Wir sind aber keine Programmierer!!
Oder doch?

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

Bernd Kolb
kolb@itemis.de
www.itemis.de
@berndkolb
Nothing teaches us better than our own experiences!
3. Product Definition Languages in the Insurance Domain

2. Benefits calculation languages for governments

7+. Languages for (non-programmer) in technical domains

* Languages for use by programmers
Business Knowledge and Software
It's what makes a business tick. Distinguishes the business.

- Business Rules
- (Financial) Calculations
- Data Structures
- Mappings or Queries
- Validations
- Scientific Processes
- Contracts
- Processes
- UIs
It's what makes a business tick. Distinguishes the business.

Contributed not by developers

... but typically implemented in software
SO HOW DOES IT GET INTO THE SOFTWARE?

Contributed not by developers

... but typically implemented in software
Trends/Challenges
Complexity
Mass Customization
Reality
Reality
Reality
Goal!?
Let Business/Domain people contribute directly!

Give them expressive, productive tools to do so!
Expressivity for Core Domain Knowledge

User-Friendly Notation
Great Tool/IDE

Testing

Meaningful Analyses

Synthesis of Software
Not a software engineer. Does not care about „software stuff“ But understands the domain very well.
Not a software engineer. Does not care about „software stuff“.
But understands the domain very well.

He is a professional, not a „casual hacker“.
Language
Workbenches
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)

There's no difference!
LWBs make Languages Easier
Blur the distinction between programming and modeling.
Several different LWBs exist.
http://languageworkbenches.net
JetBrains MPS
A Language Workbench – a tool for defining, composing and using ecosystems of languages.
Open Source
Apache 2.0
http://jetbrains.com/mps
V 3.3 is current
V 3.4 to be released Summer 2016
Comprehensive Support for many aspects of Language Definition.

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

Parsing

1. Concrete Syntax
2. Abstract Syntax Tree

Projectional Editing

1. Concrete Syntax
2. Abstract Syntax Tree
[Projectional Editing]  
Syntactic Flexibility

Regular Code/Text

Mathematical

Tables

Graphical
// [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];
    }
}
aSummingFunction (function)

double midnight2(int32 a, int32 b, int32 c) {
    return \(-b + \sqrt{b^2 - \sum_{i=1}^{4} a * c}\) / 2 * a;
}
midnight2 (function)

int16 decide(int8 spd, int8 alt) {
    return spd > 0 spd > 100 otherwise 0;
    alt < 0  1    1
    alt == 0 10   20
    alt > 0  30   40
    alt > 100 50   60
}
decide (function)

Cst.Customer

Contract
starts: date
ends: date

Tariff
attributes

cust 1

trf 1
[Projectional Editing]
Language Composition

Separate Files
Type System
Transformation
Constraints

In One File
Type System
Transformation
Constraints
Syntax
IDE

50+ extensions to C
10+ extensions to requirements lang.
Projectional Editing provides syntactic flexibility and lang. extensibility.

Usability Issues are mostly solved.

MPS is great, but alternatives exist.
Most business people are able to and want to express themselves precisely!

Let’s give them the tools to do it!
Examples
# Rigid Structures

<table>
<thead>
<tr>
<th>Rule Set Type</th>
<th>DemoRuleSetType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business objects</td>
<td></td>
</tr>
<tr>
<td>person : Person</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th>Parent</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRMI : int</td>
<td>&lt;no parent&gt;</td>
<td></td>
</tr>
<tr>
<td>FR : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TT : int</td>
<td></td>
<td>Libraries</td>
</tr>
<tr>
<td>J : int</td>
<td>Standard</td>
<td>Extra</td>
</tr>
<tr>
<td>A3 : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G3 : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANUI : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X : int</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rule set DemoRuleSet2 is of type DemoRuleSetType

EU0 : int [ save false print false ]
CATEG : string [ save false print false ]
CATEG1 : double [ save true print true ]

PREMIO = [ A1 > 10 ⇒ EU0 ]
<always> ⇒ FLAG

FLAG = [ CATEG1 equals 60 or CATEG1 equals 63 or CATEG1 equals 64 ⇒ 160 ]
PREMIO equals 0 ⇒ 162
CATEG1 > 0 or substr(inga[4], 1, 1) equals "V" ⇒ 163
<always> ⇒ PREMIO + FLAG

