Representing Software using a Domain Workbench (is software same as programming?)

Charles Simonyi
CEO Intentional Software
February 2008

Summary

- Software and its problems
- Advantages of Input + Process models of representation
- Role of Domain Experts
- Domain Workbench as the missing tool
- Design and use of Domain Workbench

What is Software?

- Represents solutions in domains
- Represented by (High Level) program that runs on the computer

Why is this important?

- Software is “optimal magic”: 10^10 times “better” than human action
- But:
  - It is hard to create: it is “complex”
  - Progress toward better software creation has been slow

Software progress?

1963

```
comment complex 2nd order equation-8th October 1963;
begin comment: SECRETARY - October 1963;
integer pagecount, linecount, job no, day, month, year, drum;

procedure outpage; outline(100);

procedure outline(a);
value a; integer a;
begin
if linecount-6 < a then a := linecount+2;
linecount := linecount-a;
for a := a-1 step -1 until 0 do outcr;

if linecount < 0 then begin
pagecount := pagecount+1;
linecount := linecount+64;
if pagecount > 1 then
begin outsp(32); output(⊲-ddd⊳,-pagecount,outtext(⊲<-⊳)) end;
outtext(⊲<
⊳);
end of linecount<0;
end of outline procedure;

procedure tape feed(n);
value n; integer n;
for n := n step -1 until 0 do outchar(63);

drum := drumplace;
linecount := 0;
tape feed(30); outclear;

start:
drumplace := drum;
pagecount := 0;
job no := inone;
if job no < 0 then goto finis;
input(day,month,year);
tape feed(30); outpage;

comment end of the first part of SECRETARY,
        USERs PROGRAM (see next page) IS INSERTED HERE;
```

2008

```
public CodeTable()
{
   rgcod = new ArrayList();
}
public ArrayList rgcod;

public void Pass4(XCOD xcod, int i, NTE nte)
{
   Console.WriteLine("P4: " + xcod.ToString());
   this.rgcod.Add(new MICOP(xcod, i, nte));
}

public MICOP MicopLast()
{
   return (MICOP)this.rgcod[this.rgcod.Count - 1];
}
public void DeleteLastMicop()
{
   this.rgcod.RemoveAt(this.rgcod.Count - 1);
}

public void Px()
{
   Console.WriteLine("Produced code");
   int i = 0;
   foreach(Micop micop in this.rgcod)
   {
      Console.WriteLine("{0,4}\t{1,-14}\t{2}\t{3}",
         i++,
         micop.xcod.ToString(),
         micop.i);
      if (micop.nte != null)
         Console.WriteLine(micop.nte.ToString());
   }

}
```

Horror stories: FBI “Virtual Case File”

- 4Y, $170M
- 700,000 lines, all wasted
  - $200 per line even if successful
  - Domain facts can not be recovered

Horror stories in Europe

Large Life Insurer

- 5 years to complete “generic product model”
- Currently 1 plan implemented, €25M+ spent
- 100+ plans to go

