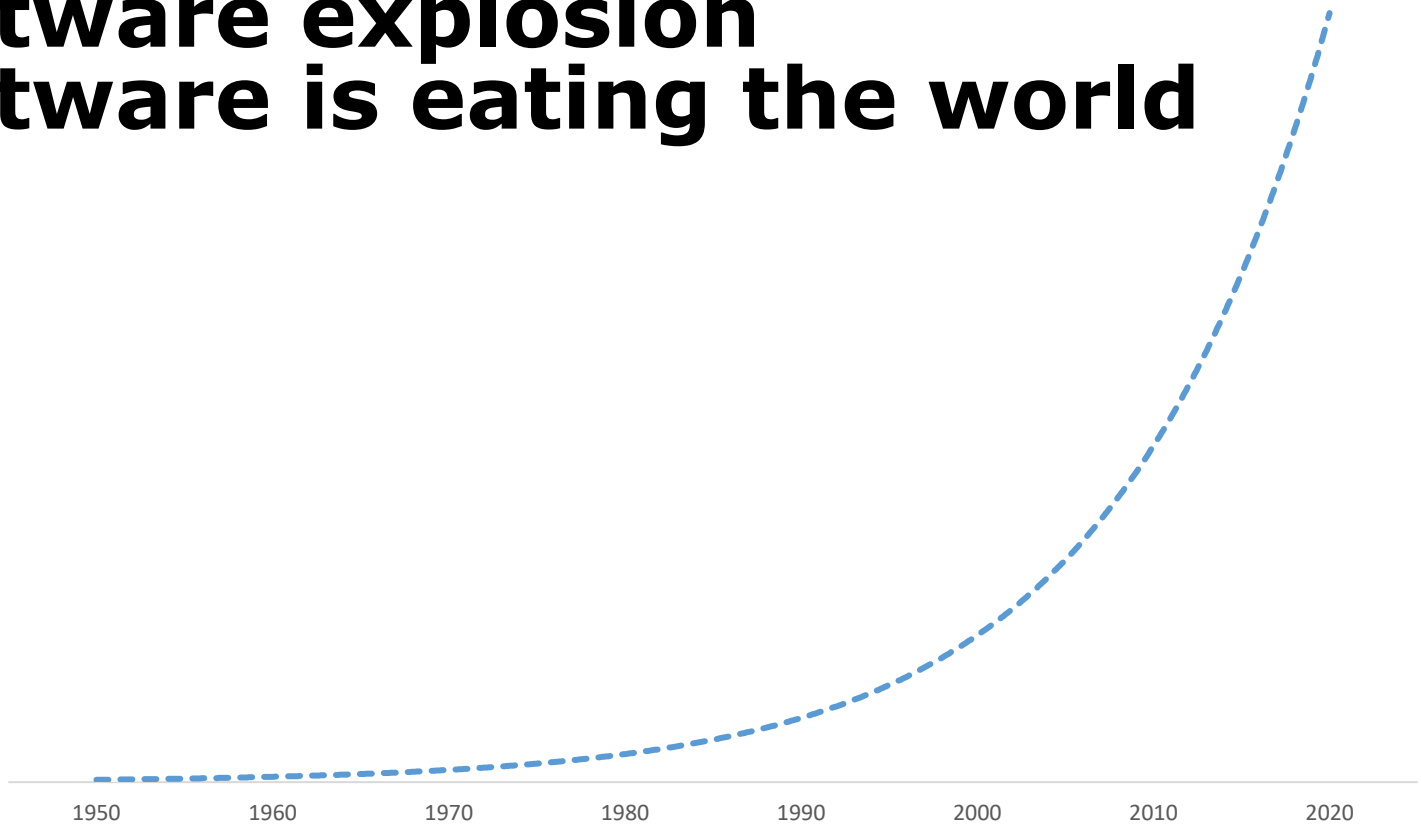
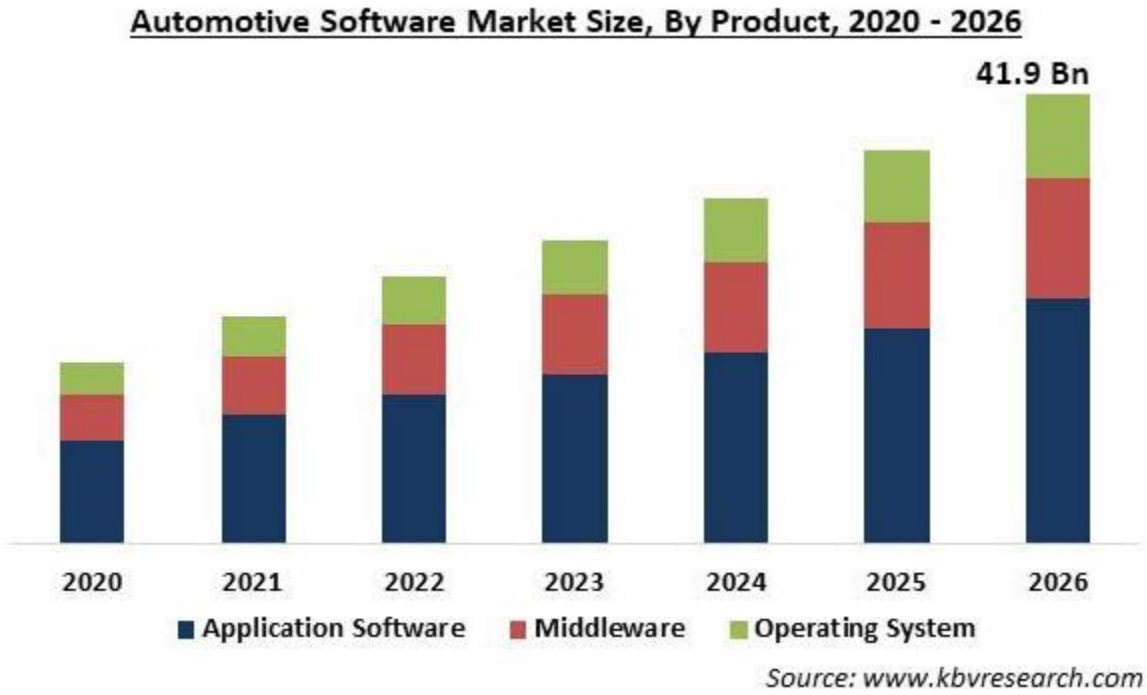
Languages for non-developers
what, how, where?

Juha-Pekka Tolvanen
jpt@metacase.com

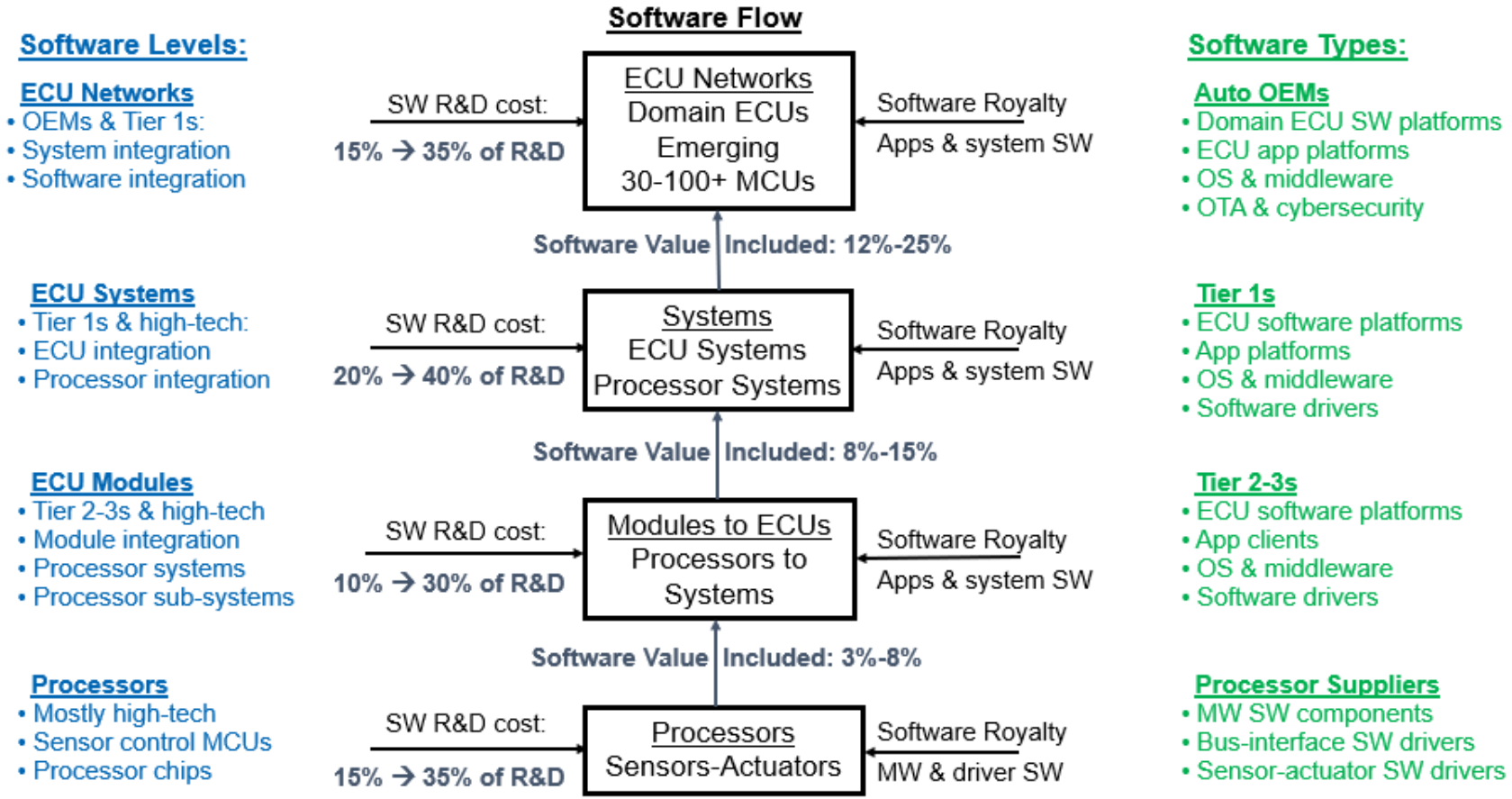
Software everywhere
Software explosion
Software is eating the world



Continuous growth of functionality



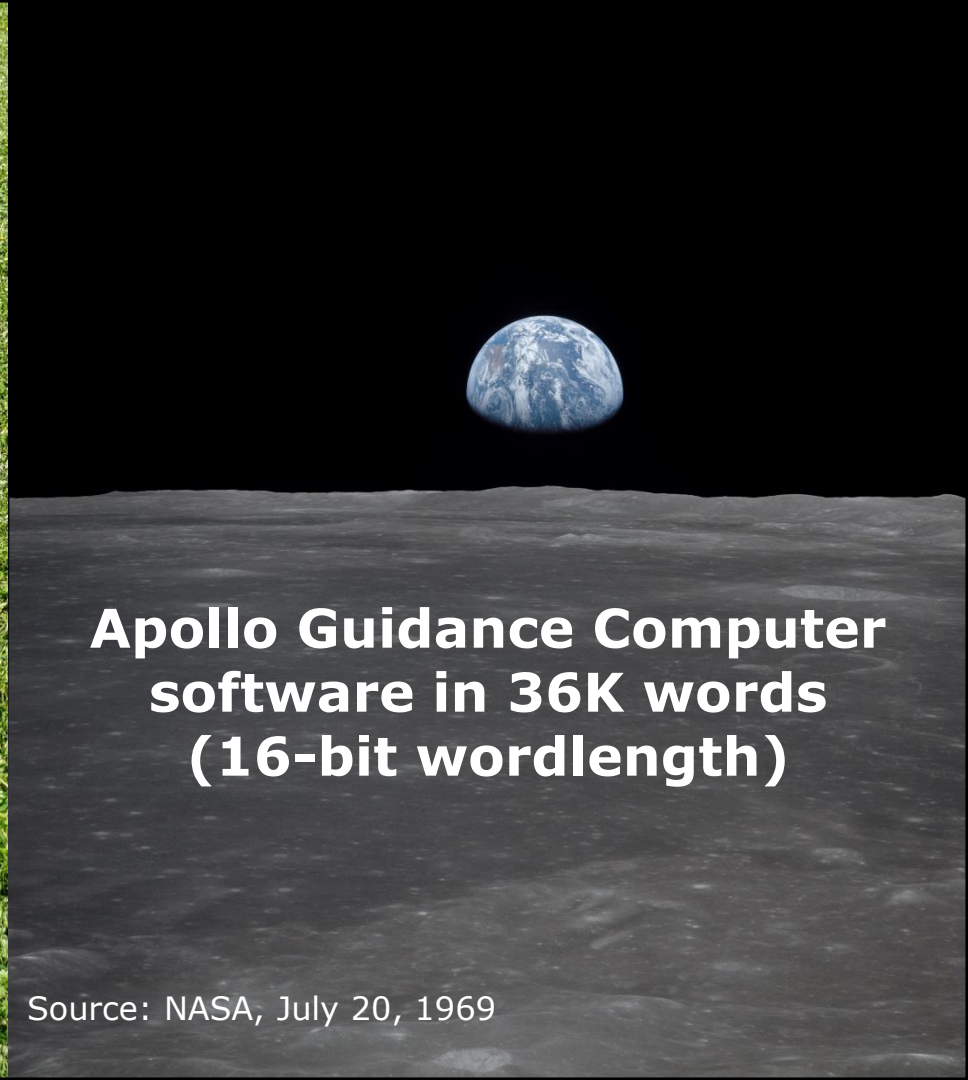
Auto Software Cost & Value Flow



Source: Egil Juliussen; May 2021

ECU=Electronic Control Unit; MW=Middleware; OS=Operating System; SW=Software

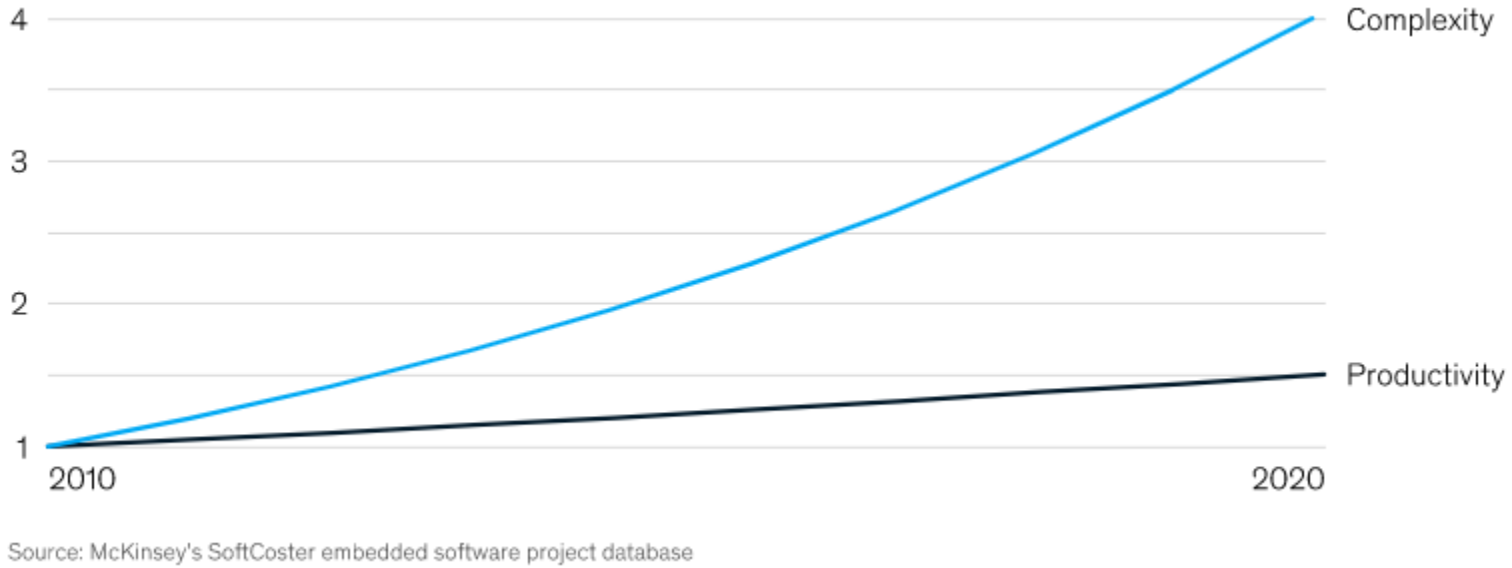
Lawn mower robot: software upgrade: 527 MB (.msi package)



