Let‘s build Fachlichkeit like they build Rockets

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

Bernd Kolb
kolb@itemis.de
www.itemis.de
@berndkolb

voelter { ingenieurbüro für softwaretechnologie // itemis

B

0

Motivation

The Story of Paula and Knut (maybe your story)

" I just finished the third revision of the specification. But I'm sure it will take another two month of annoying ping-pong with the IT. "

Paula 45, Product Engineer

" Once again the spec was crap. Another 3 month wasted ... "

Knut 32, Software Engineer

1

Examples

M

Healthcare

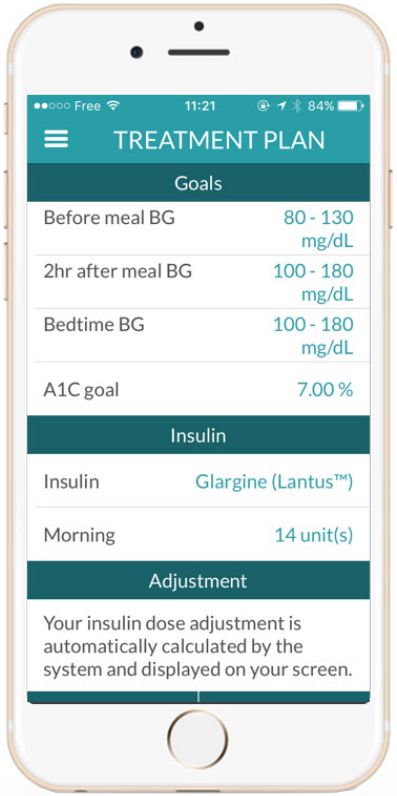


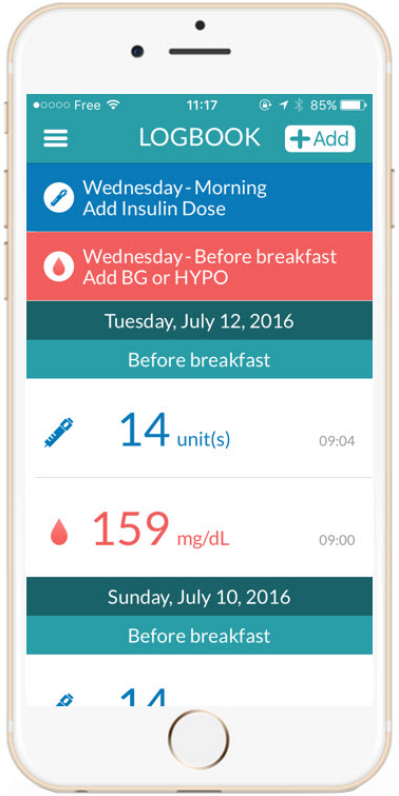
Context & Motivation

Mobile Apps that help patients w/ treatments

Monitor side-effects and recommend actions

Manage dosage of medications





VOLUNTIS
Connected Therapeutics

Context & Motivation

Mobile Apps that help patients w/ treatments

Monitor side-effects and recommend actions

Manage dosage of medications

“Algorithms“ for recommendations and dosage at the core of these apps.

Safety-critical, since they could hurt patients.

Customer develops many different apps/algos like this, efficiency of algo development is key.

VOLUNTIS
Connected Therapeutics

Context & Motivation

Health care professionals directly „code“ algos, using a suitable language.

Avoids indirections through requirements docs.

Speed up dev significantly.

}

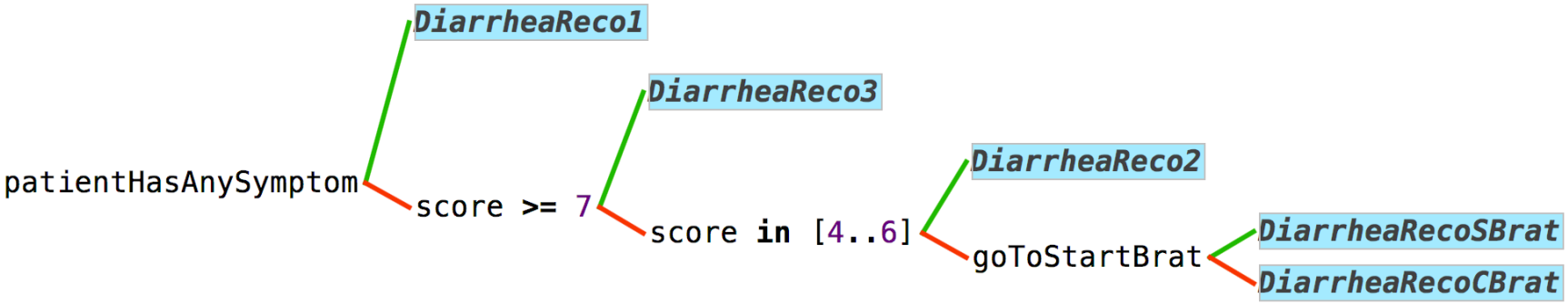
Pretty typical DSL-based dev-approach.

Some Language Impressions I

```
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)
```




```
type temperature: number[36|42]{1}
type measuredTemp: number[35|43]{2}
```

Error: type number[32.55|39.99]{4} is not a subtype of number[36|42]{1}

```
val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93
```

Some Language Impressions II

```
PASS
function test gradeStools
  given 7 expected 3
  given 6 expected 2
  given 5 expected 2
  given 4 expected 2

PASS
function test DiarrheaStoolsDecisionTree
  given false, 1, true, false   expected   DiarrheaUSRecoLevel1Symptom
  given false, 9, false, false  expected   DiarrheaUSRecoGrade3

PASS
function test checkScreeningQuestion
  given   answers to DiarrheaScreeningQuestionnaire {   expected  true
            dietarySupplements: false
            medication        : true
            hospitalized      : false
          }
```

Some Language Impressions III

Simulator MVPSimulator controls

0 days 00:00:00 >>> >>>>

Sep 1, 2017 at 11:39:33

+ Inputs

Starting time: 2017-09-01 ... 11 39 33

Inputs for Diarrhea

Number of Stools Baseline 2

Patient has Ostomy

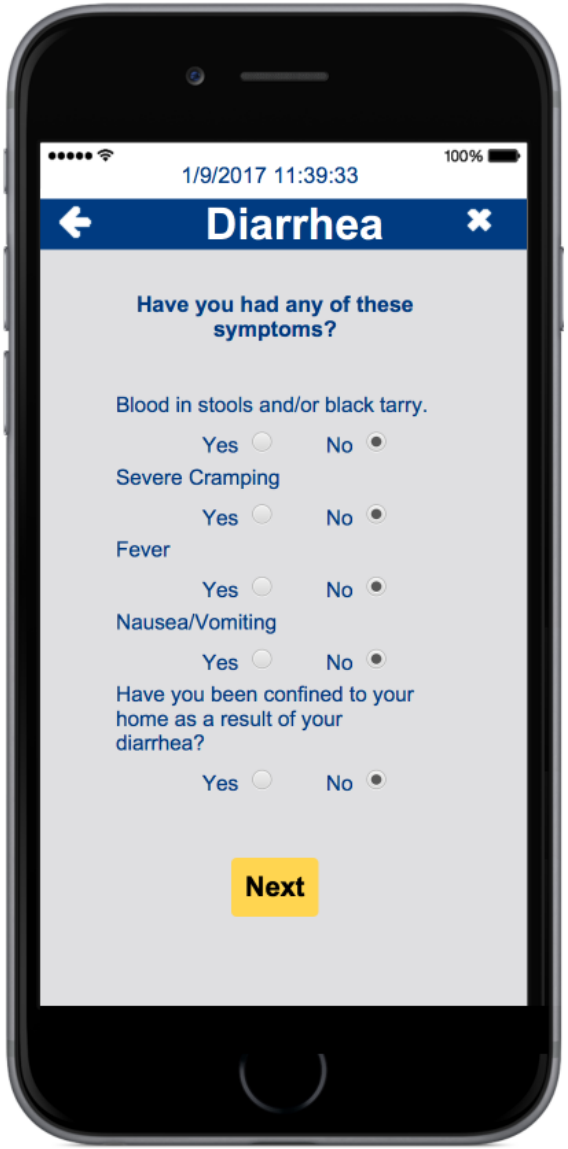
Patient has Immuno-Oncology treatment

Update Inputs Restart

Inputs for Fever

inputTemperatureUnit Celsius

Submit Inputs Restart



B

Insurance



Context & Motivation

Specify/Program

Insurance Programs

Write formal code in a DSL mixed with tables and text

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No varibility

Formale Beschreibung

Funktion:	berbwvekFF		
Programmquelle:	vmsctfa1.c		
Produkt-Typ:	FONDS	**PK-Typ:**	KAPITAL-KONTO
	Name	Verw.	Entität
verwendete Attribute:	lkm_akt_param	E	PARAMETER
	lkm_faell_param	E	PARAMETER
	ber_zweck_param	E	PARAMETER
	kz_rzw_param	E	PARAMETER
	bwvek	A	RETURN
aufgerufene Funktionen:	berbweinzelFF		
	VTRKermbtgfaellFF		
Status:	18.1		

Verarbeitungen

Die Funktion liefert den Barwert per lkm_akt_param des vorschüssigen Zahlungsstroms der Höhe 1 von Monat lkm_akt_param bis lkm_faell_param – jeweils einschließlich. Zahlungszeitpunkte sind jeweils die Monatsbeginne, also **lkm_akt_param** -1 bis **lkm_faell_param** – 1.

Der Parameter **kz_rzw_param** steuert die zu berücksichtigende Zahlweise des Zahlungsstroms. Möglich sind zur Zeit nur die Ausprägungen 0 (Zahlungen zu den Beitragsfälligkeiten) und 12 (monatliche Zahlungsweise).

Schleife über lkm_faell_hilf = **lkm_akt_param** bis **lkm_faell_param**

Falls **kz_rzw_param** = 12

kz_bf_hilf = 1

sonst

kz_bf_hilf = *VTRKermbtgfaellFF*(lkm_faell_hilf)

Ende Falls **kz_rzw_param** = 12

bwvek = **bwvek**

+ kz_bf_hilf * *berbweinzelFF*(**lkm_akt_param**, lkm_faell_hilf - 1, **ber_zweck_param**)

Ende Schleife

return **bwvek**

Context & Motivation

Specify/Program

Insurance Programs

Write formal code in a DSL mixed with tables and text

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No varibility

Formale Beschreibung

Funktion: rg_kk_beta_satzTF
Programmquelle: vmscfo2.c
Produkt-Typ: FONDS, RSR **PK-Typ:** Kapital-Konto

verwendete Attribute:

Name	Verw.	Entität
fo_beta_satz	E	Kosten-Regeln
beta_satz	E	Rechnungsgrundlagen-KK
ko_ra_id	E	KOSTEN-RABATT
zmt_param	E	PARAMETER
kz_zus_gar	E	
zm	E	
beta_satz_fakt	E	VORGABEDATEN-KOSTENRABATT
zw	E	TVDKONTO_G
vtrk_zb	E	VTRK_BTG
kz_mandant	E	T_KK
satz_beta	A	Rückgabewert

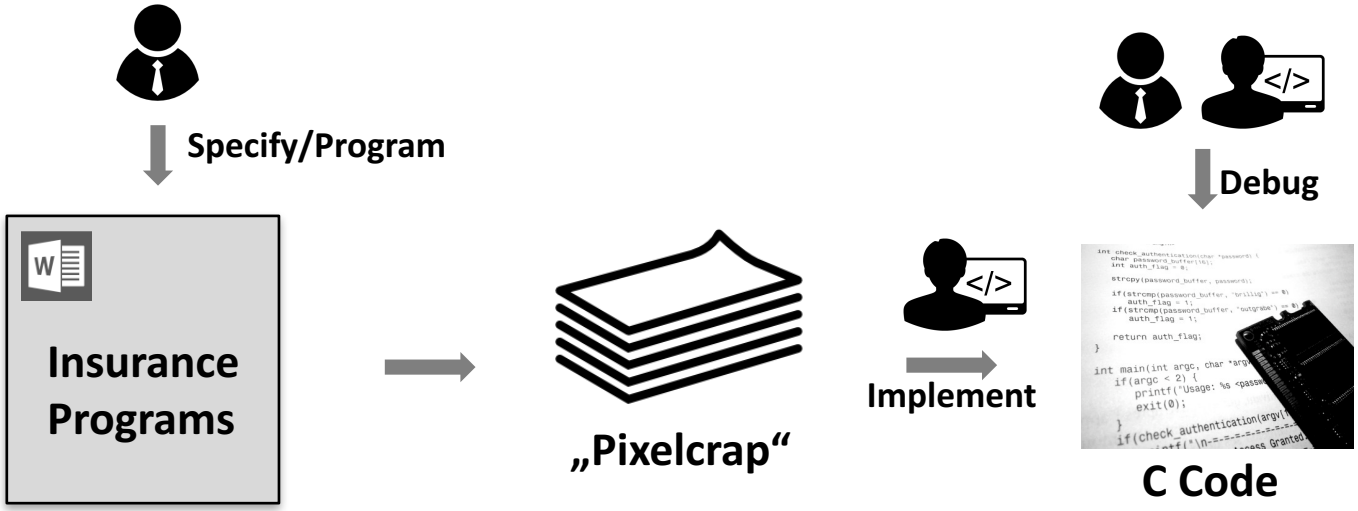
Aufgerufene Funktionen: rg_kk_beta_bp_satzTF

In dieser Funktion wird der Kostensatz β ermittelt.

Verarbeitung

fo_beta_satz	Berechnung **satz_beta**	Bemerkung
0	**satz_beta = beta_satz** Falls **zmt_param** <= 120 und **kz_zus_gar** = JA **satz_beta** = **satz_beta** * min(0,01 * max(**zmt_param** - 12; 0); 1) Ende (Falls **zmt_param** <= 120 und **kz_zus_gar** = JA)	Standard
1	**satz_beta** = **beta_satz** * min(0,01 * max(**zm** – 12; 0); 1)	PF
2	grenze = **vtrk_zb** * **zw** Falls grenze < 10000.0 satz_beta = **beta_satz** Sonst satz_beta = 0,074	GULPP

A real Example: Legacy



Debugging directly in C
Search-for-use by text search
Don‘t trust the documents – may be outdated!

Write formal code in a DSL mixed with tables and text

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No varibility

Printed, PDF

Developer reads „spec“
Very idiomatic implementation

Dev acts as a human compiler and implements it in C

Solution Approach

Specify/Program

Insurance Programs

Write formal code in a DSL mixed with tables and text

Now with IDE support and e

The same notati

Funktionsmodell VKzahlbtgTF

Formale Beschreibung

```
Funktion:                  VKzahlbtgTF
Enthaltende Quelldatei:    vmsctfa
Produkt-Typ:               Produkt-Typen auswählen
PK-Typ:                    PK-Typ auswählen
Status:                    Status auswählen

Parameter-Attribute:         tvk_el_ptr:  tvk_el<>  E  Beschreibung hinzufügen
                             buzbBfr:     Ganzzahl  A  Beschreibung hinzufügen
                             tech_ptr     techptr

Rückgabe-Typ:                Kommazahl

Verwendete VADM-Attribute:   ...

Aufgerufene Funktionen:      VKversartTF  (tvk_ptr: tvk_el<> E; tech_ptr  techptr  ):  VERSART
```

Beschreibung

Berechnet den Zahlbeitrag auf Vertragskomponenten-Ebene zurück

Hilfsvariablen

vkzb: Kommazahl Beschreibung hinzufügen

Verarbeitungen

:pk_typ_id	Beschreibung hinzufügen	Bemerkung
PK_TYP_ID.KAPITAL_KONTO	If (:vtrk_zustand = ZUSTAND.BPFL) vkzb = :vtrk_zb End If :vtrk_zustand = ZUSTAND.BPFL	Beschreibung hinzufügen

PK_TYP_ID.LV_TARIF	If (:stamm_ptr <> NULL)
	If (:zustand = ZUSTAND.BPFL)
	vkzb = :vtzb
	If (VKversartTF (tvk_el_ptr; tech_ptr) = VERSART.BUZB)
	buzbBfr = 0
	End If VKversartTF(tvk_el_ptr; tech_ptr) = VERSART.BUZB
	End If :zustand = ZUSTAND.BPFL
	End If :stamm_ptr <> NULL
		Beschreibung hinzufügen

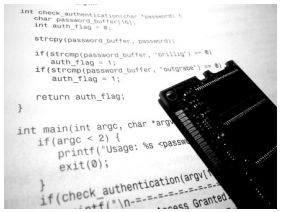
Andernfalls	Fehler (PK_TYP_NICHT_IMPLEMENTIERT)	Beschreibung hinzufügen

return vkzb

A real Example: Future

Specify/Program/Test/Debug

Generate



C Code

Still exactly the same C code,
or improved as needed.