PREMIO = [ <always> ⇒ round(PREMIO * (1 + factacer), 0) ]
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
}
**Data Contract**

proxy for Customer.Customer

<table>
<thead>
<tr>
<th>core data entity</th>
<th>BillingRegion</th>
</tr>
</thead>
<tbody>
<tr>
<td>code [key]:</td>
<td>string</td>
</tr>
<tr>
<td>name:</td>
<td>string</td>
</tr>
<tr>
<td>baseMinPrice:</td>
<td>float</td>
</tr>
<tr>
<td>maxRebateFactor:</td>
<td>float</td>
</tr>
</tbody>
</table>

**entity Contract**

| starts: date | customer: | Customer 1 | contracts 0..* |
| ends: date   | applicableTariff: | Tariff | 1 |

**entity Tariff**

| attributes: | references: |
Diagrams for Data Modeling

```
Contract
starts: date
ends: date

applicableTariff 1
attributes

Tariff

contracts 0..*

customer 1

Customer.Customer

BillingRegion

code [key]: string
name: string
baseMinPrice: float
maxRebateFactor: float
```

# Tables for Reference Data

## Core Data DefaultRegions for entity BillingRegion

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

Calculations CallCalculations for Call

| flag isLocal := magic of type boolean |
| flag isLongDistance := magic of type boolean |
| flag isRoaming := magic of type boolean |

| value cust := entity.customer |
| value pricingFactor := |\| | isLocal | isLongDistance | isRoaming | otherwise 1 |
|---------------------------|---------------------------|----------------|----------------|----------------|
| cust.isRebated            | 0.5 0.6 0.8              |
| !cust.isRebated           | 0.8 0.9 1.0              |

Here is a comment added in the gutter, just as in MS Word.
Business Rules, Math, Tooltips

Calculations CustomerBasic for Customer

```

Node: isRebated [FlagVar]
Kind: implements

1st Target: Users should be rebated
   [ Some users should get cheaper phone calls. The reasons for the rebates are outlined below. ]

flag isRebatelyActive := entity.calls.last.startTime.isOlderThan(30 day)
flag isRebated := magic of type boolean[T]
```

[A couple of statistics about the last month's activity]
```
value callsLastMonth := entity.calls.where(!it.startTime.isOlderThan(30 day))
flag activeThisMonth := !callsLastMonth.isEmpty
value devicesUsedLastMonth := callsLastMonth.select(it.sourceDevice).distinct
```
```
value totalPriceLastMonth := \sum_{i = 0}^{callsLastMonth.size} \text{callsLastMonth.at}(i).price.value
```
```
value averageCallPriceLastMonth := \frac{\text{totalPriceLastMonth}}{\text{callsLastMonth.size}}
```

[Some random examples.]
```
value example := all[Call].first.customer.calls.first.startTime
```
Tests executed in the Editor

```haskell

-- group Calculate and Test calls

flag hasEverMadeACall := !entity.callsOfCustomer.isEmpty
value amount of calls := ((hasEverMadeACall))?(entity.callsOfCustomer.size):0

tests:
  (entity := Peter M) => 0 actual: 2
  (entity := Peter M) => 2
  (entity := Hanna B) => 2 actual: 3
  (entity := Hanna B) => 3
endtests

value all calls := entity.callsOfCustomer

value discountFactor := magic of type double

value current price := \sum_{i = 0}^{\text{amount of calls} - 1} (all calls.at(i).price.value) * discountFactor

tests:
  (entity := Hanna B, discountFactor := 0.9) => 10.8
  (entity := Hanna B, discountFactor := 1.0) => 8.55 actual: 12.0
  (entity := Peter M, discountFactor := 1.0) => 0.5 actual: 4.9
  (entity := Peter M, discountFactor := 1.0) => 4.9
endtests

value averageCallPrice := current price / amount of calls

tests:
  (entity := Hanna B, discountFactor := 1.0) => 4.0
  (entity := Hanna B, discountFactor := 1.0) => 2 actual: 4.0
  (entity := Peter M, discountFactor := 1.0) => 2.45
endtests
```
contract BaseContract specializes <no baseContract> imports: << ... >>