Apollo Guidance Computer software in 36K words (16-bit wordlength)

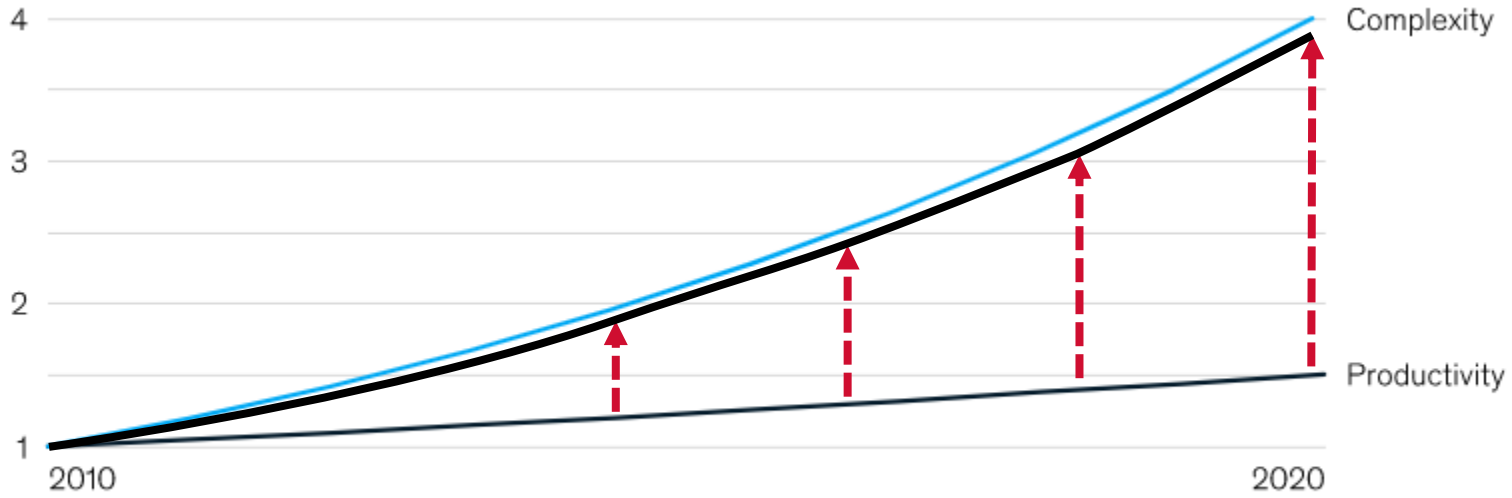
Source: NASA, July 20, 1969

Also complexity is growing faster than software development productivity



Source: McKinsey's SoftCoster embedded software project database

How to improve productivity?

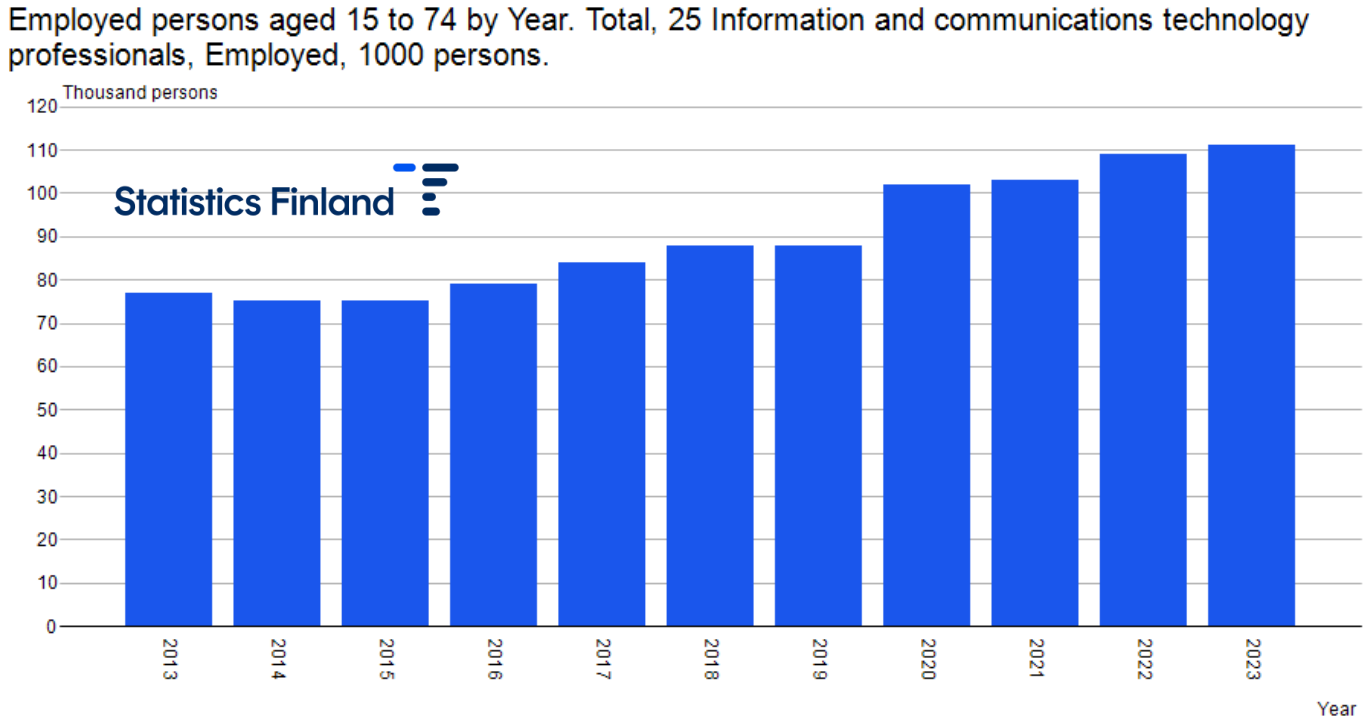


Source: McKinsey's SoftCoster embedded software project database

Solution?

- More developers
- New frameworks
- Better tools
- More automation
- AI
- ?
- **Better languages, also for non-developers**

Solution?



API query: https://pxdata.stat.fi:443/PxWeb/api/v1/en/StatFin/tyti/statfin_tyti_pxt_13au.px

Abstraction

```
0111 1111
0100 0101
0100 1100
0100 0110
0000 0010
0000 0001
0000 0000
0000 0000
0000 0010
0010 0000
0000 0010
0000 0010
0000 0000
0000 0000
```

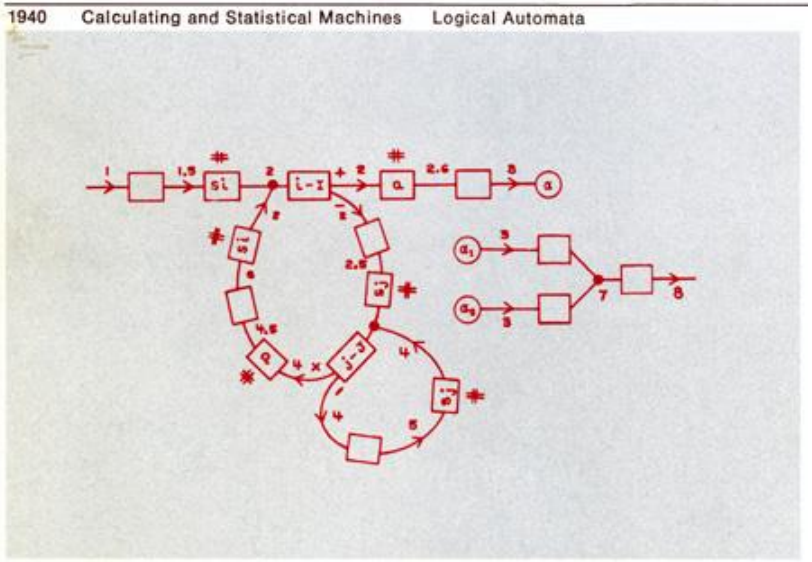
```
org 100h
 mov ah,9h
 mov dx,offset text
 int 21h
 ret
text: db 'hello$'
```

```
int main(void)
{
 printf("Hello\n");
 return 0;
}
```

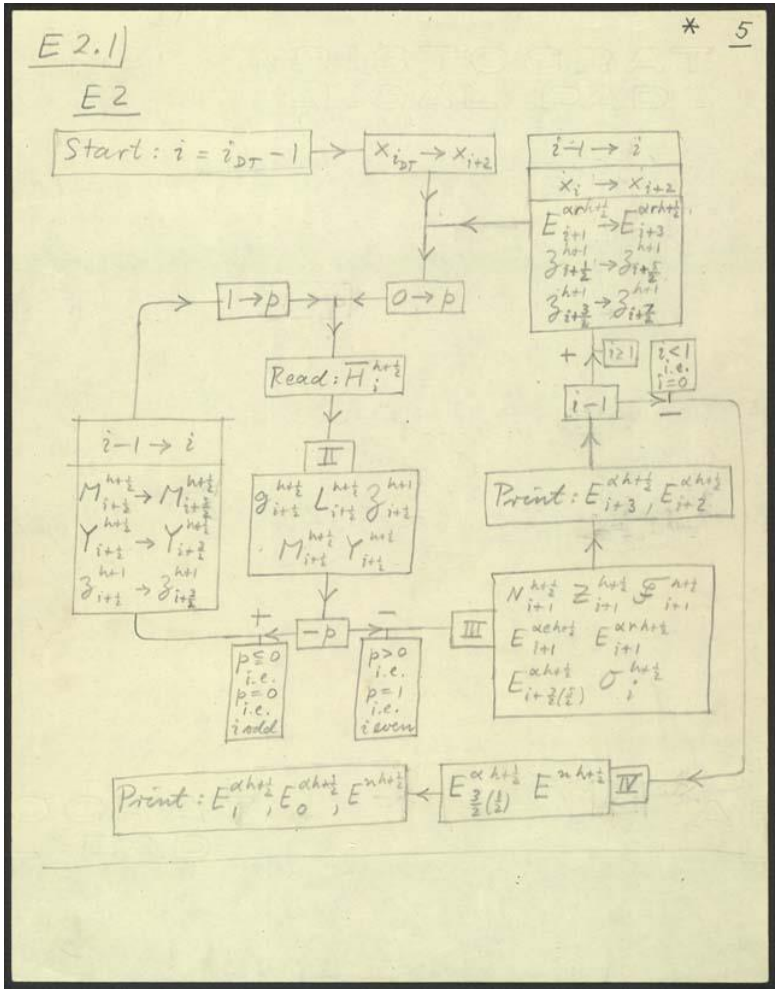

```
int main(void)
{
 cout<<"Hello"<<endl;
 return 0;
}
```

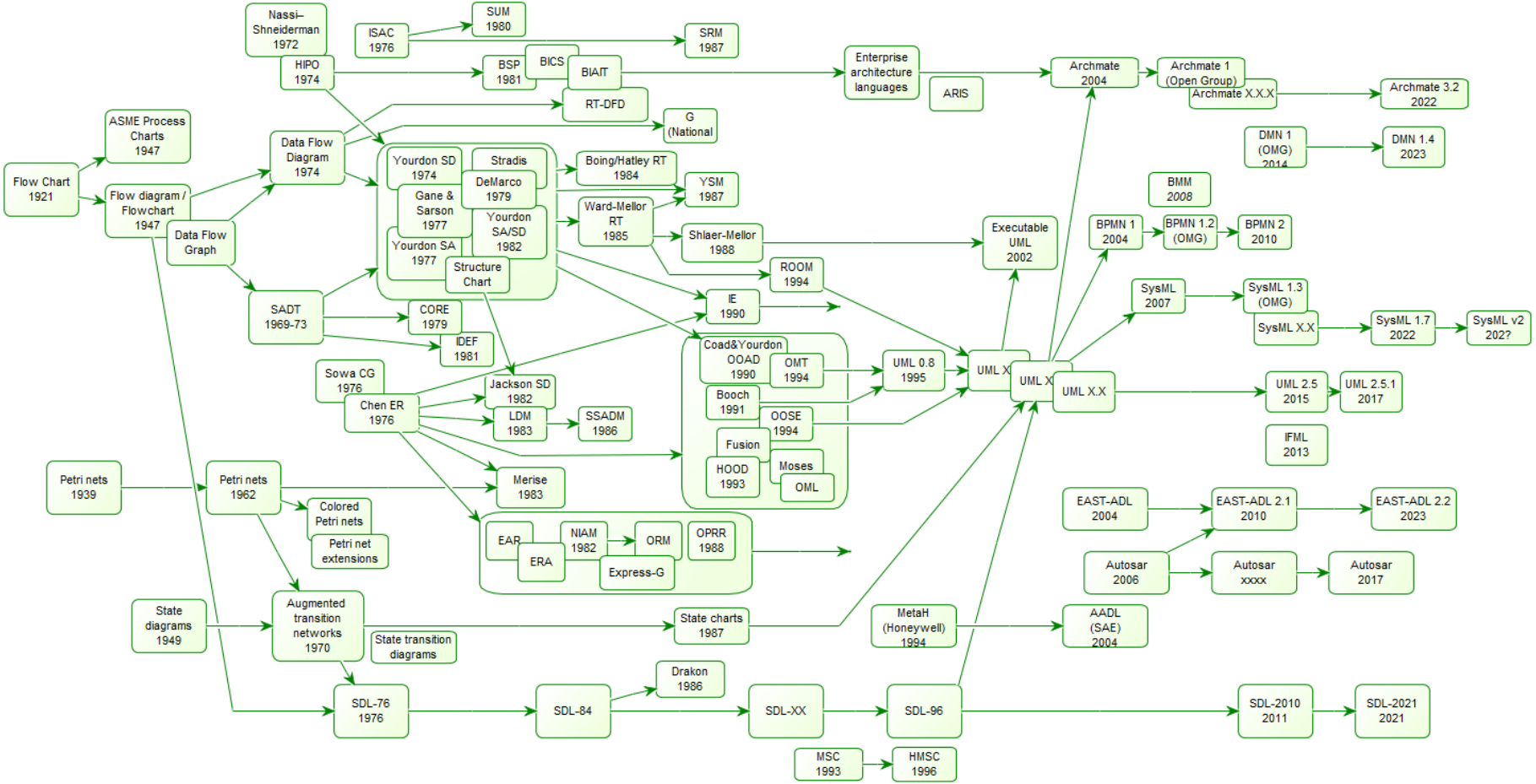

```
Transcript show:'Hello'.
```

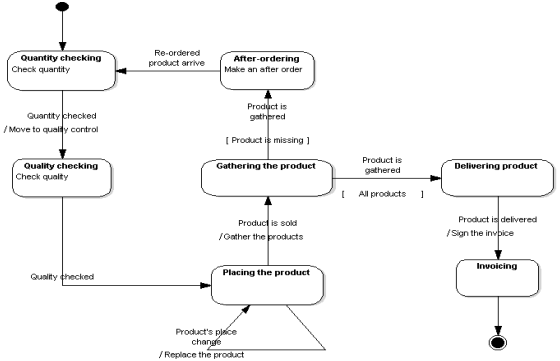
Automation

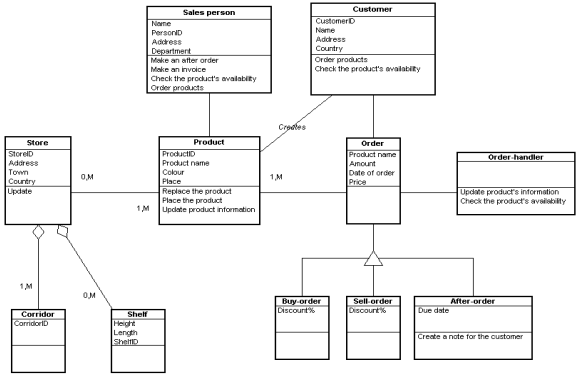


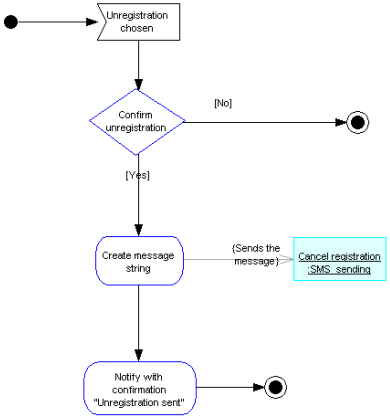
Flow diagram for ENIAC, John von Neumann, 1950 Manuscript Division, Library of Congress (164)



