Domain-Specific Languages in SPLE
Why and How?

voelter@acm.org
www.voelter.de
@markusvoelter
Markus Völter
itemis voelter { ingenieurbüro für softwaretechnologie //

1

The Big Picture

Digitalization – What & Why?

“So we gave everybody an iPad...“

Bringing Products Online
“Computering” Product Development

SPEED

Accelerate development & maintenance

CUSTOM

Simplify expected diversification & customization

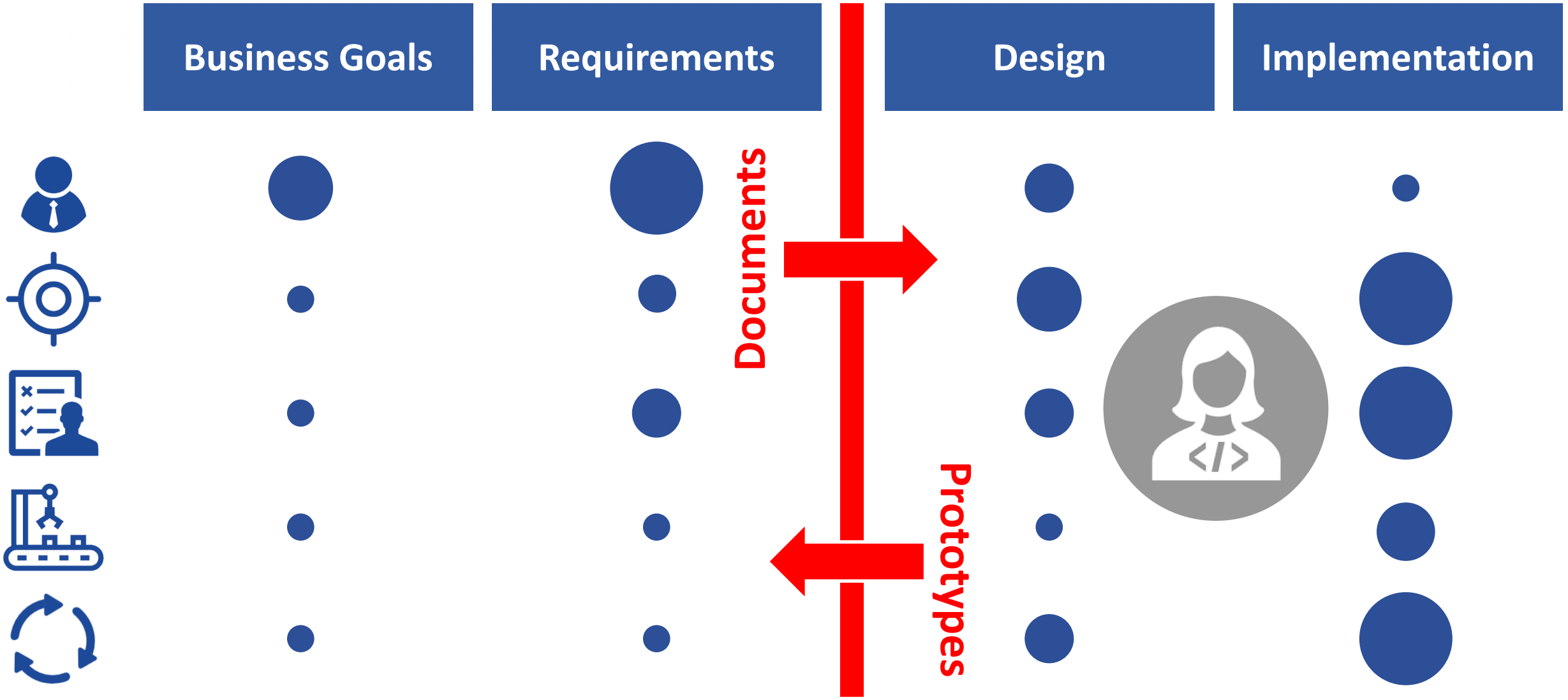
Digitalization – How?

People & Culture

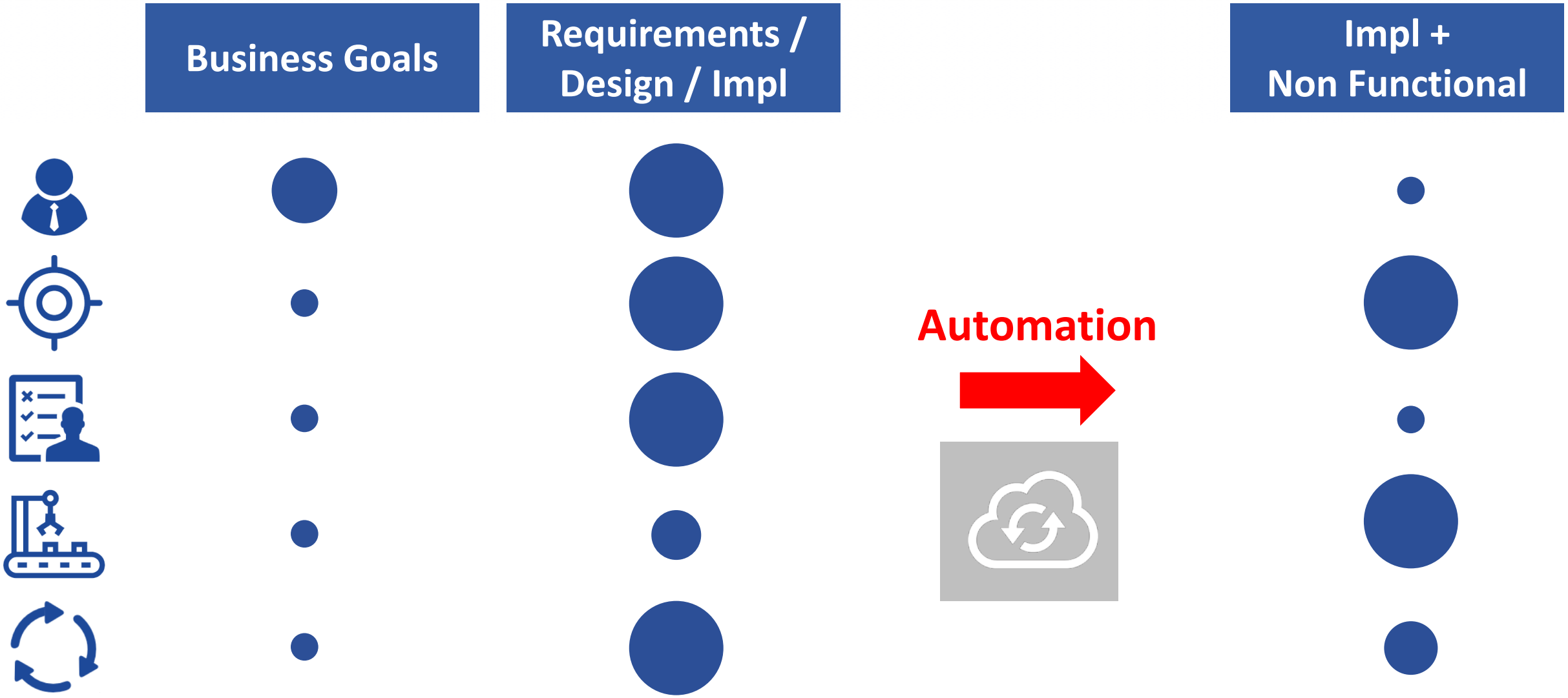
IT Infrastructure & Devops

Process, Methodology & Tools

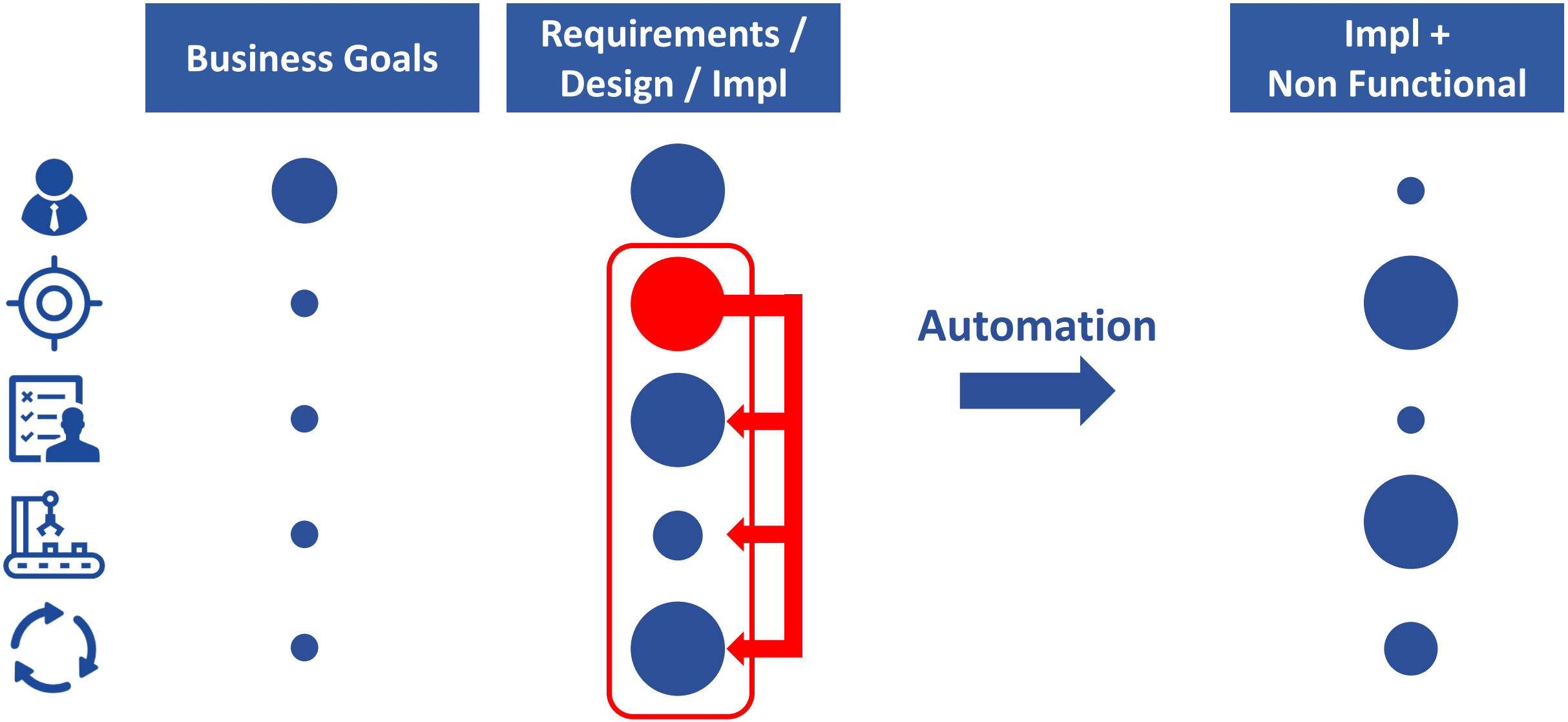
Digitalization – Process, Methodology & Tools



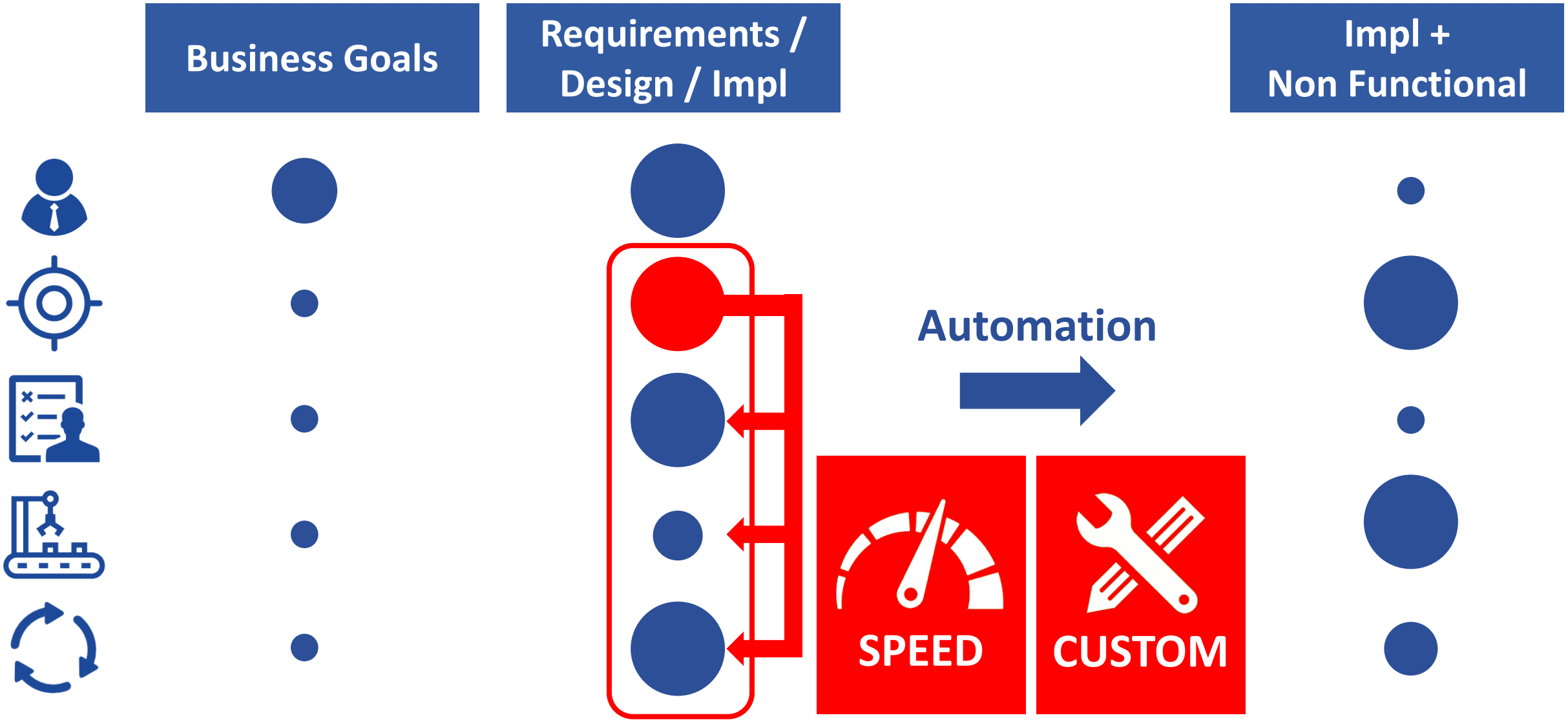
Digitalization – Process, Methodology & Tools



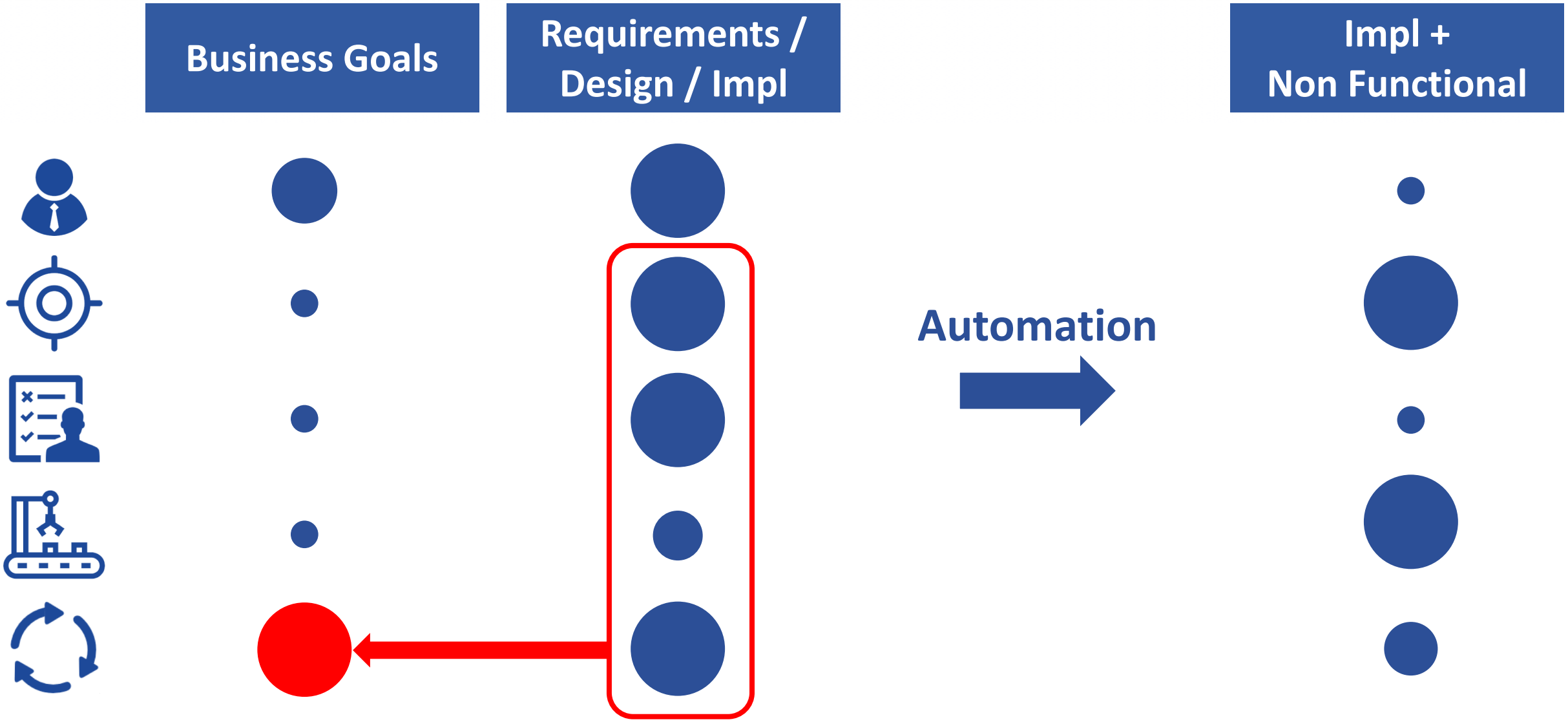
Digitalization – Process, Methodology & Tools