Incremental Refinement/Refactoring of languages:

Partially automated migration of models
Add model natural notations (insurance-specific, math)
Add Support for modularity, reuse, variants

M

Salary



Context & Motivation

Calculate Salaries + Taxes for Employees

Various deployment platforms

Complex Business Logic

Variability over 16 States

Based on evolving Law

Dates & Currencies

Temporal Data + Arithmetics

Reactive Rules

Currencies and Dates

```
fun printDate(d: date) {...}
val today = /2018 01 23/  // date type inferred
{ printDate(today) }
```

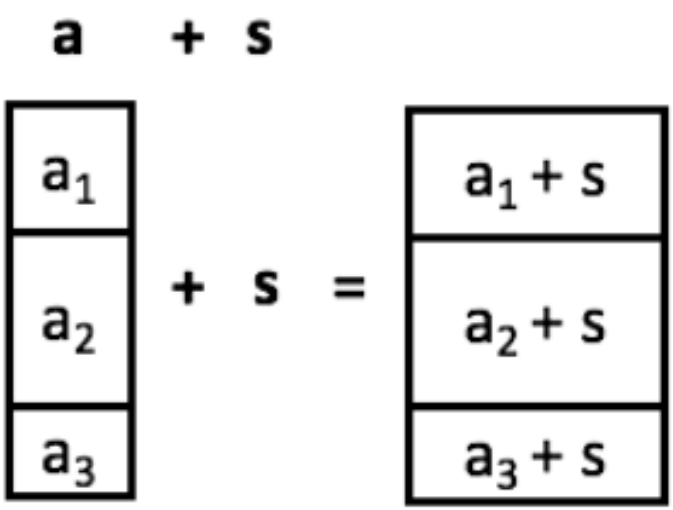
```
val nextWeek: date         = today + 7
val lastYear: date         = today - 365    // ignoring leap years for now :-)
val howLongIsAYear: number = lastYear - today
```

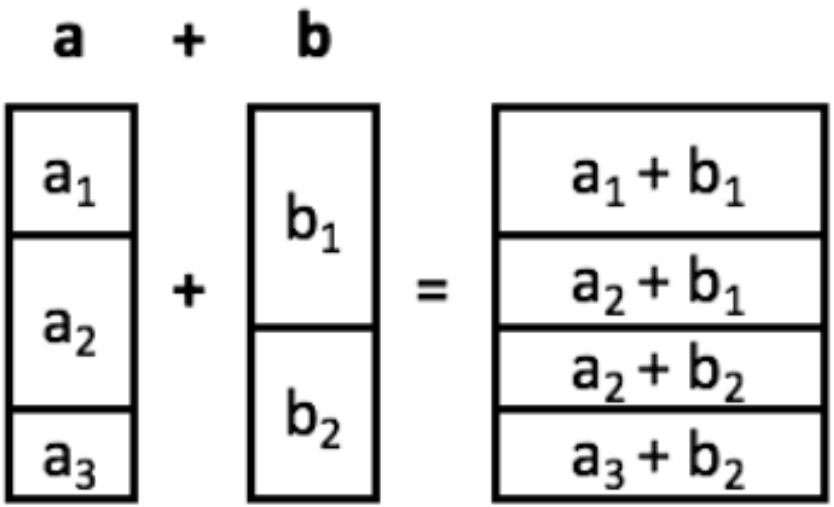
```
val salary : TT[currency] = TT | /2017 01 01/  =>  5.000 EUR |
                               | /2017 05 01/  =>  6.000 EUR |
```

Temporal Data

```
val salary#/2017 10 07/ = TT | /2017 01 01/  =>  5.000 EUR |
                             | /2017 05 01/  =>  6.000 EUR |
val salary#/2017 11 05/ = TT | /2017 01 01/  =>  5.000 EUR |
                             | /2017 05 01/  =>  5.500 EUR |
```

```
val salary : TT[currency] = TT | /2017 01 01/  =>  5.000 EUR |
                               | /2017 05 01/  =>  6.000 EUR |
```





Result Data and Rules

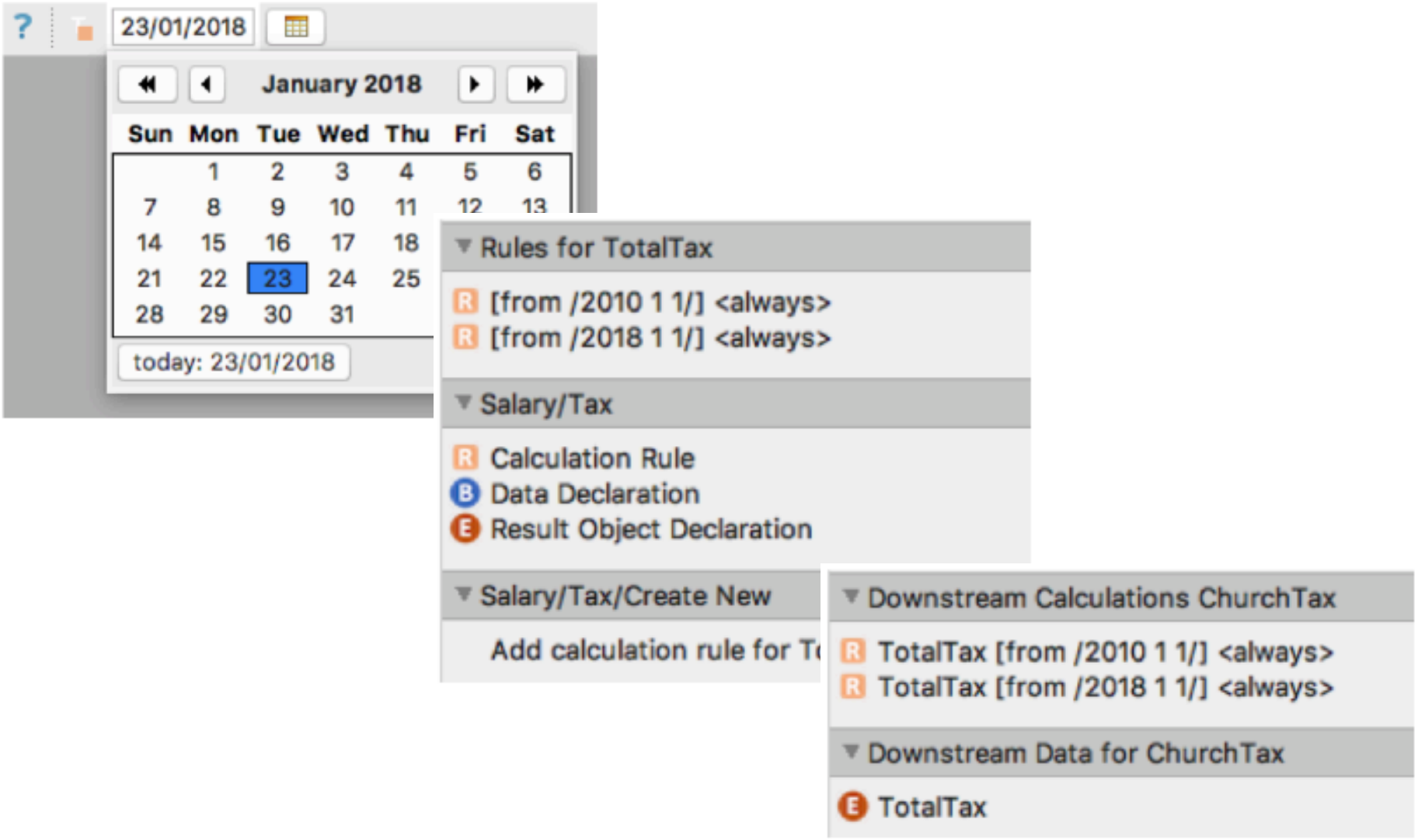
```
result data [monthly] Salary {
  employment -> Employment // basic data
  amount     : currency
}

result data [monthly] Tax {
  person -> Person // basic data
  amount : currency
}

calculation for Tax
  depends Salary foreach person.employments    // depends on Salaries of all employments
          as salaries                          // of the Tax bill's person
                                               // in the respective time

  calculate [monthly] {
    val factor = // do some weird tax math
    val total   := salaries.amount.sum       // sum up all salaries in current month
    amount      := total * factor            // populate fields of the result data item
    employment  := ctx.employment            // ctx is available in all calculations
  }
```

Result Data and Rules

```
calculation for SalaryReport // data structure indexed to an Employment
  depends Salary              as s
          Salary[month.prev] as s_last
  calculate [monthly] {
    currentSalary    := s.amount
    lastMonthsSalary := s_last.amount
    delta            := s.amount - s_last.amount
  }
```

```
calculation for Salary
  depends ...
  calculate [monthly] {
    val e                 = ctx.employment
    val totalHoursWorked = e.workedHours.reduce(SUM)
    val averageWage       = e.wage.reduce(WEIGHTED_AVERAGE)
    val religion          = e.person.religion.reduce(LAST, increment.year)
  }
```

IDE Support


23/01/2018
January 2018
Sun Mon Tue Wed Thu Fri Sat
today: 23/01/2018
Rules for TotalTax
[from /2010 1 1/] <always>
[from /2018 1 1/] <always>
Salary/Tax
Calculation Rule
Data Declaration
Result Object Declaration
Salary/Tax/Create New
Add calculation rule for T
Downstream Calculations ChurchTax
TotalTax [from /2010 1 1/] <always>
TotalTax [from /2018 1 1/] <always>
Downstream Data for ChurchTax
TotalTax

B

2

Approach in a Nutshell

Integration of Fachler



[Business Knowledge]

It‘s what makes a business tick.

Distinguishes the business.

Business Rules
(Financial) Calculations
Data Structures
Mappings or Queries
Validations
Scientific Processes
Contracts

Processes
UIs

[Business Knowledge]

It‘s what makes a business tick.
Distinguishes the business.

Contributed **not** by developers

... but typically implemented
in software

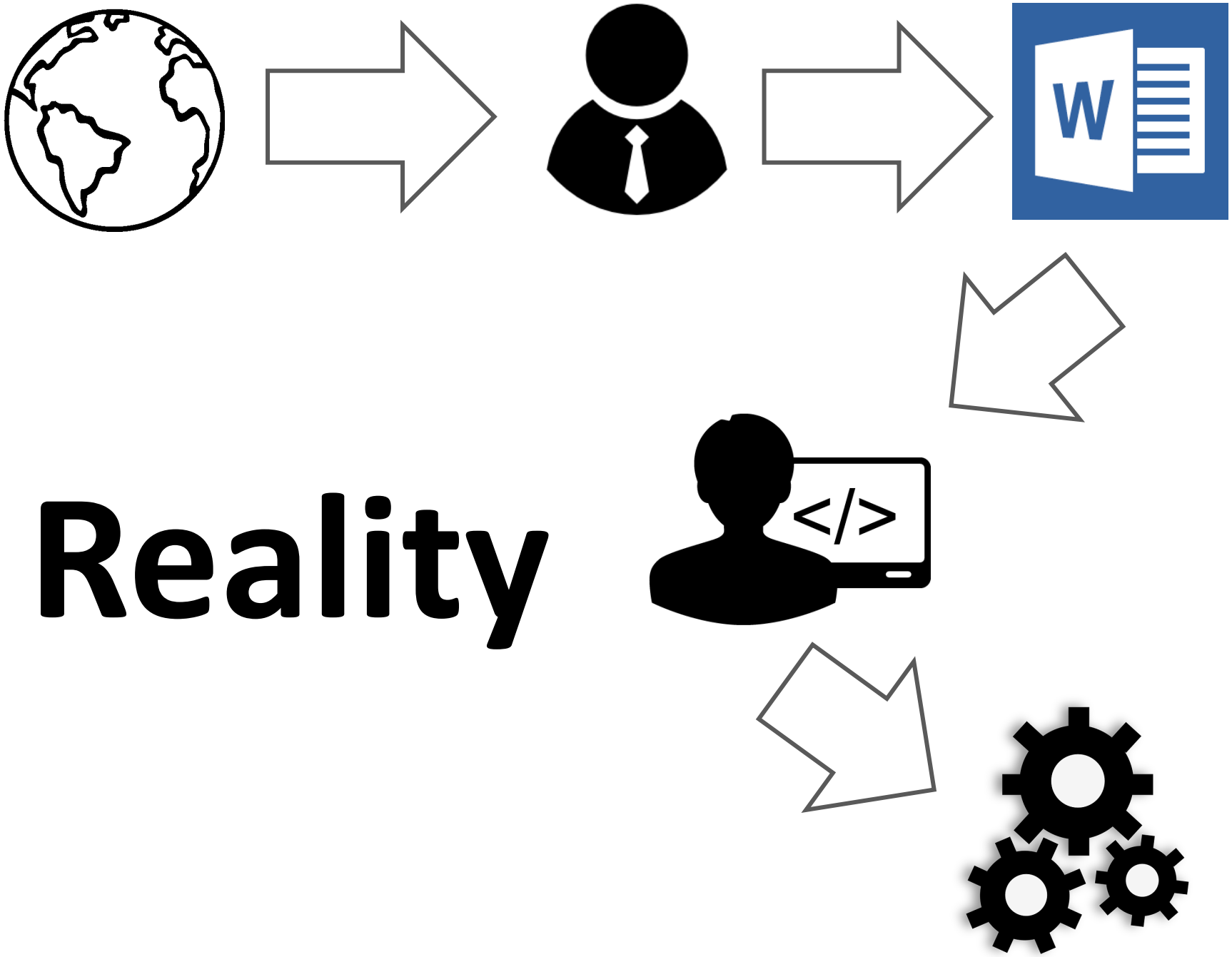
[Business Knowledge]

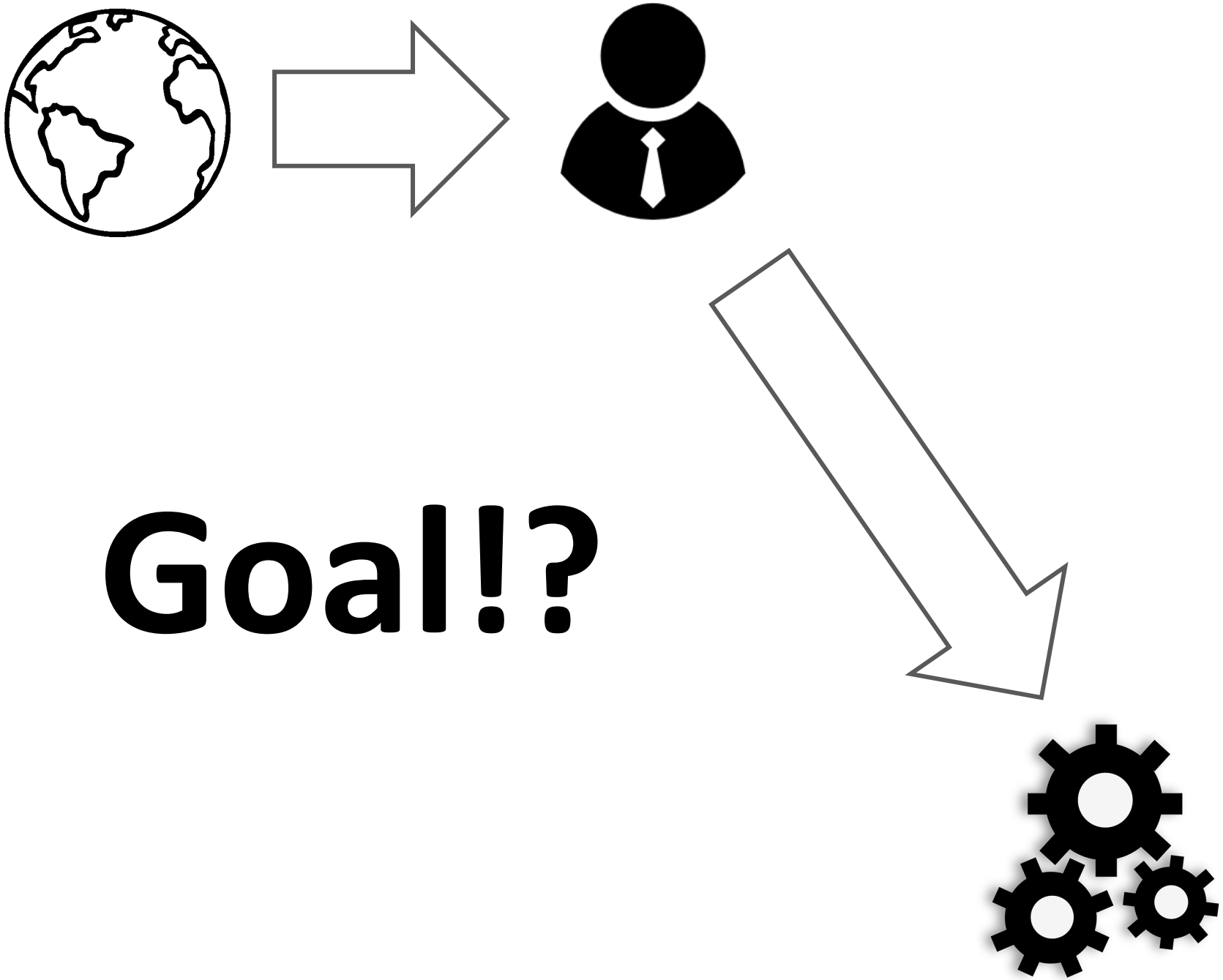
SO HOW DOES IT GET INTO THE SOFTWARE?

Contributed **not** by developers

... but typically implemented

in software


W
Reality
</>


Goal!?

