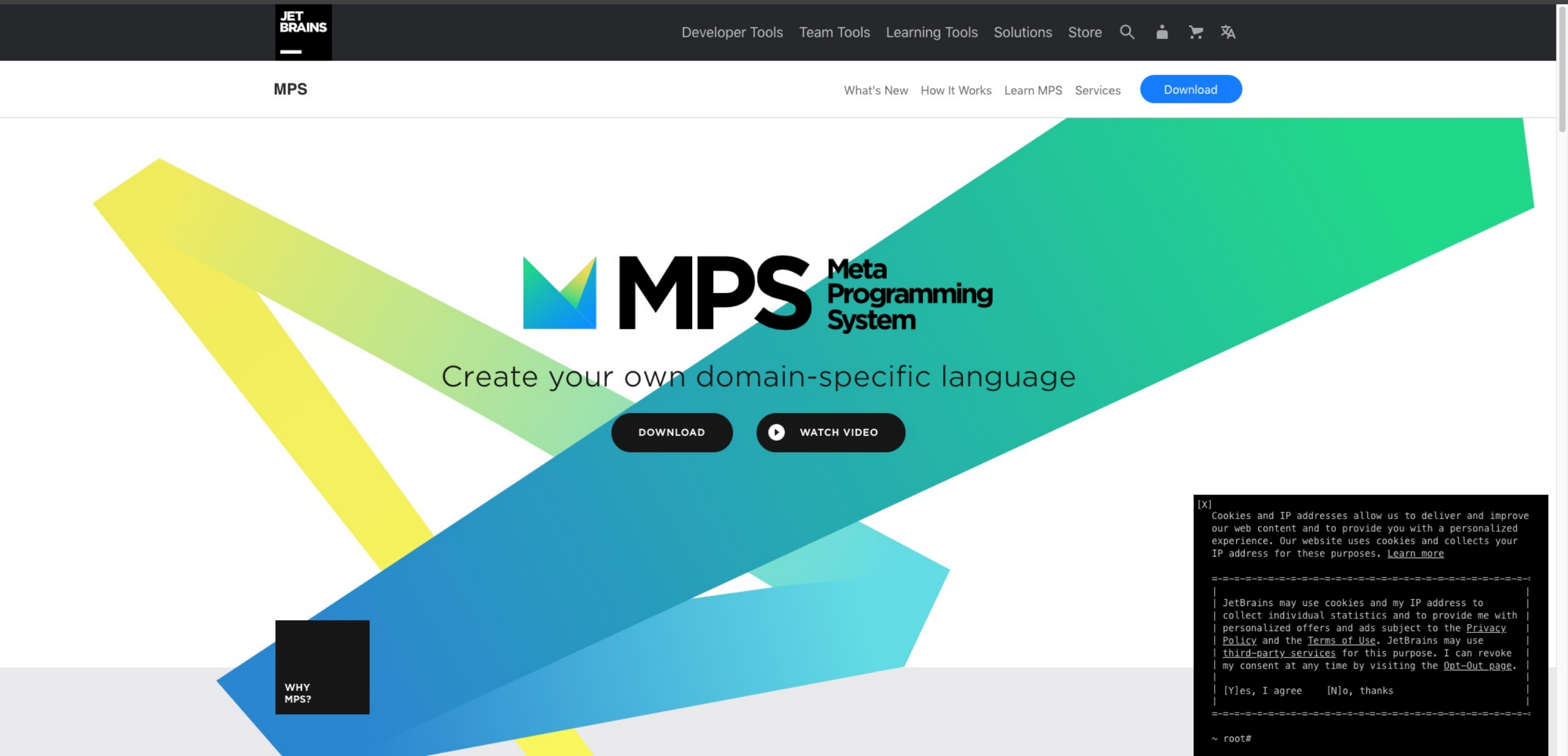
Robust support for language modularity and composition

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



Language Workbench – the future

Modelix is a open source platform that aims to bring modeling on the web. Modelix goal is to allow editing models in the browser and to interact with models and services around them over web-standard APIs. Right now Modelix can act as a drop in replacement to create webbased editor for existing languages in JetBrains MPS.

See Modelix in action!

Get started with our example projects and try modelix yourself.

Read more ...

Contributions welcome!

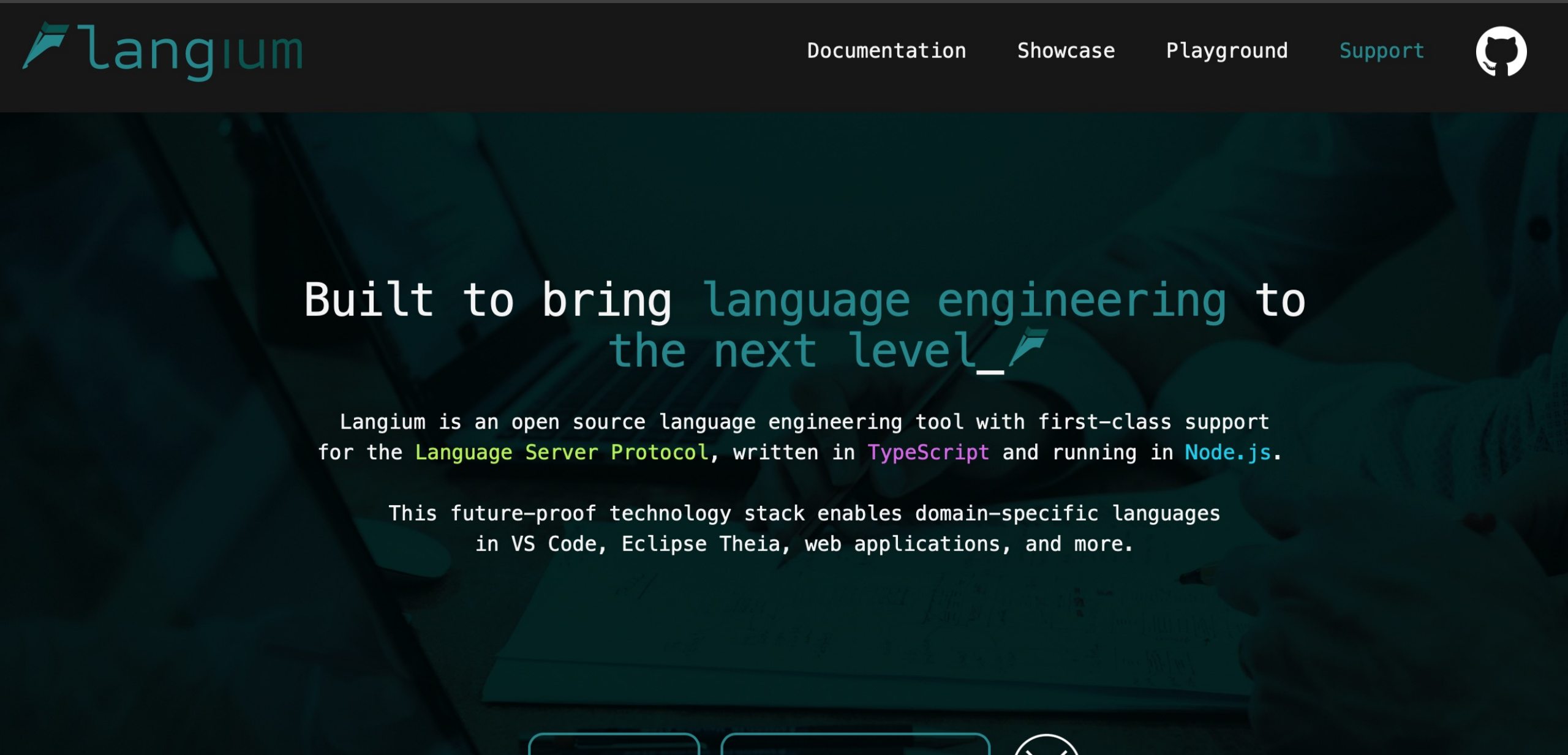
We do a Pull Request contributions workflow on **GitHub**. New contributors are always welcome!