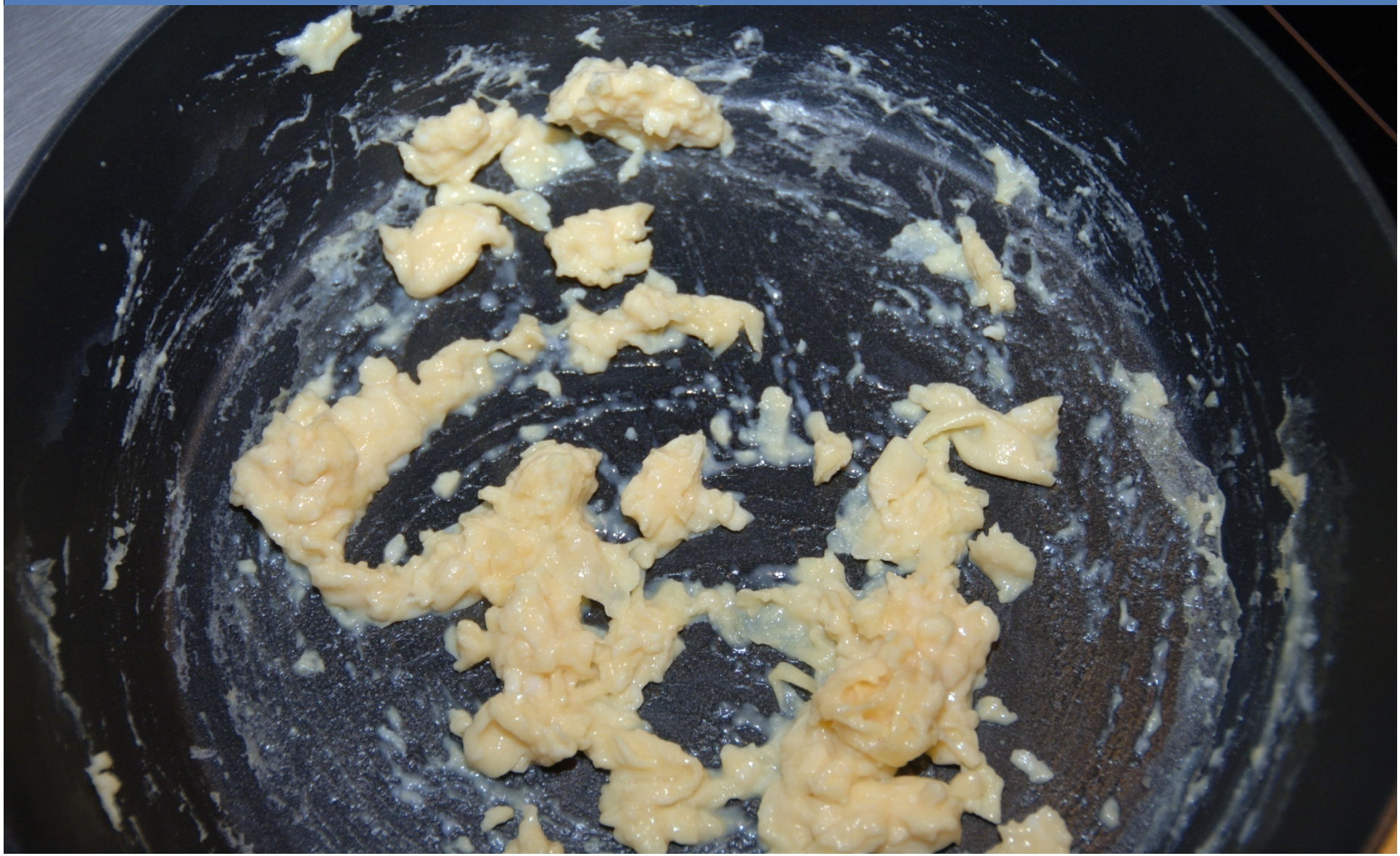
Large Pension Fund Company

- Recently acquired another Pension Fund Company
- Need to finish IT integration in 6 months
- IT integration estimated 70+ experts for 3 years

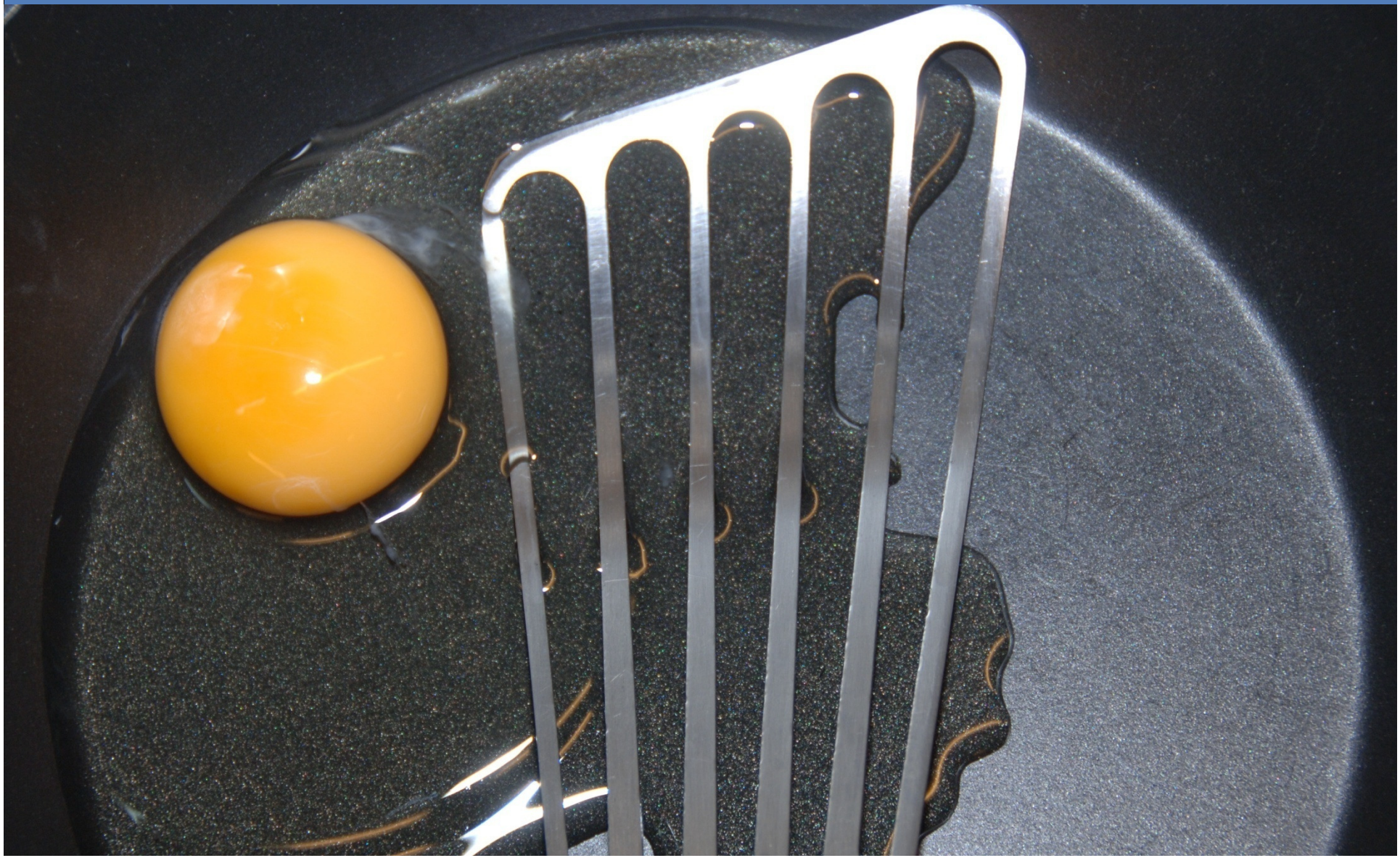
“Causes”

- Complexity of the code
- Changing requirements
- Some managers were not software engineers
- ….

