Domain-Specific Modeling
76 cases of MDD that works

17 November 2009
15:00-16:30 GMT
Steven Kelly

Outline

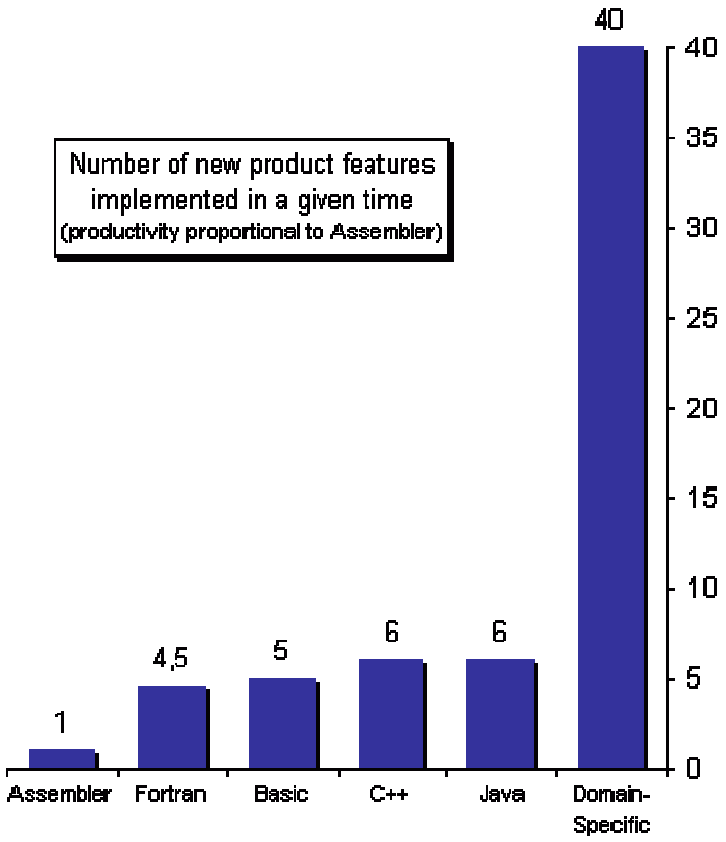
- Why Domain-Specific Modeling?
- Examples
- How to build
- Is this for you?

A rise in productivity is overdue

- "The entire history of software engineering is that of the rise in levels of abstraction"
  Grady Booch

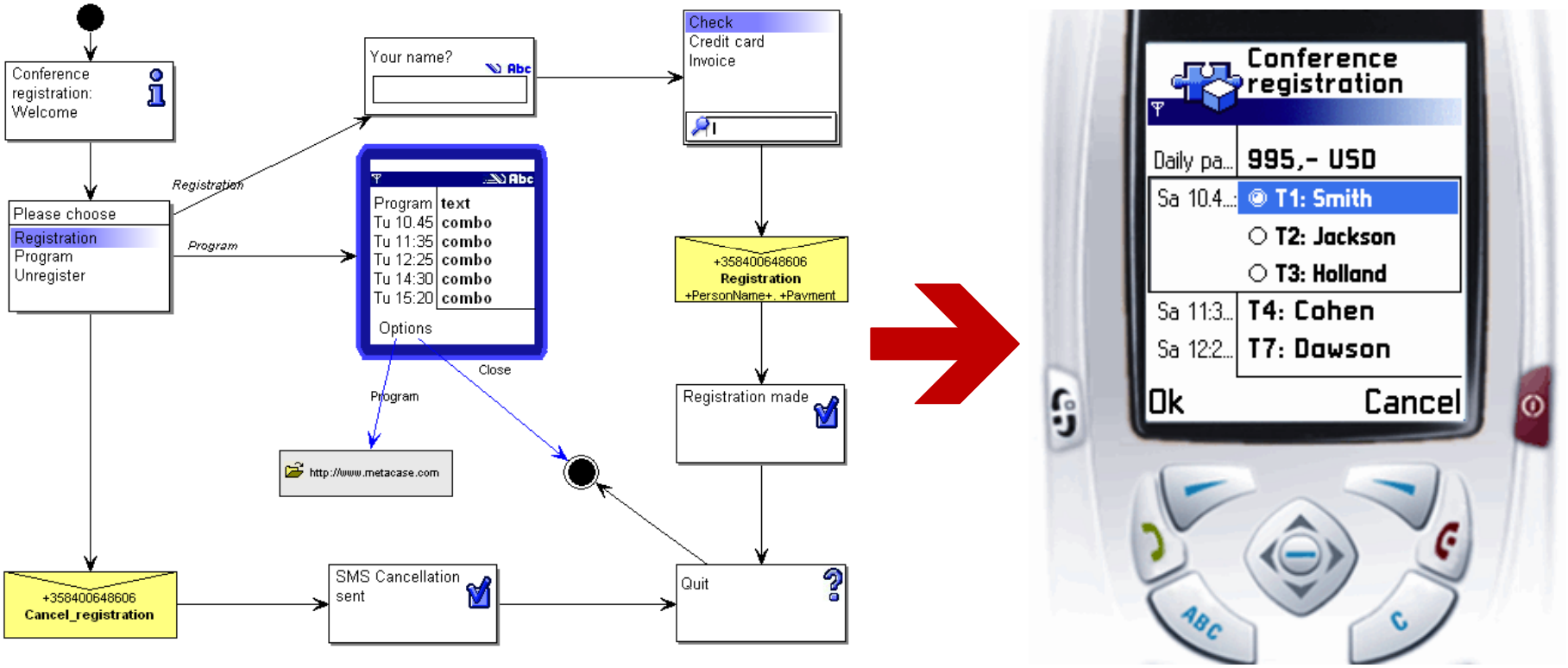
- New programming languages have not increased productivity

- Abstraction of development can be raised above current level...
- ... without losing control or accepting substandard results



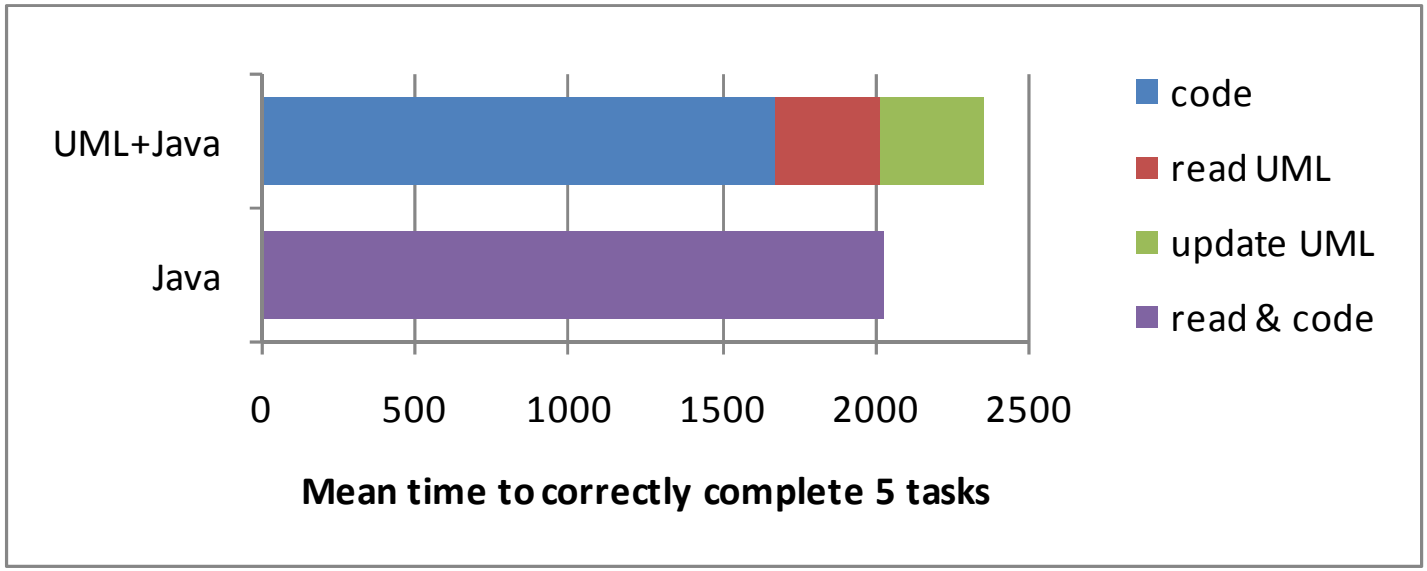
*Software Productivity Research & Capers Jones, 2002

Enterprise smartphone app demo

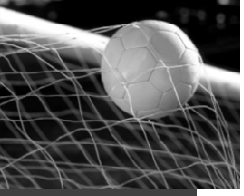


Why not use standard UML?