Join the discussion on Slack!

Sign up to the MPS community Slack and join us in #modelix

Read more ...

Language Workbench – the future



Language Workbench – the future

Monaco - The Editor of the Web

The Monaco Editor is the code editor that powers VS Code. A good page describing the code editor's features is here. It is licensed under the MIT License and supports Edge, Chrome, Firefox, Safari and Opera. The Monaco editor is not supported in mobile browsers or mobile web frameworks. Find more information at the Monaco Editor repo.

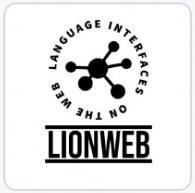
CodeMirror

Examples **Documentation** **Try** **Discuss** **GitHub** **Version 5**

Extensible Code Editor

CodeMirror is a code editor component for the web. It can be used in websites to implement a text input field with support for many editing features, and has a rich programming interface to allow further extension.

Language Workbench – the future



LIonWeb

Language Interfaces on the Web

Follow

Overview | Repositories 7 | Projects 1 | Packages | Teams 1 | People 9 | Settings

README.md

LIonWeb -- Language Interfaces on the Web

The language engineering community around MPS is working on bringing a projectional language workbench into the cloud/browser. A rough vision of how the result could look like is summarized in this paper. As of now, several relatively independent activities (are|have been) going on, including

- ProjectIt/Freon by Jos Warmer, Anneke Kleppe (https://www.projectit.org/), a set of components and associated meta-languages geared towards creating projectional editors in the cloud/browser,
- MPSServer by Strumenta. It is an http and websockets server that can be started from standard and headless MPS to permit interaction with MPS from outside it. It permits to read and modify models, trigger builds, get typesystem information, etc. There is also a TypeScript client library available on NPM. It is called MPSServer-client
- WebEditKit by Strumenta. It is a prototypal framework for defining projectional editors that can interact with MPS through MPSServer
- Modelix by itemis, an open Source platform for models on the Web
- Sergej's Service APIs (Sergej) [short description, URL]
- JetBrains Projectional Web Editor (Alex) [short description, URL]
- StarLasu by Strumenta. It is a set of libraries to define and work with ASTs in Kotlin, Java, Python, TypeScript. They have been used in production for years

View as: **Public**

You are viewing the README and pinned repositories as a public user.

You can pin repositories visible to anyone.

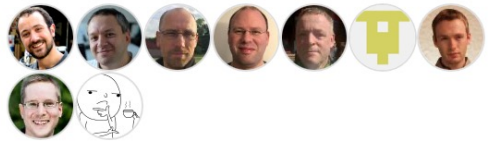
Get started with tasks that most successful organizations complete.

Discussions

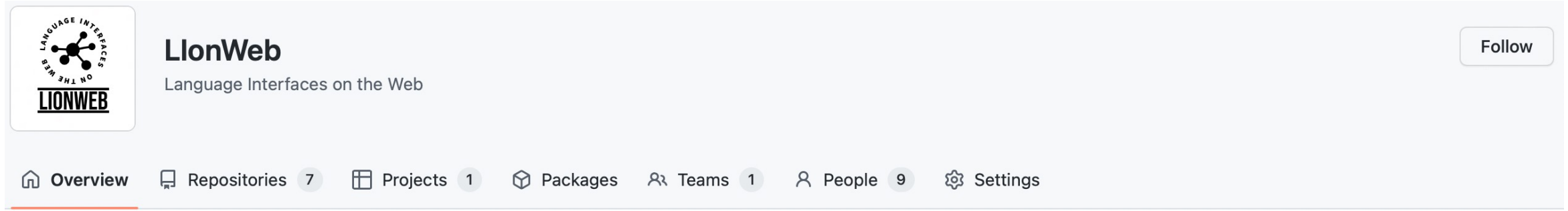
Set up discussions to engage with your community!