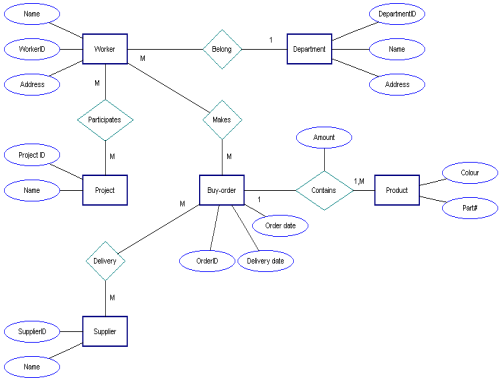

Nassi–Shneiderman 1972
ISAC 1976
SUM 1980
SRM 1987
HIPO 1974
BSP 1981
BICS
BIAIT
RT-DFD
G (National
Enterprise architecture languages
ARIS
Archmate 2004
Archmate 1 (Open Group)
Archmate X.X.X
Archmate 3.2 2022
ASME Process Charts 1947
Flow Chart 1921
Flow diagram / Flowchart 1947
Data Flow Diagram 1974
Data Flow Graph
Yourdon SD 1974
Stradis
DeMarco 1979
Gane & Sarson 1977
Yourdon SA/SD 1982
Yourdon SA 1977
Structure Chart
Boing/Hatley RT 1984
YSM 1987
Ward-Mellor RT 1985
Shlaer-Mellor 1988
ROOM 1994
IE 1990
DMN 1 (OMG) 2014
DMN 1.4 2023
BMM 2008
BPMN 1 2004
BPMN 1.2 (OMG)
BPMN 2 2010
Executable UML 2002
SysML 2007
SysML 1.3 (OMG)
SysML X.X
SysML 1.7 2022
SysML v2 202?
SADT 1969-73
CORE 1979
IDEF 1981
Sowa CG 1976
Chen ER 1976
Jackson SD 1982
LDM 1983
SSADM 1986
Merise 1983
Coad&Yourdon OOAD 1990
OMT 1994
Booch 1991
OOSE 1994
Fusion
HOOD 1993
Moses
OML
UML 0.8 1995
UML X
UML X
UML X.X
UML 2.5 2015
UML 2.5.1 2017
IFML 2013
Petri nets 1939
Petri nets 1962
Colored Petri nets
Petri net extensions
EAR
ERA
NIAM 1982
ORM
Express-G
OPRR 1988
EAST-ADL 2004
EAST-ADL 2.1 2010
EAST-ADL 2.2 2023
Autosar 2006
Autosar xxxx
Autosar 2017
State diagrams 1949
Augmented transition networks 1970
State transition diagrams
State charts 1987
MetaH (Honeywell) 1994
AADL (SAE) 2004
SDL-76 1976
SDL-84
Drakon 1986
SDL-XX
SDL-96
SDL-2010 2011
SDL-2021 2021
MSC 1993
HMSC 1996

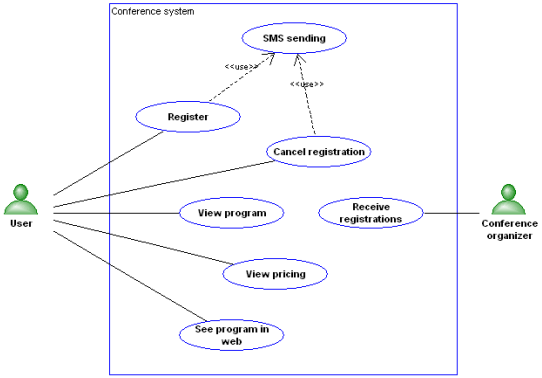


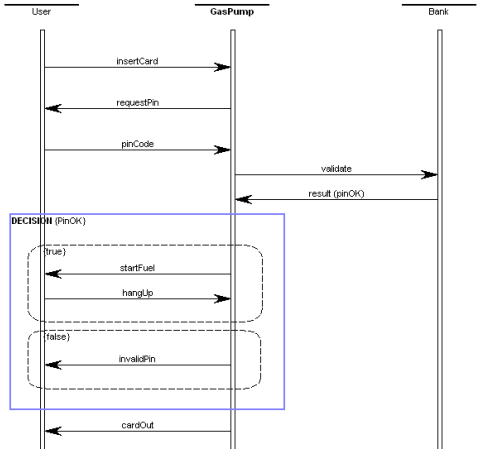


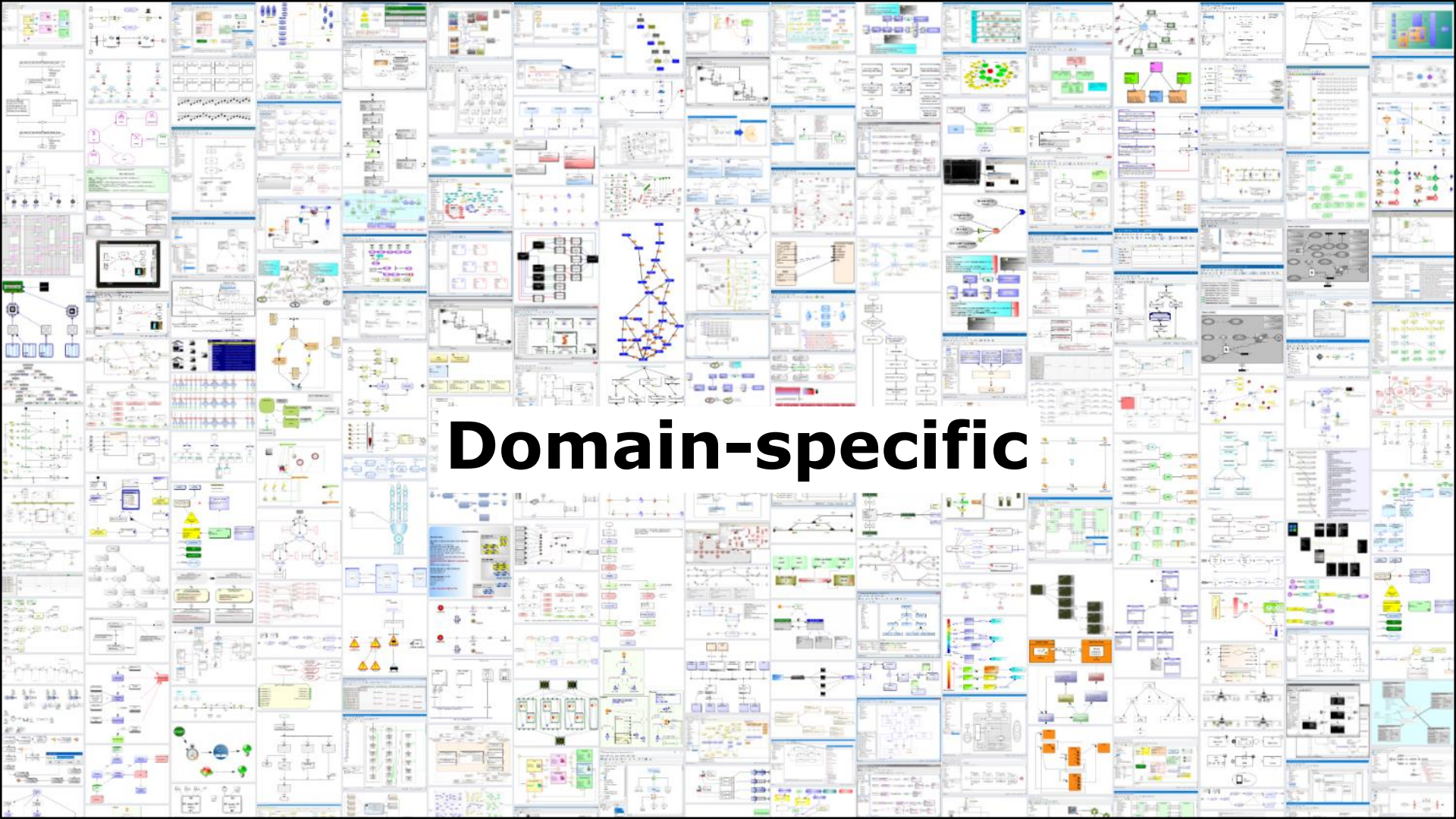


General purpose





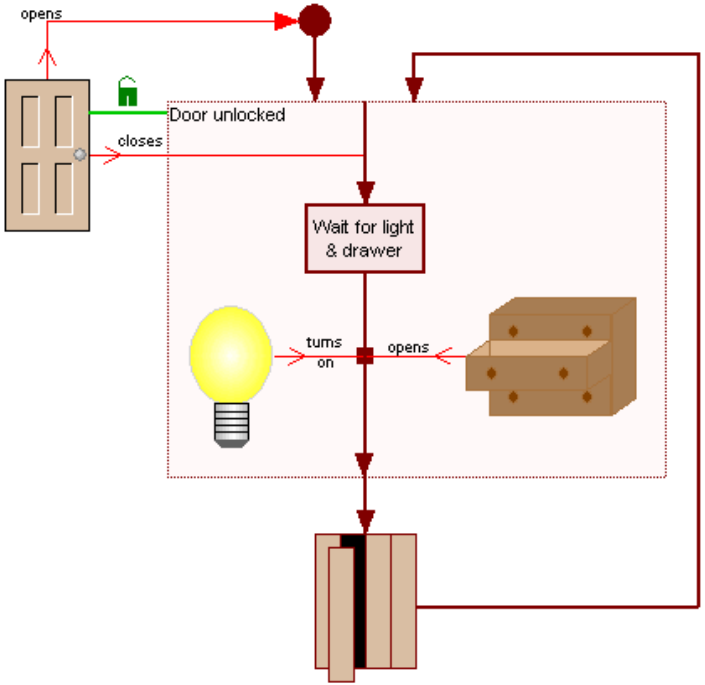



Domain-specific

Domain-Specific Languages

- **Narrow**, very narrow
- Aim for 100% **fit** with the problem domain
- **Raise** the level of abstraction, **hide** unnecessary details
- External and internal

Gothic Security*

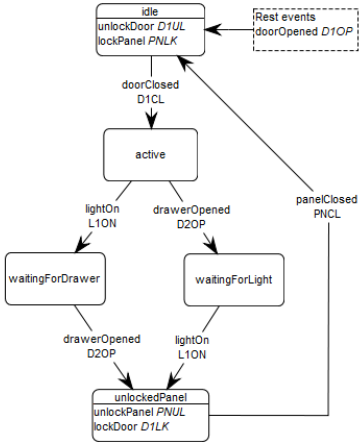


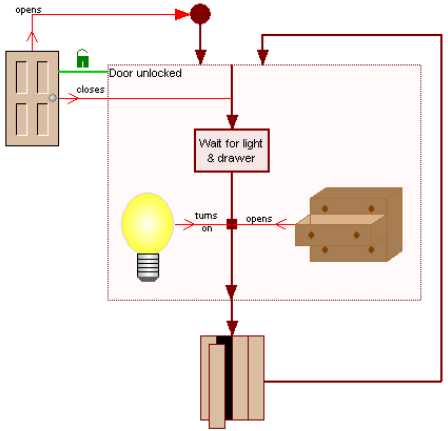

```
Event doorClosed = new Event("doorClosed", "D1CL");Builder {
Event doorOpened = new Event("doorOpened", "D1OP");Closed;
Event lightOn = new Event("lightOn", "L1ON");, unlockDoor;
Event drawerOpened = new Event("drawerOpened", "D2OP");wer, unlockedPanel;
Event panelClosed = new Event("panelClosed", "PNCL");

Command unlockDoorCmd = new Command("unlockDoor", "D1UL");
Command lockPanelCmd = new Command("lockPanel", "PNLK");
Command unlockPanelCmd = new Command("unlockPanel", "PNUL");
Command lockDoorCmd = new Command("lockDoor", "D1LK");
        panelClosed.code("PNCL");
StateMachine machine = new StateMachine(idle);
        unlockPanel.code("PNUL");
State activeState = new State("active");
State idleState = new State("idle");
State unlockedPanelState = new State("unlockedPanel");
State waitingForDrawerState = new State("waitingForDrawer");
State waitingForLightState = new State("waitingForLight");
        .actions(unlockDoor, lockPanel)
activeState.addTransition(lightOn, waitingForDrawerState);
activeState.addTransition(drawerOpened, waitingForLightState);

idleState.addAction(unlockDoorCmd);
idleState.addAction(lockPanelCmd);to(waitingForLight)
idleState.addTransition(doorClosed, activeState);
        ;
unlockedPanelState.addAction(unlockPanelCmd);
unlockedPanelState.addAction(lockDoorCmd);
unlockedPanelState.addTransition(panelClosed, idleState);
        ;
waitingForDrawerState.addTransition(drawerOpened, unlockedPanelState);
        ;
waitingForLightState.addTransition(lightOn, unlockedPanelState);
        unlockedPanel
machine.AddResetEvents(doorOpened);oor)
        .transition(panelClosed).to(idle)
        ;
    }
}
```

*Fowler, Domain-Specific Languages, Addison-Wesley, 2008

Domain-Specific Languages

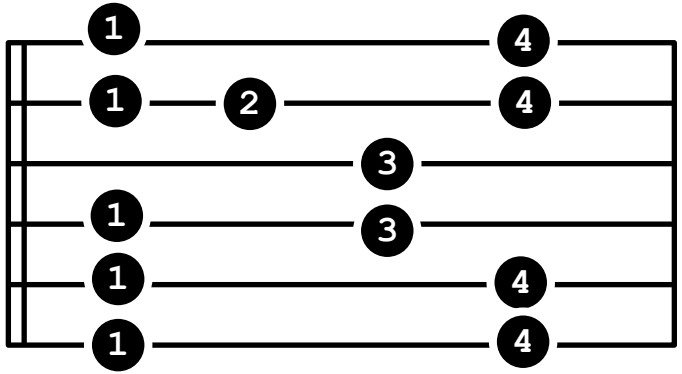
- **Narrow**, very narrow
- Aim for 100% **fit** with the problem domain
- **Raise** the level of abstraction, **hide** unnecessary details
- External and internal

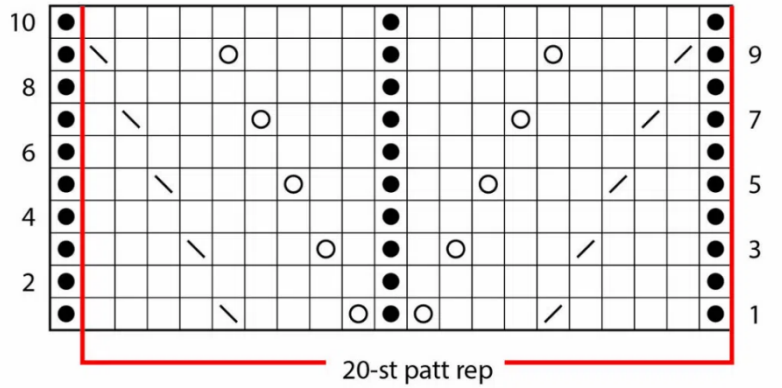


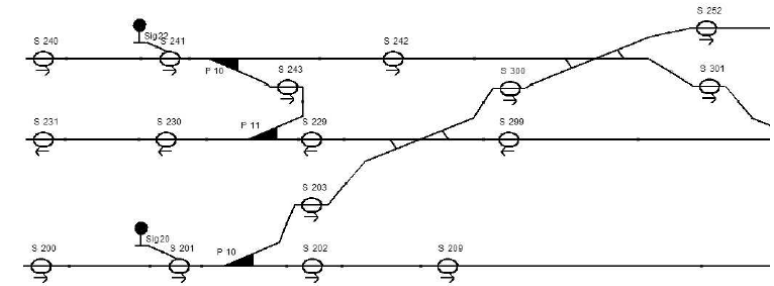


- Not a new idea!
- Applied also outside the software world!

