The Magic of Software

Charles Simonyi

10/2/2013

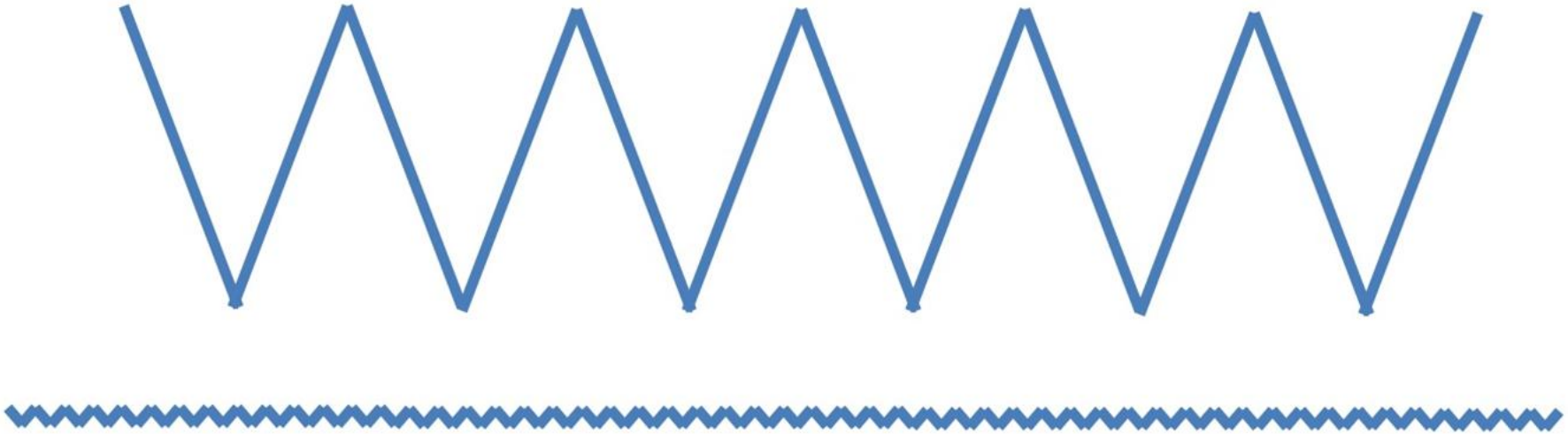
- Act1: It’s a Domain Specific World
- Act2: History of Software Engineering
- Act3: Domain Specific / Modeling Languages
- Act4: Knowledge Processing
- Conclusion: Simonyi’s law (wish)

Moore’s Law

42

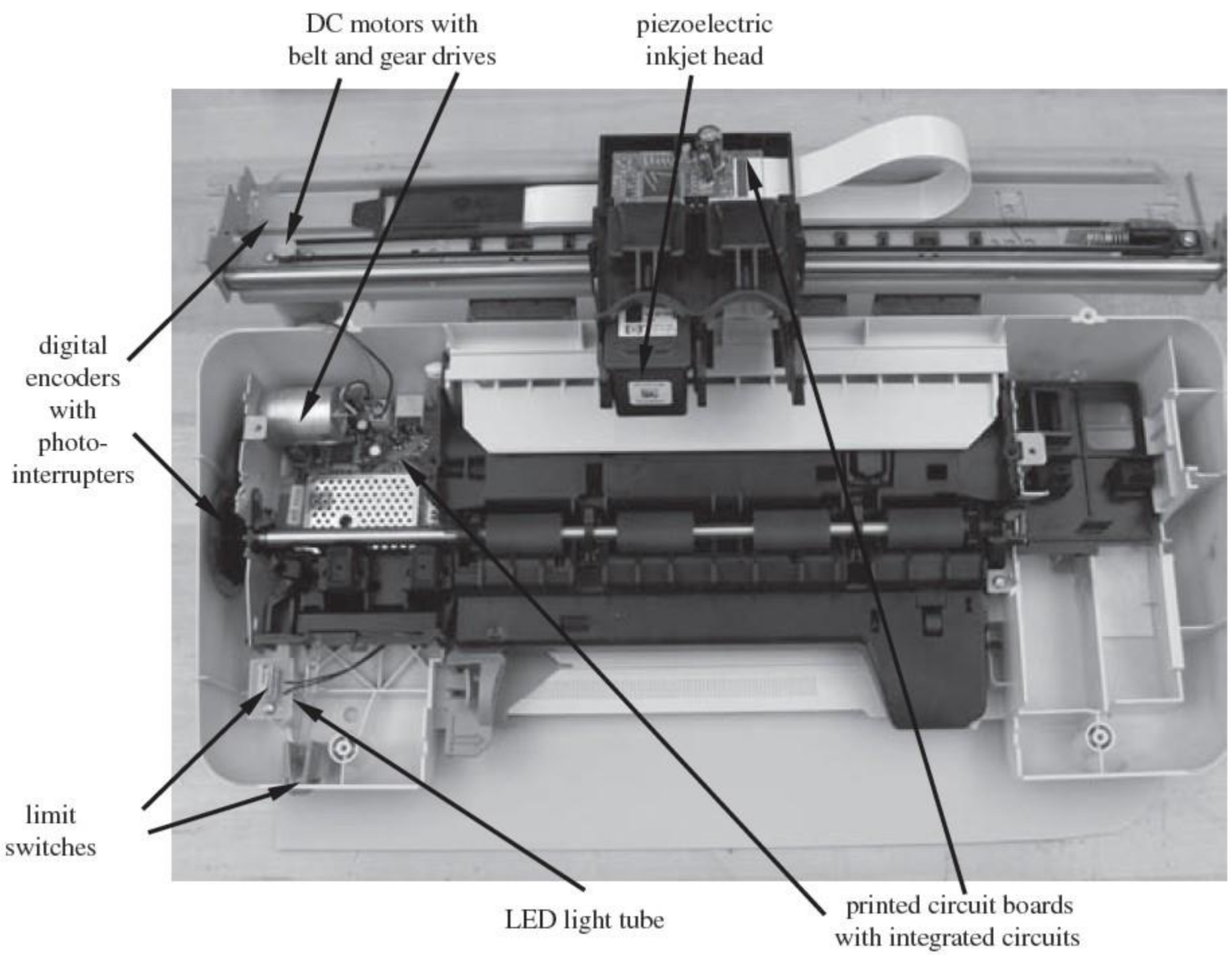
101010

Moore’s Law (cont.)