Complexity of scrambled eggs



Input + Process



After refactoring ;-)

Other input + process metaphors

- Kolmogorov complexity
- DNA
  - Not in the sense of protein synthesis but as describing the growth of an organ, eg. optic nerve
  - Brevity of DNA makes evolution possible

Software as *input* + *process*

- The *input* could be domain knowledge
- The *process* could be the computation that mixes input with the Software Engineering knowledge

Domain Orientation trend:

- Domain Specific Languages (DSL)
- Code Generation
- Generative Programming (GP)
- Domain Specific Modeling (DSM)
- Model Driven Architecture (MDA)
- Model Driven Development (MDD)
- ...

A brief history of software

1950: user, domain expert, programmer are the same person

1970: programmer writes program, domain expert, if any, is consultant

2000: domain experts' contributions is paramount, but Domain Orientation is not yet the norm

von Neumann 1954

by John von Neumann

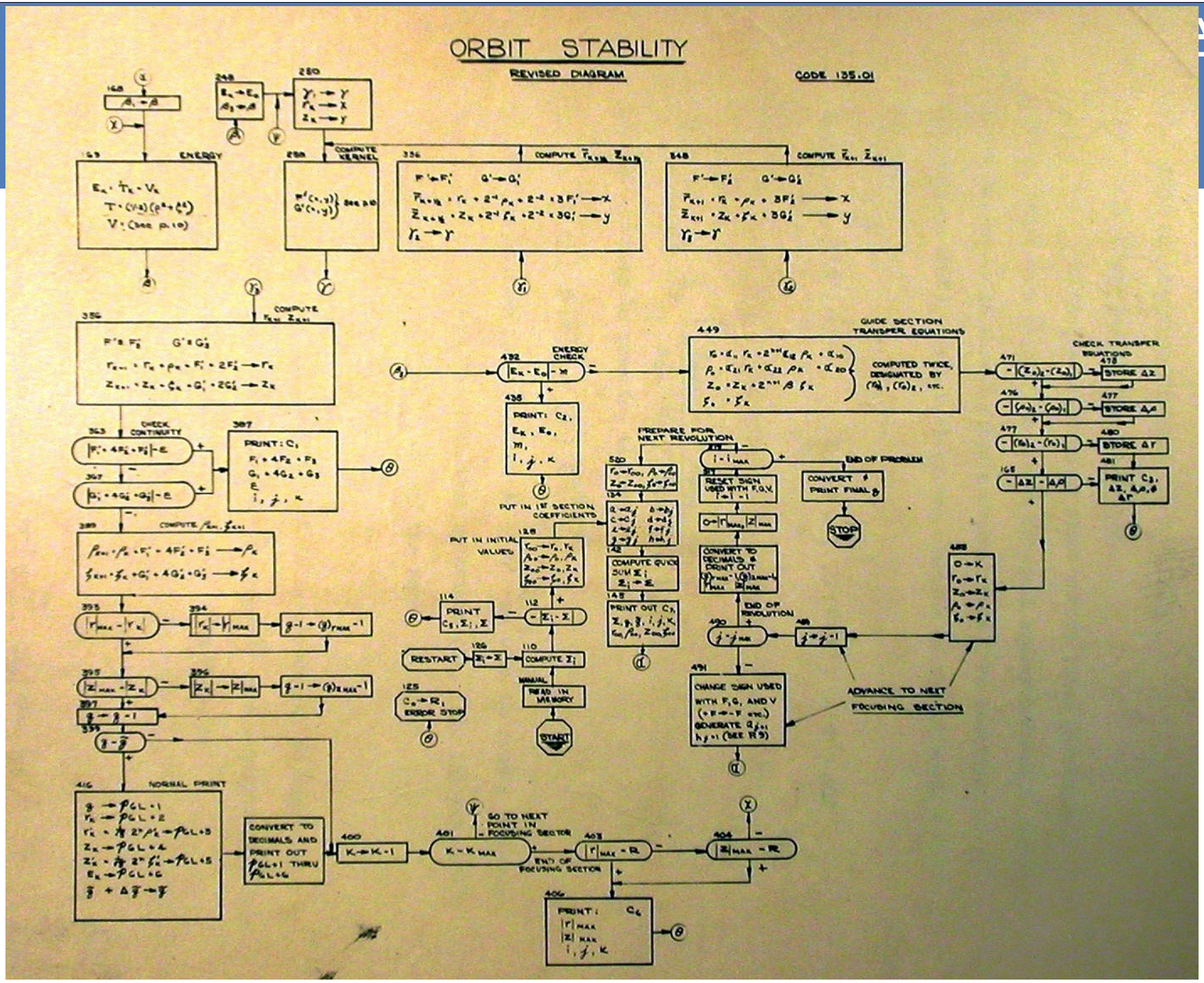
The Computer and the Brain

Yale University Press

totality of the (electrical) connections referred to constitutes the set-up of the problem—the expression of the problem to be solved, i.e. of the intention of the user. So this is again a plugged control. As in the case referred to, the plugged pattern can

Of course, the order-system—this means the problem to be solved, the intention of the user—is communicated to the machine by "loading" it into the memory. This is usually done from a previously

For its proper functioning—to solve the problems for which it is intended—a machine may need a capacity of a certain number, say N words, at a