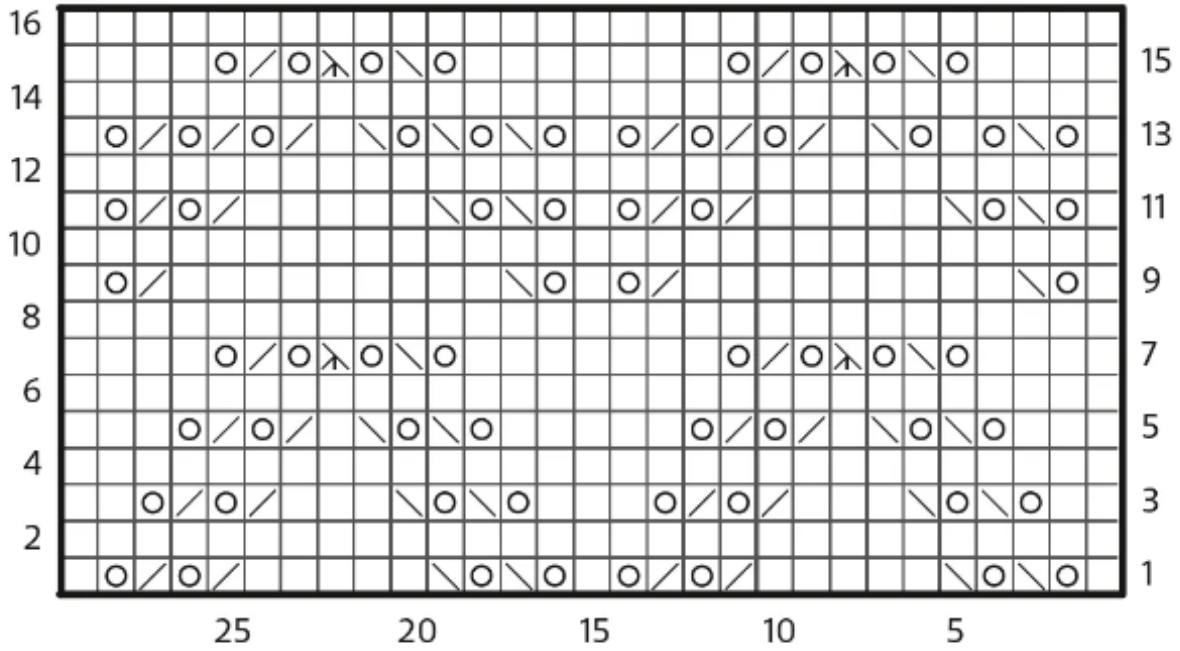
Mimic closely the domain







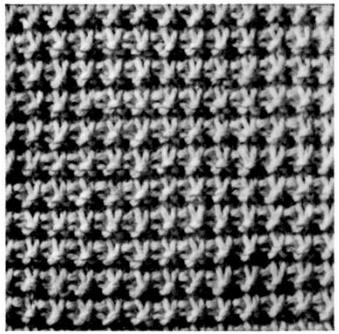
Debug, confirming errors



Row 13 requires 28 sts... yet row 12 produces 29!
Can you spot where the mistake is on row 13?

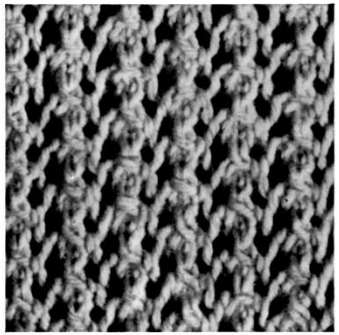
Pattern No. 1

Cast on 35 sts. **1st ROW:** K across row. **2nd ROW:** K 1, * yarn in back of work, sl 1 as if to P, K 1, repeat from * across row. **3rd ROW:** K 1, * yarn to front, sl 1 as if to P, K 1, repeat from * across row. **4th ROW:** K across row. Repeat these 4 rows for pattern.



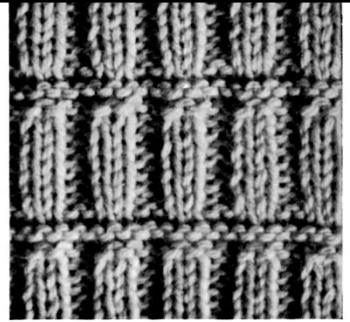
Pattern No. 2

Cast on 34 sts. **1st ROW:** K 2, * Y O, sl 1, K 2 tog, p.s.s.o., Y O, K 3, repeat from * across row ending with K last 2 sts. **2nd ROW:** P across row. **3rd ROW:** K 5, * Y O, sl 1, K 2 tog, p.s.s.o., Y O, K 3, repeat from * across row ending with Y O, K 2. **4th ROW:** P across row. Repeat these 4 rows for pattern.



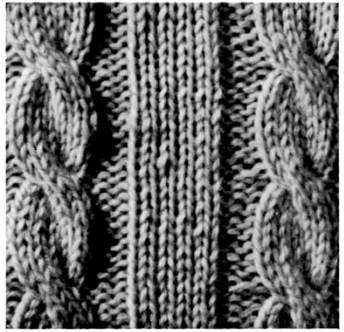
Pattern No. 3

Cast on 36 sts. **1st ROW:** * K 2, P 2, repeat from * across row. Repeat 1st row 7 times. **9th, 10th, 11th and 12th ROWS:** K across each row. Repeat these 12 rows for pattern.



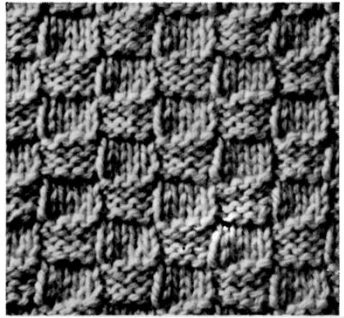
Pattern No. 4

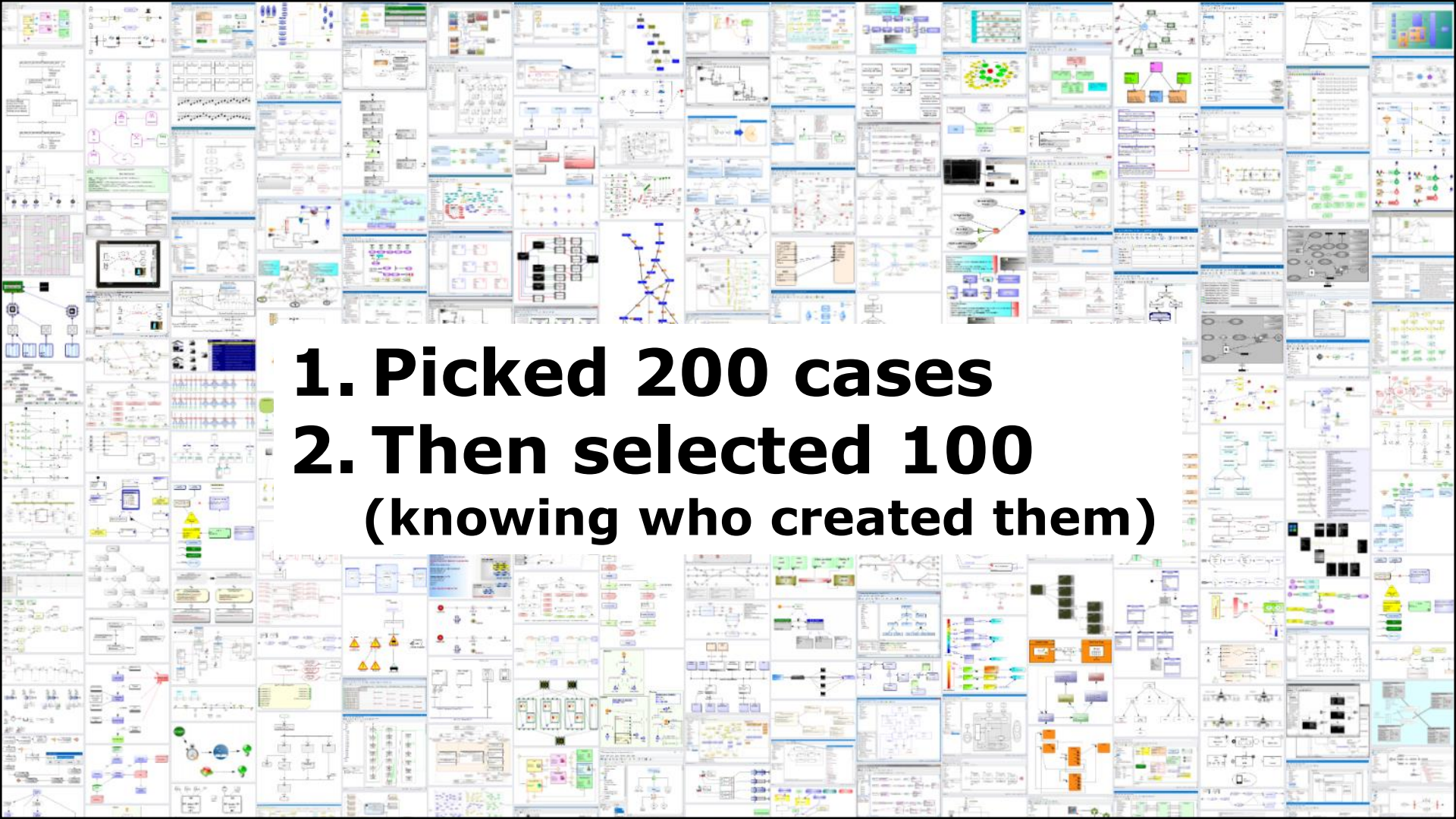
Cast on 35 sts. **1st ROW:** K 3, P 3, K 6, P 3, K 5, P 3, K 6, P 3, K 3. **2nd ROW:** P 3, K 3, P 6, K 3, P 5, K 3, P 6, K 3, P 3. Repeat last 2 rows 3 times. **9th ROW:** K 3, P 3, sl next 3 sts on d p n and hold in front of work, K next 3 sts, K 3 sts from d p n (cable twist), P 3, K 5, P 3, cable twist, P 3, K 3. **10th ROW:** Repeat 2nd row. **NEXT 8 ROWS:** Repeat 1st and 2nd rows 4 times. Repeat from 9th row for pattern.



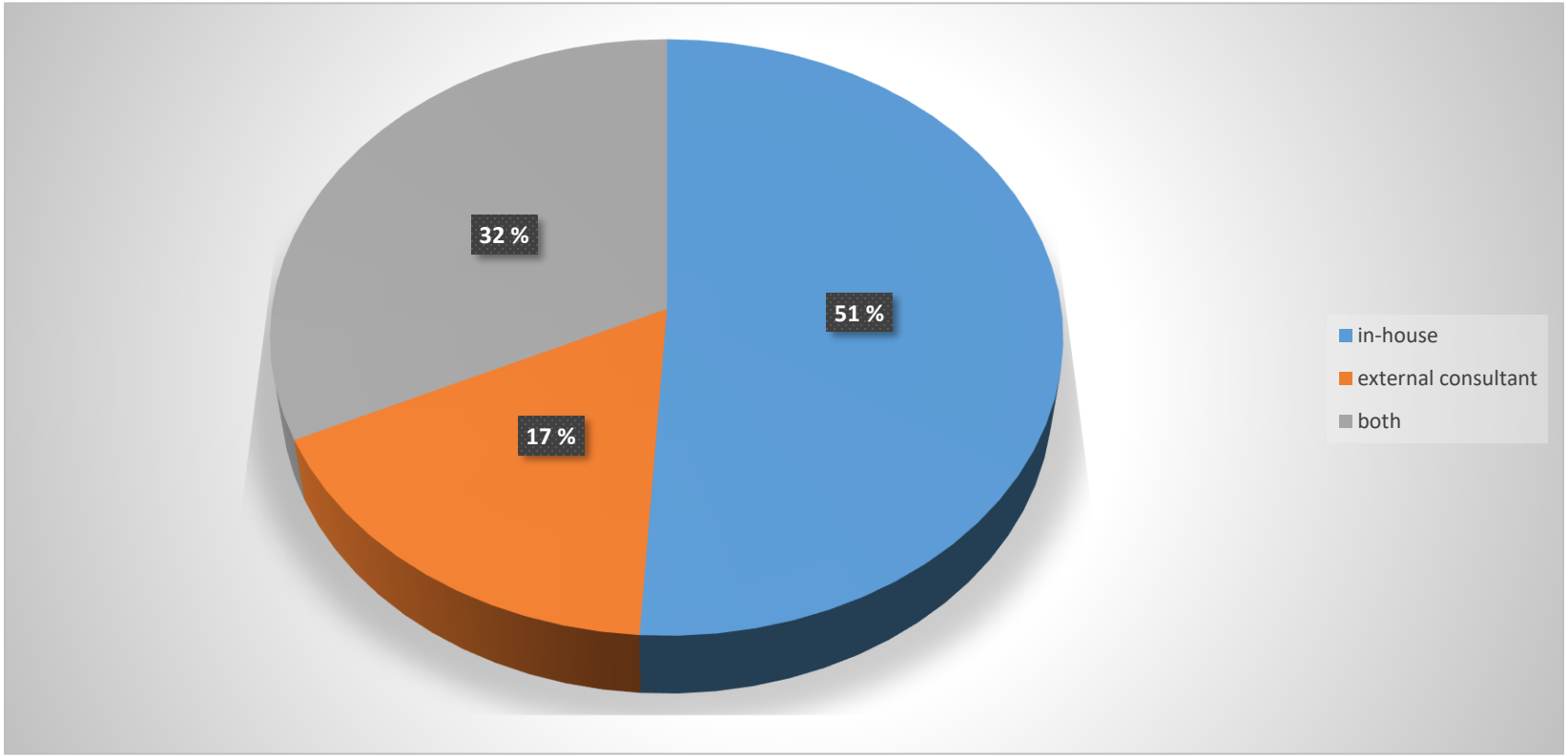
Pattern No. 5

Cast on 36 sts. **1st and 3rd ROWS:** K 4, * P 4, K 4, repeat from * 3 times. **2nd and 4th ROWS:** P 4, * K 4, P 4, repeat from * 3 times. **5th and 7th ROWS:** P 4, * K 4, P 4, repeat from * 3 times. **6th and 8th ROWS:** K 4, * P 4, K 4, repeat from * 3 times. Repeat these 8 rows 7 times.