Act 1

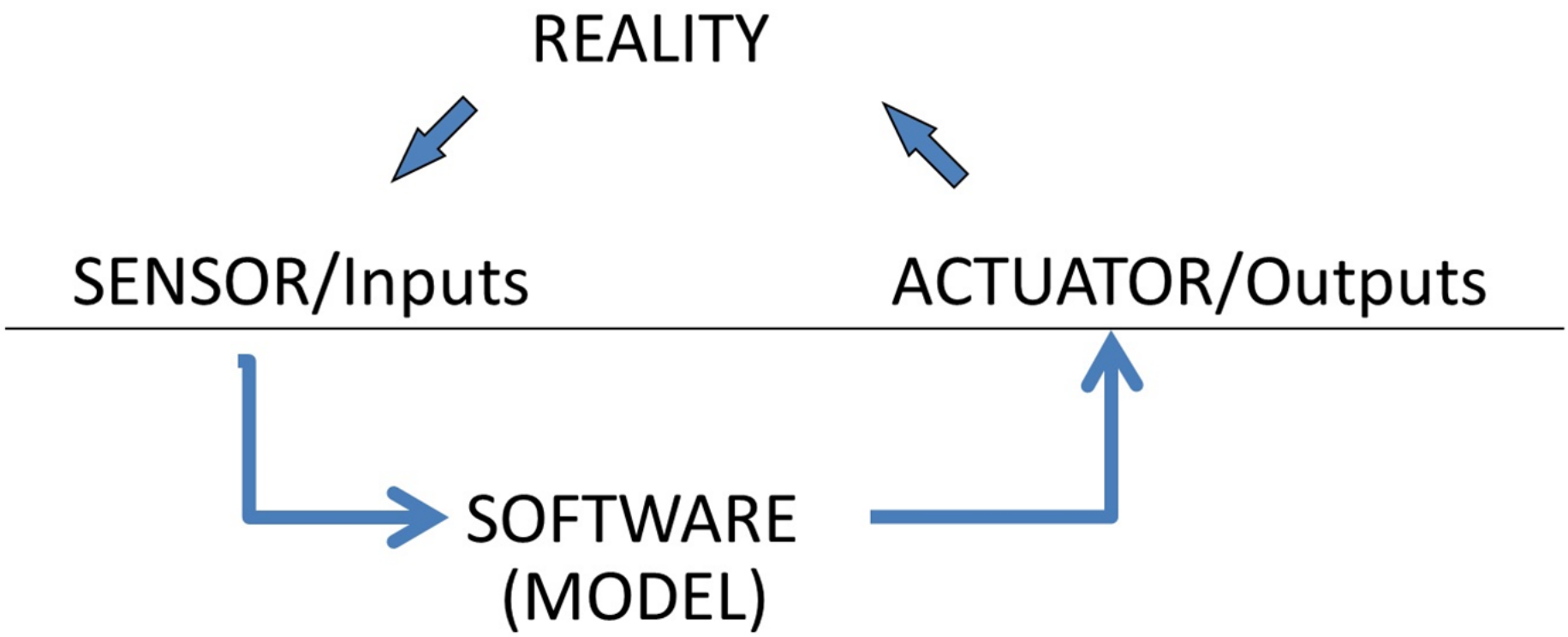
- Domain examples:
  - Printing
  - Word processing
  - Internal combustion engine


DC motors with
belt and gear drives
piezoelectric
inkjet head
digital
encoders
with
photo-
interrupters
limit
switches
LED light tube
printed circuit boards
with integrated circuits

Inkjet printer

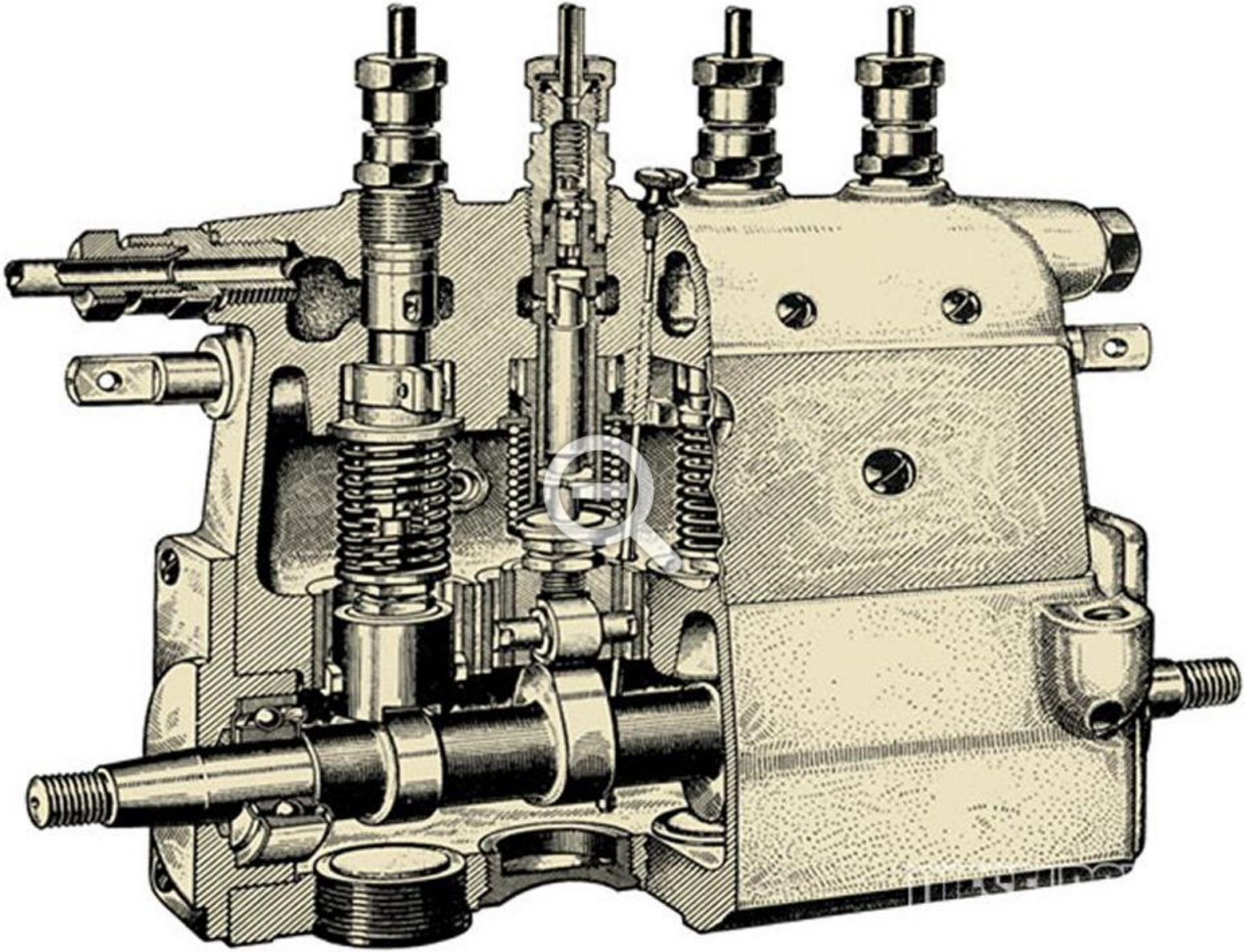
- 100 times cheaper
- 100 times more capable
- 100 times more reliable, rugged, simpler
- Where did the complexity go?

