Complete Isolation of Business Logic using DSLs

THE ARCHITECTURE GATHERING

Markus Völter

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

voelter {ingenieurbüro für softwaretechnologie //itemis

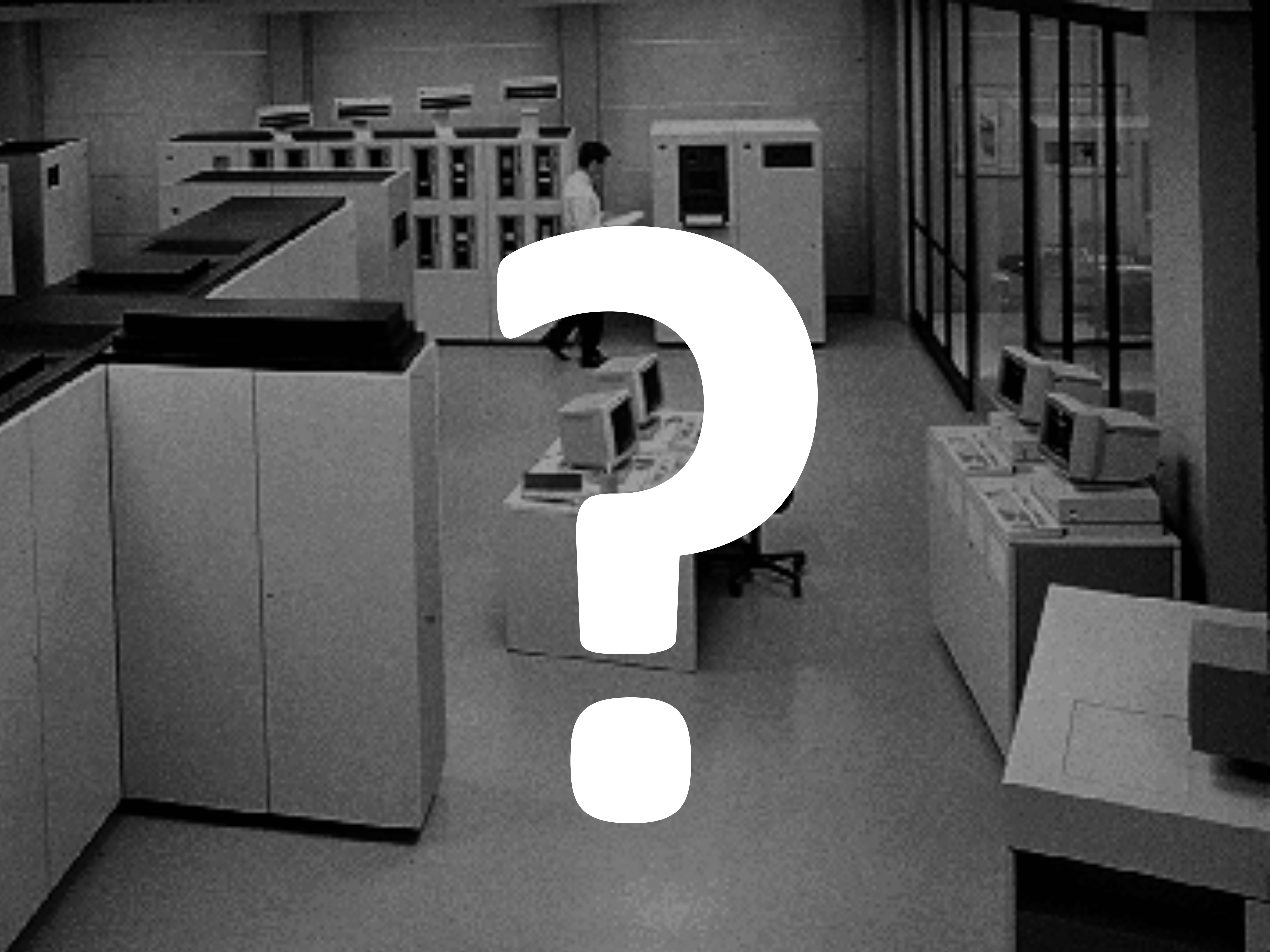
Complete Isolation of Business Logic using DSLs

One of the most important architectural goals, IMHO!

I hope to help move it out of the niche it‘s in right now.

1

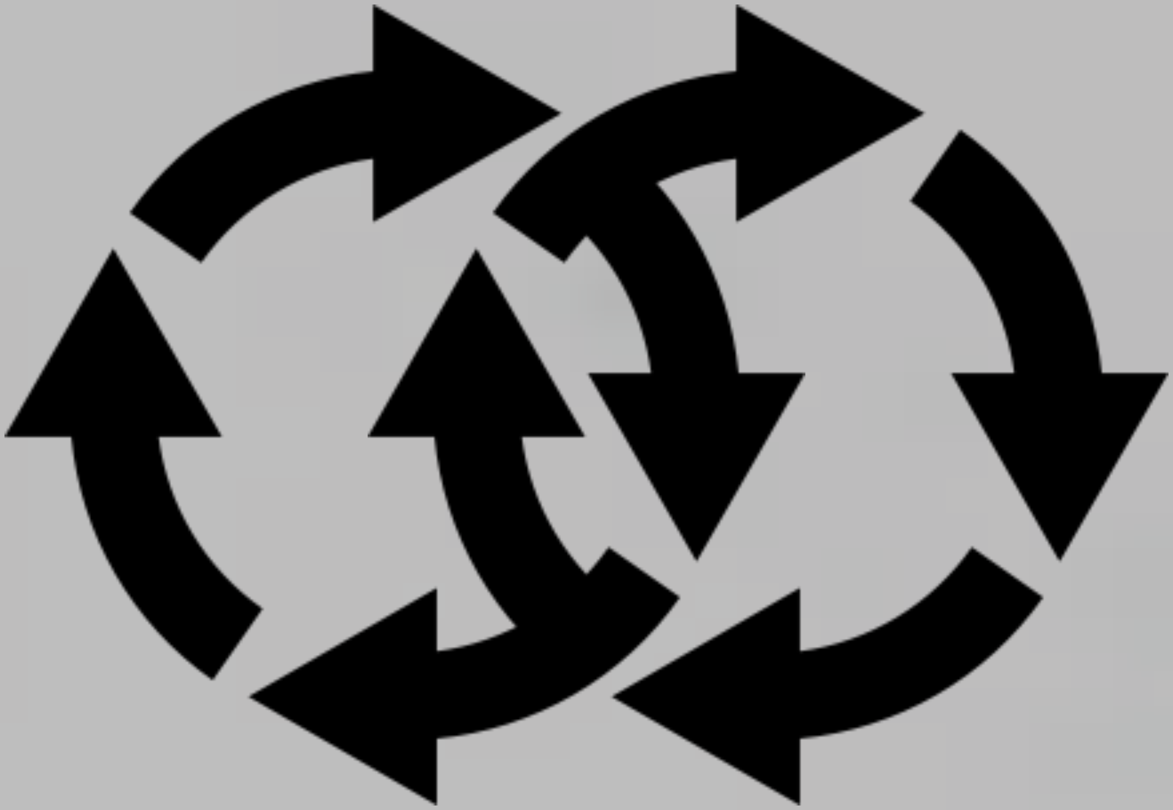
Legacy Systems



Outdated Technology
Obscure Business Logic

You can‘t understand/evolve/ extract them independently.

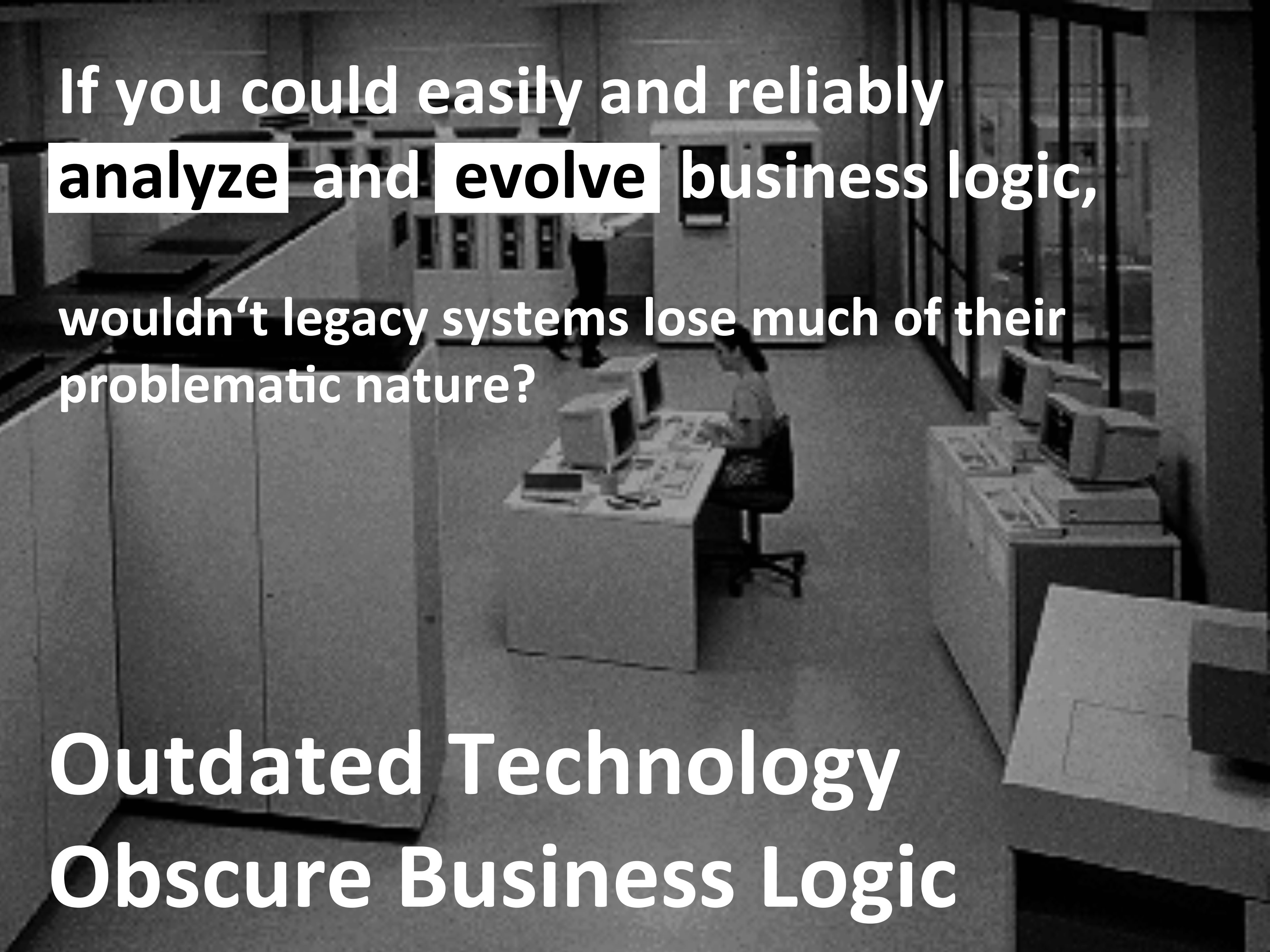
Technology & Business Logic now have connected lifecycles.



If you could reliably extract the business logic and automatically transform it to run on a new technology platform,

wouldn‘t legacy systems lose much of their problematic nature?

Outdated Technology
Obscure Business Logic



If you could easily and reliably analyze and evolve business logic,

wouldn‘t legacy systems lose much of their problematic nature?

Outdated Technology
Obscure Business Logic

[Business Logic]

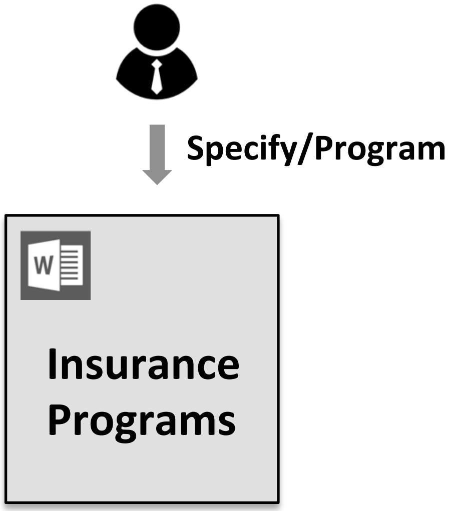
It‘s what makes a business tick.
Distinguishes the business.

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

Processes
UIs

Example

A real Example: Legacy



Specify/Program

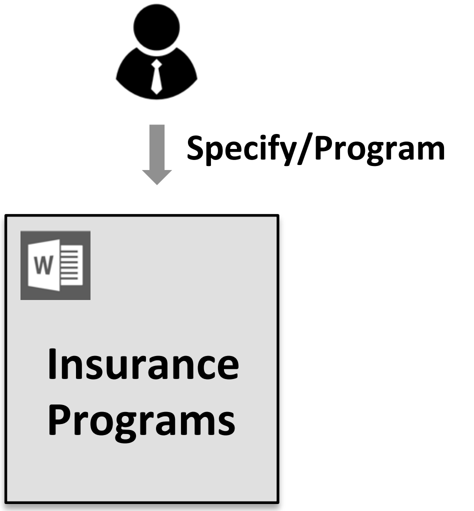
Insurance Programs

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

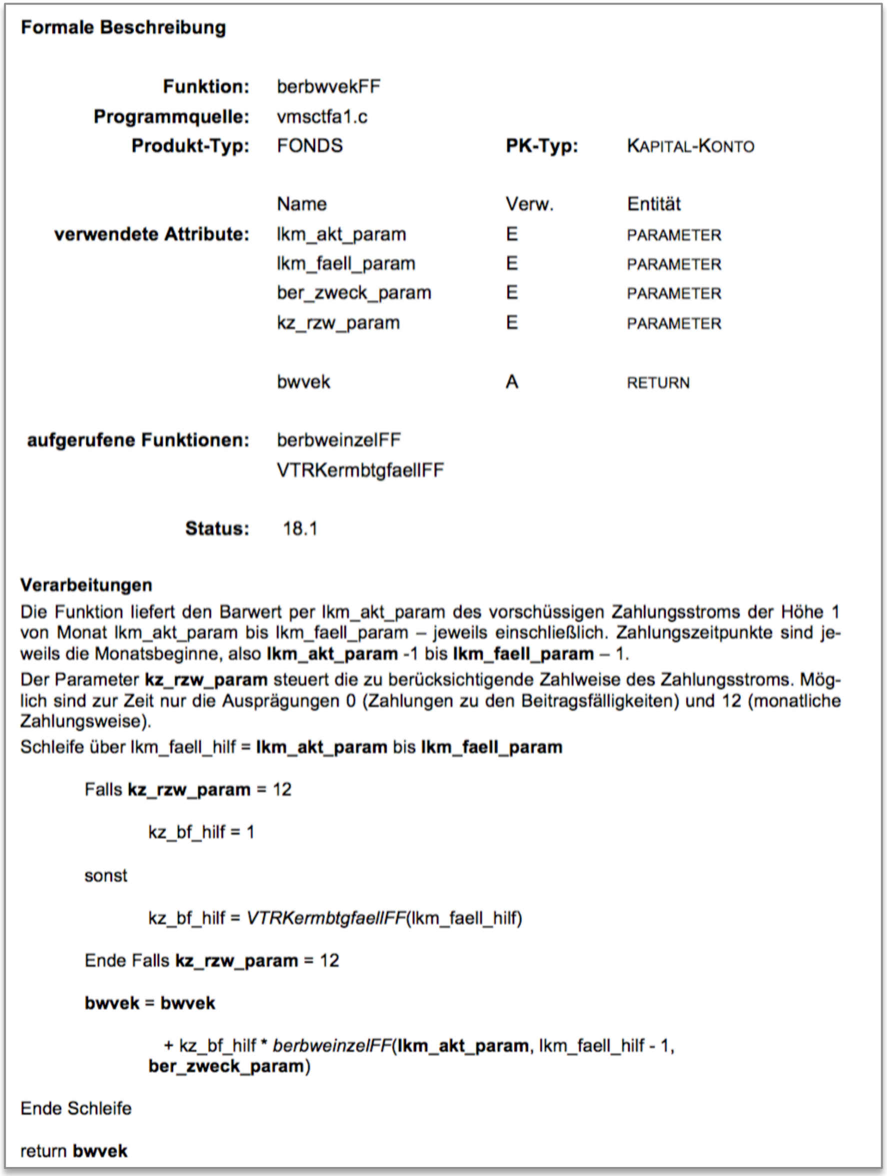
A real Example: Legacy



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		

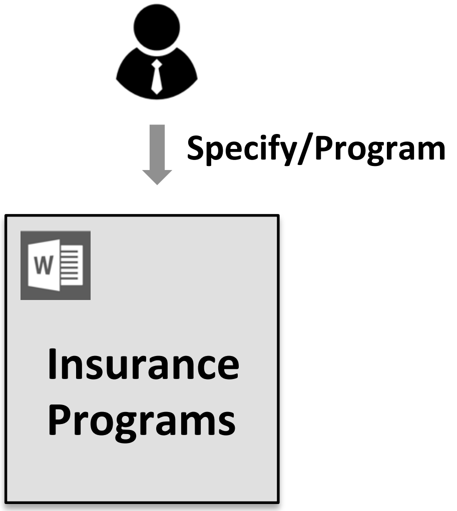
Verarbeitungen

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
```

A real Example: Legacy



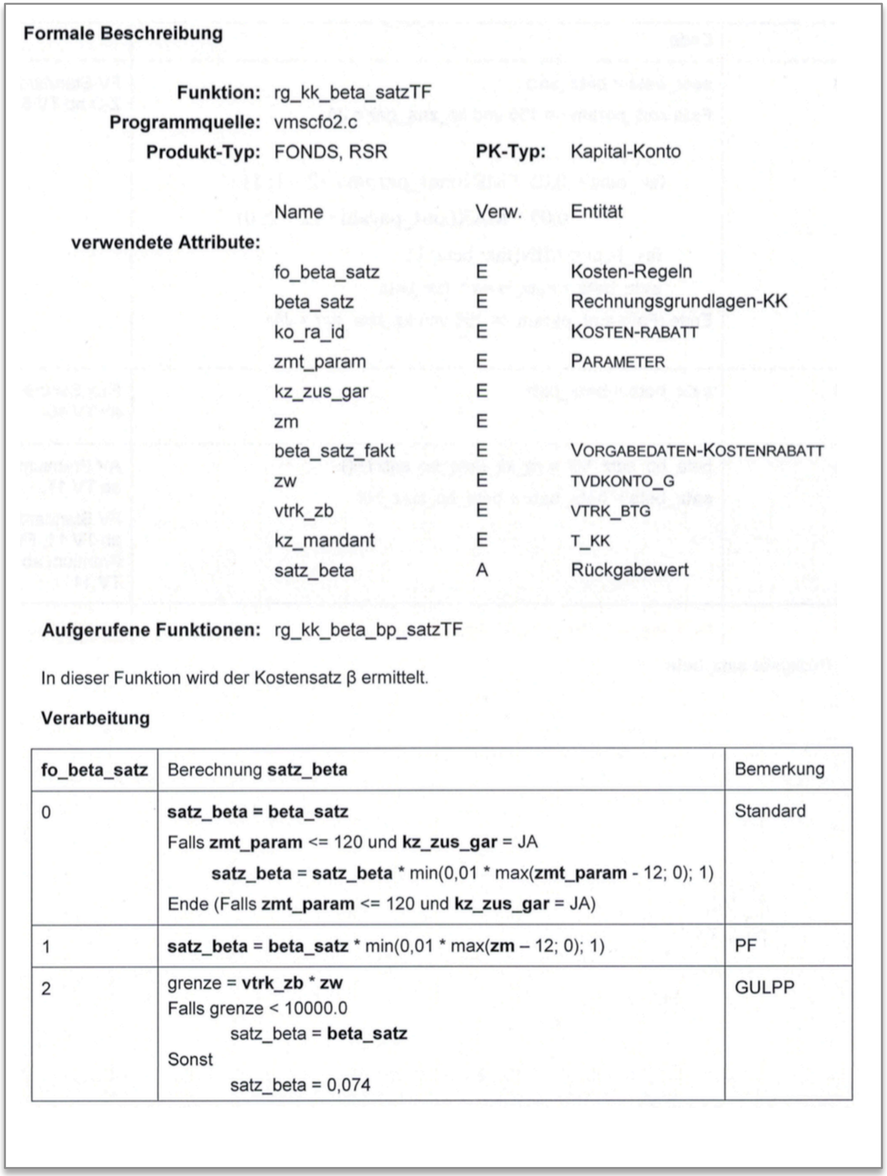
Specify/Program

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

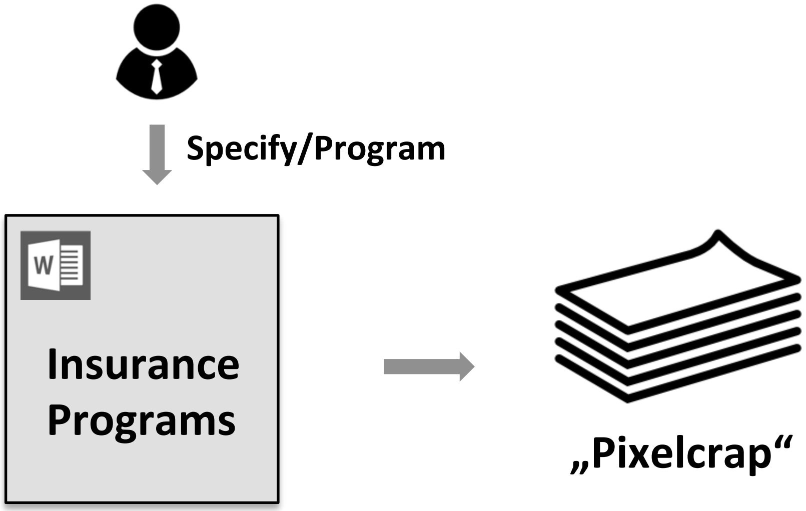
In dieser Funktion wird der Kostensatz β ermittelt.

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



Specify/Program

Insurance Programs

„Pixelcrap“

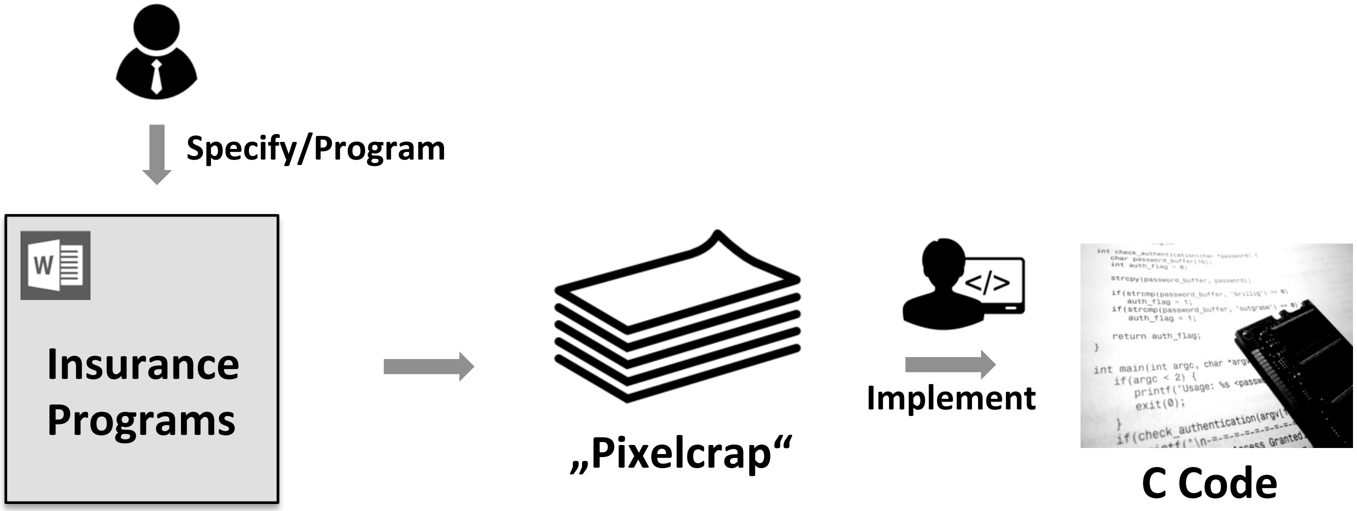
Write formal code in a DSL mixed with tables and text

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No varibility

Printed, PDF

A real Example: Legacy



Specify/Program

Insurance Programs

„Pixelcrap“

Implement

C Code

Write formal code in a DSL
mixed with tables and text

No tool support whatsoever
No testing (except inspection)

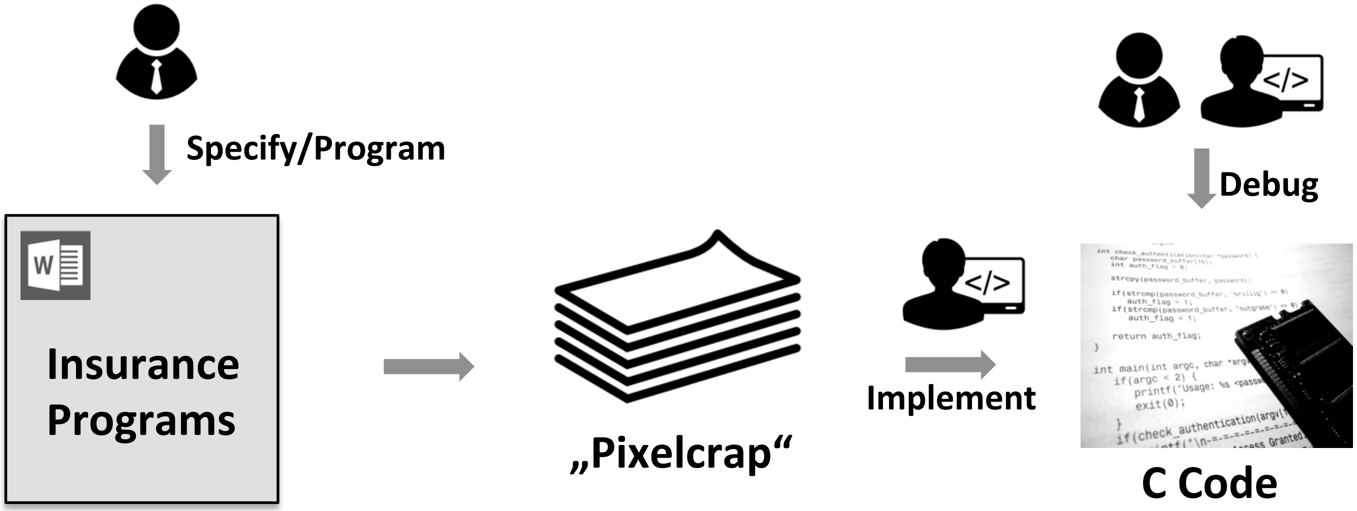
No reuse
No modularity
No varibility

Printed, PDF

Developer reads „spec“
Very idiomatic implementation

Dev acts as a human compiler and implements it in C

A real Example: Legacy



Specify/Program

Insurance Programs

„Pixelcrap“

Implement

Debug

C Code

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)

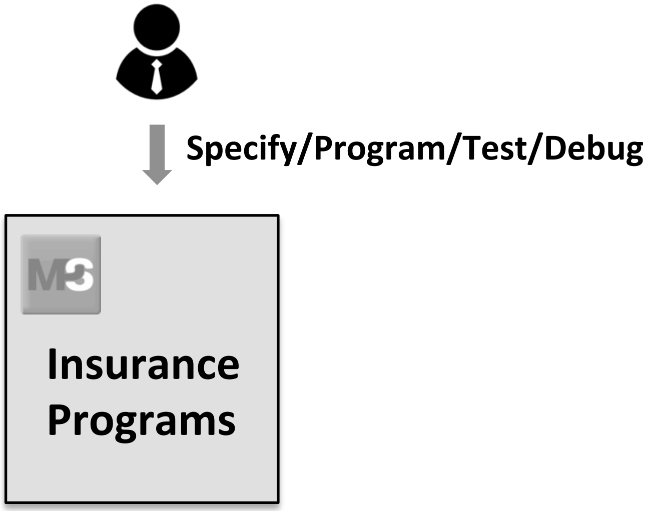
No reuse
No modularity
No varibility

Printed, PDF

Developer reads „spec“
Very idiomatic implementation

Dev acts as a human compiler and implements it in C

A real Example: Current



Specify/Program/Test/Debug

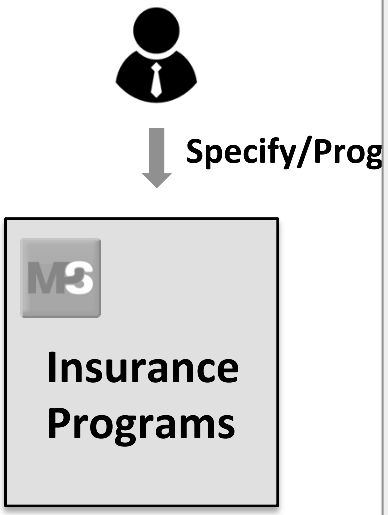
Insurance Programs

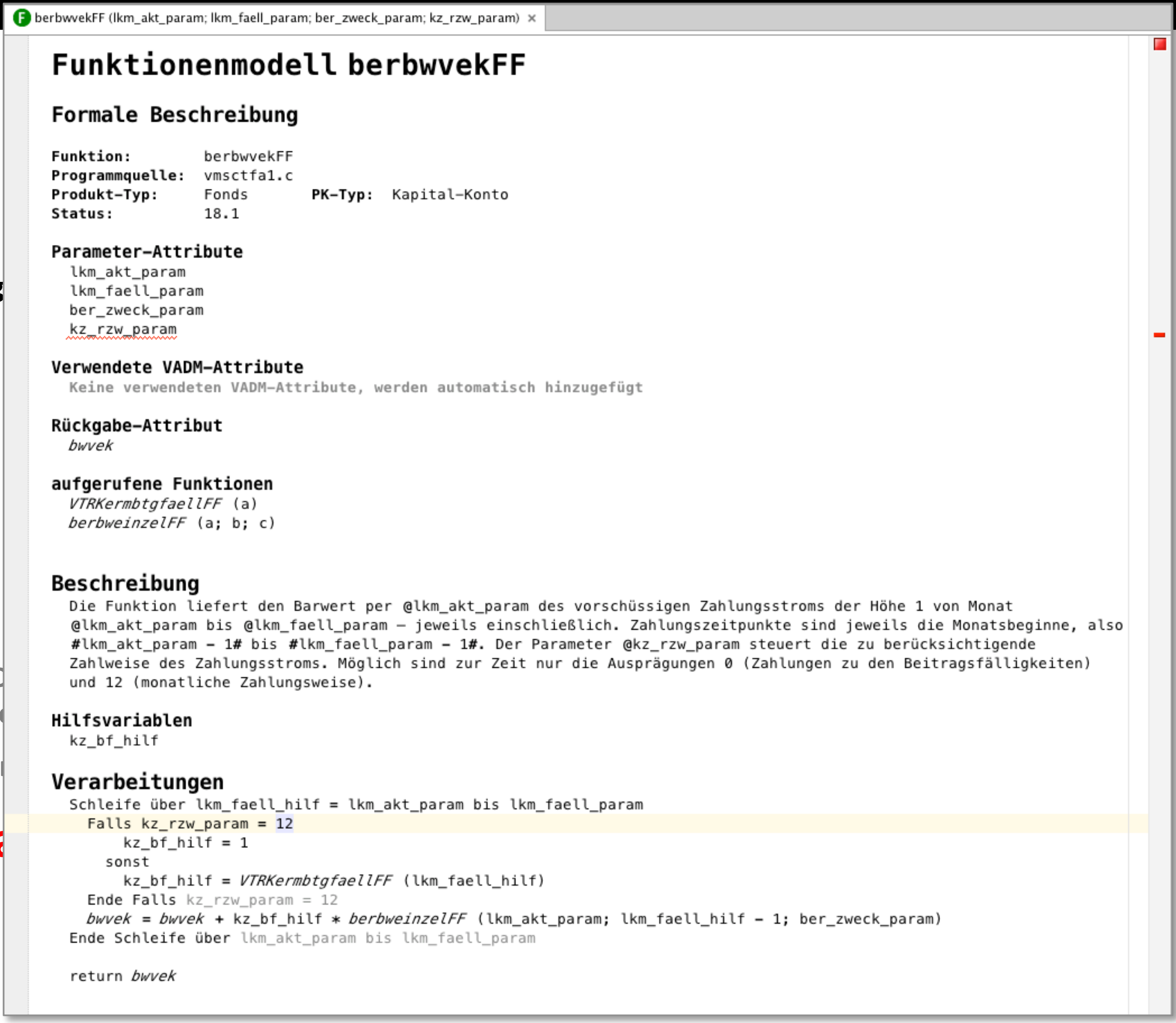
Write formal code in a DSL
mixed with tables and text

Now with IDE support and executable tests

The same notation!

A real Example: Current




berbwvekFF (lkm_akt_param; lkm_faell_param; ber_zweck_param; kz_rzw_param)

Funktionenmodell berbwvekFF

Formale Beschreibung