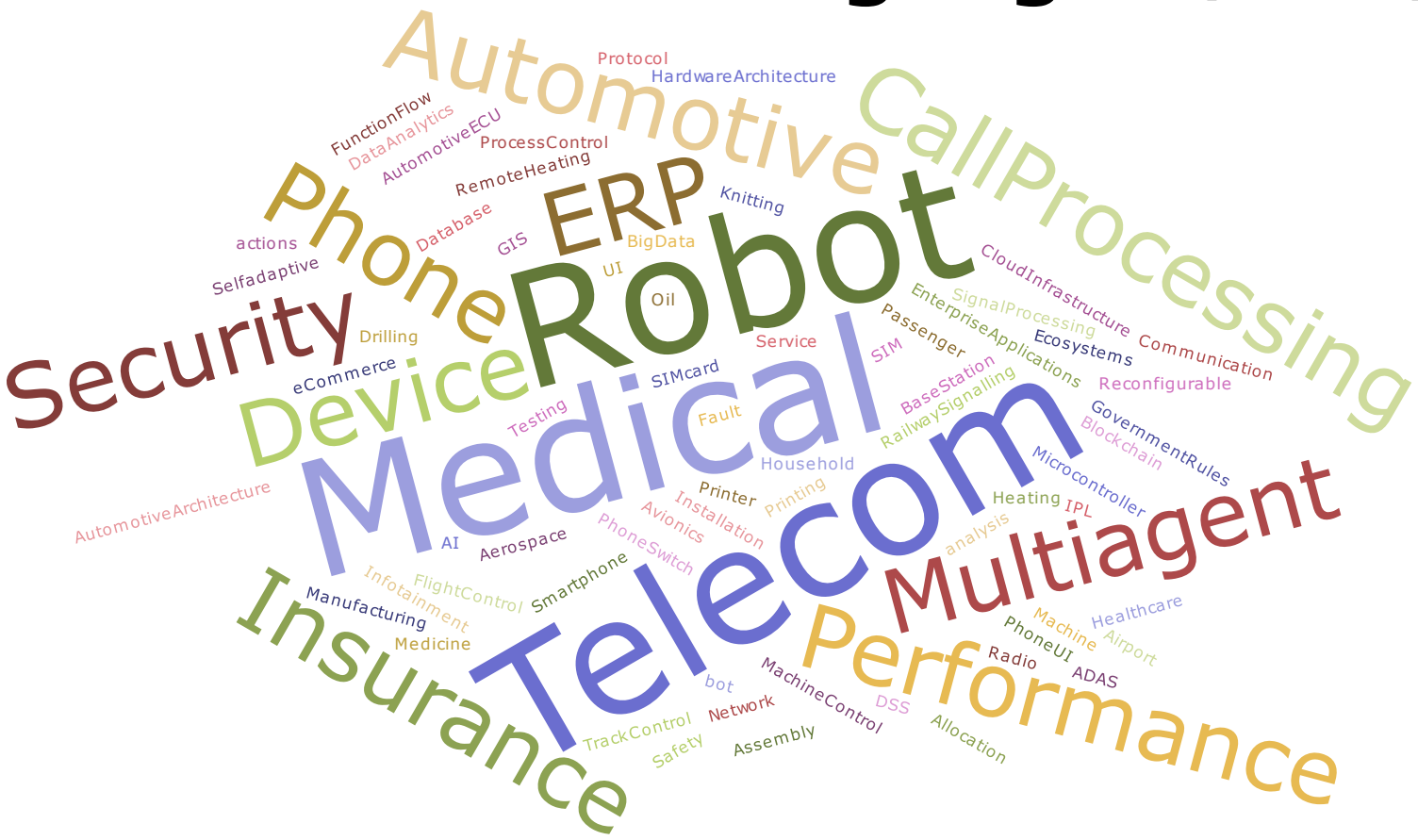



1. Picked 200 cases
2. Then selected 100
(knowing who created them)

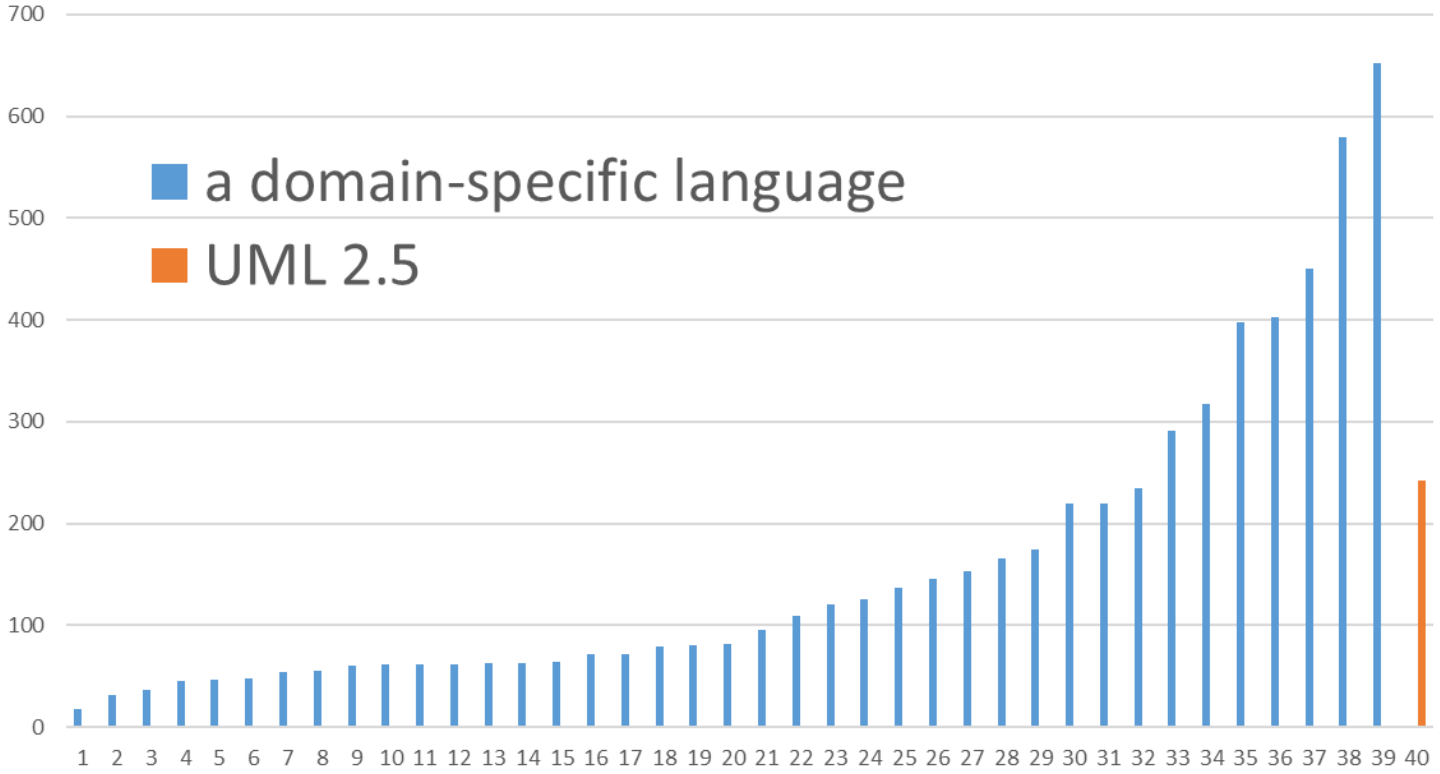
Who implemented languages? (n=100)



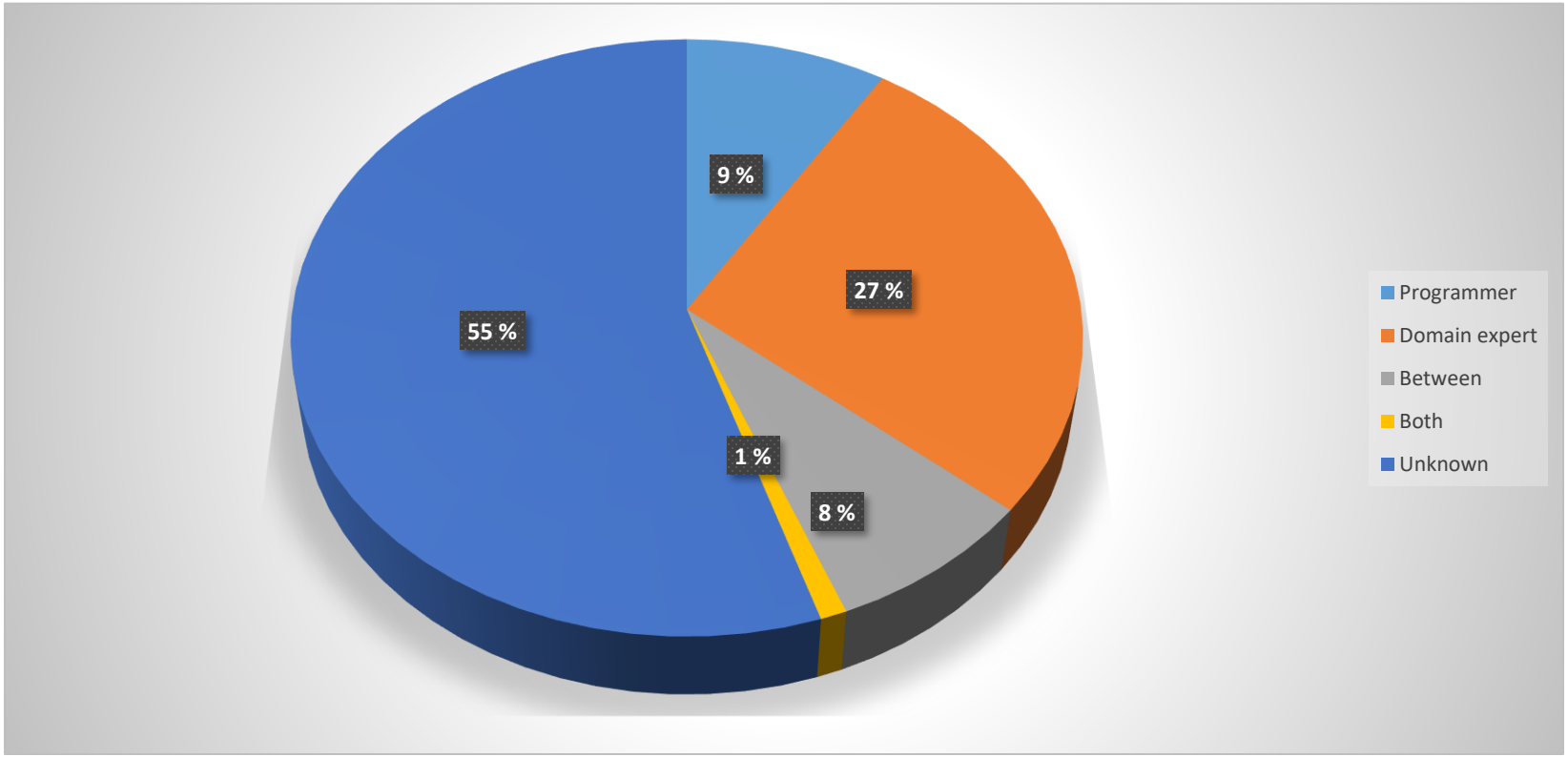
Domains of the languages (n=100)



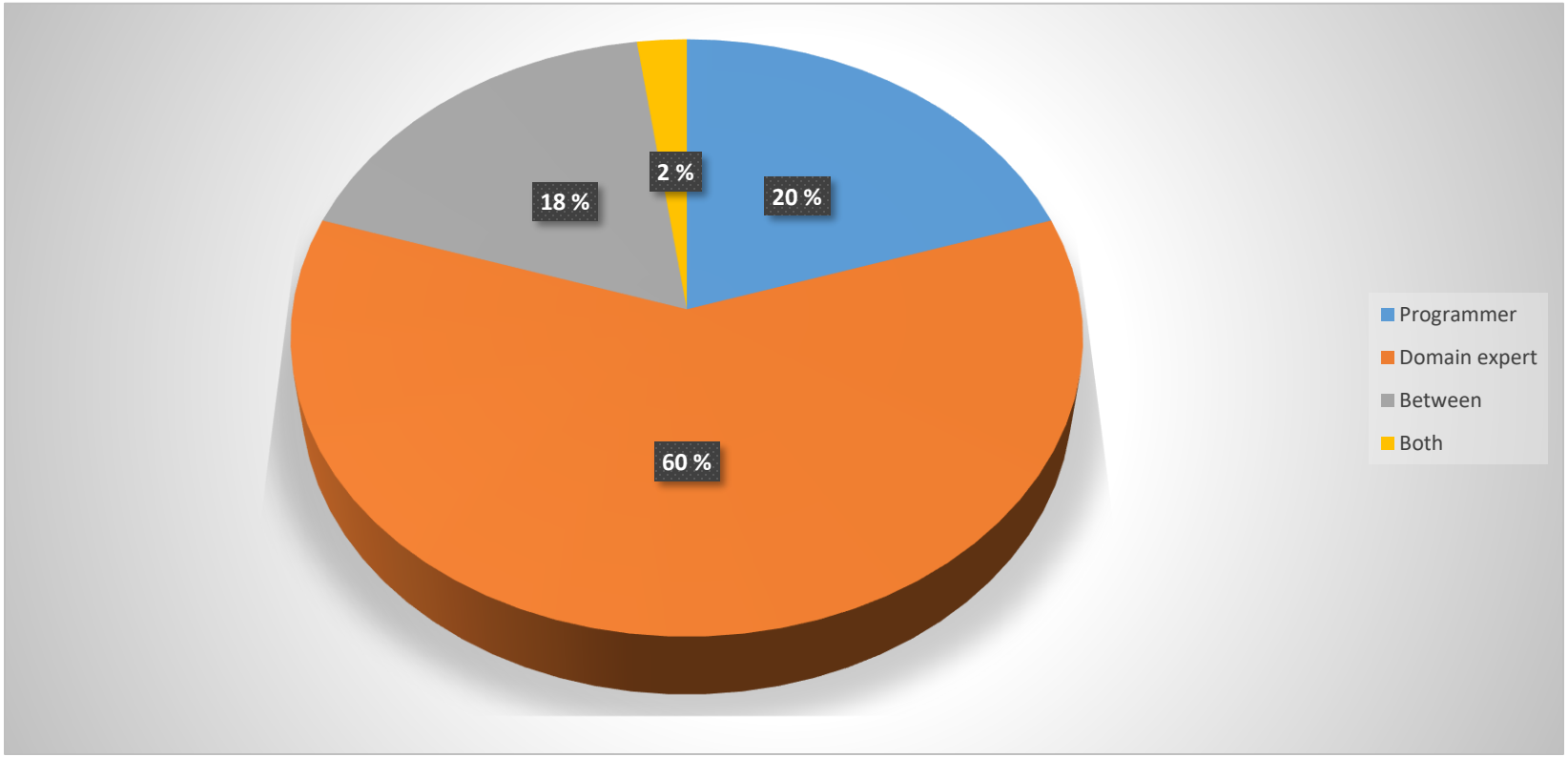
Size of the metamodels (n=39)