- Using standard UML is really no faster than just coding
  - Scientific measurements from 48% slower to 10% faster
  - Recently –15% in: WJ Dzidek, E Arisholm, LC Briand, A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance, IEEE ToSE 34:3, 5/08



- **We need something more than standard UML!**



Abstraction benefits

… work on a higher level

… do more with less

… insulate from technology

Goals

Worst practices: Initial conditions

Worst Practices for Domain-Specific Modeling
Steven Kelly, Risto Pohjonen
IEEE Software, vol. 26, no. 4, pp. 22-29, July/Aug. 2009
Free from: www.metacase.com/stevek.html

- 76 DSM cases
- 15 years
- 4 continents
- several tools
- 100 DSL creators
- 3–300 modelers

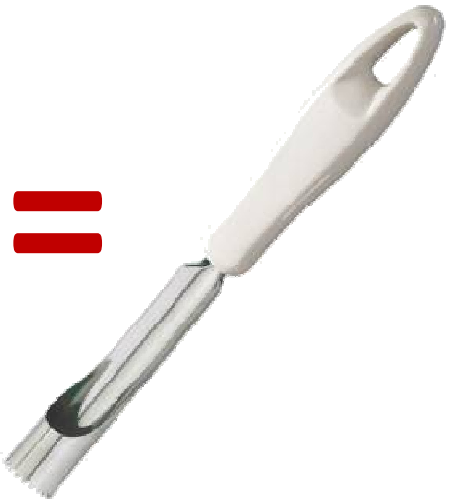
Worst practices: Concept Source

UML: Old Wineskins

Extend a large, general-purpose language

+

!=



3GL: Visual Program

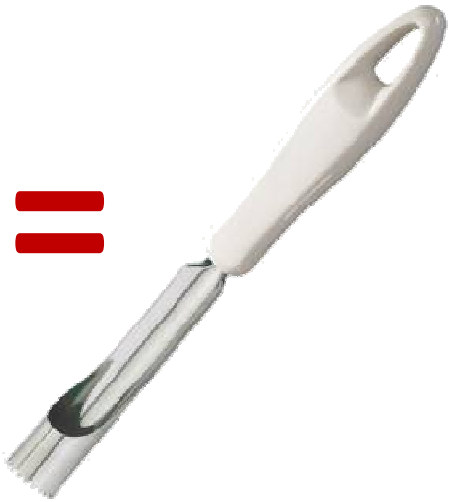
Traditional programming language + graphics

1 symbol => 1 keyword ☹

MDA: UML+UML+UML

Multiple semi-automatic transformations

!=



MDA: UML+UML+UML

Multiple semi-automatic transformations

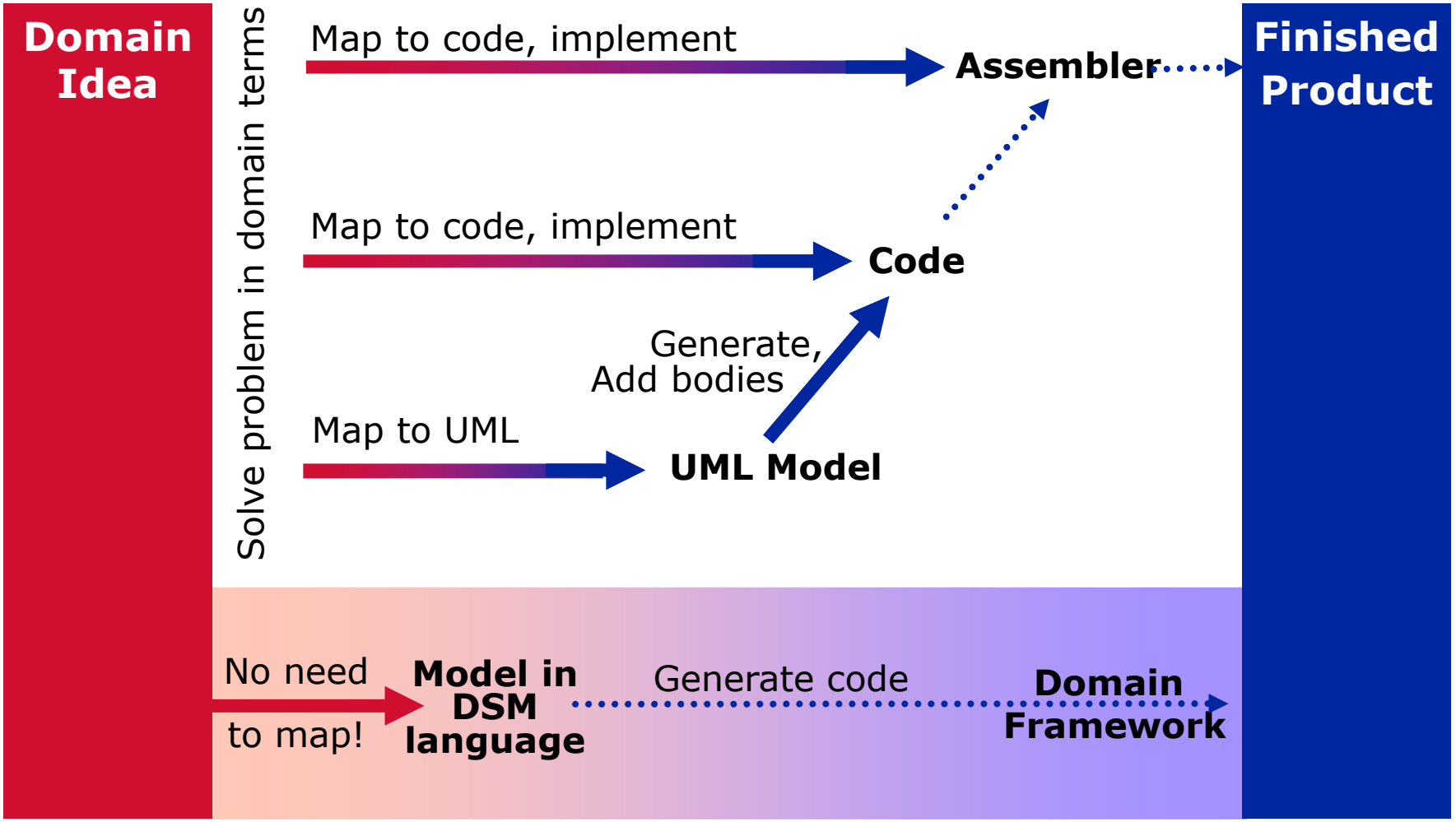
Manufacturers' claims:
+22% (Obeo) … +35%(OpJ)
Not enough!

Booch, Rumbaugh and Selic say:

- "the full value of MDA is only achieved when the modeling concepts map directly to domain concepts rather than computer technology concepts"
  - An MDA Manifesto, MDA Journal, May 2004

- Use language of problem domain
- Generate language of solution domain

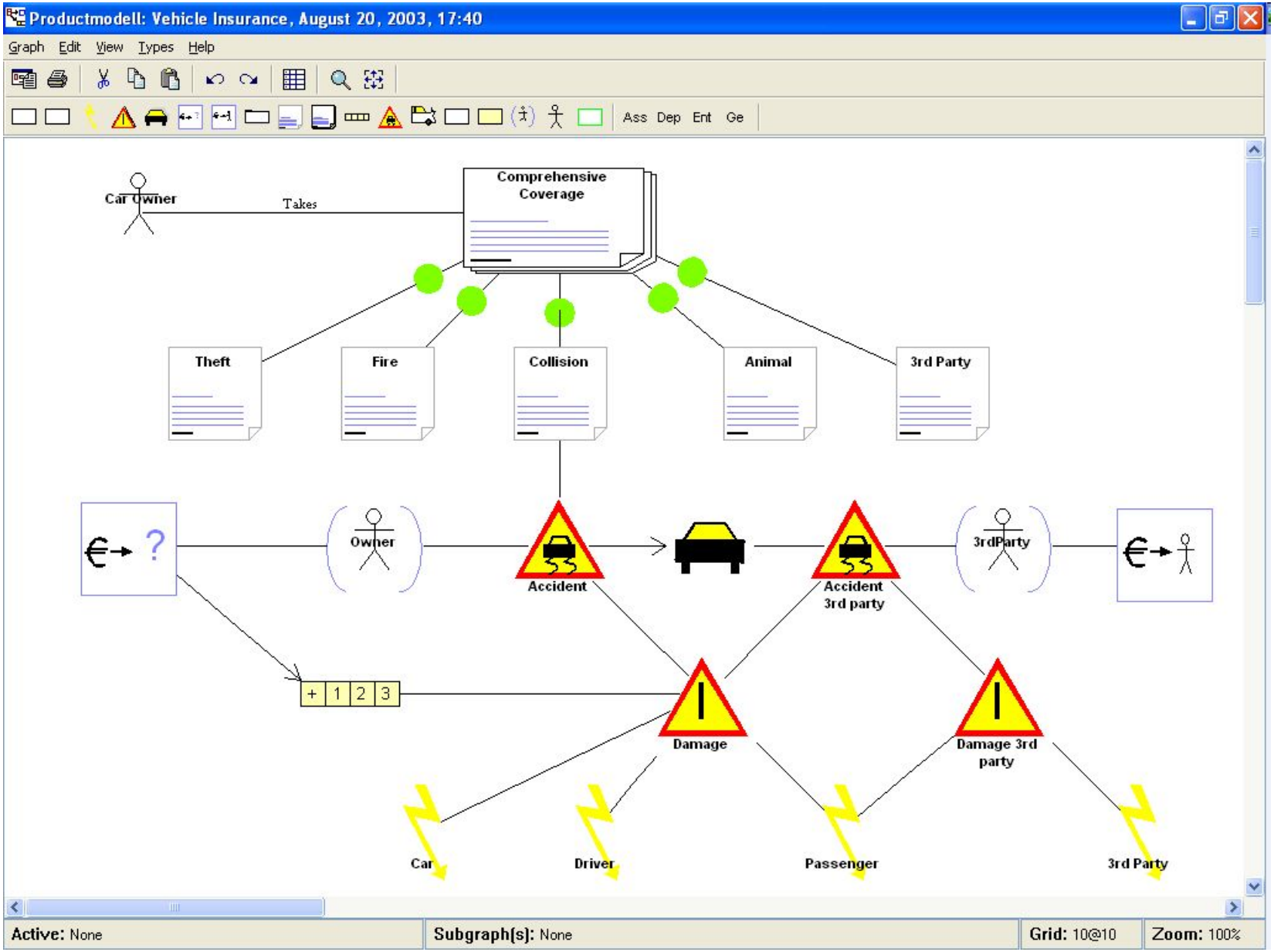
Modeling functionality vs. modeling code