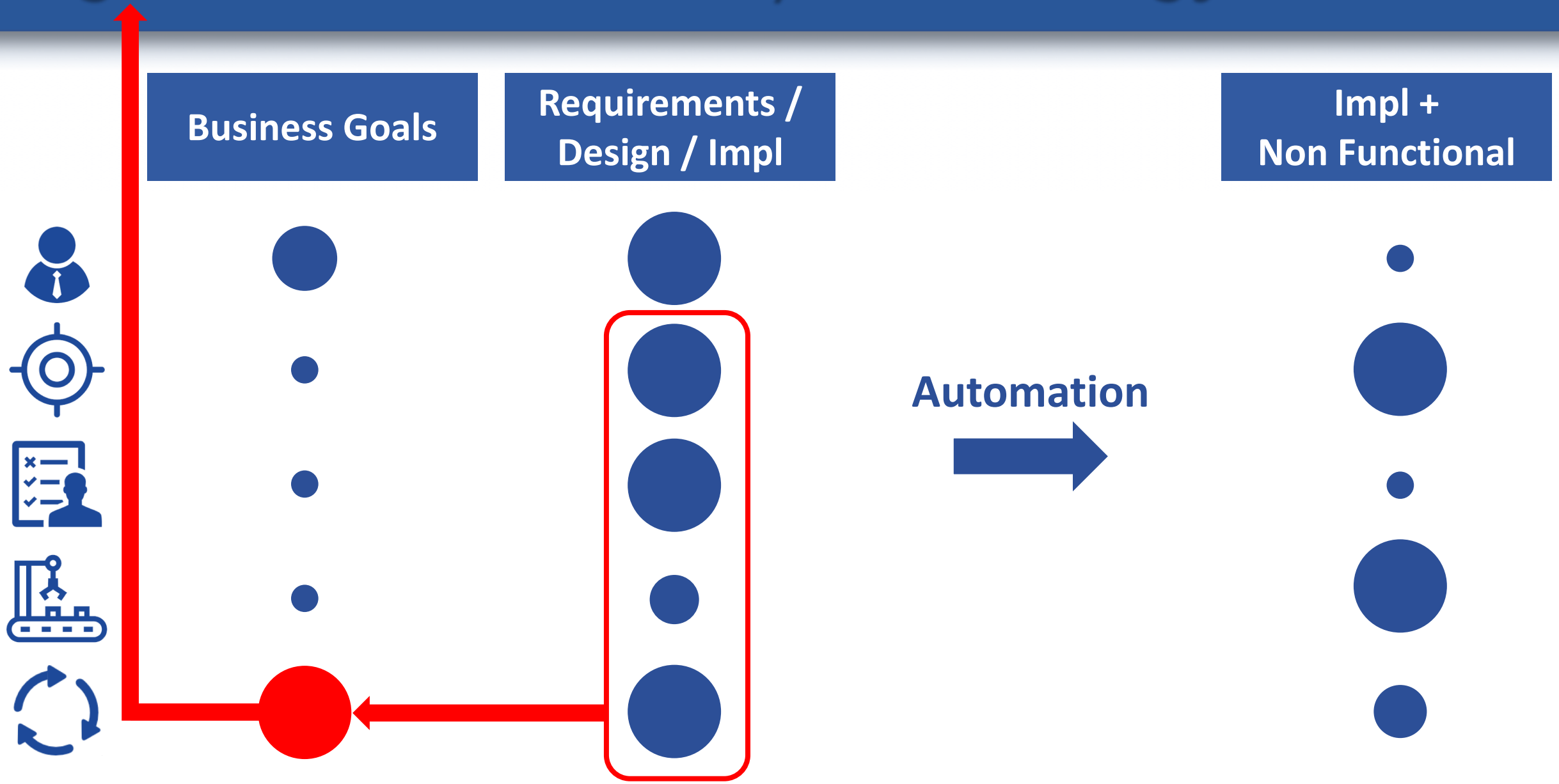
Digitalization – Process, Methodology & Tools



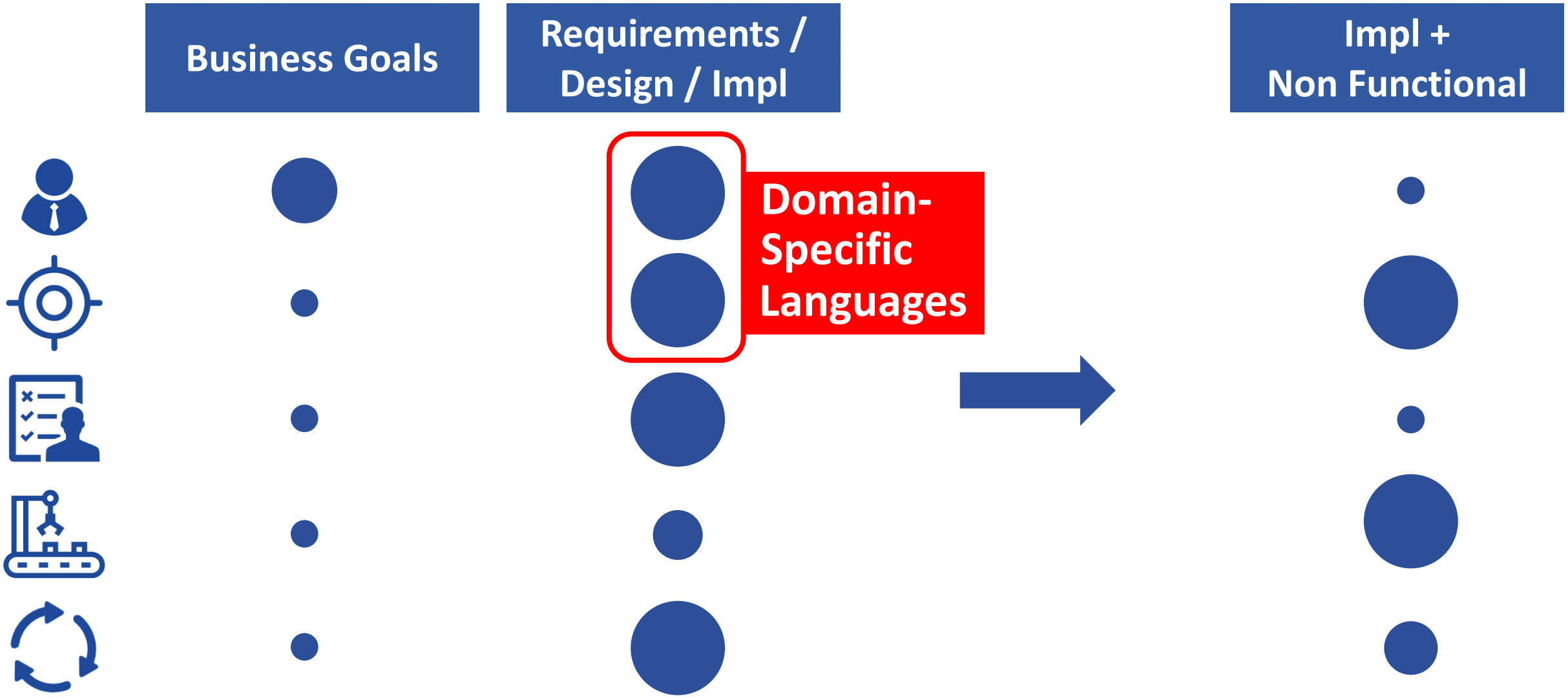
Digitalization – Process, Methodology & Tools



Digitalization – Process, Methodology & Tools



Digitalization – Process, Methodology & Tools



Digitalization – DSLs

Domain-Specific Languages

Good Abstractions and familiar Notations

Good Abstractions, Clear Semantics

Test Support & Good Abstractions & Notations

Verification | Validation

CI-Server Integration

Good IDE, with Interpreter

Express the right kind of variability

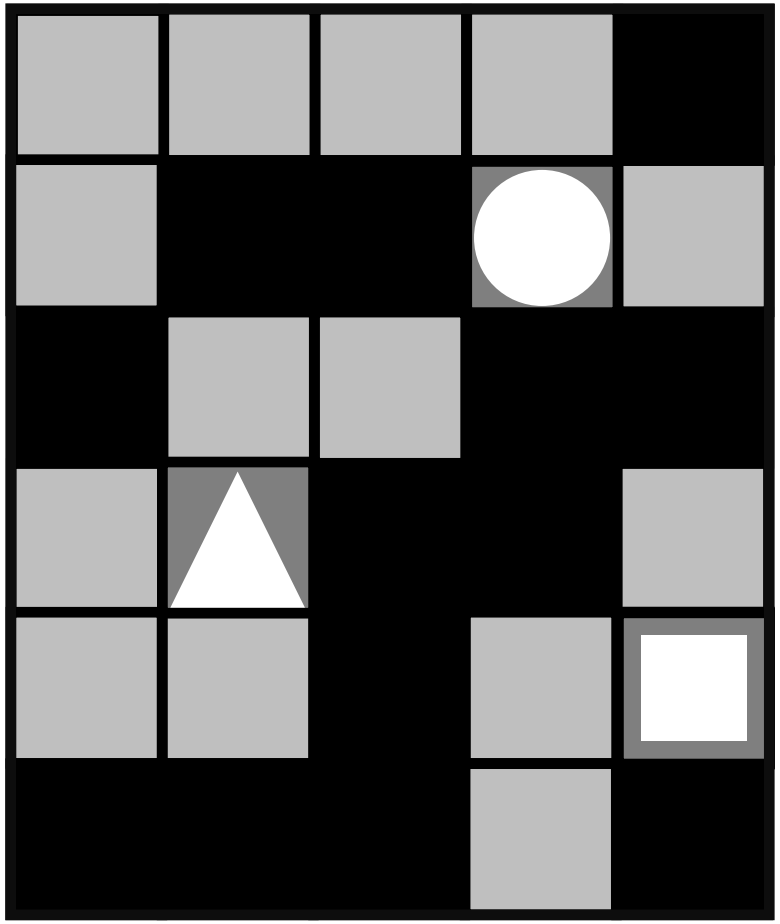
SPEED

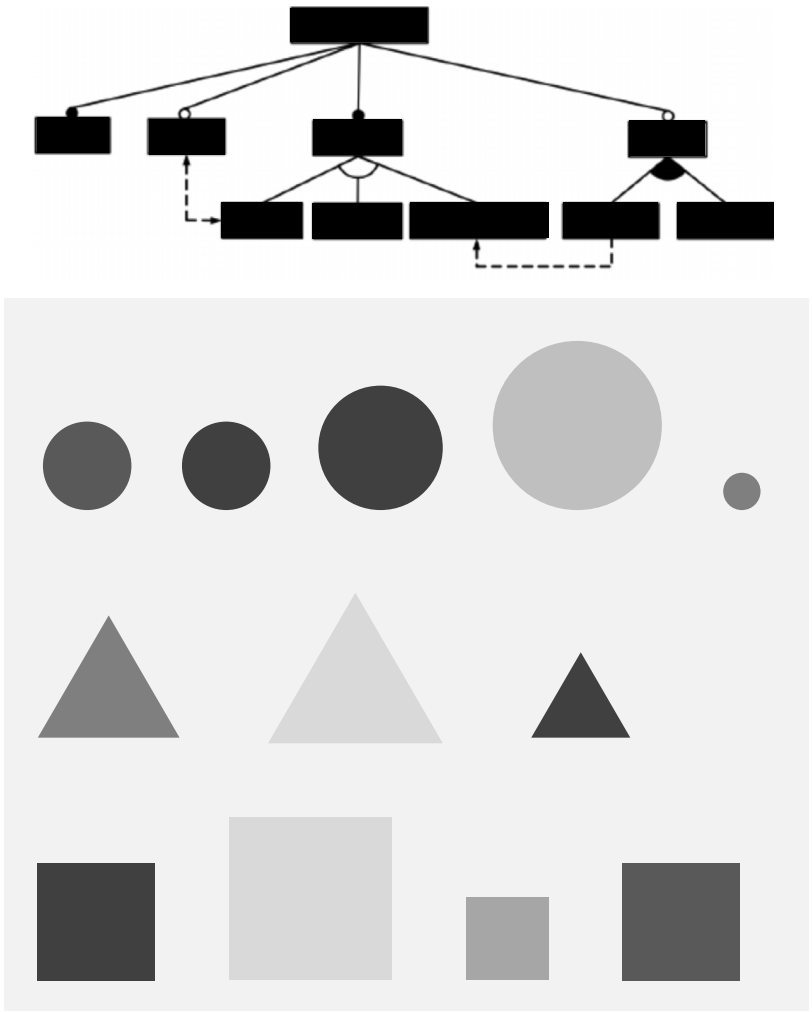
CUSTOM

2

Configuration vs. Construction

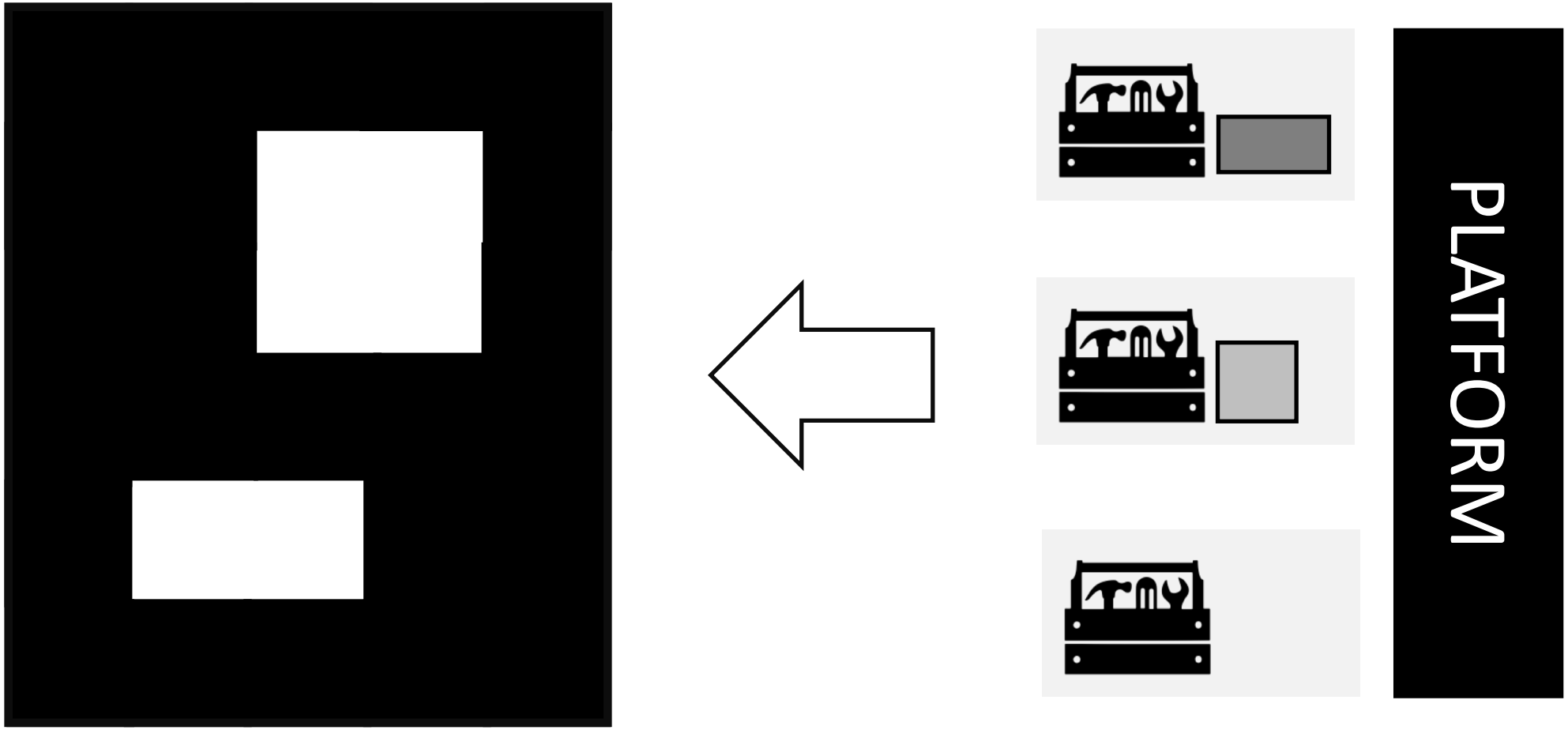
CONFIGURATION



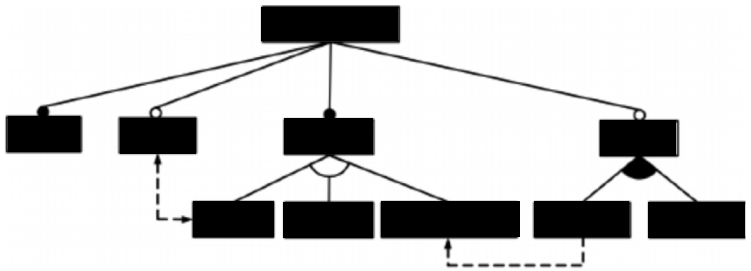


PLATFORM

CONSTRUCTION



COMPARISON



CONFIGURATIVE

CONSTRUCTIVE

Platform = Artifacts
"Selection"
Limited Set of Options
Easy to "assemble" product.

Platform = Tools
"Building"
Unlimited Set of Options.
More effort per Product.