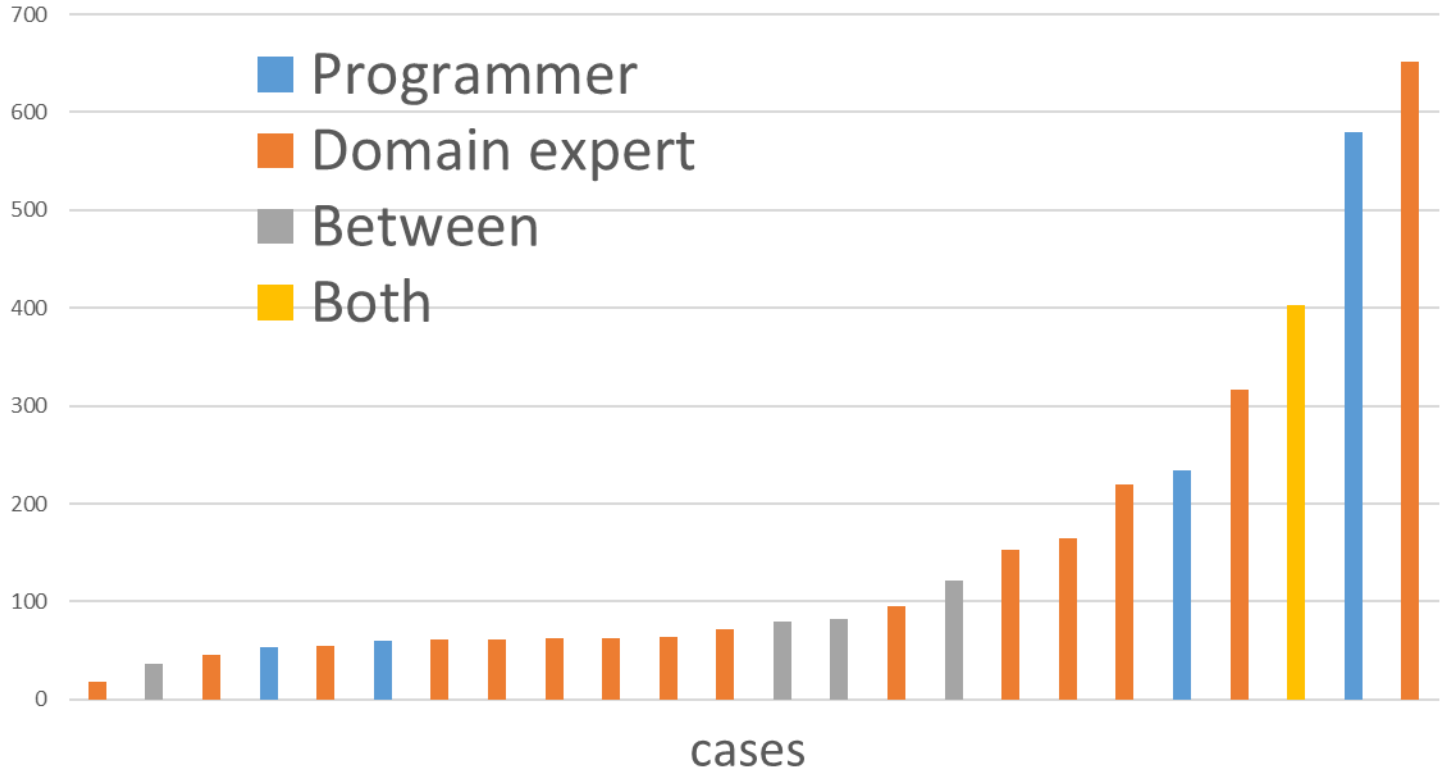
Primary language user (n=100)



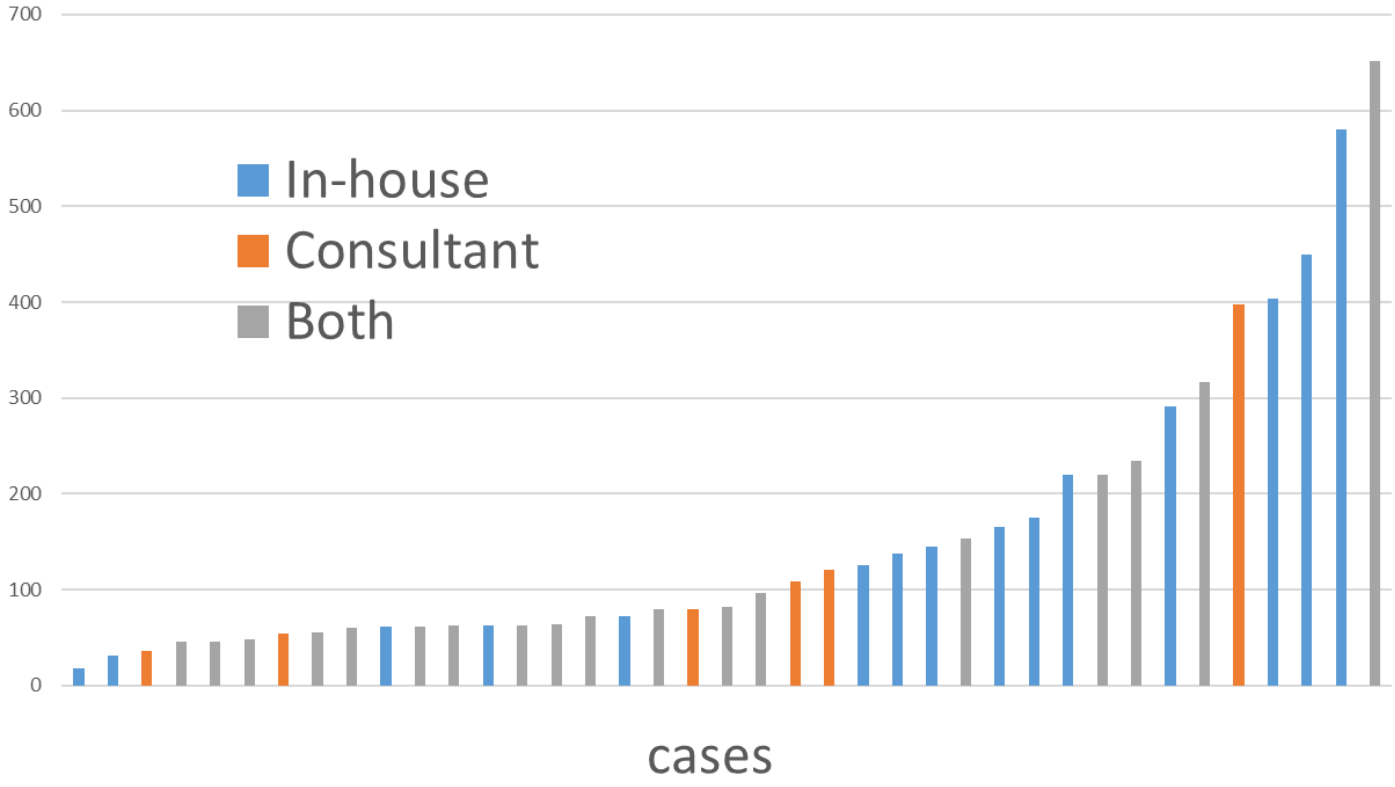
Primary language user (n=45)



Size and user (n=24)

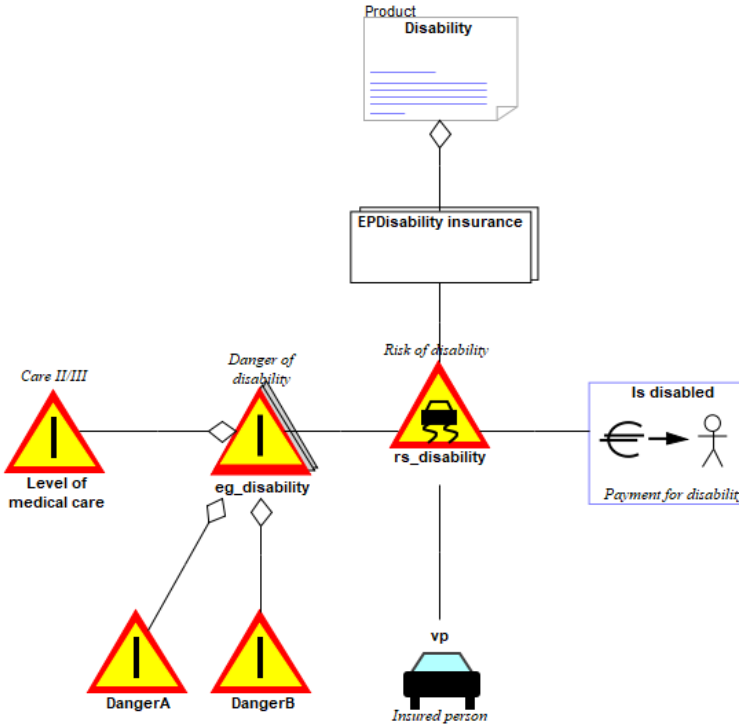


Size and language creator (n=39)

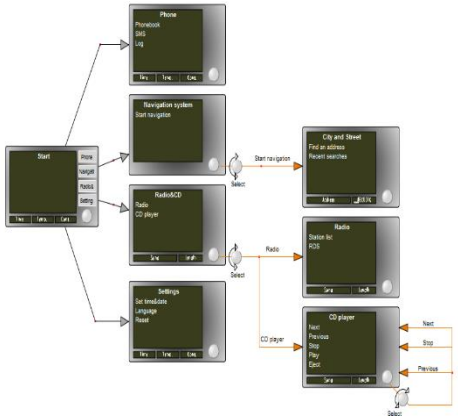


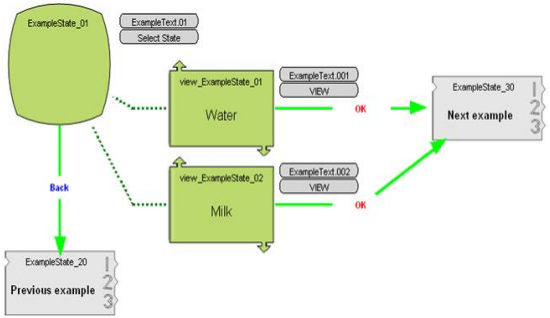
A language for insurance experts

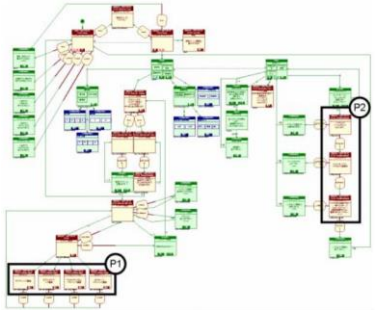
- Key language elements:
  - Product & product bundle
  - Calculation basis
  - Damage
  - Insured object
  - Danger
  - Product cover
  - Event
  - Payment
  - Policyholder
  - Risk
  - Insured person
  - Tariff



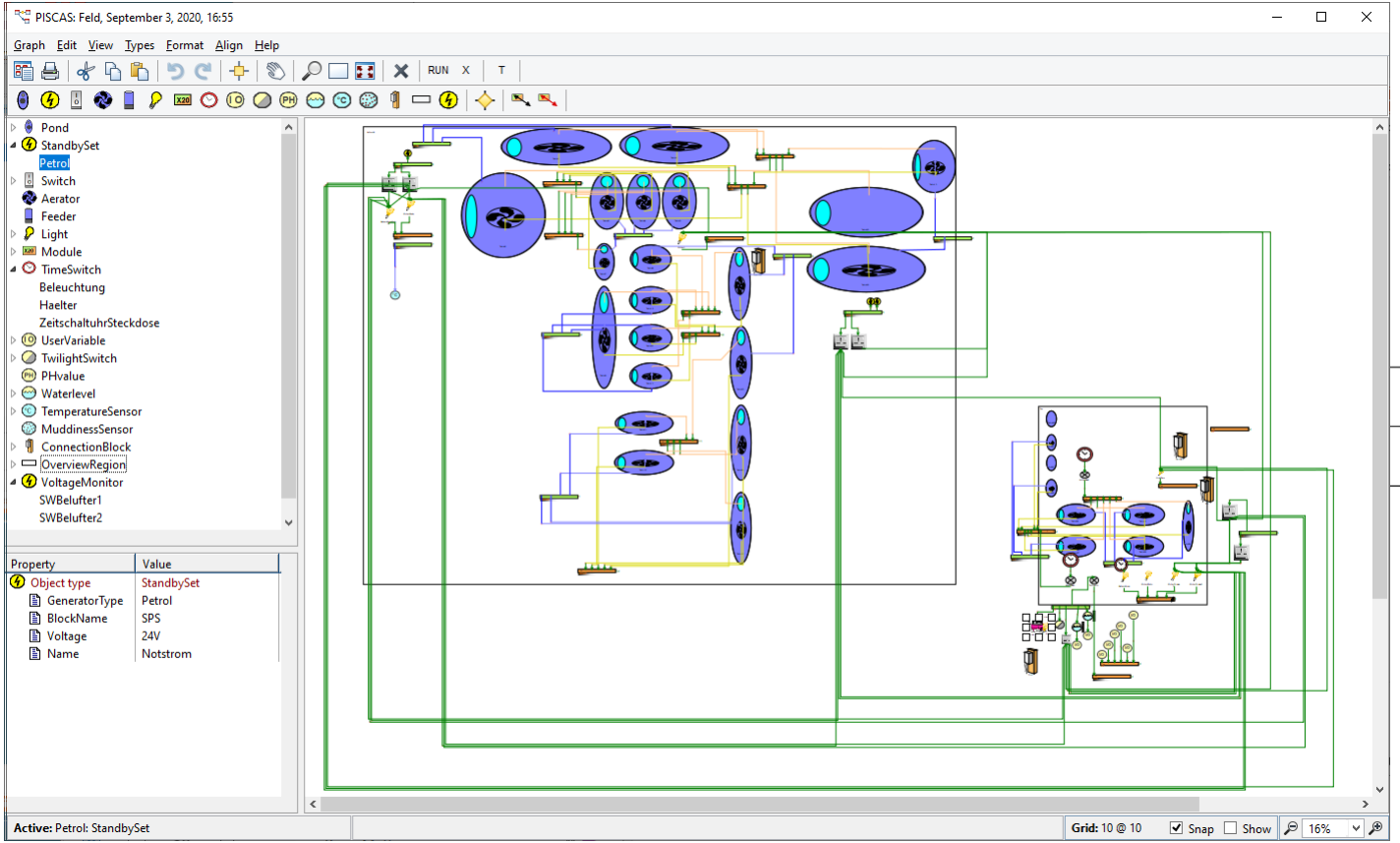
Languages for UX, usability experts





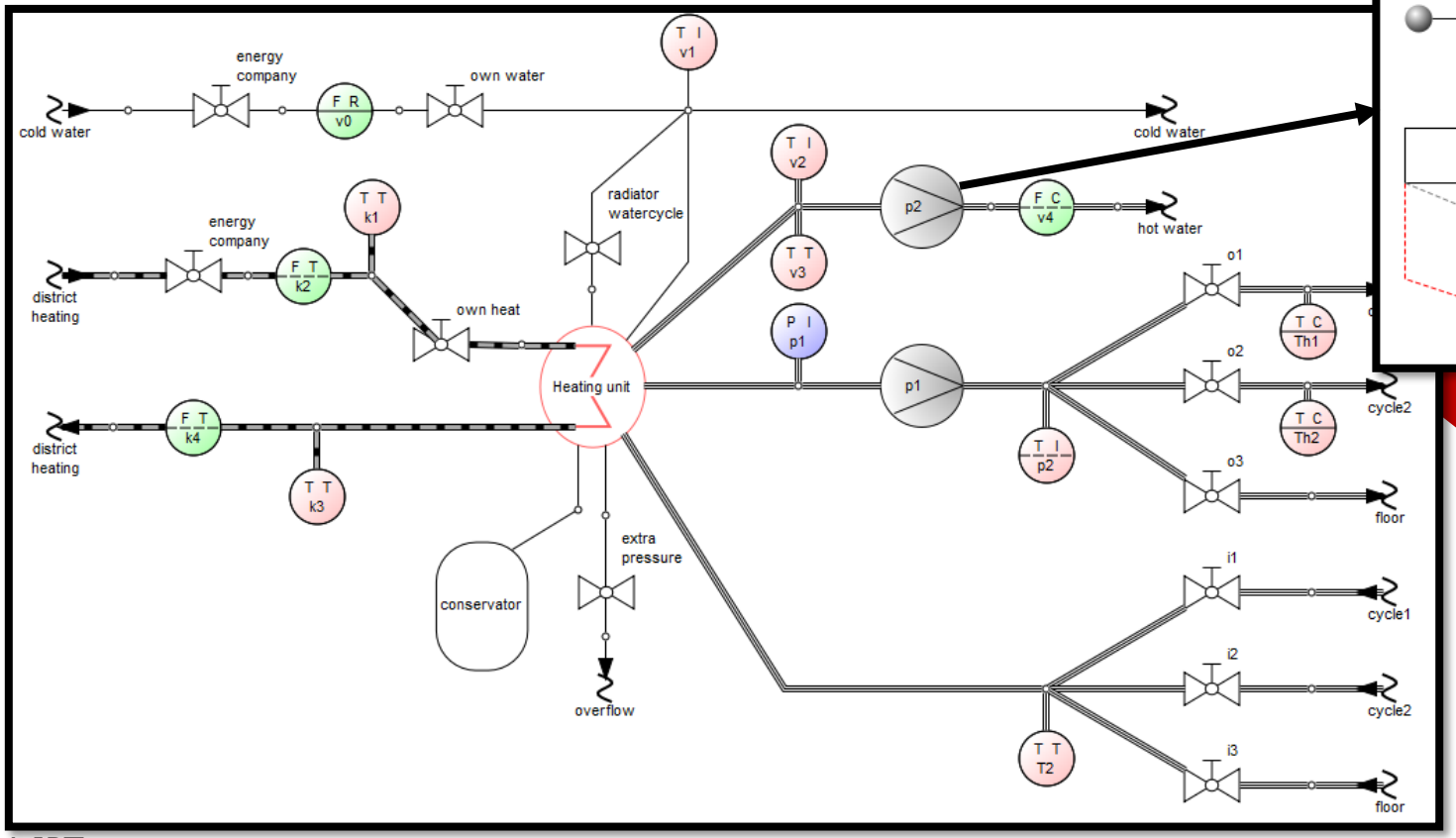


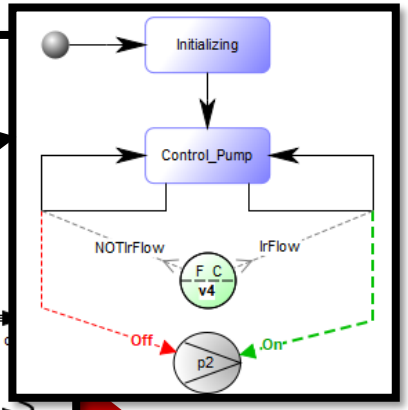
Example: fish farm automation



- Automation Software
  - Hardware mapping
  - IEC 61131 source code
  - Visualization
- Configuration Files
  - Web portal SQL files
  - Network device configuration
- Documentation
  - Graphical overview
  - Wiring plan
  - List of parts
  - Labels for wiring closet