Let Business/Domain people contribute directly!

Give them expressive, productive tools to do so!

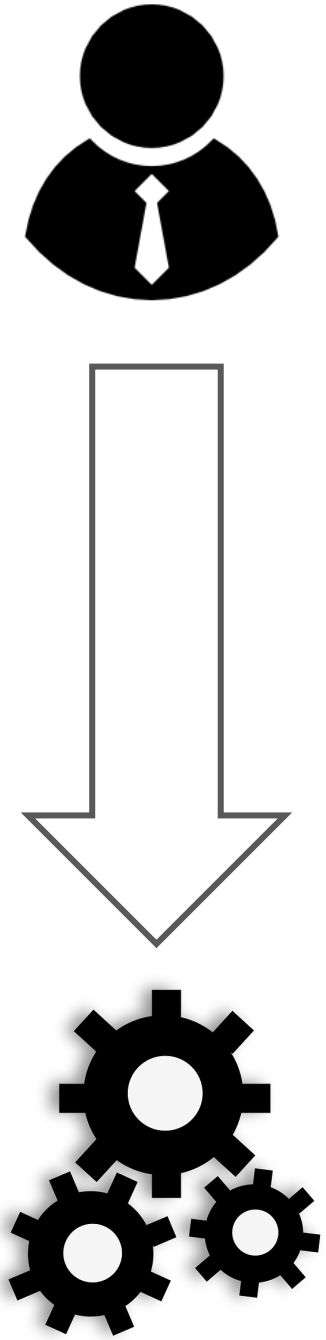
Expressivity for Core Domain Knowledge

User-Friendly Notation
Great Tool/IDE

Testing

Meaningful Analyses

Synthesis of Software



B

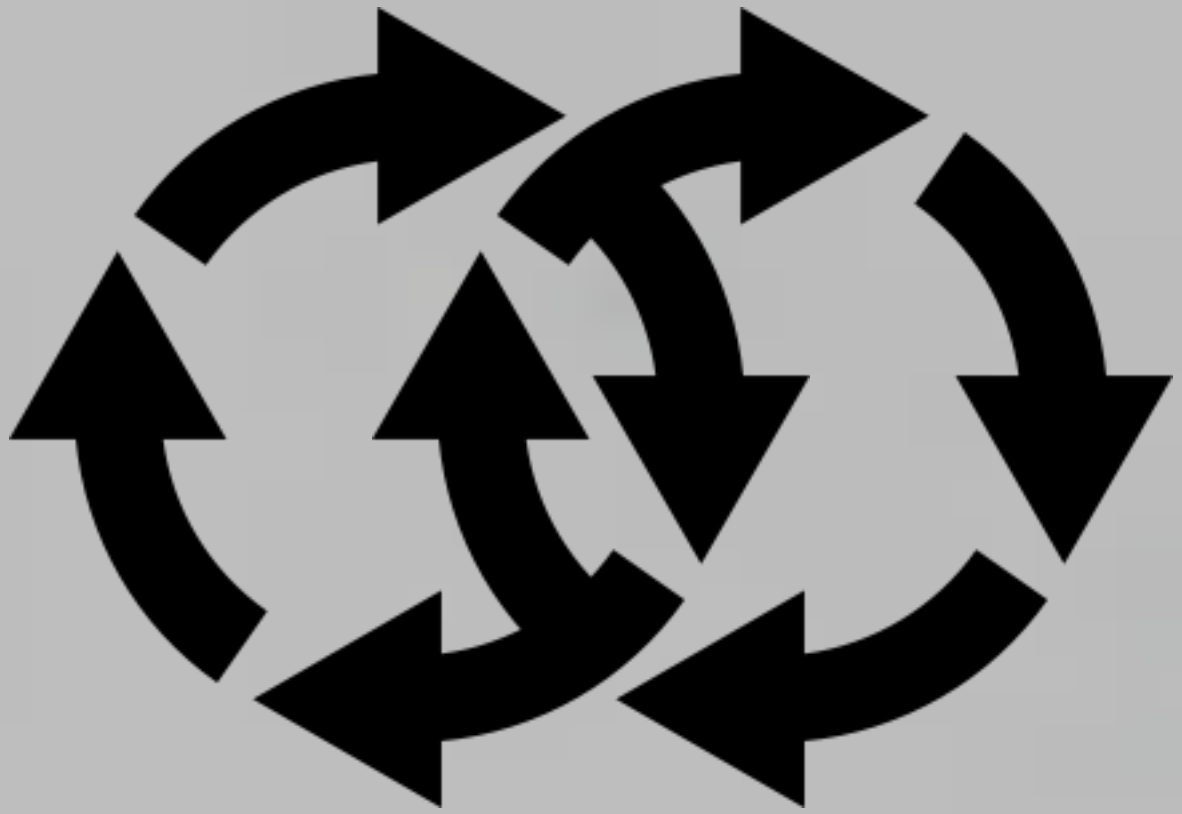
Separation from Technology



Outdated Technology
Obscure Business Logic

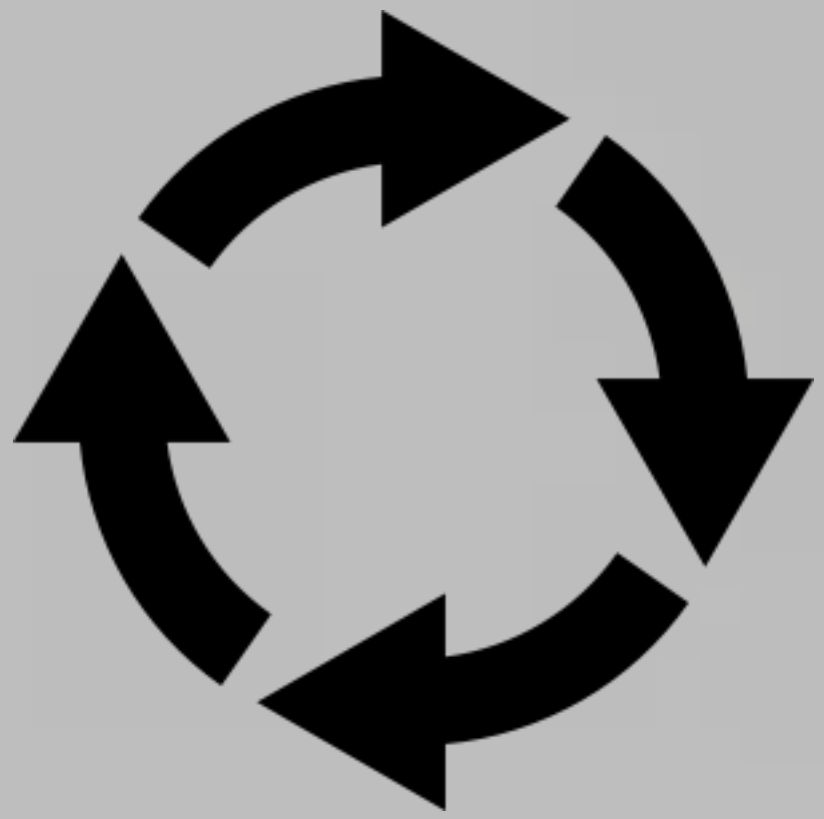
Fachlichkeit „burried“ in implementation code.

Technology & Business Logic now have connected lifecycles.

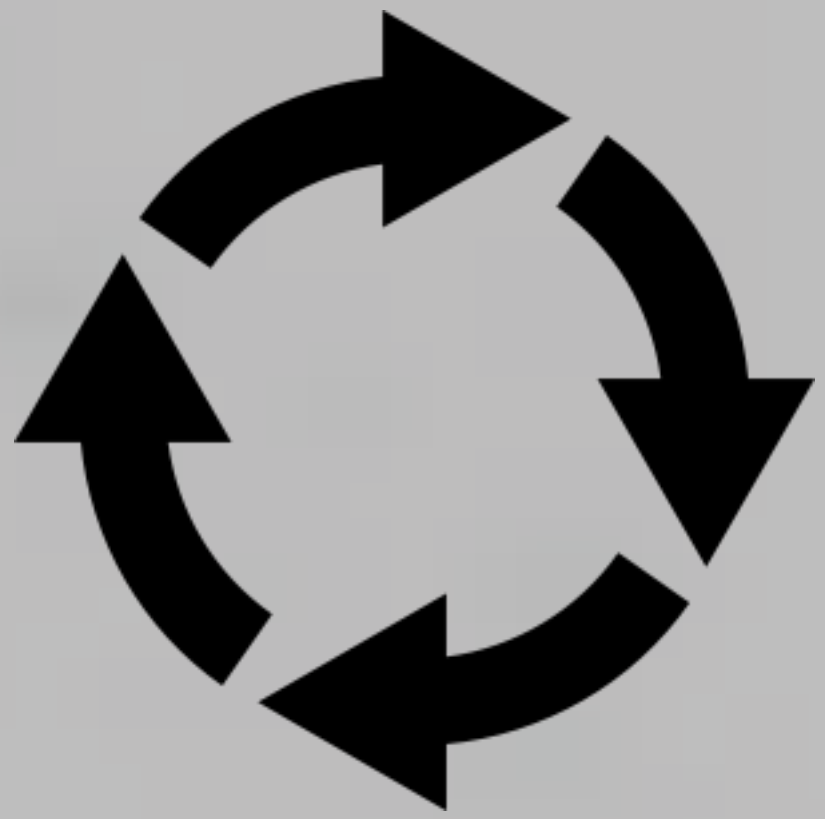


Goal: Separate the Lifecycles

Fachlichkeit



Technology



M

3

Technical

WIKIPEDIA
The Free Encyclopedia

Separation of concerns

From Wikipedia, the free encyclopedia

In computer science, **separation of concerns** (**SoC**) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate. A program that embodies SoC well is called a modular[1] program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of information hiding.[2] Layered designs in information systems are another embodiment of separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).[3]

The value of separation of concerns is simplifying development and maintenance of computer programs. When concerns are well-separated, individual sections can be reused, as well as developed and updated independently. Of special value is the ability to later improve or modify one section of code without having to know the details of other sections, and without having to make corresponding changes to those sections.

Business Logic

Technology

Metamodel for Business Logic

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Metamodel for Business Logic

- Data Structures
- Behavioral Rules
- Expressions
- Validations
- Special Types (eg. temporal)
- ...

Metamodel for Business Logic

- Data Structures
- Behavioral Rules
- Expressions
- Validations
- Special Types (eg. temporal)
- ...

Domains often have a rich language/vocabulary anyway; it just needs to be formalized.
(DDD: Ubiquituous Languge)

Metamodel for Business Logic

Semantics

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Well-defined meaning of this data structure

Metamodel for Business Logic

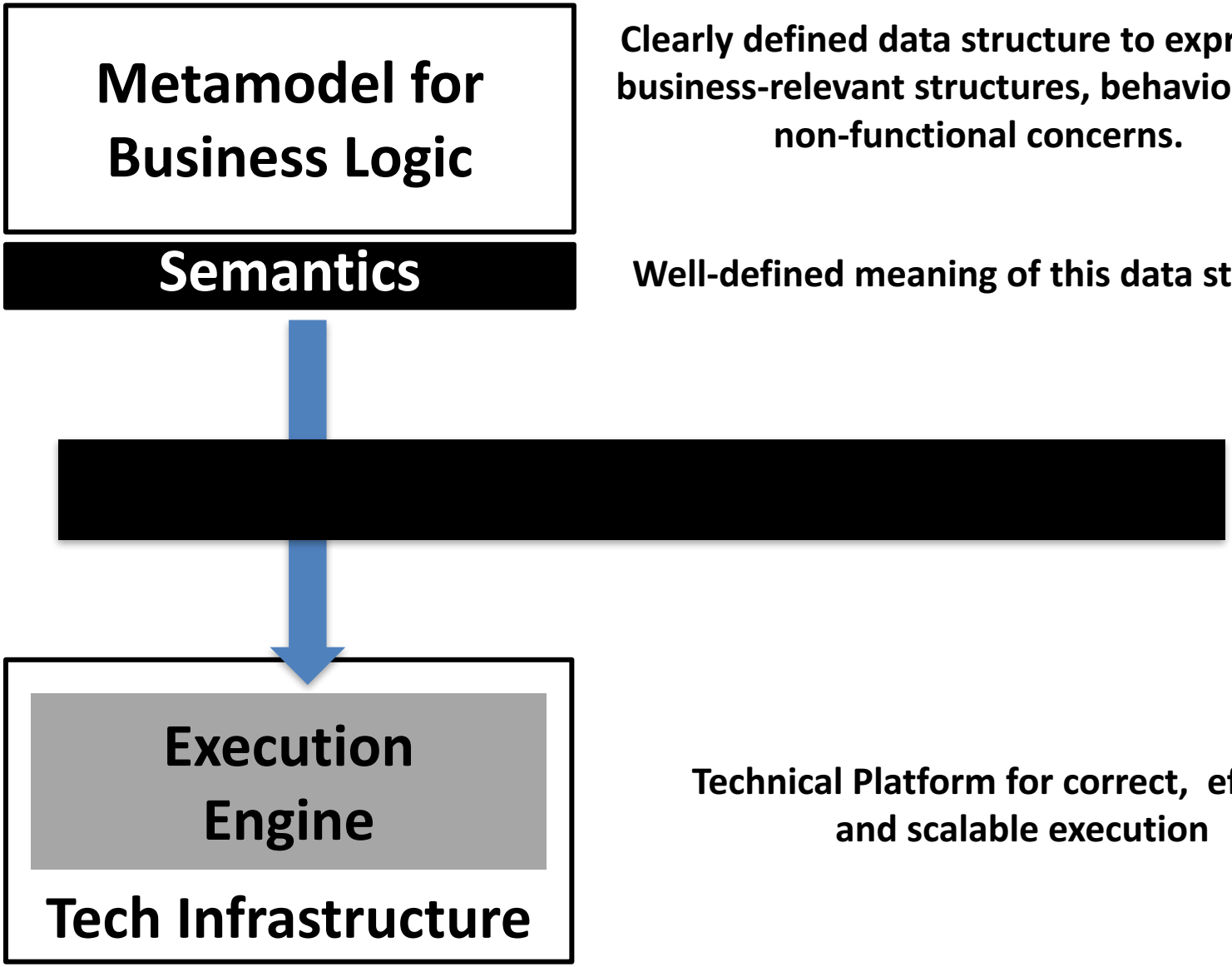
Semantics

Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Well-defined meaning of this data structure

- IDE Support is possible
- Evolution is possible
- Portability is possible

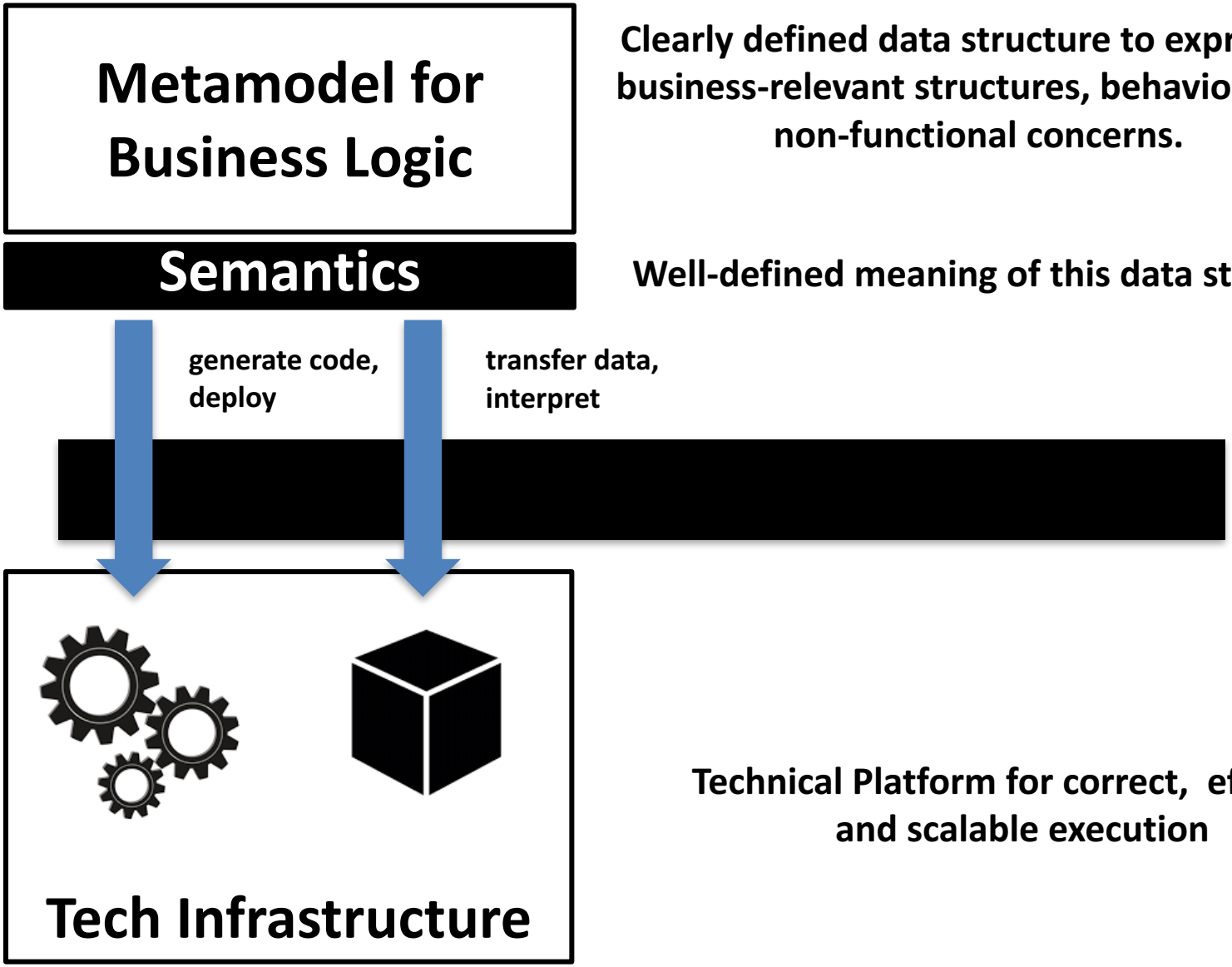
- Type Checking
- Solver-Integration
- Model Checking
- Contracts



Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Well-defined meaning of this data structure

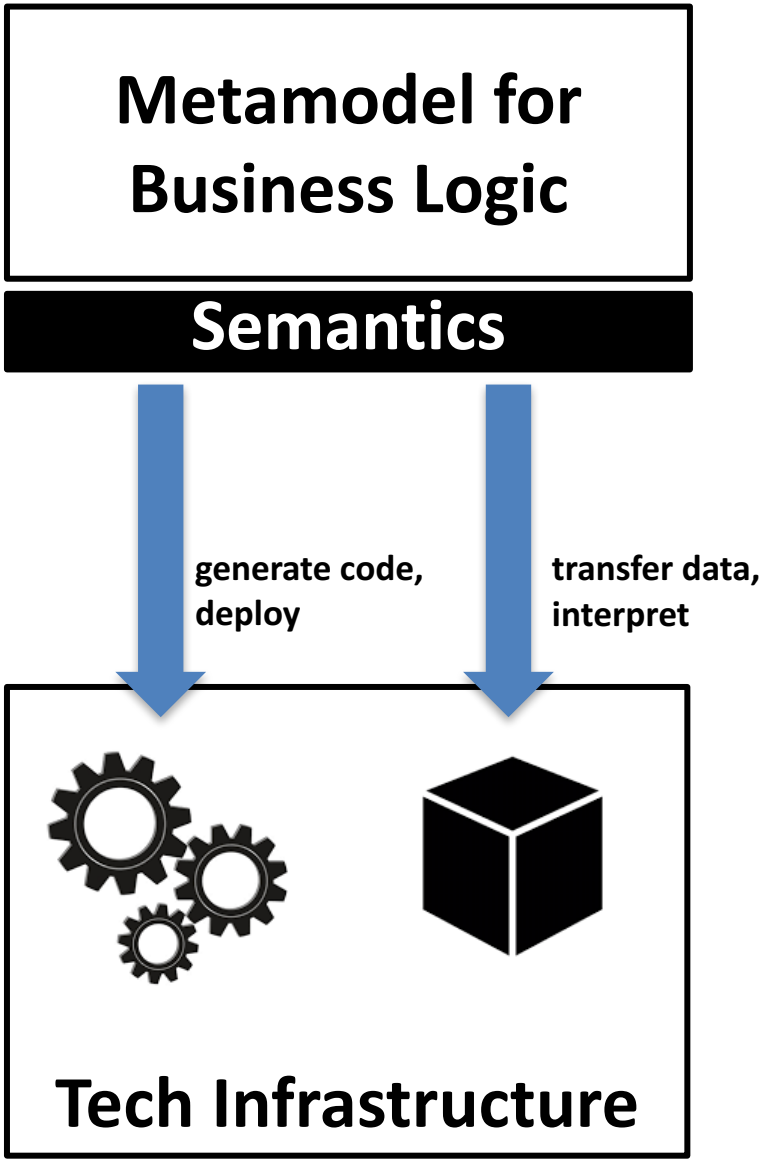
Technical Platform for correct, efficient and scalable execution

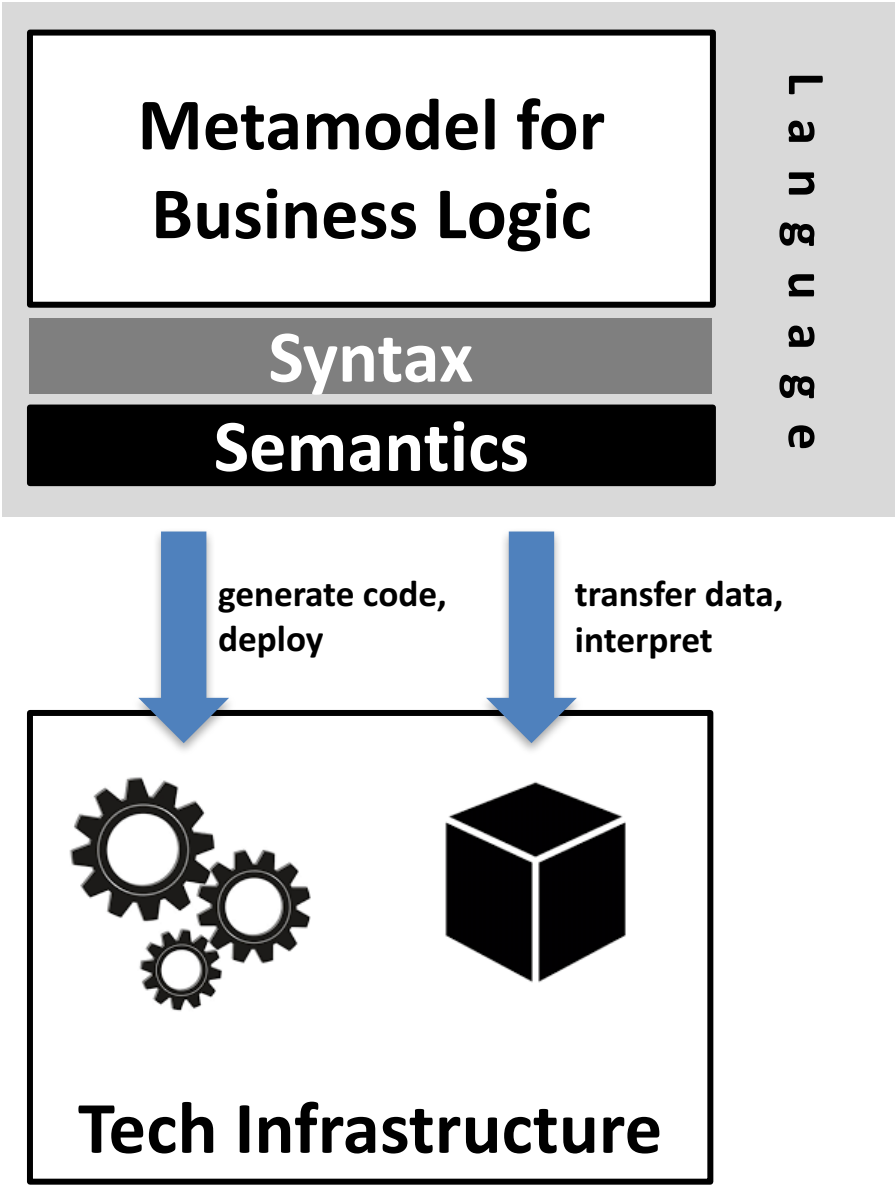


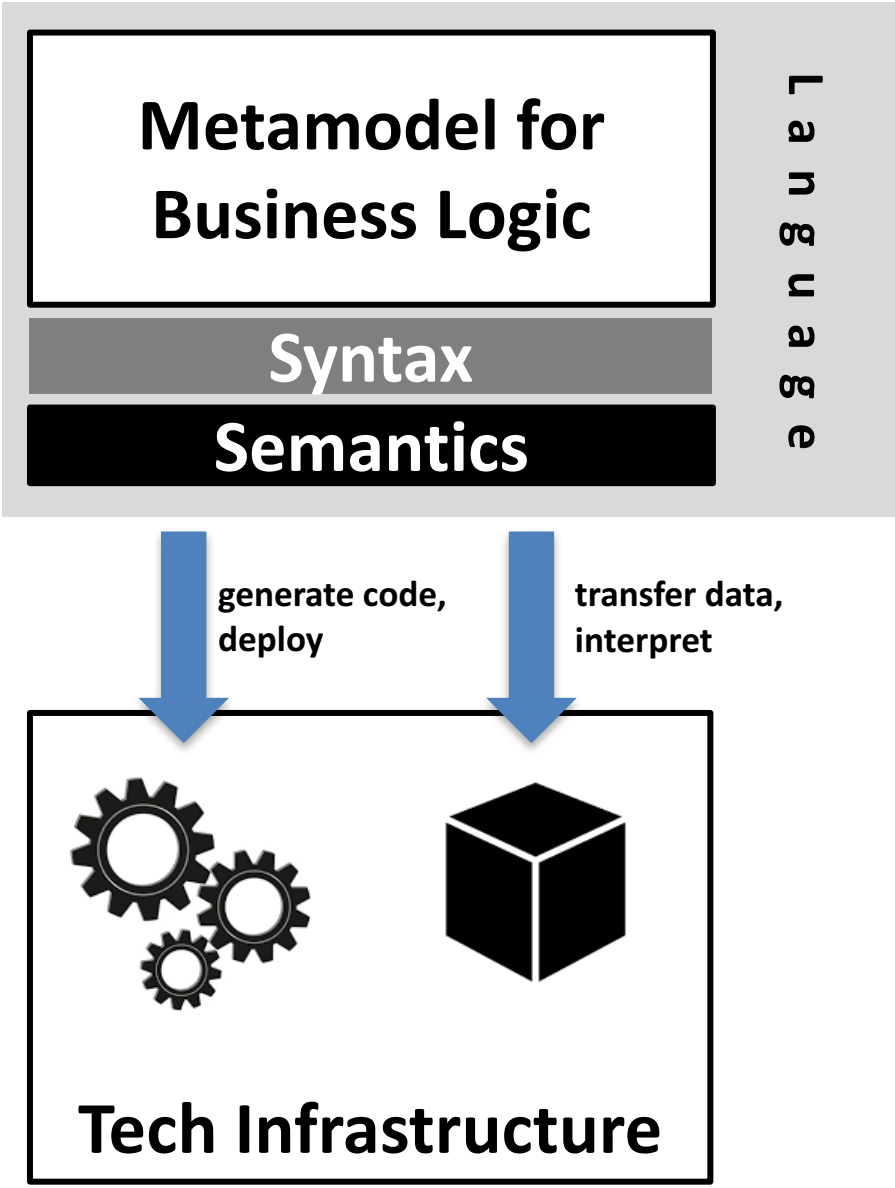
Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Well-defined meaning of this data structure

Technical Platform for correct, efficient and scalable execution

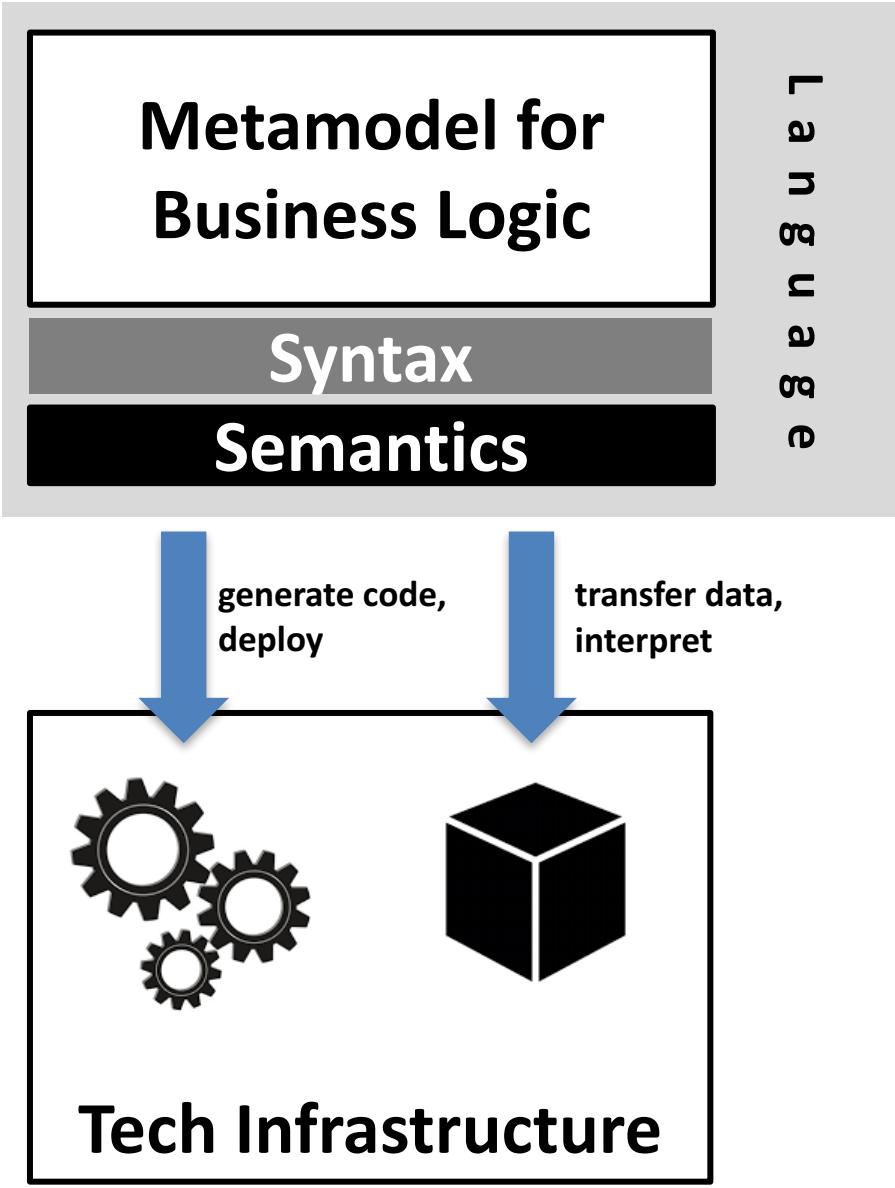

Metamodel for Business Logic
Semantics
generate code, deploy
transfer data, interpret
Tech Infrastructure

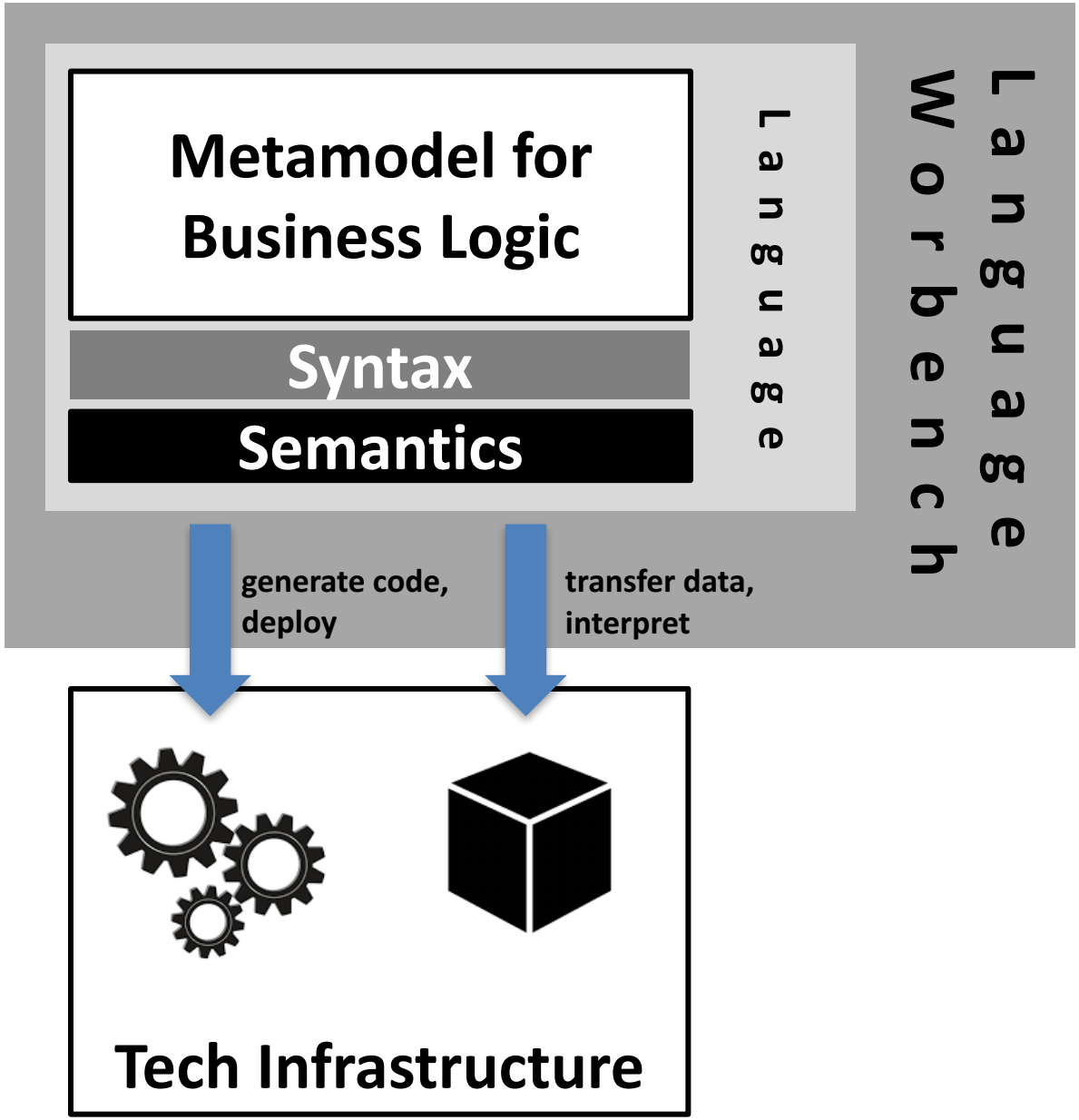

Metamodel for Business Logic
Syntax
Semantics
Language
generate code, deploy
transfer data, interpret
Tech Infrastructure