Context Objects:
   c: Customer

[final] assign callsThisMonth
callsThisMonth := c.callsLastMonth

assign amountThisMonth
amountThisMonth := 0

[final] store storeBill
c.bills := new MonthlyBill {
   amount := amountThisMonth
}

Business Rules for Contracts

```plaintext
contract FlatrateContract specializes BaseContract imports: BusinessRequirements

Context Objects:
    c: Customer

[final] assign BaseContract.callsThisMonth
callsThisMonth := c.callsLastMonth

[final] store BaseContract.storeBill
c.bills := new MonthlyBill {
    amount := amountThisMonth
}

conditional assign overrides BaseContract.amountThisMonth as of 16/8/2014
amountThisMonth :=
    if c.isRebated then 40
    else 50

conditional assign overrides BaseContract.amountThisMonth as of 20/8/2014
amountThisMonth :=
    if c.isRebated then 40
    else 60
```
BDD-style Tests for Business Rules

```
rule checkStuff
  given anything
  when [the `customer.calls.size` is equal to 10] and
       [the `call.endTime` is smaller than 20]
  then [set `call.price` to 20]
       [execute cancelContract with `customer`]
```
<table>
<thead>
<tr>
<th>Assessment: UnusedCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>query: unused code</td>
</tr>
<tr>
<td>sorted: □ must be ok: □ hide ok ones: □</td>
</tr>
<tr>
<td>last updated: Sep 18, 2014 (3 days ago) by markusvoelter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BaseContract</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ storeBill</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CustomerBasic</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ example</td>
</tr>
<tr>
<td>□ isMale</td>
</tr>
<tr>
<td>□ activeThisMonth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FlatrateContract</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ FlatrateContract.amountThisMonth</td>
</tr>
<tr>
<td>□ FlatrateContract.amountThisMonth</td>
</tr>
</tbody>
</table>

**total 11, new 0, ok 1**
1. Initially you have no points.
   
   InitiallyNoPoints /functional: tags

   ![Add Comment] ![Add Other Data] ![Add Child Requirement] ![Add Next Requirement]

   [ When the game starts, you have no points. ]

   workpackage initial scope: 1 responsible: peter prio: 1 effort: 1 days

   ![ ]

2. Once a flight lifts off, you get 100 points.
   
   PointsForTakeoff /functional: tags

   ![Add Comment] ![Add Other Data] ![Add Child Requirement] ![Add Next Requirement]


3. The factor of points
   
   PointsFactor /functional: tags

   ![Add Comment] ![Add Other Data] ![Add Child Requirement] ![Add Next Requirement]

   [ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. ]
**Math Notations**

```cpp
vector<int16, 3> aVector = \begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix} \times 512;
```

```cpp
vector<int16, 3> resultOfCrossProduct = aVector \times aVector;
```

```cpp
matrix<int16, 2x3> aMatrix = \begin{bmatrix}
1 + 2 & 2 \times 7 & 42 \\
3 & 51 & 24
\end{bmatrix};
```

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

```cpp
int32 averageIntArray(int32[] arr, int32 size) {
    \sum_{i = 0}^{size} arr[i] \\
    return \frac{\sum_{i = 0}^{size} arr[i]}{size};
}
```

```cpp
averageIntArray (function)
```
Explorability of the Language

```c
#include <stdint.h>

int32 sumUpIntArray(int32 arr[], int32 size) {
    int32 sum = 0;
    for (int32 i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}

int32 averageIntArray(int32 arr[], int32 size) {
    int32 total = sumUpIntArray(arr, size);
    return total / size;
}
```
Points you get for each trackpoint

InFlightPoints /functional: tags


Duis tempus justo magna. Nunc lobortis libero sed eros interdum aliquet ele. It uses @req(PointsFactor) sdf @cfmod(ArchitecturalComponents) to calculate the total points.