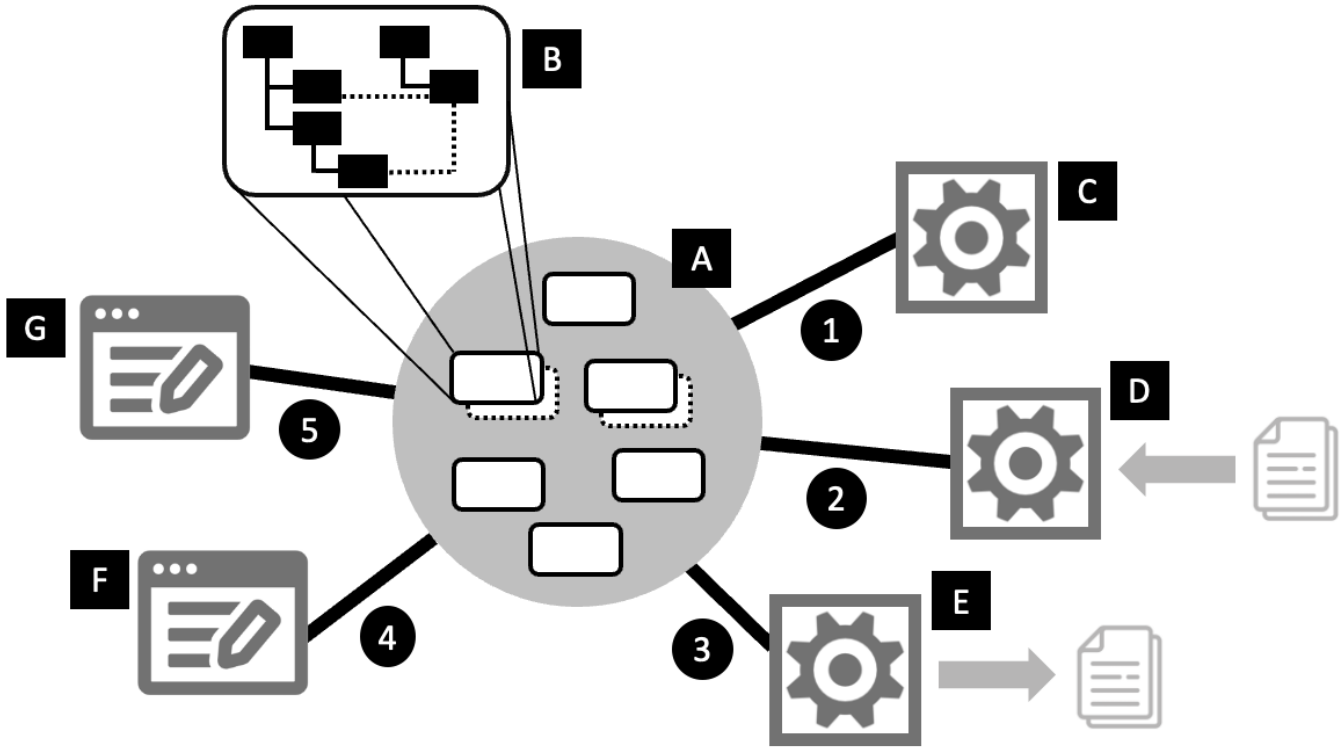
Turn on discussions

People



Language Workbench – the future





The LIonWeb initiative aims to facilitate the community-based development of language engineering and modeling tools on the web.

1. Protocols for communication between participating software components
2. Meta-meta model as well as a reference architecture
3. APIs to access models and metamodels and to encapsulate the protocols
4. Hub for the developers of such components and to empower other software developers to develop web-based modeling solutions.

Roll your own – what do you need?

A robust M3

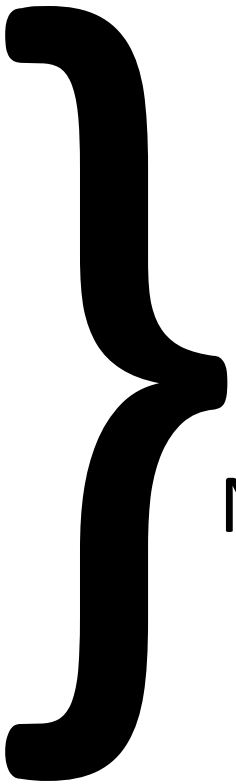
- represent models in memory,
- persist them somehow using a metamodel-specific serialization format (not a syntax, see my last post),
- provide an API to read, traverse and modify models,
- and to support a a rudimentary but generic way of editing them.

Model API

```
modelelement.getChildren(role: string)

modelelement.getReferences(role: string)

modelelement.getPropertyValue(propName: string)
```

Meta Model Def

```
Machine.getStates() : list<State> {
 (list<State>)this.getChildren("states");
}

State.getName() : string {
  (string)this.getProperty("name")
}

Transition.getTarget() : State {
  (State)this.getReference("target").deref()
}
```



Store
Query
Notification

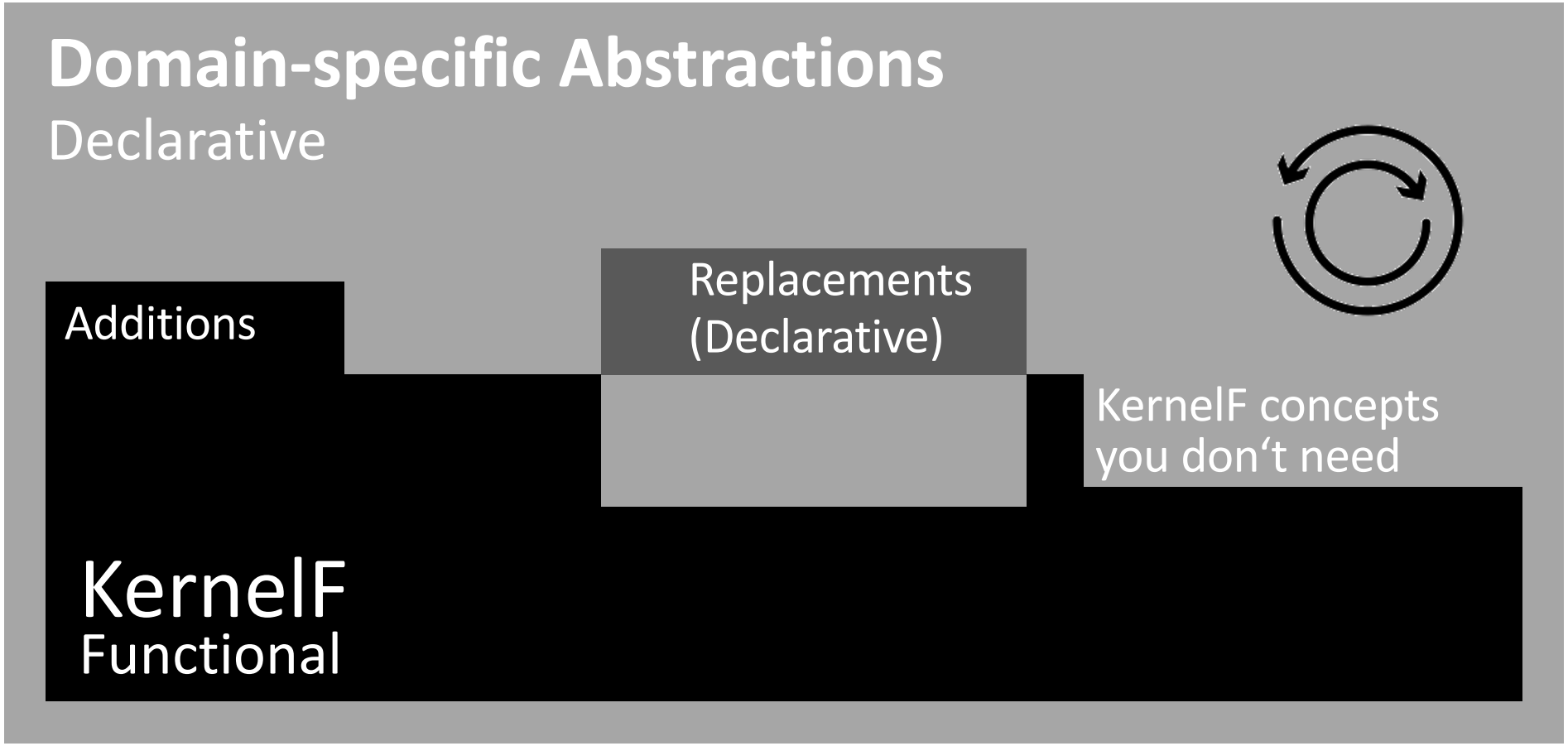
Find more details at:
https://medium.com/@markusvoelter/the-minimum-infrastructure-for-running-languages-and-models-da922aa3b4b4