ORBIT STABILITY
REVISED DIAGRAM
CODE 135.01
ENERGY
COMPUTE KERNEL
COMPUTE
GUIDE SECTION TRANSFER EQUATIONS
COMPUTED TWICE, DESIGNATED BY
ENERGY CHECK
CHECK TRANSFER EQUATIONS
STORE Δz
STORE Δρ
STORE Δr
PRINT: C3, Δz, Δρ, Δr
CHECK CONTINUITY
PRINT: C1
PREPARE FOR NEXT REVOLUTION
END OF PROBLEM
CONVERT & PRINT FINAL
STOP
RESET SIGN USED WITH F, G, V
PUT IN 1ST SECTION COEFFICIENTS
PUT IN INITIAL VALUES
COMPUTE QUICK SUM Σi
PRINT OUT C7
CONVERT TO DECIMALS & PRINT OUT
END OF REVOLUTION
PRINT C5, Σi, Σ
RESTART
COMPUTE Σi
MANUAL
READ IN MEMORY
START
C6 → R1 ERROR STOP
NORMAL PRINT
CONVERT TO DECIMALS AND PRINT OUT
K → K-1
K = K MAX
GO TO NEXT POINT IN FOCUSING SECTOR
END OF FOCUSING SECTION
PRINT: C6
CHANGE SIGN USED WITH F, G, AND V
ADVANCE TO NEXT FOCUSING SECTION

1969 Software Engineering

- Software Engineers worked on better practices and tools to serve the computer
- Software Engineers were little concerned with representation and processing of domain knowledge – the problem

Fred Brooks: “No Silver Bullet“ 1984

~~No such faith comforts~~ the software engineer. Much of the complexity he must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and sys-

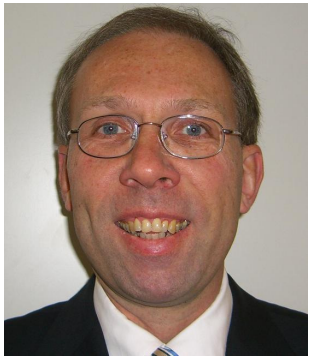
Consequences

- The focus has been on the least tractable representation of the problem, namely on the code:
  - Extremely complex
  - Hard to reason about
  - Hard to change, adapt, parameterize, reuse
  - Degrees of freedom too large

Domain Orientation

- Focus is on the domain, on the problem
  - Complexity is not exacerbated
  - Domain theorems can be used for reasoning
  - Changes, adaptation, reuse etc. are simpler

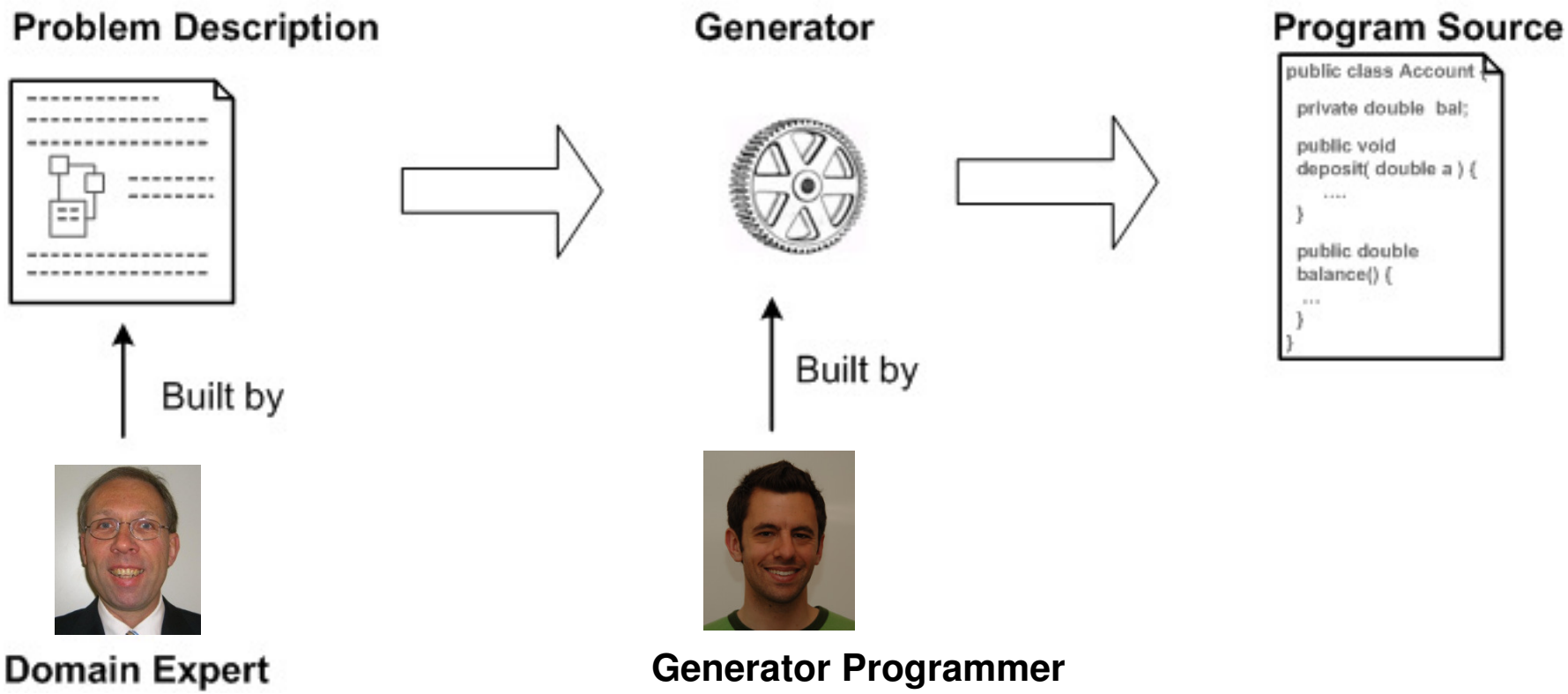
The Key Players



Domain Expert

Programmer

DSL + Code Generation



Domain oriented organization

- Domain Expert and Programmer agree on domain code format (schema)
- Programmer must still understand what the program will need to do and how to do it
- Domain Expert maintains domain code
- Programmer maintains implementation through the generator

Benefits of domain oriented organization

- Domain Expert direct participation is becoming feasible
- Programmers do not do repetitive work
- Programmers create a more valuable artifact

What prevents wide use of domain orientation?