calculation PointForATrackpoint
Points you get for each trackpoint

InFlightPoints /functional: tags


Duis tempus justo magna. Nunc lobortis libero sed eros interdum aliquet ele. It uses @req(PointsFactor) sdf @cfmod(ArchitecturalComponents) to calculate the total points.

calculation PointForATrackpoint: This rule computes the points awarded for a Trackpoint.
It does so by taking into account the @alt and the @speed passed as arguments.

```plaintext
parameters: [ int16 alt: current altitude of the trackpoint ] => (uint8 || int8 )
int16 speed: current speed of the trackpoint
```

result = (BASEPOINTS * 1) *

| speed > 180 | 30 | 15 |
| alt > 2000 | alt > 1000 | otherwise 0 |
| speed > 130 | 10 | 20 |

tests: ...
Points you get for each trackpoint

```
InFlightPoints /functional: tags

Duis tempus justo magna. Nunc loboartis libero sed eros interdum aliquet ele. It uses @req(PointFactor) sdf @cfmod(ArchitecturalComponents) to calculate the total points.
```

calculation PointForATrackpoint: This rule computes the points awarded for a Trackpoint.
It does so by taking into account the @alt and the @speed passed as arguments.

```
parameters: [ int16 alt: current altitude of the trackpoint ] => (uint8 || int8 )
int16 speed: current speed of the trackpoint
```

```
result = (BASEPOINTS * 1) *

<table>
<thead>
<tr>
<th>speed</th>
<th>&gt; 180</th>
<th>180</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt</td>
<td>&gt; 2000</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>&gt; 1000</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>98</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>97</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>93</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>92</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>89</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>86</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>85</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>84</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>83</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>78</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>77</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>76</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>74</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>71</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>63</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>61</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>56</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>55</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>48</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

tests: PointForATrackpoint(500, 46) == 0
PointForATrackpoint(500, 1200) == 0
PointForATrackpoint(1100, 165) == 200
PointForATrackpoint(2100, 140) == 100
PointForATrackpoint(2100, 200) == 300
```
### Live Tests for Business Rules

#### Points you get for each trackpoint

**InFlightPoints /functional: tags**


Duis tempus justo magna. Nunc lobortis libero sed eros interdum aliquet ele. It uses @req(PointsFactor) sdf @cfmod(ArchitecturalComponents) to calculate the total points.

**calculation** PointForATrackpoint: This rule computes the points awarded for a Trackpoint. It does so by taking into account the @alt and the @speed passed as arguments.

**parameters:**

| int16 alt: current altitude of the trackpoint |
| int16 speed: current speed of the trackpoint |

**result** = (BASEPOINTS * 1) *

<table>
<thead>
<tr>
<th>condition</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt &gt; 2000</td>
<td>1</td>
</tr>
<tr>
<td>alt &gt; 1000</td>
<td>0.5</td>
</tr>
<tr>
<td>otherwise</td>
<td>0</td>
</tr>
<tr>
<td>speed &gt; 180</td>
<td>30</td>
</tr>
<tr>
<td>speed &gt; 130</td>
<td>10</td>
</tr>
</tbody>
</table>

**tests:**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointForATrackpoint(500, 46)</td>
<td>0</td>
</tr>
<tr>
<td>PointForATrackpoint(1100, 165)</td>
<td>210</td>
</tr>
<tr>
<td>PointForATrackpoint(2100, 140)</td>
<td>100</td>
</tr>
<tr>
<td>PointForATrackpoint(2100, 200)</td>
<td>300</td>
</tr>
</tbody>
</table>
4 | Points you get for each trackpoint

*InFlightPoints* /functional: tags

Duis tempor justo magna. Nunc lobortis libero sed eros interdum aliquet ele. It uses `@req(PointFactor)` sdf
`@cfmod(ArchitecturalComponents)` to calculate the total points.

decaluation PointForATrackpoint: This rule computes the points awarded for a Trackpoint.
  It does so by taking into account the @alt and the @speed passed as arguments.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int16 alt</td>
<td>current altitude of the trackpoint</td>
</tr>
<tr>
<td>int16 speed</td>
<td>current speed of the trackpoint</td>
</tr>
</tbody>
</table>

```
result = 200
          10
          （———） * 20
          10 BASEPOINTS * 1
```