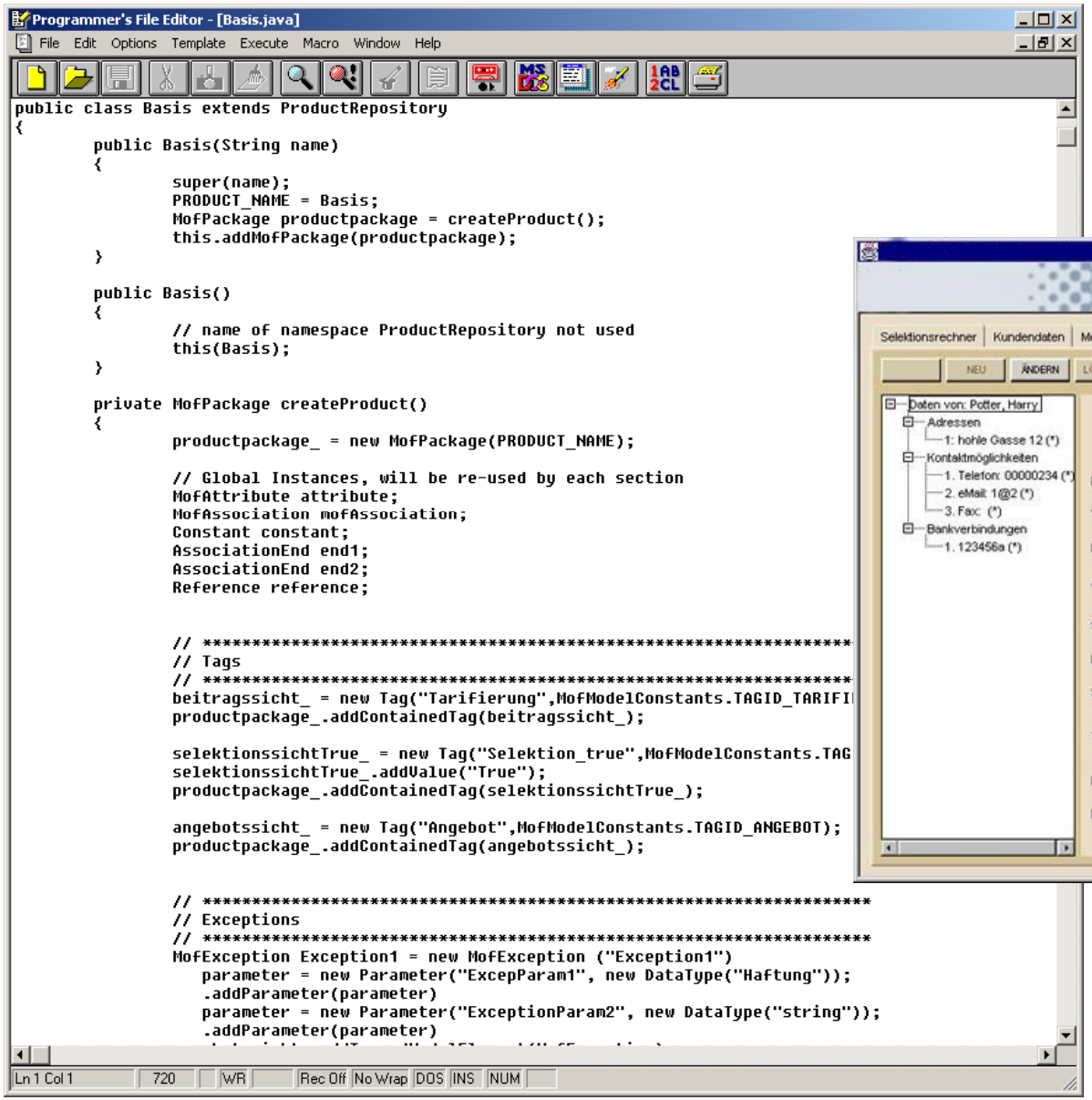


Case: Financial web application

- Developing portal for insurances and financial products
- Need to specify several hundred financial products
- Insurance experts visually specify insurance products and generate code to the portal

- Comparison to hand-writing Java after first 30 products = DSM 3-5 times faster, 50% fewer errors