```
Funktion:         berbwvekFF
Programmquelle:   vmsctfa1.c
Produkt-Typ:      Fonds        PK-Typ:   Kapital-Konto
Status:           18.1

Parameter-Attribute
  lkm_akt_param
  lkm_faell_param
  ber_zweck_param
  kz_rzw_param

Verwendete VADM-Attribute
  Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut
  bwvek

aufgerufene Funktionen
  VTRKermbtgfaellFF (a)
  berbweinzelFF (a; b; c)
```

Beschreibung

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).

```
Hilfsvariablen
  kz_bf_hilf
```

Verarbeitungen

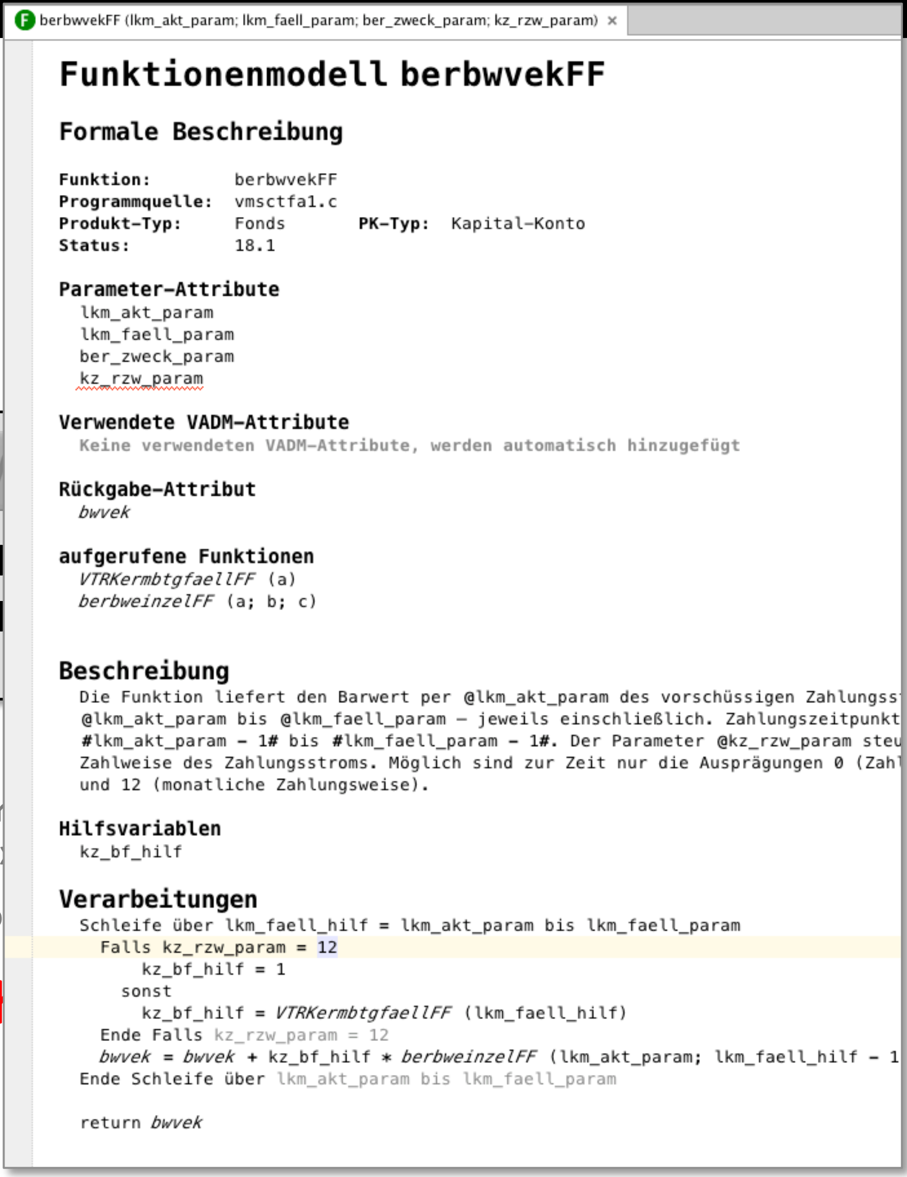
```
Schleife über lkm_faell_hilf = lkm_akt_param bis lkm_faell_param
  Falls kz_rzw_param = 12
      kz_bf_hilf = 1
    sonst
      kz_bf_hilf = VTRKermbtgfaellFF (lkm_faell_hilf)
  Ende Falls kz_rzw_param = 12
  bwvek = bwvek + kz_bf_hilf * berbweinzelFF (lkm_akt_param; lkm_faell_hilf - 1; ber_zweck_param)
Ende Schleife über lkm_akt_param bis lkm_faell_param

return bwvek
```

Write formal code in a D
mixed with tables and te

Now with IDE support a

The same nota

A real Example: Current



berbwvekFF (lkm_akt_param; lkm_faell_param; ber_zweck_param; kz_rzw_param) ×

Funktionenmodell berbwvekFF

Formale Beschreibung

Funktion:	berbwvekFF		
Programmquelle:	vmsctfa1.c		
Produkt-Typ:	Fonds	**PK-Typ:**	Kapital-Konto
Status:	18.1		

Parameter-Attribute

```
  lkm_akt_param
  lkm_faell_param
  ber_zweck_param
  kz_rzw_param
```

Verwendete VADM-Attribute

Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut

bwvek

aufgerufene Funktionen

```
  VTRKermbtgfaellFF (a)
  berbweinzelFF (a; b; c)
```

Beschreibung

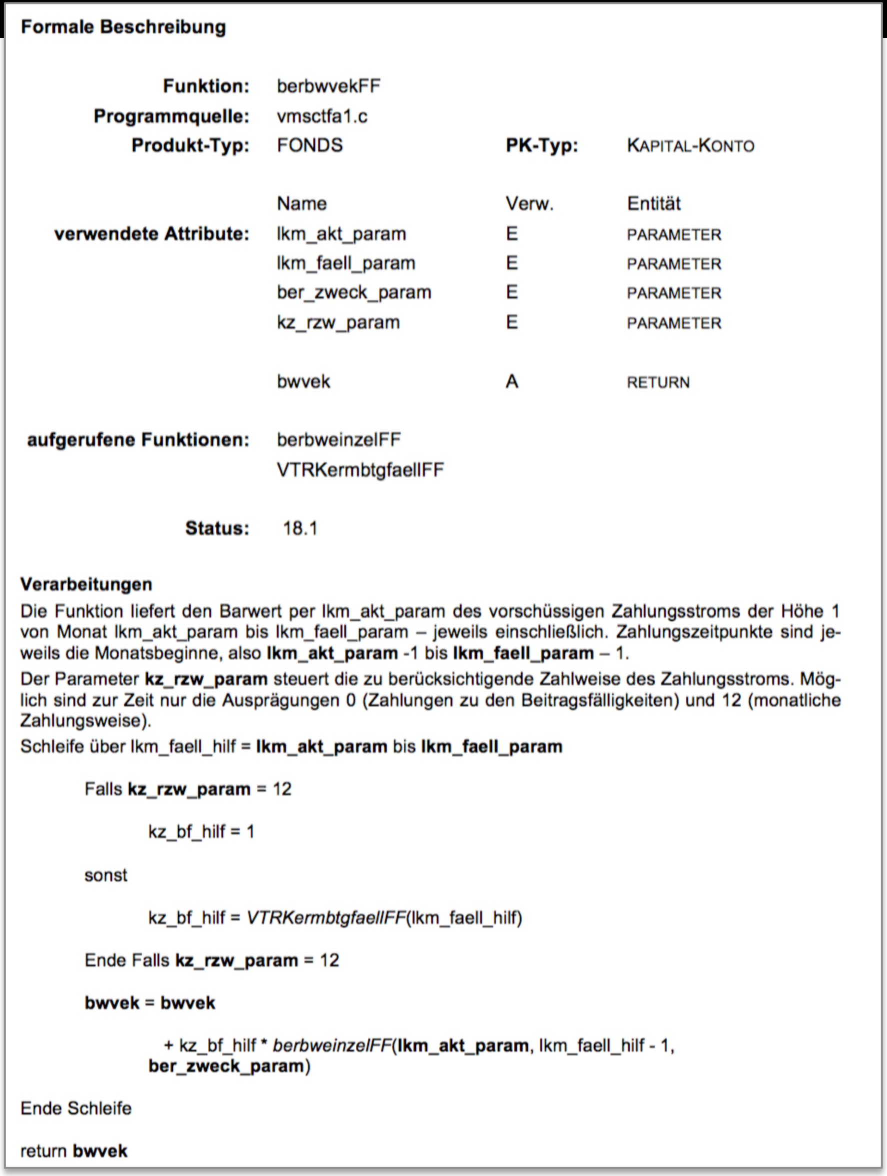
```
  Die Funktion liefert den Barwert per @lkm_akt_param des vorschüssigen Zahlungss
  @lkm_akt_param bis @lkm_faell_param – jeweils einschließlich. Zahlungszeitpunkt
  #lkm_akt_param – 1# bis #lkm_faell_param – 1#. Der Parameter @kz_rzw_param steu
  Zahlweise des Zahlungsstroms. Möglich sind zur Zeit nur die Ausprägungen 0 (Zahl
  und 12 (monatliche Zahlungsweise).
```

Hilfsvariablen

```
  kz_bf_hilf
```

Verarbeitungen

```
  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
  Ende Schleife über lkm_akt_param bis lkm_faell_param

  return bwvek
```



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
```

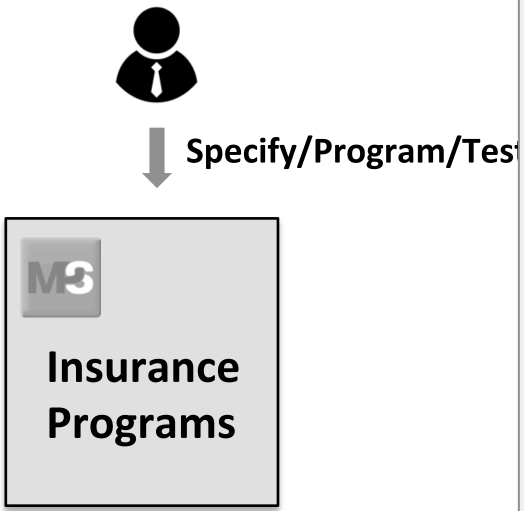
M

Wr
mi

No

Th

A real Example: Current

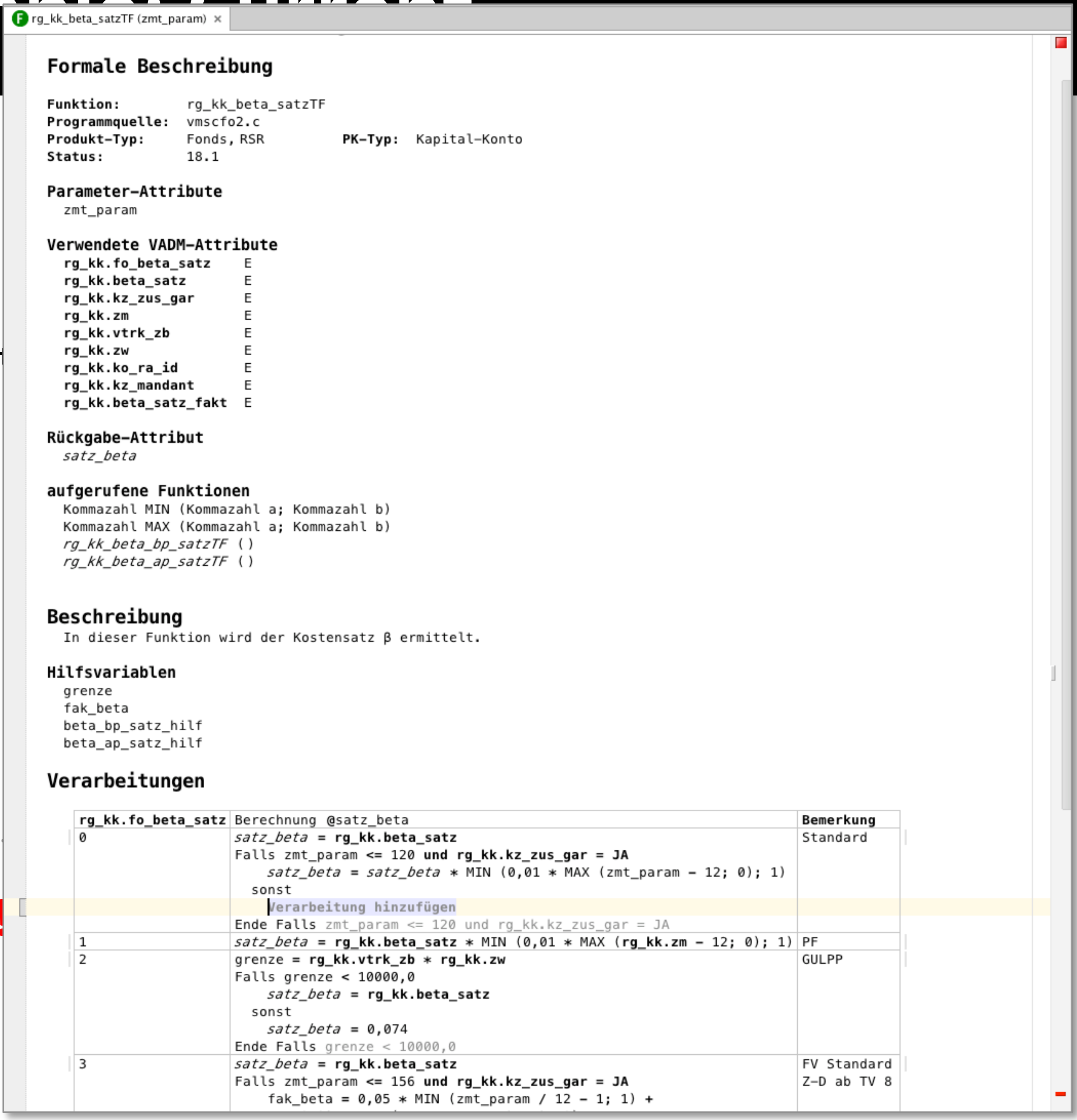

Specify/Program/Test

Insurance Programs

Write formal code in a DSL
mixed with tables and text

Now with IDE support and execu

The same notation!



Formale Beschreibung

```
Funktion:         rg_kk_beta_satzTF
Programmquelle:   vmscfo2.c
Produkt-Typ:      Fonds, RSR          PK-Typ:  Kapital-Konto
Status:           18.1
```

Parameter-Attribute

```
  zmt_param
```

Verwendete VADM-Attribute

```
  rg_kk.fo_beta_satz     E
  rg_kk.beta_satz        E
  rg_kk.kz_zus_gar       E
  rg_kk.zm               E
  rg_kk.vtrk_zb          E
  rg_kk.zw               E
  rg_kk.ko_ra_id         E
  rg_kk.kz_mandant       E
  rg_kk.beta_satz_fakt   E
```

Rückgabe-Attribut

```
  satz_beta
```

aufgerufene Funktionen

```
  Kommazahl MIN (Kommazahl a; Kommazahl b)
  Kommazahl MAX (Kommazahl a; Kommazahl b)
  rg_kk_beta_bp_satzTF ()
  rg_kk_beta_ap_satzTF ()
```

Beschreibung

In dieser Funktion wird der Kostensatz β ermittelt.

Hilfsvariablen

```
  grenze
  fak_beta
  beta_bp_satz_hilf
  beta_ap_satz_hilf
```

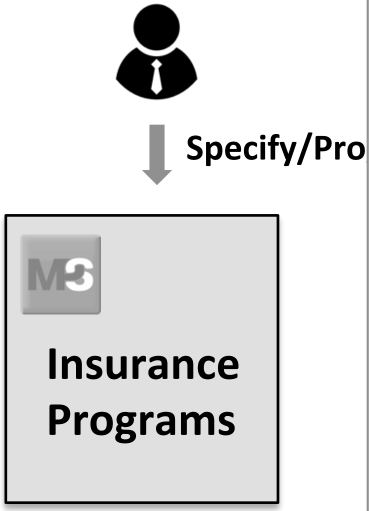
Verarbeitungen

rg_kk.fo_beta_satz	Berechnung @satz_beta	Bemerkung
0	satz_beta = rg_kk.beta_satz Falls zmt_param <= 120 und rg_kk.kz_zus_gar = JA satz_beta = satz_beta * MIN (0,01 * MAX (zmt_param - 12; 0); 1) sonst Verarbeitung hinzufügen Ende Falls zmt_param <= 120 und rg_kk.kz_zus_gar = JA	Standard
1	satz_beta = rg_kk.beta_satz * MIN (0,01 * MAX (rg_kk.zm - 12; 0); 1)	PF

2	grenze = rg_kk.vtrk_zb * rg_kk.zw
	Falls grenze < 10000,0
	satz_beta = rg_kk.beta_satz
	sonst
	satz_beta = 0,074
	Ende Falls grenze < 10000,0
		GULPP

3	satz_beta = rg_kk.beta_satz
	Falls zmt_param <= 156 und rg_kk.kz_zus_gar = JA
	fak_beta = 0,05 * MIN (zmt_param / 12 - 1; 1) +
		FV Standard
		Z-D ab TV 8

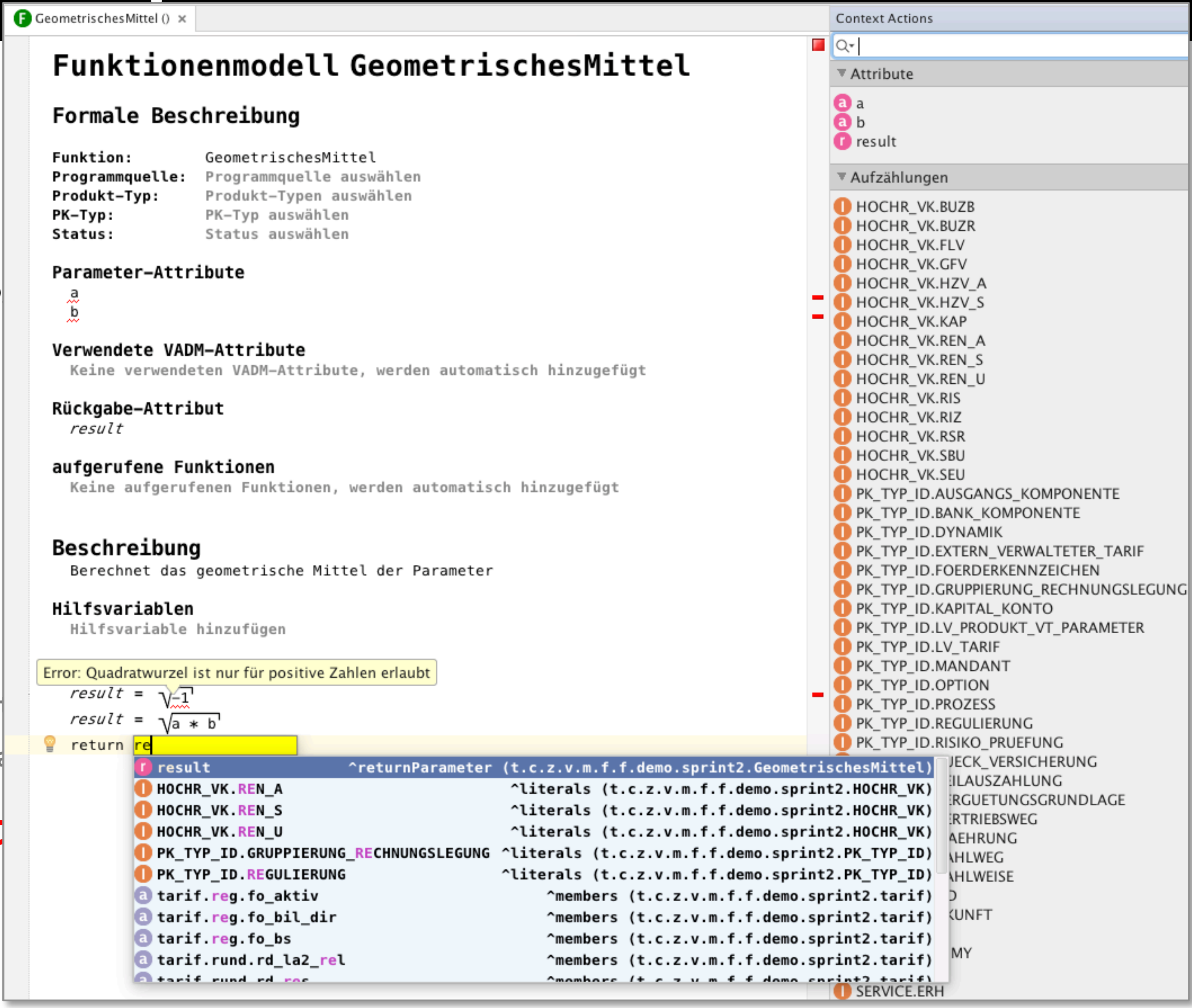
A real Example: Current



Write formal code in a
mixed with tables and

Now with IDE support a

The same not



```
Funktionenmodell GeometrischesMittel

Formale Beschreibung

Funktion:        GeometrischesMittel
Programmquelle:  Programmquelle auswählen
Produkt-Typ:     Produkt-Typen auswählen
PK-Typ:          PK-Typ auswählen
Status:          Status auswählen

Parameter-Attribute
  a
  b

Verwendete VADM-Attribute
  Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut
  result

aufgerufene Funktionen
  Keine aufgerufenen Funktionen, werden automatisch hinzugefügt

Beschreibung
  Berechnet das geometrische Mittel der Parameter

Hilfsvariablen
  Hilfsvariable hinzufügen

Error: Quadratwurzel ist nur für positive Zahlen erlaubt
  result = √-1
  result = √a * b
  return re
```

```
r result                                   ^returnParameter (t.c.z.v.m.f.f.demo.sprint2.GeometrischesMittel)
I HOCHR_VK.REN_A                                  ^literals (t.c.z.v.m.f.f.demo.sprint2.HOCHR_VK)
I HOCHR_VK.REN_S                                  ^literals (t.c.z.v.m.f.f.demo.sprint2.HOCHR_VK)
I HOCHR_VK.REN_U                                  ^literals (t.c.z.v.m.f.f.demo.sprint2.HOCHR_VK)
I PK_TYP_ID.GRUPPIERUNG_RECHNUNGSLEGUNG        ^literals (t.c.z.v.m.f.f.demo.sprint2.PK_TYP_ID)
I PK_TYP_ID.REGULIERUNG                        ^literals (t.c.z.v.m.f.f.demo.sprint2.PK_TYP_ID)
a tarif.reg.fo_aktiv                                 ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
a tarif.reg.fo_bil_dir                               ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
a tarif.reg.fo_bs                                    ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
a tarif.rund.rd_la2_rel                              ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
a tarif.rund.rd_res                                  ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
```