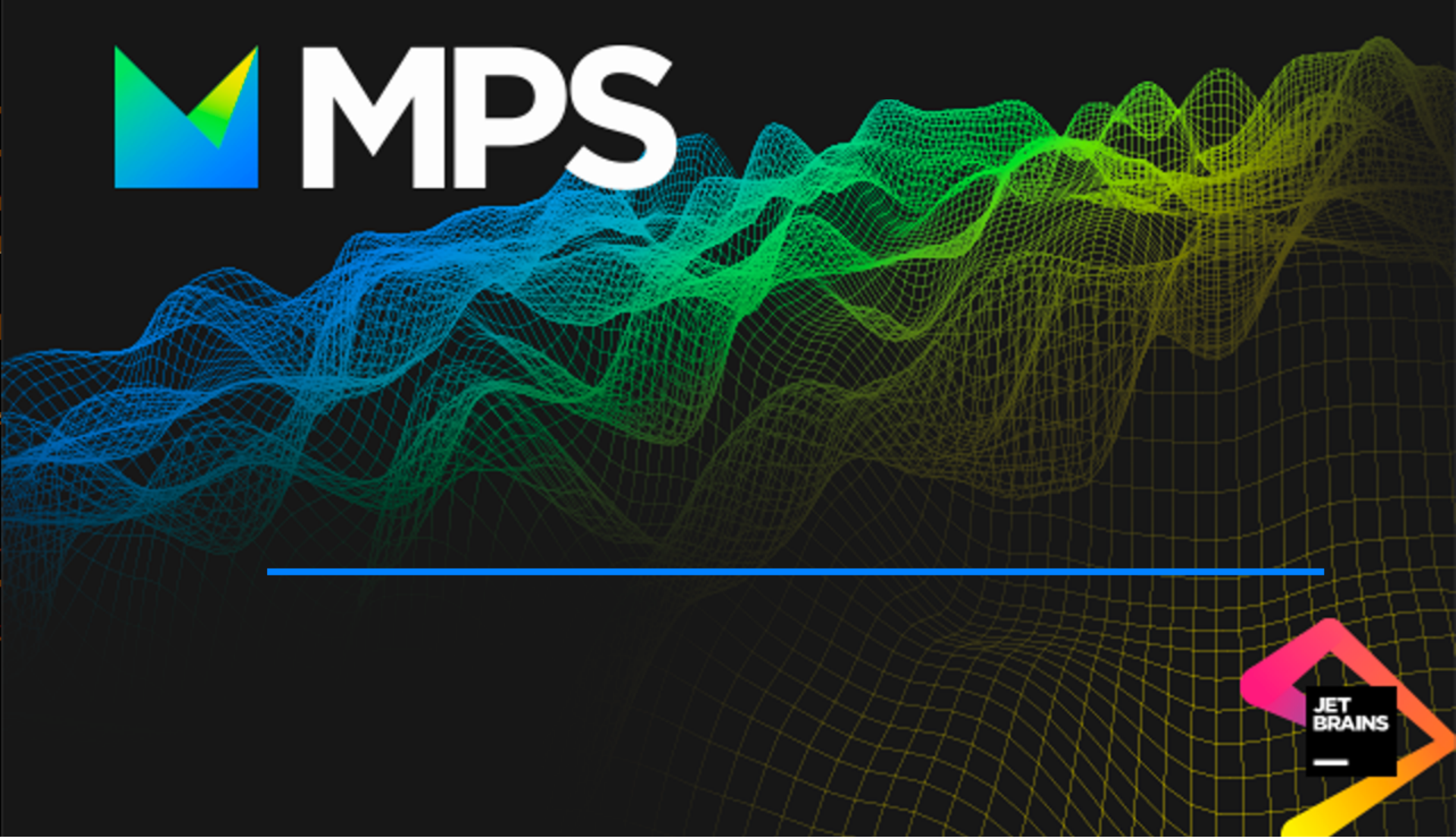


Syntax is **critically important** for

- Productivity
  Communication and Review
  Domain Expert Integration
- Only Buttons and Forms don‘t work!


Metamodel for Business Logic
Syntax
Semantics
Language
generate code, deploy
transfer data, interpret
Tech Infrastructure


Metamodel for Business Logic
Syntax
Semantics
Language
Language Workbench
generate code, deploy
transfer data, interpret
Tech Infrastructure



A Language Workbench –

a tool for defining, composing and using ecosystems of languages.

Other Language Workbenches

{S} spoofax — TU Delft

xtext — itemis/Typefox

Rascal — CWI Amsterdam

The Whole Platform — Solmi/Persiani

ELSEVIER

Computer Languages, Systems & Structures

Volume 44, Part A, December 2015, Pages 24–47

Special issue on the 6th and 7th International Conference on Software Language Engineering (SLE 2013 and SLE 2014)

Evaluating and Comparing Language Workbenches

Existing Results and Benchmarks for the Future

Sebastian Erdweg[d], Tijs van der Storm[a], Markus Völter[e], Laurence Tratt[b], Remi Bosman[f], William R. Cook[c], Albert Gerritsen[f], Angelo Hulshout[g], Steven Kelly[h], Alex Loh[c], Gabriël Konat[1], Pedro J. Molina[j], Martin Palatnik[f], Risto Pohjonen[h], Eugen Schindler[f], Klemens Schindler[f], Riccardo Solmi[1], Vlad Vergu[1], Eelco Visser[1], Kevin van der Vlist[k], Guido Wachsmuth[1], Jimi van der Woning[l]

[a] *CWI, The Netherlands*
[b] *King's College London, UK*
[c] *University of Texas at Austin, US*
[d] *TU Darmstadt, Germany*
[e] *voelter.de, Stuttgart, Germany*
[f] *Sioux, Eindhoven, The Netherlands*
[g] *Delphino Consultancy*
[h] *MetaCase, Jyväskylä, Finland*
[i] *TU Delft, The Netherlands*
[j] *Icinetic, Sevilla, Spain*
[k] *Sogyo, De Bilt, The Netherlands*
[l] *Young Colfield, Amsterdam, The Netherlands*

http://voelter.de/data/pub/LWB-ResultsAndBenchmarks.pdf

4

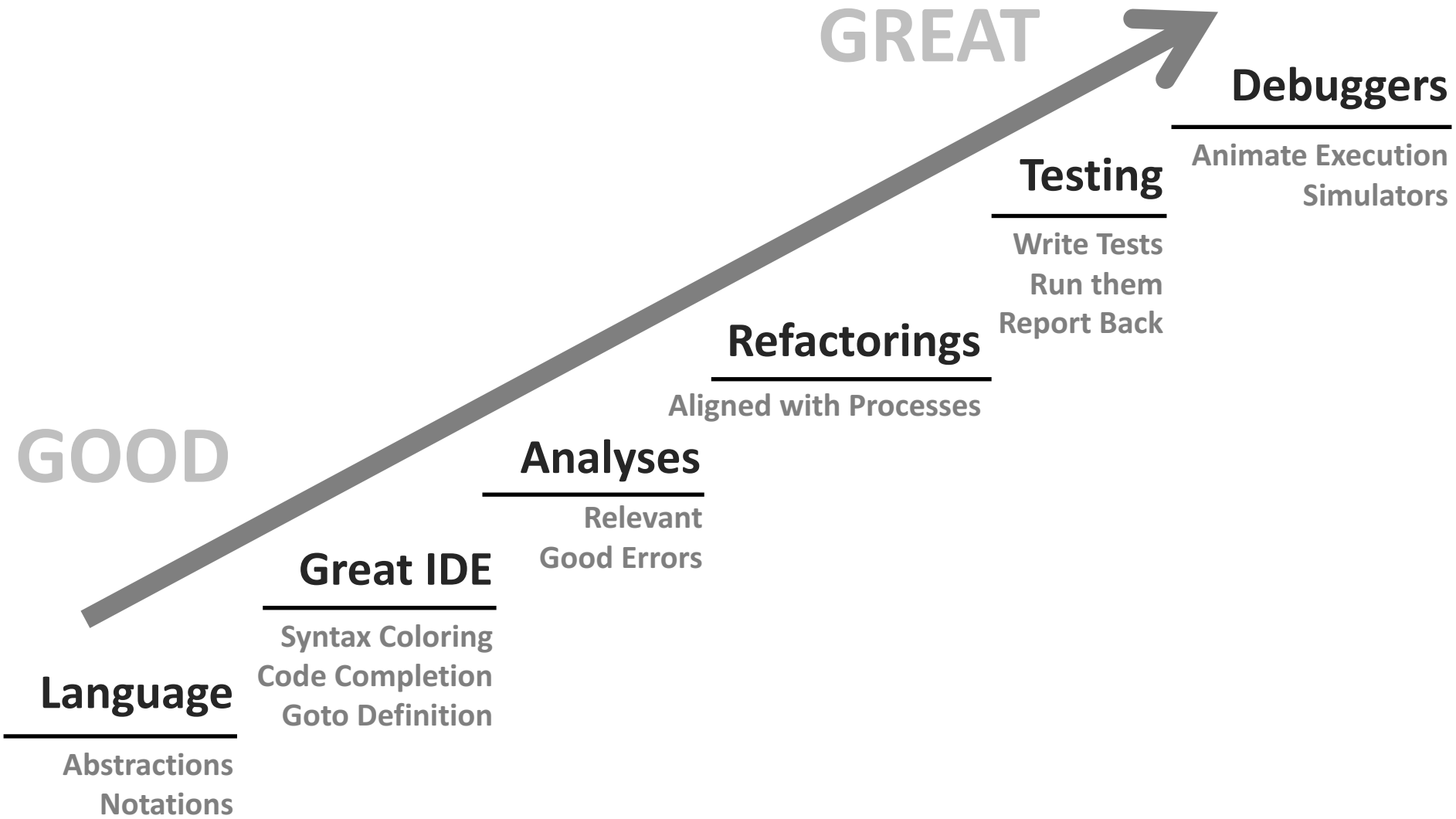
Lessons Learned

B

A Language is not Enough



Language Design::More than Lang



B

Notation
Notation
Notation



Math

```
fun midnight1(a: number, b: number, c: number) = (-b + sqrt(pow2(b) - 4 * a * c)) / (2 * a)

fun midnight2(a: number, b: number, c: number) {
  val bSquared = pow2(b)
  val sqrtPart = sqrt(bSquared - 4 * a * c)
  (-b + sqrtPart) / (2 * a)
}

fun midnight3(a: number, b: number, c: number) {
```