- Programming languages are still the model and this leaves major issues:
  - Notations offered by the tools are unsatisfactory
  - Multi-domain is unaddressed
  - Domain evolution is unaddressed

Intentional Software™ Proposition

- Develop a specific tool, the

  Domain Workbench

  to support domain orientation.

Domain Workbench requirements

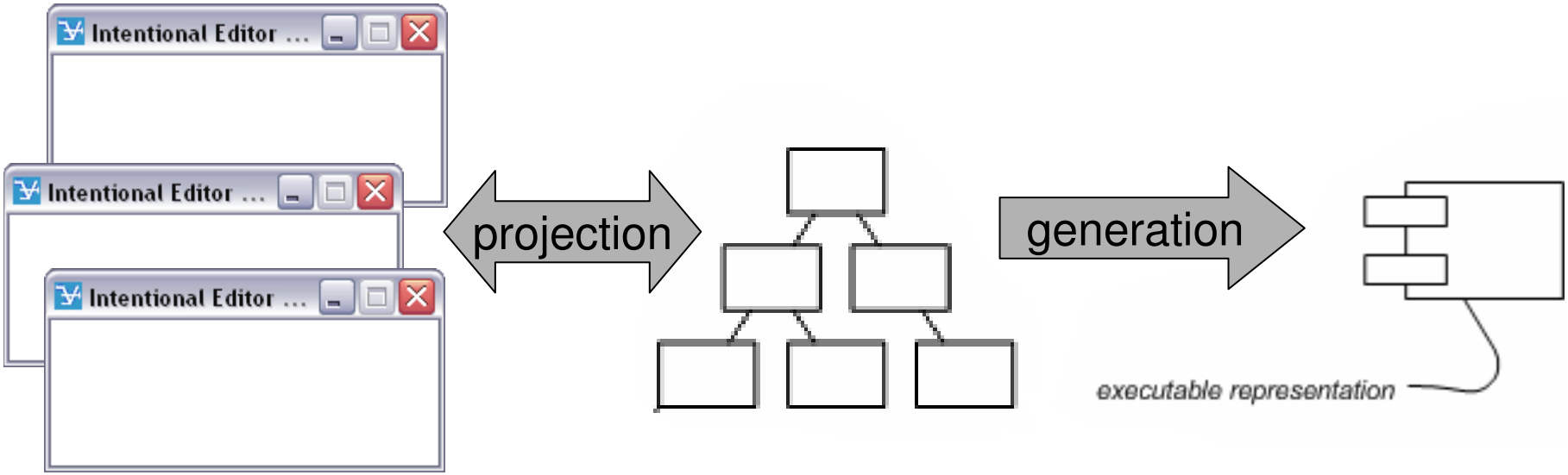
- Documents to be easily transformable
- Any customary notation to be acceptable
- Multi-domain documents (incl. meta)
- Domains must be able to evolve without limitations (structure and notation)
- Common editing & groupware environment

Intentional Software™ Technology

- **Intentional Domain Workbench**
  - Intentional Tree domain representation
  - Projectional multi-view editing

The Intentional Domain Workbench

1. Intentional tree
- Domain Schema
- Domain Code



2. Projectional Editor

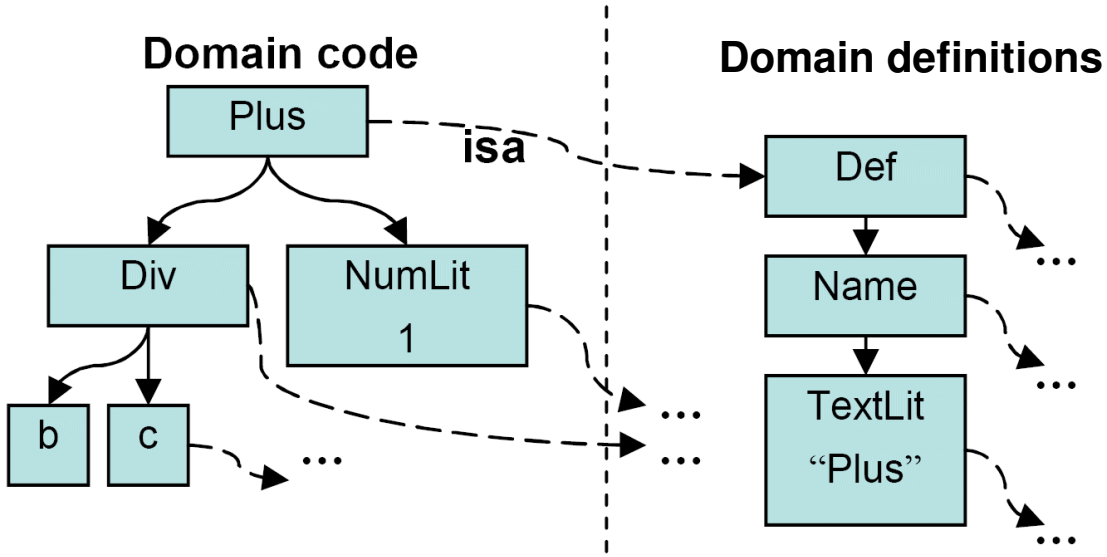
3. Generator

Intentional Tree

- Tree of nodes
- Each node has
  - own identity
  - isa identity: pointer to another node
  - labeled groups of offspring nodes
  - optional binary (incl. text) data