Example: heating system



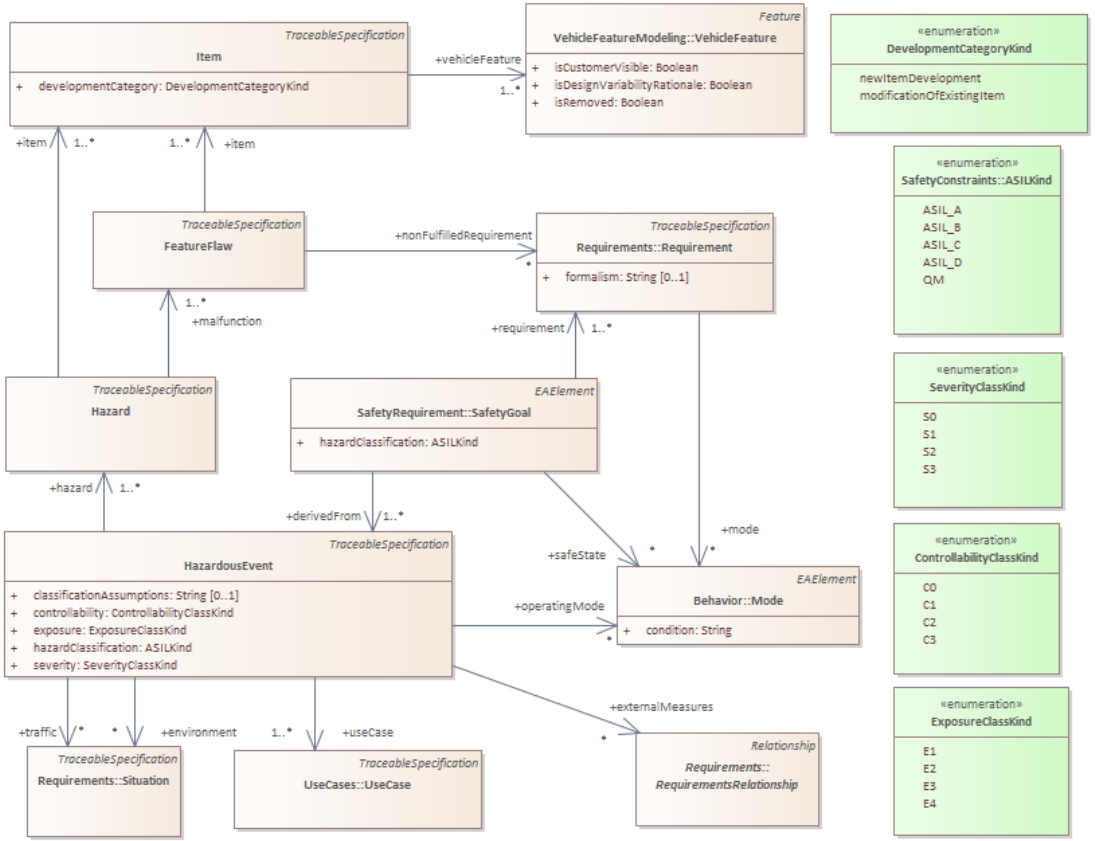


Code
Autobuild
API desc
BoM
Deployment guide

4 railway DSLs

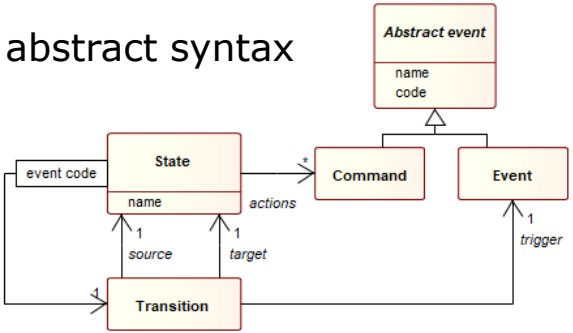
Languages for safety (e.g. ISO26262)

- Item
- Hazard
- Hazard event
- Safety goal
- Safety concept
- Feature flaw
- ASIL
  - Exposure
  - Severity
  - Controllability

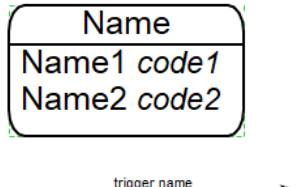


How languages are created?

abstract syntax



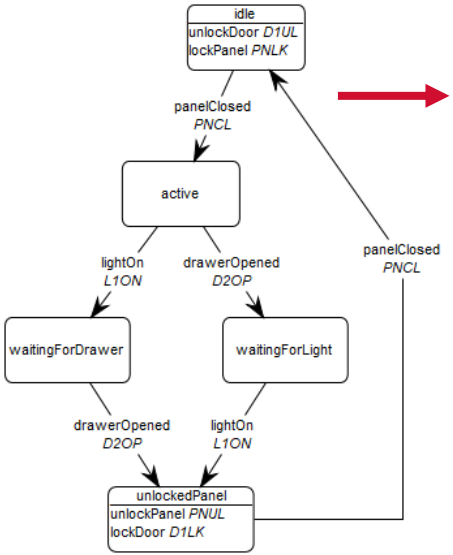
concrete syntax



semantics

```
foreach .State; {
   do :Command {
      id;l 'State.addAction(' :Name 'Cmd);'
   }
   do ~Source>Transition {
      id;l 'State.addTransition('
      do :Trigger {:Name} ', '
      do ~Target.() {id}
      'State);' newline
   }
```

Modeling



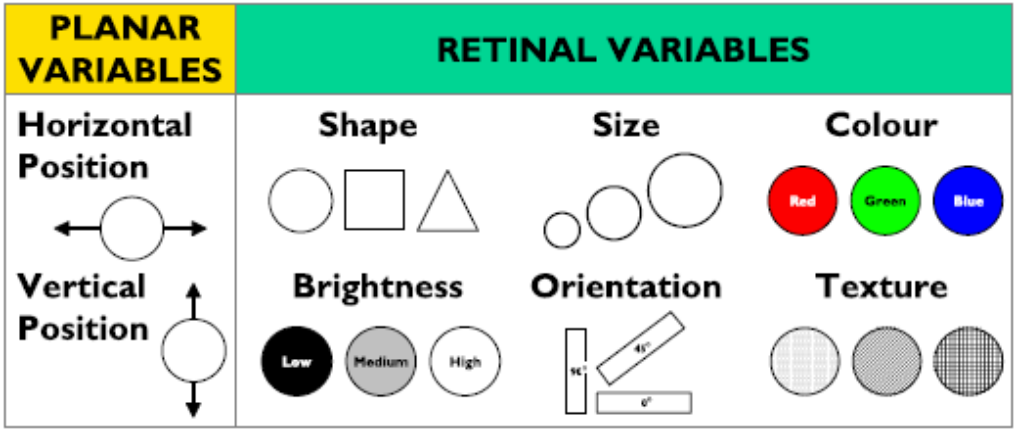

```
Event doorClosed = new Event("doorClose
Event drawerOpened = new Event("drawe
Event lightOn = new Event("lightOn", "L1
Event panelClosed = new Event("panelClo

Command unlockDoorCmd = new Comma
Command lockPanelCmd = new Comma
Command unlockPanelCmd = new Comm
Command lockDoorCmd = new Command

State activeState = new State("active");
State idleState = new State("idle");
State unlockedPanelState = new State("u
State waitingForDrawerState = new State
State waitingForLightState = new State("

activeState.addTransition(drawerOpened,
activeState.addTransition(lightOn, waiting

idleState.addAction(unlockDoorCmd);
idleState.addAction(lockPanelCmd);
idleState.addTransition(doorClosed, active

unlockedPanelState.addAction(unlockPa
unlockedPanelState.addAction(lockDoorC
unlockedPanelState.addTransition(panelC
```

What is needed beyond metamodel

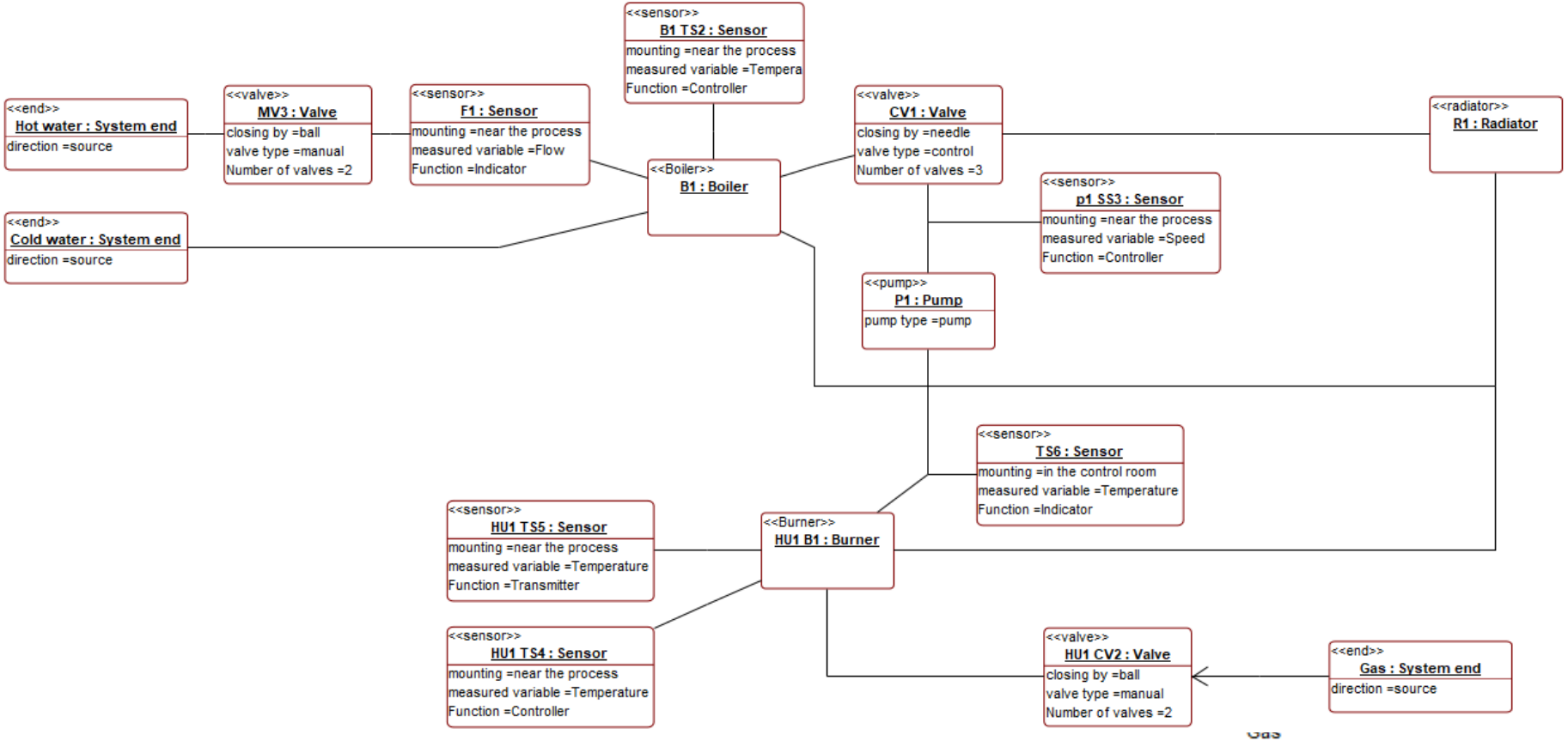
1. Concrete syntax matters
2. Involvement of language users = active participation
3. (Automated) support for language use
   - Errors
   - Warnings
   - Guidance
   - Simulation
   - Animation
4. Expect evolution and co-evolution with work done

1. Concrete syntax matters

- Mimic the problem domain
- Accepted by users
- Symbols should use full range of visual variables*



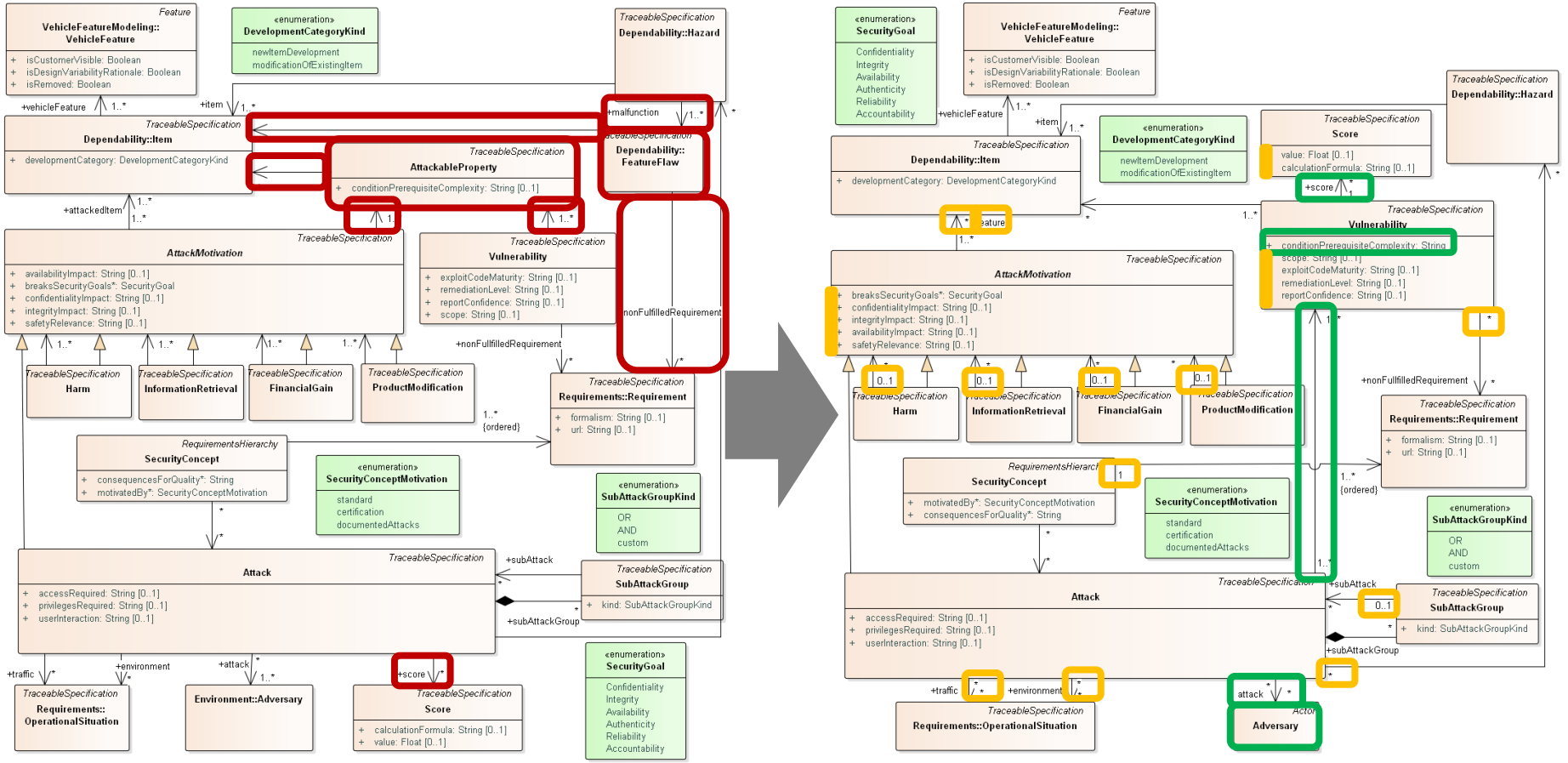
* D. Moody, The "Physics" of Notations, IEEE Transactions on Software Engineering, vol. 35, no. 6, 2009