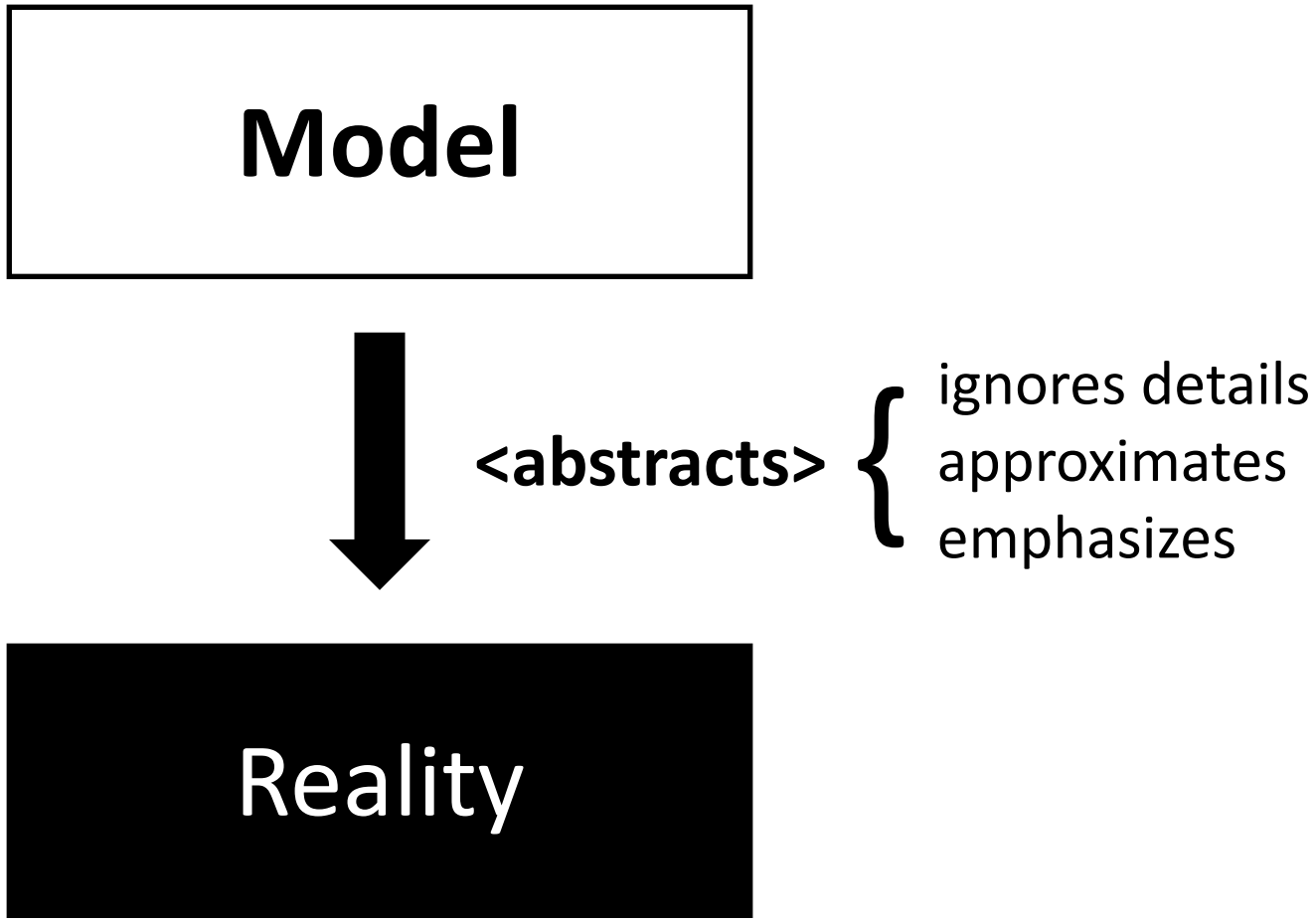
3

Domain-Specific Languages

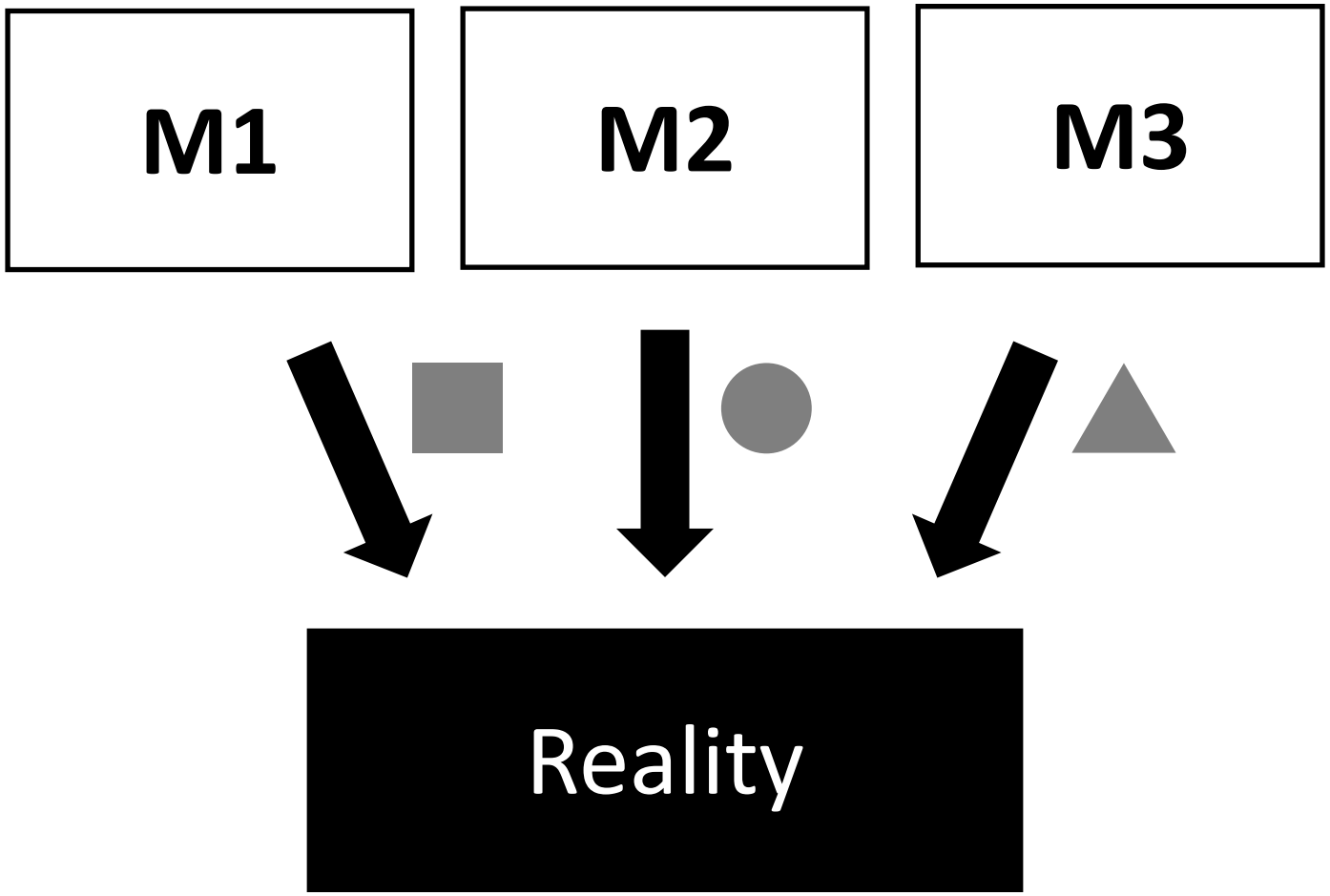
“A domain-specific language (DSL) is a computer language **specialized** to a particular application domain.”

Abstractions/Metamodel
Notation/Syntax
Analyses + Error Reporting
IDE + other Tools