Intentional Tree

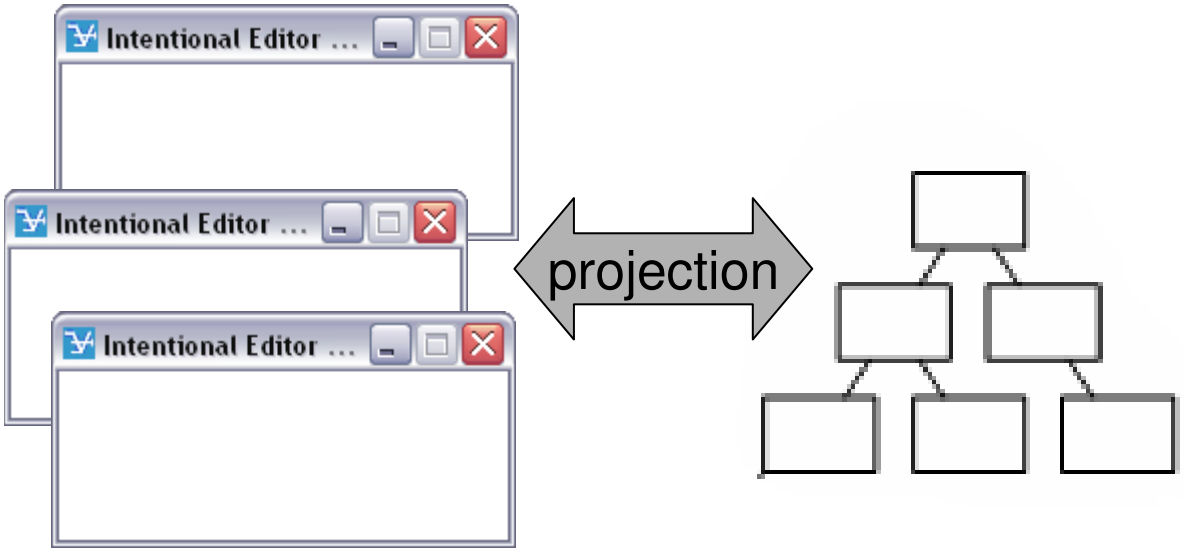
```
b / c + 1
```



Intentional Tree benefits

- Meaning / identities are invariant
- Parameterization is always possible and existing parameters are not affected
  - properties, params, attributes, modifiers..
- Perfect groupware is possible
- Transformations are efficient
- Text and binary data can be included

Projectional Editor



2. Projectional Editor

Projectional Editor

- Separates underlying representation from notation (syntax)
- Works in two directions: output *and* clicking
- There are special selections that take tree structure into account

Tree Editing

Selection	Paste	Result
b / c + 1	*	b * c + 1
b / c + 1	2	2 + 1
b / c	+ 2	b / (c + 2)
b / c	3	b / c(3)

Tree editing (different notation)

Selection	Paste	Result
$\frac{b}{c}$	+ 2	$\frac{b}{c+2}$
$\frac{b}{c()}$	3	$\frac{b}{c(3)}$

Projectional Editor implementation

- Series of lazy T2T bi-directional transformations
- First one distinguishes different domains
- One in the series is domain specific – written by programmer to define domain notation structure
- Last one shows image on display
- Each stage can have many formatting options

Projectional Editor benefits

- Large number of notations for:
  - matching existing notations
  - multi-domain
  - ambiguity resolving
  - domain evolution
- Minimal editing interaction, valid for all notations
- Can also edit Programs, Schema

Project organization

- Domain Expert and Programmer agree on domain code format (schema)
- They enter the schema
- Programmer must still understand what the program will need to do and how to do it
- Domain Expert maintains domain code
- Programmer maintains implementation through the generator and implements notations the Domain Expert likes

Weinberg: “Psychology of Programming” 1974

~~tires, engines, and so forth.~~ The tables themselves would be compiled from input written in essentially the form of the engineering specifications. This would allow the engineering personnel, rather than the programmers, to check the specifications, ~~and also permit one part of the specification~~

Application Example

- Pension systems in Holland
  - very complex, constantly changing domains
- Old approach: Domain experts were consultants to programmers
- New approach: ask Domain experts
  - “Tell us everything you want, in the way that is comfortable to you”

Exisiting pension notations

Rekenstructuur BPMT 08-11-06

Id	Sal	Ud	Es	Ve	WI	Inv	IV	IO	Ov	Ix	Groep	Toez	Stat
Id	Sal	-	Es	Ve	WI	Inv	-	-	-	Ix	Aanspraken	VP	Ac
Id	Sal												
Id	Sal												
Id	Sal												
-	-												
-	-												
-	-												
-	-												
-	-												
Id	Sal												
Id	Sal												
Id	Sal												
Id	Sal	Ud	Es	Ve	WI	Inv	-	-	-	Ix	Aanspraken	NP	AO

Invoergegevens:

- Code pensioensysteem
- Opgebouwde diensttijd vorige mutatie
- Toekomst diensttijd parttime
- Uitzicht diensttijd parttime
- Opgebouwde aanspraak vorige mutatie
- Aanspraak per dienstjaar fulltime vorige mutatie
- Aanspraak per dienstjaar fulltime deze mutatie
- Code afronding aanspraak

Uitvoergegevens:

- Opgebouwde aanspraak
- Toekomst aanspraak
- Uitzicht aanspraak

Berekening:

begin
als Code pensioensysteem = 1 (Eindloon)
dan
begin
Opgebouwde aanspraak deze mutatie :
Opgebouwde diensttijd
* Aanspraak per dienstjaar full