Language Design

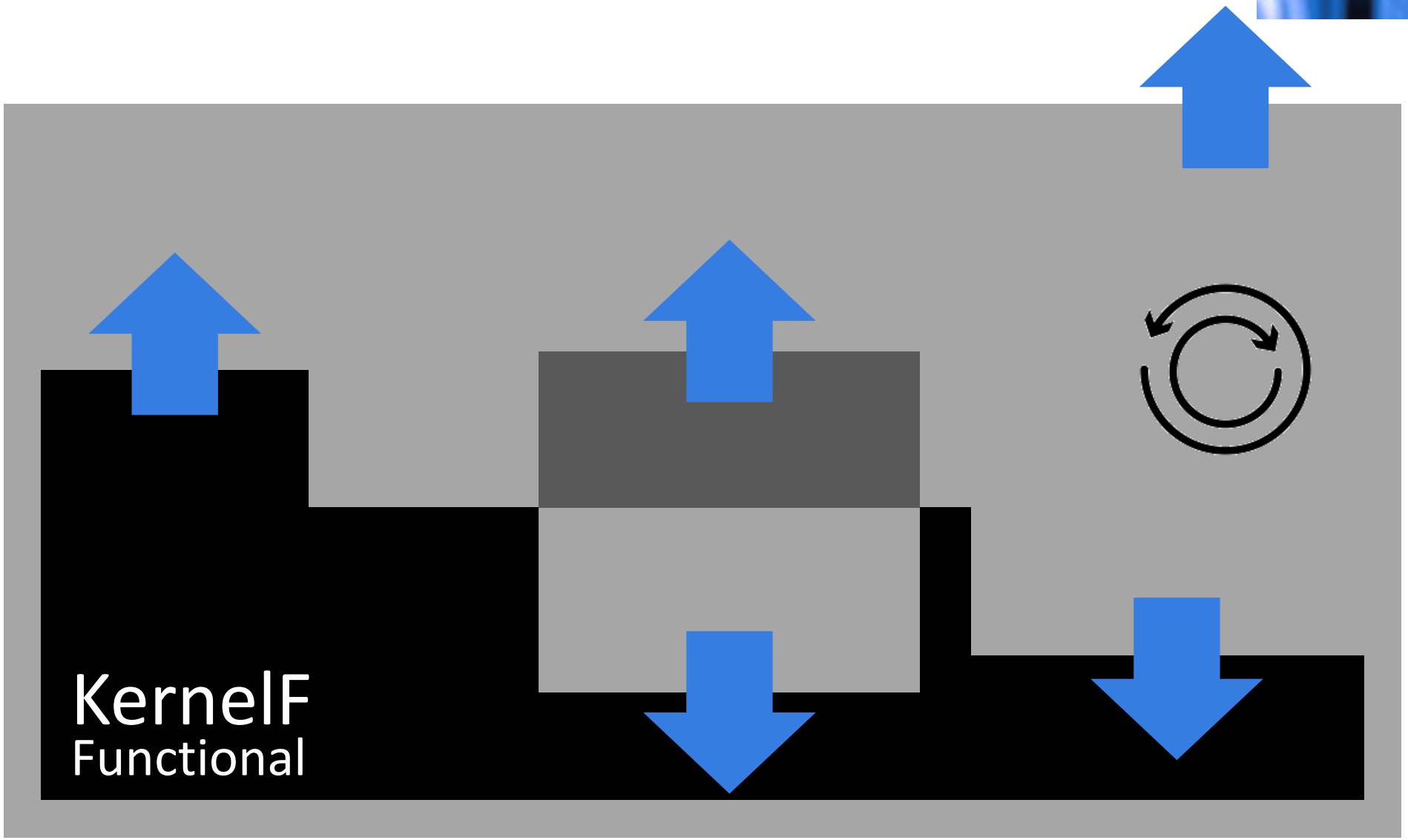
Growing a DSL on top of KernelF

- 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

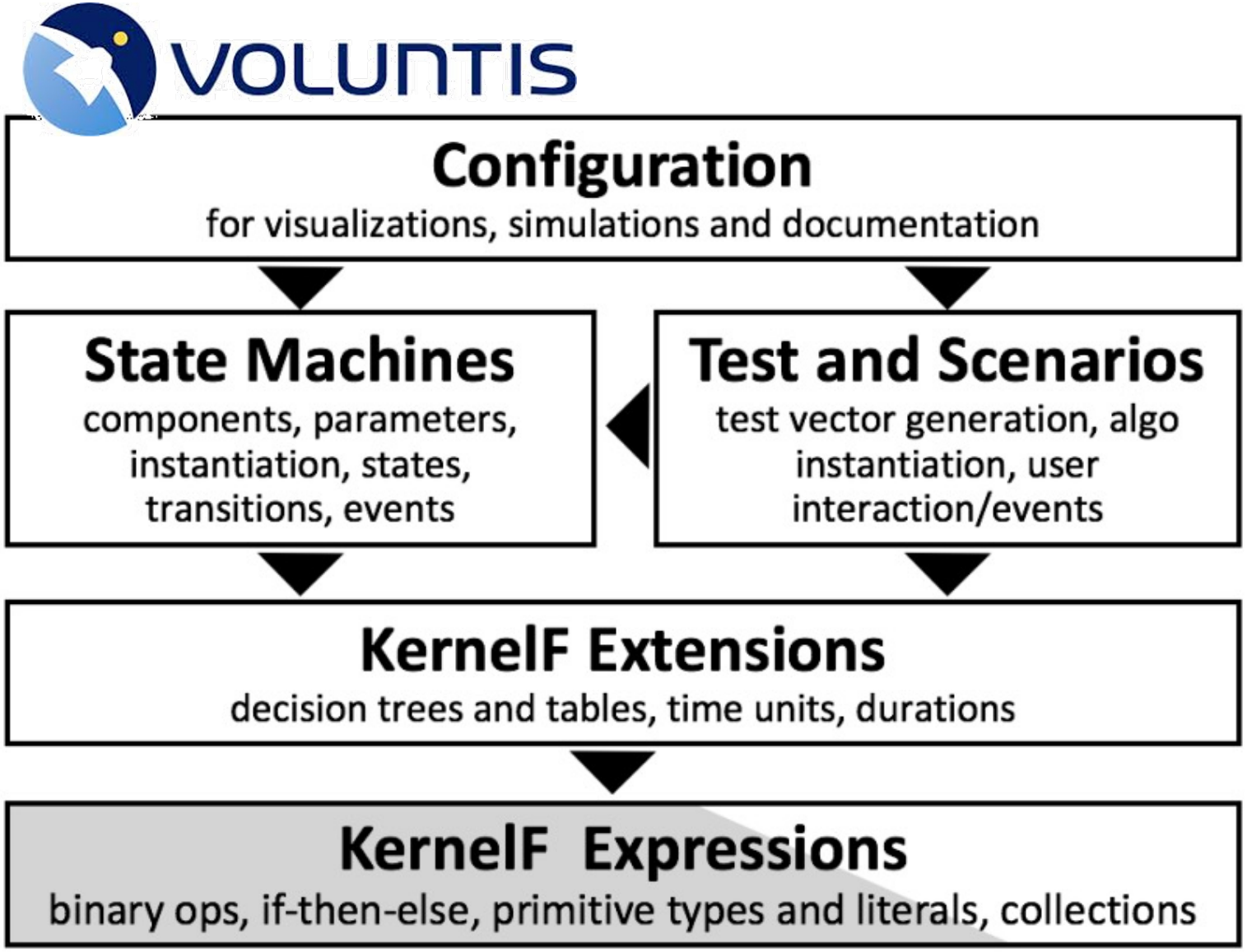
Growing a DSL on top of KernelF



Growing a DSL on top of KernelF



Language Architecture and Sizes



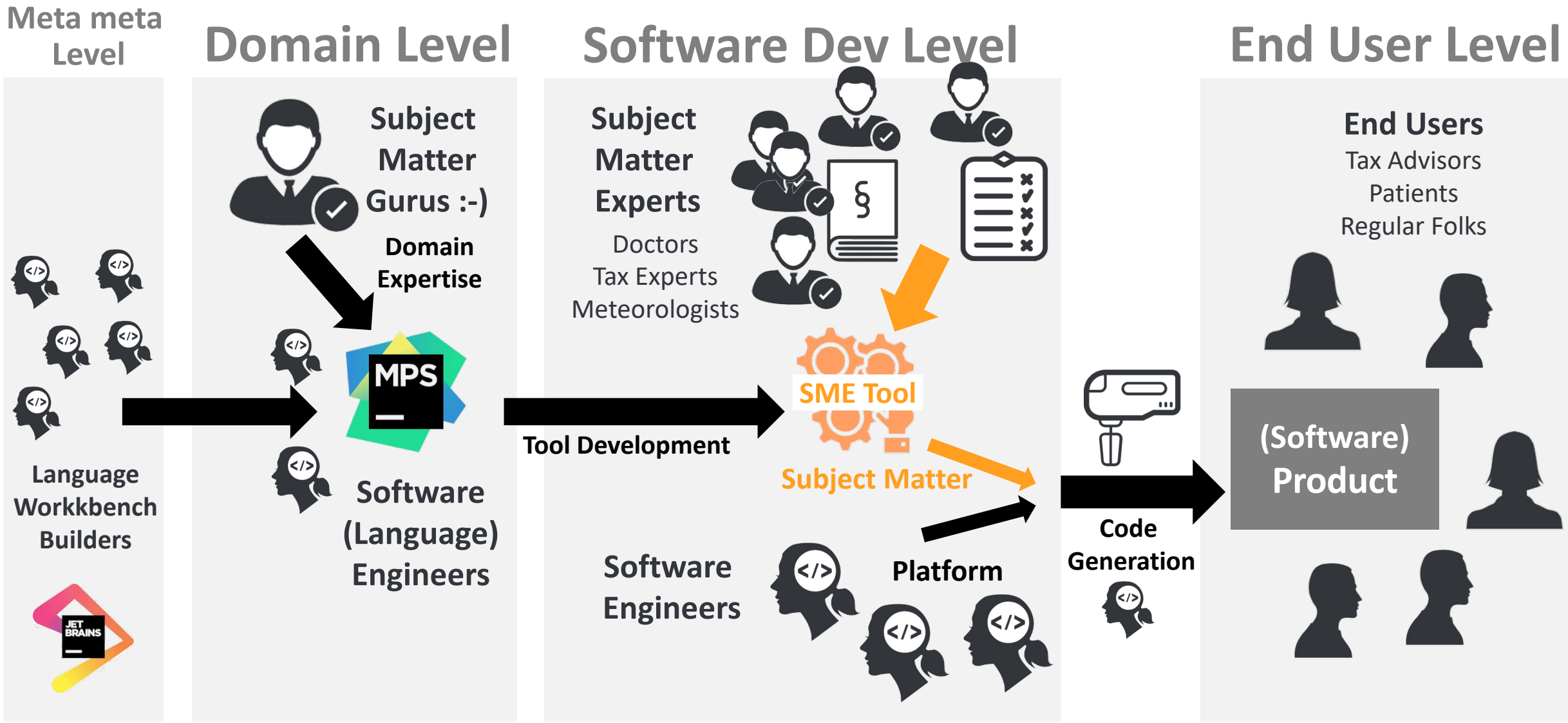
Language Part	# of concepts	percentage of total
Expressions (KernelF)	83	31%
Expressions (Extended)	63	23%
State Machines	29	11%
Testing, Scenarios	41	15%
Configuration	54	20%
Total	270	100%