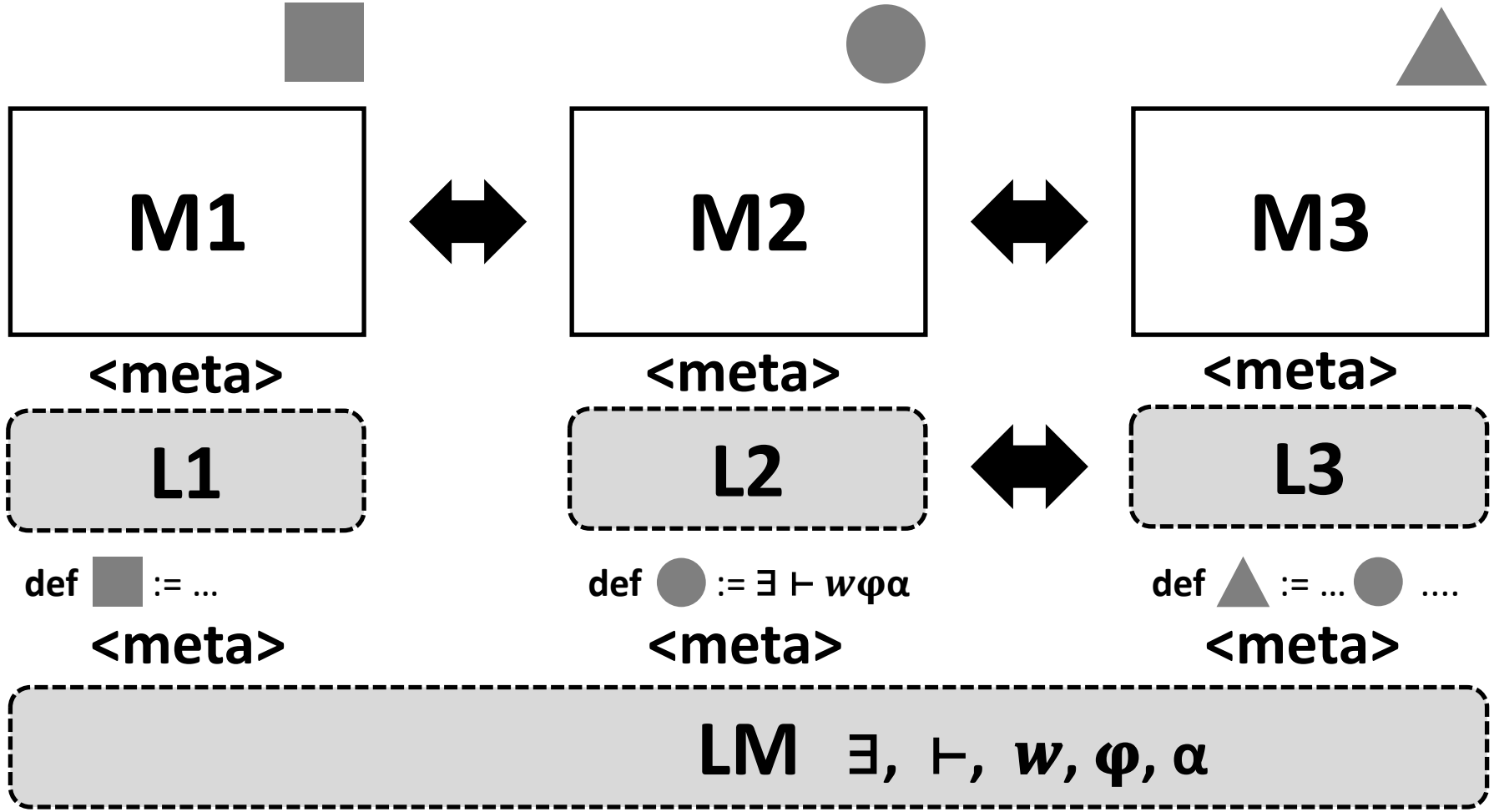
MODEL


Model
<abstracts>
ignores details
approximates
emphasizes
Reality

SEPARATION OF CONCERNS



LANGUAGE DEFINITION



MODEL KINDS

Climate Science
Particle Physics
Social Science
FEM Models

describe

understand
explain
analyze
predict

Modeling

Model

Reality

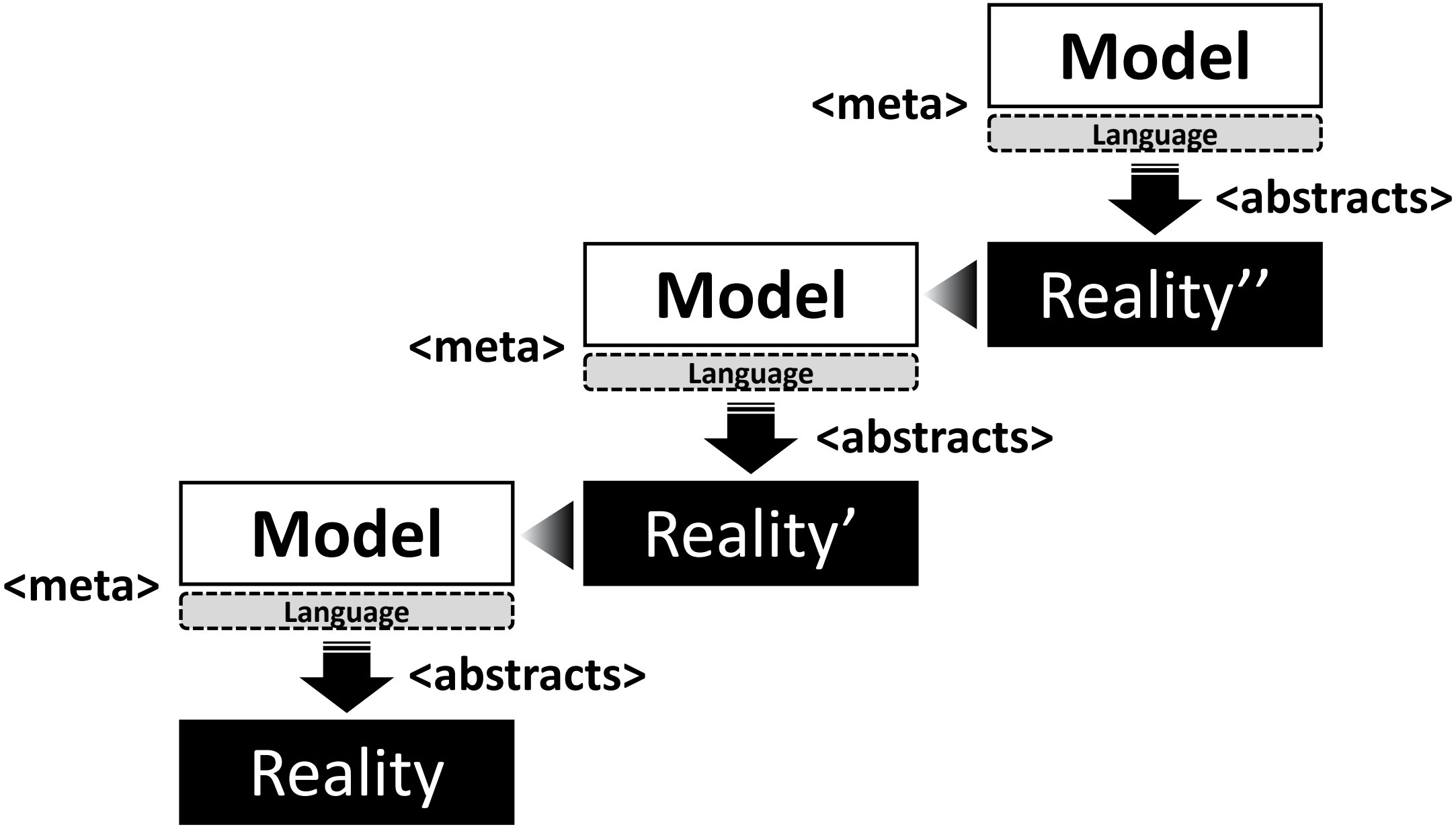
Manufacturing
Construction
Software Eng

prescribe

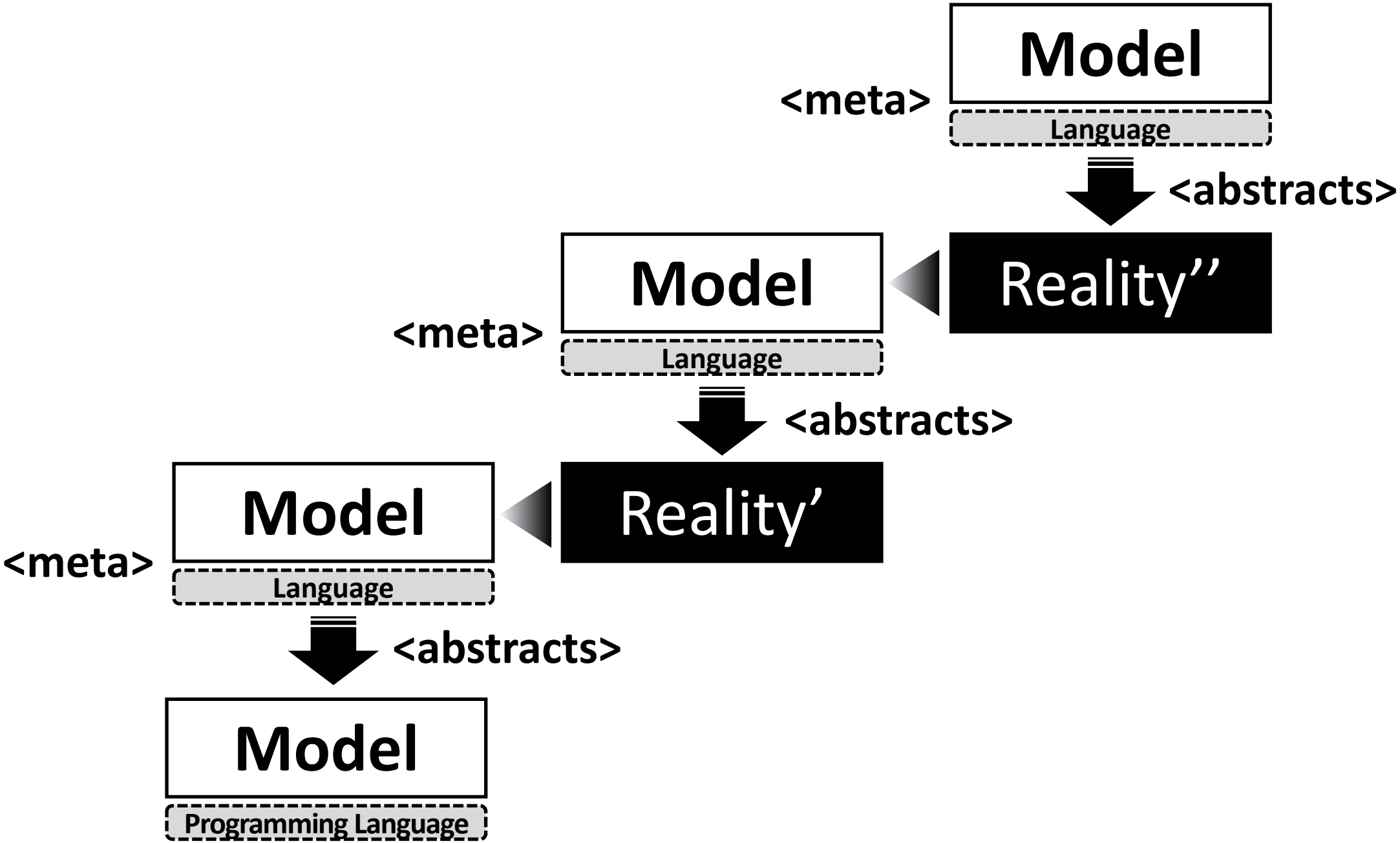
derive
manufacture

Model-Driven

MODEL DRIVEN SE



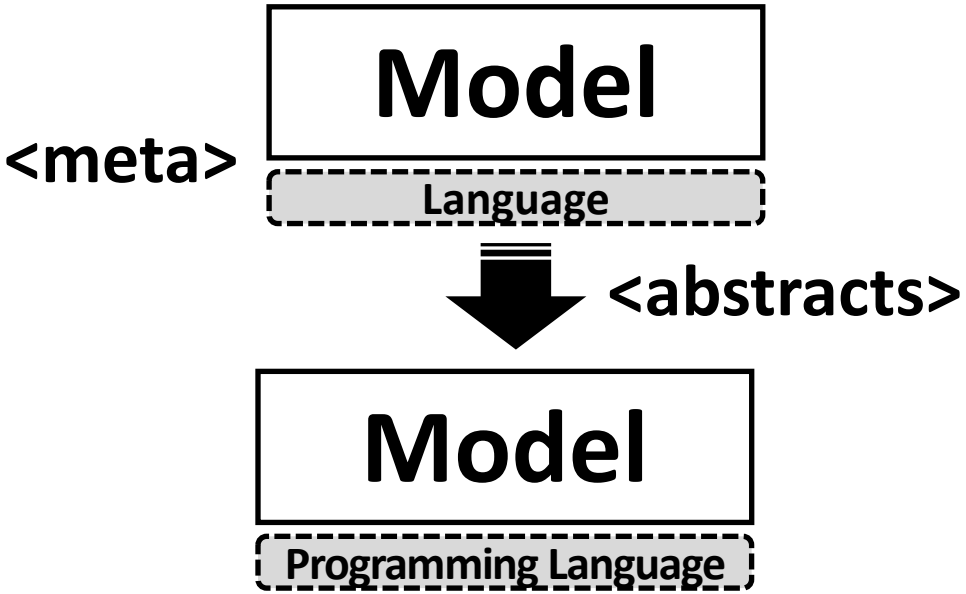
MODEL DRIVEN SE



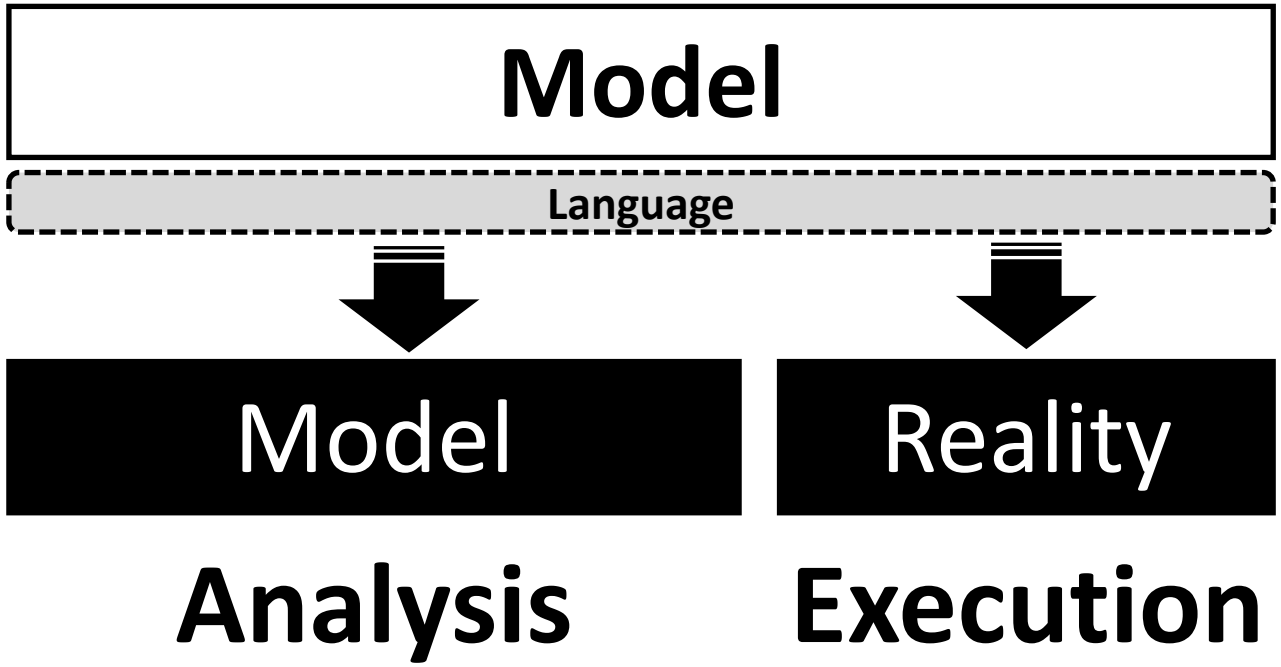
MODEL DRIVEN SE

Programming is just model-driving using a programming language.

And caring about particular concerns.

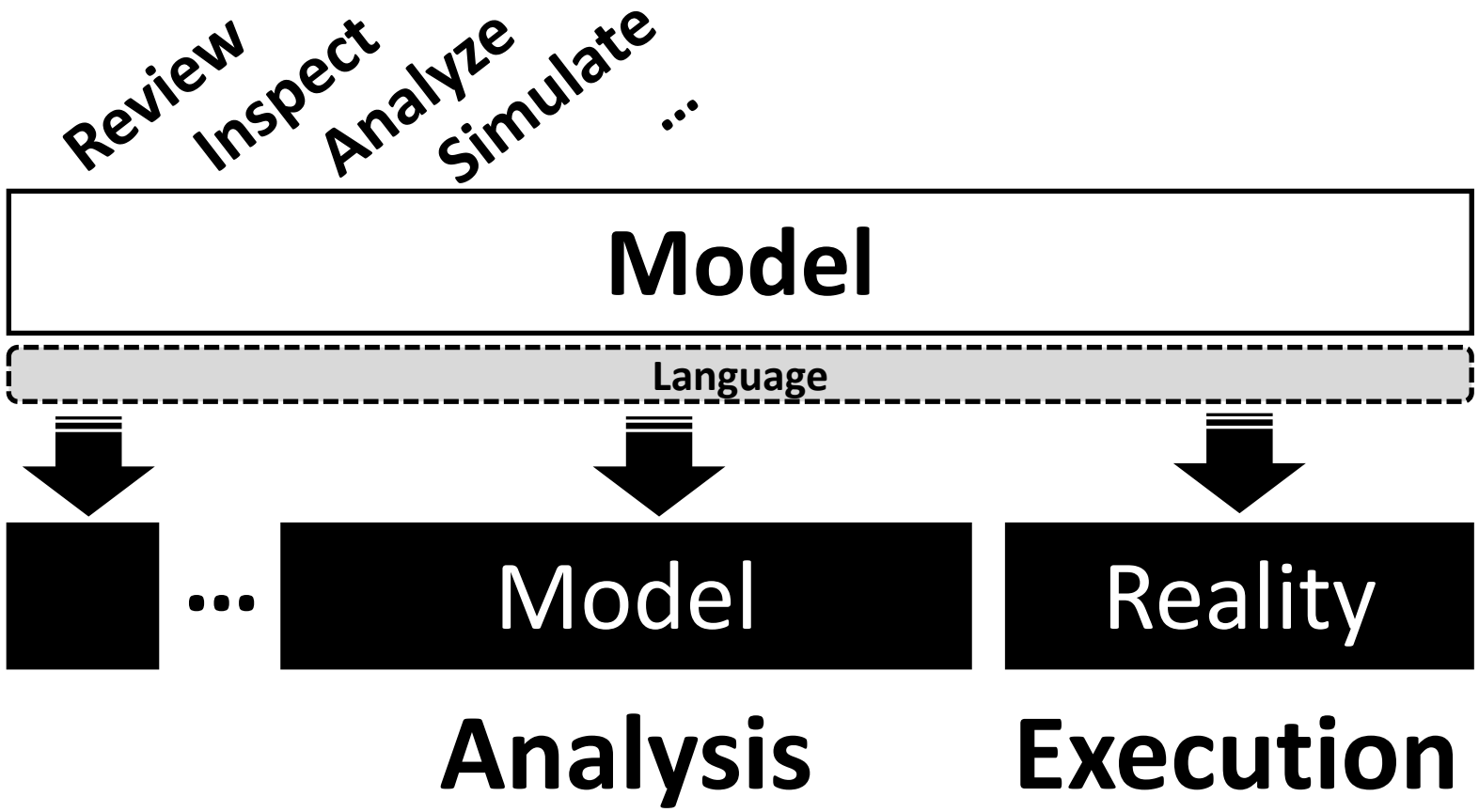


MODEL DRIVEN SE



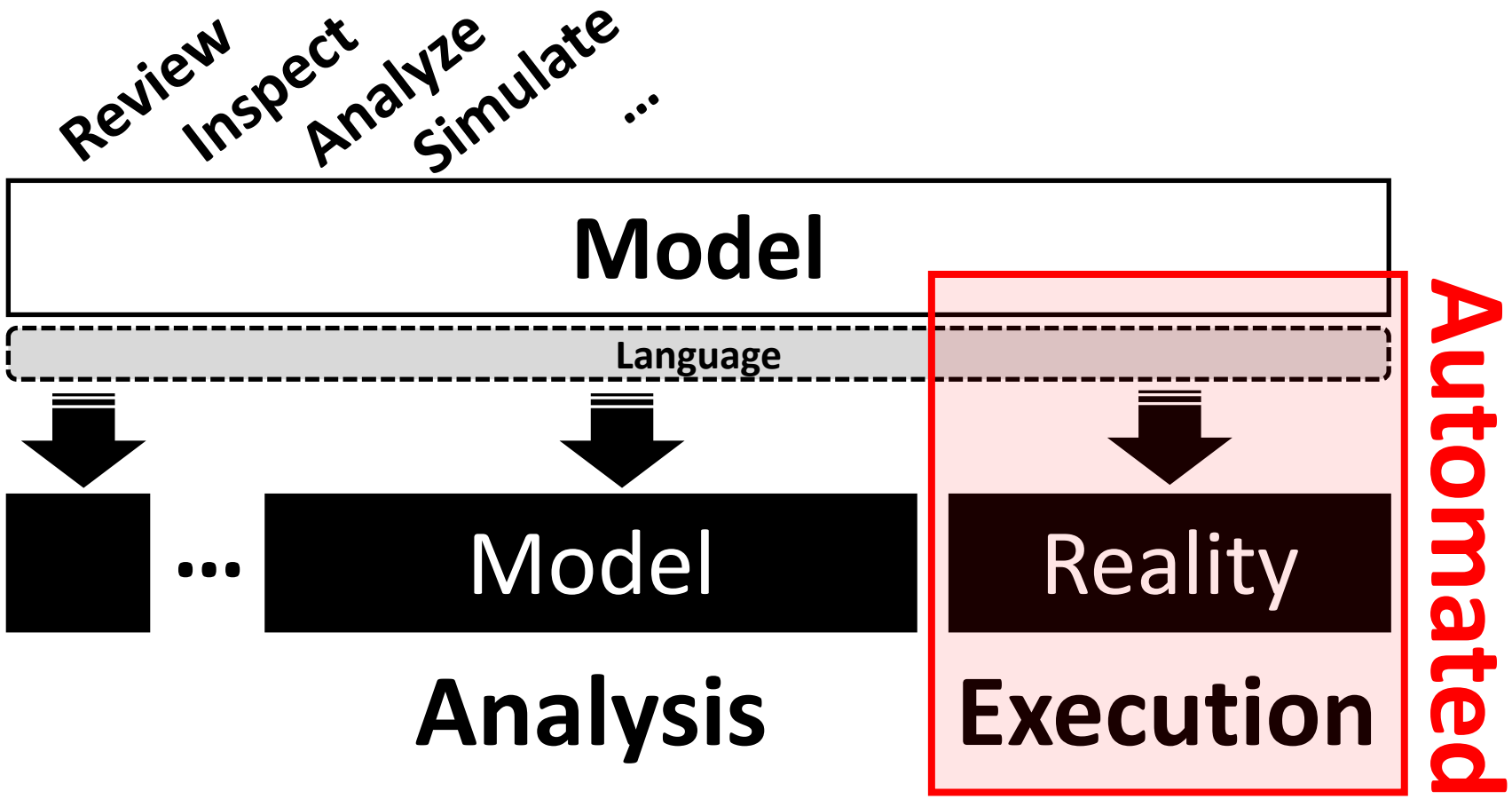
Aligning the Semantics of the two ■ and ⬇ is key.

MODEL PURPOSE



Diverse reasons why you would want to model beyond reducing the coding effort by generating the implementation from more concise models.

MODEL PURPOSE



But: Automated Execution is essential to ”connect” the models to the actual system.

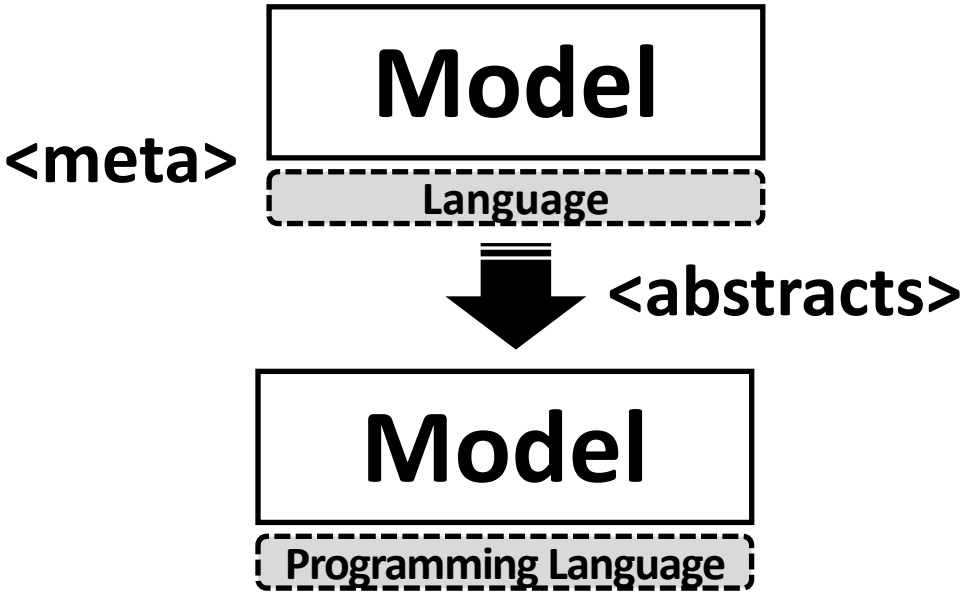
Rabbit Hole

Modeling vs. Programming

MODEL DRIVEN SE

Programming is just model-driving using a programming language.

And caring about particular concerns.



Programming vs. MD: What it can be

	Model-Driving	Programming	21st Century MD
High-level, Domain-Specific Concepts			DSLs
User-definable Abstractions			Language Extension
Focus on Behavior and Algorithms			Expr., Math, State M.,
Multiple Integrated Languages			
Notational Freedom			Txt, Tables, Math, Diag
Type Systems (vs. Constraint Checks)			
Focus on Technical Concerns			Separated Concerns
Focus on Execution			
Focus on Analysis			
Powerful, productivity-focused IDEs			
Integration of Non-Programmers			
Language Modularity and Comp'n			

4

Example DSLs

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

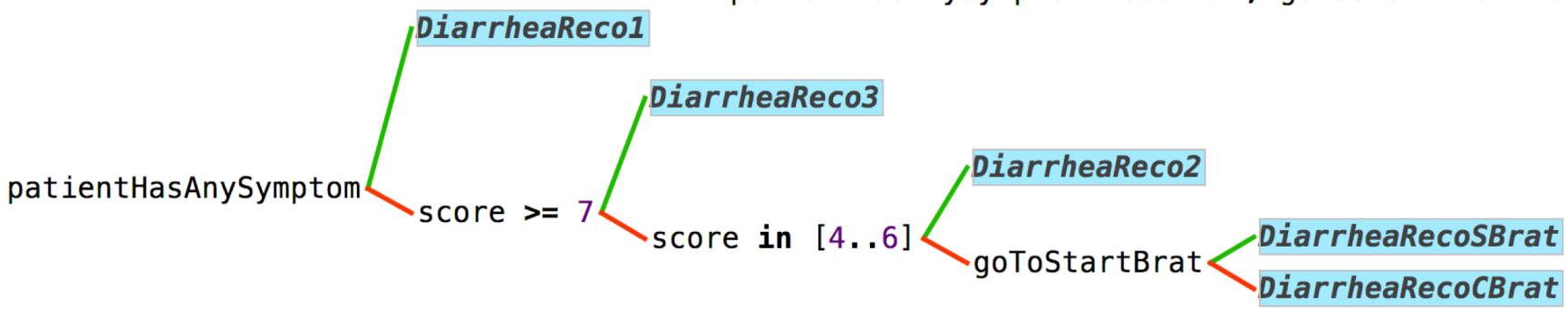
:pk_typ_id	Beschreibung hinzufügen	Bemerkung
PK_TYP_ID.KAPITAL_KONTO	**If** (**:vtrk_zustand** = ZUSTAND.BPFL) vkzb **= :vtrk_zb** **End If** :vtrk_zustand = ZUSTAND.BPFL	Beschreibung hinzufügen
PK_TYP_ID.LV_TARIF	**If** (**:stamm_ptr** <> **NULL**) **If** (**:zustand** = ZUSTAND.BPFL) vkzb **= :vtzb** **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

```
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}

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

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

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.**lookup**(x, geschlecht))

```
multi-party-decision Unanimous
parties: bernd, markus
dynamic? ☐
procedure: unanimous    time limit: <none>
turnout:   <none>       revokable? ☐
```

```
d.[ ] }
  decisionTaken
  toList  (BaseConcept in jetbrains.mps.lang.core)
  vote
  whoVoted
```

```
multi-party-decision Unanimous
parties: bernd, markus
dynamic? ☐
procedure: unanimous    time limit: <none>
turnout:   all          revokable? ☐
```

```
d.[ ] }
  decision
  toList  (BaseConcept in jetbrains.mps.lang.core)
  turnoutAchieved
  voteAgainst
  voteFor
  whoVoted
  whoVotedAgainst
  whoVotedFor
```

```
multi-party-decision Unanimous
initial parties: bernd, markus
dynamic? ☒   sealable? ☐
procedure: unanimous    time limit: <none>
turnout:   <none>       revokable? ☐
```

```
d.[ ] }
  addParties
  addParty
  decisionTaken
  registeredParties
  toList  (BaseConcept in jetbrains.mps.lang.core)
  vote
  whoVoted
```

```
state machine Game (initialPlayers: list<party>) {
  var players = initialPlayers
  var pendings = nopendings
  var bids = nobids
  var lastWinner: opt<party> = none

  event requestAccess
  event startVoting
  event voteCandidate(candidate: party, vote: boolean)

  event terminatePendings
  event play
  event offerBid(money: bid)
  event endGame

  fun clearGame/RM() {
    lastWinner := none
    bids := nobids
  }
```

```
  state playing [senderIs(players)] {
    entry: clearGame/M(EMPTY )
    state bidding [takeTurns(players|ordered|after 500 boot out)] (initial) {
      on offerBid(money) : bids := bids.put(sender->money)
      if [timeInState > 2000] -> finished
    }
    state finished {
      entry: {
        val maxOfferedValue = bids.values.max
        lastWinner := bids.keys.findFirst(bids[it] == maxOfferedValue)
      }
      on endGame -> setup
    }
    exit: clearGame/M(EMPTY )
  }
}
```

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}

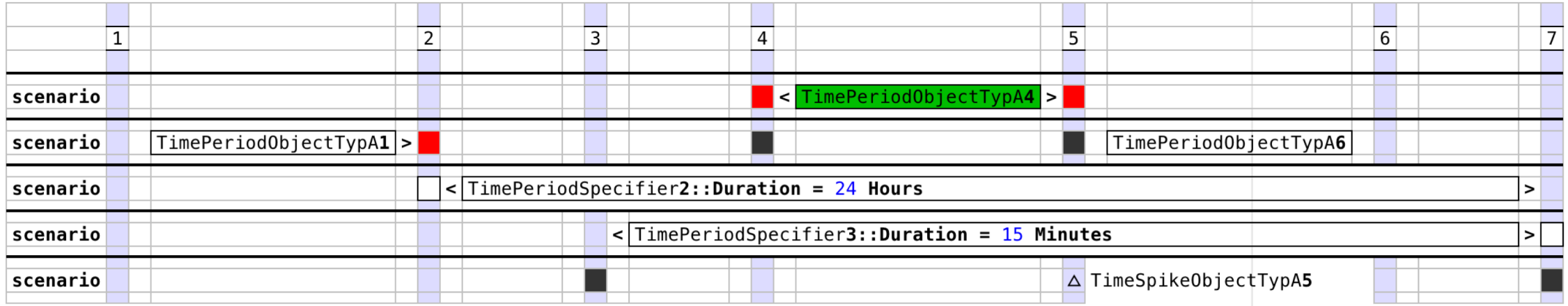
---

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**

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

parameters: TimePeriodObjectTypA4

patterns:
```




```
}
```

database databaseOneElementAcrossRows

Type	Begin	End	Duration	Occurence
eTimePeriodObjectTypA	500	550	50	

database databaseOneAndMoreIterationsHappy

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

Risks Table		Required attack potentials				
		Beyond high	High	Moderate	Low	Very low
Damage potentials	Very low	Low risk	Low risk	Low risk	Moderate risk	Moderate risk
	Low	Low risk	Low risk	Moderate risk	High risk	High risk
	Moderate	Low risk	Moderate risk	High risk	High risk	Very high risk
	High	Moderate risk	High ris			
	Very high	Moderate risk	High ris			