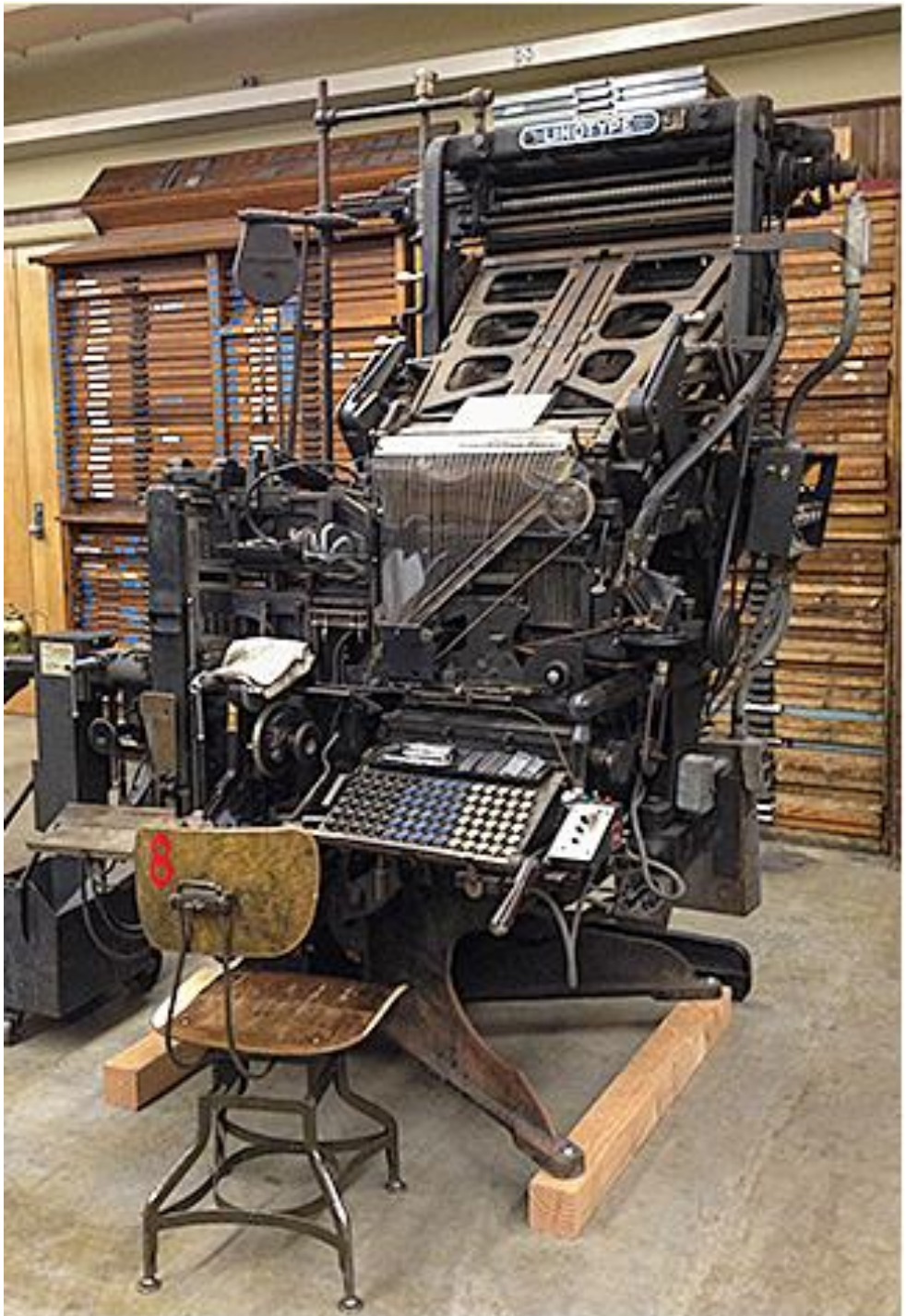
Digital Artifact/Process

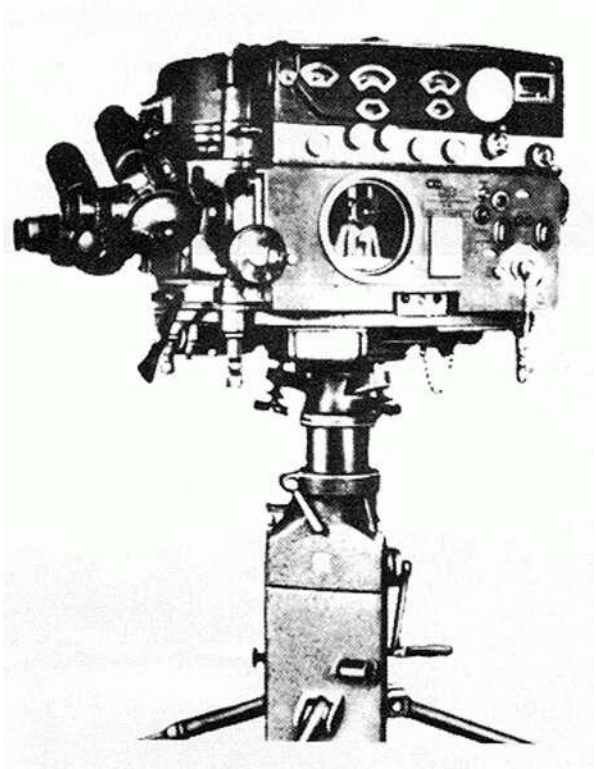


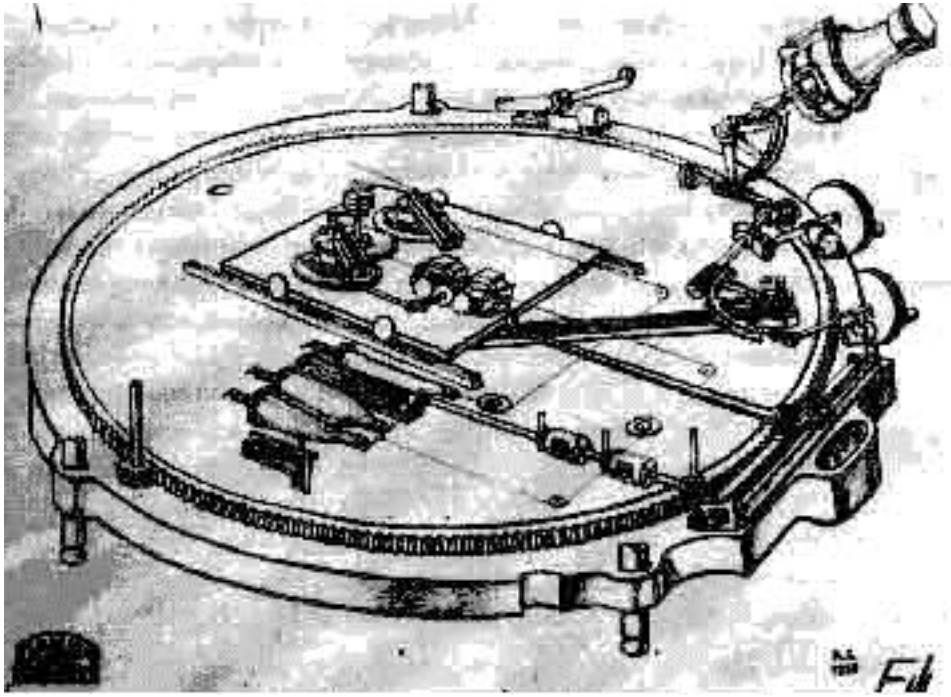
Example Artifact: Printer

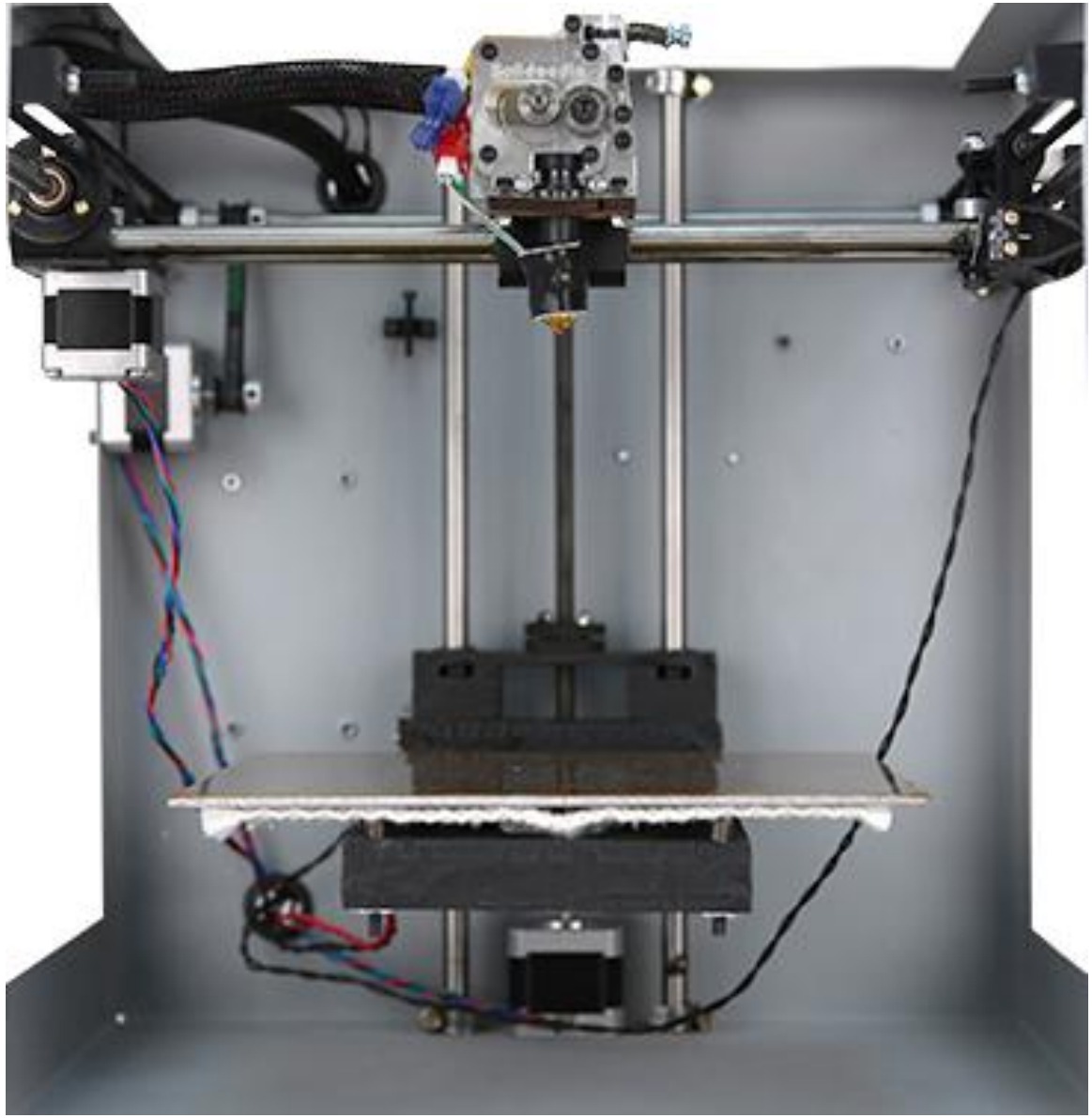
- Sensors: platen and print head positions
- Actuators:
  - Step paper
  - Step print head
  - Squirt ink jets of the selected color
- Model: typefaces, protocols











Models are everywhere

Aviation: flight control, engine control, GPWS, FMS, air traffic control CP-DLC…

Health: automated analysis, robotic surgery, CAD/CAM dentistry, diagnostic expert systems, health monitoring, health administration…

…Automotive, Manufacturing, Energy, etc…

Act 1: Summary

- Software is uniquely suited to absorb complexity
- A model of relevant parts of the world is a part of the software

Act 2: So what's the problem

- Software is expensive
  - Its value obscures its cost
  - Development slow
  - Often wasteful of resources
  - Wasteful of opportunities

Antique Process

writes

```
0000     16 0014 4    0040     02 6215 0
   1     02 0000 0       1     22 6156 4
   2     16 0013 0       2     02 6077 0
   3     40 0004 0       3     22 5011 0
   4     21 0003 0       4     02 7526 0
   5     40 0004 0       5     22 6156 4
   6     27 0014 0       6     02 7552 0
   7     25 0003 0       7     16 5500 0

0010     11 1010 0    0050     02 7546 0
   1     40 0004 4       1     16 5501 0
   2     24 0010 0       2     22 7342 4
   3     25 0000 4       3     00 0003 0
   4     25 0000 4       4     14 5062 4
   5    -16 7572 4       5     21 5022 0
   6     27 0013 0       6     02 5064 4
   7     24 0005 0       7     22 6671 0

0020     16 0021 0    0060     02 5501 0
   1     22 7560 0       1     16 7546 0
   2     00 0000 0       2     22 7342 4
   3     00 0000 0       3     00 0002 0
   4     00 0000 0       4     16 5501 0