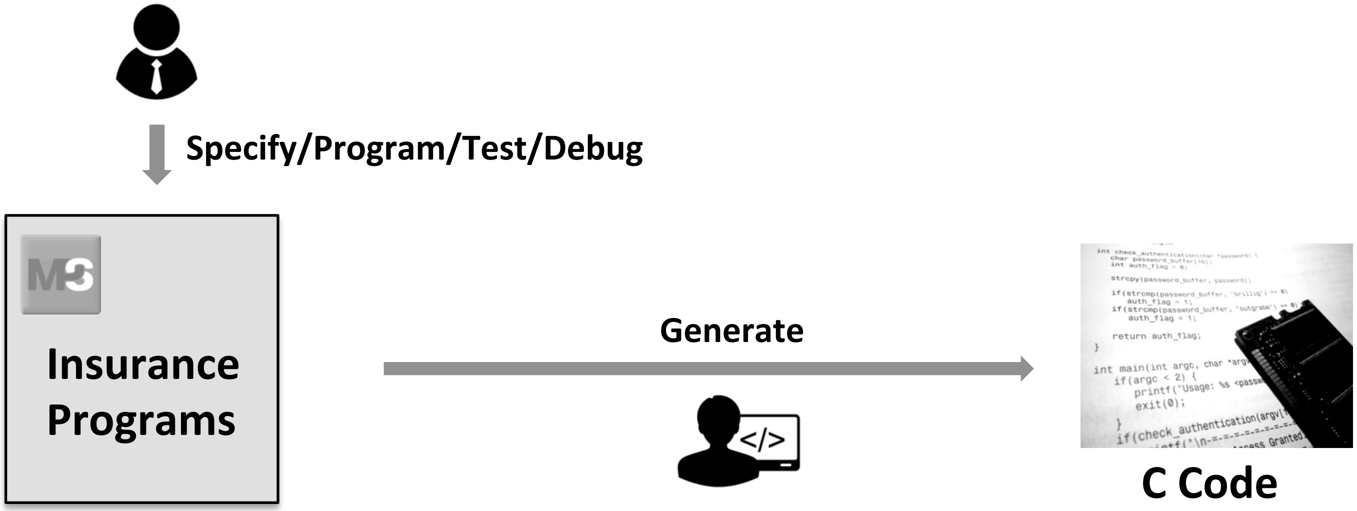
Attribute

- a a
- a b
- r result

Aufzählungen

- HOCHR_VK.BUZB
- HOCHR_VK.BUZR
- HOCHR_VK.FLV
- HOCHR_VK.GFV
- HOCHR_VK.HZV_A
- HOCHR_VK.HZV_S
- HOCHR_VK.KAP
- HOCHR_VK.REN_A
- HOCHR_VK.REN_S
- HOCHR_VK.REN_U
- HOCHR_VK.RIS
- HOCHR_VK.RIZ
- HOCHR_VK.RSR
- HOCHR_VK.SBU
- HOCHR_VK.SEU
- PK_TYP_ID.AUSGANGS_KOMPONENTE
- PK_TYP_ID.BANK_KOMPONENTE
- PK_TYP_ID.DYNAMIK
- PK_TYP_ID.EXTERN_VERWALTETER_TARIF
- PK_TYP_ID.FOERDERKENNZEICHEN
- PK_TYP_ID.GRUPPIERUNG_RECHNUNGSLEGUNG
- PK_TYP_ID.KAPITAL_KONTO
- PK_TYP_ID.LV_PRODUKT_VT_PARAMETER
- PK_TYP_ID.LV_TARIF
- PK_TYP_ID.MANDANT
- PK_TYP_ID.OPTION
- PK_TYP_ID.PROZESS
- PK_TYP_ID.REGULIERUNG
- PK_TYP_ID.RISIKO_PRUEFUNG
- JECK_VERSICHERUNG
- EILAUSZAHLUNG
- ERGUETUNGSGRUNDLAGE
- ERTRIEBSWEG
- AEHRUNG
- AHLWEG
- AHLWEISE
- D
- KUNFT
- MY
- SERVICE.ERH

A real Example: Current



Specify/Program/Test/Debug

Insurance Programs

Generate

C Code

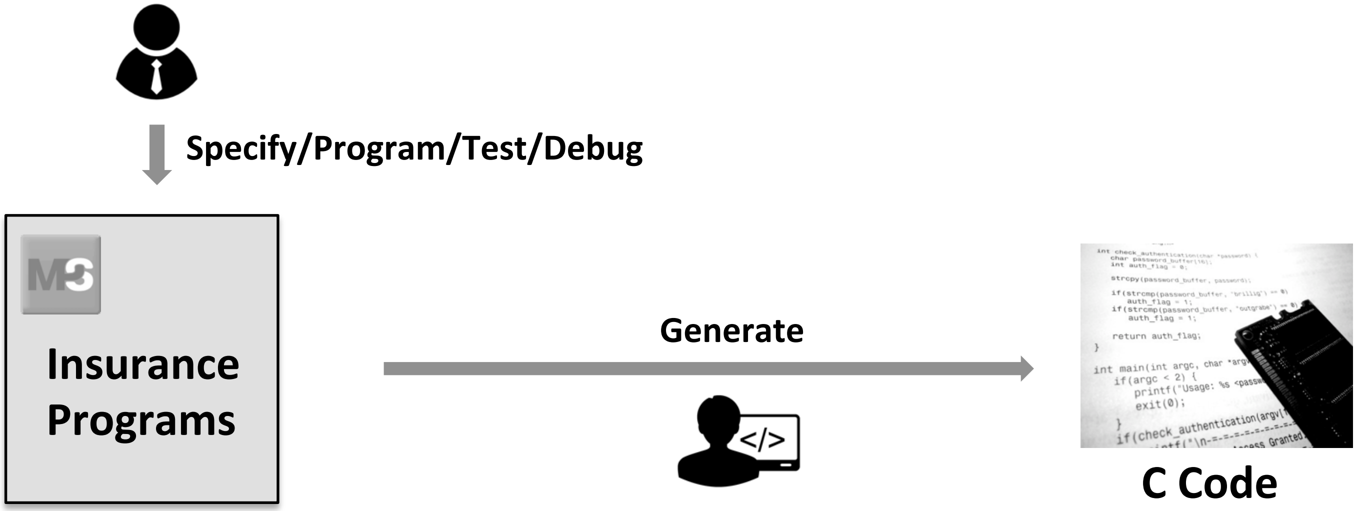
Write formal code in a DSL
mixed with tables and text

Now with IDE support and executable tests

The same notation!

Exactly the same C code.

A real Example: Future



Specify/Program/Test/Debug

Insurance Programs

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

Challenge: analyzing existing C Code

C is too flexible, too low-level and too „uncontrolled“ to implement analyzable business logic:

`malloc` vs. `free` mixed with business logic; pointer escaped.

Custom memory management inconsistent with standard `malloc`/`free`

different, inconsistent `#defines` for YES/NO or TRUE/FALSE

Misuse of the preprocessor

No „architecture“, dependencies everywhere

Bad modularity, no fine-grained unit tests

No functional abstractions, sideeffects everywhere

Missing first-class abstractions for core domain concepts (date)

String and double comparisons with `==`

Obscure Business Logic

Example Domains

Health and Medicine

Automotive

Aerospace

Robotics

Finance

Embedded Software

Science

Government

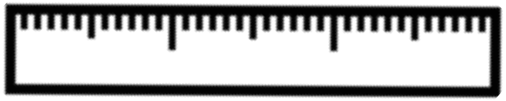
Law

Law enforcement

Example Domains

- Algorithms for diagnsis and medicine dosage
- Specifying communication relationships between software components
- Satellite behavior and telemetry/telecommanding
- Robotics behavioral algorithms, movement, collision avoidance
- Insurance contracts/rules, product specification
- Embedded algorithms for math
- Biomedical analysis algorithms
- Tax and public benefits rules
- Precise specification of logistics contracts, interactive execution
- Digital Forensics, identifying „bad“ patterns in files

$$$



?

$$$

Employee of a user:

I am committing myself to implement our next [system] within one person year [instead of 50].

A customer:

Using a prototype language/tool they built, they could reimplement months of work in a few days.
All tests ran.

$$$

Another case:

A customer had to schedule two weeks of work for their current supplier for a change that literally took minutes using the DSL.

Public Benefits Calculation:

We have been using such an approach for many years and could not imagine doing it any other way.

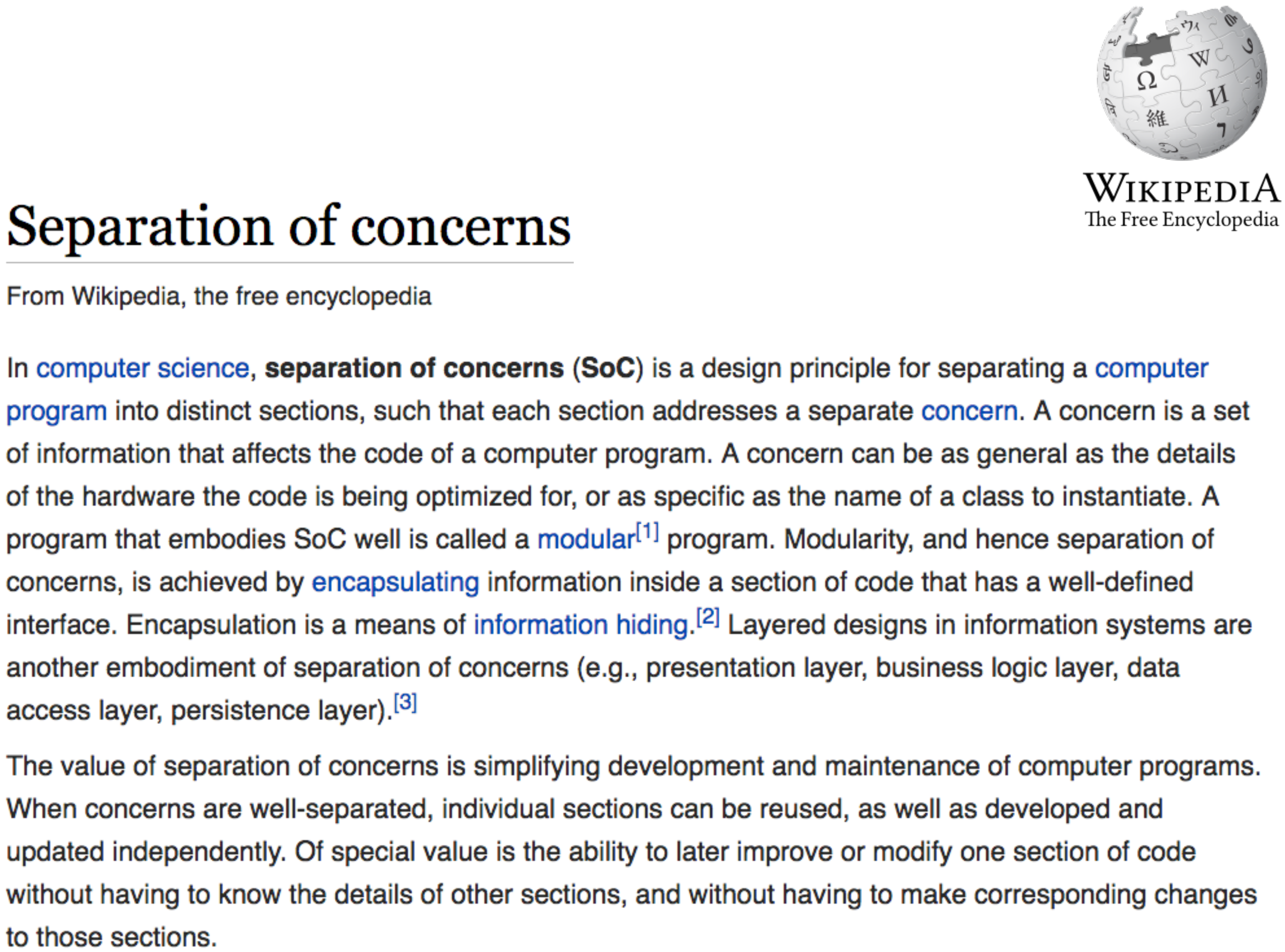
$$$

?

Judge for yourself :-)

2

Separation of Concerns



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

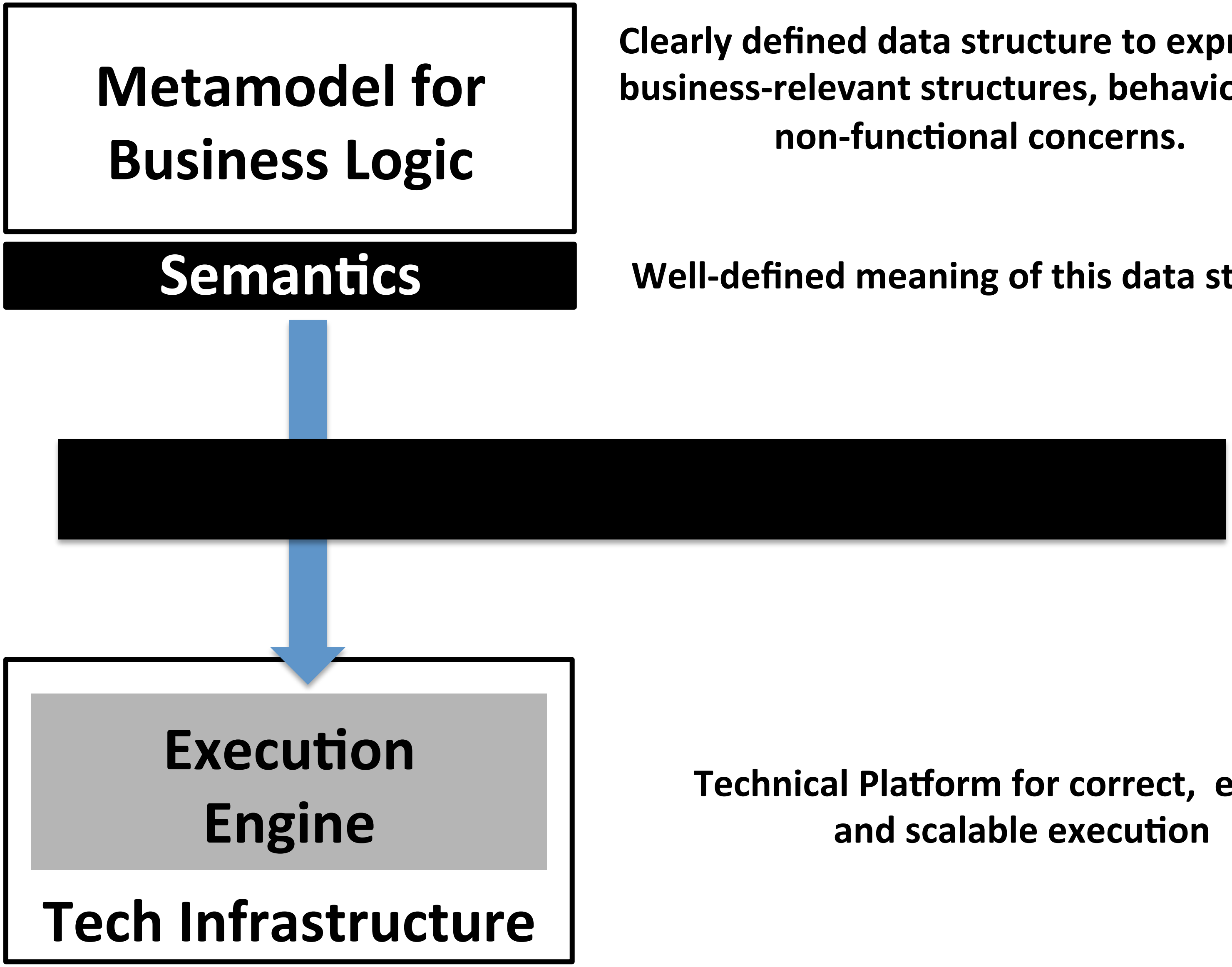
Clearly defined data structure to express all business-relevant structures, behaviors and non-functional concerns.

Semantics

Well-defined meaning of this data structure

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

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



Metamodel for Business Logic

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

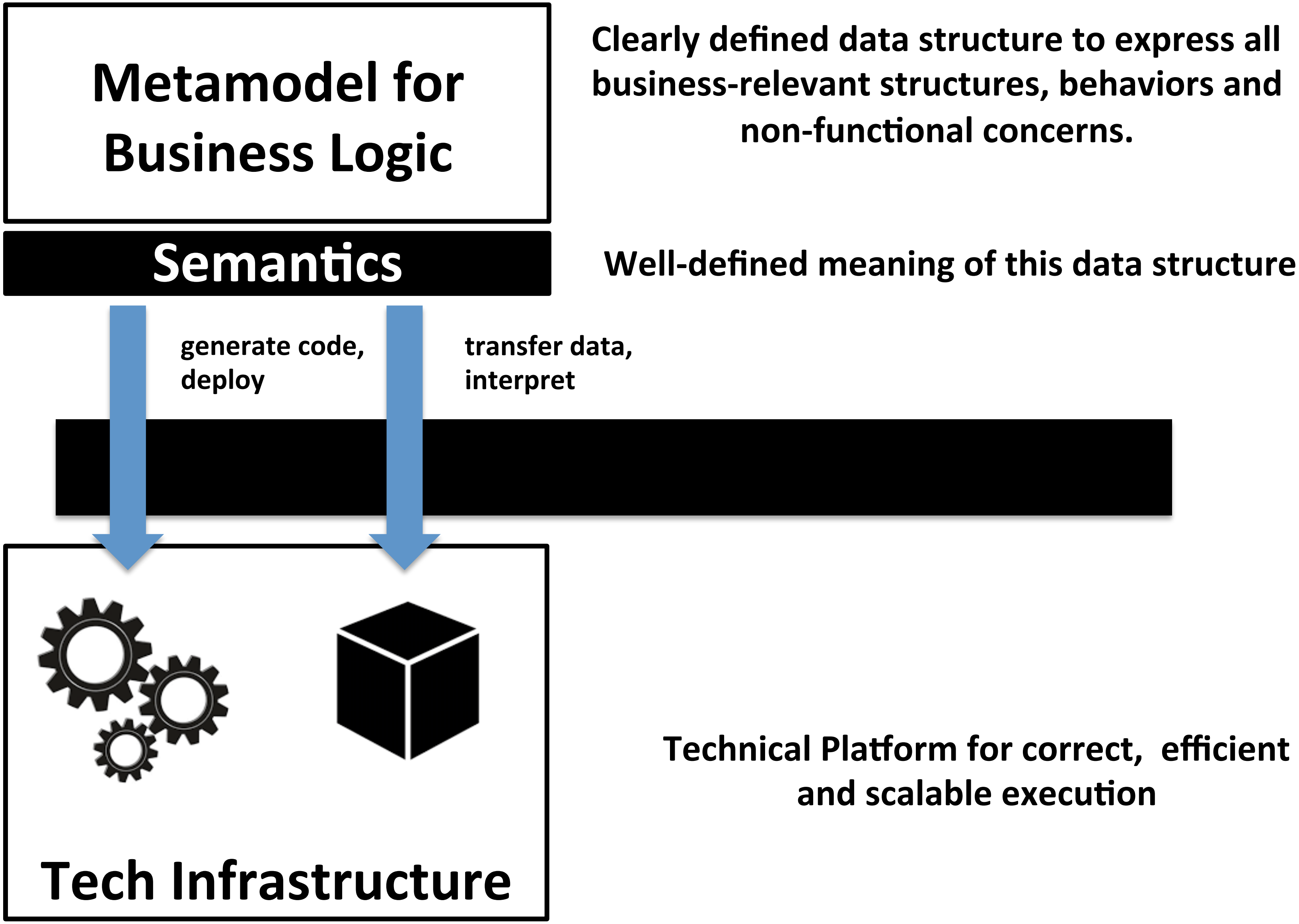
Semantics

Well-defined meaning of this data structure

Execution Engine

Tech Infrastructure

Technical Platform for correct, efficient and scalable execution



Metamodel for Business Logic

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

Semantics

Well-defined meaning of this data structure

generate code, deploy

transfer data, interpret

Technical Platform for correct, efficient and scalable execution

Tech Infrastructure

Transformation

+ Code Inspection

+ Debugging

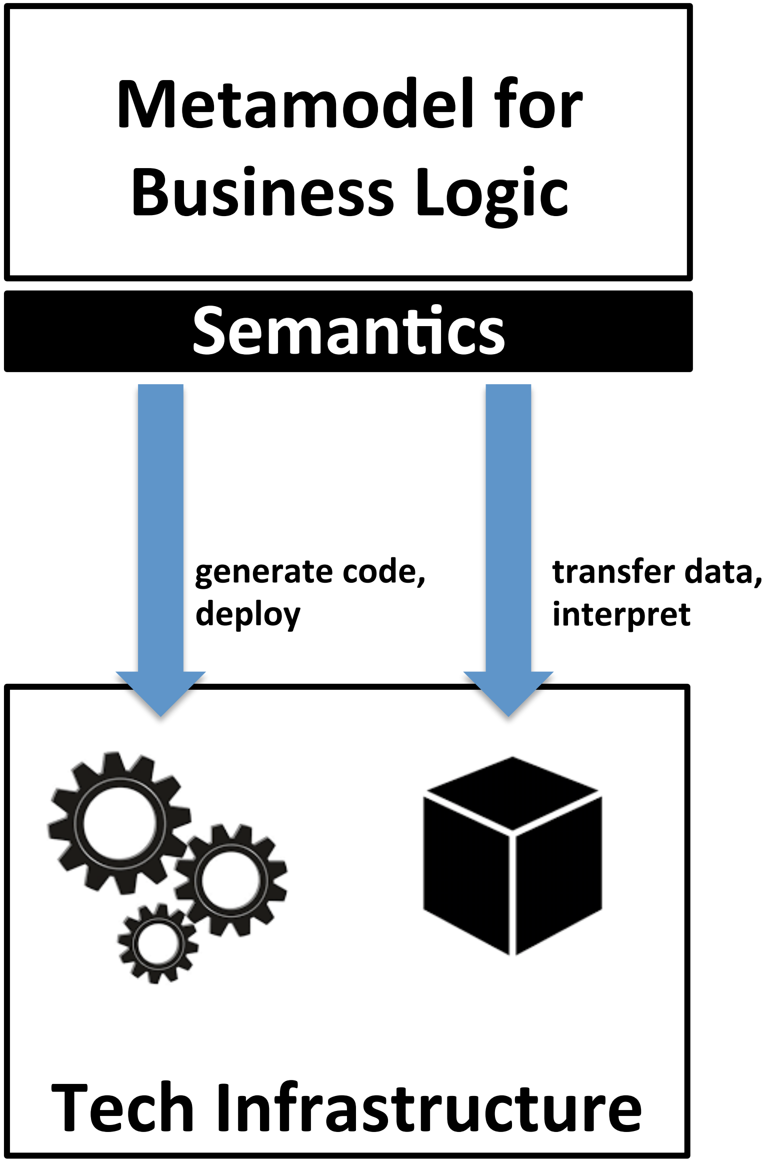
+ Performance & Optimization

+ Platform Conformance

Interpretation

+ Turnaround Time

+ Runtime Change



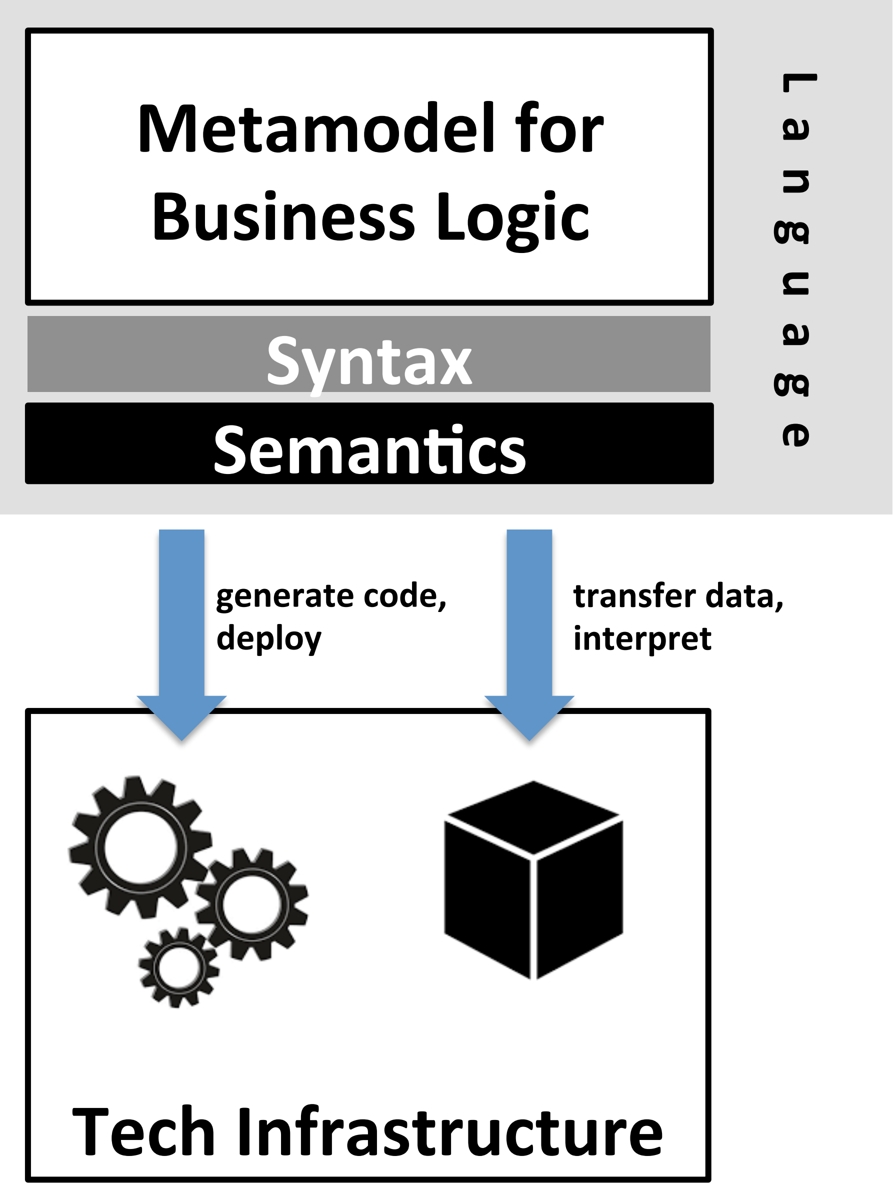
Metamodel for Business Logic

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!