$$\frac{-b + \sqrt{b^2 - 4 * a * c}}{2 * a}$$

```
}
```

Insurance Math

D : Kommutationswerte

Ergebnistyp:	Laufvariable:	Parameter:
number{3}	x	i geschlecht q

$$D_x := l_x * \frac{1}{(1 + i)^x}$$

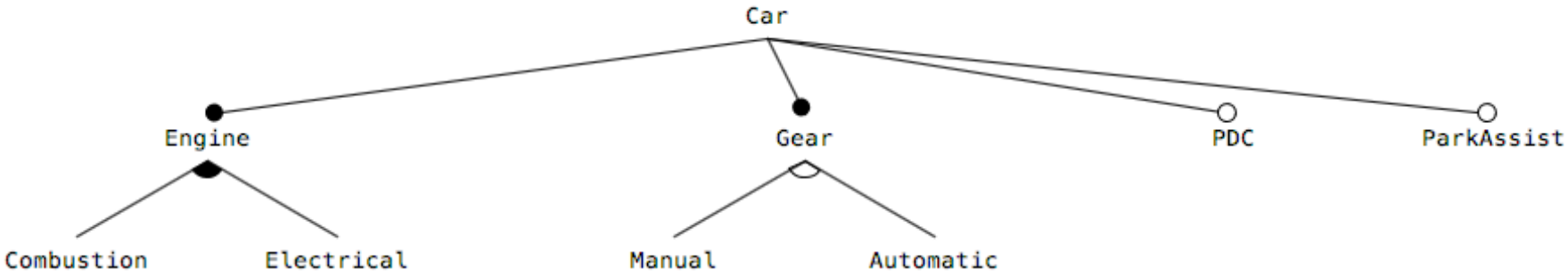
l : Lebende im Jahr x

Ergebnistyp:	Laufvariable:	Parameter:
number{0}	x	geschlecht q

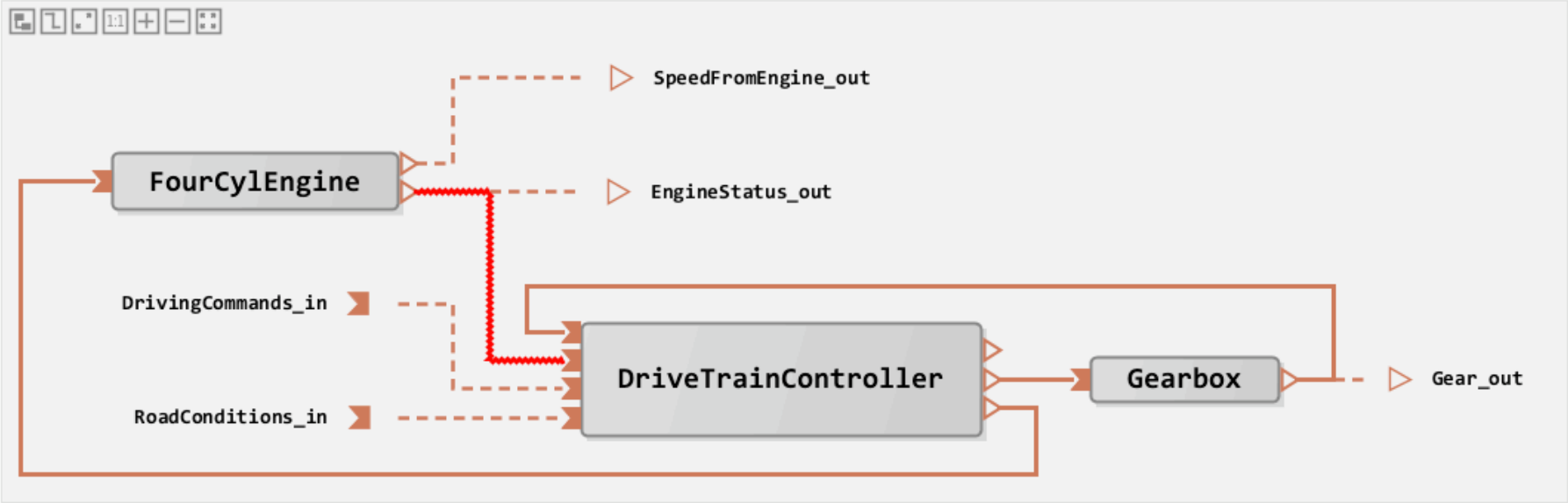
l_0 := startwertLebende

$l_x := l_{x-1} * (1 - q.\mathbf{lookup}(x, geschlecht))$

Feature Models

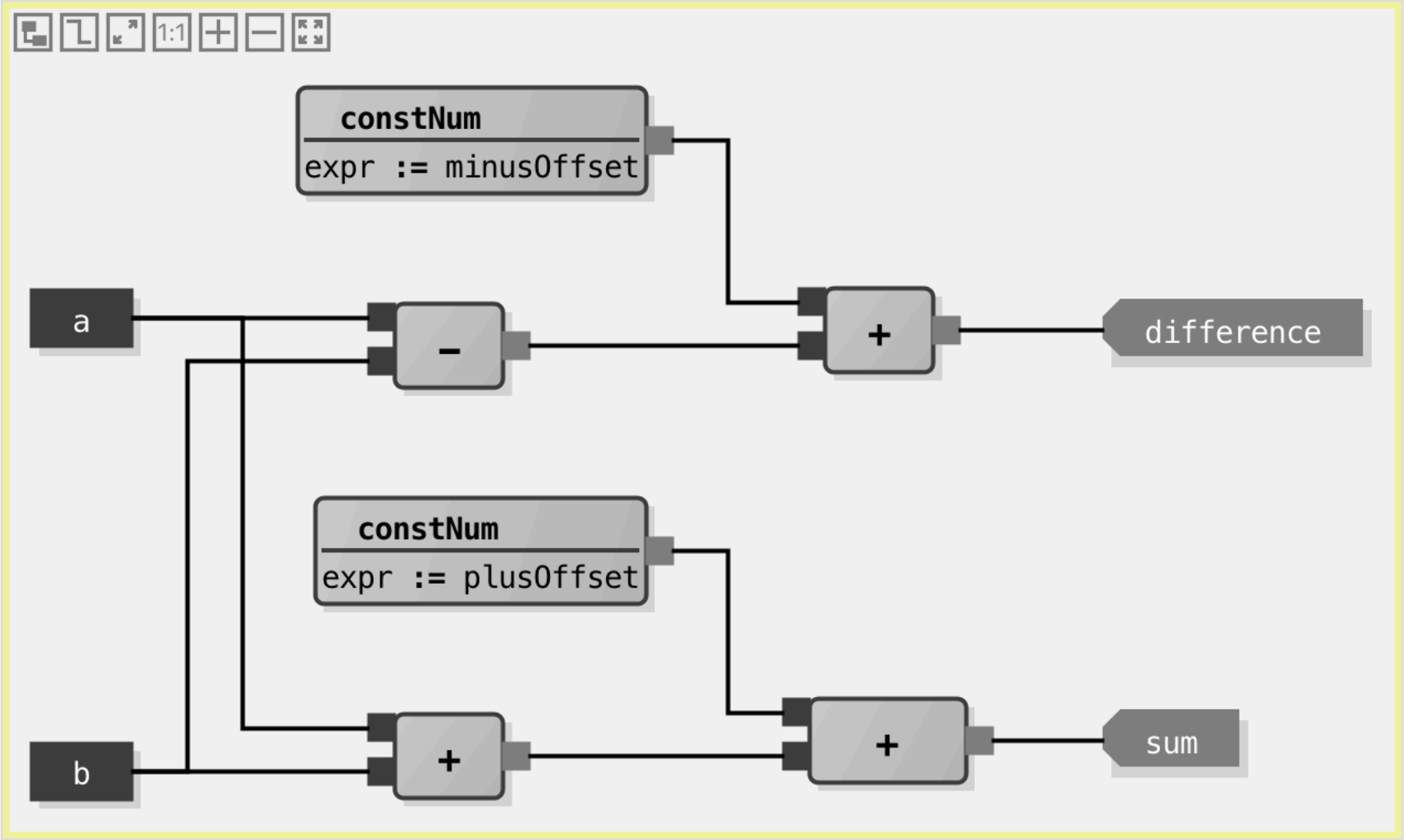


Component Architectures



Dataflow Diagrams

```
composite block[plusOffset: number, minusOffset: number]
  plusMinus_Composite_Offset(a: number, b: number)->(sum, difference)
```



Form Style

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**.geburtsdatum + 12 Jahre ,
datum + 72 Monate – **Anzahl Monate mit uvg)**

Zeitraum für Berechnung: Anfang – Unbegrenzt: {standardzeitraum, standardzeitraum}

---

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 : **dreißigstel**

Bezugsobjekte:
Attribute: bemerkung : **string** wird validiert
antragsdatum : **Datum**

Form Style

Fixed

DS

Unterhaltsvorschuss

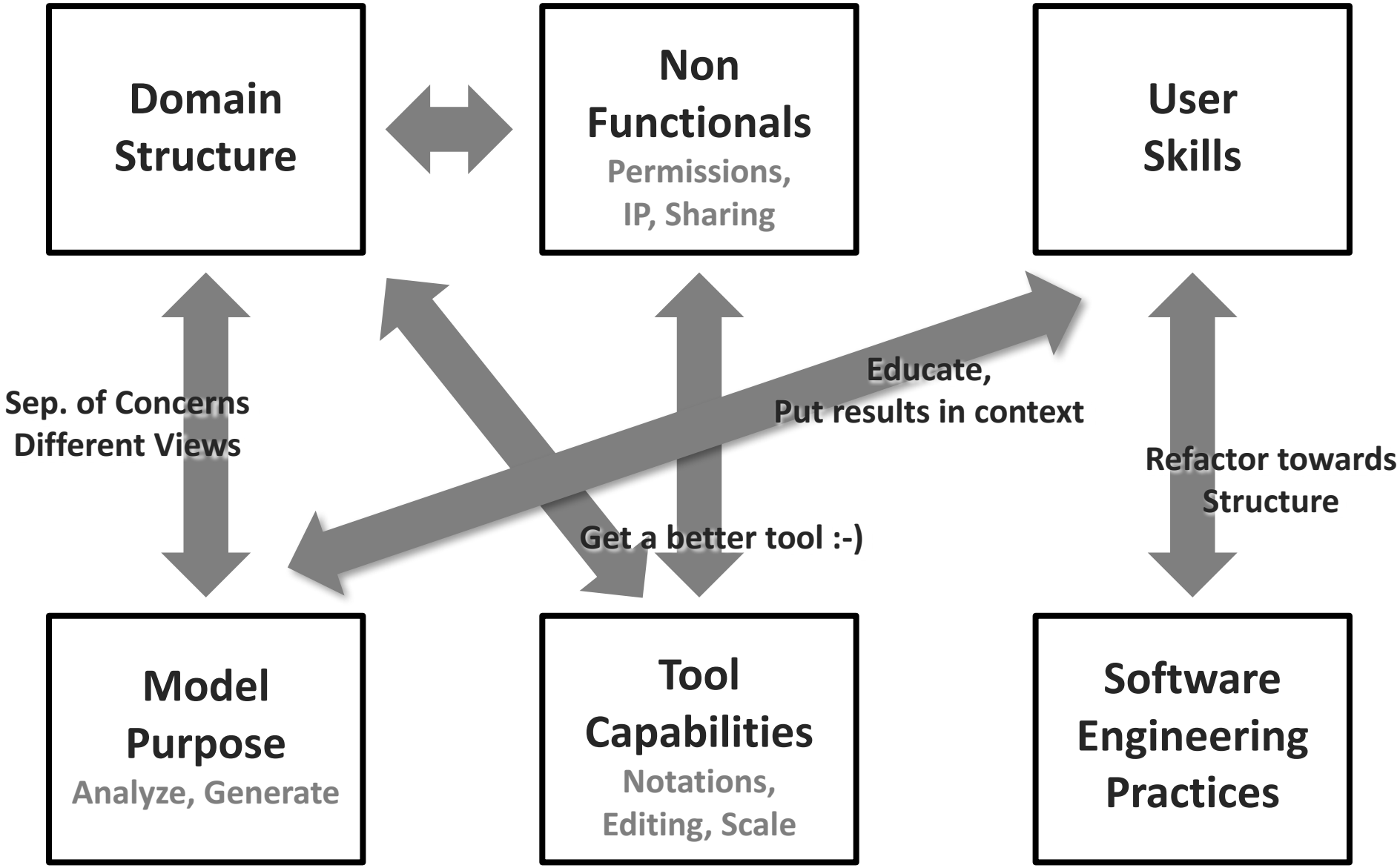
```
Zeitangabe:          laufend
Häufigkeit:          monatlich einmal
Leistungskontext:
Leistungsart:        Leer
Zählart: uvg
Anspruch Beginn: Anfang - Unbegrenzt: junger Mensch.geburtsdatum
Anspruch Ende:   01.01.1800 - 31.12.9999  : min(junger Mensch.geburtsdatum + 12 Jahre ,
                                                 datum + 72 Monate - Anzahl Monate mit uvg)

Zeitraum für Berechnung: Anfang - Unbegrenzt: {standardzeitraum, standardzeitraum}
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 : dreißigstel
Bezugsobjekte:
Attribute: bemerkung     : string wird validiert
           antragsdatum  : Datum
```

M

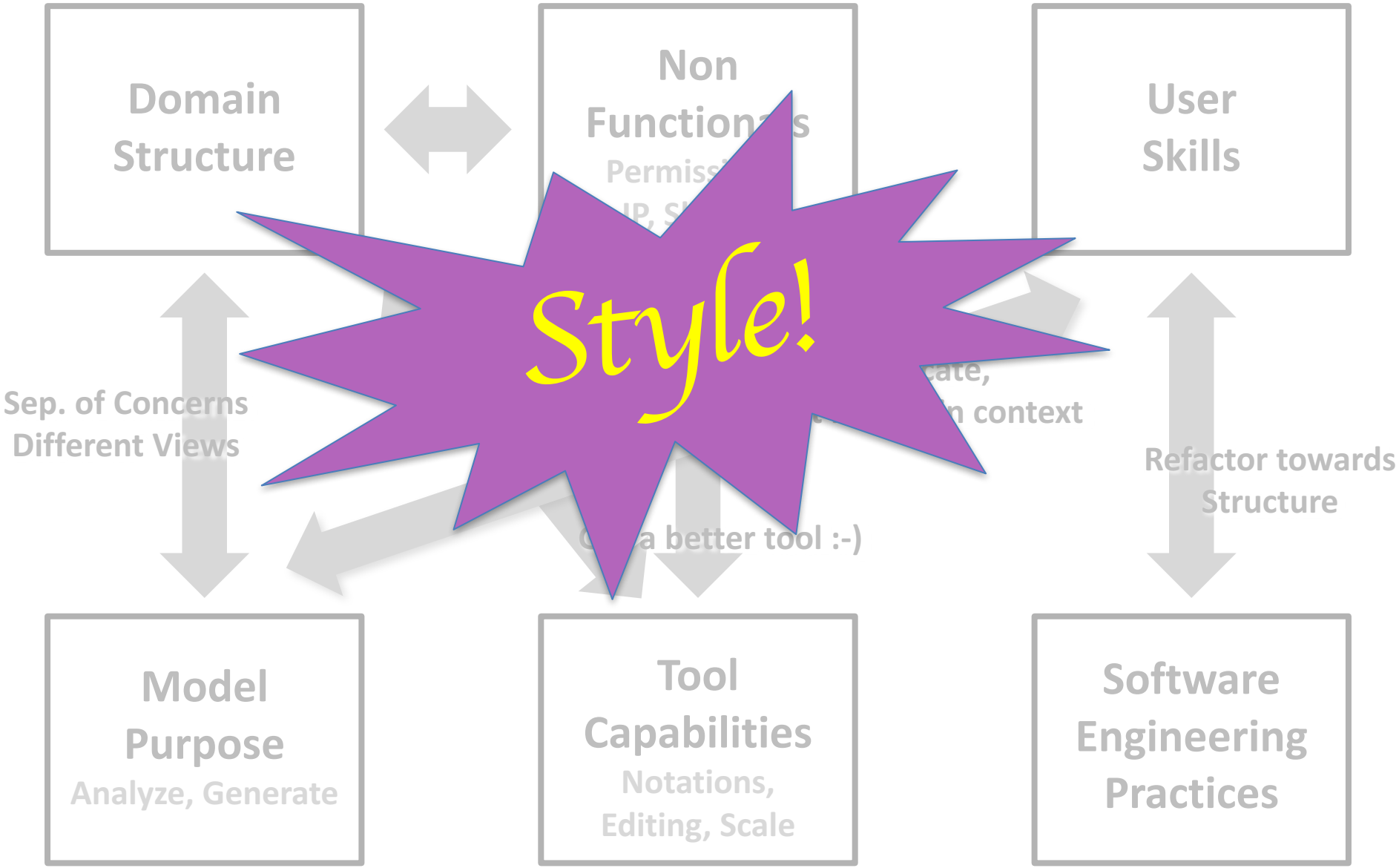
Influences on the Language



Language Design::Influences



Language Design::Influences



M

How to make People precise?



Precision != Programming

Formulas, Rules
Data Structures
Tables
Values

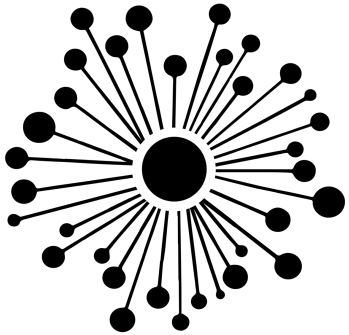
Performance
Scalability
Robustness
Deployment

B

Does this scale?



Does the approach scale?

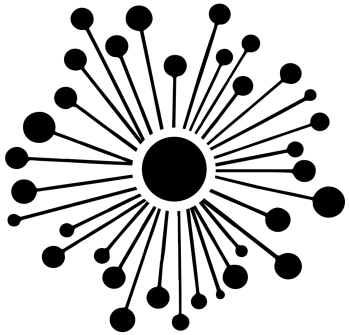


If structure, formalization, and tool support don't scale, then what will??

What are the alternatives?

- Excel?
- Wikis?
- Prose Documents?

Do the tools scale?



In terms of overall system size?

Yes, the system has to be broken down into models of manageable size, as usual. This requires some thought.

In terms of team size?

Yes, since we rely on established version control systems (git) to deal with groupware aspects; and yes, diff/merge works as expected.

In terms of language complexity?

Yes, in particular, since you can modularize the language definitions.

Can I find the people to do this?

Yes, but it is a significant change, so:

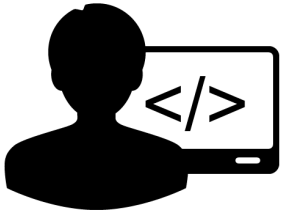
- it may be a significant education/training effort.
- a few people might not get it
- a few people may not want to do it.

B

This is a threat!

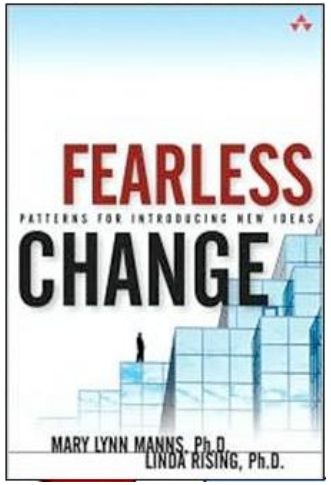


Precision and Formality	**-> New Skills**
Different Processes	**-> Role Change**
Higher Efficiency	**-> Job Loss**



Automation	**-> Job Loss**
Focus on Engineering	**-> Role Change**
Empower Business Ppl	**-> Less Importance**

Some people are afraid of this.
Take them seriously.



A change of

that must be managed!

M

Is this the next legacy system?



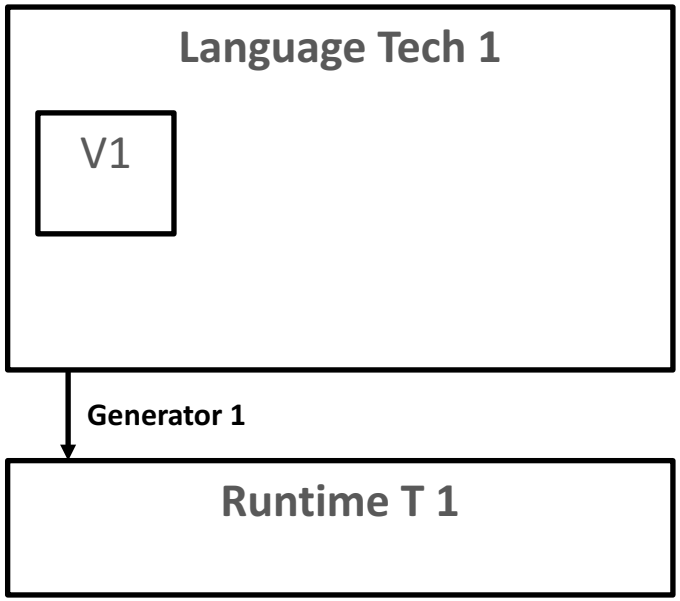
“Today‘s software is tomorrow‘s legacy system.”

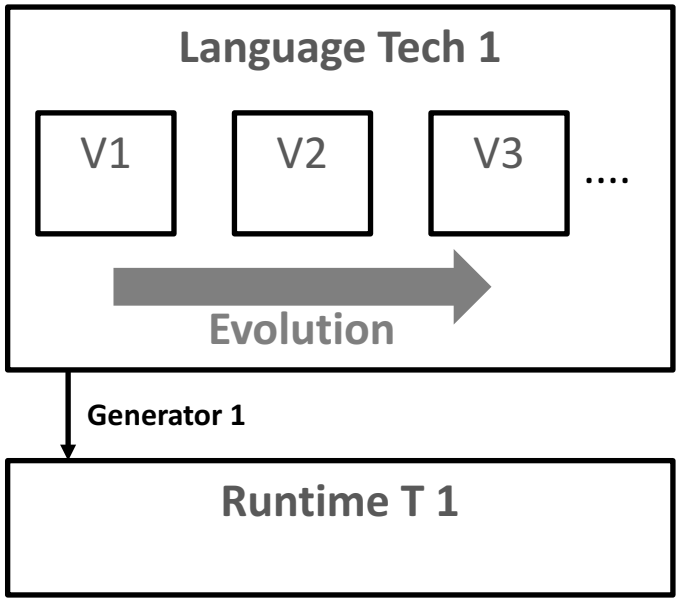
Or is it?