   5     00 0000 0       5     03 7565 0
   6     00 0000 0       6     21 5032 0
   7     00 0000 0       7     22 6631 4

0030     00 0000 0    0070     22 6570 4
   1     00 0000 0       1     02 6077 0
   2     00 0000 0       2     01 7552 0
   3     00 0000 0       3     30 5500 0
   4     00 0000 0       4     11 0000 4
   5     00 0000 0       5     16 0000 0
   6     25 7534 4       6     02 6776 0
   7    -16 7572 4       7     13 5500 0
```

Domain Expert/Programmer/User

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

1969 Software Engineering (NATO)

Software Engineers worked on better algorithms and tools within the computers' limitations

Software Engineers were little concerned with representation and processing of domain knowledge – the problem

Patterns of practice were established which are still with us today

1969

As long as there were no machines programming was no problem at all.

And now we have gigantic computers programming has become an equally gigantic problem.

E. W. Dijkstra

Software Engineering Process

talks to

writes

```
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;
```

Domain Expert Programmer

Fred Brooks: “No Silver Bullet“ 1984

Much of the complexity the software engineer must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems that his interfaces must conform.

Cost of Engineered Software

- Proportional to: $E_{impl} * A_{impl}$

- E is number of (implementation) entities
- A is number of (implementation) aspects (cross-cutting concerns)

Act 2: Summary

- The focus has been on the least tractable representation of the problem, namely on the source code
  - Degrees of freedom does not match domain
  - Connection with domain is informal
  - Must reason, change, adapt, parameterize, reuse mostly in implementation space

Complexity: is it real?