<table>
<thead>
<tr>
<th>Condition</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>true</td>
<td>15</td>
</tr>
<tr>
<td>false</td>
<td>10</td>
</tr>
<tr>
<td>true</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt &gt; 2000</td>
<td></td>
</tr>
<tr>
<td>alt &gt; 1000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
</tr>
</tbody>
</table>

**Tests:**

- `PointForATrackpoint(500, 46) == 0`
- `PointForATrackpoint(500, 1200) == 0`
- `PointForATrackpoint(1100, 165) == 200`
- `PointForATrackpoint(2100, 140) == 100`
- `PointForATrackpoint(2100, 200) == 300`
Lessons Learned
How to make People precise?
Precision

!=

Programming
Precision != Programming

{ Formulas, Rules
  Data Structures
  Tables
  Values }
Precision

Performance
Scalability
Robustness
Deployment

!=

Formulas, Rules
Data Structures
Tables
Values

Programming
Precision != Formalization

\{ Formulas, Rules, Data Structures, Tables, Values \}
Precision ≠ Formalization

- Greek Letters
- Analyses
- Proofs

Formulas, Rules
Data Structures
Tables
Values

Formalization
Benefits of being precise

Make changes to system without waiting for IT
Directly Test and Debug business knowledge
Explore Alternatives and Experiment
How to get business people to be precise

Willingness to take responsibility
Very good fit with domain
„Friendly“ Abstractions and Notations
Good Tools (see later)
Education and Training
How to get business people to be precise

Technical People: „It‘s not my job!“. (and it really isn‘t)
Business L vs. Programming L
Structure/Guid.
Notation  +
Views  Mixed
IDE/Tool  *
Learn/Effective  L

- Text
1 Powerful
Business oriented languages are very different from what we have learned about languages for developers. LWBs let you build such languages.
Language Workbenches enable developers to build really expressive tools for business people to work with data effectively.
A hybrid of many worlds
Expressions
Code Completion
Syntax Highlighting
Error Markup
Version Control
Refactoring
Debugging
Scalability
Code Reviews

Languages/IDEs
Abstraction Levels
Multiple Abstractions
Multiple Notations
Languages/IDEs

Applications/Forms

Modeling Tools

Languages/IDEs

Process-Orienta8on
Rigid Structures
Visualizations
Tree Views
Guidance
Buttons

Modeling Tools

Applications/Forms

Languages/IDEs
Live Execution
„Visible Computation“
Document-Oriented

- Languages
- IDEs
- Applications
- Forms
- Modeling Tools
- Spreadsheets
- Languages/IDEs
- Wikis
- Applications
- Forms
- Wikis
Why Version Control
Why Version Control

Consistency across Team
Why Version Control

Consistency across Team Development History
Why Version Control

Consistency across Team Development History

Time Machine
Why Version Control

Consistency across Team Development History
Time Machine Branching (Feature, Version)
Why Version Control

Consistency across Team Development History
Time Machine
Branching (Feature, Version)
Support Staging
Use Staging
Change

Test System
For Business People
Real-Like Data
May Have Bugs

Production System
Live for Customers
Real Data
Mission Critical

Integration Tests
Simulations
Reviews
How do you achieve Consistency
Strict Language
Cross-References
Modularization and Reuse

Automatic Derivation based on rules (transformation, generation)
Common Repository
Version Control System
Periodic, Global Checks/Reports
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
Great IDE

Analyses

Refactorings

Testing

Debuggers

Good

GREAT

Language

Great IDE

Syntax Coloring
Code Completion
Goto Definition

Abstractions
Notations

Good Errors

Relevant
Aligned with Processes

Write Tests
Run them
Report Back

Animate Execution
Simulators
Requirements on the tool
Be a great LWB obviously

Support all the language goodness we talked about so far.
Productivity

Quickly evolve the language as the (understanding of) domain changes.
Performance

Nobody wants to work with a sluggish tool
Scalability

Non-trivial languages and significant model sizes
Evolution Support

Migrate existing models as the languages evolve.
Friendliness

Don’t overwhelm end users with too much „cruft“
Explorability

Ensure the language can be explored

```c
int32 sumArr(int32[] arr, int32 size) {
    size
    return \[\sum_{i=0}^{\text{size}} \text{arr}[i]\];
} sumUpIntArray (function)
```
A tool is not enough
Methodology = Process + Tool (+ Metrics)
Precision/Consistency refers to Artifacts and not to a rigid Process.
Discipline: do the right thing.