Today‘s software is tomorrow‘s **legacy system.**

Business change is hard
Technology change is hard

Separation of Concerns
Keep BL free of technology
Make it „portable“

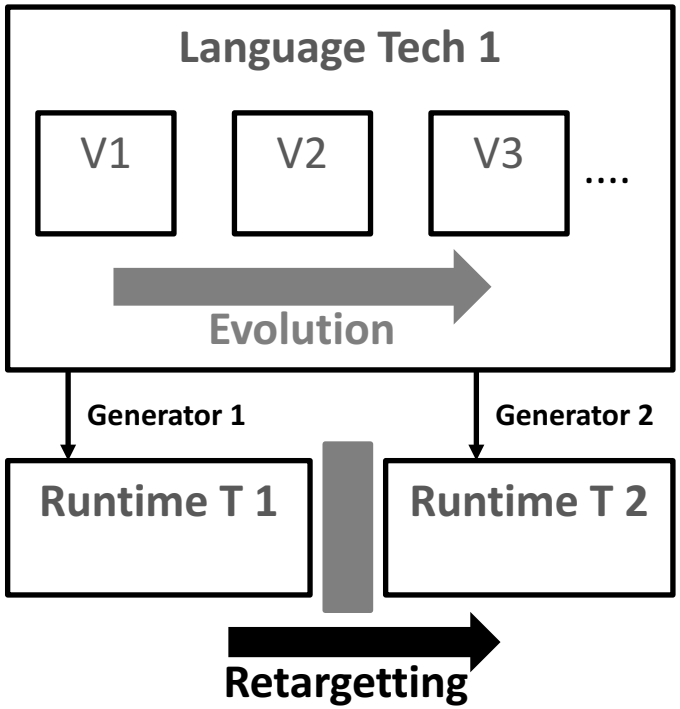

Language Tech 1
V1
Generator 1
Runtime T 1



Existing models become incompatble with new language

⇒ **Language Versions**

Migration Scripts

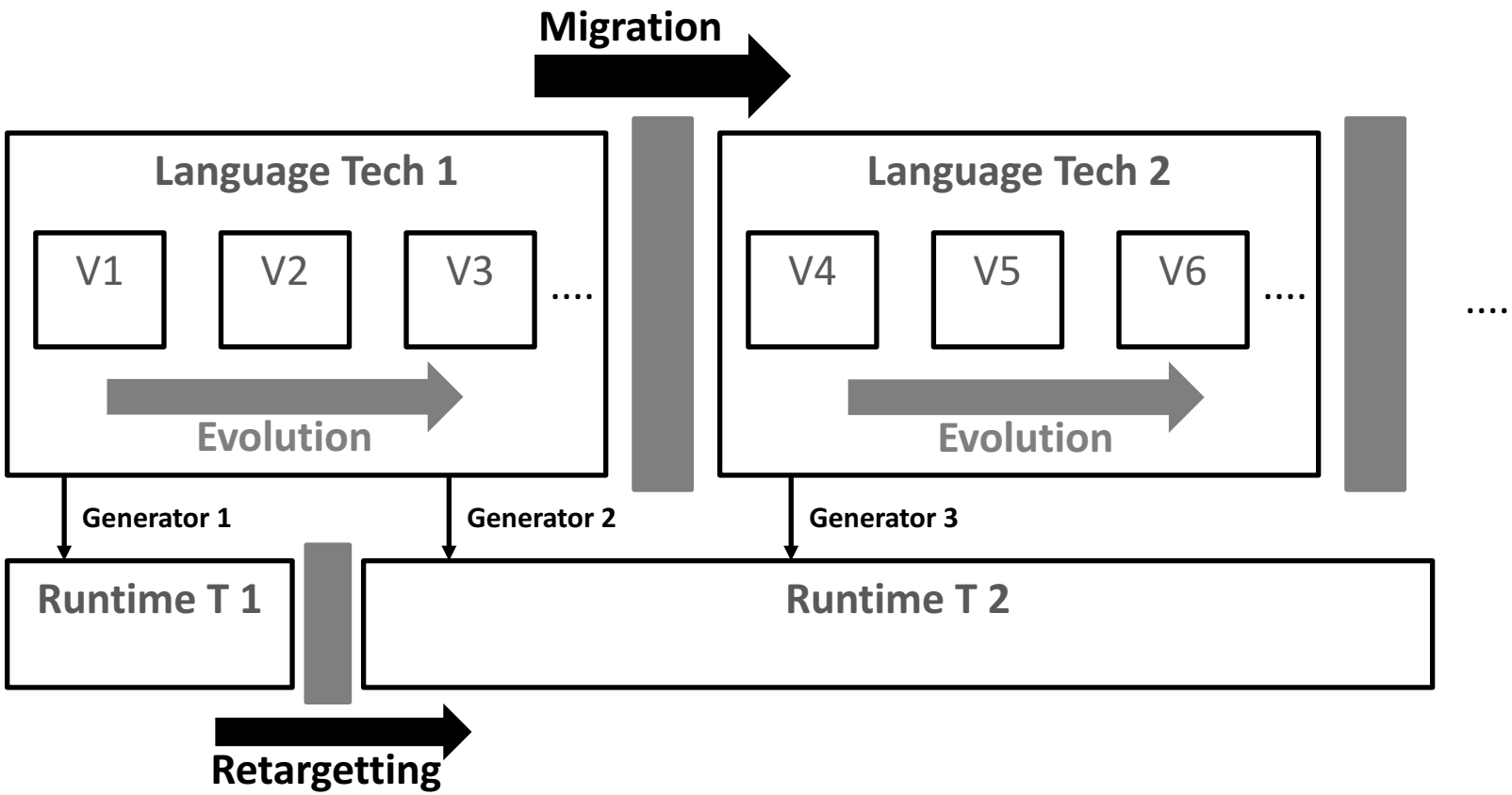


Runtime Tech outdated, uncool or slow

⇒ **Keep Lang Technology**

Keep Models

Build new Generator



Language Tech outdated, uncool

⇒ **Build new Tool**

Migrate Data Simple, because it well-defined domain semantics and free from „technology stuff“

“Today‘s software is tomorrow‘s legacy system.”

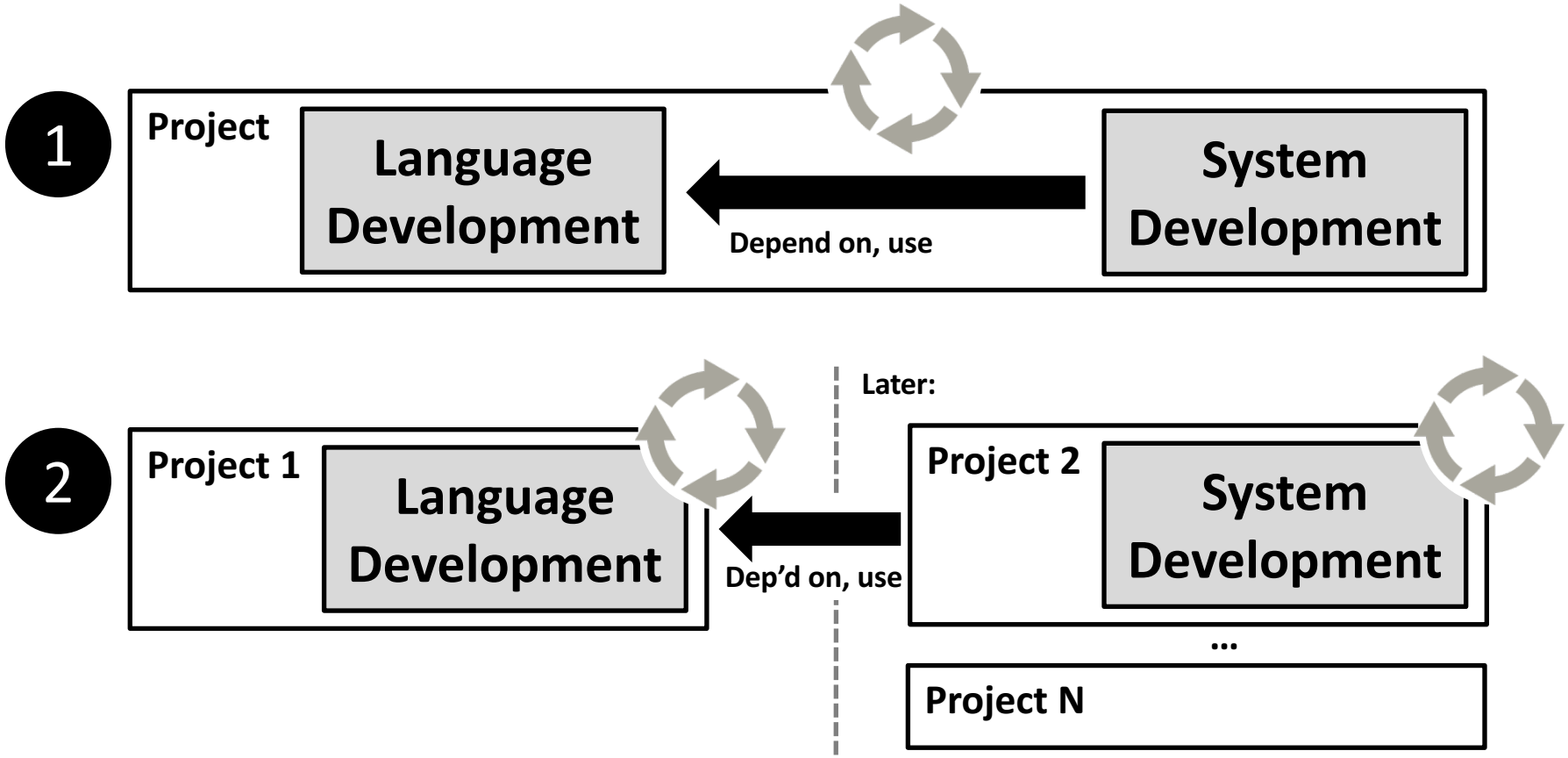
No, it is not.

M

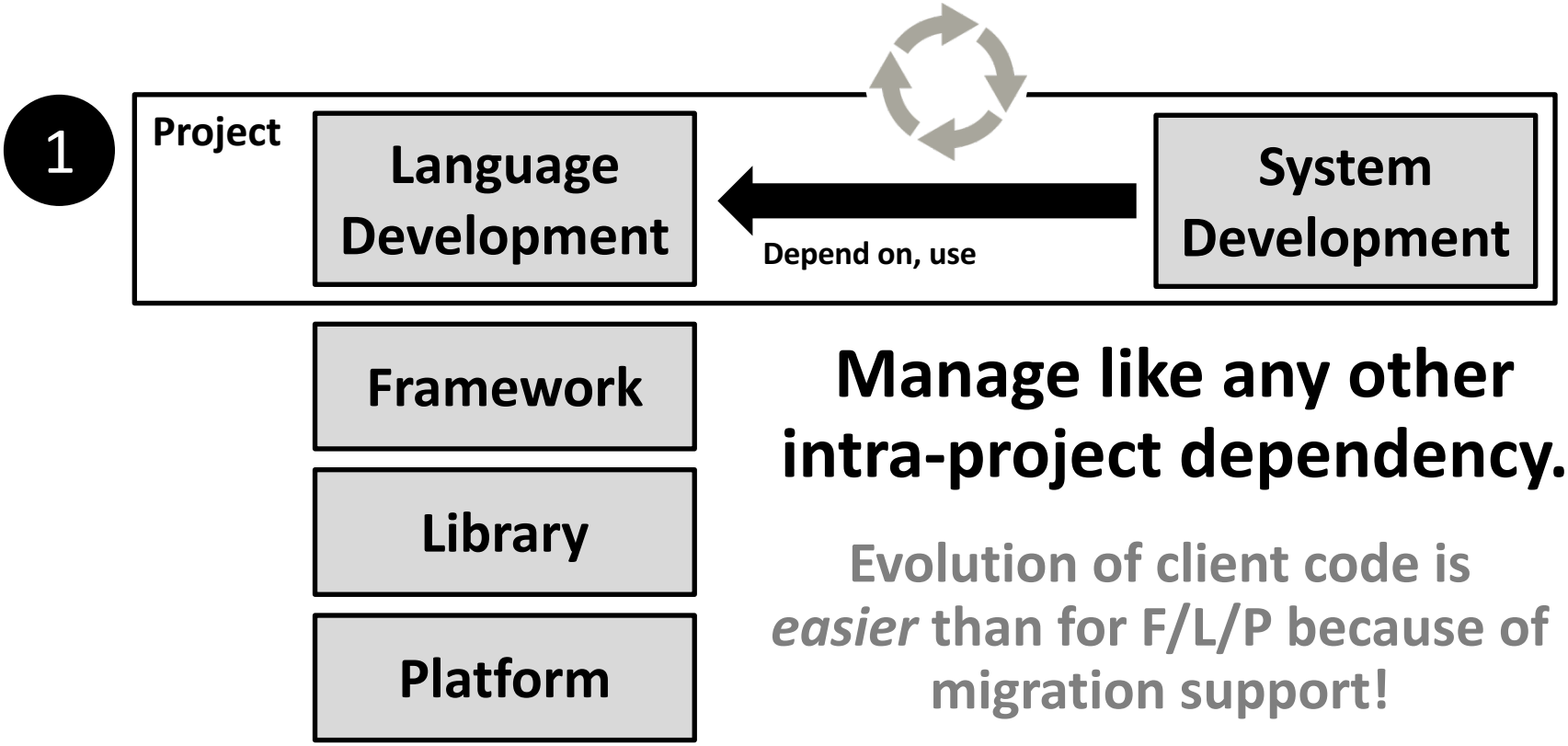
In conflict with Agile?



„MD* and Agile is in Conflict.“



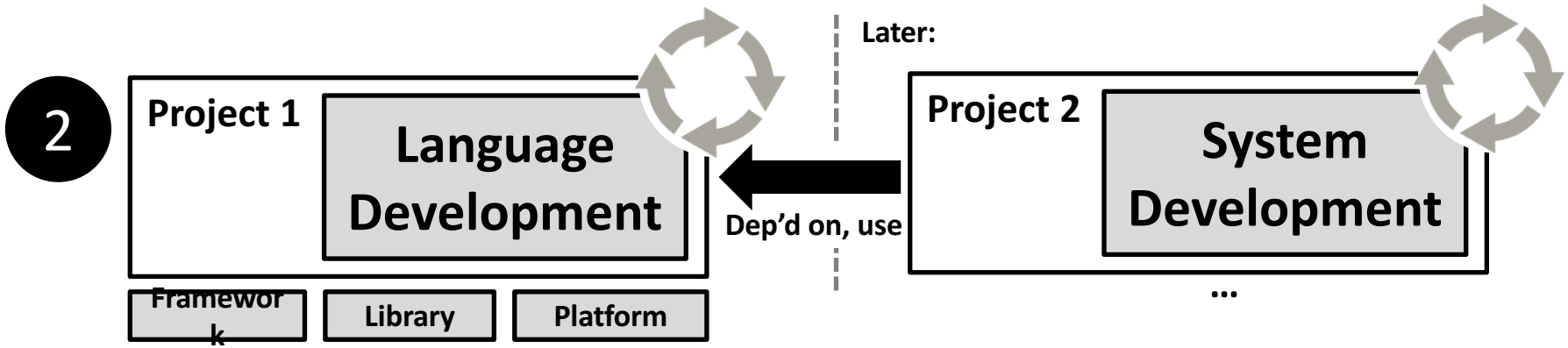
„MD* and Agile is in Conflict.“



Manage like any other intra-project dependency.

Evolution of client code is *easier* than for F/L/P because of migration support!

„MD* and Agile is in Conflict.“



Manage like any other
3rd party depencency: Development Roadmap
Issue Tracker
Release Notes
...

„MD* and Agile is in Conflict.“

WTF?

3

Models and DSLs are an Enabler for Agility:

Integration of Domain Experts
„Living“ Requirements
Decoupled Fachlichkeit & Technik

„MD* and Agile is in Conflict.“

4

Leading LWBs are so productive, you can literally sit with the domain experts and interactively prototype languages (and then clean up later)

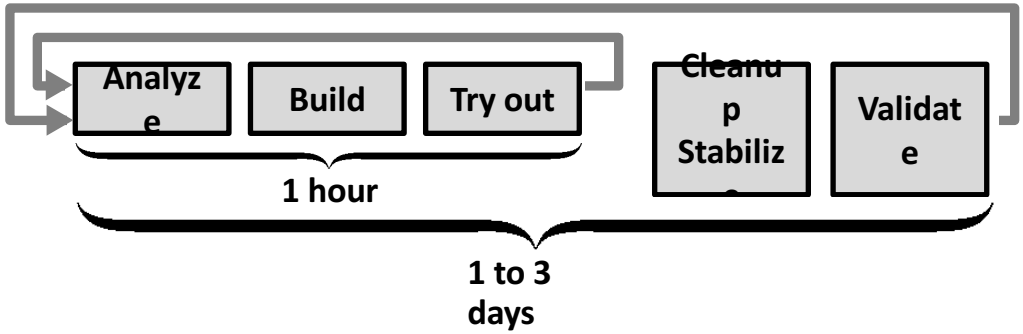
I've looked at the implementation of the language in MPS, but I didn't find much. Is this all there is? Where's the magic?