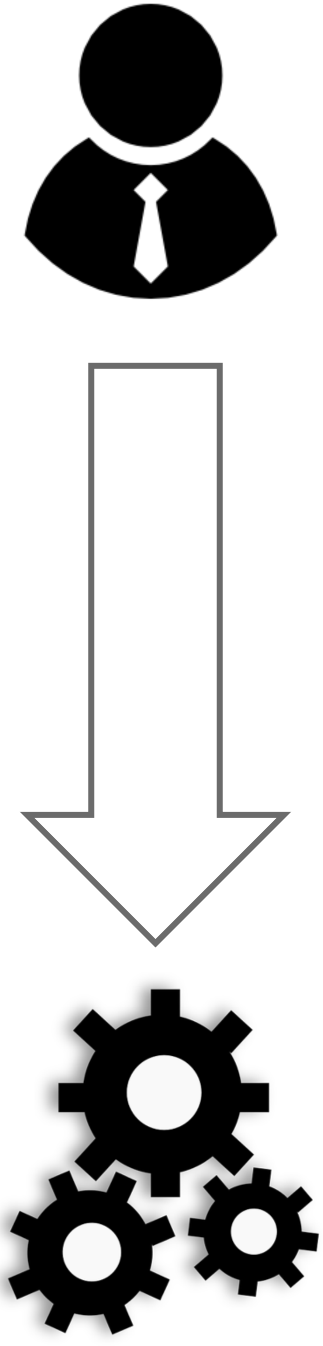
Expressivity for Core Business Logic

User-Friendly Notation
Great Tool/IDE

Testing

Meaningful Analyses

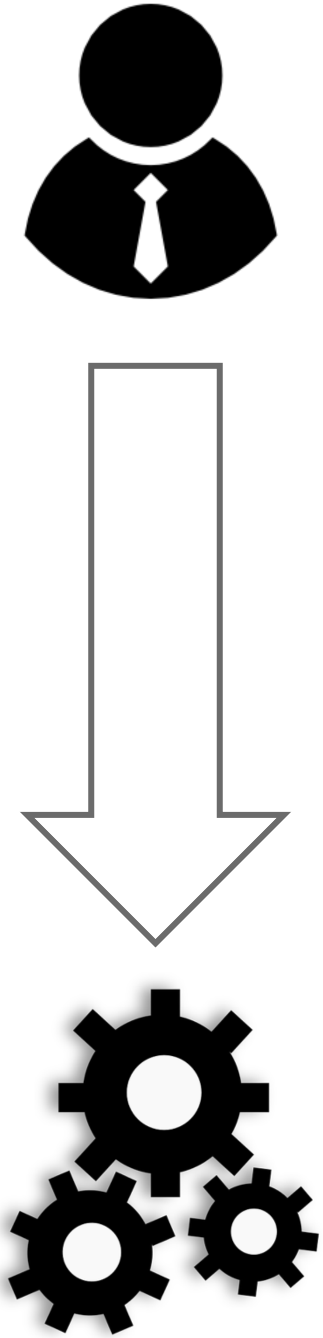
Execution



Levels of

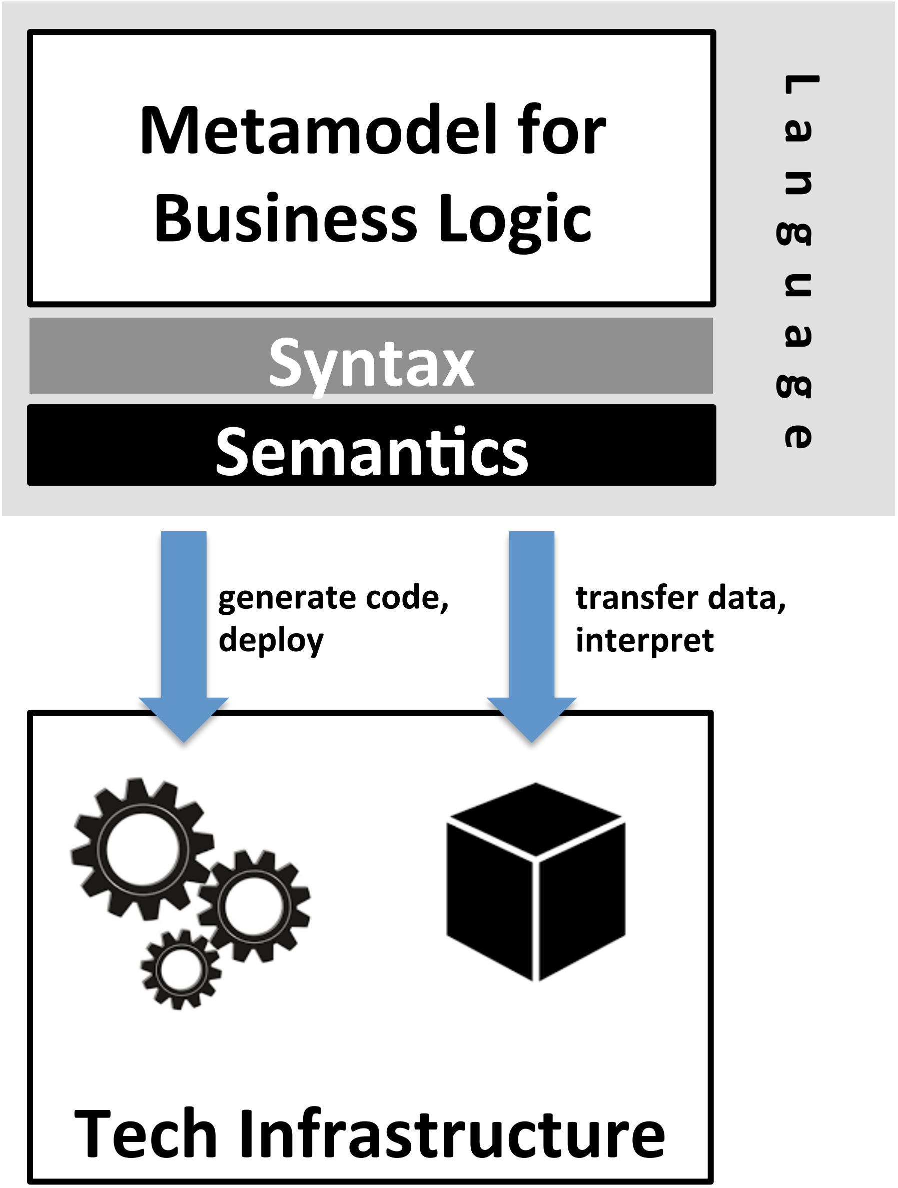
Domain Expert Integration

- Generate derived artifacts
- Review the DSL sources
- Pair programming
- Independent Development



Domain expert integration is great, but even without it, the approach is useful to avoid the legacy trap.

Exchangable Technology
Understandable Business Logic



Metamodel for Business Logic

Syntax

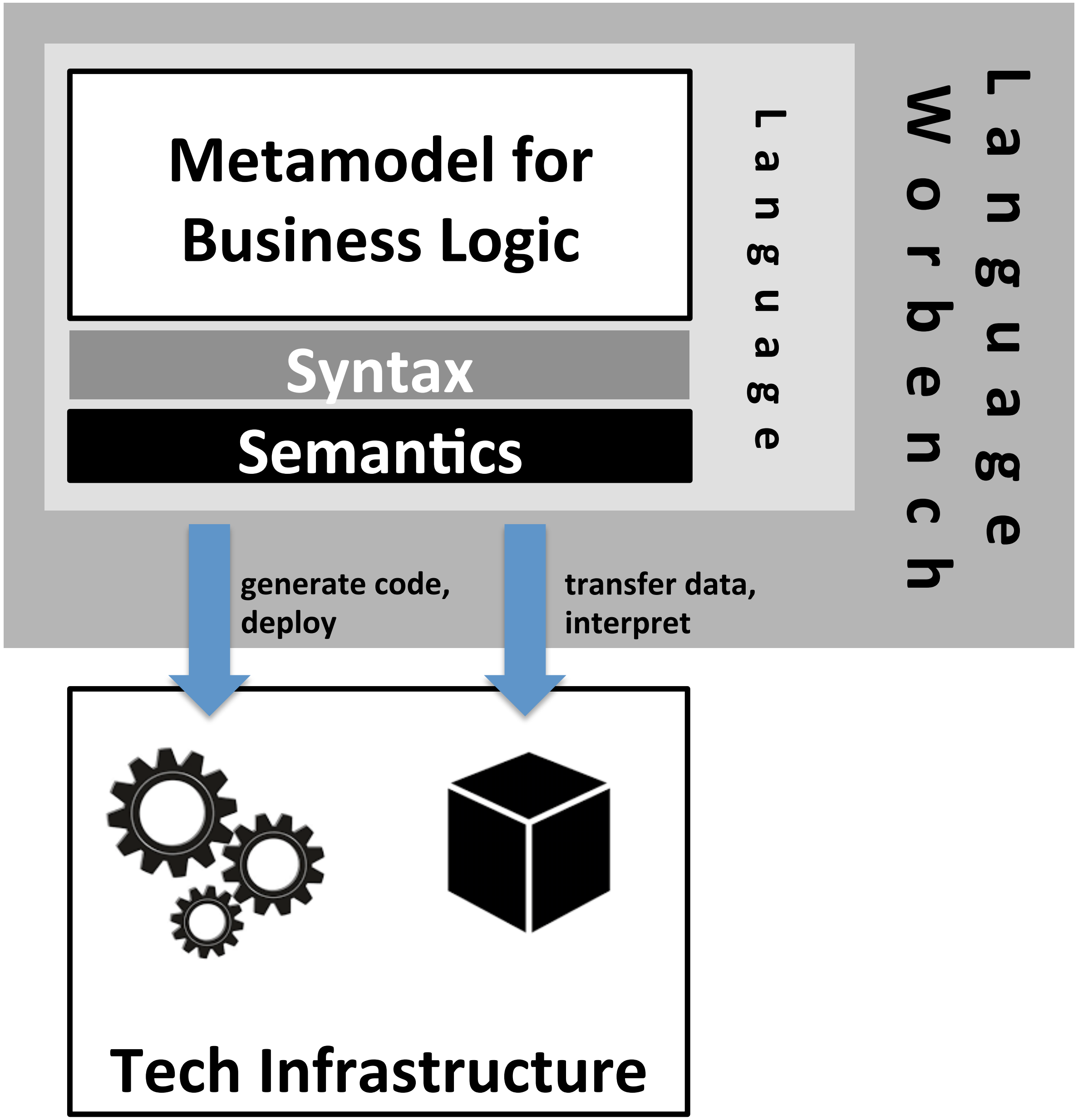
Semantics

Language

generate code, deploy

transfer data, interpret

Tech Infrastructure



Metamodel for Business Logic

Syntax

Semantics

Language

Language Workbench

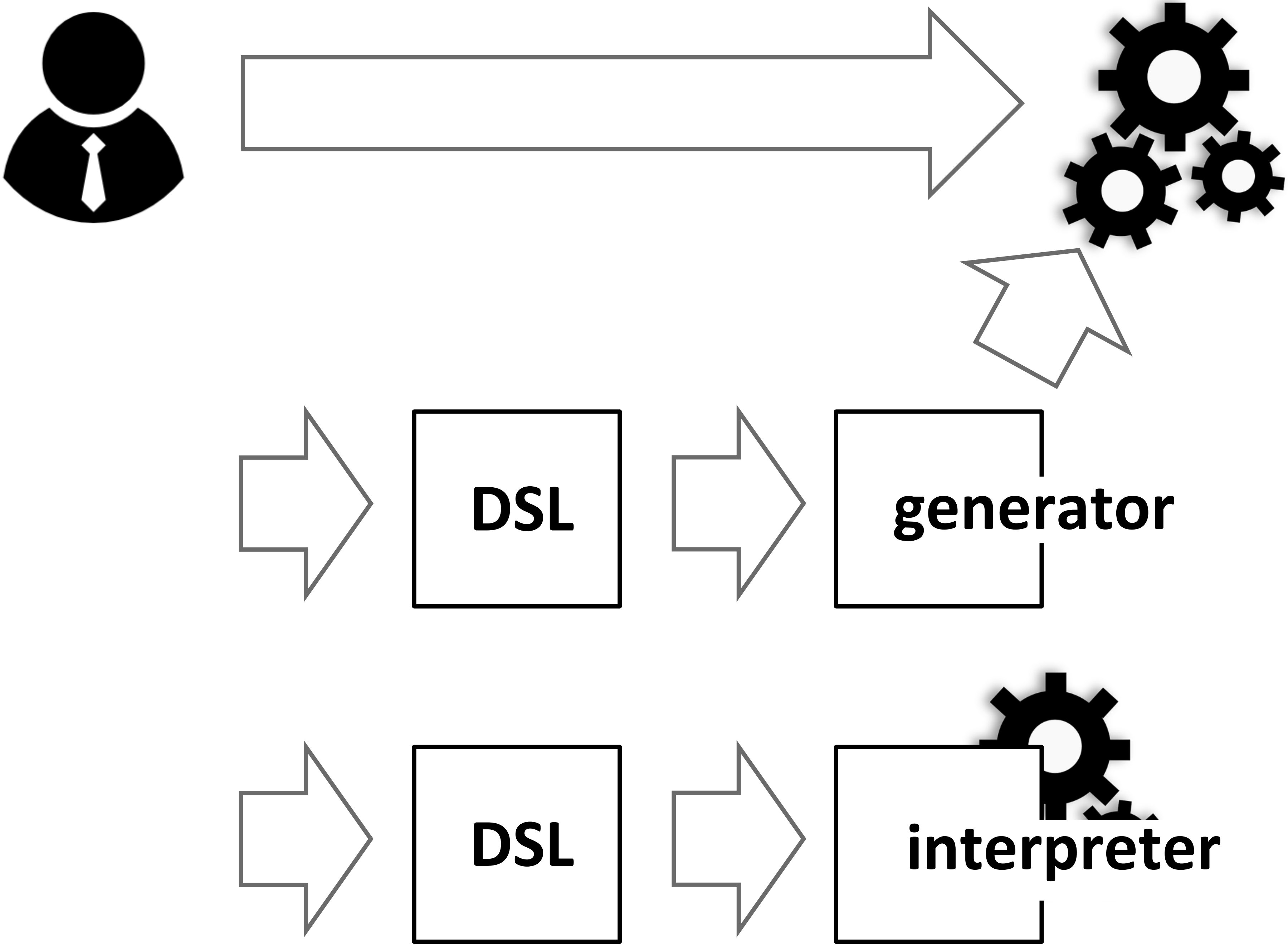
generate code, deploy

transfer data, interpret

Tech Infrastructure

3

Language Workbenches

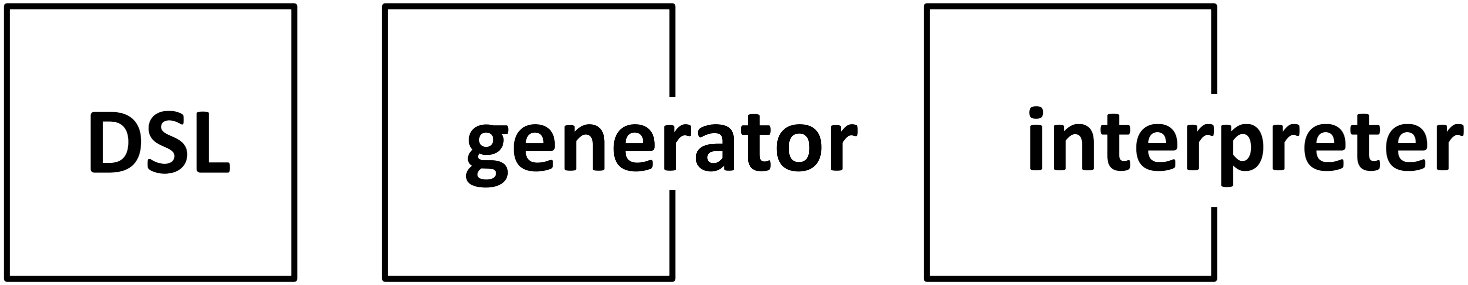


DSL

generator

DSL

interpreter



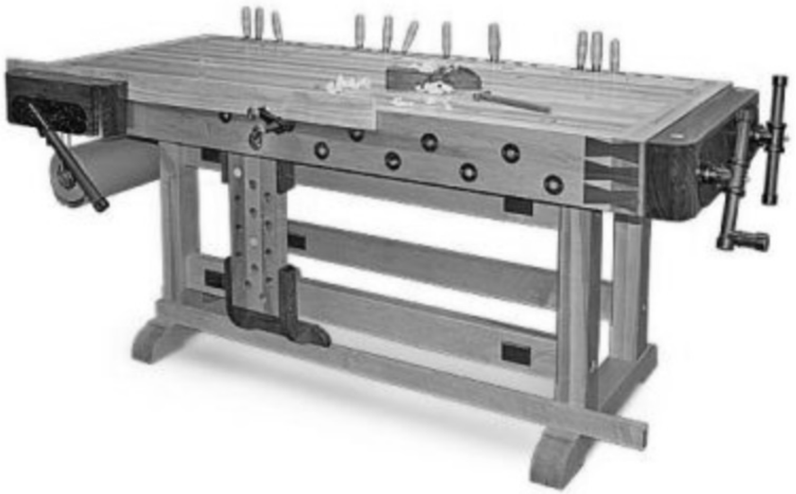
An old idea from the 1970s.

BUT…

Language Workbench

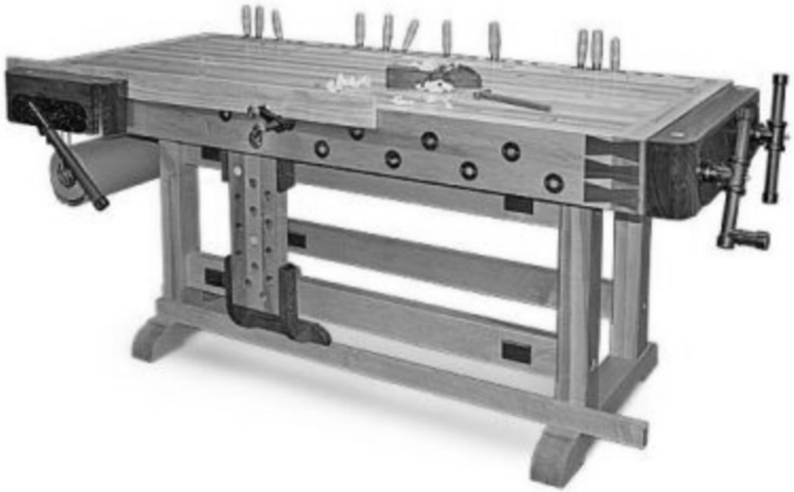
(Martin Fowler, 2004)

Freely **define** languages and **integrate** them



Language Workbench

(Martin Fowler, 2004)



powerful

+

editing

testing

refactoring

debugging

groupware

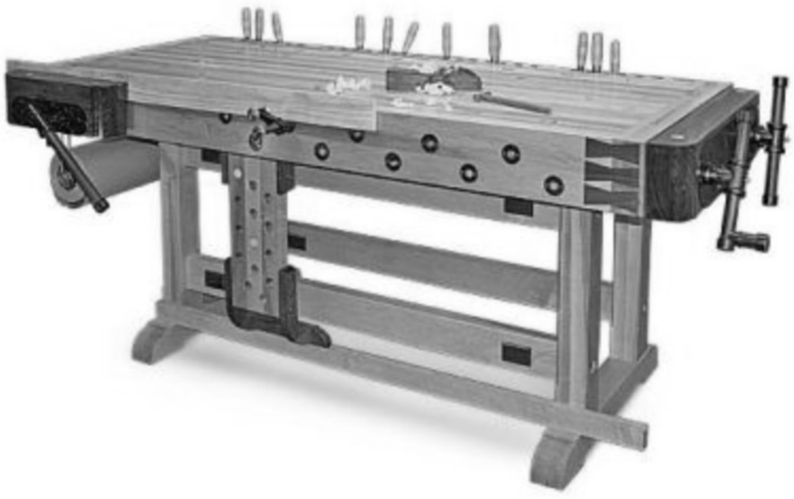
language definition

implies

IDE definition

Language Workbench

(Martin Fowler, 2004)

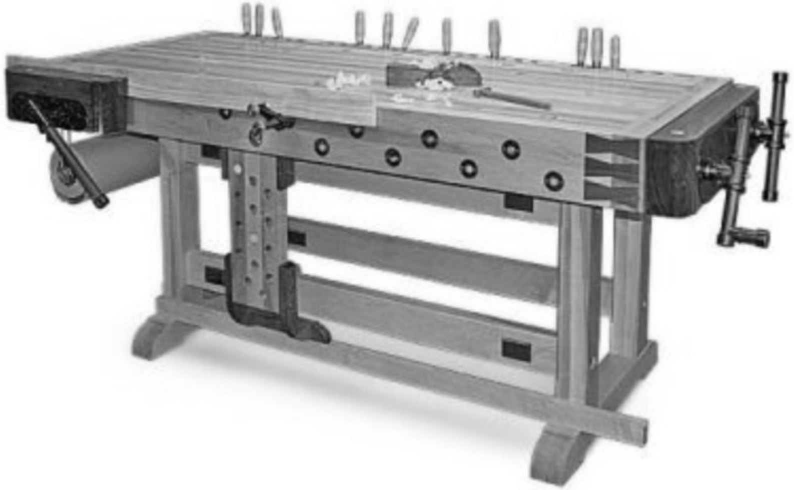


support for „classical“ programming and „classical“ modeling

There‘s no difference!

Language Workbench

(Martin Fowler, 2004)



Language Development

is

engineering

not

science!

A Language Workbench –

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

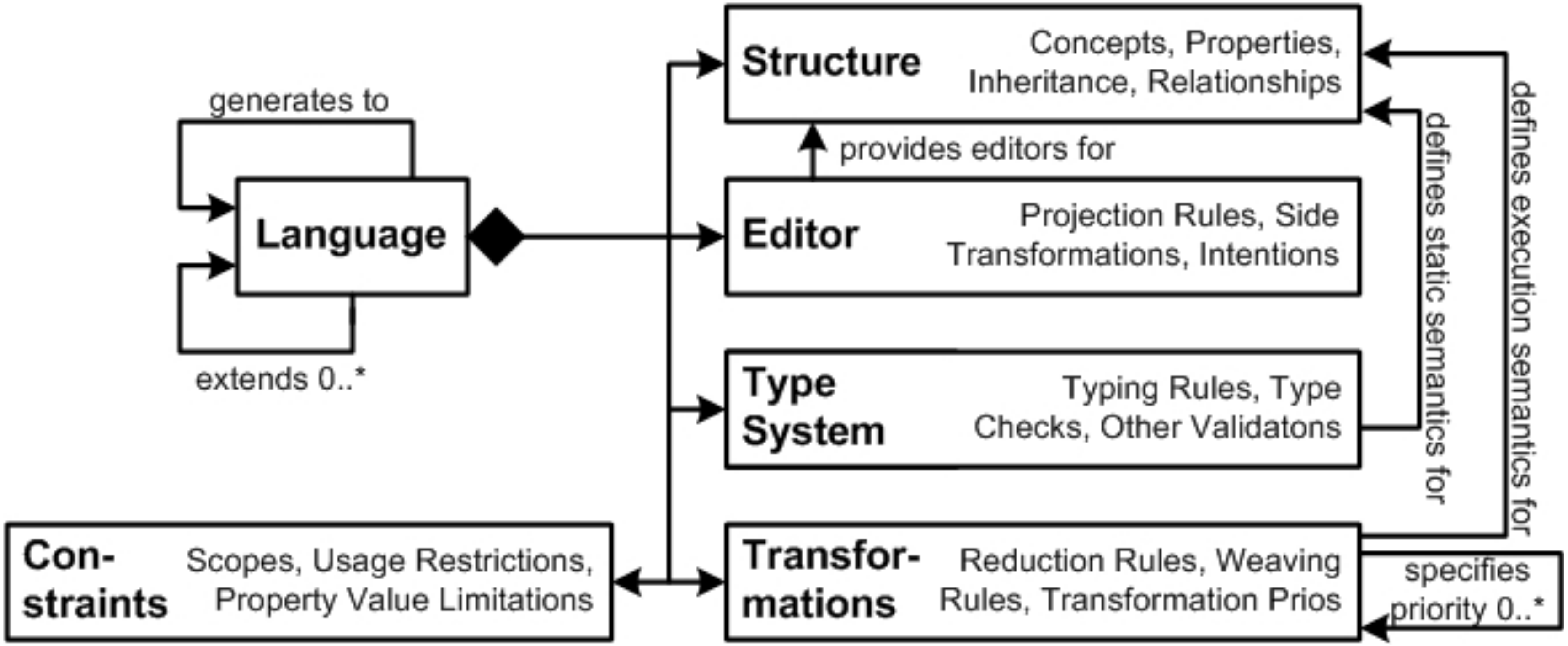
Open Source

Apache 2.0

http://jetbrains.com/mps

[Language Workbench]

Comprehensive Support for many aspects of Language Definition.



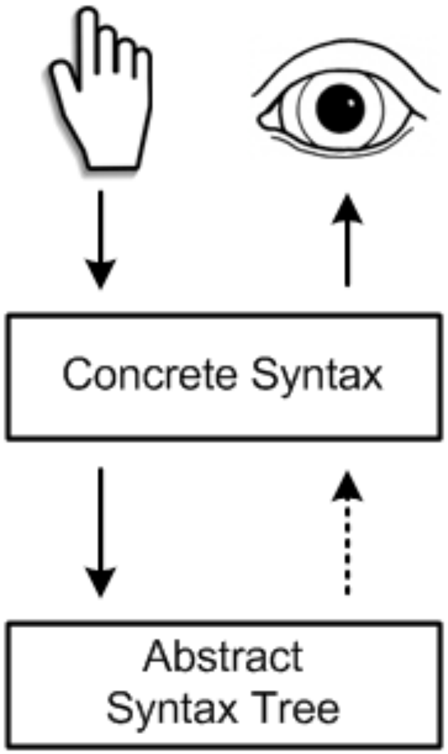
generates to
Language
extends 0..*
Structure
Concepts, Properties, Inheritance, Relationships
provides editors for
Editor
Projection Rules, Side Transformations, Intentions
Type System
Typing Rules, Type Checks, Other Validatons
Con-straints
Scopes, Usage Restrictions, Property Value Limitations
Transfor-mations
Reduction Rules, Weaving Rules, Transformation Prios
defines static semantics for
defines execution semantics for
specifies priority 0..*

+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...