Define what is „right“
Force People?
Tool should makes the right thing easy.
Error Messages
Process-Guidance in the tool
Checklists to finish manual processes
Tool must fit the process!

Tool should makes the right thing easy.
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.
This is a threat!
Precision and Formality
Different Processes
Higher Efficiency

-> New Skills
-> Role Change
-> Job Loss

Automation
Focus on Engineering
Empower Business Ppl

-> Job Loss
-> Role Change
-> Less Importance
Some people are afraid of this. Take them seriously.
A change of culture that must be managed!
We tried it before, and it failed.
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!!

MDA
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.
How is this not an EDM?
Requires coordination with the whole enterprise – never works.
Language Modularization, Composition and Extensions

Narrow interfaces between languages (and between the models build with the languages)

Delayed global consistency checks (in contrast to local, eager checks)

Can be limited to one or more subdomains
Why now?
What has changed?
Complexity rises, time to market reduces, variability increases.

What is the alternative?

Tools have gotten better in terms of flexibility, usability, scalability.

It seems realistic now.
Contraindications
No structure in domain
  -> language would be too low level

No availability of domain experts
  -> cannot retrieve knowledge for building the language

No resources available
  -> initially it will be additional work...

Immature Organization
  -> never heard of unit test, CI and VCS? Bad sign!
How do you introduce this?
YOU NEVER KNOW HOW STRONG YOU ARE... UNTIL BEING STRONG IS THE ONLY CHOICE YOU HAVE.
1 Agree this is the right way

Self-Learning and considering alternatives
Consulting & Look at relevant similar cases
Analysis of your own situation

2 Prototype it

Possibly with external help to learn tool and guide
Small but meaningful sub problem
Evaluate Approach and tools
Integrate Stake Holders -> Sales Job!

3 Go for the real thing

See next slides.
Create a dedicated team/organization whose goal it is to be successful with the approach.

Decouple from Daily Business.

Staff with people who are driven, open to change and good communicators.
Introducing the Approach

Step by Step 1
Vertical Slice through Domain, then expand

Step by Step 2
Increasing Levels of Formality
- Prose
- Prose + Glossary
- Prose + Glossary + Calculation Rules + Code Generation
...
Introducing the Approach

Step by Step 1
Vertical Slice through Domain, then expand

Step by Step 2
Increasing Levels of Formality

Keep the end goal (formalization, automation) in sight, otherwise it is hard to justify „strange tools“ as opposed to a Wiki, e.g.
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 variability
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 variability

Formale Beschreibung

Funktion: berbwvekFF
Programmquelle: vmscf1a1.c
Produkt-Typ: FONDS
PK-Typ: KAPITAL-KONTO

verwendete Attribute:
- lkm_akt_param
- lkm_faell_param
- ber_zweck_param
- kz_rzw_param

bwvvek

aufgerufene Funktionen:
- berbwinezilFF
- VTRKermbtgfaeilFF

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 je- weils die Monatsbeginne, also lkm_akt_param -1 bis lkm_faell_param – 1.

Der Parameter kz_rzw_param steuert die zu berücksichtigende Zahlweise des Zahlungsstroms. Mög- lich sind zur Zeit nur die Ausprägungen 0 (Zahlungen zu den Beitragsfällen) und 12 (monatliche Zahlungsweise).