Toekomst aanspraak :=
Toekomst diensttijd parttime
* Aanspraak per dienstjaar full

Uitzicht aanspraak :=
Uitzicht diensttijd parttime
* Aanspraak per dienstjaar full

20.6 T-PREMIE-TM-VJ-2006 [401/KGT-DNM; 403/KGT-IDX]

Definitie: T-PREMIE-TM-VJ-2006 („EDAT)
Functie: Berekenen van de totale pensioenpremie vanaf de conversie-datum t/m het verslagjaar.
Weergave: 1
Algoritme: Extra weergave t.b.v. het jaarwerk.
Ing-dat: 01-01-1999
Nabew: P,2
RRT:
ADS T-PREMIE-TM-VJ(,BDAT,EDAT)

Weergave: 2
Algoritme: Als het de oude regeling betreft wordt de oude rekenregel aangeroepen, anders de som van de oude premie t/m 2005 en de nieuwe premie vanaf 2006.
Ing-dat: 01-01-2006
Nabew: P,2
RRT:
ALS I-IDC-RGL(„EDAT) = 1
DAN T-PREMIE-TM-VJ(,BDAT,EDAT)
ADS T-PREMIE-TM-VJ(„31/12/2005) + T-PME-OP-TM-VJ-VA-06(,BDAT,EDAT)

20.7 T-PME-OP-TM-VJ-VA-06 [sub]

Definitie: T-PME-OP-TM-VJ-VA-06 („EDAT,<werkgever-nr>,<contract-nr>,<idc-PVD>)
Functie: Berekenen van de totale pensioenpremie in het nieuwe reglement t/m het verslagjaar. De idc-PVD geeft aan of alleen gerekend moet worden voor PVD-ers:
0 = normale premieberekening
1 = premie PVD
2 = premie PVD herleid
3 = premie PVD herleid geïndexeerd
Weergave: 1
Algoritme: Extra weergave t.b.v. het jaarwerk.
Ing-dat: 01-01-1999
Nabew: P,2
RRT:
ADS 0

Weergave: 2
Algoritme: Bepaal de som van de pensioenpremies van alle weergavedelen in het verslagjaar en tel deze op bij de totale pensioenpremie t/m het vorige verslagjaar.
Ing-dat: 01-01-2006
Nabew: P,2
RRT:
ADS I-PREMIE-OP65-VA-06(JR,BDAT,EDAT„PAR-3) + T-PME-OP-TM-VJ-VA-06(JR - 1„„PAR-3)

- Overgangspercentage OP-NP

Capgemini Pension Workbench

File Edit Format View Search Transform Project Window Generate Options Users NN Scripting

Sort

NNRepo/15_loc1

Table of Contents

Library

- 1. Change Tracking...
- 2. Product Model...
- 3. Object Model...
- 4. Rules and Functions
  - 1. Rule: A001
  - 2. Rule: A002
  - 3. Rule: A004
  - 4. Rule: A005
  - 5. Rule: A009
  - 6. Rule: A013A
  - 7. Rule: A014
  - 8. Rule: A015
  - 9. Rule: A020
  - 10. Rule: A041
  - 11. Rule: A058
  - 12. Rule: A092
  - 13. Rule: A110
  - 14. Rule: A179
  - 15. Rule: A180
  - 16. Rule: A181
  - 17. Rule: A184
  - 18. Rule: A200
  - 19. Rule: A201
  - 20. Rule: A206
  - 21. Rule: A216
  - 22. Rule: A301
  - 23. Rule: A307
  - 24. Rule: A319
  - 25. Rule: A320
  - 26. Rule: A400
  - 27. Rule: A541
  - 28. Rule: A542
  - 29. Rule: C055
  - 30. Rule: C056a
  - 31. Rule: C057
  - 32. Rule: C058
  - 33. Rule: C059
  - 34. Rule: C062
  - 35. Rule: C063
  - 36. Rule: C083
  - 37. Rule: C084
  - 38. Rule: C085
  - 39. Rule: C086
  - 40. Rule: C101
  - 41. Rule: C102
  - 42. Rule: C111
  - 43. Rule: C151
  - 44. Rule: C152
  - 45. Rule: C153
  - 46. Rule: C154
  - 47. Rule: C155
  - 48. Rule: C156
  - 49. Rule: C195
  - 50. Rule: C196
  - 51. Rule: C199
  - 52. Rule: C200

All

Object Model...

Rules and Functions

Rules:

Rule A001...

Rule A002...

Rule A004

ID:
A004

Name:
A004

Description:

Audit comment:

Is checked:

Output attribute:
Einddatum meetellende periode

Inputs:

Attribute	Obj
Geboortedatum	THIS
Eindleeftijd meetellende periode	THIS
Afronding eindmoment	THIS

Functions used:
A191a
F0001

Local variables:

Pseudocode (text):
Einddatum meetellende periode := CALL METHOD A191a "Datum+periode" WITH [geboortedatum ; eindleeftijd meetellende periode *30}

Einddatum meetellende periode := CALL METHOD F0001 "Afronding datum" WITH {Einddatum meetellende periode; afronding eindmoment}

Body:
OUTPUT(Einddatum meetellende periode)
:= **CALL FUNCTION** A191a "A191a" **WITH** (
Datum1 := Geboortedatum[]
Periode := Eindleeftijd meetellende periode[] * 30
)
OUTPUT(Einddatum meetellende periode)
:= **CALL FUNCTION** F0001 "F0001" **WITH** (
Datum := **OUTPUT**(Einddatum meetellende periode)
Afrondingswijze := Afronding eindmoment[]
)

Usage switch:
☑

Unit tests:

Rule A005

ID:
A005

Name:
A005

Description:

All

Object Model...