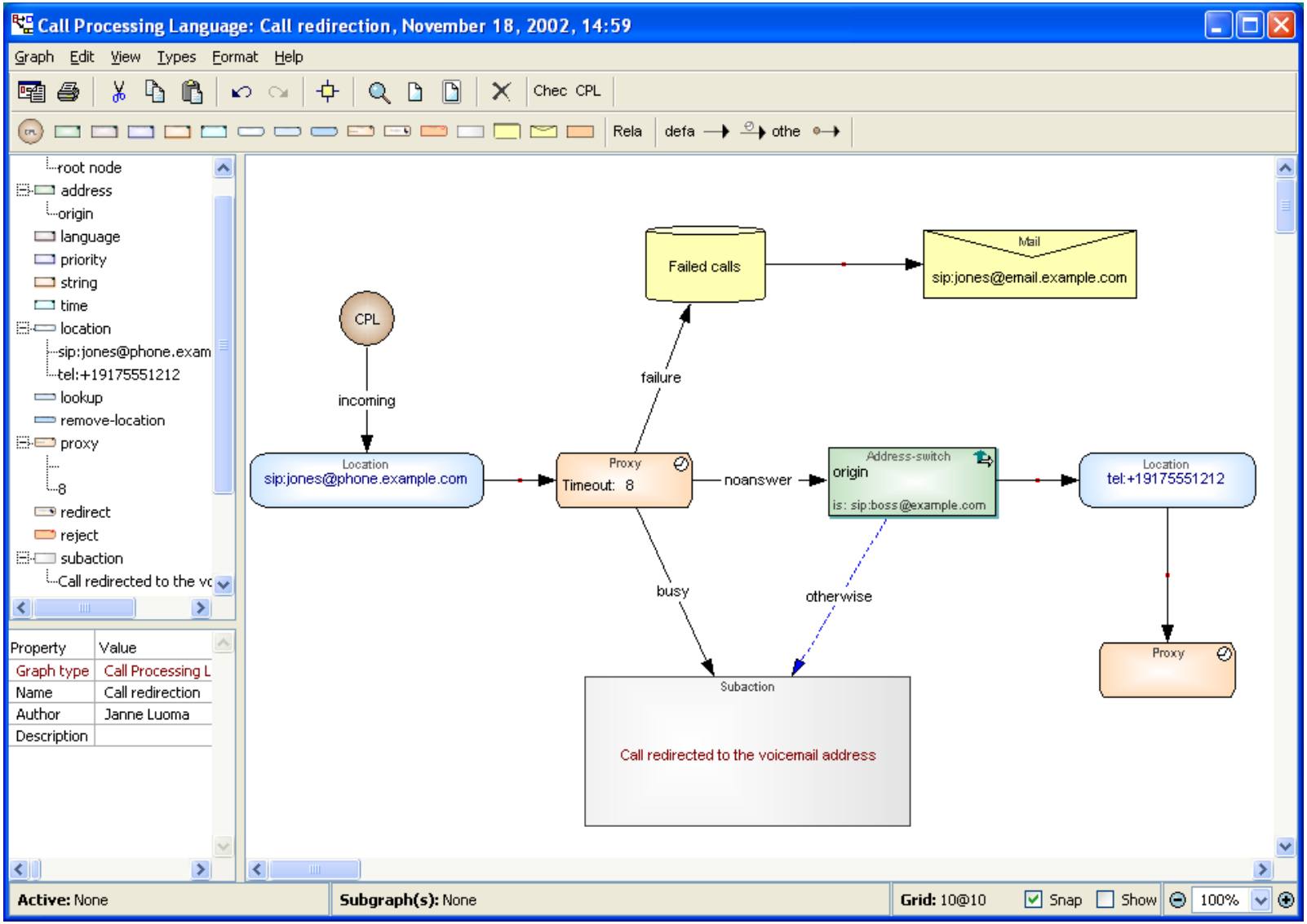


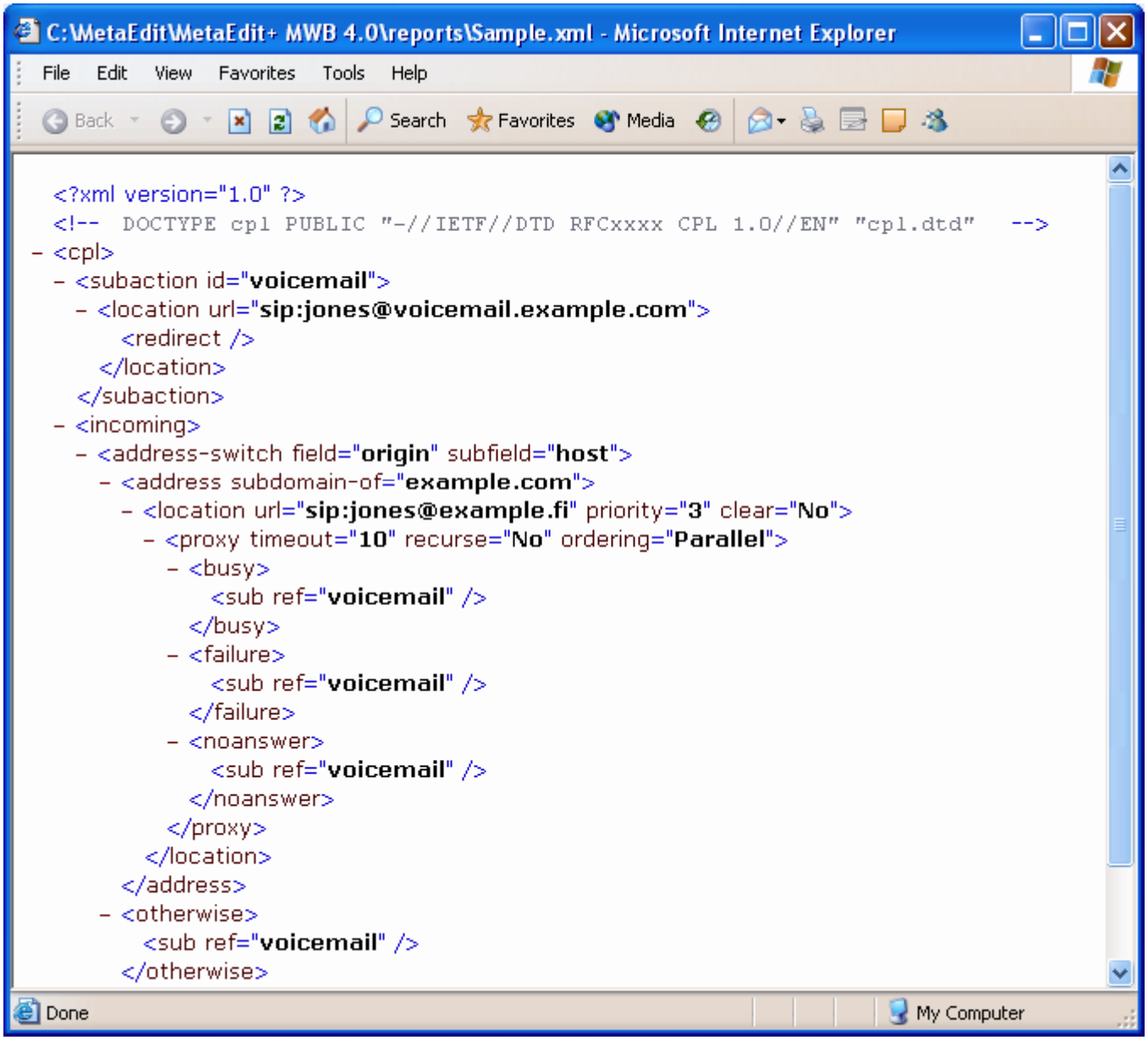


Case: Call Processing Services

- Specify services than can run safely on Internet telephony servers
- Designs can be considered valid and well-formed right from the design stage
- Language uses concepts familiar to the service developer
  - Switches, Locations and Signaling actions etc.
- Generate full service from the model

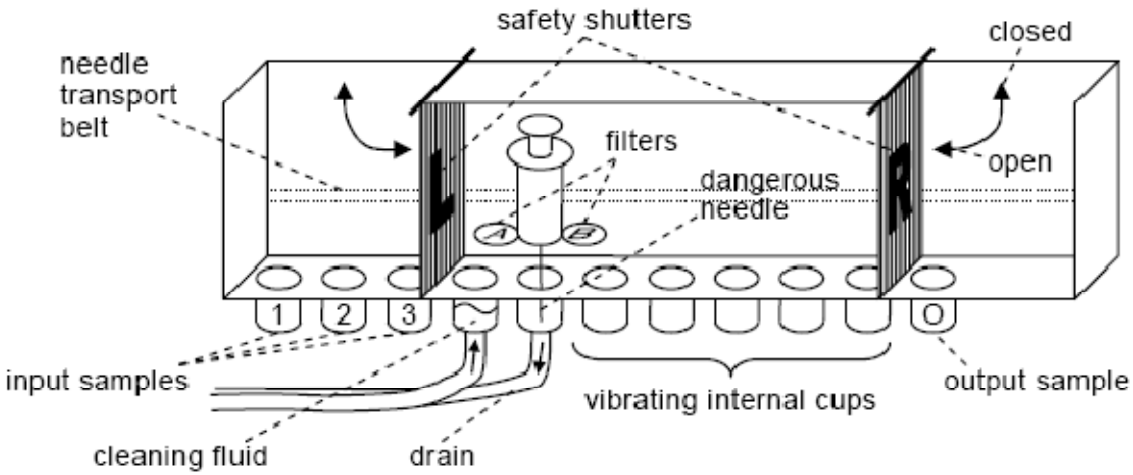
➢ Creation of new services 6 times faster compared to manual practices