[Projectional Editing]

What it is

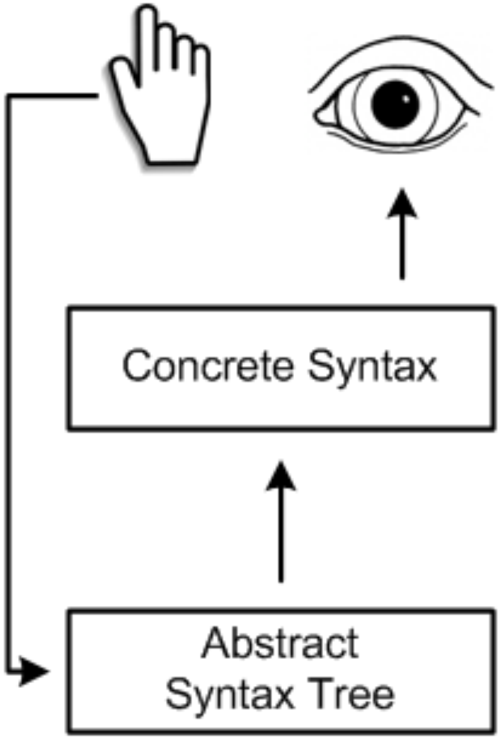
Parsing



Concrete Syntax

Abstract Syntax Tree

Projectional Editing



Concrete Syntax

Abstract Syntax Tree

[Projectional Editing]

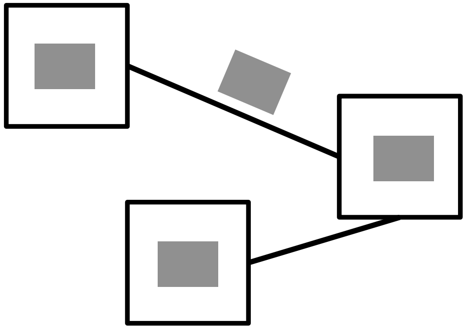
Syntactic Flexibility

Regular Code/Text

Mathematical

Tables

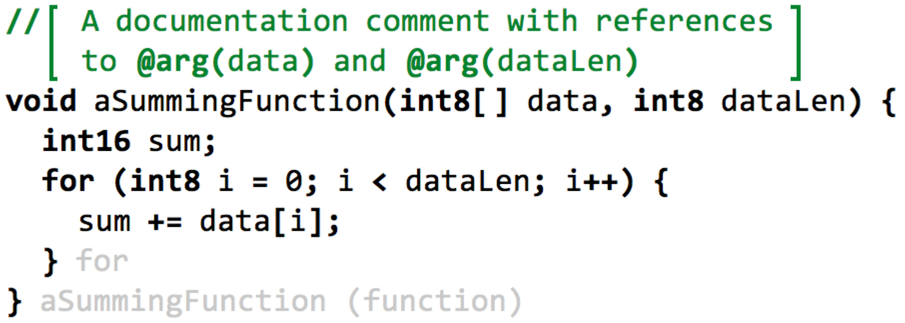
Graphical



[Projectional Editing]

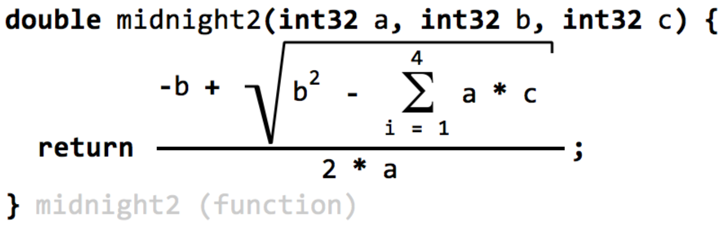
Syntactic Flexibility

Regular Code/Text



```
//[ A documentation comment with references ]
  [ to @arg(data) and @arg(dataLen)          ]
void aSummingFunction(int8[] data, int8 dataLen) {
  int16 sum;
  for (int8 i = 0; i < dataLen; i++) {
    sum += data[i];
  } for
} aSummingFunction (function)
```

Mathematical

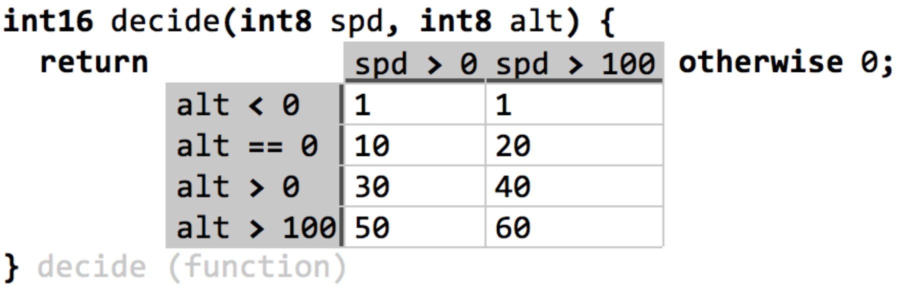


```
double midnight2(int32 a, int32 b, int32 c) {
```

$$\textbf{return}\ \frac{-b + \sqrt{b^2 - \sum_{i = 1}^{4} a * c}}{2 * a};$$

```
} midnight2 (function)
```

Tables

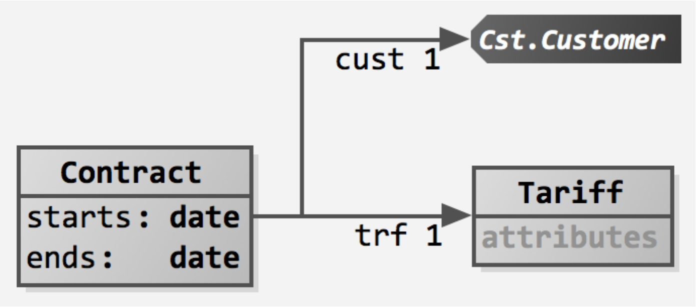


```
int16 decide(int8 spd, int8 alt) {
  return
```

	spd > 0	spd > 100
alt < 0	1	1
alt == 0	10	20
alt > 0	30	40
alt > 100	50	60

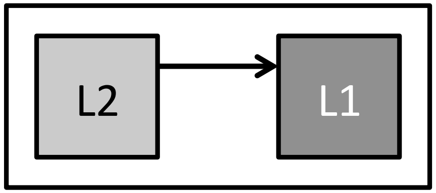
```
otherwise 0;
} decide (function)
```

Graphical



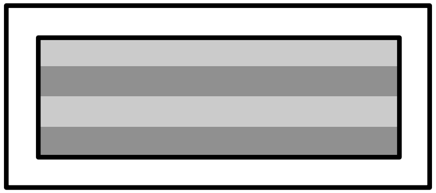
[Projectional Editing]

Language Composition



Separate Files

Type System
Transformation
Constraints



In One File

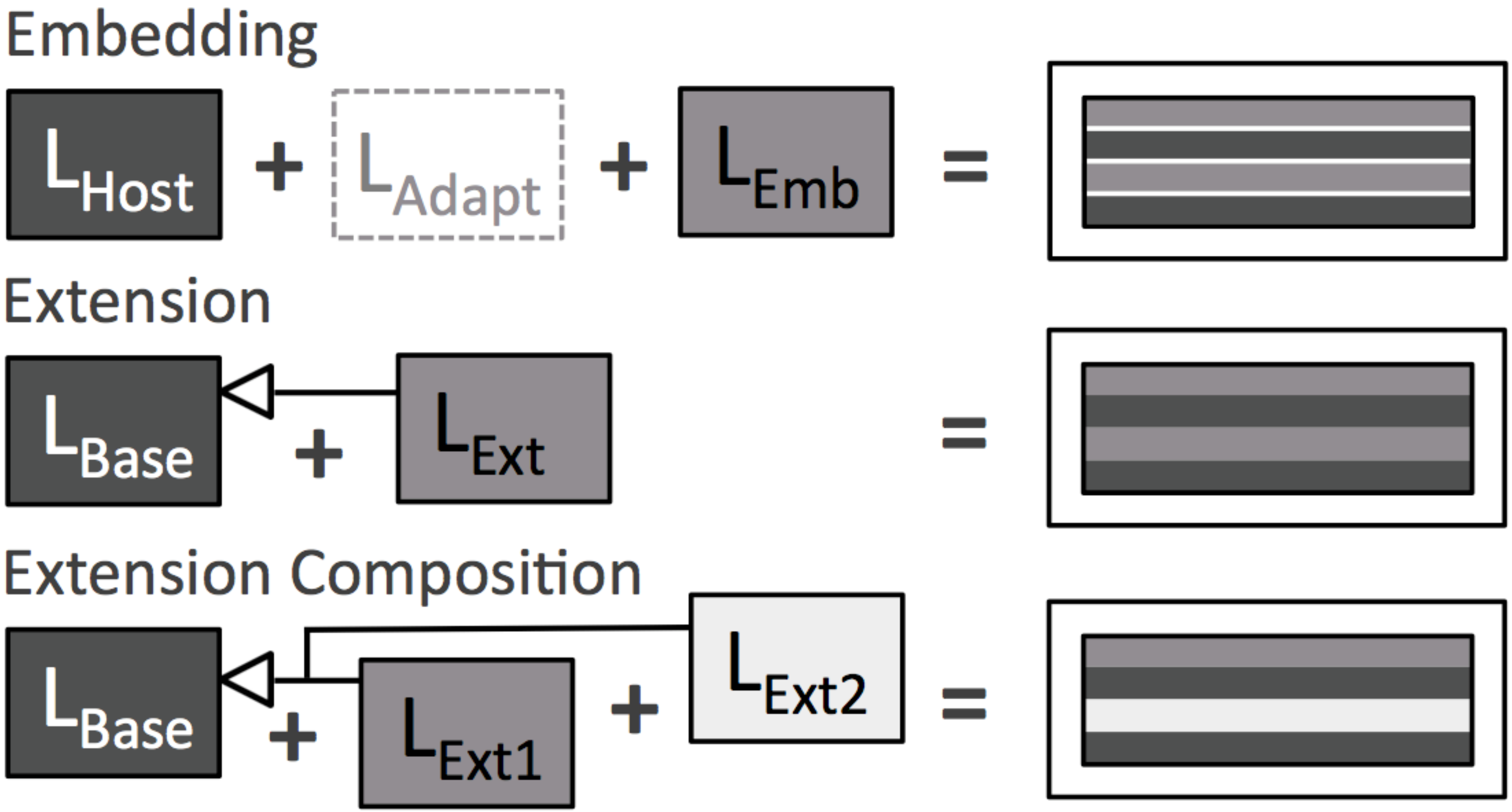
Type System
Transformation
Constraints
Syntax
IDE

mbeddr

50+ extensions to C
10+ extensions to requirements lang.

[Projectional Editing]

Language Composition



Embedding

L_{Host} + L_{Adapt} + L_{Emb} =

Extension

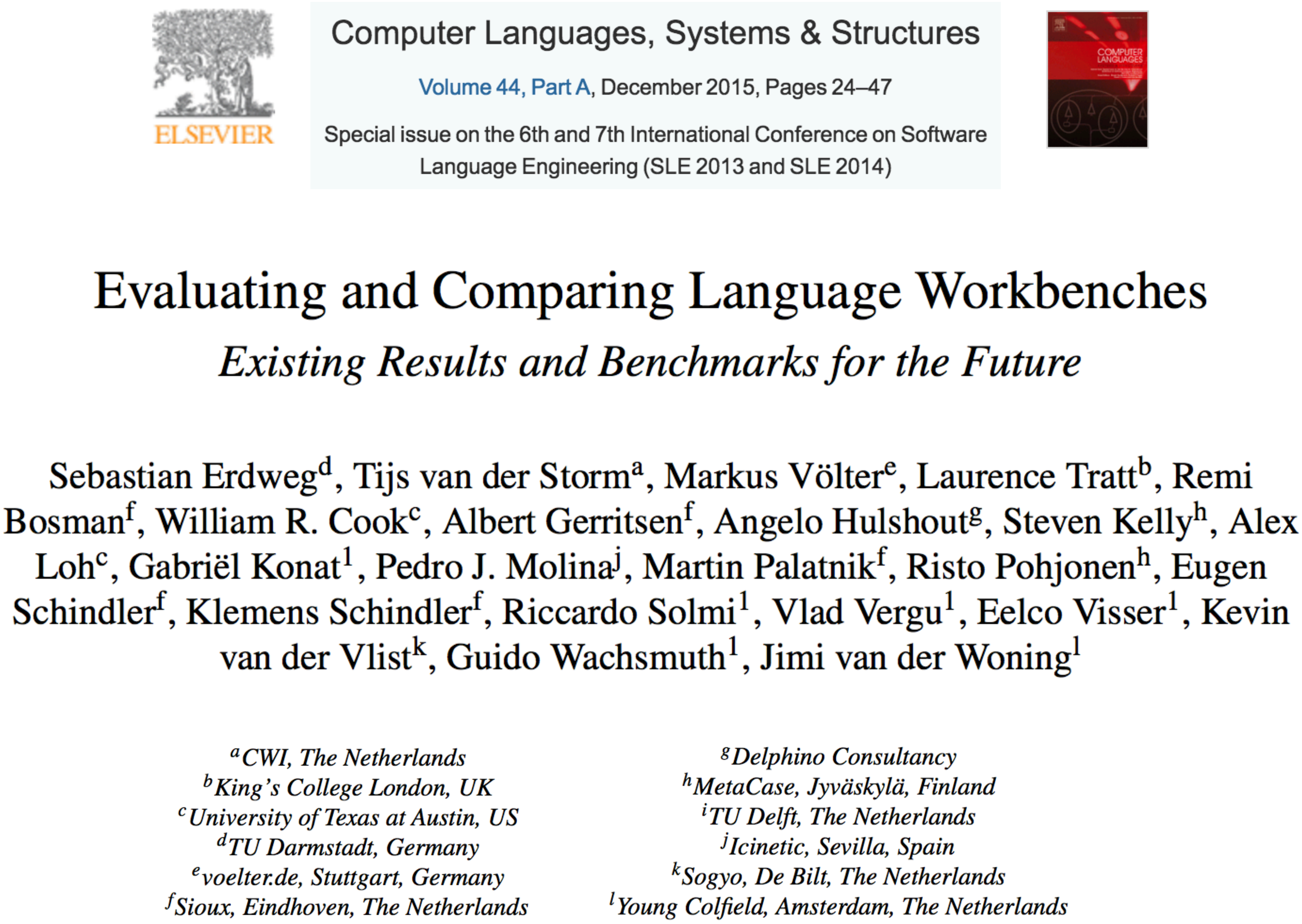
L_{Base} + L_{Ext} =

Extension Composition

L_{Base} + L_{Ext1} + L_{Ext2} =

Other Language Workbenches

{s} spoofax	TU Delft
xtext	itemis/Typefox
Rascal	CWI Amsterdam
The Whole Platform	Solmi/Persiani



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

Precision vs. Programming

Precision { Formulas, Rules
Data Structures
Tables
Values

!=

Programming

Precision

Formulas, Rules
Data Structures
Tables
Values

!=

Programming

Performance
Scalability
Robustness
Deployment

Precision {

Formulas, Rules
Data Structures
Tables
Values

!=

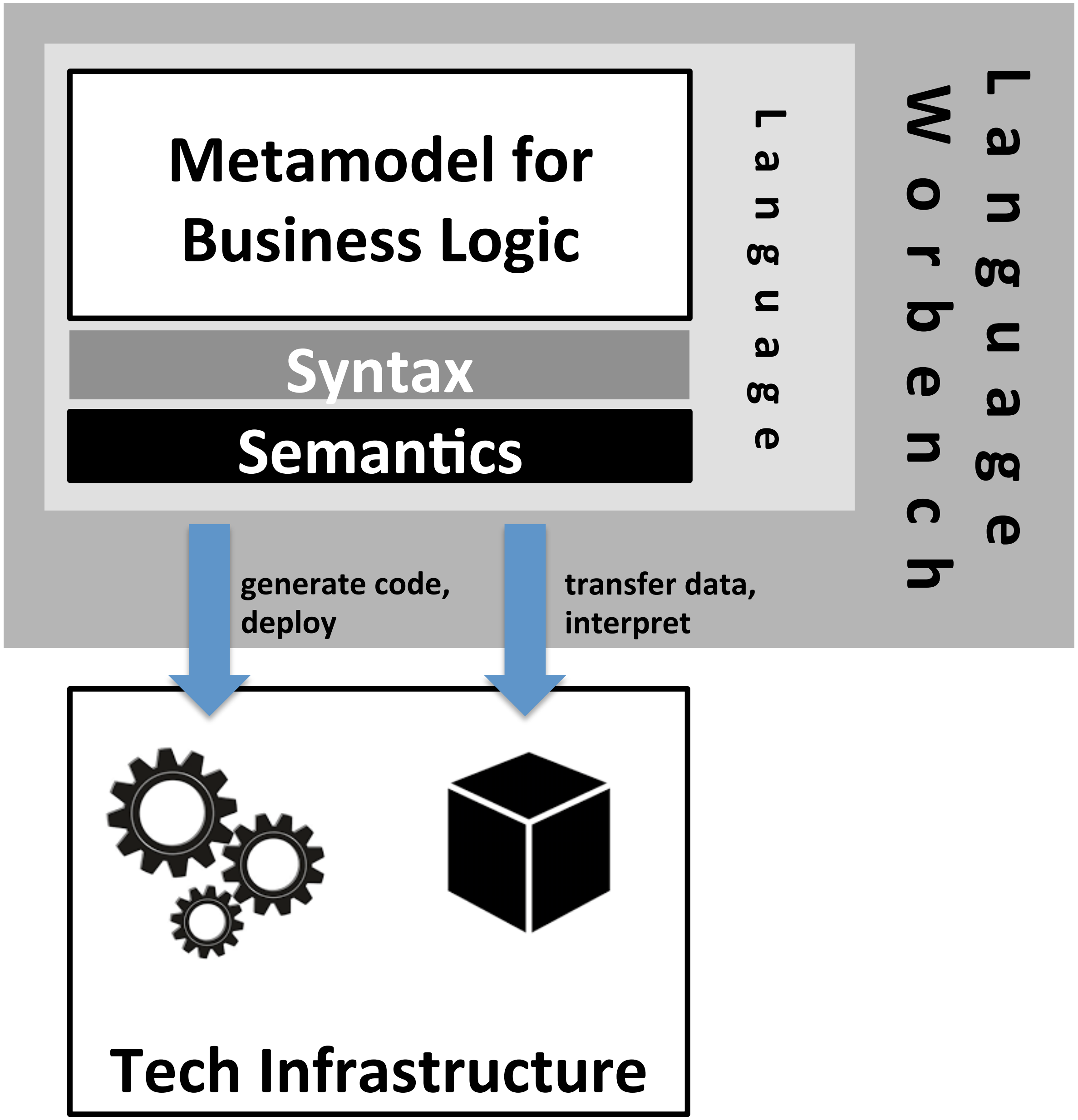
Greek Letters
Analyses
Proofs

} **Formalization**

5

Why a DSL?

Isn‘t a framework or code-populated model good enough?



Metamodel for Business Logic

Syntax

Semantics

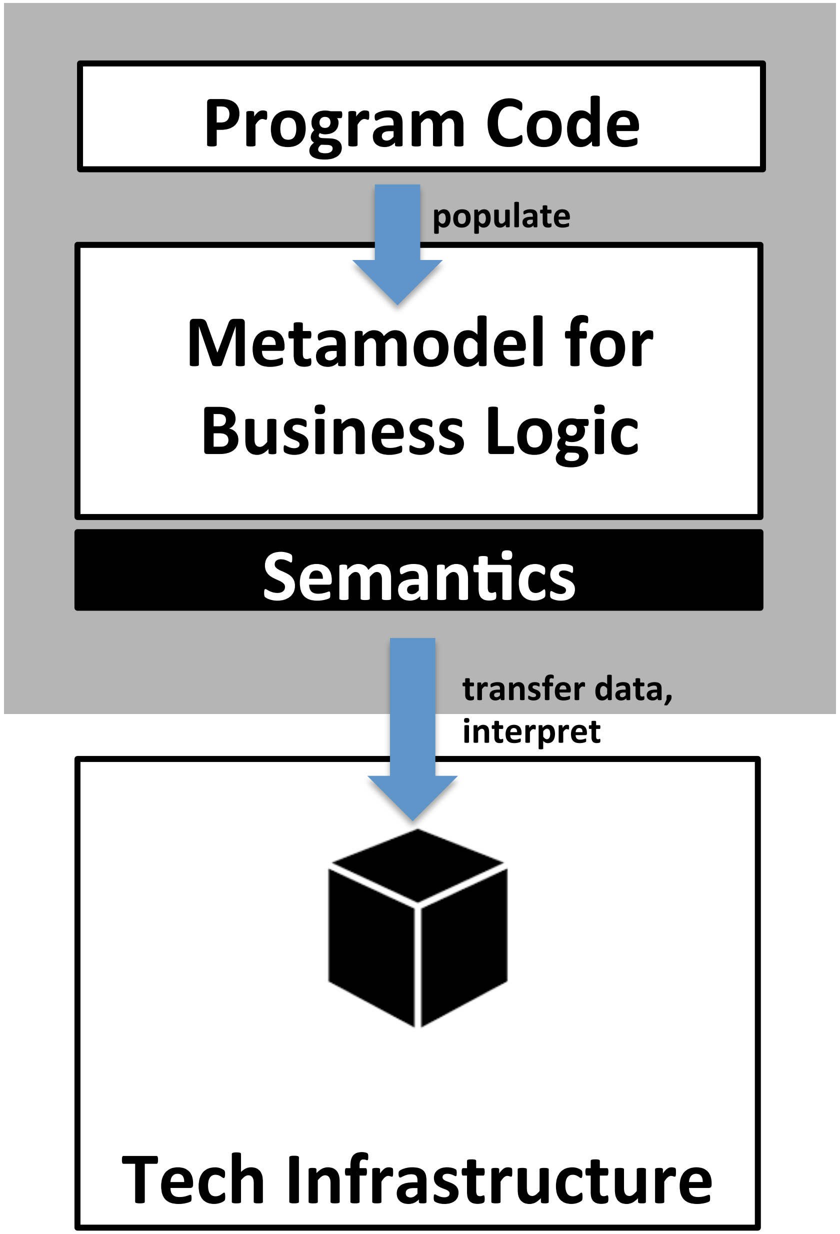
Language

Language Workbench

generate code, deploy

transfer data, interpret

Tech Infrastructure



For (fine-grained) behaviors, this is just way too tedious. Expressions!

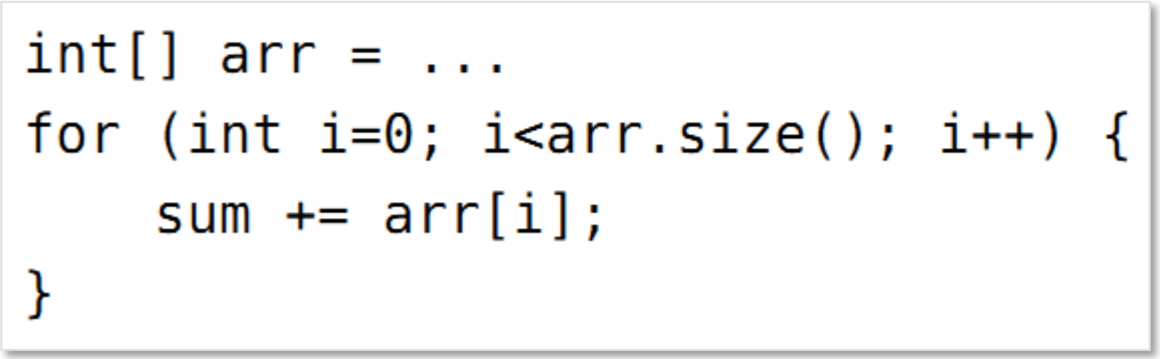
Domain-Specific Syntax lets domain experts contribute.

Static Checks provide more static assurances.

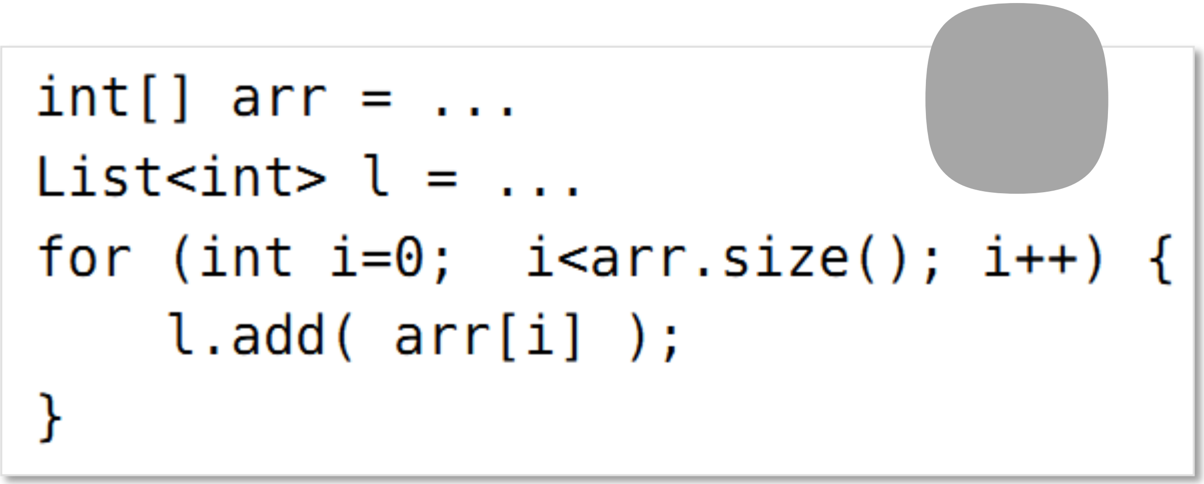
Language can evolve over time (in contrast to PL) to cover additional things first-class.

Imagine you are a Compiler

How would you parallelize these two pieces of code?



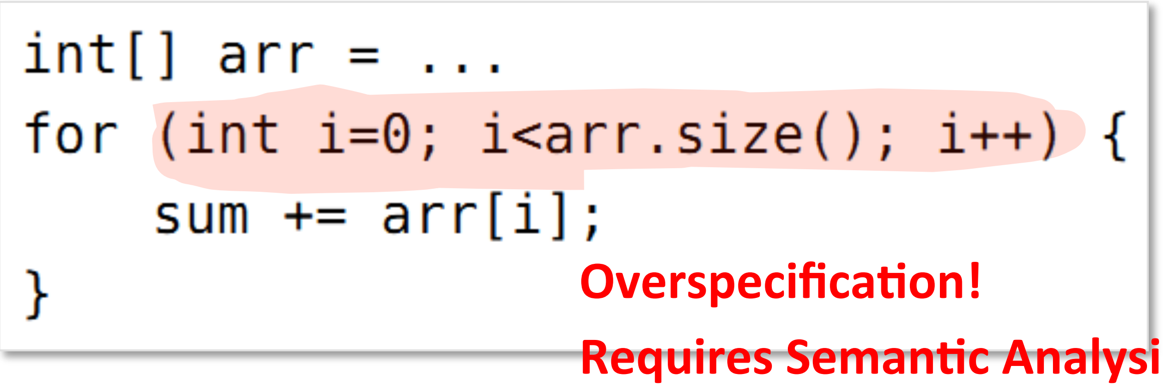
```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```



```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Imagine you are a Compiler

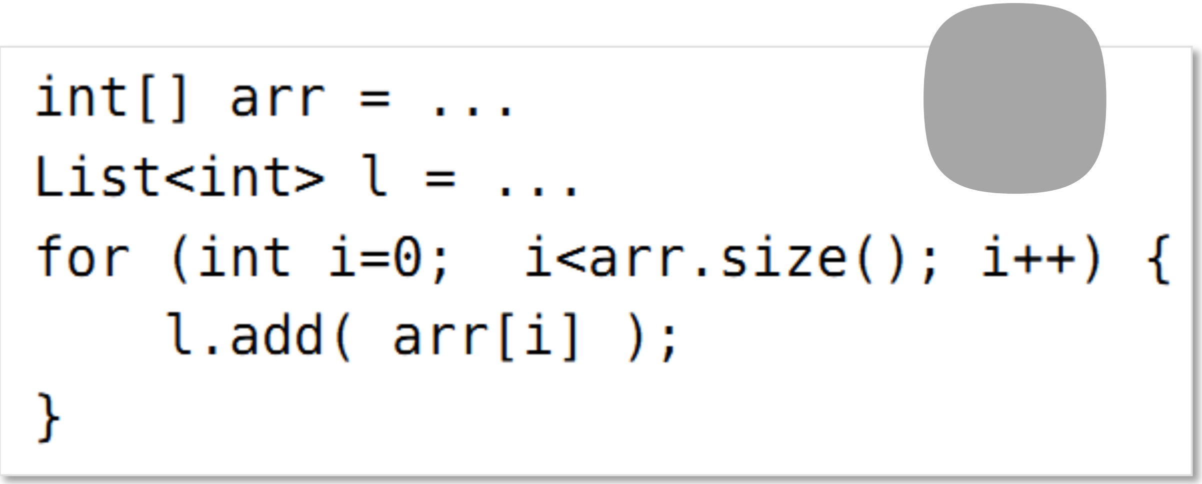
How would you parallelize these two pieces of code?



```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

Overspecification!

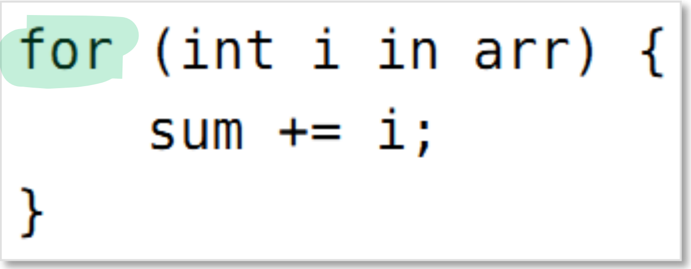
Requires Semantic Analysis!