SZ.42
Integrity Webservice
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

Manipulation/Tampering...
RAP: Very low

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	
Independent: Very low					
PRFC	RMAC	HRS	STRD	PUBL	
PRFC	RMAC	HRS	STRD	PUBL	

+

B.14
(Identity) Spoofing of...
RAP: Beyond high

SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
Independent: Very low					
PRFC	RMAC	HRS	STRD	RSTR	
PRFC	RMAC	HRS	STRD	RSTR	

+

B.15
Man-in-the-Middle Atta...
RAP: Beyond high

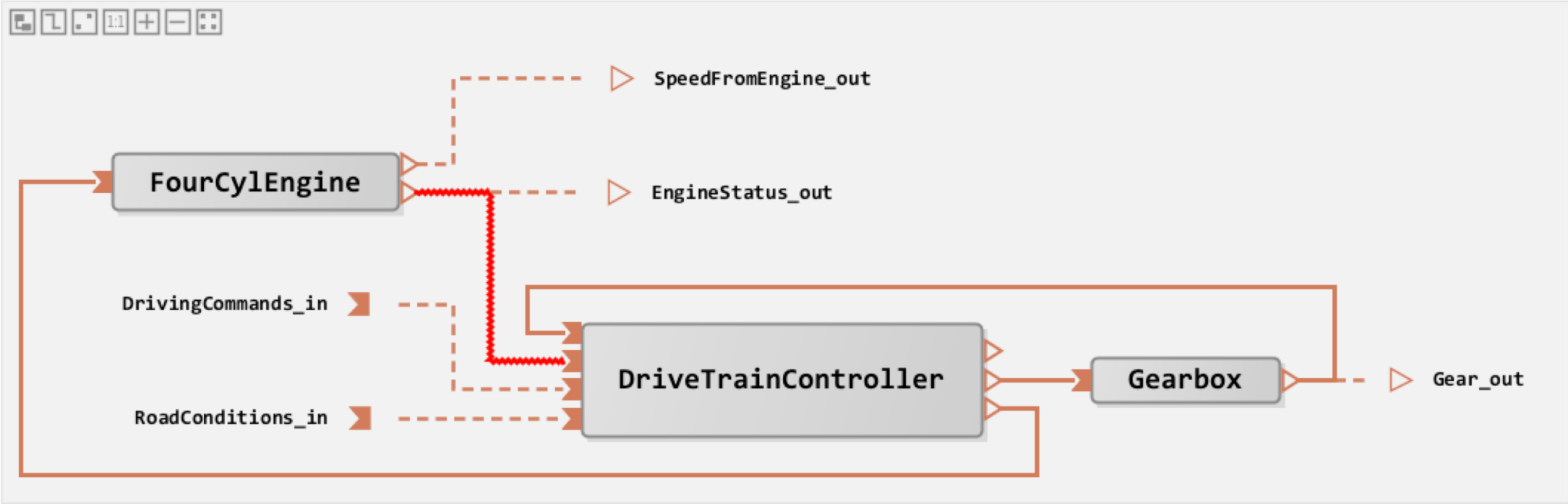
SPEX	ACCS	TIME	EQUIP	KNOW	eRAP
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
PRFC	RMAC	HRS	STRD	RSTR	Beyond high
Independent: Very low					
PRFC	RMAC	HRS	STRD	RSTR	
PRFC	RMAC	HRS	STRD	RSTR	

SZ.30
Authenticity WebsiteData
RAP: Very low

					eRAP

Name	Title	Security goal classes	SUD Elements	Criteria				Depends on
				SAF	FIA	LAW	QUA	
SZ.16	Confidentiality UpdateRequest	Confidentiality	UpdateRequest			DPI		SZ.49 && SZ.37
SZ.17	Availability UpdateRequest	Availability	UpdateRequest				NSD	SZ.18 && SZ.50
SZ.18	Authenticity UpdateRequest	Authenticity	UpdateRequest			DPE		SZ.51
SZ.19	Confidentiality UpdateContainer	Confidentiality	UpdateContainer		LLO	DPE		SZ.21 && SZ.37 && SZ.52 && SZ.49
SZ.20	Availability UpdateContainer	Availability	UpdateContainer				NSD	SZ.21 && SZ.38 && SZ.50 && SZ.53
SZ.21	Authenticity UpdateContainer	Authenticity	UpdateContainer	ISL			NSD	SZ.39 && SZ.51 && SZ.54
SZ.22	Confidentiality ConfigurationProfiles	Confidentiality	ConfigurationProfiles			DPE		SZ.24 && SZ.52 && SZ.55
SZ.23	Availability ConfigurationProfiles	Availability	ConfigurationProfiles				NSD	SZ.24 && SZ.53 && SZ.56
SZ.24	Authenticity ConfigurationProfiles	Authenticity	ConfigurationProfiles				NSD	SZ.54 && SZ.57
SZ.26	Availability D4g	Availability	ProfileSelection				NSD	SZ.56
SZ.27	Authenticity ProfileSelection	Authenticity	ProfileSelection				NSD	SZ.57
SZ.28	Confidentiality WebsiteData	Confidentiality	WebsiteData			DPE		SZ.30 && SZ.40
SZ.29	Availability WebsiteData	Availability	WebsiteData				NSD	SZ.30 && SZ.41
SZ.30	Authenticity WebsiteData	Authenticity	WebsiteData			DPE	NSR	SZ.42
SZ.52	Confidentiality GCU	Confidentiality	GCU		LMO	DPE		SZ.54
SZ.53	Availability GCU	Availability	GCU				NFV	SZ.54
SZ.54	Integrity GCU	Integrity	GCU	ISL	LMO	DPE	NFV	<no expr>

```
public functional component DriveTrain {
  produces SpeedFromEngine
  produces EngineStatus
  produces Gear where it < gearsCount
  consumes RoadConditions
  param int gearsCount
  consumes DrivingCommands
```




```
}
```

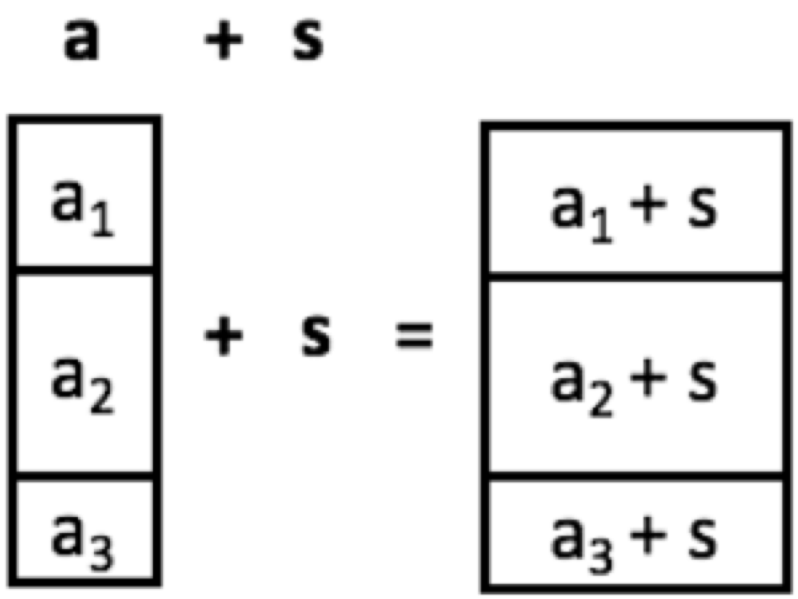
```
vector<int16, 3> aVector = [1; 2; 3] * 512;

vector<int16, 3> resultOfCrossProduct = aVector x aVector;

matrix<int16, 2x3> aMatrix = [1 + 2   2 * 7   42; 3   51   24];

matrix<int16, 3x2> transposedMatrix = aMatrix^T;
```

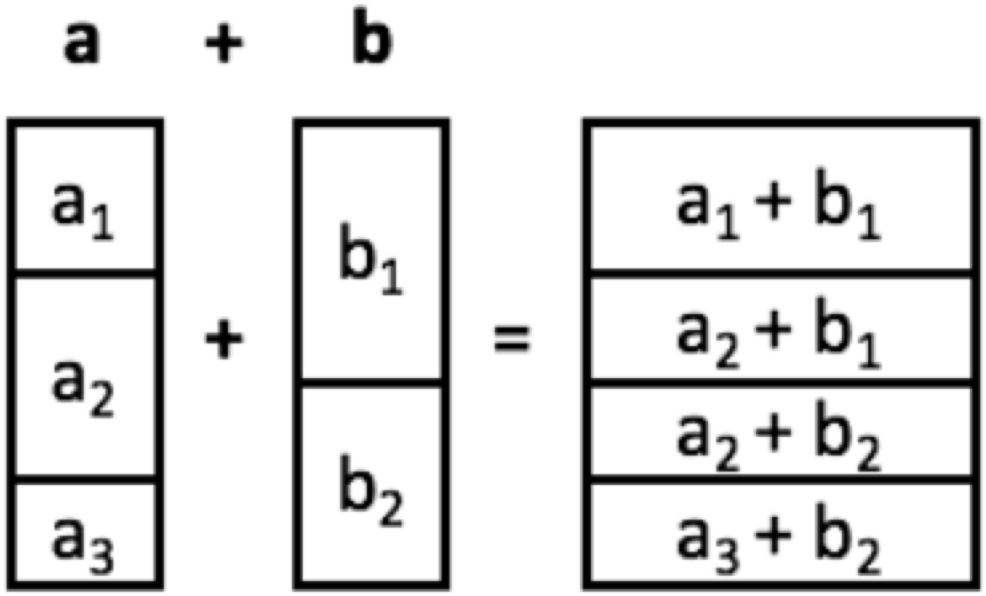
```
int32 averageIntArray(int32[] arr, int32 size) {
```

$$\text{return } \frac{\sum_{i=0}^{size} arr[i]}{size};$$

```
} averageIntArray (function)
```

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





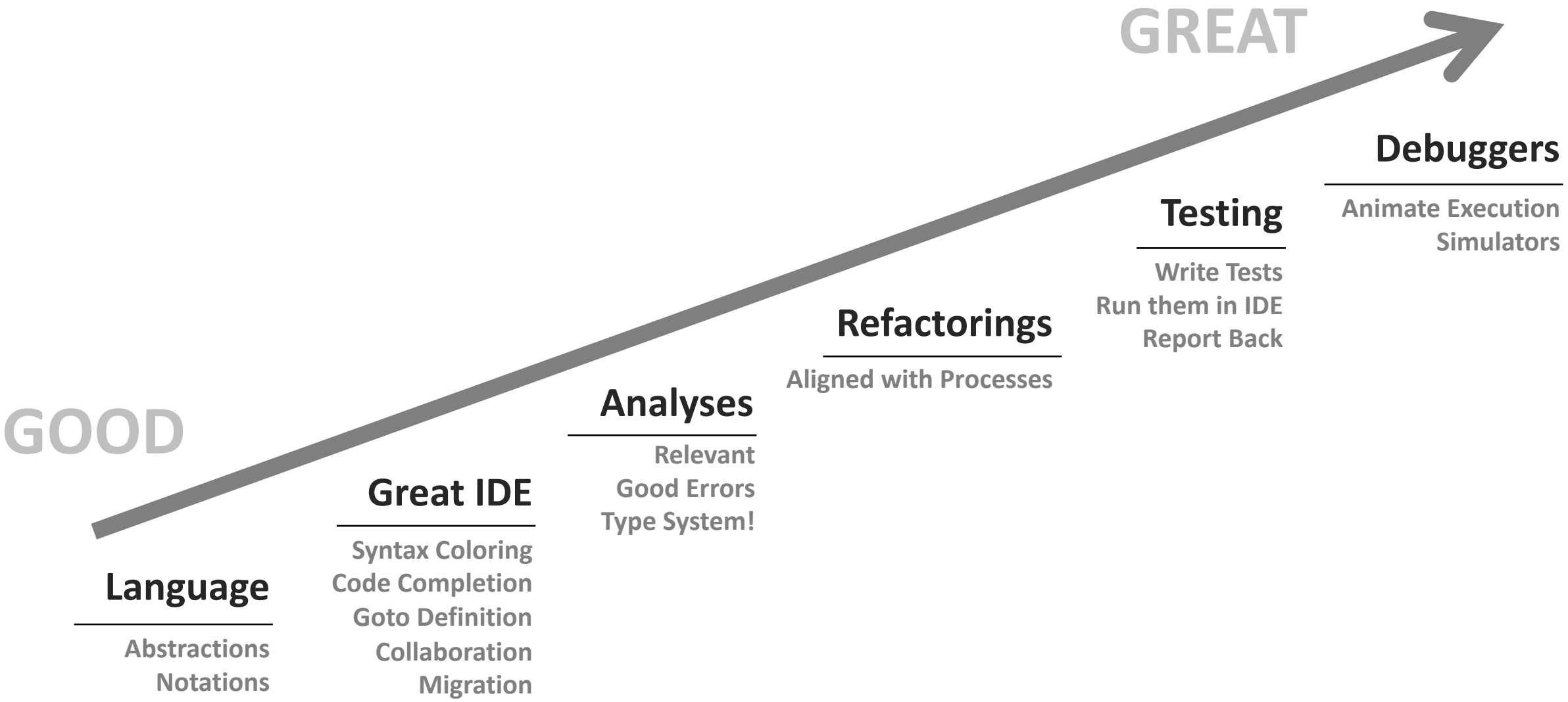
```
calculation for Tax
  depends Salary as s
    valid from /2017 01 01/
  calculate [monthly] {
    ...
  }
```

```
calculation for Tax
  depends Salary as s
          SomeOtherThing as t
    valid from /2017 07 01/
       if ctx.employment.person.homeAddress.state == BW
  calculate [monthly] { .. }
```

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

```
result data [monthly] Salary from /2017 10 01/ {
  employment -> Employment
  amount     : currency
  taxFree    : boolean
}
```

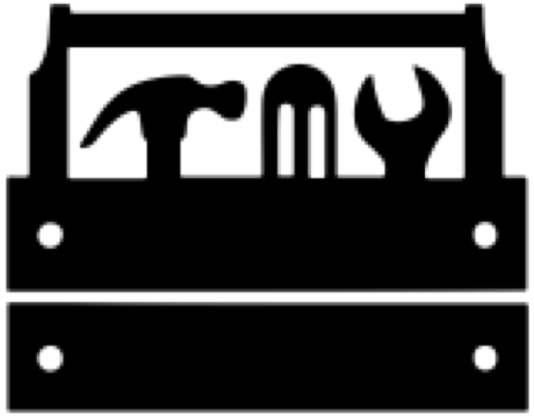
More than Language



5

Language Workbenches

DSL Process



Understand Domain

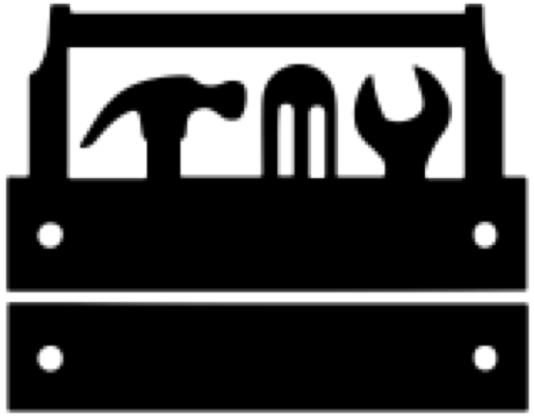
Design & Implement Language + IDE

Validate with Users

Generator/Interpreter

Deploy / Use

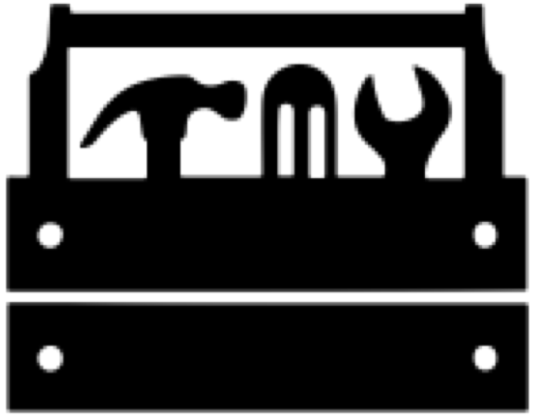
DSL Process – Meta?