DATEV

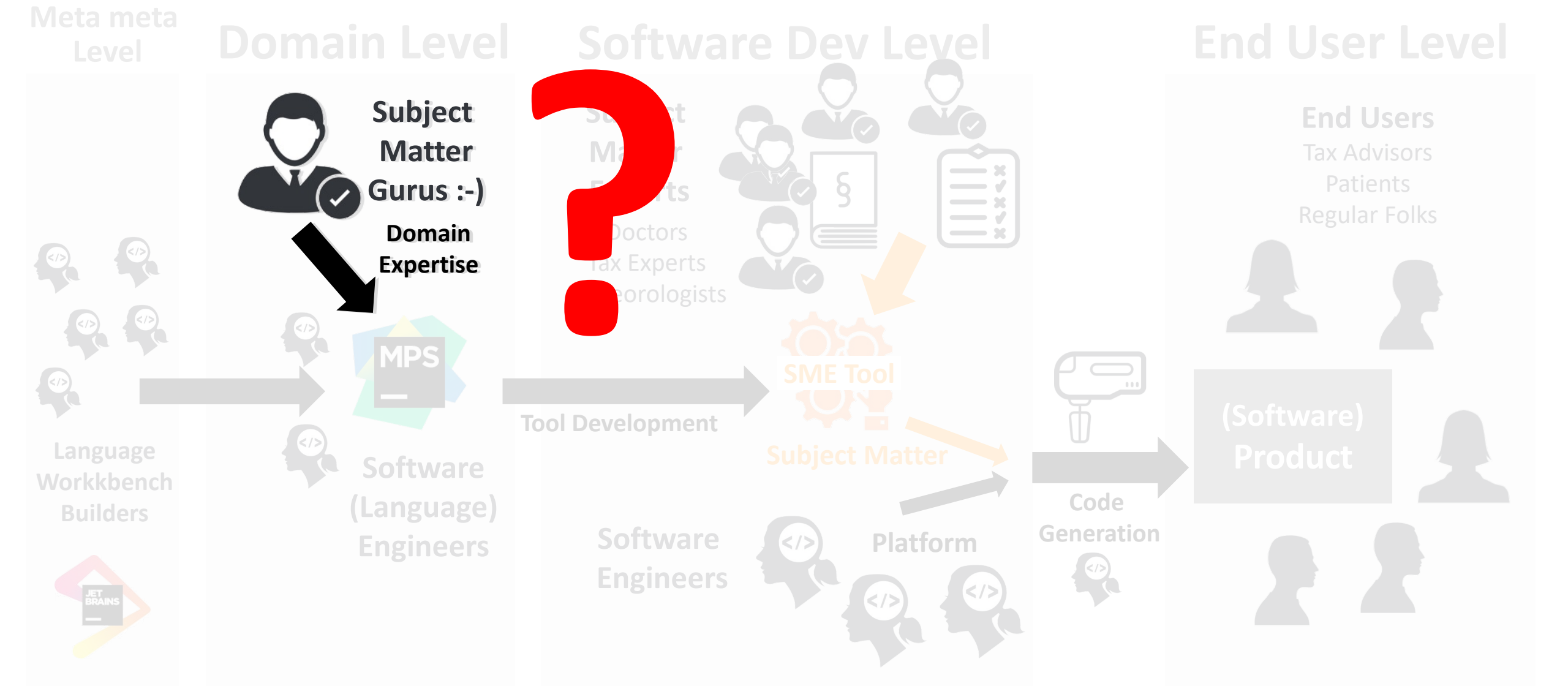
Kind	Language	# of concepts
DATEV-proprietary	`payroll.dsl.core`	166
	`payroll.dsl.test`	32
	`payroll.dsl.institution`	26
	`payroll.dsl.migration`	7
Developed for DATEV and then open sourced	`kernelf.datetime`	27
	`kernelf.temporal`	24
Use as-is from KernelF	KernelF	120
	Total	**402**

Domain Analysis

Big Picture: How does knowledge get into software



Big Picture: How does knowledge get into software



Big Picture: How does knowledge get into software

Meta meta Level

Domain Level

Software

User Level

Subject Matter Gurus :-)