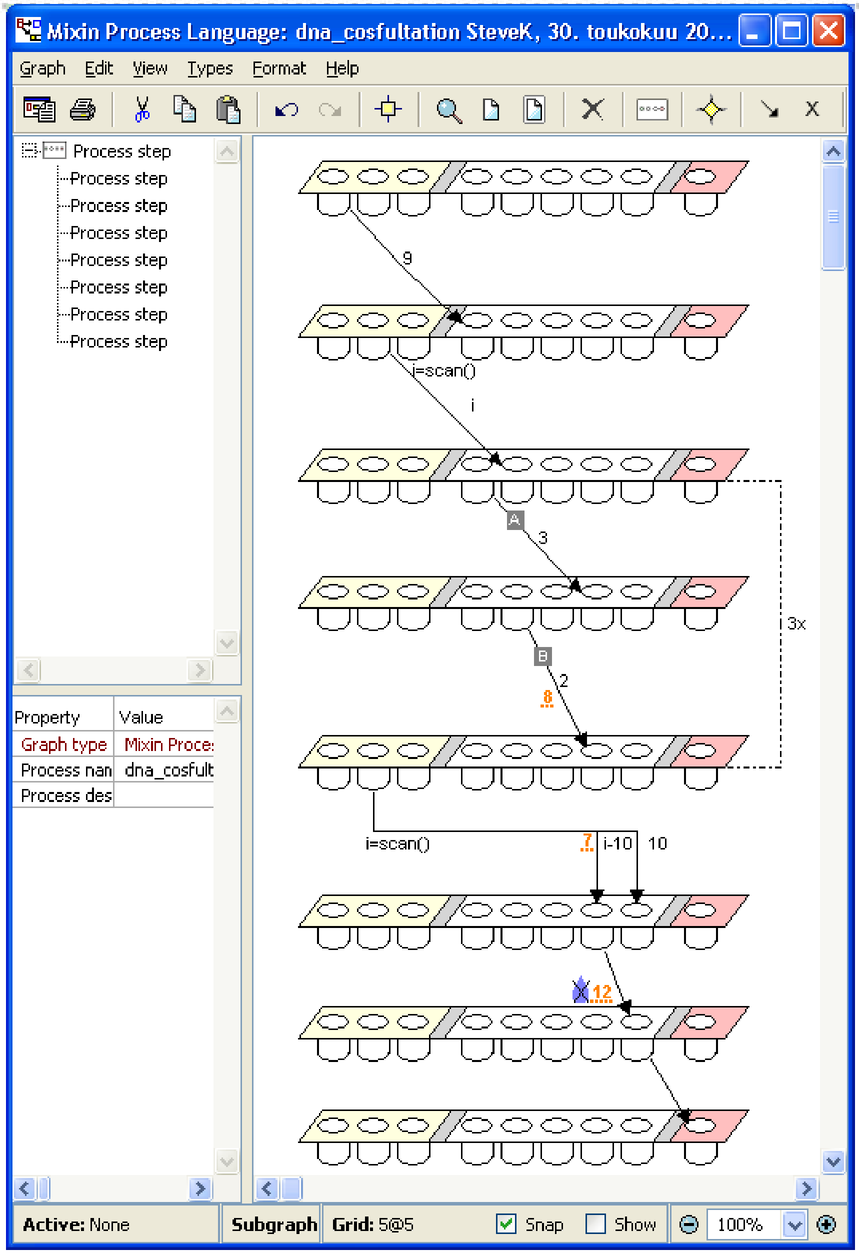


Case: Process of medical mixing

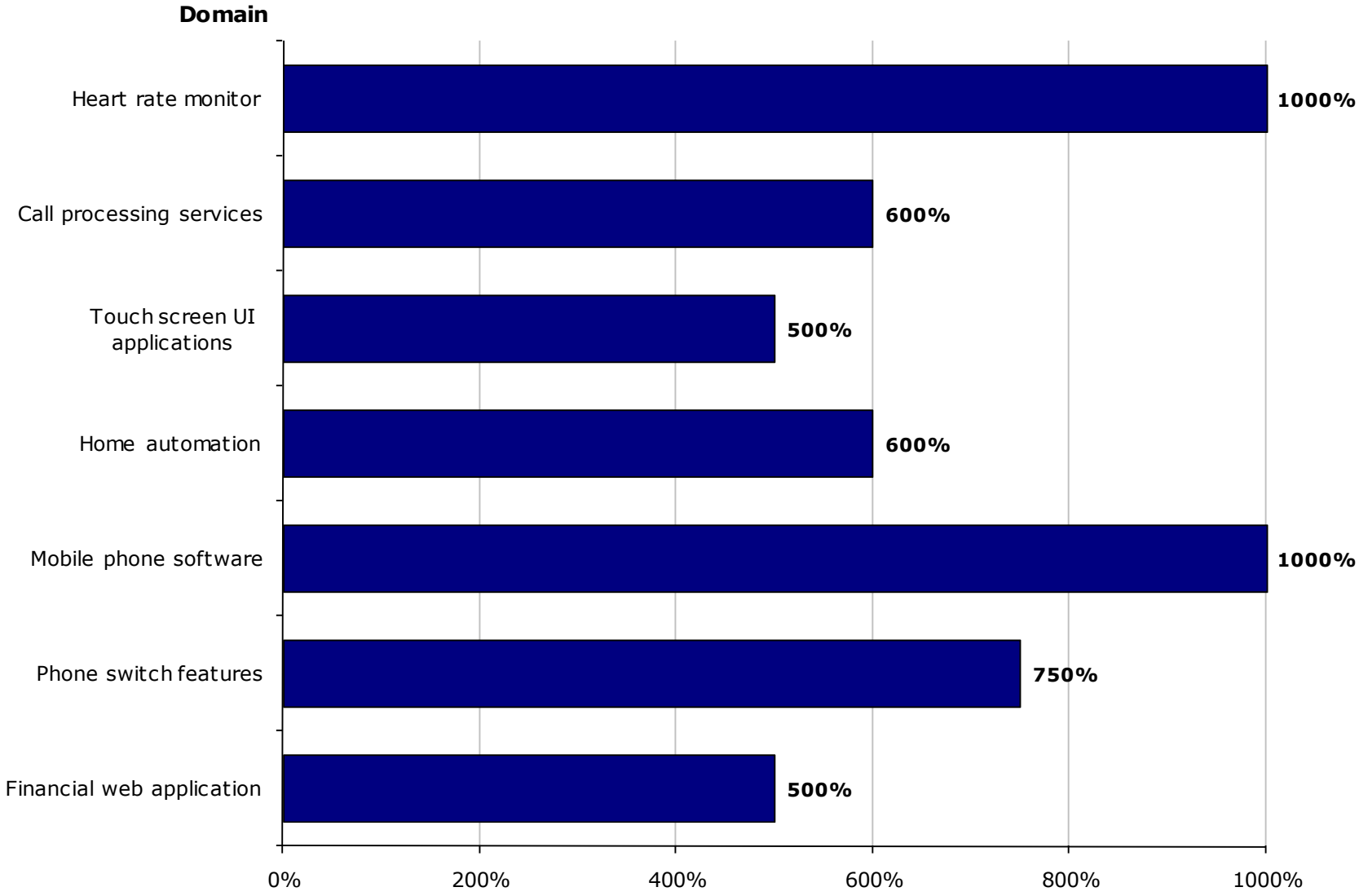
- Single fixed physical machine
- Product variability in software: how to mix
- Platform has 11 cups, syringe to transfer between
- Problems in hand-coded software quality:
  - Syringe broken, operator died, patient treatment error
- Competition by ICT to find best DSM solution
  - Reduced from 277 pieces of input to 29



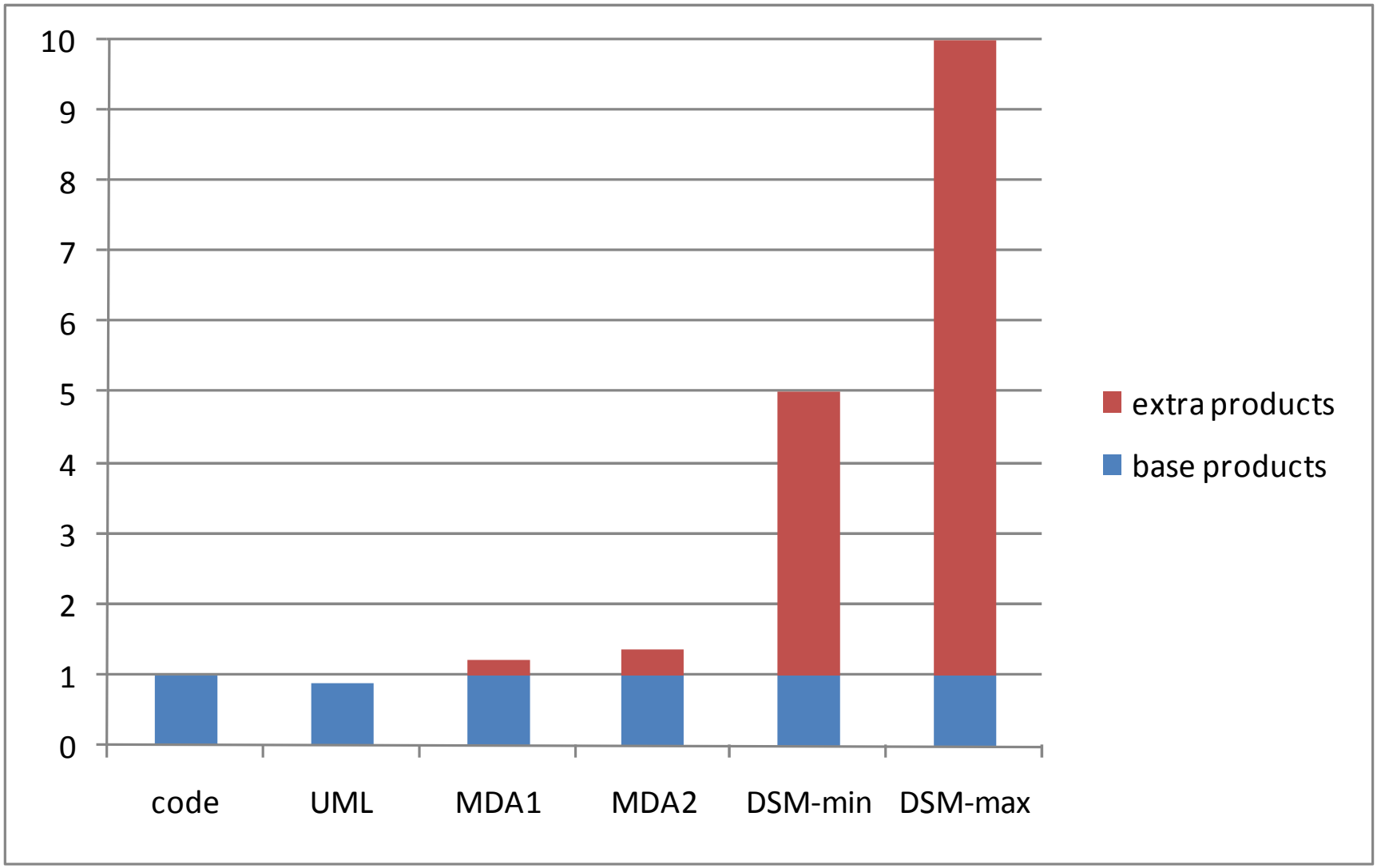
```
class DNAMixingMachine extends MixingMachine
{   int dna_cosfultation()
    {   double i,j;
        shut(0);
        move(-4);
        if(scan()!=9) return -1;
        suck(9);move(5);blow(9);
        move(-2);suck(30);move(1);blow(30);
        move(-3);
        if(scan()<6) return -1;
        i=scan();suck(i);move(5);open(0);blow(i);
        move(-3);suck(30);move(1);blow(30);
        move(1);filt(1);suck(3);move(3);filt(0);blow(3);
        move(-5);suck(30);move(1);blow(30);
        move(2);filt(2);suck(2);
        move(2);filt(0);blow(2);
        wait(8);
        move(-5);suck(30);move(1);blow(30);
        move(1);filt(1);suck(3);move(3);filt(0);blow(3);
        move(-5);suck(30);move(1);blow(30);
        move(2);filt(2);suck(2);
        move(2);filt(0);blow(2);
        wait(8);
        move(-5);suck(30);move(1);blow(30);
        move(1);filt(1);suck(3);move(3);filt(0);blow(3);
        move(-5);suck(30);move(1);blow(30);
        move(2);filt(2);suck(2);
        move(2);filt(0);blow(2);
        wait(8);
        move(-5);suck(30);move(1);blow(30);
        move(2);i=scan();suck(i);move(3);
        if(i<=10) {blow(i);j=i;}
        else {blow(10);j=10;}
        move(-1);blow(i-j);
        wait(7);
        move(-5);suck(30);move(1);blow(30);
        move(4);suck(3+2+3+2+3+2+1-j);move(1);blow(3+2+3+2+3+2+1-j);
        wait(12);
        suck(3+2+3+2+3+2+i);shut(1);move(1);blow(3+2+3+2+3+2+i);
        move(-7);open(1);suck(30);move(1);blow(30);
        return 0;
    }
```