Act 3: Domain Orientation trend:

Domain Specific Languages (DSL)
Code Generation
Generative Programming (GP)
Domain Specific Modeling (DSM)
Model Driven Architecture (MDA)
Model Driven Development (MDD)
Knowledge Processing
…

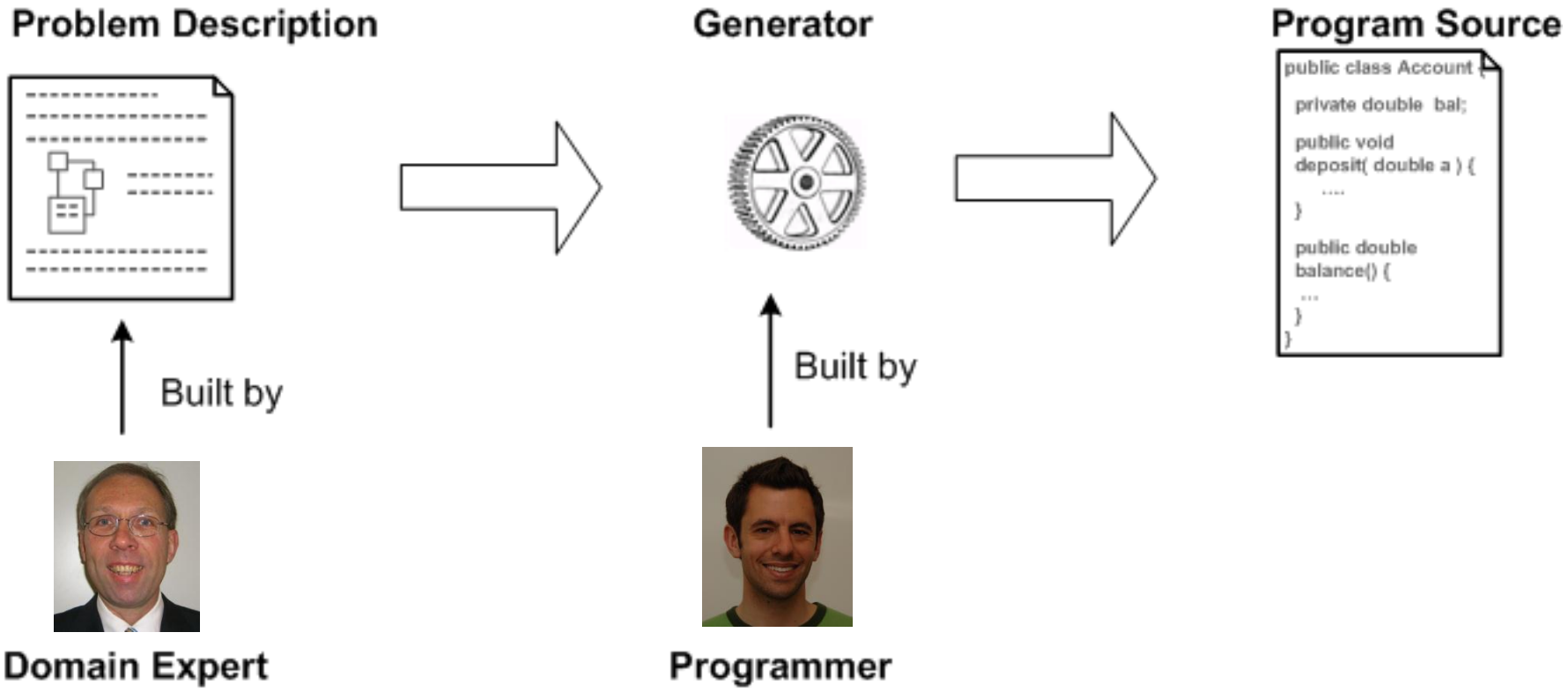
Same thing, different emphasis

- Model
- DSL
- Intention

Weinberg: “Psychology of Programming” 1974

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

Domain Oriented Development



Cost of Domain Software

- Proportional to: $E_{impl} + A_{impl}$

- E is number of (implementation) entities
- A is number of (implementation) aspects (cross-cutting concerns)

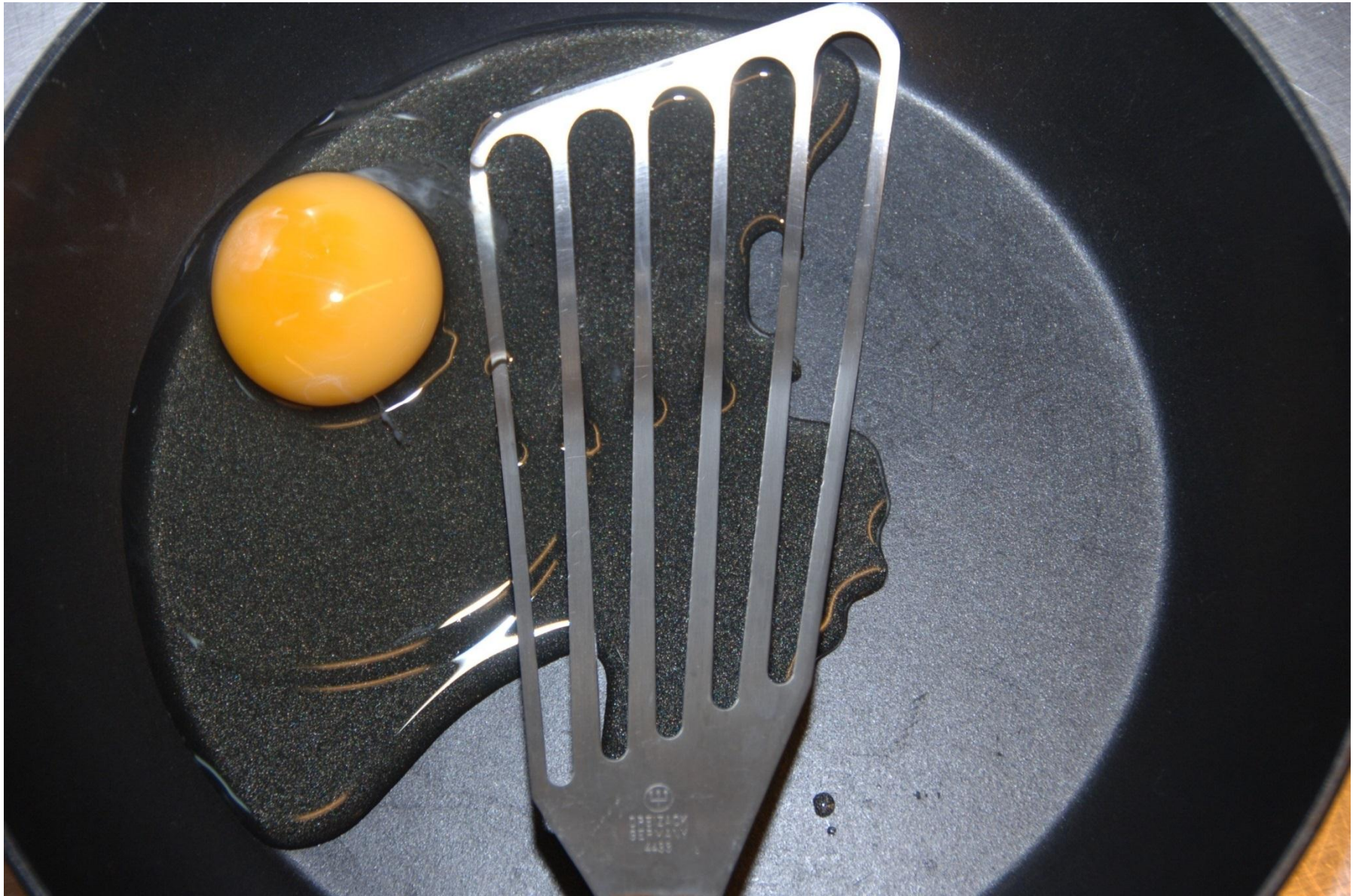
- Cf: Software Engineering: $E_{impl} * A_{impl}$

Domain Orientation

Focus is on the problem, on the “most tractable representation”

- Domain code is not a “solution”, it is a description of the problem.
- It is not executable, it can simply state facts. It is a model.
- Changes, adaptation, reuse, proofs etc. are simpler
- Domain Experts directly involved

Input + Process: less complex



What prevents wide use of domain orientation?