```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Imagine you are a Compiler

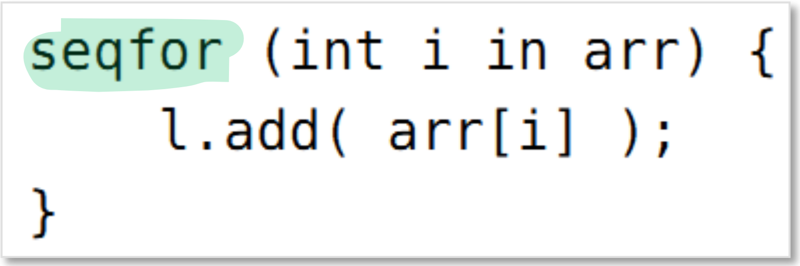
How would you parallelize these two pieces of code?



```
for (int i in arr) {
    sum += i;
}
```

First-Class Abstractions.

Directly represents Semantics.



```
seqfor (int i in arr) {
    l.add( arr[i] );
}
```

Def: Domain-Specific Language

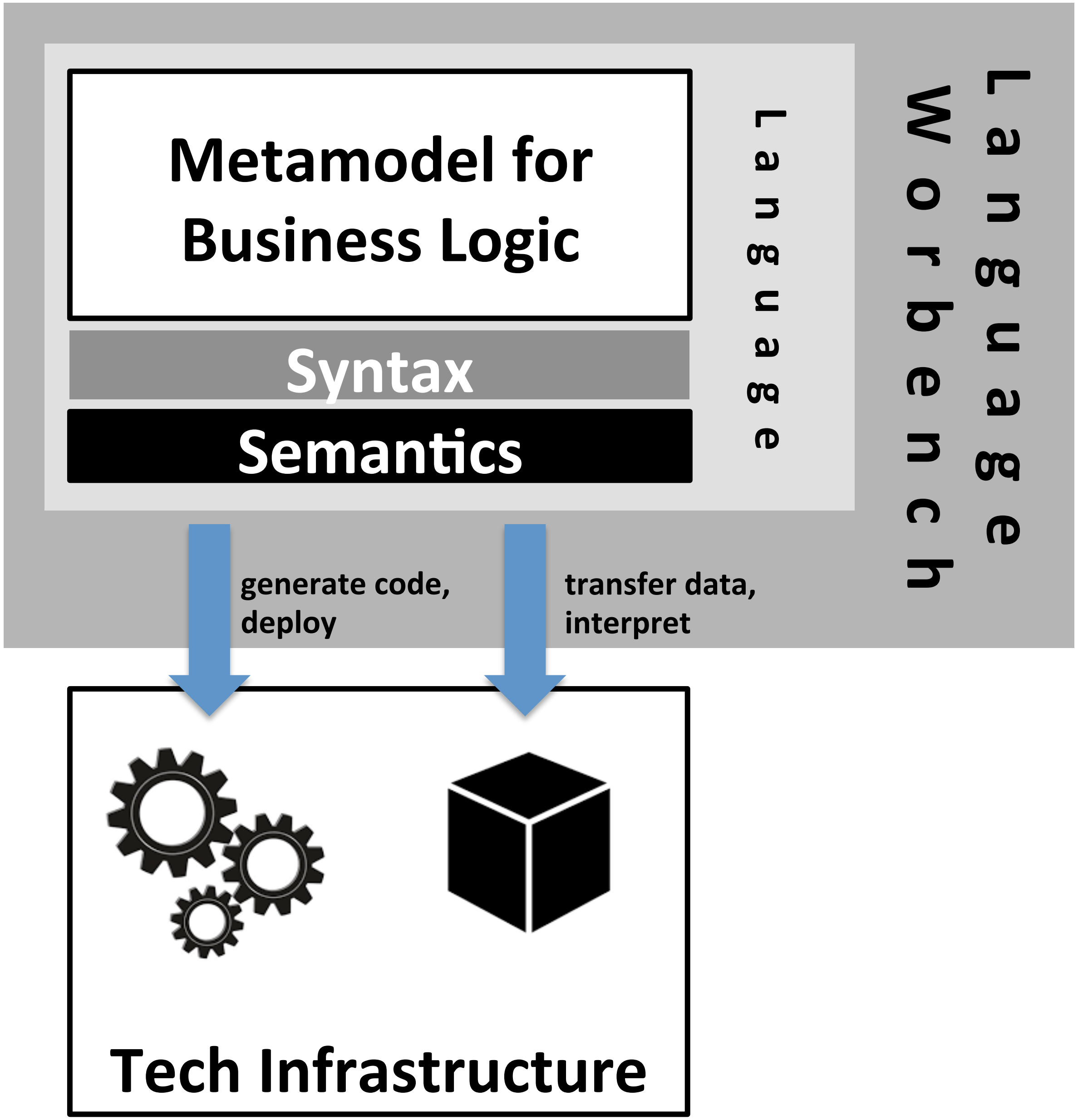
A DSL is a **language** for a domain D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D. Processing tools do not have to do “semantic recovery” on D programs.

As you understand D over time, you add **additional** first-class **abstractions** to the DSL.

6

Integration on the Platform



Metamodel for Business Logic

Syntax

Semantics

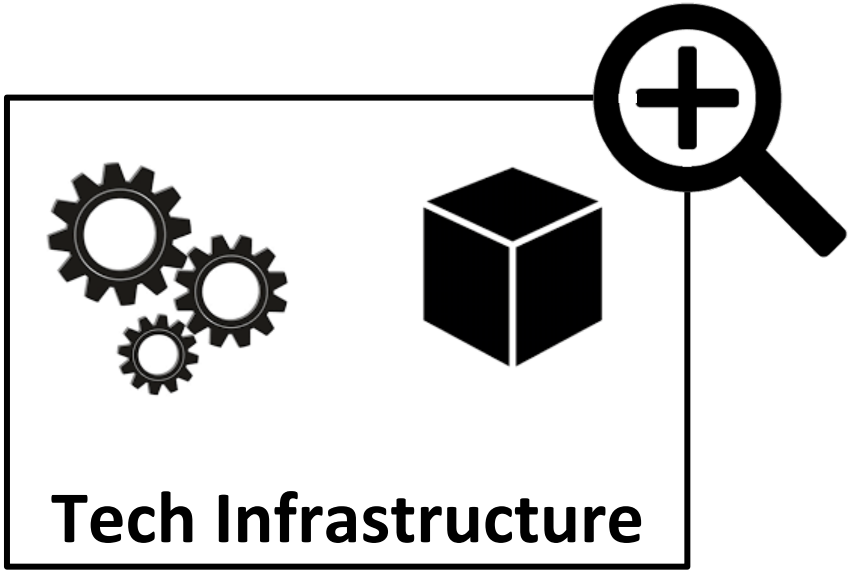
Language

Language Workbench

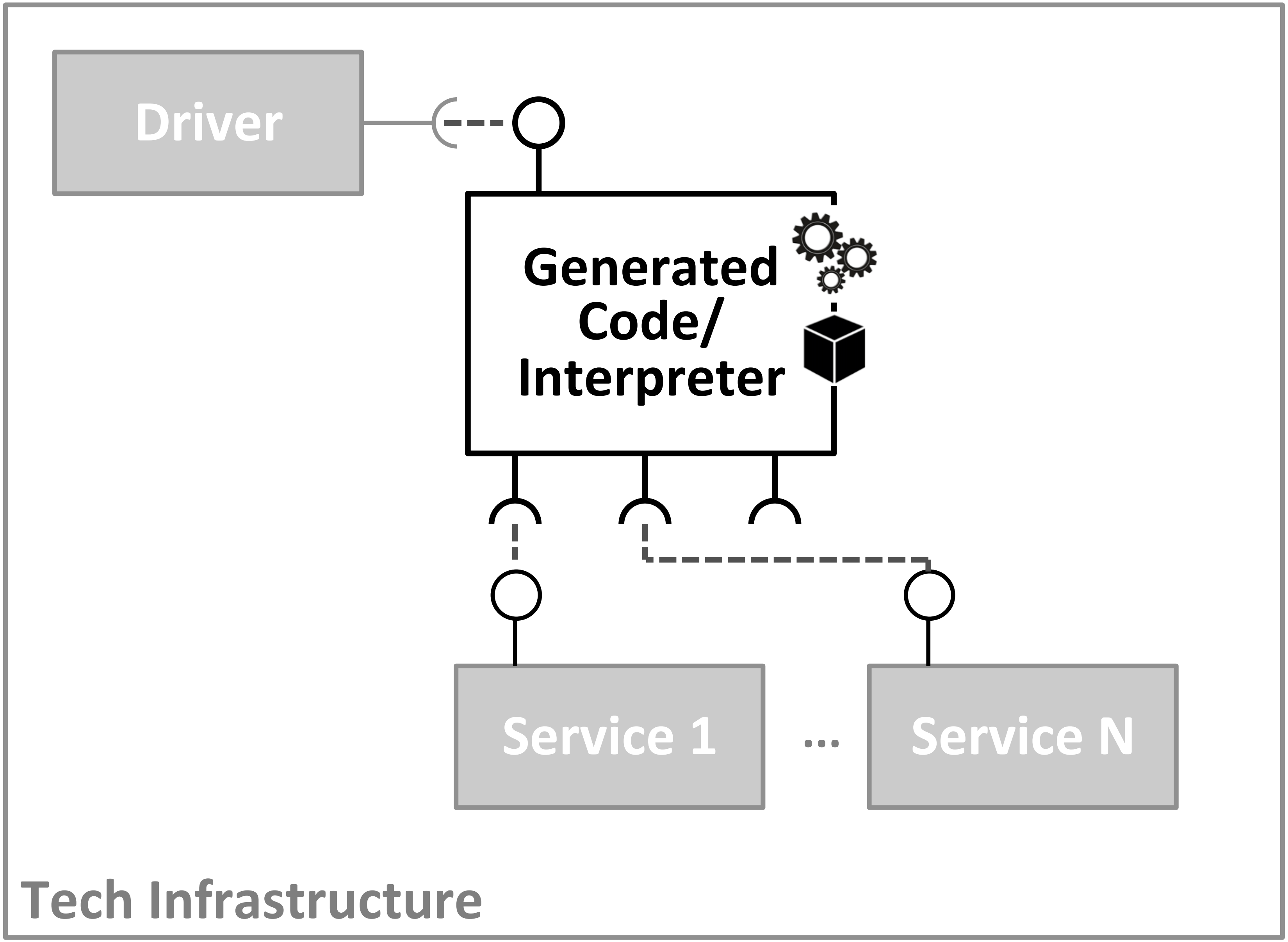
generate code, deploy

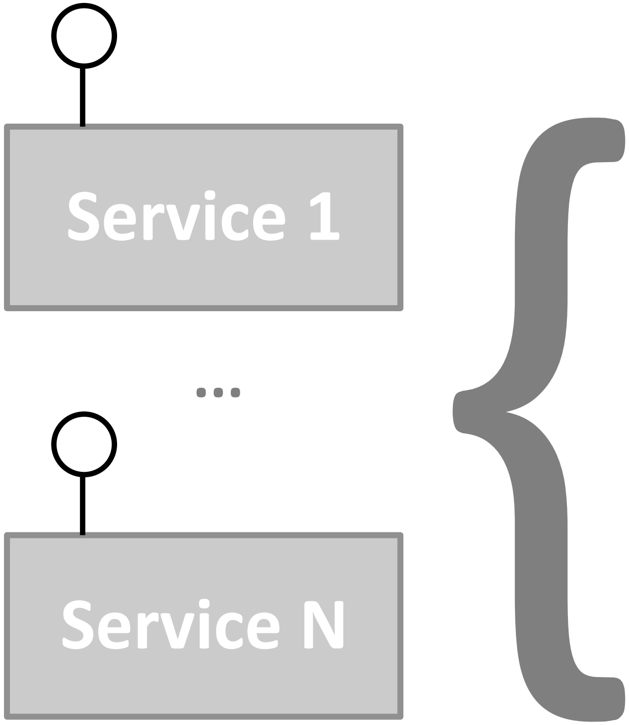
transfer data, interpret

Tech Infrastructure



Tech Infrastructure


Driver
Generated Code/ Interpreter
Service 1
...
Service N
Tech Infrastructure



Service 1

...

Service N

Persistence/Database
Sensors
Transactions
Permissions

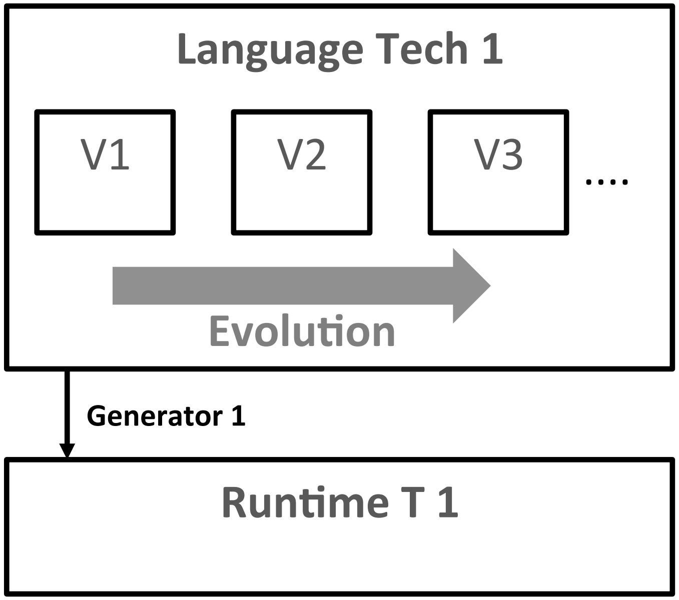
... typical technical services, also found in app servers etc.

7

Evolution

Today‘s software is tomorrow‘s legacy system.

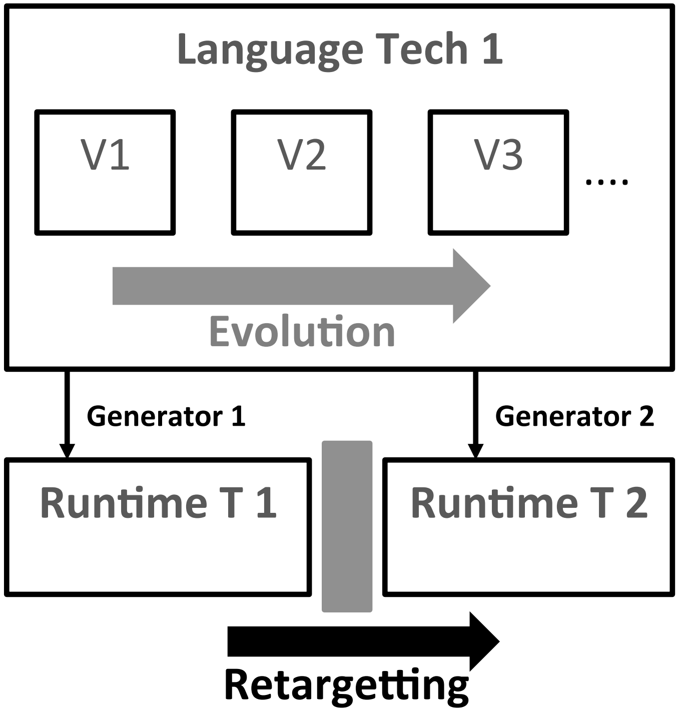
Or is it?



Existing models become incompatble with new language

⇒ **Language Versions**

Migration Scripts

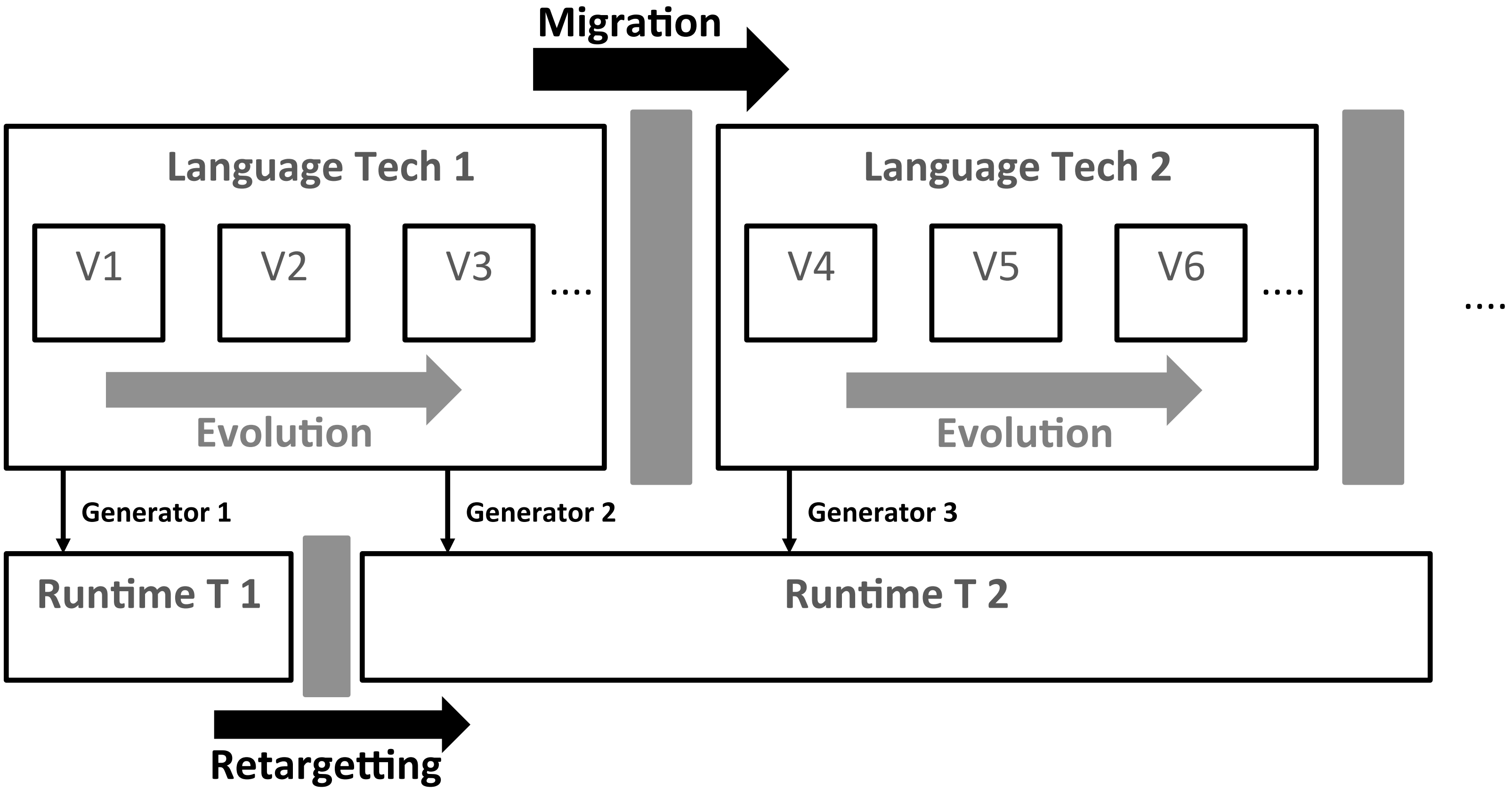


Runtime Tech outdated, uncool or slow

⇒ **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.

8

Some Lessons Learned

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.

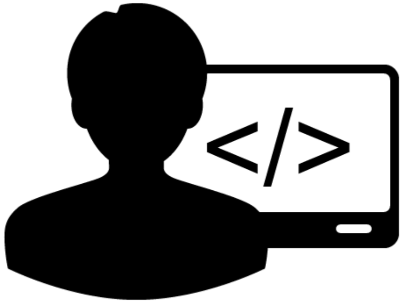
Can I find the people to do this?

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.

THREAT

Business L
vs. Programming L

		
Structure/Guid.	+	-
Notation	Mixed	Text
Views	*	1
IDE/Tool	Clean	Powerful
Learn/Effective	L	E

Structure/Guid.

Notation

Views

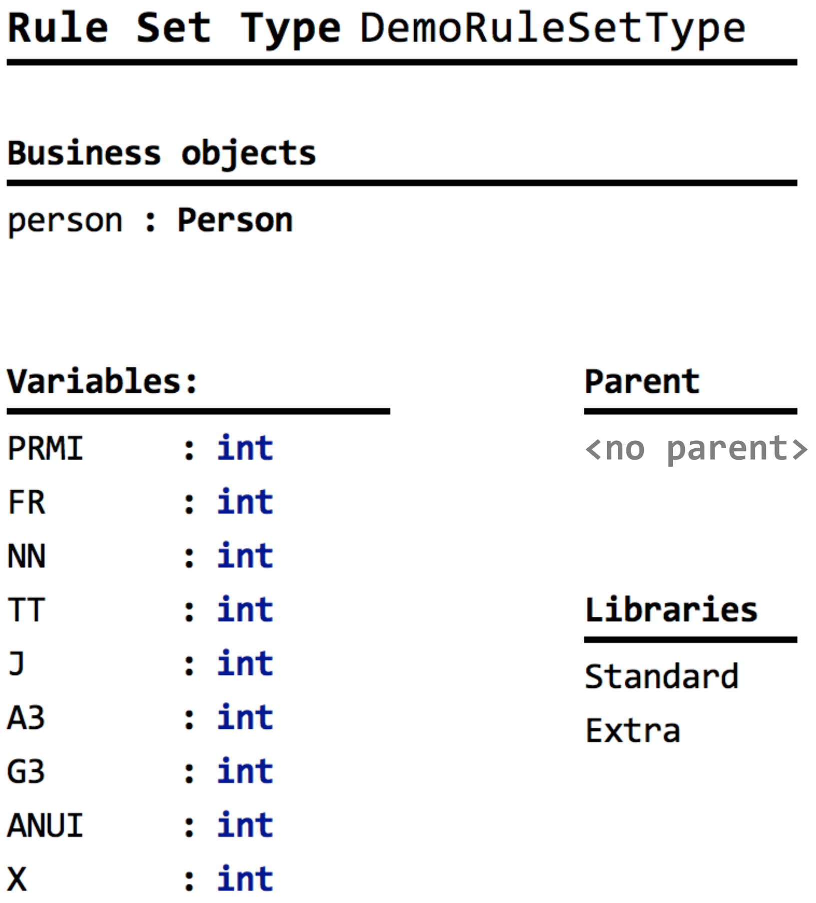
IDE/Tool

Learn/Effective

Business oriented languages are very different from what we have learned about languages for developers. LWBs let you build such languages.