Design & Implement Language + IDE

Generator/Interpreter

DSLs are the Tools

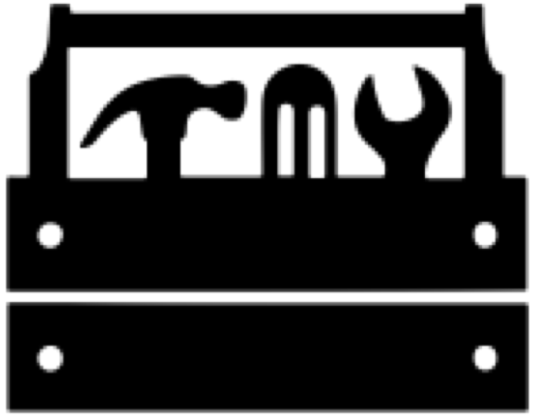


Design & Implement Language + IDE

Generator/Interpreter

Language Workbench

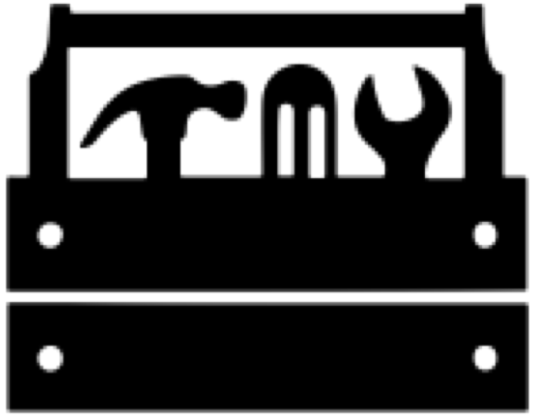
DSLs are the Tools



Tools for efficiently building eco-systems of languages and their IDEs

Language Workbench

DSLs are the Tools



Language Workbench

MPS Language Workbench

Language Workbench

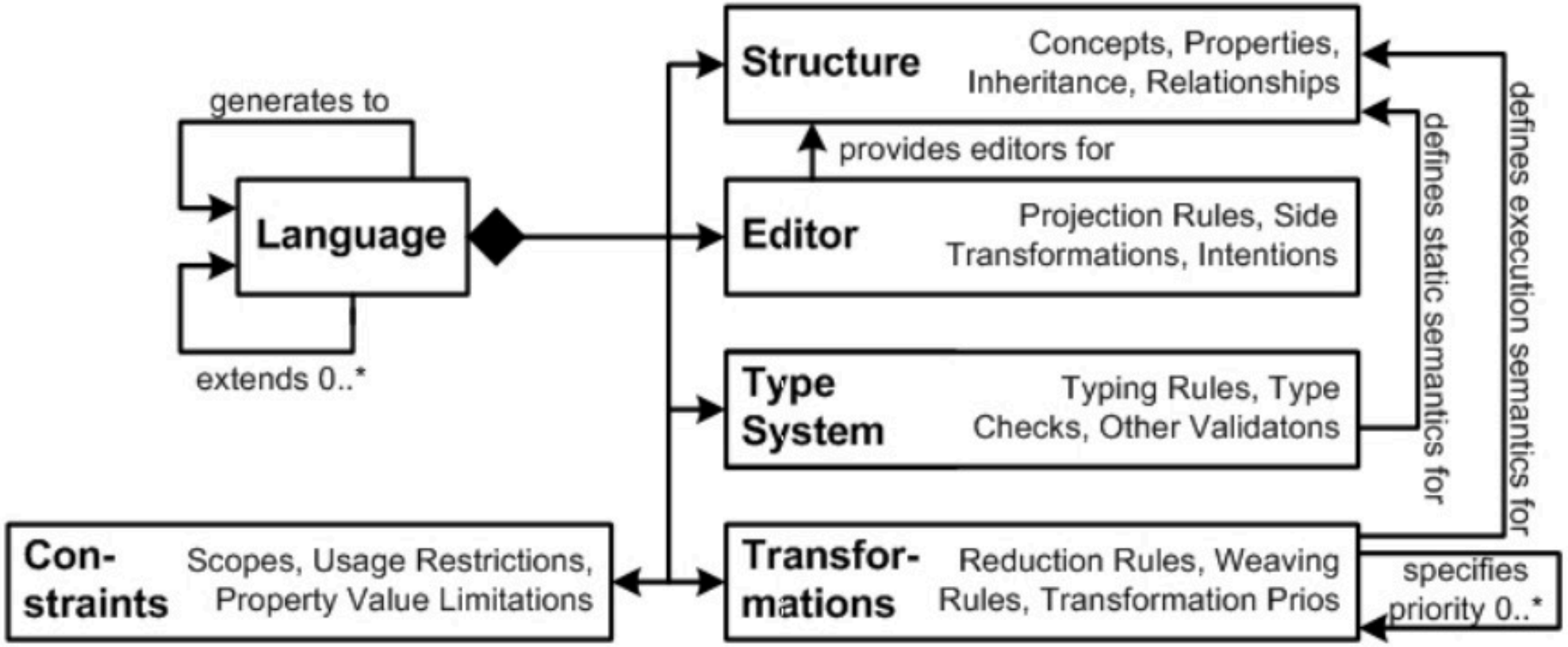
Open Source, by Jetbrains

Very Powerful

Used for years by itemis and others

Vast Experience

MPS Language Workbench



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

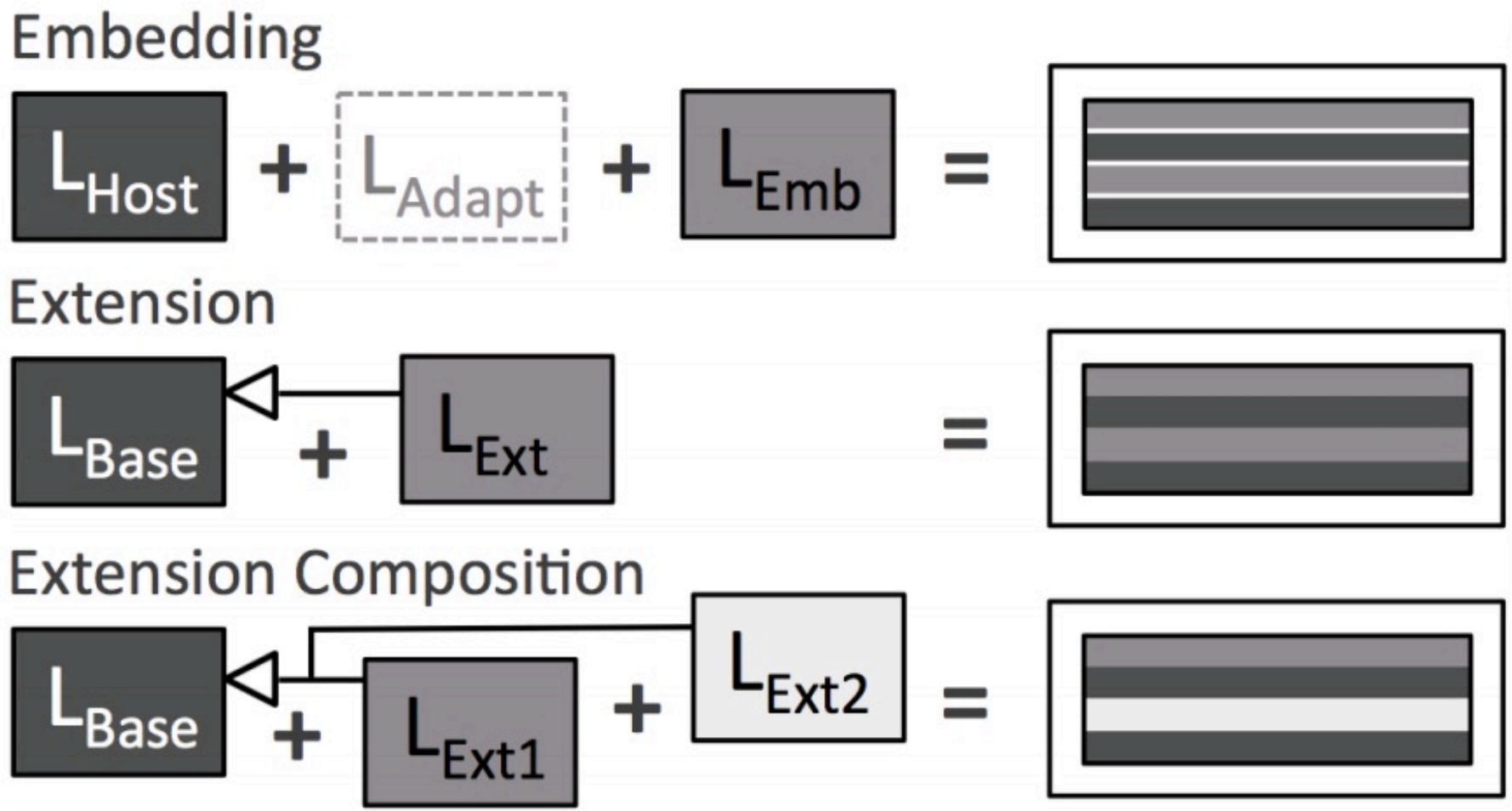
MPS: Notational Freedom

100% crucial for acceptance in domain!

Rabbit Hole

Modeling is more than diagrams!

MPS: Language Composition



SPLE on Language Level!

Other Language Workbenches

{S} spoofax	TU Delft
xtext	itemis/Typefox
Rascal	CWI Amsterdam
The Whole Platform	Solmi/Persiani
	Metacase
MontiCore	RWTH Aachen

5

The Big Picture (Revisited)

Modeling vs. Programming

Formulas, Rules
Data Structures
Tables
Values

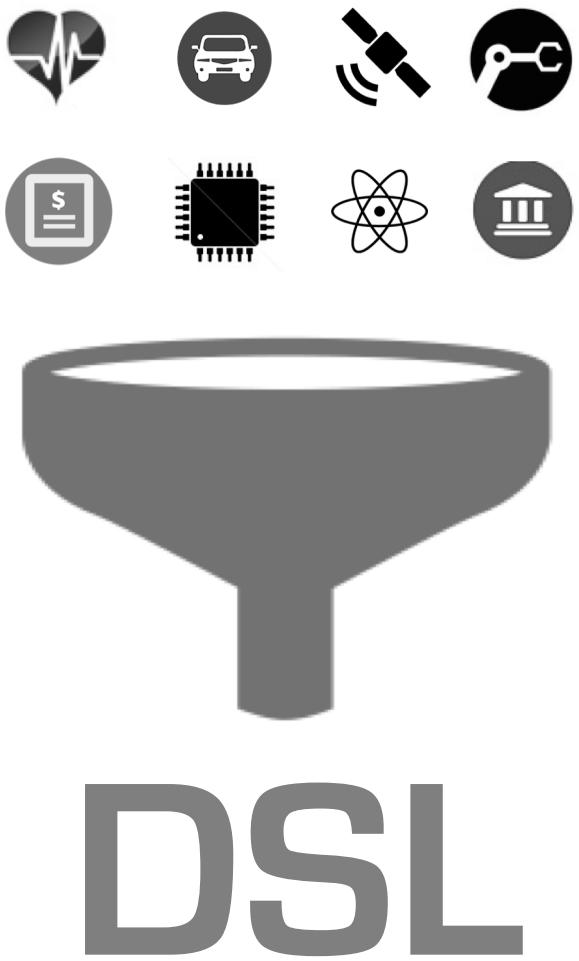
Modeling

!=

Programming

Performance
Scalability
Robustness
Deployment

DSL Aligned with the Domain



The concepts in the DSL are well **aligned** with the domain.

This makes learning the language much simpler.

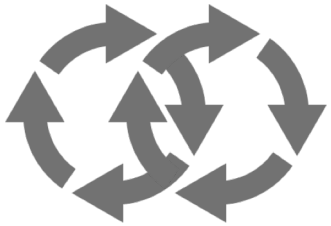
And there is tool support that helps create valid models.

DSL Aligned with the Domain

More Efficient Development

Meaningful Validation

Integrate Domain Experts

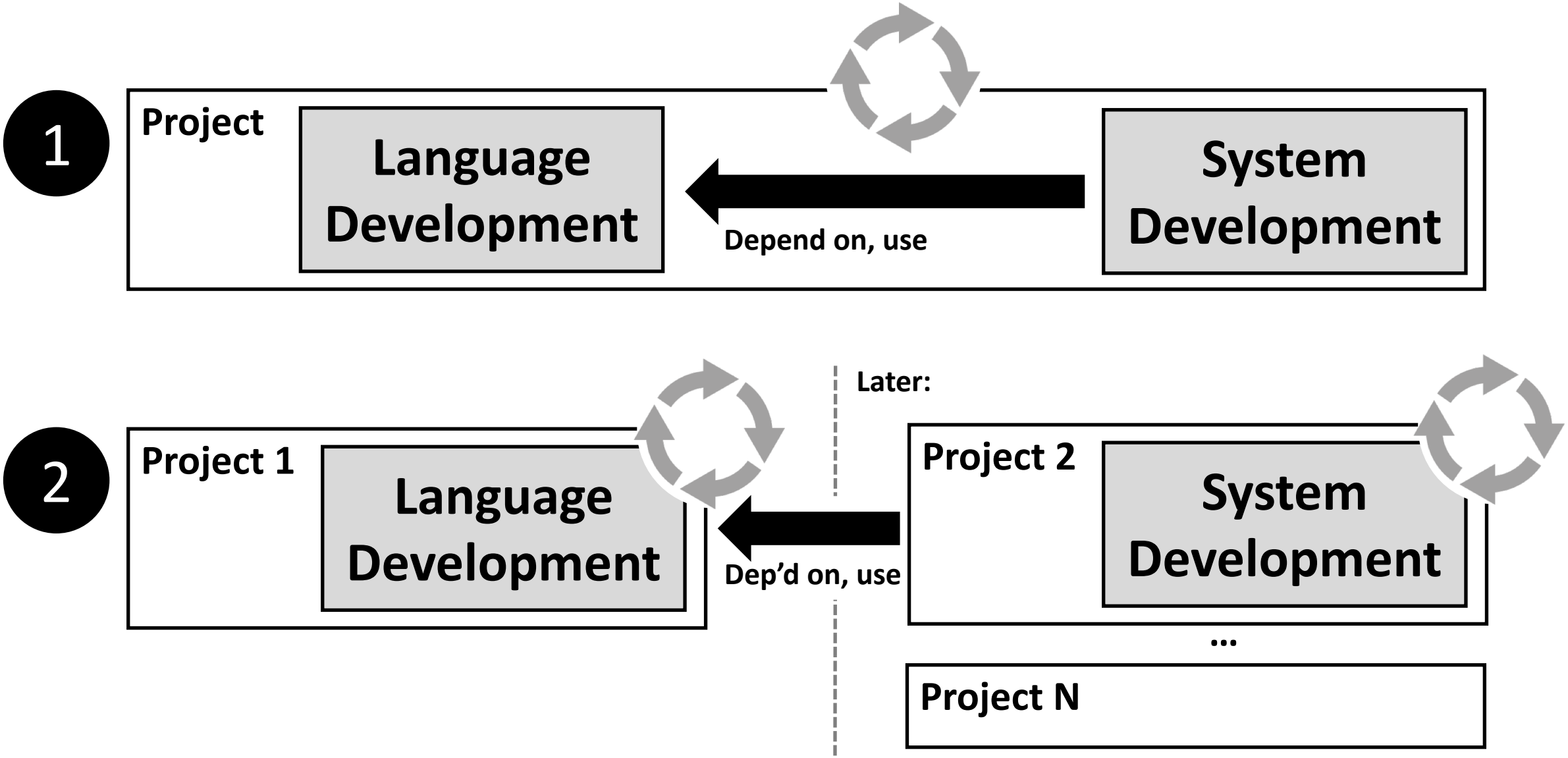


Decouple from Technology

Rabbit Hole

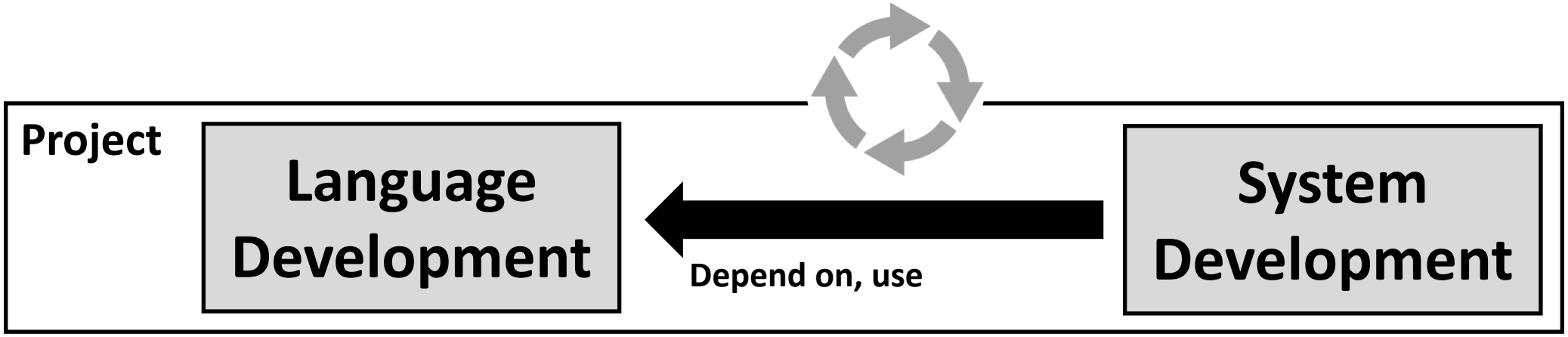
DSLs & Agile

„MD* and Agile is in Conflict.“



„MD* and Agile is in Conflict.“

1



Framework

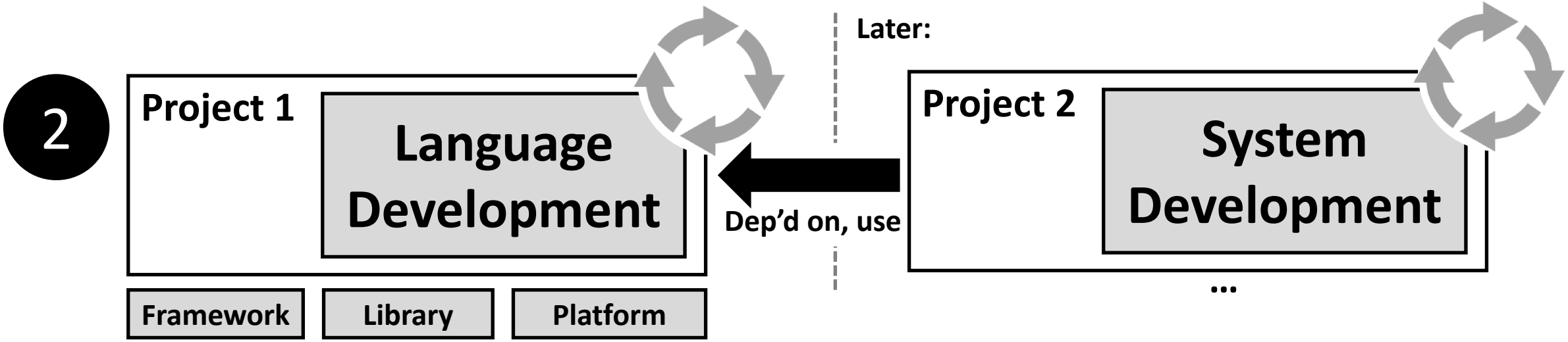
Library

Platform

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

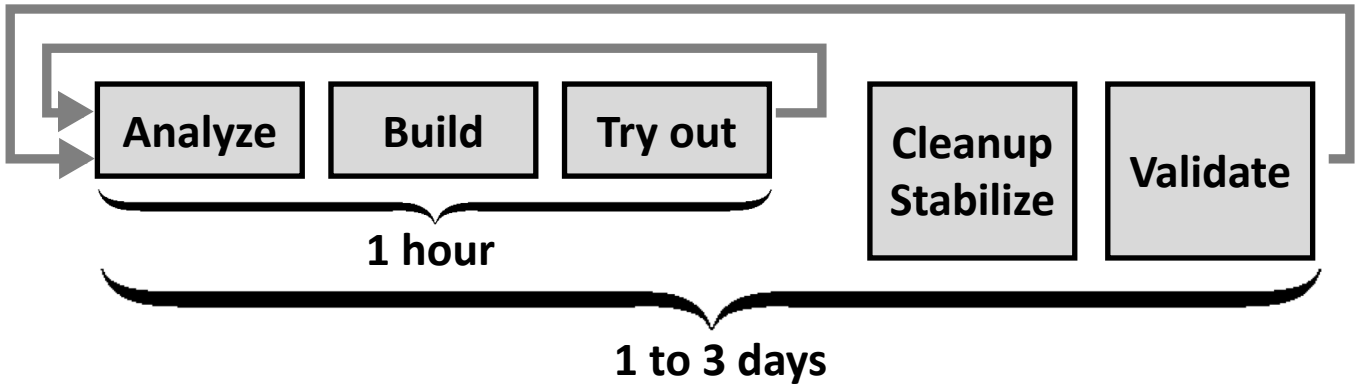
Models and DSLs are an Enabler for Agility:

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]

6

Expressing Configurative Variability with DSLs

1 Write each product from scratch

Languages are optimized for the domain
Syntax is concise, very little boilerplate
Just write every product from scratch.