Domain Expertise

Doctors

Tax Experts

Meteorologists

End Users

Tax Advisors

Patients

Regular Folks

MPS

Tool Development

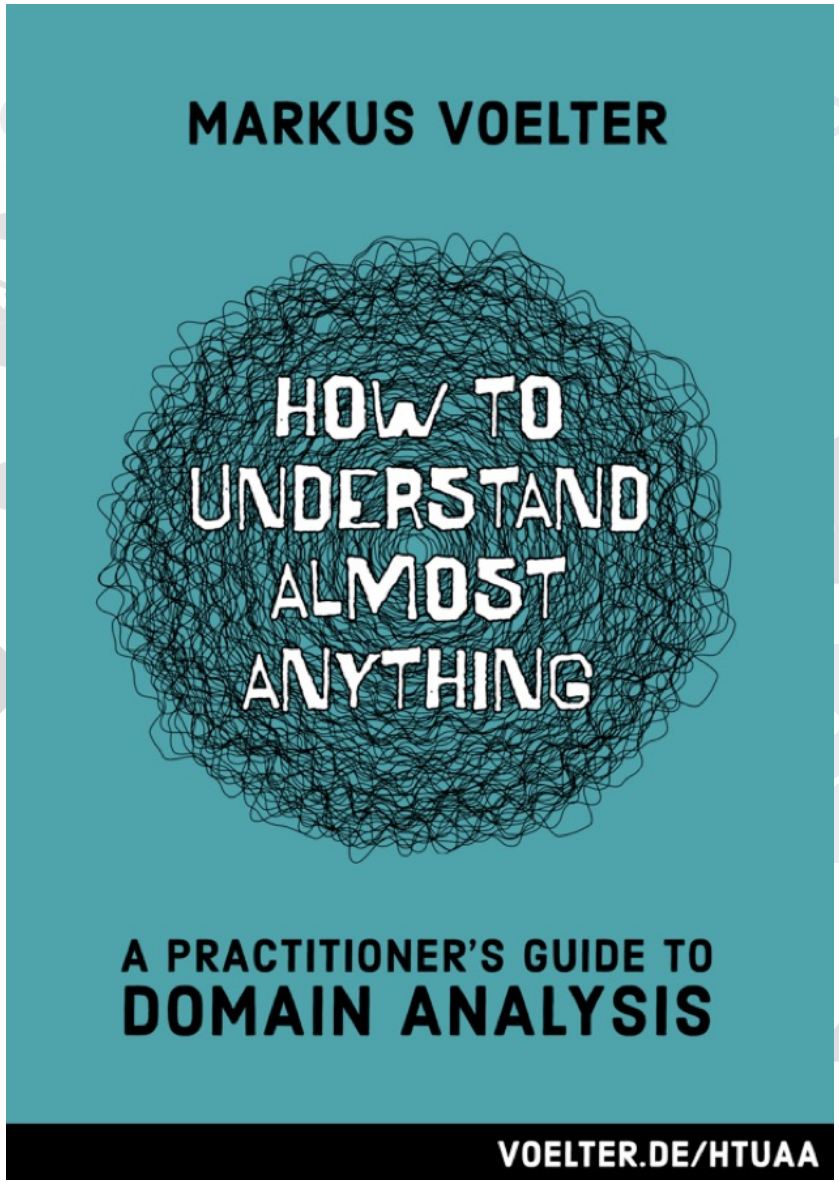
Language Workkbench Builders

Software (Language) Engineers

Software Engineers

ftware)