Examples

Rigid Structures



Rule Set Type DemoRuleSetType

Business objects

person : **Person**

Variables:

PRMI	:	int
FR	:	int
NN	:	int
TT	:	int
J	:	int
A3	:	int
G3	:	int
ANUI	:	int
X	:	int

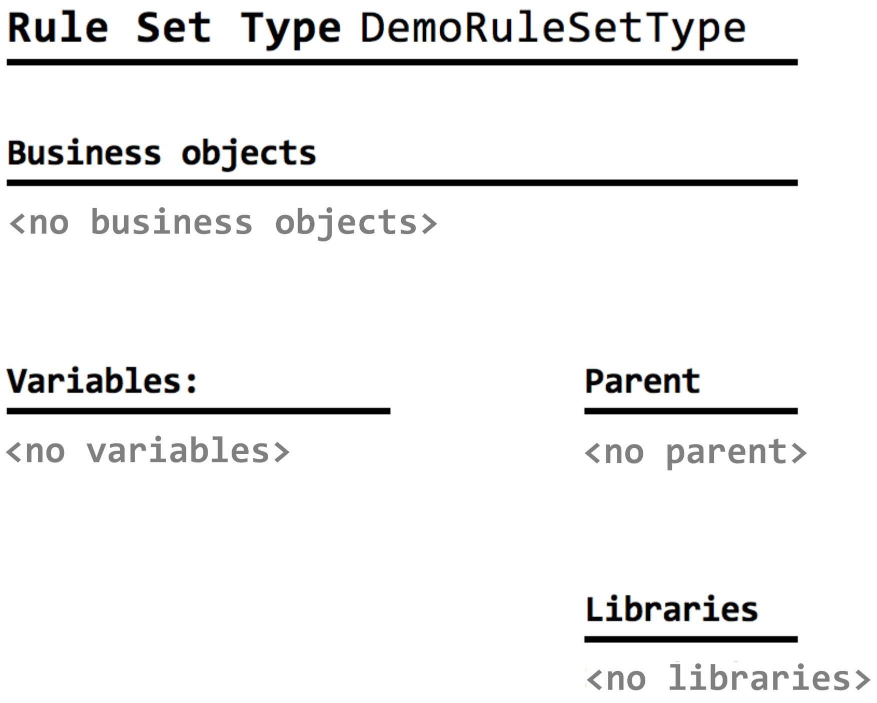
Parent

<no parent>

Libraries

Standard

Extra



Rule Set Type DemoRuleSetType

Business objects

<no business objects>

Variables:

<no variables>

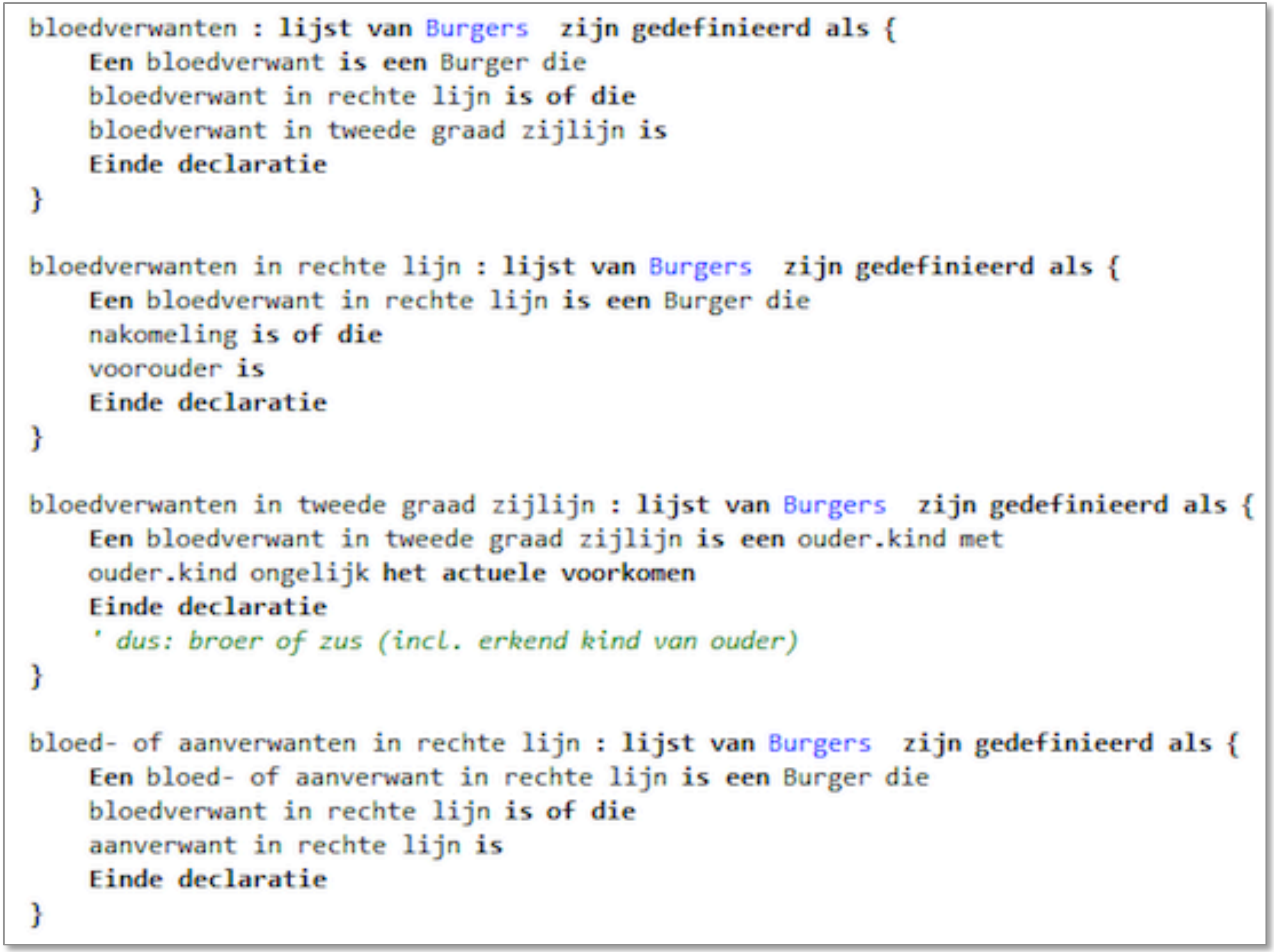
Parent

<no parent>

Libraries

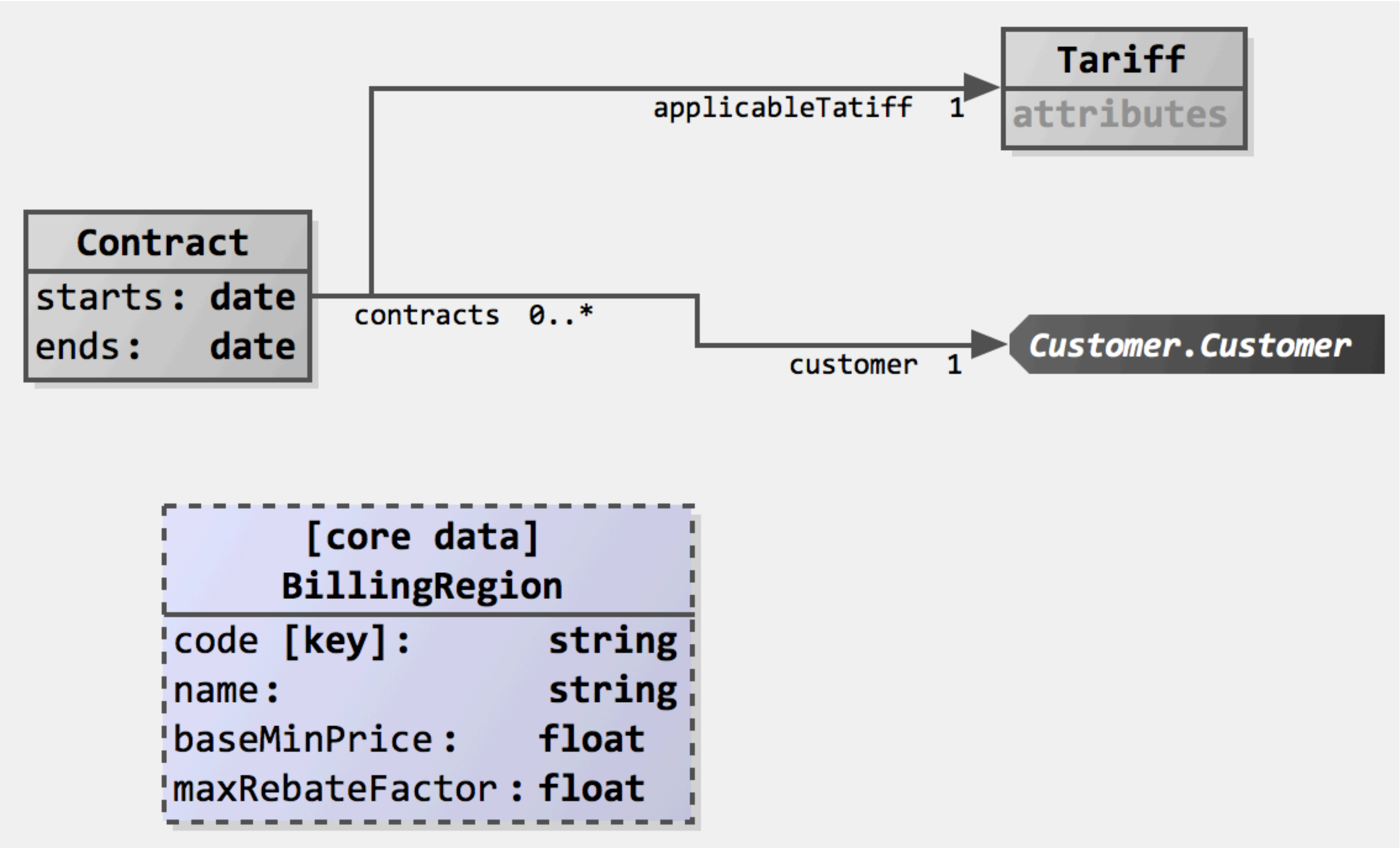
<no libraries>

Prose-Like Language for Calc Rules

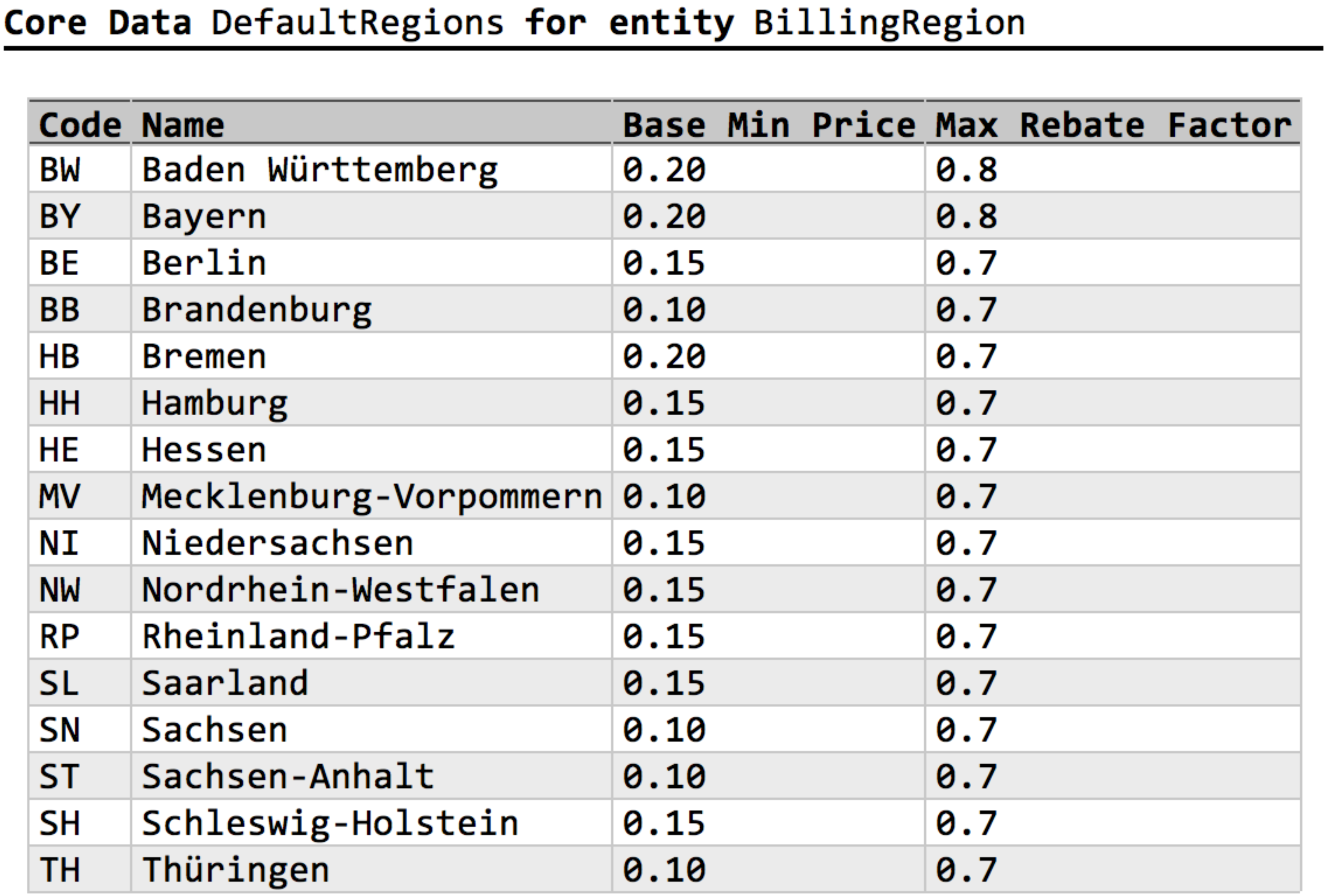


```
bloedverwanten : lijst van Burgers  zijn gedefinieerd als {
    Een bloedverwant is een Burger die
    bloedverwant in rechte lijn is of die
    bloedverwant in tweede graad zijlijn is
    Einde declaratie
}

bloedverwanten in rechte lijn : lijst van Burgers  zijn gedefinieerd als {
    Een bloedverwant in rechte lijn is een Burger die
    nakomeling is of die
    voorouder is
    Einde declaratie
}

bloedverwanten in tweede graad zijlijn : lijst van Burgers  zijn gedefinieerd als {
    Een bloedverwant in tweede graad zijlijn is een ouder.kind met
    ouder.kind ongelijk het actuele voorkomen
    Einde declaratie
    ' dus: broer of zus (incl. erkend kind van ouder)
}

bloed- of aanverwanten in rechte lijn : lijst van Burgers  zijn gedefinieerd als {
    Een bloed- of aanverwant in rechte lijn is een Burger die
    bloedverwant in rechte lijn is of die
    aanverwant in rechte lijn is
    Einde declaratie
}
```

Diagrams for Data Modeling



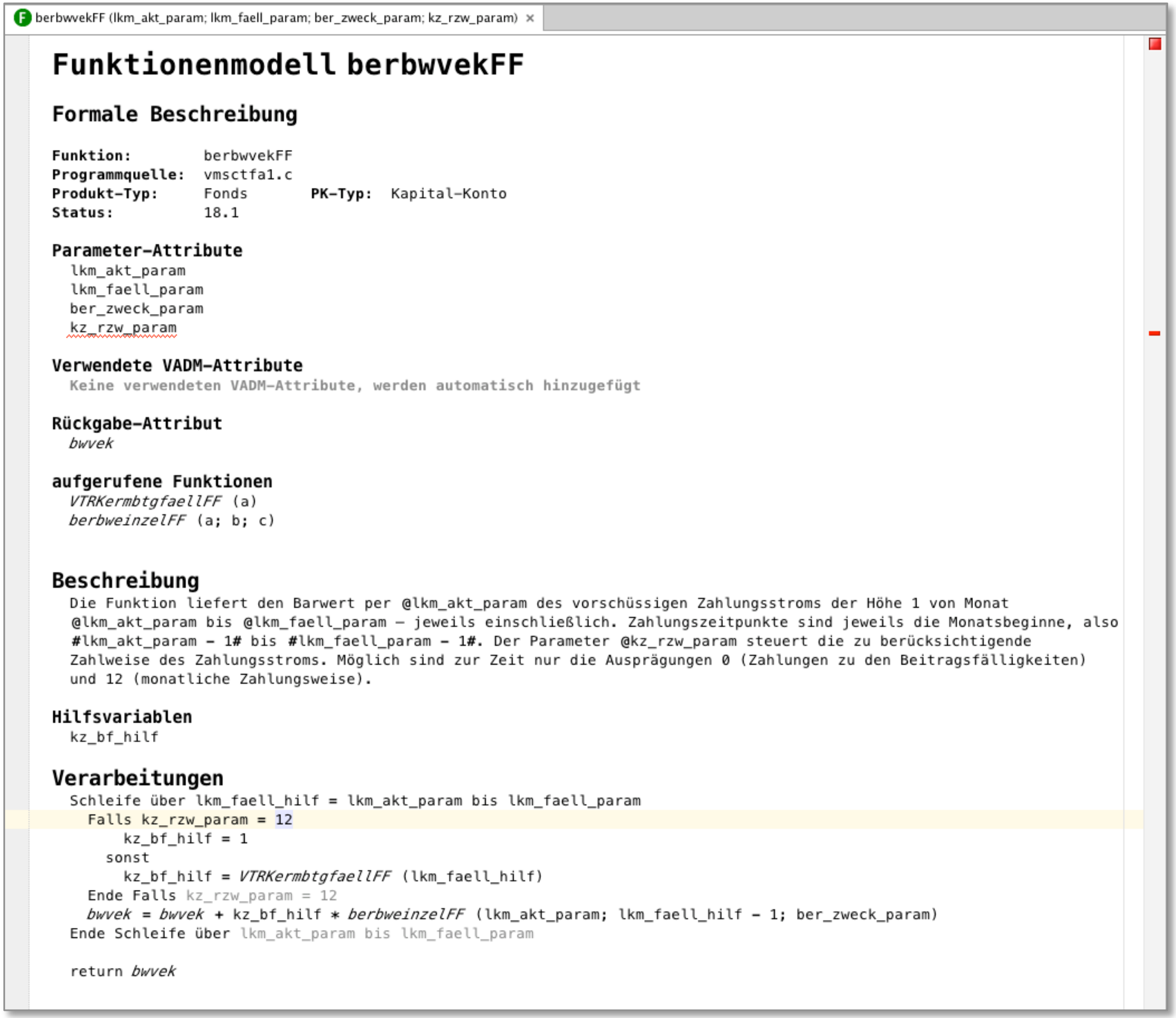
Tables for Reference Data



Core Data DefaultRegions **for entity** BillingRegion

Code	Name	Base Min Price	Max Rebate Factor
BW	Baden Württemberg	0.20	0.8
BY	Bayern	0.20	0.8
BE	Berlin	0.15	0.7
BB	Brandenburg	0.10	0.7
HB	Bremen	0.20	0.7
HH	Hamburg	0.15	0.7
HE	Hessen	0.15	0.7
MV	Mecklenburg-Vorpommern	0.10	0.7
NI	Niedersachsen	0.15	0.7
NW	Nordrhein-Westfalen	0.15	0.7
RP	Rheinland-Pfalz	0.15	0.7
SL	Saarland	0.15	0.7
SN	Sachsen	0.10	0.7
ST	Sachsen-Anhalt	0.10	0.7
SH	Schleswig-Holstein	0.15	0.7
TH	Thüringen	0.10	0.7

Insurance Specifications


berbwvekFF (lkm_akt_param; lkm_faell_param; ber_zweck_param; kz_rzw_param)

Funktionenmodell berbwvekFF

Formale Beschreibung

```
Funktion:        berbwvekFF
Programmquelle:  vmsctfa1.c
Produkt-Typ:     Fonds        PK-Typ:  Kapital-Konto
Status:          18.1
```

Parameter-Attribute

```
  lkm_akt_param
  lkm_faell_param
  ber_zweck_param
  kz_rzw_param
```

Verwendete VADM-Attribute

Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut

bwvek

aufgerufene Funktionen

```
  VTRKermbtgfaellFF (a)
  berbweinzelFF (a; b; c)
```

Beschreibung

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).

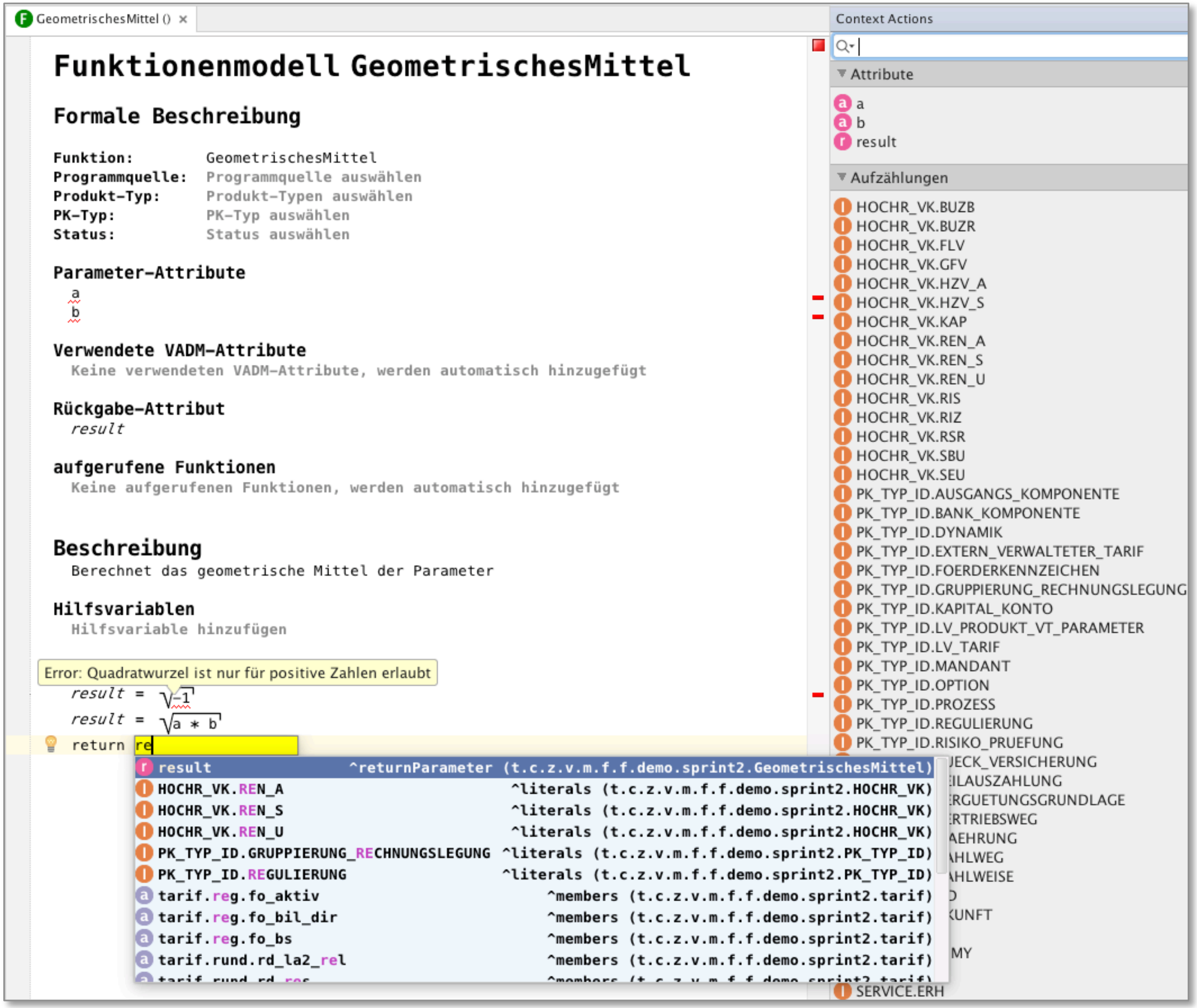
Hilfsvariablen

kz_bf_hilf

Verarbeitungen

```
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 über lkm_akt_param bis lkm_faell_param

return bwvek
```

Insurance Specifications



Funktionenmodell GeometrischesMittel

Formale Beschreibung

Funktion: GeometrischesMittel
Programmquelle: Programmquelle auswählen
Produkt-Typ: Produkt-Typen auswählen
PK-Typ: PK-Typ auswählen
Status: Status auswählen

Parameter-Attribute
a
b

Verwendete VADM-Attribute
Keine verwendeten VADM-Attribute, werden automatisch hinzugefügt

Rückgabe-Attribut
result

aufgerufene Funktionen
Keine aufgerufenen Funktionen, werden automatisch hinzugefügt

Beschreibung
Berechnet das geometrische Mittel der Parameter

Hilfsvariablen
Hilfsvariable hinzufügen

Error: Quadratwurzel ist nur für positive Zahlen erlaubt

result = $\sqrt{-1}$
result = $\sqrt{a * b}$
return re

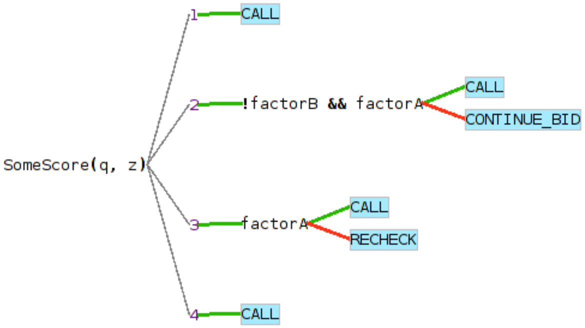
```
result                                 ^returnParameter (t.c.z.v.m.f.f.demo.sprint2.GeometrischesMittel)
HOCHR_VK.REN_A                                  ^literals (t.c.z.v.m.f.f.demo.sprint2.HOCHR_VK)
HOCHR_VK.REN_S                                  ^literals (t.c.z.v.m.f.f.demo.sprint2.HOCHR_VK)
HOCHR_VK.REN_U                                  ^literals (t.c.z.v.m.f.f.demo.sprint2.HOCHR_VK)
PK_TYP_ID.GRUPPIERUNG_RECHNUNGSLEGUNG ^literals (t.c.z.v.m.f.f.demo.sprint2.PK_TYP_ID)
PK_TYP_ID.REGULIERUNG                 ^literals (t.c.z.v.m.f.f.demo.sprint2.PK_TYP_ID)
tarif.reg.fo_aktiv                                 ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
tarif.reg.fo_bil_dir                               ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
tarif.reg.fo_bs                                    ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
tarif.rund.rd_la2_rel                              ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
tarif.rund.rd_res                                  ^members (t.c.z.v.m.f.f.demo.sprint2.tarif)
```

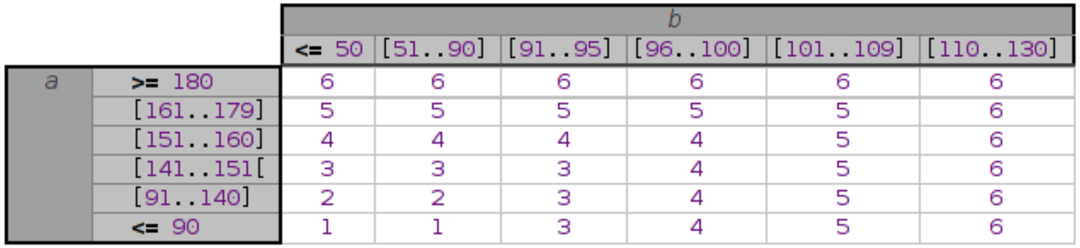
Context Actions

Attribute
- a
- b
- result

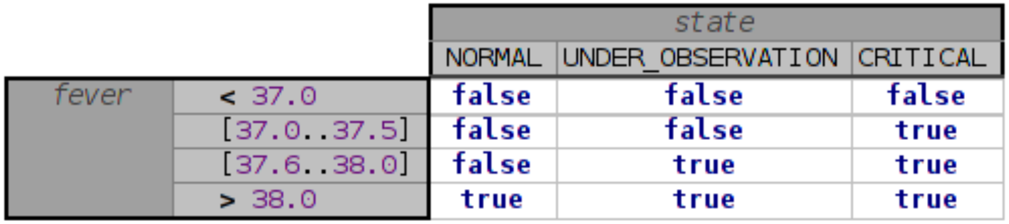
Aufzählungen
- HOCHR_VK.BUZB
- HOCHR_VK.BUZR
- HOCHR_VK.FLV
- HOCHR_VK.GFV
- HOCHR_VK.HZV_A
- HOCHR_VK.HZV_S
- HOCHR_VK.KAP
- HOCHR_VK.REN_A
- HOCHR_VK.REN_S
- HOCHR_VK.REN_U
- HOCHR_VK.RIS
- HOCHR_VK.RIZ
- HOCHR_VK.RSR
- HOCHR_VK.SBU
- HOCHR_VK.SEU
- PK_TYP_ID.AUSGANGS_KOMPONENTE
- PK_TYP_ID.BANK_KOMPONENTE
- PK_TYP_ID.DYNAMIK
- PK_TYP_ID.EXTERN_VERWALTETER_TARIF
- PK_TYP_ID.FOERDERKENNZEICHEN
- PK_TYP_ID.GRUPPIERUNG_RECHNUNGSLEGUNG
- PK_TYP_ID.KAPITAL_KONTO
- PK_TYP_ID.LV_PRODUKT_VT_PARAMETER
- PK_TYP_ID.LV_TARIF
- PK_TYP_ID.MANDANT
- PK_TYP_ID.OPTION
- PK_TYP_ID.PROZESS
- PK_TYP_ID.REGULIERUNG
- PK_TYP_ID.RISIKO_PRUEFUNG
- JECK_VERSICHERUNG
- ILAUSZAHLUNG
- RGUETUNGSGRUNDLAGE
- RTRIEBSWEG
- AEHRUNG
- HLWEG
- HLWEISE
- D
- KUNFT
- MY
- SERVICE.ERH

Decision Mechanisms



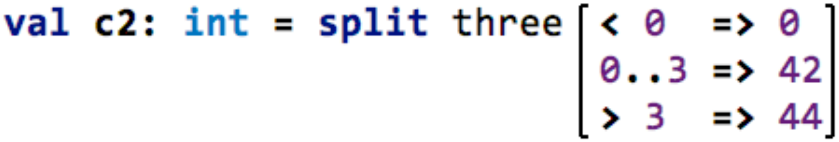


		b					
		<= 50	[51..90]	[91..95]	[96..100]	[101..109]	[110..130]
a	>= 180	6	6	6	6	6	6
	[161..179]	5	5	5	5	5	6
	[151..160]	4	4	4	4	5	6
	[141..151[	3	3	3	4	5	6
	[91..140]	2	2	3	4	5	6
	<= 90	1	1	3	4	5	6