[Works only for simple cases]
[Is at odds with SE best practices]

2 Use Variability Patterns in the DSL

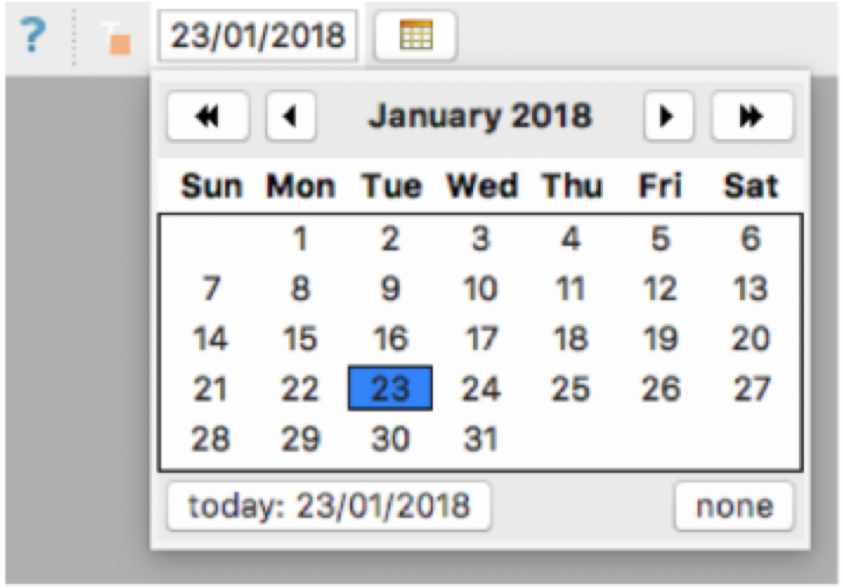
Design Variability into the Language
Similar to object-oriented programming

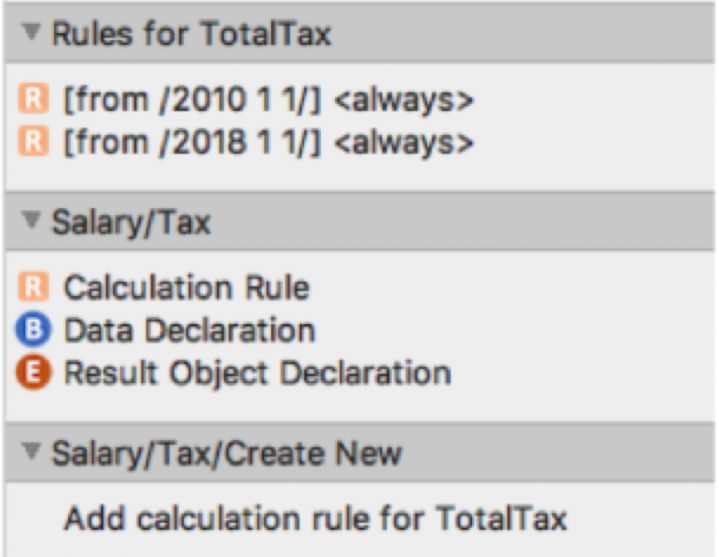
[Scales very well, esp. with verification]
[Might be harder to understand for DEs]

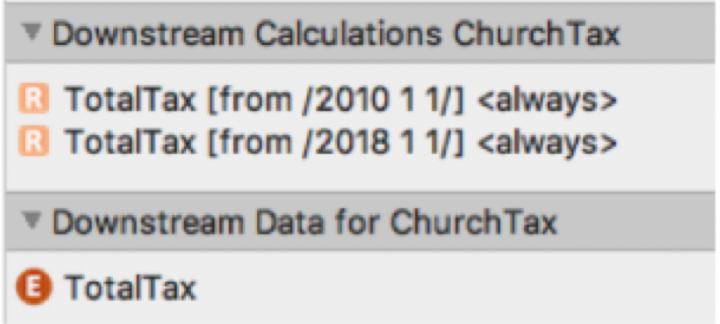
2 Use Variability Patterns in the DSL

```
calculation for Tax
  depends Salary as s
    valid from /2017 01 01/
  calculate [monthly] {
    ...
  }
```

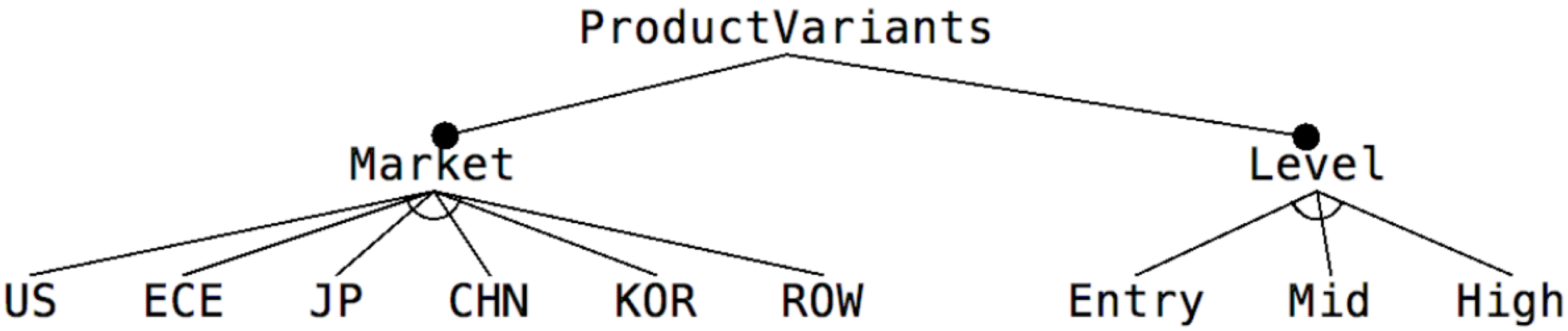
```
calculation for Tax
  depends Salary as s
          SomeOtherThing as t
    valid from /2017 07 01/
       if ctx.employment.person.homeAddress.state == BW
  calculate [monthly] { .. }
```







3 Overlay Configuration



Feature model for high-level variability

3 Overlay Configuration

```
Configuration HighPlatform for HardwareVariants
  ☑ M Processors
    ☑ M MainCPU
      ? A CPU_1GHz
      ? A CPU_3GHz
  ☑ M MassStorage
    sizeMB: 2048
    ☒ A Harddisk
    ☑ A SSD
```

Interactive, solver-backed configuration

3 Overlay Configuration

```
map_visible {
  precondition Graphics::available
  use 300
  read MapDatabasePart :
```

Feature condition	Result
fm .pf .Map2Dim	15
fm .pf .Map2_5Dim	40
fm .pf .Map3Dim	80
default:	<no default>

```
MB via Harddisk.standard
}
```

Variation points in DSL code.

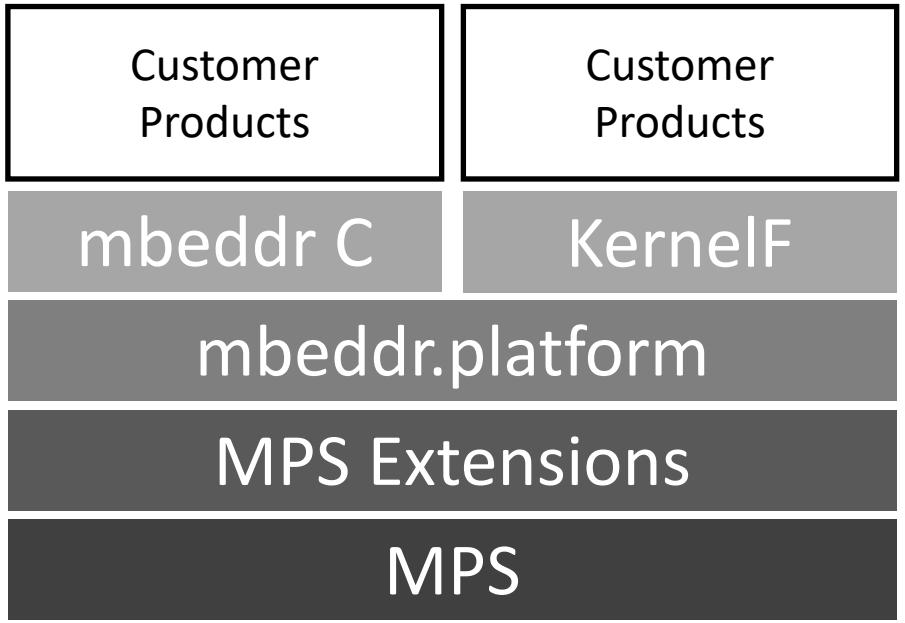
here: system modeling and performance simulation

7

SPLE on Meta Level

Projects are evil!

“Ideally each product is just a side effect of the constant evolution of the product line.”



“Don‘t just win the current game, make sure you also prepare for the next.”

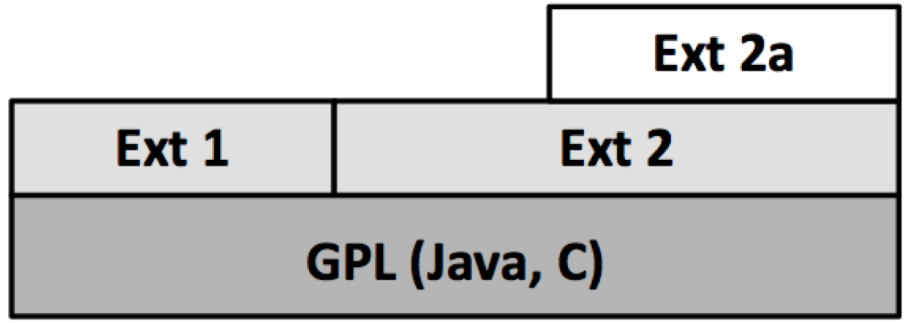
We need industry-strength language and IDE modularity!

DSL Development

GPL Extension

Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL

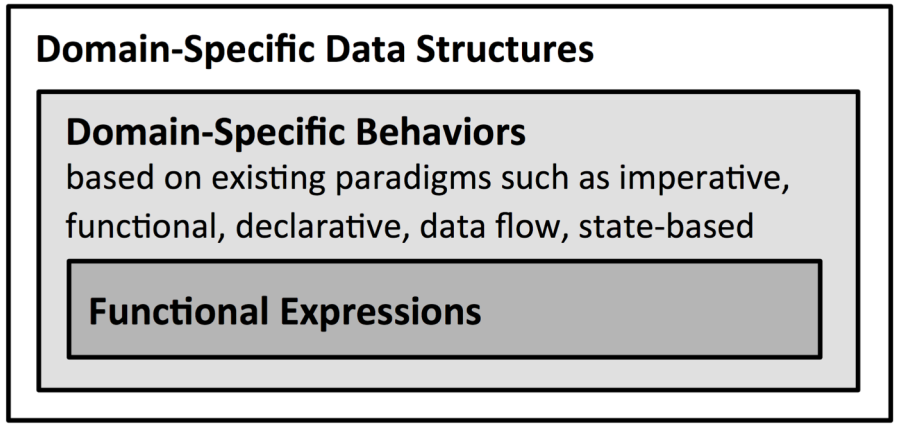
mbeddr



New Language

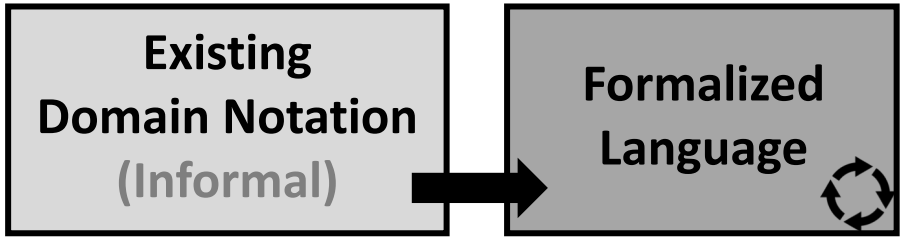
Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

KernelF



Formalization

Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“

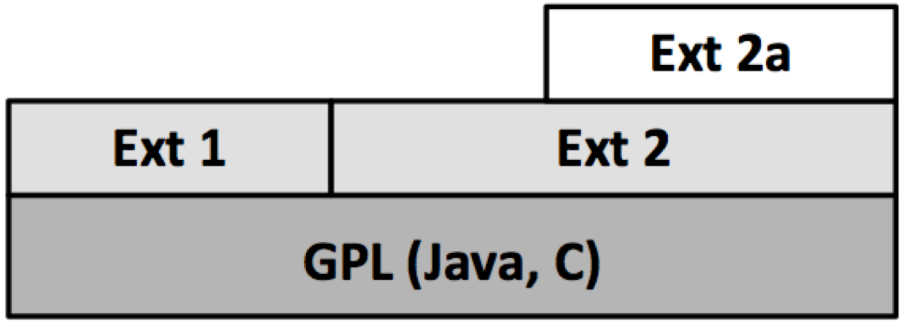


DSL Development

GPL Extension

Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL

mbeddr

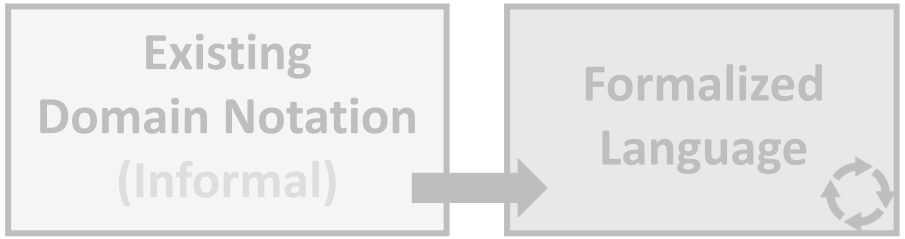


New Language

Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

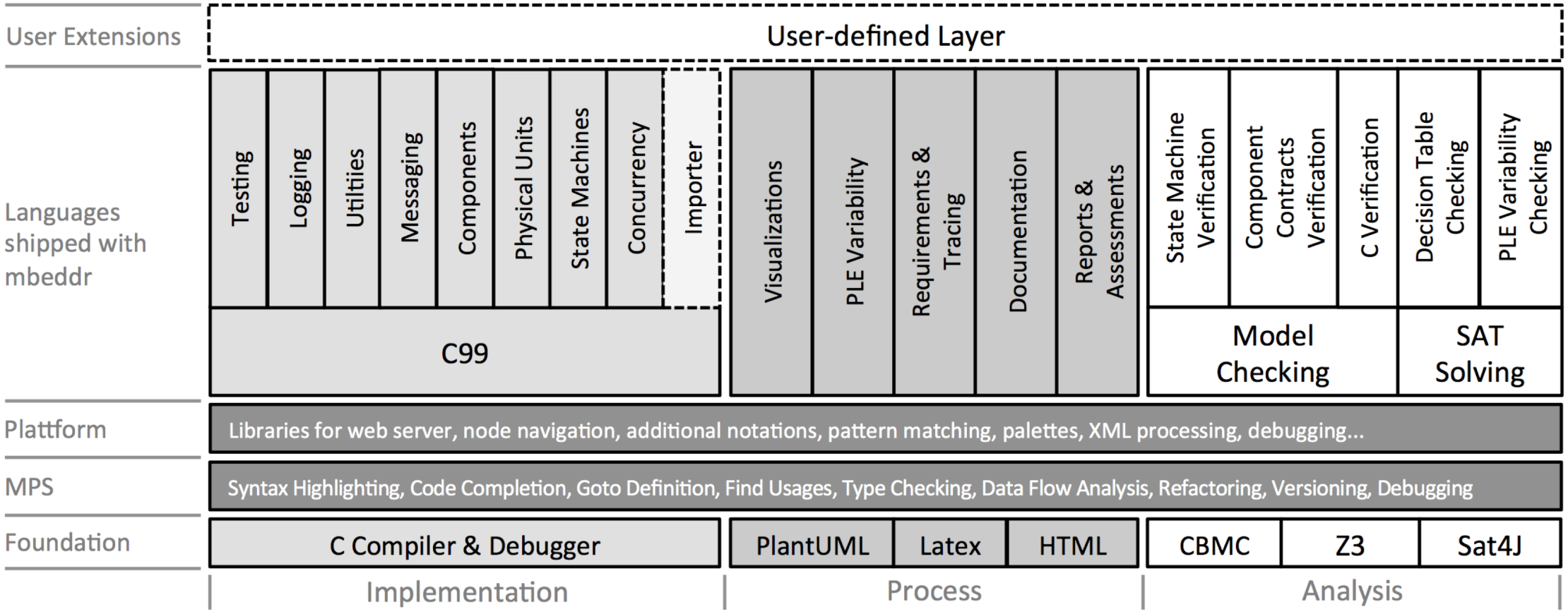
Formalization

Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“

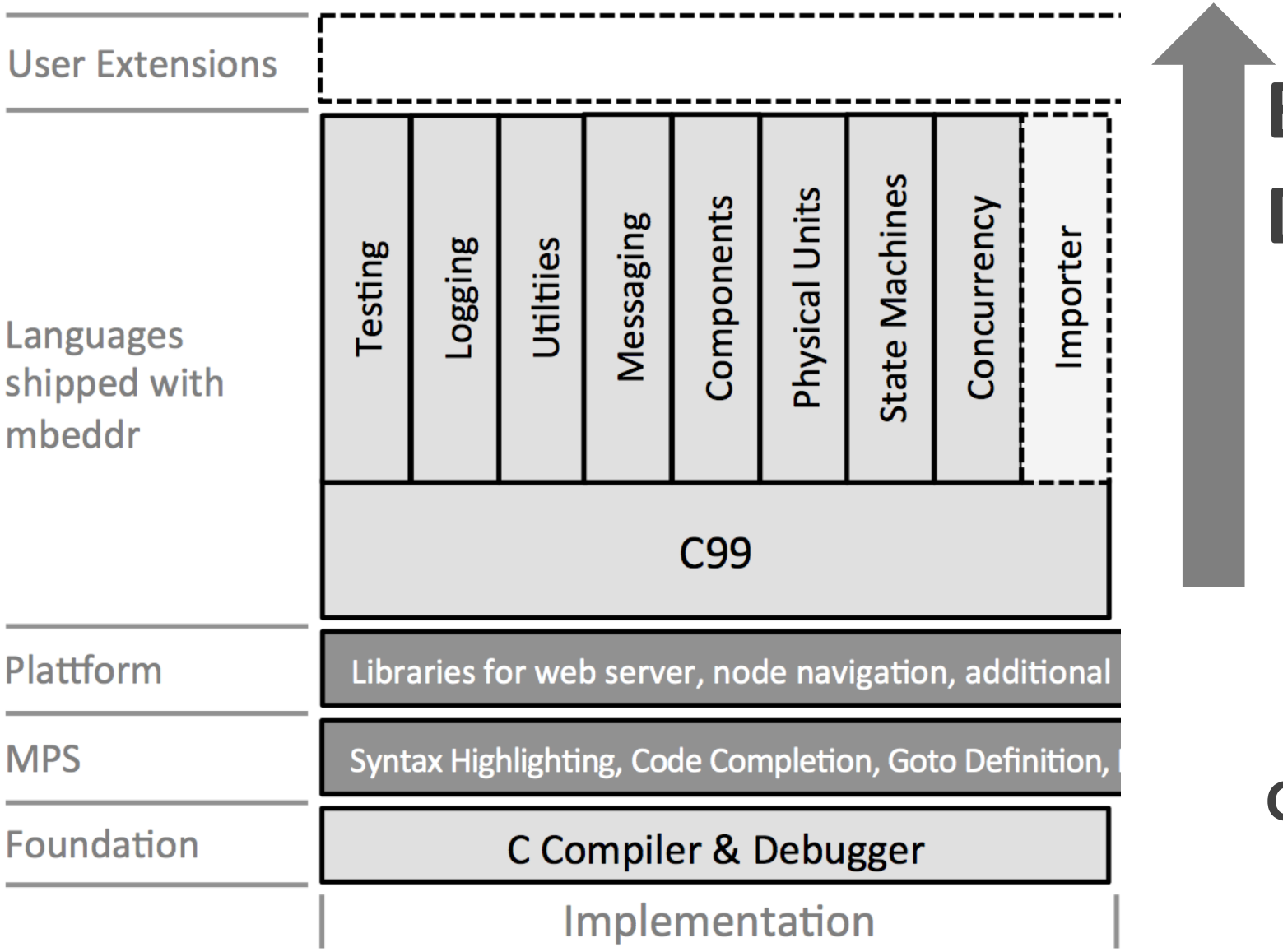


mbeddr

An extensible set of integrated languages for embedded software engineering.