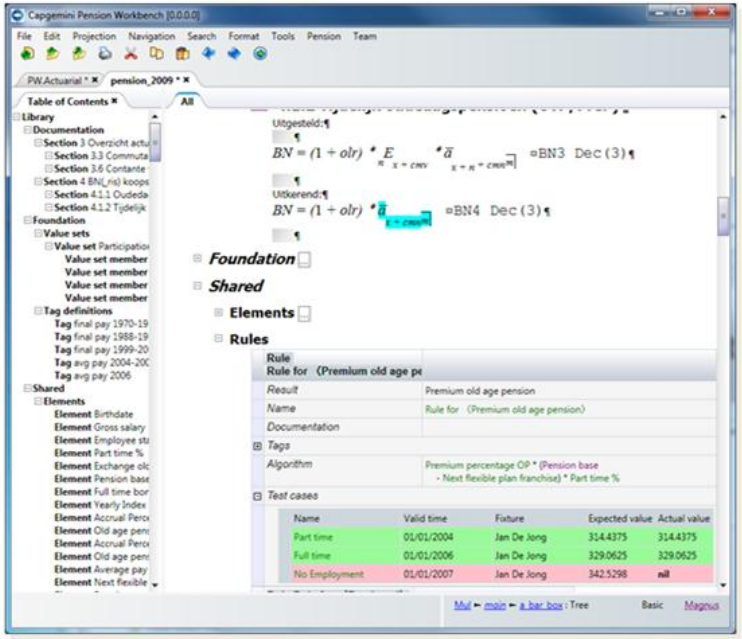
- Tradition of emphasis on execution rather than description
  - (imperative vs. declarative)
- Notations are unsatisfactory – not what domain experts use in their practice
- Multi-domain is not addressed
- Domain evolution is not addressed

Act 4: Knowledge Processing

Knowledge Capture → **Knowledge Processing** → **Generated Work Products**

Domain Knowledge

Domain Expert



Knowledge Workbench

Work Products:
Applications, Tools, Reports, Instructions

The Missing Link

Domain Knowledge

Ideas

Knowledge

PowerPoint

Word

Private Notes

Excel

Knowledge Workbench

Domain Experts

Software Experts

Java/C#/Objective-C
Ruby/Javascript
C/C++

Non-executable

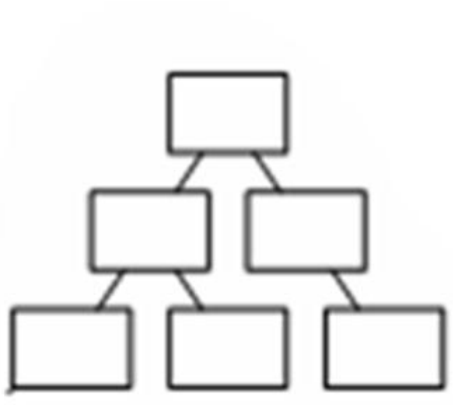
Executable

Computability

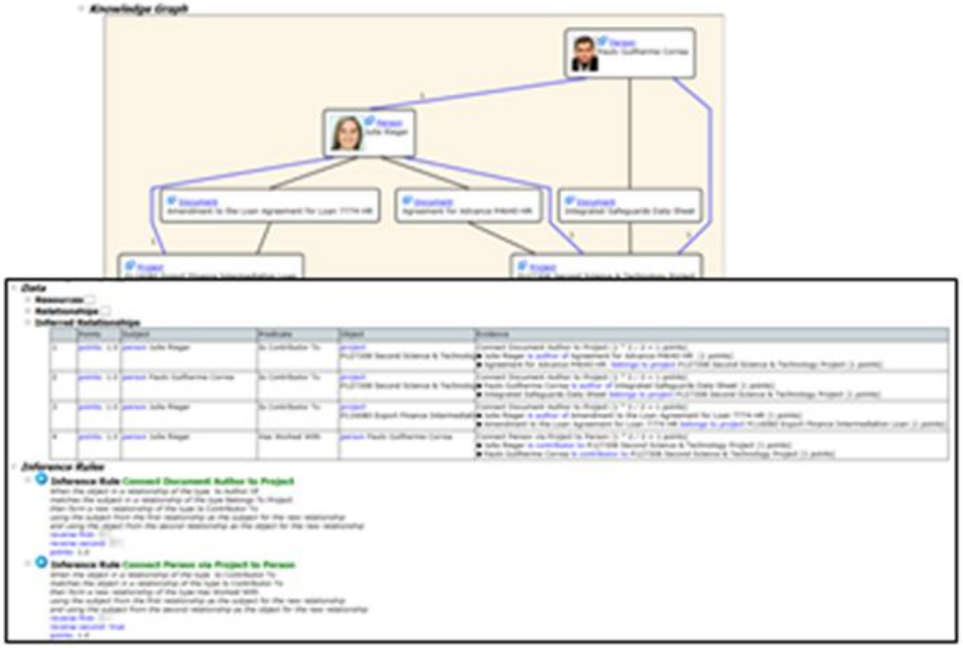
Knowledge Workbench depends on

- Uniform intentional domain representation
- Projectional multi-view editing
- Transformational composition

Projectional Editor



projections



Uniform representation for domains

Benefits of Intentional Representation

- Meanings/identities can be invariant
  - (even when domains are mixed)
- Parameters (degrees of freedom) can be added
- Uniform high quality groupware
- Images, datasets can be used in addition to text

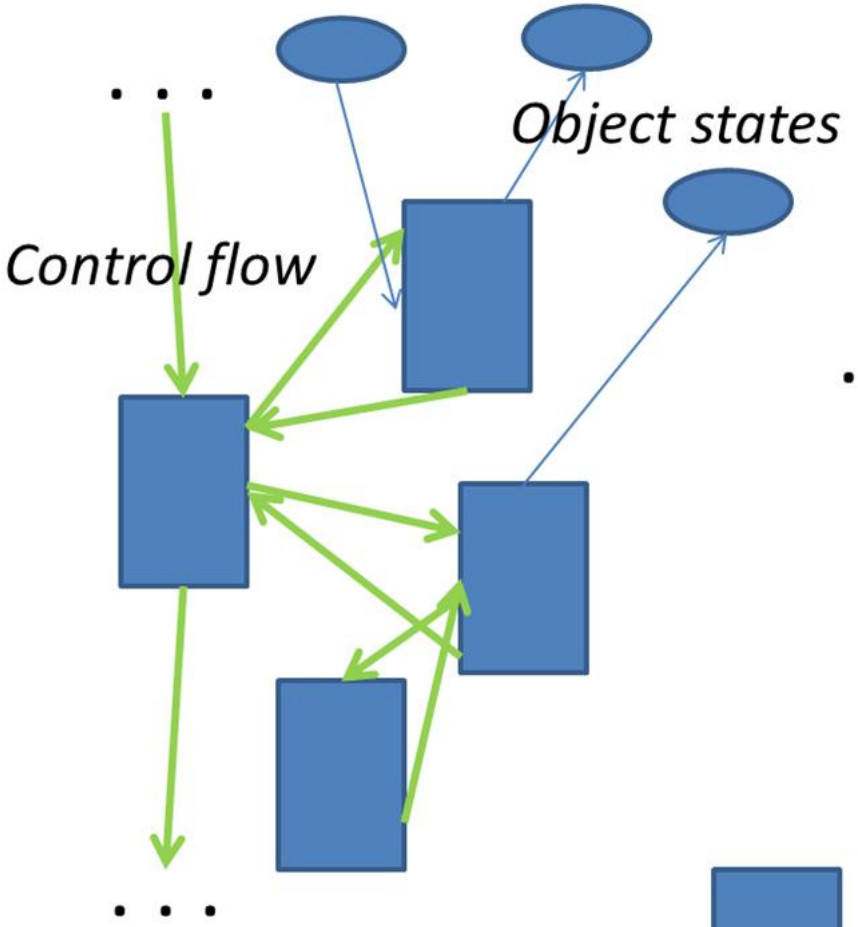
Benefits of Projectional Editor

- No limit on notations makes possible:
  - Matching traditional domain notation
  - Editing across domain boundaries
  - Domain evolution
  - Flipping notation as needed for convenience or disambiguation
- Parts can be composed
- Common editing interaction