Rules and Functions

Rules:

Rule A001...

Rule A002...

Rule\A004
(
Name: Name(StringAtom "A004");
Output attribute: Einddatum meetellende periode;
Pseudocode (text):
Paragraph(StringAtom "Einddatum meetellende periode := CALL METHOD A191a "Datum+periode" WITH [geboortedatum ; eindleeftijd meetellende periode *30}"),
Paragraph(StringAtom ""),
Paragraph(StringAtom "Einddatum meetellende periode := CALL METHOD F0001 "Afronding datum" WITH {Einddatum meetellende periode; afronding eindmoment}")
Body:
Assign
(
Output,
A191a
(
Datum1(Geboortedatum),
Periode
(
Mul(Eindleeftijd meetellende periode, IntLit 30)
)
)
)
Assign
(
Output,
F0001
(
Datum(Output),
Afrondingswijze(Afronding eindmoment)
)
)
Usage switch: PresenceLit
)

Rule A005

ID:
A005

Name:
A005

Description:

Audit comment:

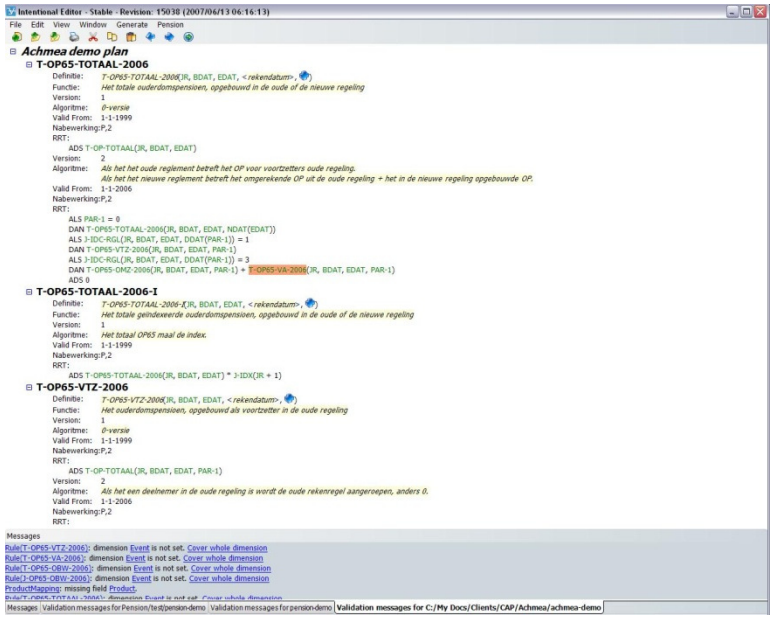
Is checked:

Output attribute:
Einddatum meetellende periode

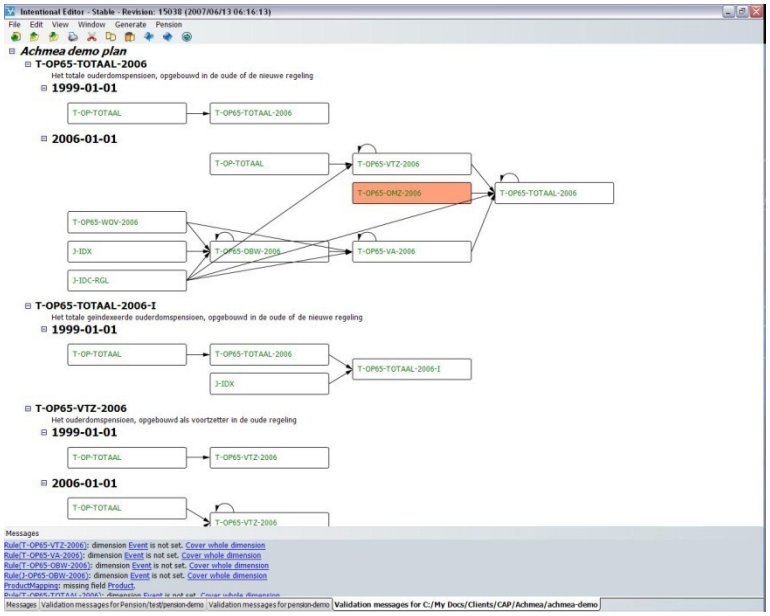
No errors found

Other views

For summaries



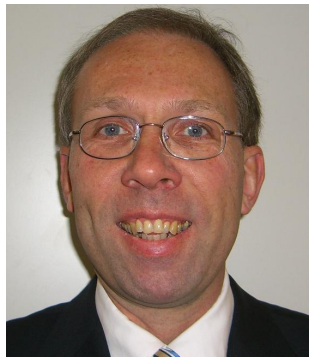
For rule dependencies



Pension Expert Testimonials

Herman Gerbscheid, Pension Architect:

- "This is the stuff I had to do mentally and keep consistent in my head all the time. It's great to finally have tools for it."



Suzanne Pront, Pension Expert:

- "Normally I know what I want, but don't know how to tell engineers. Now I can do this myself. This is a revolution!"

Sybren den Hartog, Java Architect:

- "Now we can generate business rules and domain structure, which we could not do in UML based MDA."

Benefits of Domain Workbench

- Quick / low cost / low risk domain development
- Domain contents and notations can match non-programming domain practices
- Multi-domain, multi-view
- Domains can evolve without losing legacy
- Domain code becomes a valuable work product in itself
- Complexity is kept in check

Future Work

- Complete system and make reference customer happy
- Extend to other domains and other partners
- Find ways to let programmers use our system with IP protection