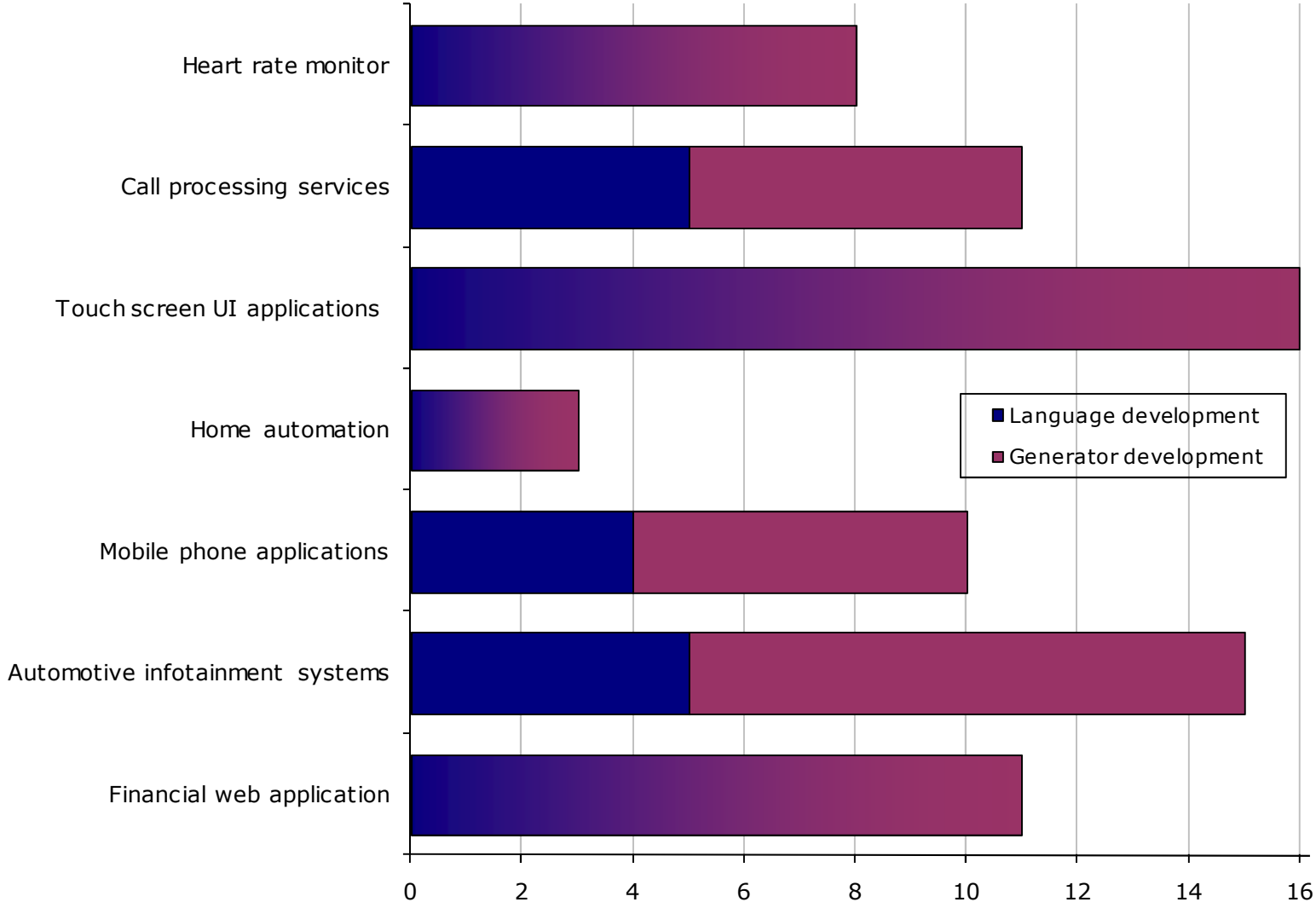
Productivity increase from DSM



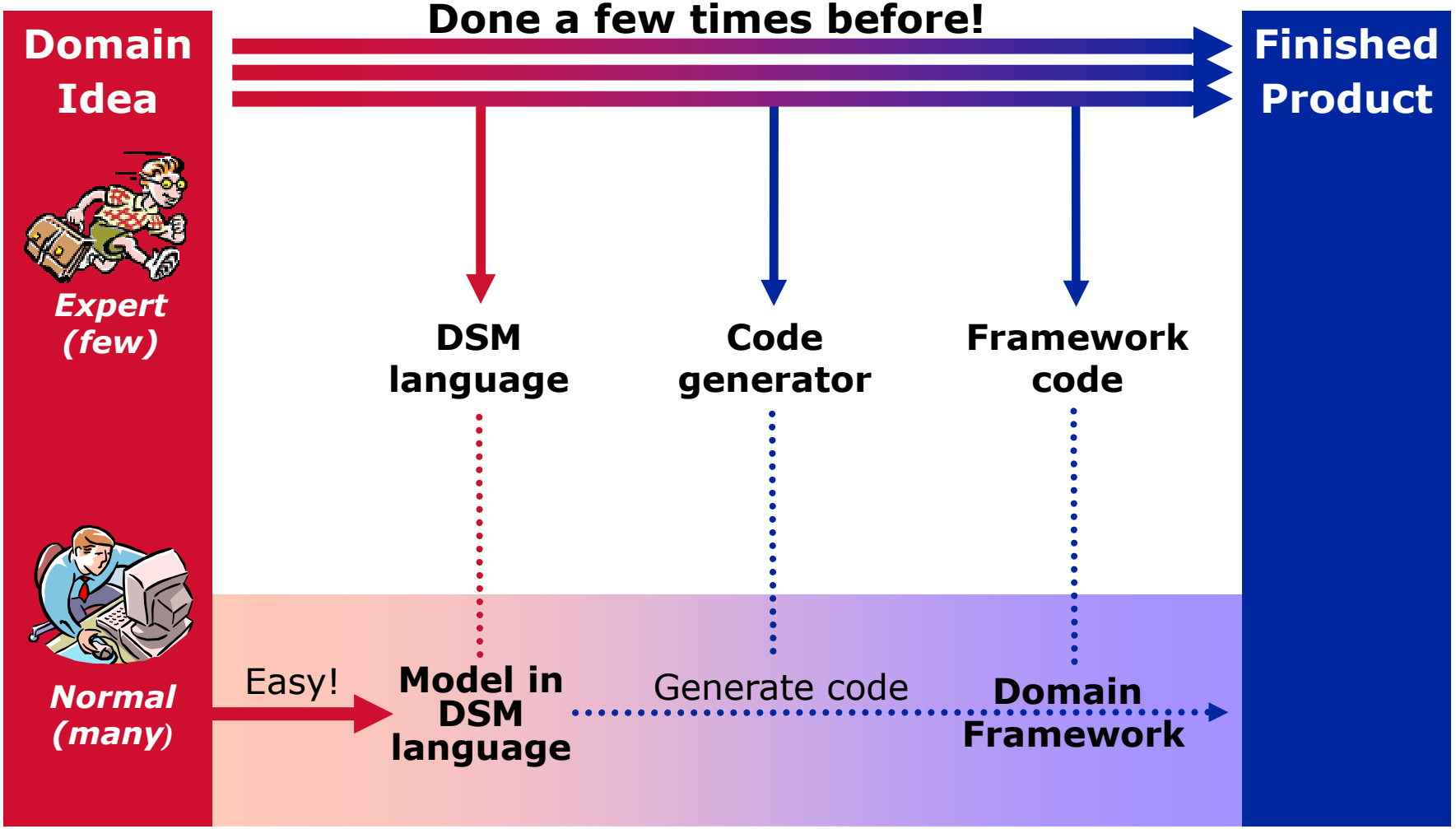
DSM consistently 5-10x faster



DSM Solution Development Time



How to implement DSM



Getting started

A journey of 1000 miles
begins with a single step

Setting off on the wrong foot
can spoil the whole journey

Only Gurus Allowed

Only gurus build languages **4%**

I'm smart & need no help **12%**

Domain Dilettante

Insufficient understanding:

- **Problem domain 17%**
- **Solution domain 5%**

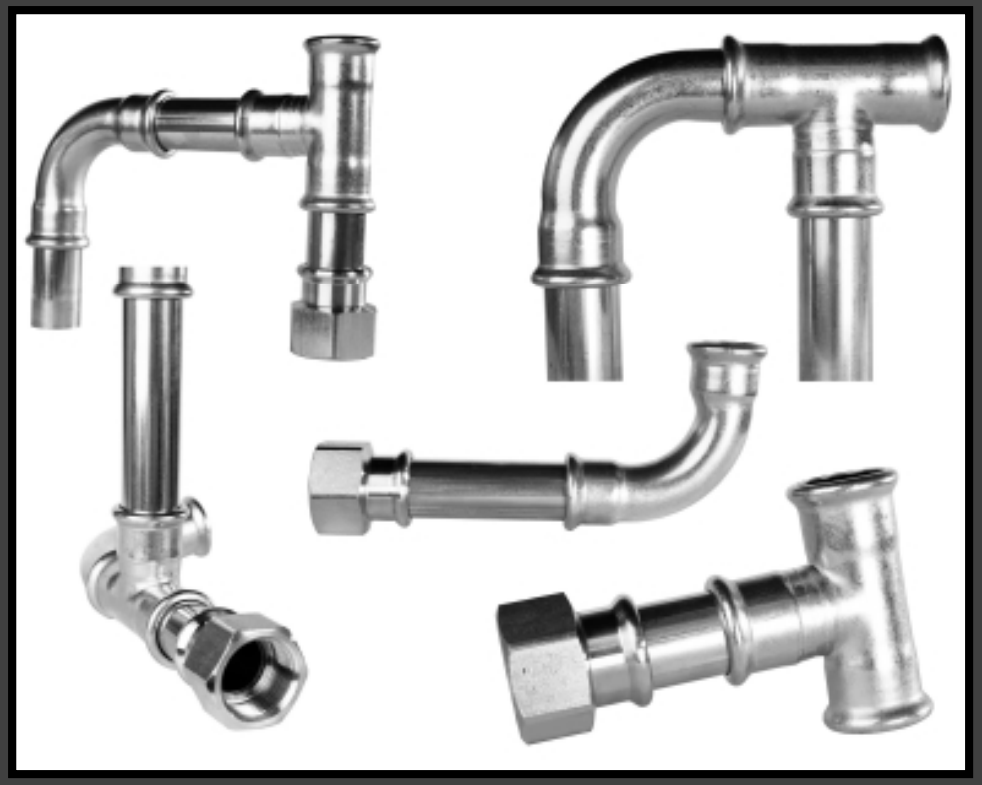
Never, ever delegate to interns!