Schleife über lkm_faell_hilf = lkm_akt_param bis lkm_faell_param

1. Falls kz_rzw_param = 12
   - kz_fh_hilf = 1
2. sonst
   - kz_fh_hilf = VTRKermbtgfaelFF(lkm_faell_hilf)

Ende Falls kz_rzw_param = 12
bwvek = bwvek
   + kz_fh_hilf * berbwinezilFF(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 variability
A real Example: Legacy

Specify/Program

Write formal code in a DSL mixed with tables and text

Printed, PDF

No tool support whatsoever
No testing (except inspection)

No reuse
No modularity
No variability
A real Example: Legacy

Specify/Program

Insurance Programs

„Pixelcrap“

C Code

Write formal code in a DSL mixed with tables and text

Printed, PDF

Developer reads „spec“
Very idiomatic implementation

No tool support whatsoever
No testing (except inspection)

Dev acts as a human compiler and implements it in C

No reuse
No modularity
No variability
A real Example: Legacy

Specify/Program

Insurance Programs ➔ „Pixelcrap“ ➔ C Code

Debug

Write formal code in a DSL mixed with tables and text
No tool support whatsoever
No testing (except inspection)
No reuse
No modularity
No variability

Printed, PDF

Developer reads „spec“
Very idiomatic implementation
Dev acts as a human compiler and implements it in C

Debugging directly in C
Search-for-use by text search
Don’t trust the documents – may be outdated!
A real Example: Current

Specify/Program/Test/Debug

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 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

Formales Modell berbwekeFF

Formale Beschreibung

Funktion: berbwekeFF
Programmquelle: vsctfa1.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
- VTRKernmbtgfaellFF (a)
- BerweinzelFF (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 Zahlzeit des Zahlungsstroms. Möglicherweise sind zur Zeit nur die Ausprägungen 0 (Zahlungszeitpunkt) und 12 (monatliche Zahlungsweise).

Hilfsvariablen
- kz_br_hilf

Verarbeitungen

Schleife über lkm_faell_hilf = lkm_akt_param bis lkm_faell_param

- Falls @lkm_rzw_param = 12
  - kz_br_hilf = 1
  - sonst
  - kz_br_hilf = VTRKernmbtgfaellFF (lkm_faell_hilf)

- Ende Schleife

bwvek = bwvek + kz_br_hilf * BerweinzelFF (lkm_akt_param, lkm_faell_hilf - 1, ber_zweck_param)

- Ende Schleife

return bwvek
### A real Example: Current 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

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

Specify/Program/Test/Debug

M3
Insurance Programs

Generate

C Code

Write formal code in a DSL
mixed with tables and text

Now with IDE support and executable tests

Exactly the same C code.

The same notation!
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
Why is this an initiative by engineers?
Business people don’t feel the pain

-> the developers find inconsistencies and problems

They don’t necessarily know the ways to solve the problem

-> don’t have the ideas of how to do it better

And by the way:
We know many organizations where the business people want to be involved more directly, but the technical people don’t know how to do it.
Is this the next legacy system?
Today’s software is tomorrow’s legacy system.

Or is it?
Today’s software is tomorrow’s legacy system.

Business change is hard
Technology change is hard
Separation of Concerns
Keep BL free of technology
Make it „portable“
Language Tech 1

V1

Generator 1

Runtime T 1
Existing models become incompatible 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.
Grass Root Excel/Access
Various Departments interact with an opaque central IT system
Departments may build their own ad-hoc tools with Excel and Access
Possibly connected to CITS
Now: central IT system is an LWB with a couple of Languages
Department-specific functionality realized as language extensions
Non-CITS-connected systems remain as they are.

Department-specific functionality realized as language extensions

B1 only visible to users from department B
B1 potentially developed by devs associated with B
5

Drawbacks
You need inhouse expertise for language engineering

or a very close and trusted vendor who does it for you.
You will bind yourself to a particular tool.

You can easily export models, but no portability for language definitions.
If you use this approach for real, you should have language engineering expertise in house.
Summary
Expressivity for Core Domain Knowledge
Build Language for Domain!

User-Friendly Notation

Great Tool/IDE
You’ve seen the demos.

Testing
An integrated DSL for testing.

Meaningful Analyses
Types, Consistency, Checking

Synthesis of Software Code Generation.
Fundamentally still manual, no AI.
But much better tooling.
Become Language Engineering Experts.
Focus on architecture & technology, engineering
If you have to build a business app, consider using an LWB as the foundation, and recasting the "application" as a set of languages.
[open source]
Thank you!

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