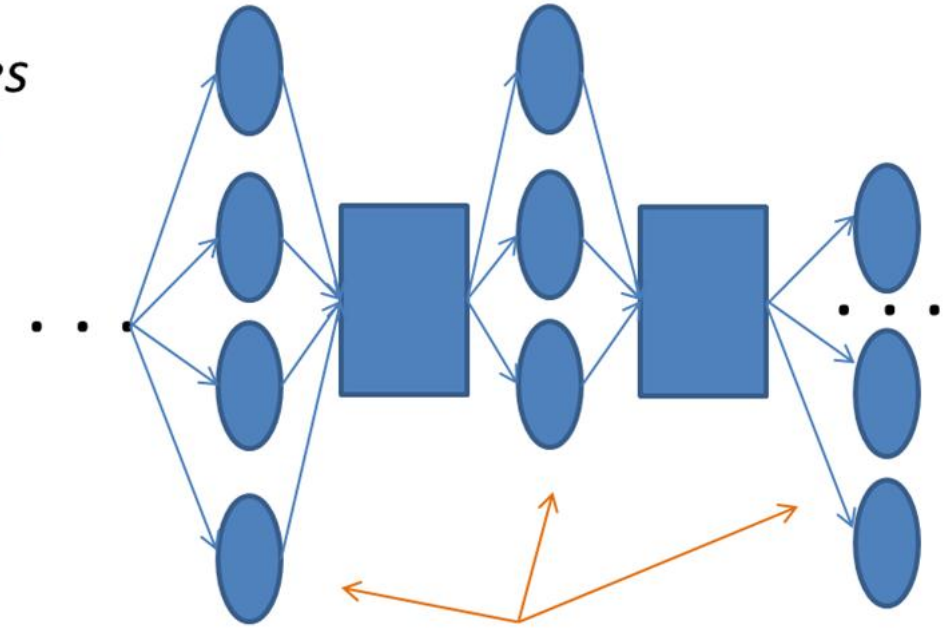
What about Aspects?

- Goal is to separate cross-cutting concerns
  - General Purpose languages can be extended: AspectJ
  - Generative programming separates aspects
  - Aspects can be made processable in run time

Imperative Composition



Transformational Composition



Declarative languages

Imperative methods

DSL Transformational Composition

- Imperative transformations input and output declarative code
- Declarative code is domain specific description of program state

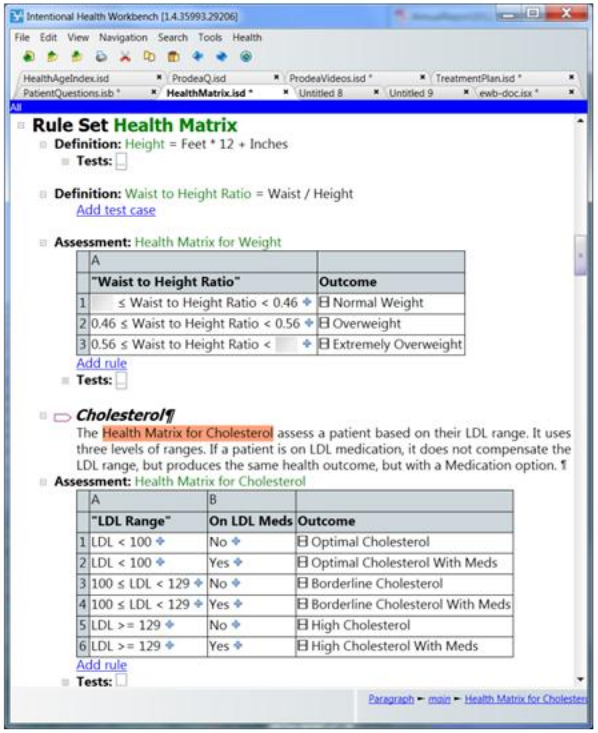
- Aspects are just predicates on the state
- DSLs are transformed from abstract to specific, from domain to implementation
- Usable for editing and for generation

Application Example

Knowledge → Record → **Knowledge Workbench** → Generate → **Medical Applications**

Medical Knowledge

Medical Expert

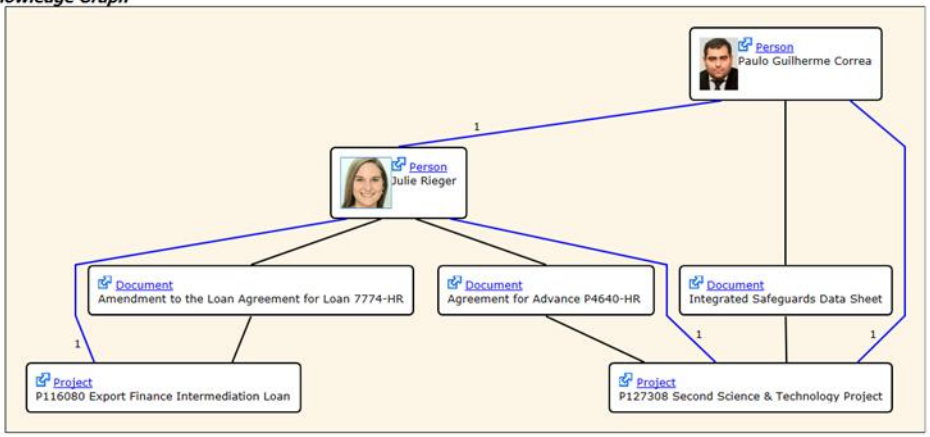


Health Workbench

Patients perform personalized health monitoring at home

DSL for Resource/task matching

Knowledge Graph



Data

Resources

Relationships

Inferred Relationships

	Points	Subject	Predicate	Object	Evidence
1	points: 1.0	person Julie Rieger	Is Contributor To	project P127308 Second Science & Technolog	Connect Document Author to Project (1 * 2 / 2 = 1 points) ● Julie Rieger is author of Agreement for Advance P4640-HR (1 points) ● Agreement for Advance P4640-HR belongs to project P127308 Second Science & Technology Project (1 points)
2	points: 1.0	person Paulo Guilherme Correa	Is Contributor To	project P127308 Second Science & Technolog	Connect Document Author to Project (1 * 2 / 2 = 1 points) ● Paulo Guilherme Correa is author of Integrated Safeguards Data Sheet (1 points) ● Integrated Safeguards Data Sheet belongs to project P127308 Second Science & Technology Project (1 points)
3	points: 1.0	person Julie Rieger	Is Contributor To	project P116080 Export Finance Intermediati	Connect Document Author to Project (1 * 2 / 2 = 1 points) ● Julie Rieger is author of Amendment to the Loan Agreement for Loan 7774-HR (1 points) ● Amendment to the Loan Agreement for Loan 7774-HR belongs to project P116080 Export Finance Intermediation Loan (1 points)
4	points: 1.0	person Julie Rieger	Has Worked With	person Paulo Guilherme Correa	Connect Person via Project to Person (1 * 2 / 2 = 1 points) ● Julie Rieger is contributor to P127308 Second Science & Technology Project (1 points) ● Paulo Guilherme Correa is contributor to P127308 Second Science & Technology Project (1 points)