mbeddr



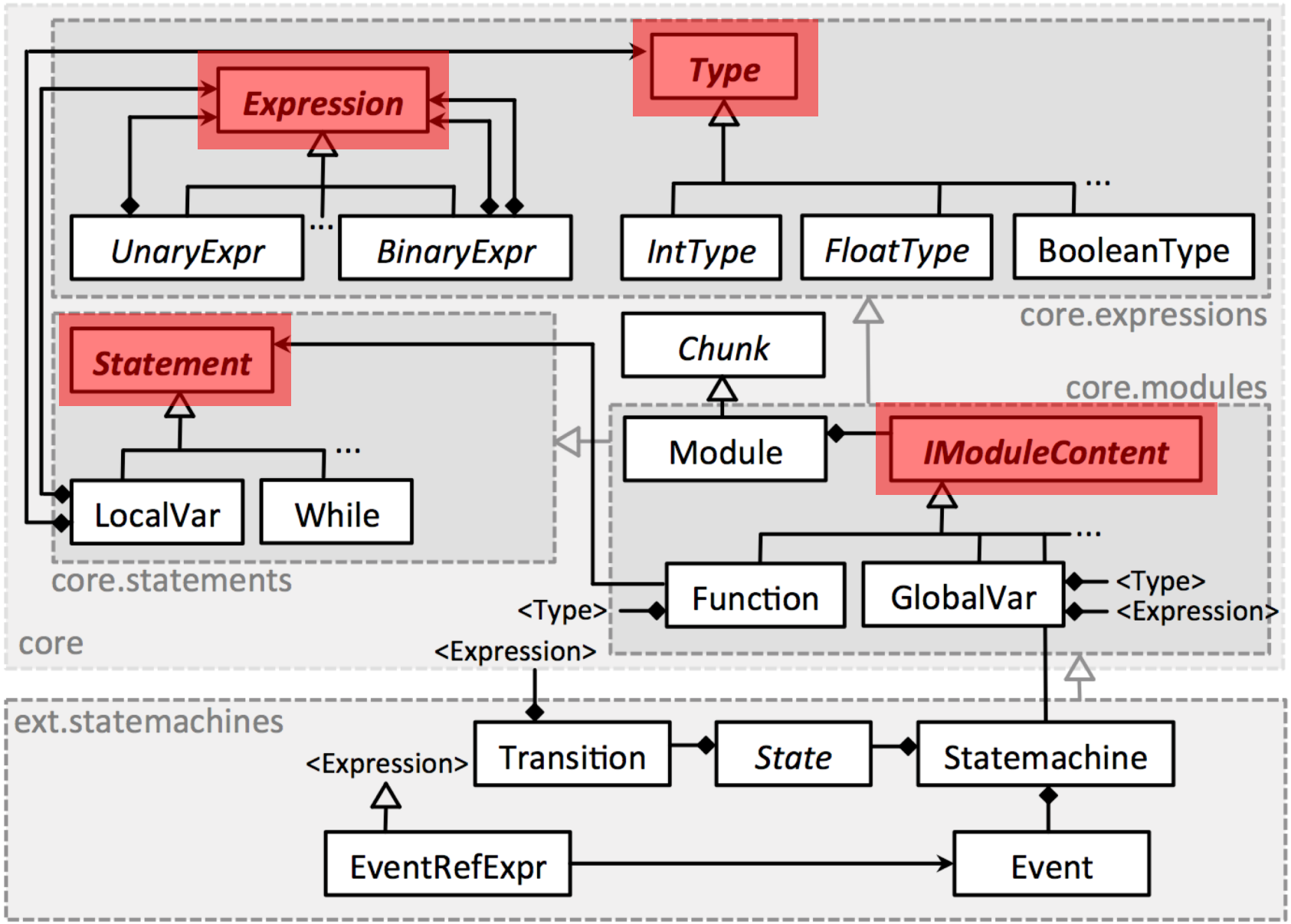
Bottom-Up Development

Growing the Language

mbeddr

- core
  - solutions
  - com.mbeddr.core.base
  - com.mbeddr.core.checks
  - com.mbeddr.core.cinterpreter
  - com.mbeddr.core.dataflow.test
  - com.mbeddr.core.expressions
  - com.mbeddr.core.legacy
  - com.mbeddr.core.make
  - com.mbeddr.core.modules
  - com.mbeddr.core.modules.gen
  - com.mbeddr.core.pointers
  - com.mbeddr.core.statements
  - com.mbeddr.core.udt

C99 itself is broken down into modules that can be reused separately.



Extension Hooks defined into the C99 language

IAssignmentLike: =, +=, *=, etc.

IFunctionLike: function, method

IBinaryLike: +-*/ **but also** . ?: !

IVoidTypeContainer: function, ftype

DSL Development

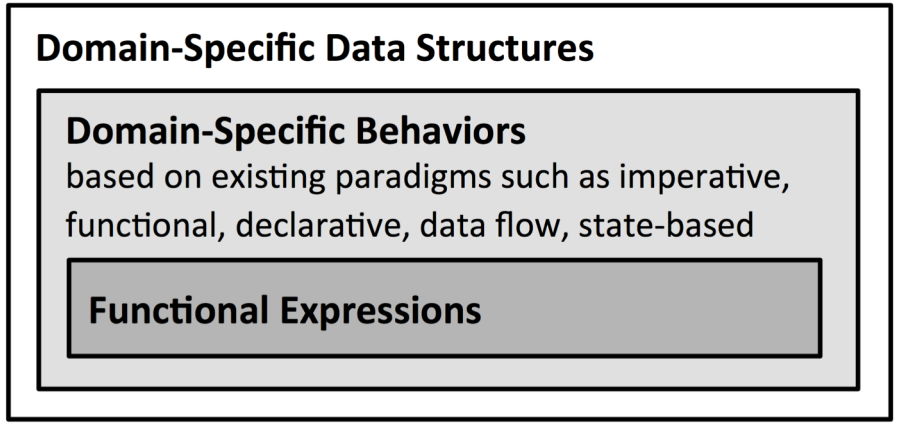
GPL Extension

Reuse GPL incl. Expressions and TS
Add/Embed DS-extensions
Compatible notational style
Reduce to GPL

New Language

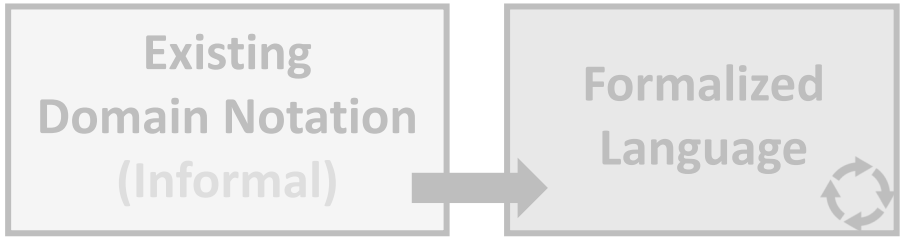
Analyze Domain to find Abstractions
Define suitable, new notations
Rely on existing behavioral paradigm
Reuse standard expression language
Interpret/Generate to one or more GPLs

KernelF

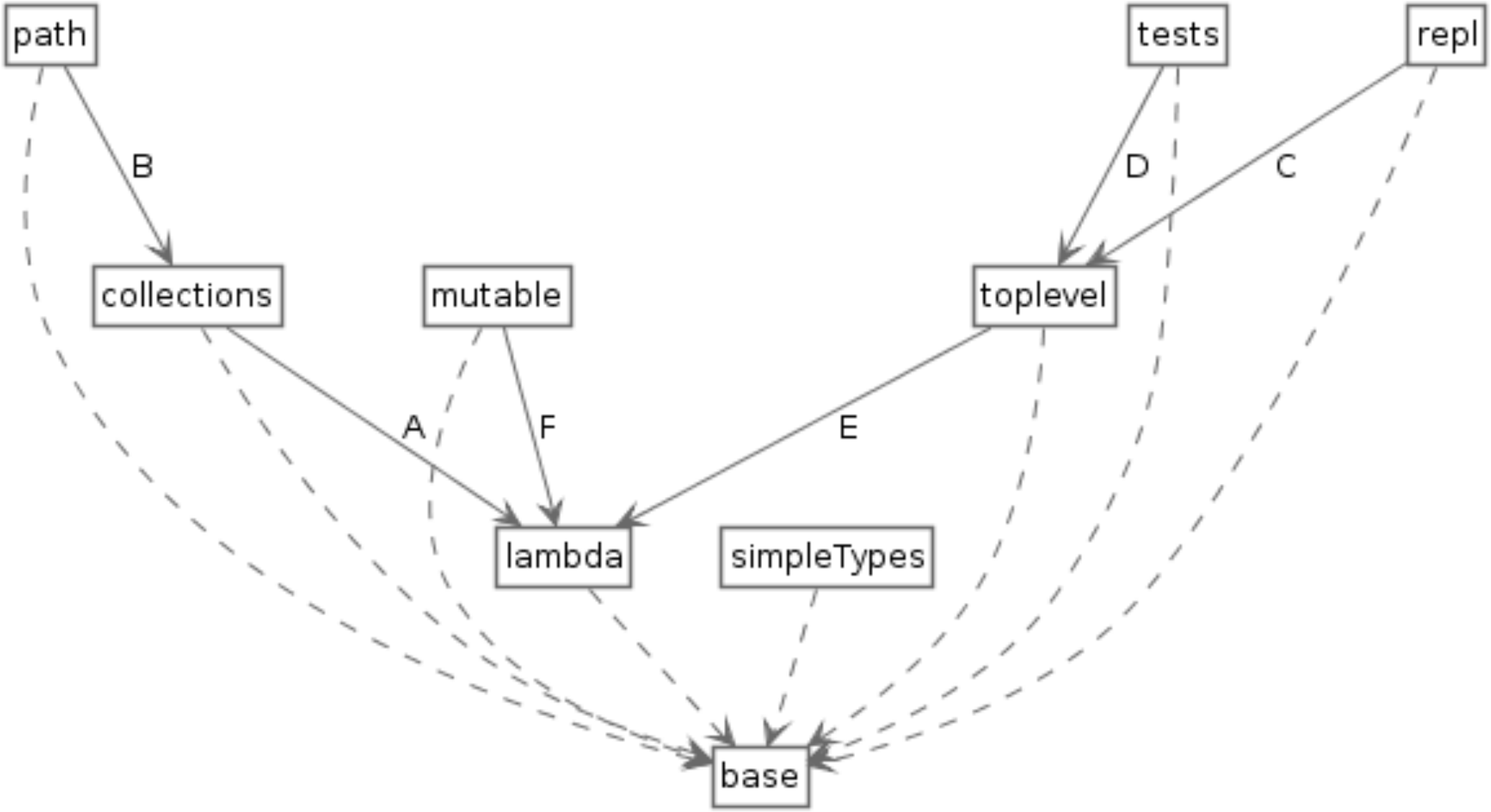


Formalization

Use existing notation from domain
Clean up and formalize
Generate/Interpret
Often import existing „models“



Modular Implementation



Abstract Concepts

```
Expression
Type
IToplevelContent
IDotTarget
```

Concept Removals

MPS Constraints can be used to effectively reduce the language.

Exchangeable Primitives

```
PTF.create<Type>()
```

Plus an extension infrastructure to contribute types.

Structure vs. Types

Types, such as `ListType` or `IRecordType` can be used for custom language concepts.

Syntax Overriding

New syntax can be defined for existing language concepts.

Extensions Overview

	Salary/Tax	Smart Contracts	Healthcare
IDE	Alternative Rules Rules for Data Items Projection for a given Date	Live Values Diff of Live Values	Simulator with Phone UI Debugger for State Machine
	Language-Specific Extensions to Functional Debugging/Tracing and Testing		
Structure	Basic Data Result Data.		Time Series Components Test Scenarios
Behaviors	Calculation Rules & Dependencies Variants Validity	Interactors & Processes State Machines & Interceptors Context Arguments Live Values	(Special) State Machines Test Generation & Mutation Coverage Measurement Utils
Functional & Types	Date Types Currency Types Temporal Types		Time & Duration Types Decision Trees and Tables Sentence-like Function Calls
KernelF			

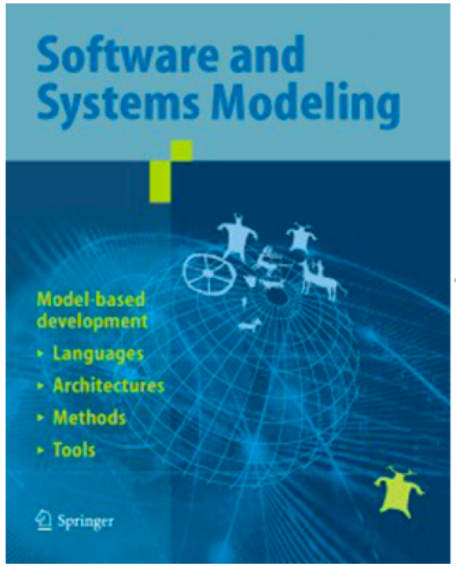
8

Wrap Up

Efforts?

≈ Size of language
Complexity of type system
Complexity of generator
# of additional tools/services

≈$^{-1}$ Capabilities of Language Workbench
Reuse of Base Langauages & FWs
Skills of dev team and domain experts
(Abstraction Potential)



Software & Systems Modeling
pp 1–24 | Cite as

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

Authors | Authors and affiliations

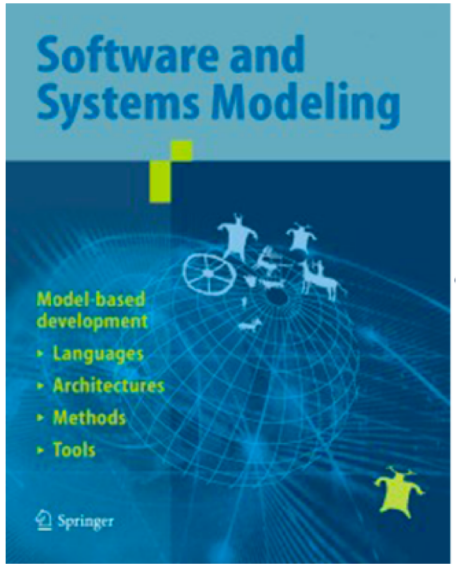
Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, Arne Nordmann

Regular Paper
First Online: 17 May 2018

Abstract

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

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



Software & Systems Modeling
pp 1–46 | Cite as

Lessons learned from developing mbeddr: a case study in language engineering with MPS

Authors Authors and affiliations

Markus Voelter, Bernd Kolb, Tamás Szabó, Daniel Ratiu, Arie van Deursen

Regular Paper
First Online: 09 January 2017

Abstract

Language workbenches are touted as a promising technology to engineer languages for use in a wide range of domains, from programming to science to business. However, not many real-world case studies exist that evaluate the suitability of language workbench technology for this task. This paper contains such a case study. In particular, we evaluate the development of mbeddr, a collection of integrated languages and language extensions built with the Jetbrains MPS language workbench. mbeddr consists of 81 languages, with their IDE support, 34 of them C extensions. The mbeddr languages use a wide variety of notations—textual, tabular, symbolic and graphical—and the C extensions are modular; new extensions can be added without changing the existing implementation of C. mbeddr's development has spanned 10 person-years so far, and the tool is used in practice and continues to be developed. This makes mbeddr a meaningful case study of non-trivial size and complexity. The evaluation is centered around five research questions: language modularity, notational freedom and projectional editing, mechanisms for managing complexity, performance and scalability issues and the consequences for the development process. We draw generally positive conclusions; language engineering with MPS is ready for real-world use. However, we also identify a number of areas for improvement in the state of the art in language engineering in general, and in MPS in particular.

http://voelter.de/data/pub/voelterEtAl2017-buildingMbeddr.pdf

SPLE is way more than configurative variability.

Not everything can be “predefined“ and „selected“.

DSLs are „platforms“ for non-config variability.

They make the construction process reusable.

Language Workbenches enable DSLs

by reducing effort to build, compose and maintain them

DSLs enable meaningful integration of domain experts

... making an important contribution to real digitalization.

This is a major contributor to business agility

Because domain experts are not just „customers“.