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

Charles Simonyi
CEO Intentional Software
February 2008
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 outer;

if linecount < 0 then begin
pagecount := pagecount+1;
linecount := linecount+64;
if pagecount > i then begin
outstep(38); output(-ddd,-pagecount,outtext(<<
pagecount, overtime(<<
end or linecount<0); end of outline procedure;

procedure tape feed(n);
value n; integer n;
for n := n step -1 until 0 do outchar(62);
drum := drumplace;
linecount := 0;
tape feed(30); outclear;

start:
drumplace := drum;
pagecount := 0;
job no := incod;
if job no < 0 then goto finis;
input(day,month,year);
tape feed(x0); outpage;

comment end of the first part of SECRETARY,

USPSa 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\t1,14\t2,1\t3",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
The Computer and the Brain

by John von Neumann

The 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
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
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

Problem Description

Built by

Domain Expert

Generator

Built by

Generator Programmer

Program Source

public class Account
private double bal;
public void deposit(double a)
{
    ....
}
public double balance();
{
    ....
}

© 2007 Intentional Software Corporation
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
• 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 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

\[ \frac{b}{c} + 1 \]
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

© 2007 Intentional Software Corporation
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

<table>
<thead>
<tr>
<th>Selection</th>
<th>Paste</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b / (c + 1)$</td>
<td>*</td>
<td>$b * (c + 1)$</td>
</tr>
<tr>
<td>$b / c + 1$</td>
<td>2</td>
<td>$2 + 1$</td>
</tr>
<tr>
<td>$b / c$</td>
<td>+ 2</td>
<td>$b / (c + 2)$</td>
</tr>
<tr>
<td>$b / c$</td>
<td>3</td>
<td>$b / c(3)$</td>
</tr>
</tbody>
</table>

© 2007 Intentional Software Corporation
### Tree editing (different notation)

<table>
<thead>
<tr>
<th>Selection</th>
<th>Paste</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$ [c]</td>
<td>+ 2</td>
<td>$\frac{b}{c+2}$</td>
</tr>
<tr>
<td>$b$ [c()]</td>
<td>3</td>
<td>$\frac{b}{c(3)}$</td>
</tr>
</tbody>
</table>
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

© 2007 Intentional Software Corporation
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
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”
## Existing pension notations

### 20.6 T-PREMIER-TM.VI-2006 [401/KGT-DNM; 403/KGT-IDX]

<table>
<thead>
<tr>
<th>Id</th>
<th>Sal</th>
<th>Ud</th>
<th>Es</th>
<th>Ve</th>
<th>WI</th>
<th>Inv</th>
<th>IV</th>
<th>IO</th>
<th>Ov</th>
<th>Ix</th>
<th>Groep</th>
<th>Toez</th>
<th>Stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Aanspraken</td>
<td>VP</td>
<td>Ac 1</td>
</tr>
</tbody>
</table>

**Invoer gegevens:**
- Code pensioensysteem
- Opgebouwde dienststijd vorige mutatie
- Toekomst dienststijd parttime
- Uitzicht dienststijd parttime
- Opgebouwde aanspraak vorige mutatie
- Aanspraak per dienstjaar fulltime vorige mutatie
- Aanspraak per dienstjaar fulltime deze mutatie
- Code afronding aanspraak

**Uitvoer gegevens:**
- Opgebouwde aanspraak
- Toekomst aanspraak
- Uitzicht aanspraak

**Berekening:**

```
begin
  als Code pensioensysteem = 1 (Eindloon)
  dan
  begin
    Opgebouwde aanspraak deze mutatie
    Opgebouwde dienststijd
    * Aanspraak per dienstjaar ful
  en
  begin
    Toekomst aanspraak :=
    Toekomst dienststijd parttime
    * Aanspraak per dienstjaar ful
  en
  begin
    Uitzicht aanspraak :=
    Uitzicht dienststijd parttime
    * Aanspraak per dienstjaar ful
  en
end
```

### 20.7 T-PME-OP-TM.VI-VA.06 [sub]

**Definitie:** T-PME-OP-TM.VI-VA.06 ([EDAT],[werkgever-er],[contract-er],[idc-PVD])

**Functie:** Berekenen van de totale pensioenpremie vanaf de werkeinde datum 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 berlijd
- 3 = premie PVD berlijd geïndexeerd

**Weergave:** 1

**Algoritme:** Extra weergave tbv. het jaarwerk.

**Ing-dt:** 01-01-1999

**Nabev:** P2

**RRT:**

```
begin
  als Code pensioensysteem = 1 (Eindloon)
  dan
  begin
    Toekomst aanspraak :=
    Toekomst dienststijd parttime
    * Aanspraak per dienstjaar ful
  en
  begin
    Uitzicht aanspraak :=
    Uitzicht dienststijd parttime
    * Aanspraak per dienstjaar ful
  en
begin
  als Code pensioensysteem = 1 (Eindloon)
  dan
end
```

**Weergave:** 2

**Algoritme:** Bepaal de som van de pensioenpremie van alle weergeschieden in het verslagjaar en tel deze op bij de totale pensioenpremie t/m het vorige verslagjaar.

**Ing-dt:** 01-01-2006

**Nabev:** P2

**RRT:**

```
begin
  als Code pensioensysteem = 1 (Eindloon)
  dan
end
```

**ADS T-PREMIER-OP65-VA.06([RDAT],[EDAT],[PAR-3]) + T-PME-OP-TM-VI-VA.06([RDAT],[EDAT],[PAR-3])**

- Overgangspercentage OP-NP
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

© 2007 Intentional Software Corporation