Worst practices: Initial conditions

Analysis Paralysis

Language must be known to be complete, fully implementable 8%

Sources for the Language



Tool: hammer ⇒ nails

Tool's technical limitations dictate language **14%**

Approaches to identify concepts

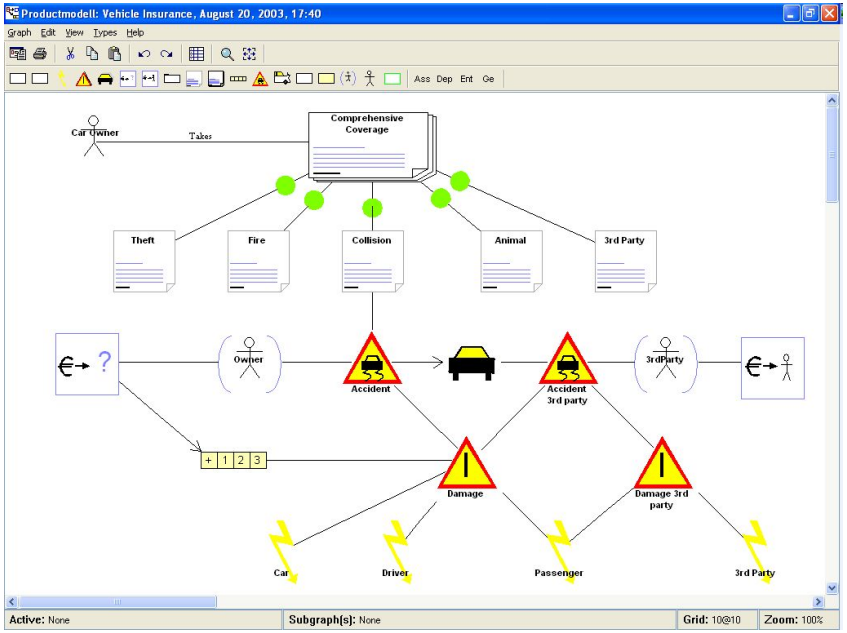
- "What concepts should my language have?"
  - Hard problem for DSM beginners
  - Analysed 23 cases to find good toolbox of approaches*
- 4 main approaches:

* Tolvanen, J.-P., Kelly, S., 2005:
Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences.
Proceedings of the 9th International Software Product Line Conference,
H. Obbink & K. Pohl (Eds.) Springer-Verlag, LNCS 3714, pp. 198–209

Problem domain	Solution domain/ generation target	Approach
Telecom services	Configuration scripts	1
Insurance products	J2EE	1
Business processes	Rule engine language	1
Industrial automation	3 GL	1, (2)
Platform installation	XML	1, (2)
Medical device configuration	XML	1, (2)
Machine control	3 GL	1, 2
Call processing	CPL	2, (1)
Geographic Information System	3 GL, propriety rule language, data structures	2
SIM card profiles	Configuration scripts and parameters	2
Phone switch services	CPL, Voice XML, 3 GL	2, (3)
eCommerce marketplaces	J2EE, XML	2, (3)
SIM card applications	3 GL	3
Applications in microcontroller	8-bit assembler	3
Household appliance features	3 GL	3
Smartphone UI applications	Scripting language	3
ERP configuration	3 GL	3, 4
ERP configuration	3 GL	3, 4
Handheld device applications	3 GL	3, 4
Phone UI applications	C	4, (3)
Phone UI applications	C++	4, (3)
Phone UI applications	C	4, (3)
Phone UI applications	C++	4, (3)

1. Domain expert's concepts

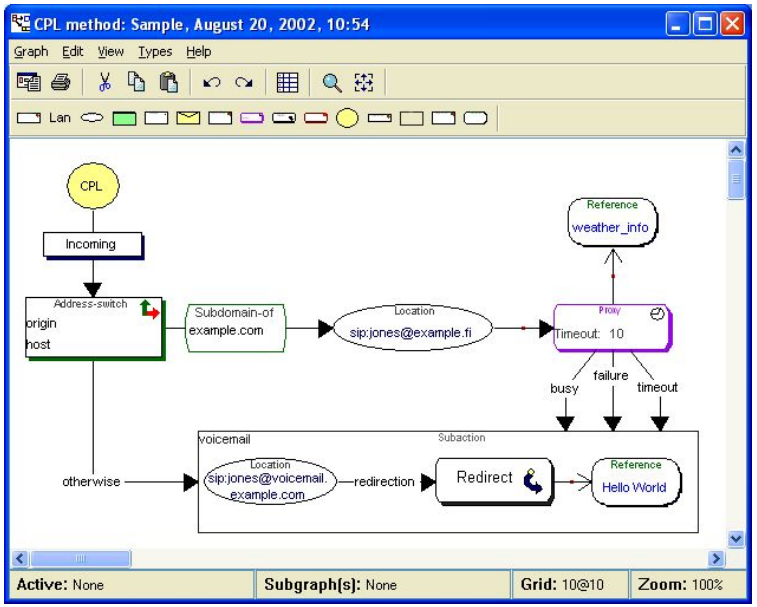
- Concepts from domain
- Mostly made without help
- Simple code generation
- OK in established domain
- Usable by non-coders



Insurance products/J2EE

2. Generation output

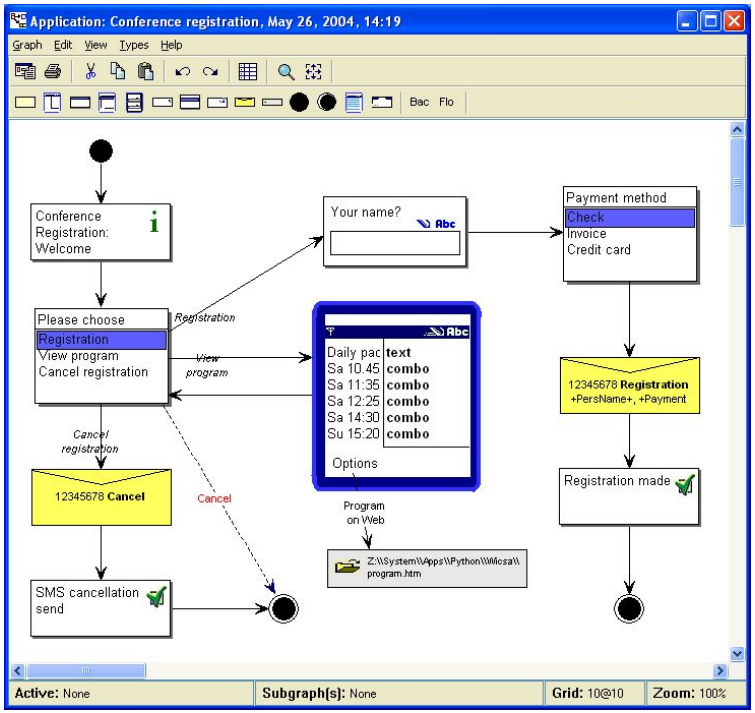
- Modelling constructs come from code artefacts
- Static parts are easy
  - Data structures
  - Core XML elements
- Dynamic behaviour harder
  - Avoid "graphical 3GL"
  - Need domain framework
- Danger: low level of abstraction
  - Little productivity gain
- But works well with DSL or XML



Internet telephony/CPL

3. Look and feel of end system

- Best for physical end product
  - UI on PC, embedded, speech
- Often state machine basis
  - Extend with data & control flow
  - Power of relationships
- Visible domain concepts
  - Easy to identify
  - High level of abstraction
- Domain framework hides code
  - Don't write code in models…
  - …unless you really have to!
- Generators considered easy



Smartphone apps/Python

4. Variability space

- Language concepts capture variability space
- Modeler makes variant choices
  - Composition, relationships, values
- Infinite variability space (Czarnecki)
  - Not just feature tree: unbounded product family
- Static variance easy, dynamic harder
- Predict future variability ⇒ high level of abstraction

Evaluation of the Approaches

- Hierarchy of approaches
  - From less to more experienced DSM practitioners
- 1. Domain expert's concepts – "we just did it"
- 2. Generation output
  - Generic/ad hoc language not so good
  - Established DSL good
- 3. Look and feel: common, easy, true DSM
- 4. Variability space: adds power to handle complexity
  - Found in very different domains

- Best results combined 3 (L&F) and 4 (Variability)