[Customer]

„MD* and Agile is in Conflict.“

4

Leading LWBs are so productive, you can literally sit with the domain experts and interactively prototype languages (and then clean up later)



I've looked at the implementation of the language in MPS, but I didn't find much. Is this all there is? Where's the magic?

[Customer]

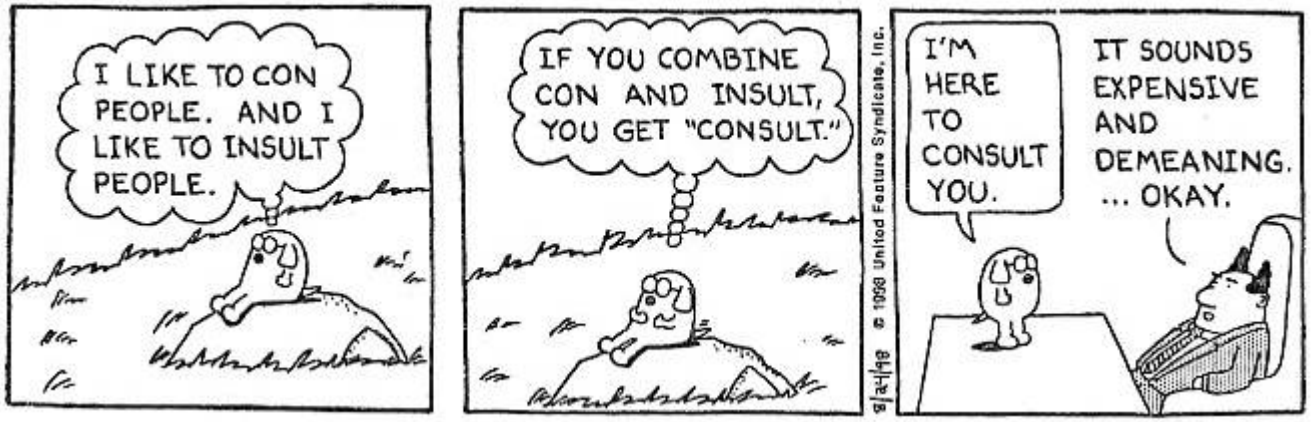
B

Skills?



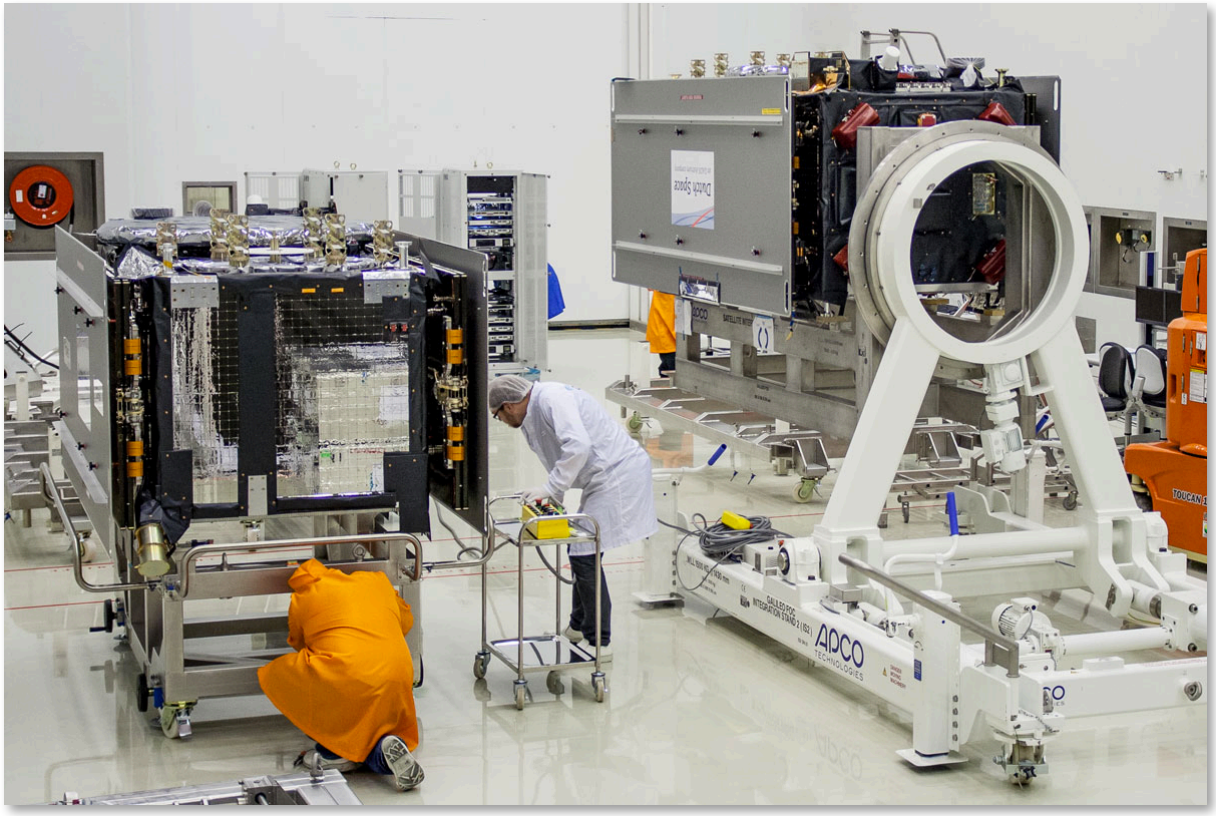
“Organizations do not have the necessary skills. True. But... AI Big Data Agile REST”

So built it. Evolve. Hire. Buy.



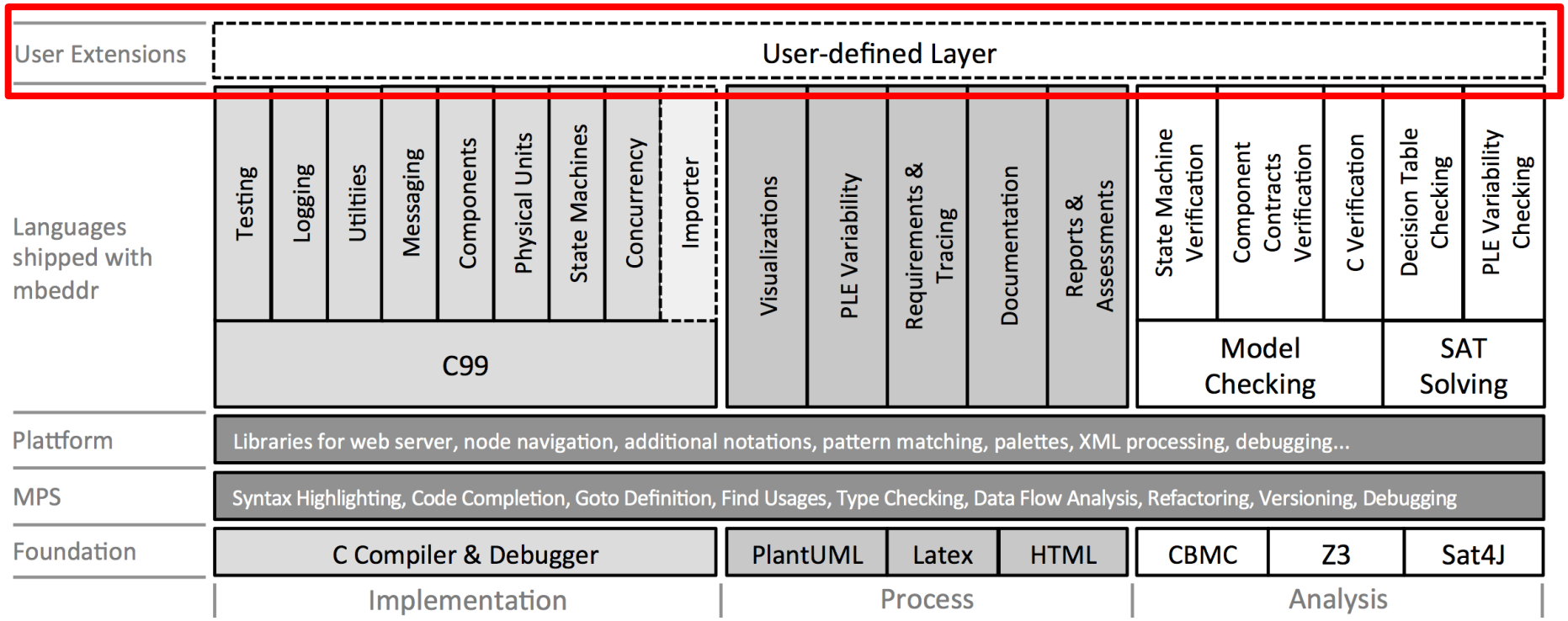
M

Rockets ????



OHB SYSTEM

Further domain-specific extensions to C. Developed by end-user lang engineer.



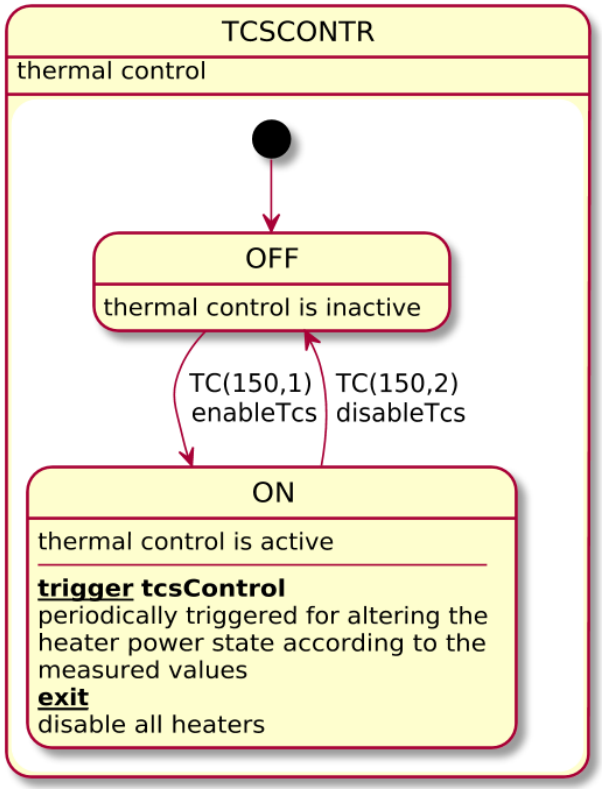
Infrastructure Specifics in C

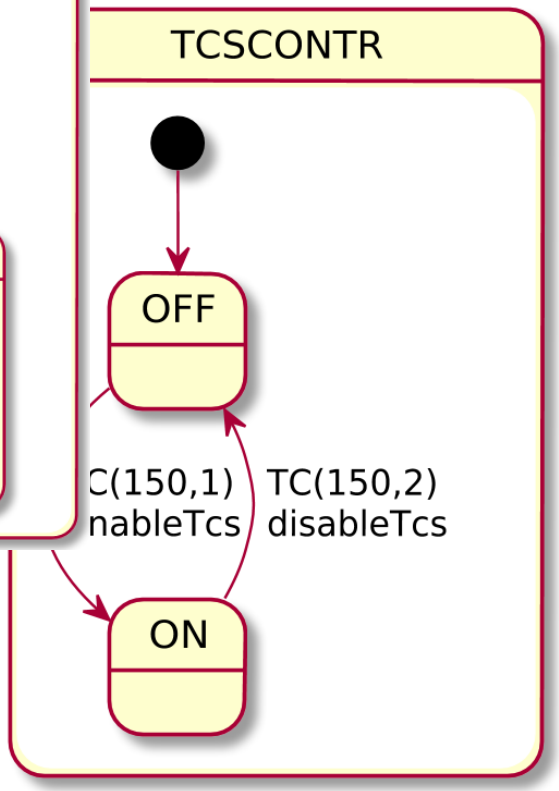
```
#constant TEMP_BUFFER_SIZE = 10;

TACQA = Instance of TemperatureAcquisition with mnemonic tail A and the Numeric Id 350
  [SENSOR = SensorA]
TACQB = Instance of TemperatureAcquisition with mnemonic tail B and the Numeric Id 351
  [SENSOR = SensorB]

Component TemperatureAcquisition with Base Mnemonic: TACQ
  Short Description: acquisition of temperatures

  Description: The components acquires the measurements of an assigned set of thermistors
{
  Attribute (hidden) int32/rawTemp/[TEMP_BUFFER_SIZE] MEASURED = <no init> ; // measured raw values

  Attribute (hidden) uint32 ACQCNT = 0 ; // index for filling data acquisition buffer

  Attribute (readwrite) tempSensor SENSOR (Id= 2 ) = <no init> ; // selected sensor for this component

  ModeChart TCSACQ (Id= 3 ) initial = OFF {
    Trigger tcsAcquisition
    Mode OFF {
      << ... >>
    }
    Mode ON {
      entry { ACQCNT = 0; }
      on trigger tcsAcquisition {

        // measure a value
        MEASURED[ACQCNT] = readTemperature(SENSOR);
        ACQCNT = (ACQCNT + 1) % TEMP_BUFFER_SIZE;

        // calculate average of the @top(TEMP_BUFFER_SIZE) latest measurements and convert to °C
                                    TEMP_BUFFER_SIZE - 1
        PUS150.AVTEMP = convert[        Σ          (MEASURED[idx]) / TEMP_BUFFER_SIZE -> °C];
                                    idx = 0
      }
    }
  }

  Activity startAcquisition with Nummeric Id 1
  ... { TCSACQ.setMode(ON); }

  Activity stopAcquisition with Nummeric Id 2
  ... { TCSACQ.setMode(OFF); }
} Component TemperatureAcquisition
```

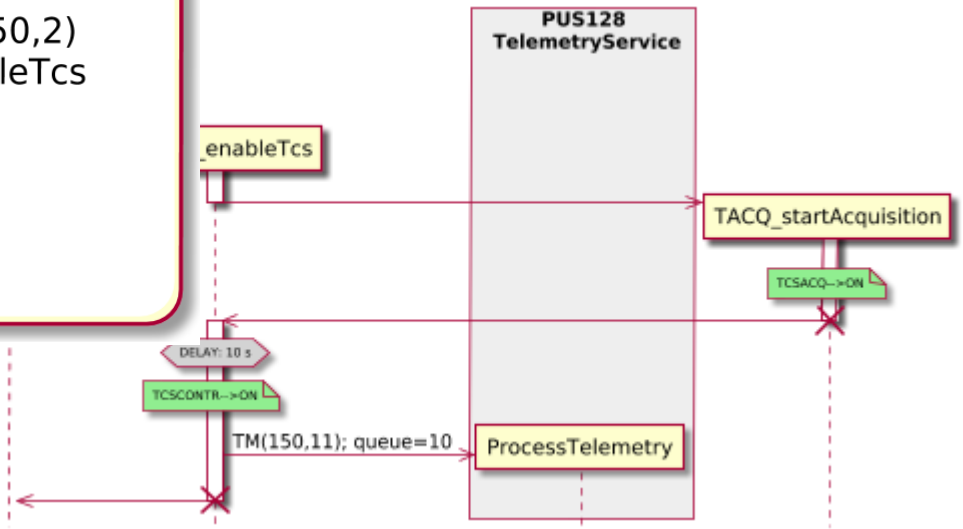
Infrastructure Specifics in C

```
Activity enableTcs with Nummeric Id 1 is commandable by TC(150,1)
  Short Description: enable thermal control
  Description: The thermal control heats the system if it is too cold. The switching histeresis can be configured.
  Constraints:
    0: TCSCONTR.inMode(OFF ) // switching on is possible only if the TCS is off
  In-Parameter:
    int16/°C/ upperThreshold: constrained :  <no constraint> // upper switching threshold
    int16/°C/ lowerThreshold: constrained :  lowerThreshold < upperThreshold // lower switching threshold
    component<TemperatureAcquisition> acq: constrained :  <no constraint> // acqusition component instance to use
  {
    REQUEST acq.startAcquisition ( << ... >> ) --> ( << ... >> )
      on error do nothing special
      on error abort
    UPTH = upperThreshold;
    LOTH = lowerThreshold;
    DELAY for 10 s
    TCSCONTR.setMode(ON);
    TELEMETRY (150,11)
      Description: Report switching on in a dedicated packet that reports the initial temperature.
      [initialTemp : int32/°C/ = PUS150.AVTEMP // initial temperature when starting thermal control   ]
  }

Activity disableTcs with Nummeric Id 2 is commandable by TC(150,2)
  Short Description: disable thermal control
  Description:
  Constraints:
    0: TCSCONTR.inMode(ON ) // switching off is possible only if the TCS is on
  In-Parameter:
    << ... >>
  {
    TCSCONTR.setMode(OFF);
    REQUEST TACQA.stopAcquisition ( << ... >> ) --> ( << ... >> )
      on error do nothing special
    REQUEST TACQB.stopAcquisition ( << ... >> ) --> ( << ... >> )
      on error do nothing special

  }
} Component ThermalControlSystem
```

Generated Documentation







Separation of concerns is key
to avoid the legacy trap

DSLs can isolate business logic
completely from technical concerns

DSLs can help integrate domain experts
with communication/review or even coding

Language Workbenches enable DSLs
by reducing effort to build, compose and maintain them

Migrating to a new LWB is feasible
b/c semantics of all models are known, by definition.