Inference Rules

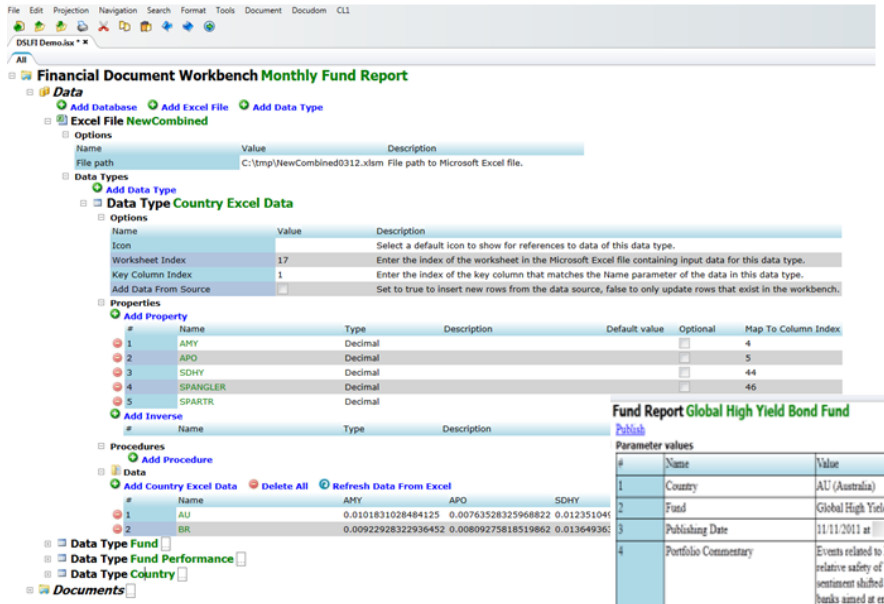
Inference Rule Connect Document Author to Project

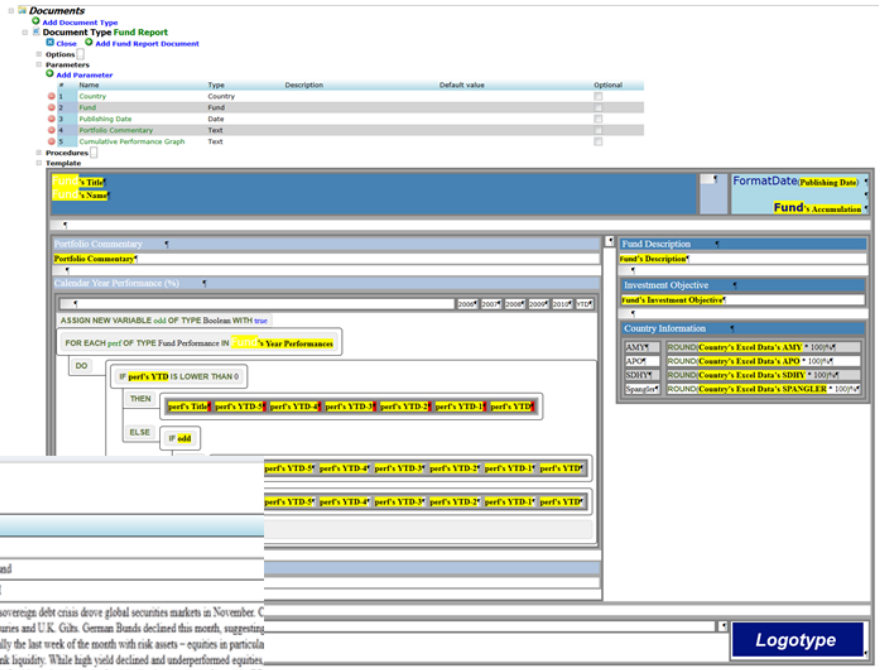
When the object in a relationship of the type Is Author Of
matches the subject in a relationship of the type Belongs To Project
then form a new relationship of the type Is Contributor To
using the subject from the first relationship as the subject for the new relationship
and using the object from the second relationship as the object for the new relationship
reverse first:
reverse second:
points: 1.0

Inference Rule Connect Person via Project to Person

When the object in a relationship of the type Is Contributor To
matches the object in a relationship of the type Is Contributor To
then form a new relationship of the type Has Worked With
using the subject from the first relationship as the subject for the new relationship
and using the subject from the second relationship as the object for the new relationship
reverse first:
reverse second: true
points: 1.0

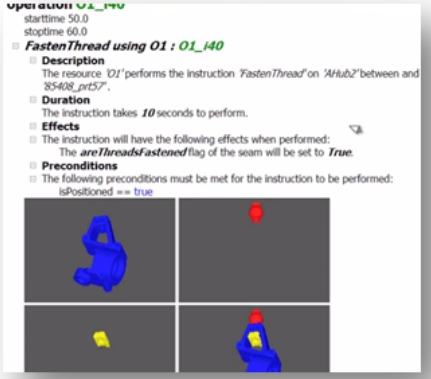
DSL for financial reports





Other DSL applications

Manufacturing

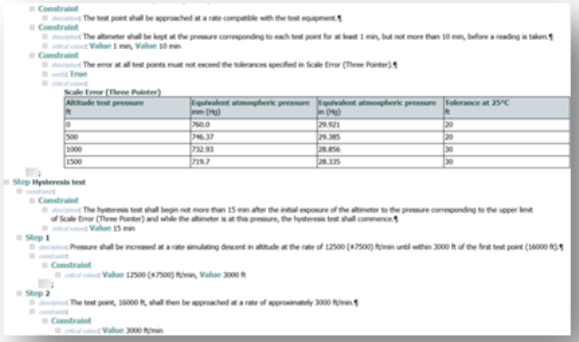


Retirement Plans

Salary gap	*Rule for (Salary gap)*		final pay 1970-1988 final pay 1988-1999 final pay 1999-2004	**when**(((**previous**(Gross salary) = 0 **and** Gross salary ≠ 0 **or previous**(Gross salary) ≠ 0 **and** Gross salary = 0) **or** (Gross salary - **previous**(Gross salary)) / **previous**(Gross salary) ≥ 10 %) **or** (**previous**(Gross salary) - Gross salary) / **previous**(Gross salary) ≥ 10 %)	

Name	Documentation	Tags	Valid time	Fixture	Expected value	Actual value
			1990-1-31	Piet Van Dijk	**true**	**true**
			1990-1-31	Jan De Jong	**true**	Nil
			1991-1-31	Piet Van Dijk	Nil	Nil
			1992-1-31	Piet Van Dijk	Nil	Nil

Aircraft Maintenance



Dragon!

Simonyi's law wish

A goal of Software Engineering should be to reduce incremental development costs to:

$E_{domain} + A_{domain}$

Cf: $E_{impl} * A_{impl}$ today

Conclusions

- Domain Orientation and Aspect Orientation have profound benefits
- Domain Code / Model need not be executable
- Work Products including executable Software can be created using generators
- Many practical problems remain but there are also promising approaches.