Too generic/specific

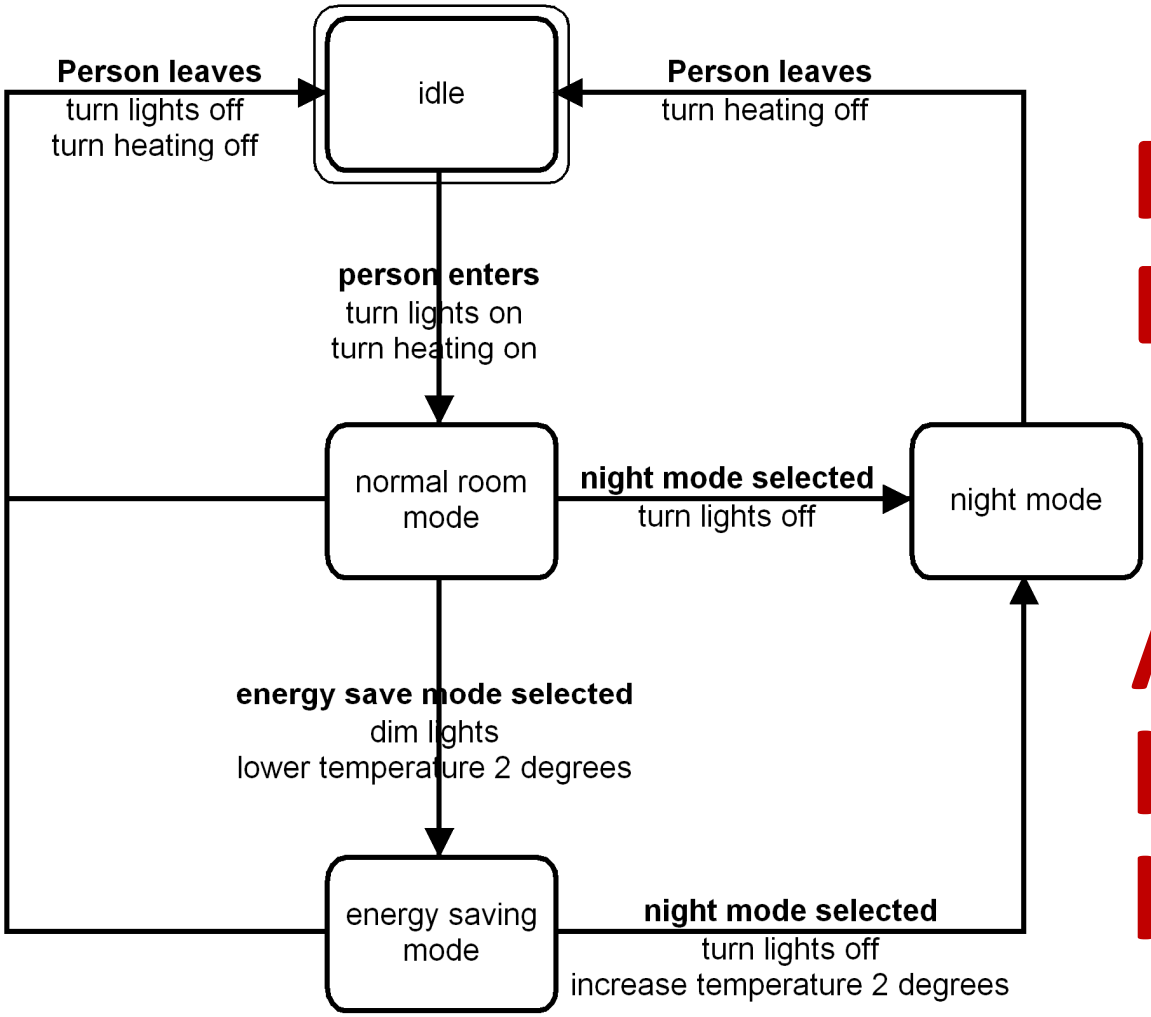
Too few/generic	21%
Too many/specific	8%
Language for 1 model	7%

Simplistic symbols

Too simple/similar	25%
Downright ugly	5%

Worst practices: Concept Source

Not like this…

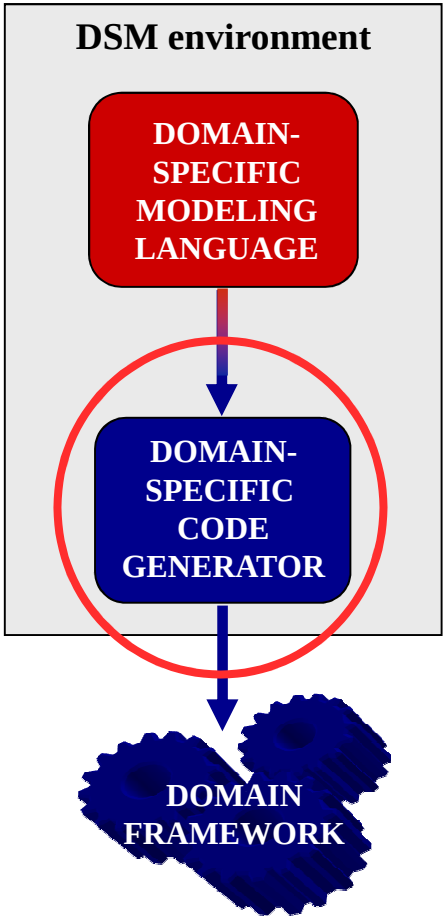


Not a DSL!

Add heating, lights?

What about the generator?

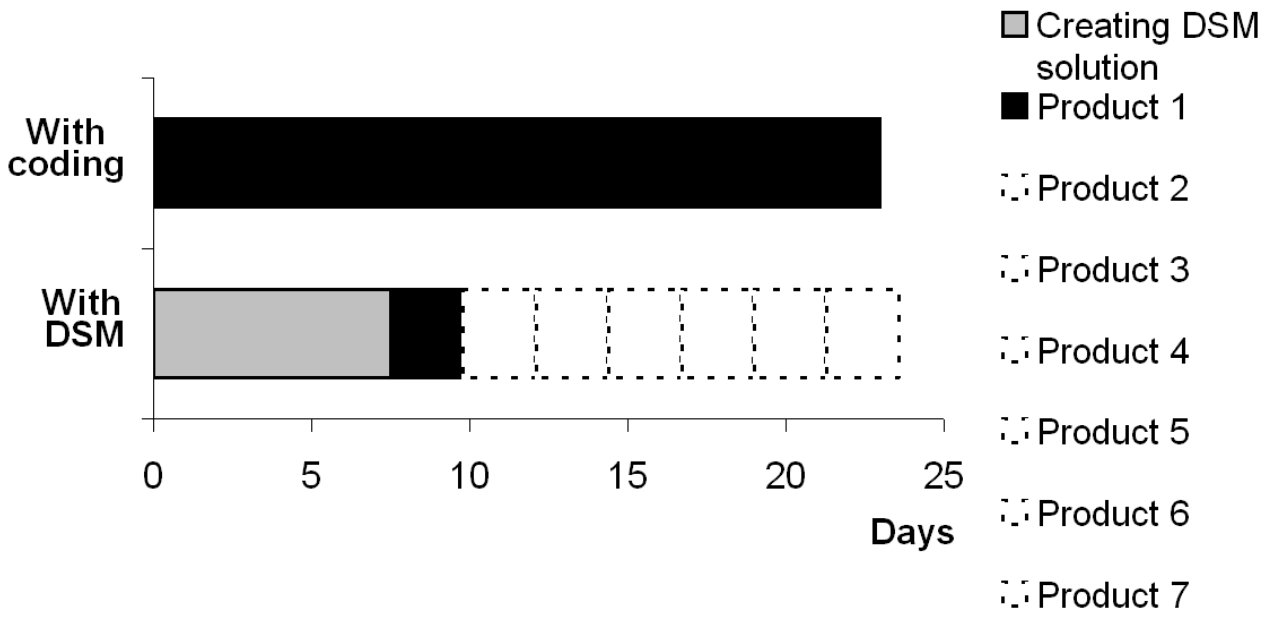
- Generator translates the models into the required output
  1. crawls through the models
     → navigation based on metamodel
  2. extracts required information
     → access data in models
  3. outputs it into the code
     → mixing fixed text and model data
  4. with translation where necessary
     → e.g. space to underscore, XML legal



How to make a generator

- Make generated code based on current hand-written code
  - Removes risk of slow, bloated or unreadable code
  - Follow good coding standards, include comments, have data to link back to models (e.g. in comment or via simulator)
- Make generation process complete, target 100% output
  - Never modify the generated code
    - Correct the generator or framework instead
    - Or use add-in hand-coded functions
- Put domain rules up-front to the language
    - Generator definition becomes easier when the input is correct
- Try to generate as little code as possible
  - Glue code only, rest in domain framework or platform
- Keep generator as simple as possible
  - Push low-level implementation issues to the framework
  - Keep generator modular to reflect changes

Is DSM worth it?



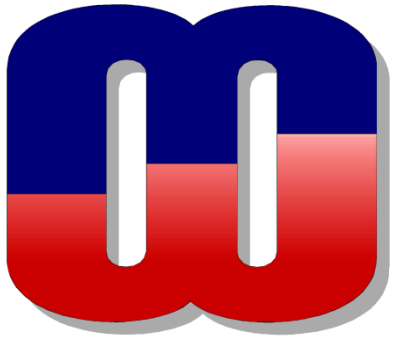
- Language workbenches make moving to DSM feasible
  - Can focus on language design, not on creating tooling
- Creation of languages does not take much time

Is DSM for you?

- You probably can't know yet!
  - Have to try it out for your situation
- If it does fit your situation, the 500-1000% makes it probably the single most important thing you can do
  - More than SOA, cloud, Ruby, agile, …
- How to see quickly if it would work for you:
  - Look for repetition in your work, e.g.:
    - many screens and database tables
    - many products in family
    - etc.
  - Try out a tool
    - Tutorials
    - Have a go with your domain (ask for hints)

Thank you!

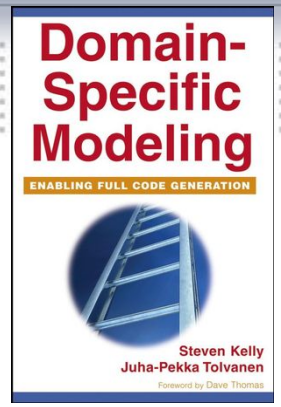
Questions?



MetaCase

www.metacase.com

Literature and further links



- **DSM Forum**, www.dsmforum.org
- **Blogs**: www.metacase.com/blogs
- Brinkkemper, S., Lyytinen, K., Welke, R., **Method Engineering** - Principles of method construction & tool support, Chapman & Hall, 1996
- Kelly, S., Pohjonen, R., **Worst Practices for Domain-Specific Modeling**, IEEE Software, DSM special issue, July/Aug, 2009 www.metacase.com/stevek.html
- Kelly, S., Tolvanen, J.-P., **Domain-Specific Modeling**: Enabling Full Code Generation, Wiley, 2008. http://dsmbook.com
- Kieburtz, R. et al., **A Software Engineering Experiment in Software Component Generation**, Proceedings of 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, 1996.
- Tolvanen, J.-P., Kelly, S., **Defining Domain-Specific Modelling Languages to Automate Product Derivation**: Collected Experiences. Procs of the 9th International Software Product Line Conference, Springer-Verlag, 2005.
- Weiss, D., Lai, C. T. R., **Software Product-line Engineering**, Addison Wesley Longman, 1999.