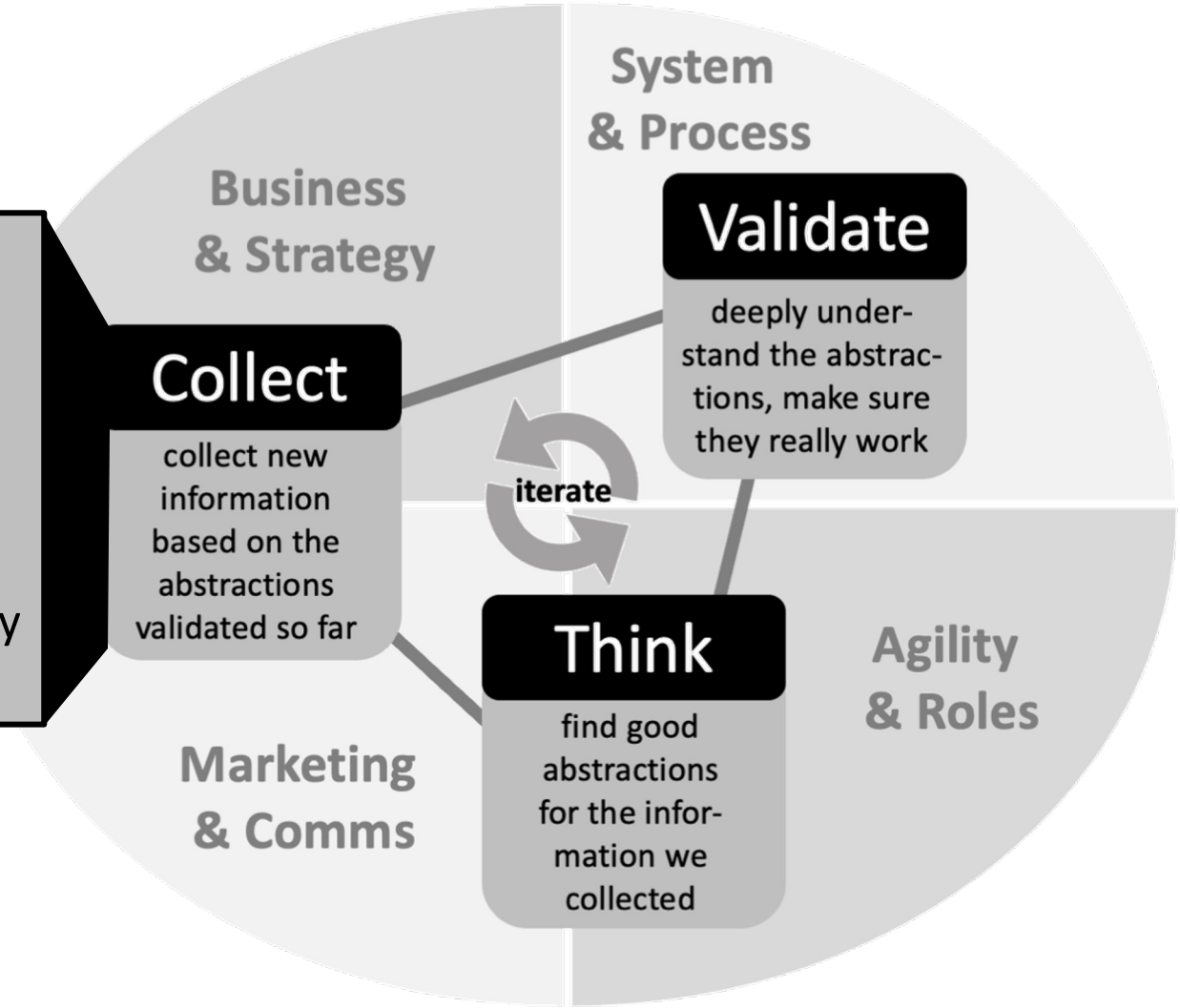
roduct



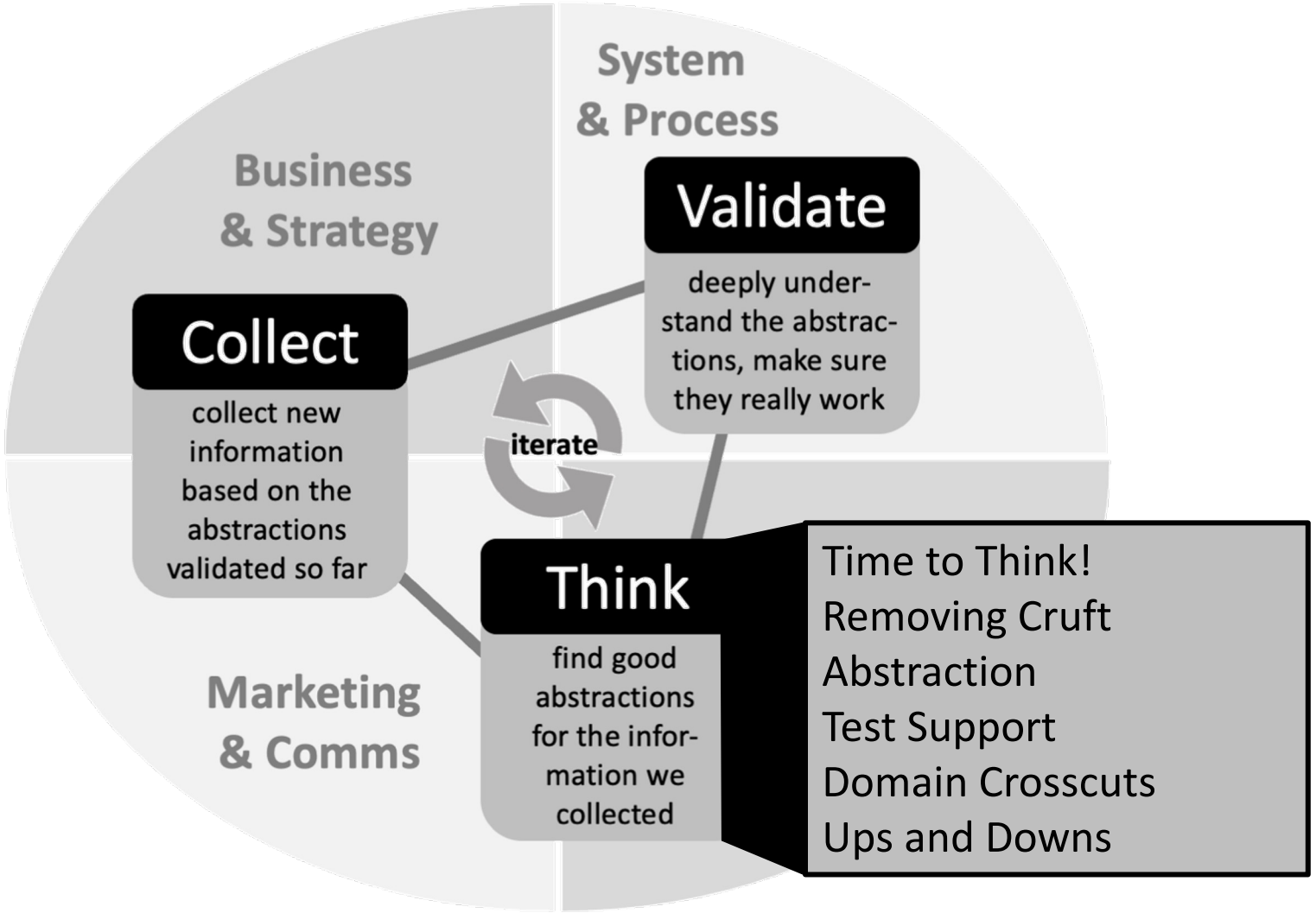
People are key

High-Level Process

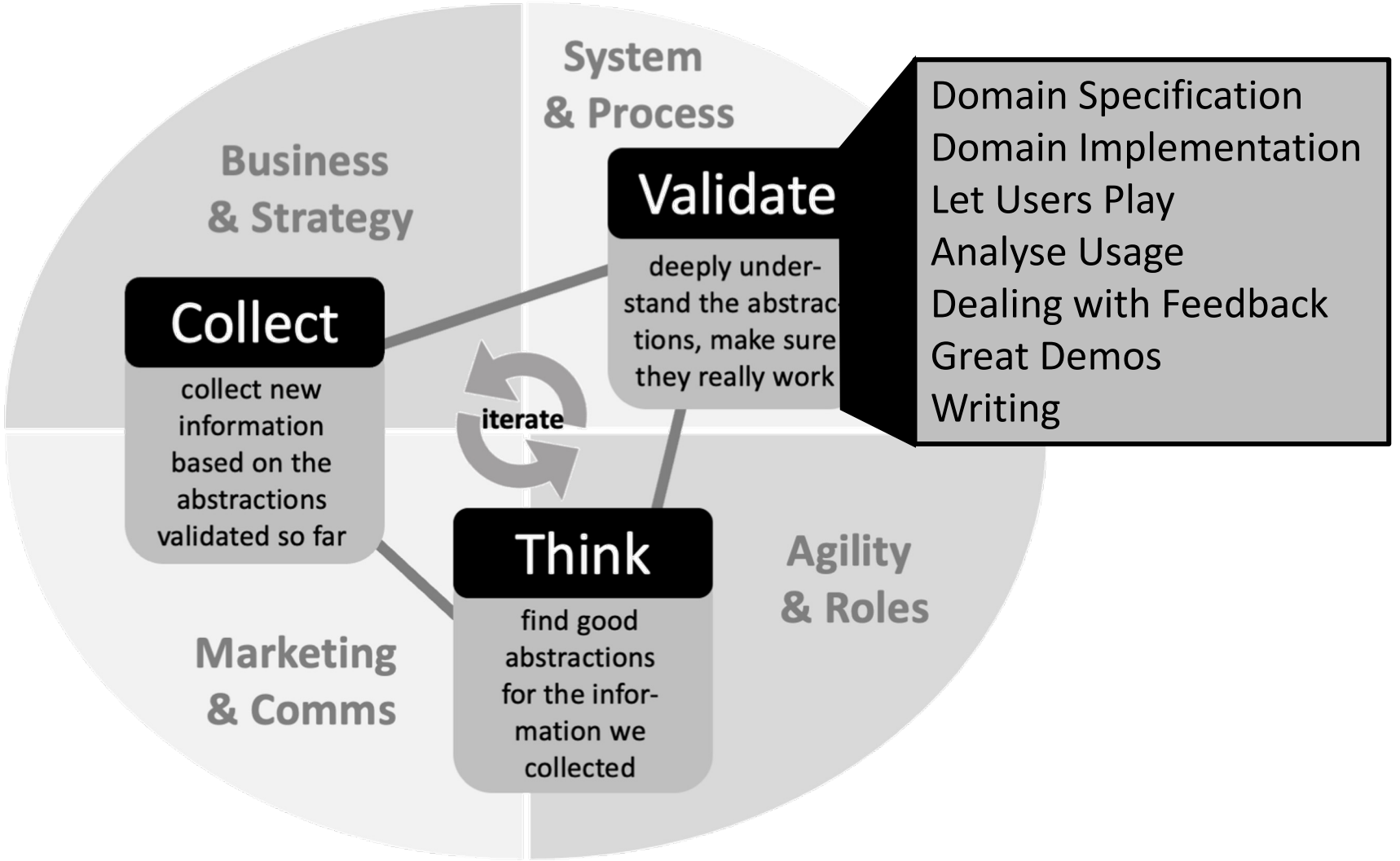
- Written Material
- Hidden Languages
- Consistent Terminology
- Working Sessions
- Active Listening
- Consistency vs. Change
- Dealing with Uncertainty
- Capture Results



High-Level Process



High-Level Process



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