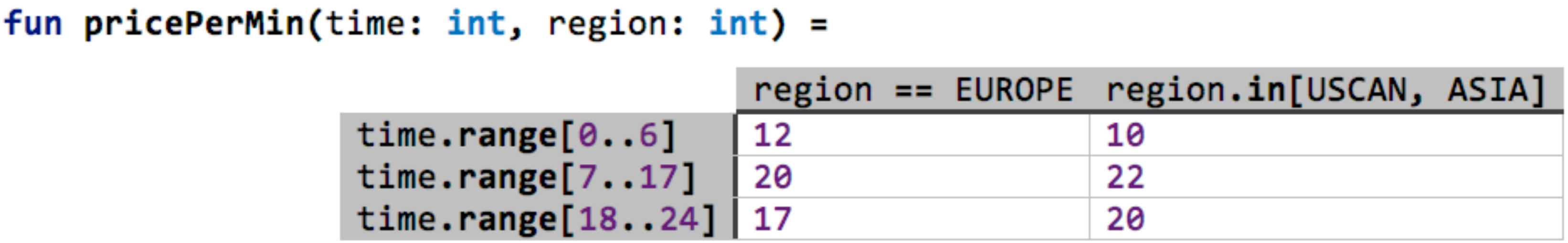


		state		
		NORMAL	UNDER_OBSERVATION	CRITICAL
fever	< 37.0	false	false	false
	[37.0..37.5]	false	false	true
	[37.6..38.0]	false	true	true
	> 38.0	true	true	true

Decision Mechanisms, directly in Expressions

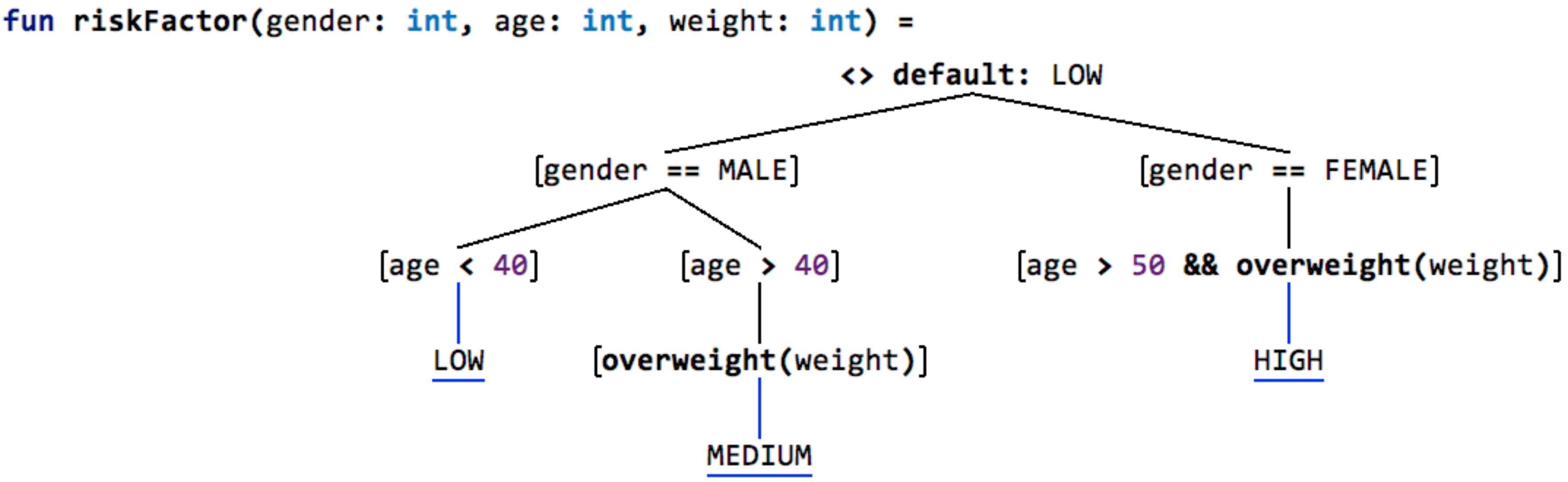


```
val c2: int = split three [< 0   => 0 ]
                          [0..3  => 42]
                          [> 3   => 44]
```



```
fun pricePerMin(time: int, region: int) =
```

	region == EUROPE	region.in[USCAN, ASIA]
time.range[0..6]	12	10
time.range[7..17]	20	22
time.range[18..24]	17	20



```
fun riskFactor(gender: int, age: int, weight: int) =
```

<> default: LOW

[gender == MALE]

[age < 40]

LOW

[age > 40]

[overweight(weight)]

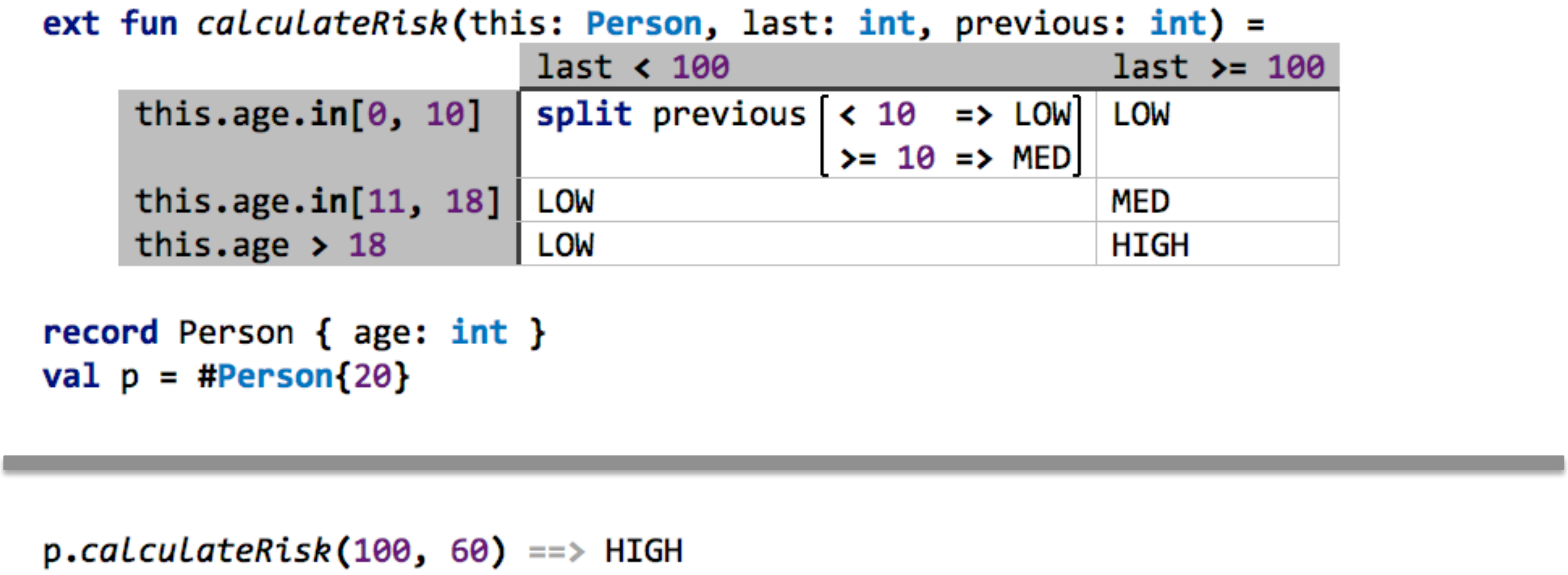
MEDIUM

[gender == FEMALE]

[age > 50 && overweight(weight)]

HIGH

Natural Language Function Calls I



```
ext fun calculateRisk(this: Person, last: int, previous: int) =
```

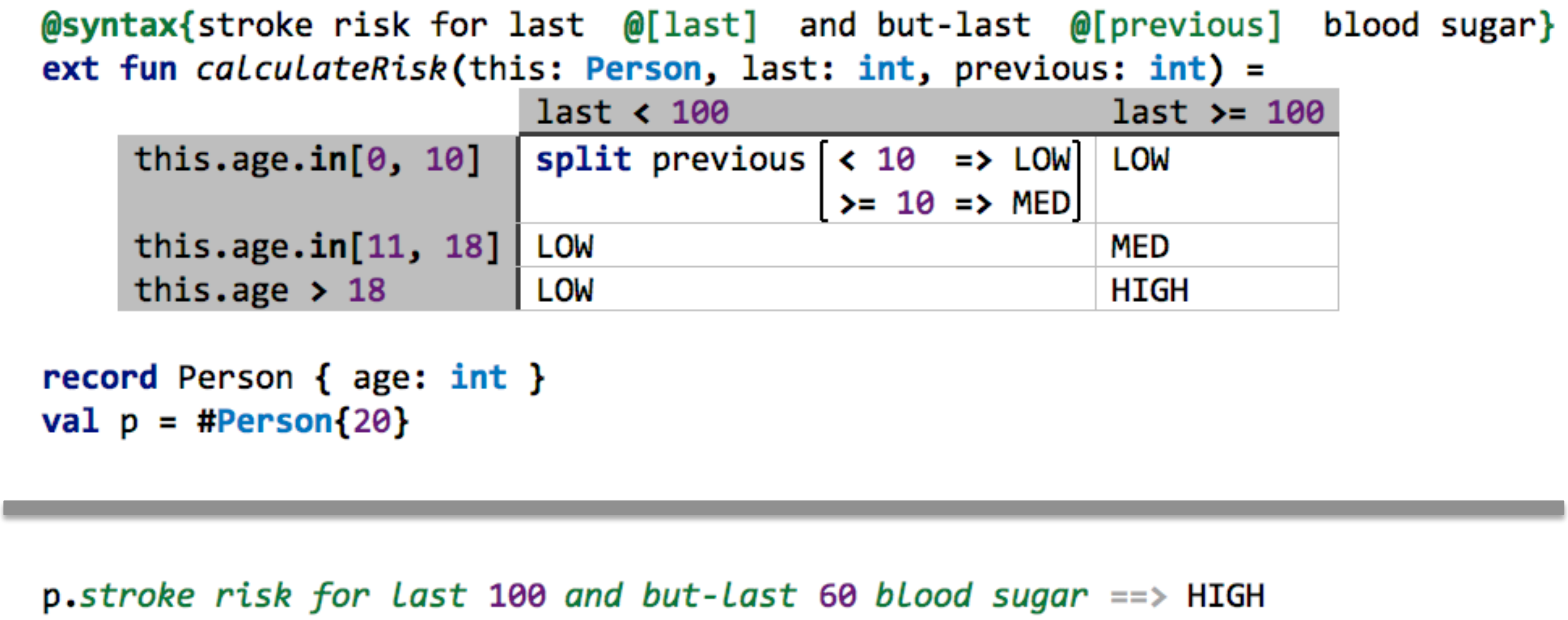
	last < 100	last >= 100
this.age.in[0, 10]	split previous [< 10 => LOW >= 10 => MED]	LOW
this.age.in[11, 18]	LOW	MED
this.age > 18	LOW	HIGH

```
record Person { age: int }
val p = #Person{20}
```

```
p.calculateRisk(100, 60) ==> HIGH
```

Extension function can be called in dot-notation, perfectly suitable for developers.

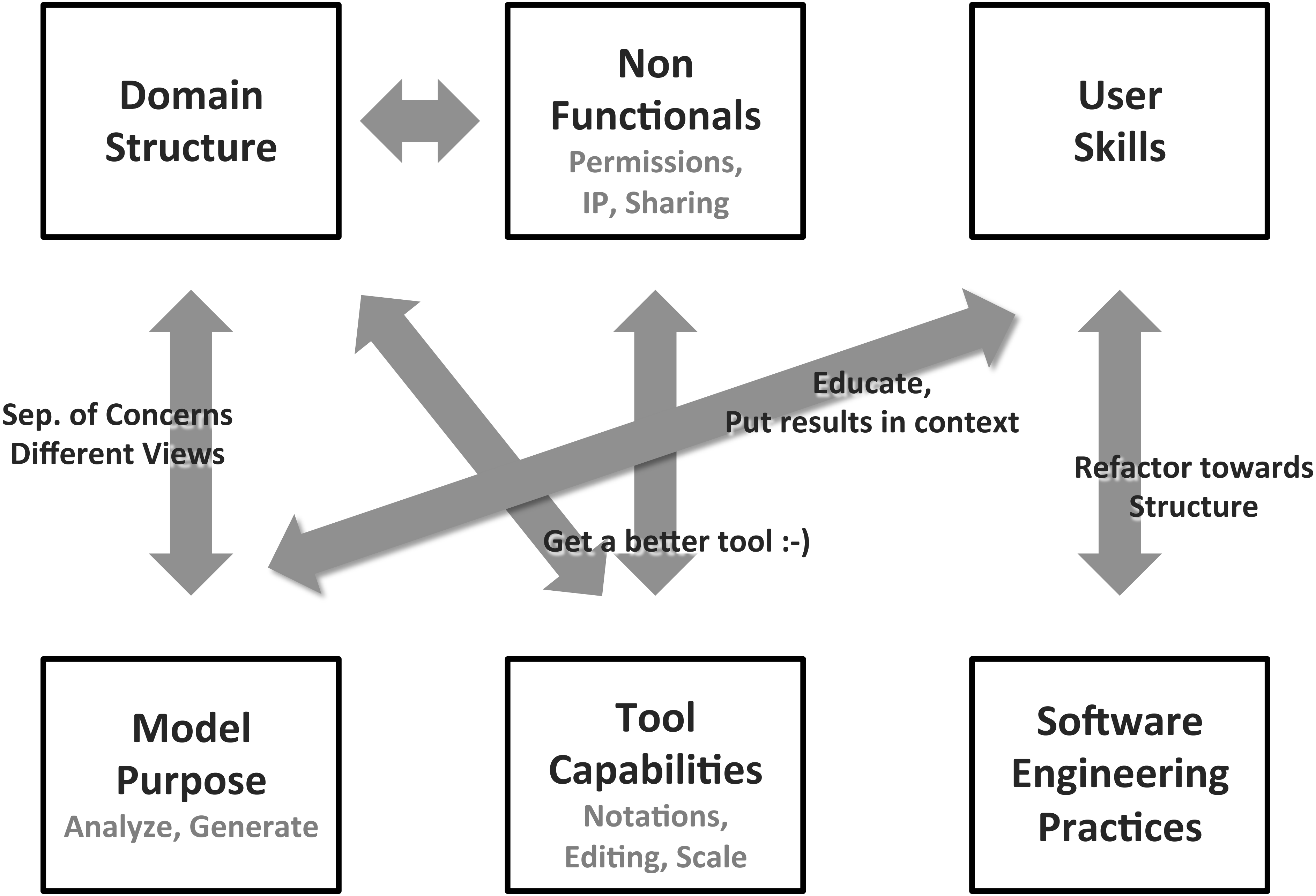
Natural Language Function Calls II



```
@syntax{stroke risk for last  @[last]  and but-last  @[previous]  blood sugar}
ext fun calculateRisk(this: Person, last: int, previous: int) =
                                 last < 100                         last >= 100
         this.age.in[0, 10]      split previous [< 10   => LOW]     LOW
                                                [>= 10  => MED]
         this.age.in[11, 18]     LOW                                MED
         this.age > 18           LOW                                HIGH

record Person { age: int }
val p = #Person{20}

p.stroke risk for last 100 and but-last 60 blood sugar ==> HIGH
```

For non-programmers, a more prose-like notation is helpful. Notice the prose-call facility is a modular extension of the expression language.

Influences on the Language



Domain Structure

Non Functionals

Permissions, IP, Sharing

User Skills

Sep. of Concerns Different Views

Educate, Put results in context

Refactor towards Structure

Get a better tool :-)

Model Purpose

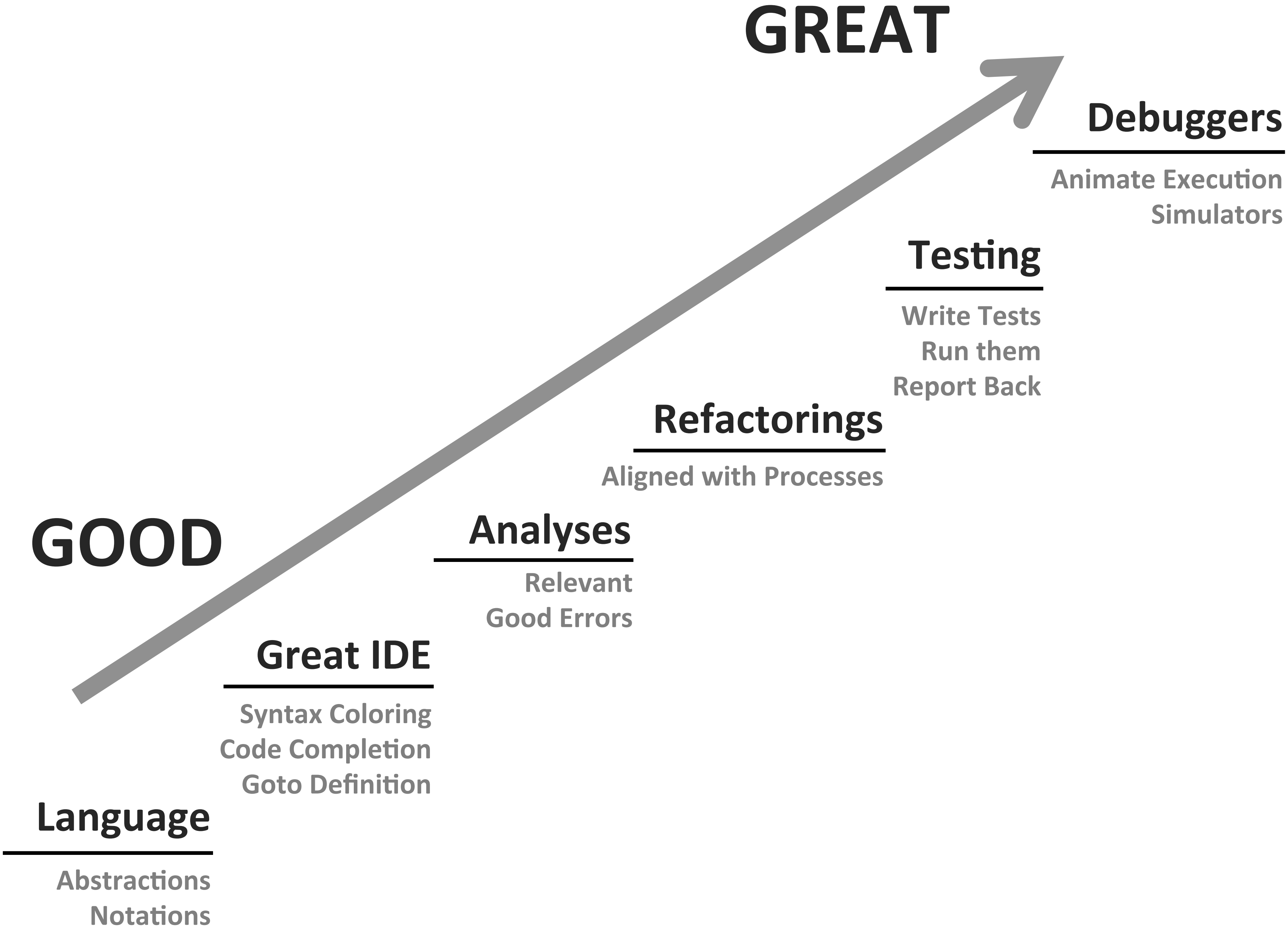
Analyze, Generate

Tool Capabilities

Notations, Editing, Scale

Software Engineering Practices

The Language is not Enough



GOOD

Language

Abstractions
Notations

Great IDE

Syntax Coloring
Code Completion
Goto Definition

Analyses

Relevant
Good Errors

Refactorings

Aligned with Processes

Testing

Write Tests
Run them
Report Back

Debuggers

Animate Execution
Simulators

GREAT

We tried it before, and it failed.

MDA

The UML tool was a bad choice

-> ok, choose a better one :-)

Hard to represent business logic in UML.

-> oh, really?? Who would have thunk.

Generate Class-Skeletons, fill in app logic.

-> how and why does this solve the challenges??

Round-Tripping did not work.

-> never works, but why use it?

Such an approach is completely pointless!!

Rule Language

No tests and debuggers for end users

-> hard to be sure about things

Language not expressive enough (tables)

Tool too limited to enhance expressivity

-> tedious to express many algorithms

Parts still had to be programmed manually

-> overall process more complex, not simpler

The right direction, but not good enough.

9

Drawbacks

You need inhouse expertise for language engineering

or a very close and trusted vendor who does it for you.

MISSION CRITICAL

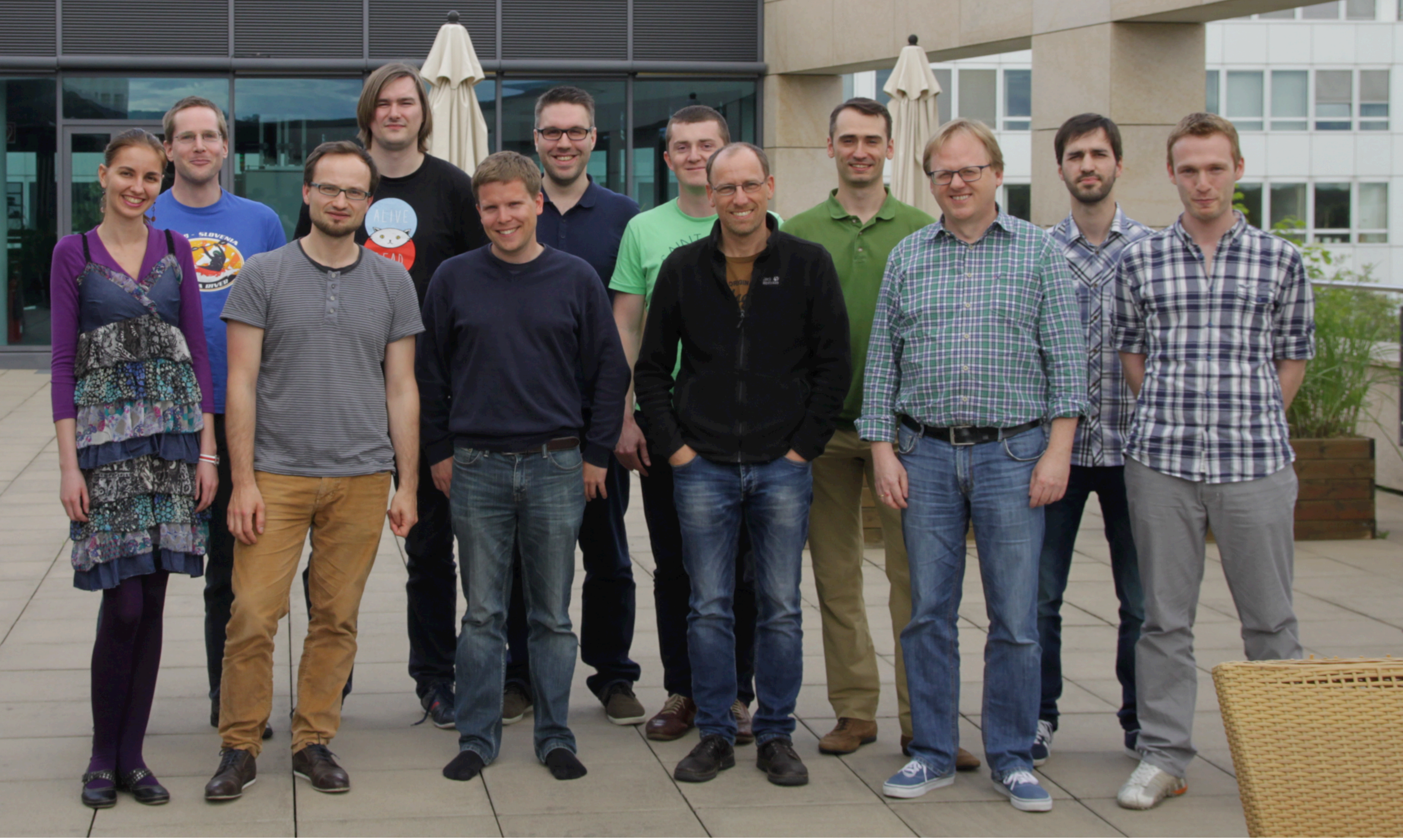
If you use this approach for real, **you should have language engineering expertise in house.**

You will invest a lot into a particular tool.

You can easily export models, but no portability for language defnitions.

10

Summary

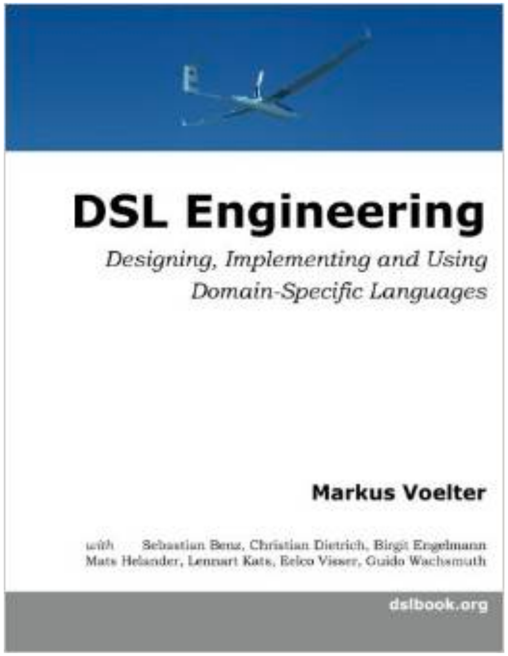


voelter {ingenieurbüro für softwaretechnologie // itemis

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

[open

source



DSL Engineering

Designing, Implementing and Using Domain-Specific Languages

Markus Voelter

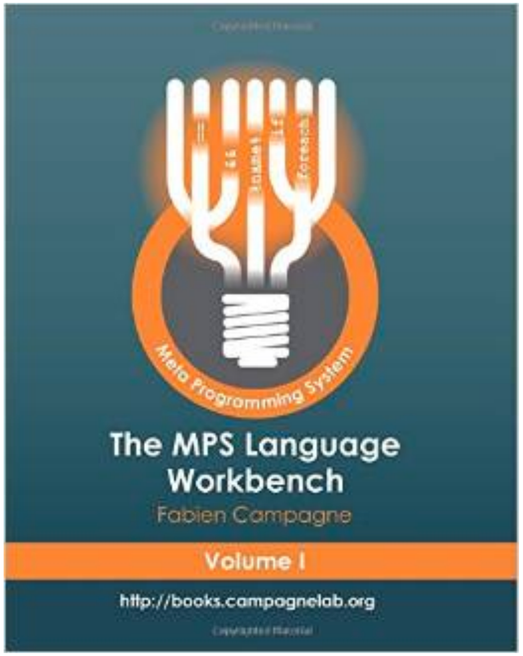
with Sebastian Benz, Christian Dietrich, Birgit Engelmann
Mats Helander, Lennart Kats, Eelco Visser, Guido Wachsmuth

dslbook.org

Generic Tools,
Specific Languages

Markus Voelter

mbeddr



Copyrighted Material

Meta Programming System

The MPS Language
Workbench

Fabien Campagne

Volume I

http://books.campagnelab.org

Copyrighted Material

mbeddr

MPS Meta Programming System

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.