<<sensor>>
B1 TS2 : Sensor
mounting =near the process
measured variable =Tempera
Function =Controller
<<end>>
Hot water : System end
direction =source
<<valve>>
MV3 : Valve
closing by =ball
valve type =manual
Number of valves =2
<<sensor>>
F1 : Sensor
mounting =near the process
measured variable =Flow
Function =Indicator
<<valve>>
CV1 : Valve
closing by =needle
valve type =control
Number of valves =3
<<radiator>>
R1 : Radiator
<<Boiler>>
B1 : Boiler
<<sensor>>
p1 SS3 : Sensor
mounting =near the process
measured variable =Speed
Function =Controller
<<end>>
Cold water : System end
direction =source
<<pump>>
P1 : Pump
pump type =pump
<<sensor>>
TS6 : Sensor
mounting =in the control room
measured variable =Temperature
Function =Indicator
<<sensor>>
HU1 TS5 : Sensor
mounting =near the process
measured variable =Temperature
Function =Transmitter
<<Burner>>
HU1 B1 : Burner
<<sensor>>
HU1 TS4 : Sensor
mounting =near the process
measured variable =Temperature
Function =Controller
<<valve>>
HU1 CV2 : Valve
closing by =ball
valve type =manual
Number of valves =2
<<end>>
Gas : System end
direction =source

2. Enable participation

- Try early
  - Examples of typical apps/features/systems, not metamodel
  - Prototype, ready to throw away
  - Narrow to minimum what is needed

Case: manually tested to released

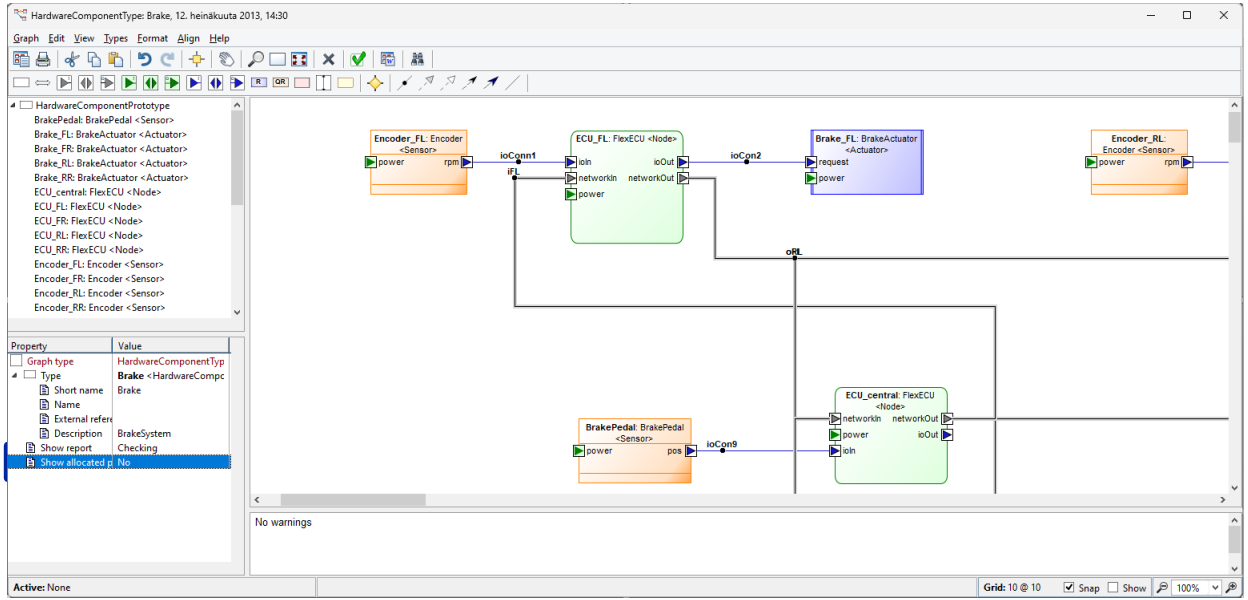


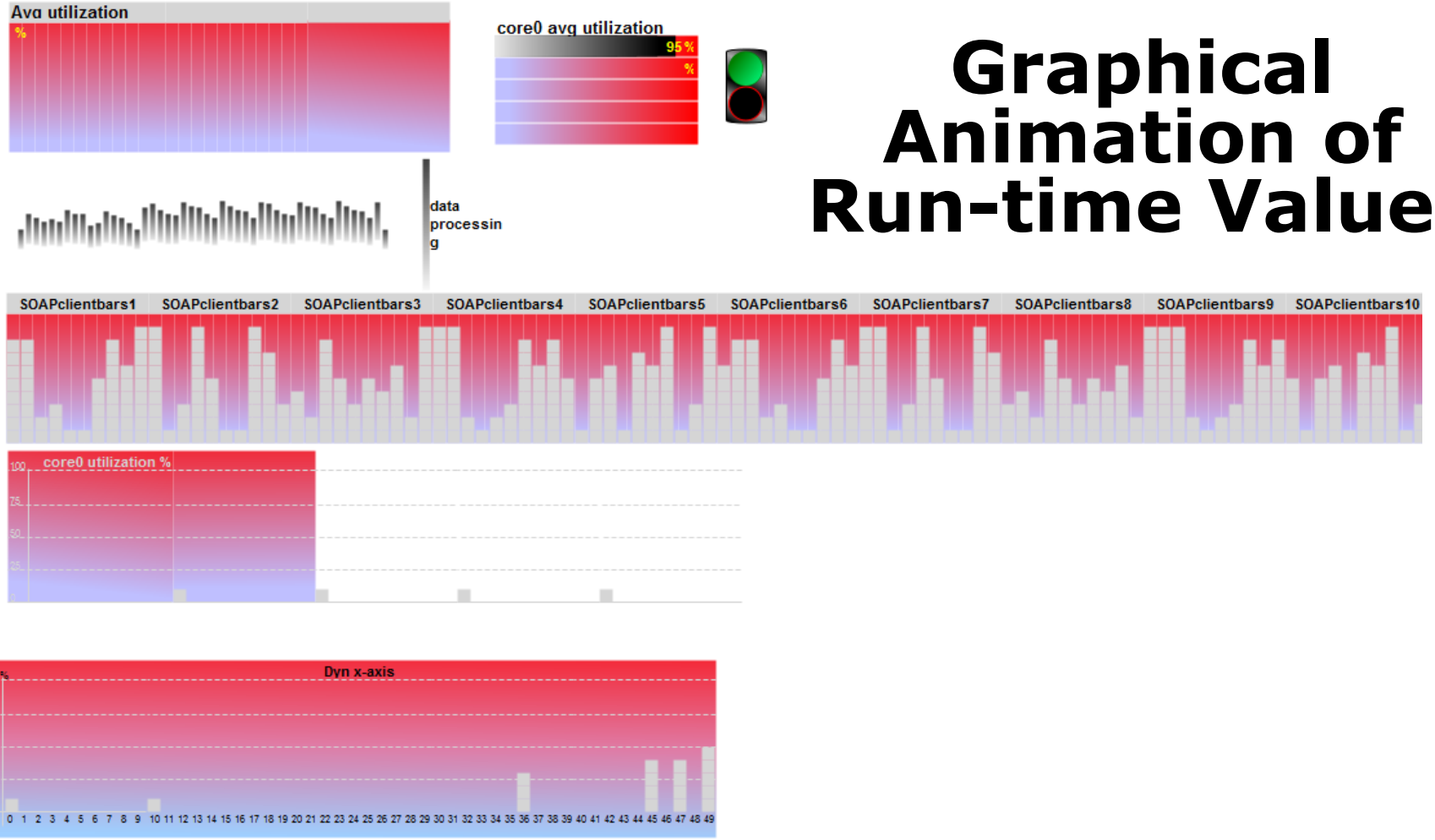
2. Enable participation

- Try early
  - Examples of typical apps/features/systems, not metamodel
  - Prototype, ready to throw away
  - Narrow to minimum what is needed
- Collaborative work: create & use DSL at the same time
  - Ask to define notation
  - Give (read-only or partial) access to the language definition
- Collect feedback
  - Get feedback when language is used
  - Via tool or even via the language itself

3. Support for language users

- Not only basics of language, but also covering
  - Errors
  - Warnings
  - Guidance
  - Views
  - Animation
  - Simulation
- Examples
  - Tutorials
  - Typical cases





Graphical Animation of Run-time Values

ID	Problem domain	Note	Years	Phase	Users	Icon	Color	Text	LiveCheck	Report	Generation
1	Home automation	*	0.1	1	1						
2	Database applications		0.2	2	2			G			
3	Data architecture		0.2	2	6					E	
4	Insurance products	*	2	4	4					W	
5	Insurance systems	* 3	4	4	40					R	R
6	Enterprise applications	3	5	4	12						E W
7	Big data applications		1	2	4				E W	E	
8	Phone UI applications	* 2	8	4	400					R	E R
9	Government EA		4	3	16		R			E W	
10	AI bot		0.3	2	2	W	A	W R			
11	Call processing	*	19	2	6				E W	E W	
12	Medical		2	2	2	R		R	E	E	E
13	Security		1	3	6		R	R	E G	E	
14	Industrial automation		4	3	2			G		E W	E W
15	Consumer electronics		6	2	1				E W	E W	E W
16	Blockchain ecosystems		0.25	2	34				E W G	E W G	
17	Software testing		3	4	55			E	E W	E W G	
18	Telecom		3	2	2	W	W	E W		E W	R
19	Performance testing		3	2	2	R A	E G	E G			E
20	Aerospace		4	2	2				E	E W R	E W R
21	Consumer electronics		12	4	24		E G	E W G		E W R	
22	Automotive ECU		5	4	2	E	E	E	G	E W	E W R
23	Automotive architecture	1	11	3	5	G	E W	E W G	W G	E W R	E W

*Kelly, Tolvanen. Automated Annotations in Domain-Specific Models: Analysis of 23 Cases. STAF Workshops, 2021

4. Evolution and Co-evolution

- Domain evolves
- Users learn
- External requirements must be met

- Language evolves
- Existing work must evolve too

- Ideally, updates automatically
  - Manual work and transformations are often not practical

Evaluation framework: 4 aspects*

2 *Location of Change* ↓	1 *Nature of Change*			
	Add	**Rename**	**Remove**	**Change**
Metamodel	1	4	7	10
Constraints	2	5	8	11
Notation	3	6	9	12

3 Location adversely impacted

- Metamodel, Constraints, Notation
- Generators, Tool, Models

4 Scale for scoring co-evolution:

1. When creating a new artifact, editor **does not open or gives errors**
2. Editor opens **without functionality**
3. Editor allows creating a new artifact but **support for viewing and editing earlier artifacts is incomplete**
4. Editor **opens and asks for human intervention** to finalize co-evolution

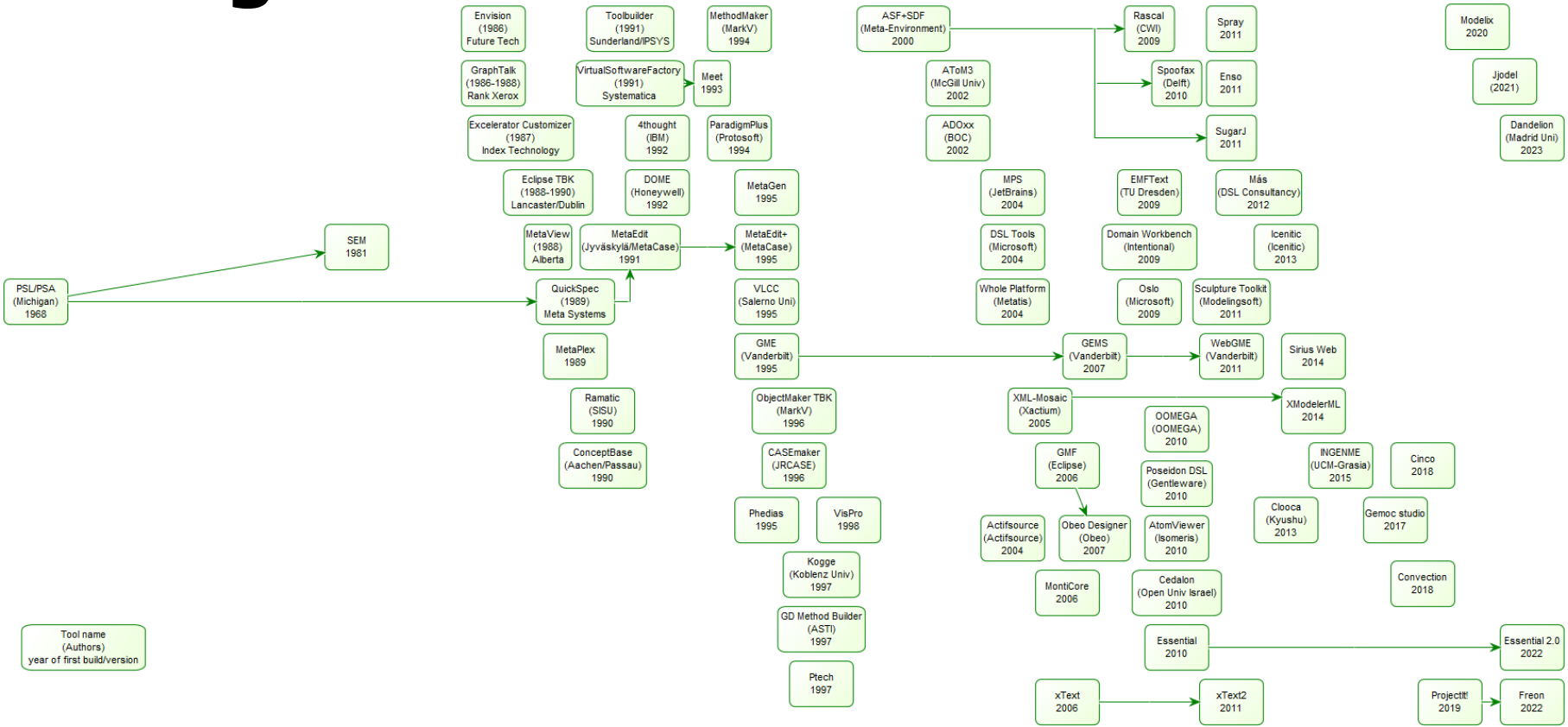
   4½ if existing models behave and generate, and deprecation guidance is provided where needed
5. Editor **opens** with **fully co-evolved** earlier artifacts

* Tolvanen and Kelly. Evaluating Tool Support for Co-Evolution of Modeling Languages, Tools and Models. ACM/IEEE MODELS Conference companion, 2023

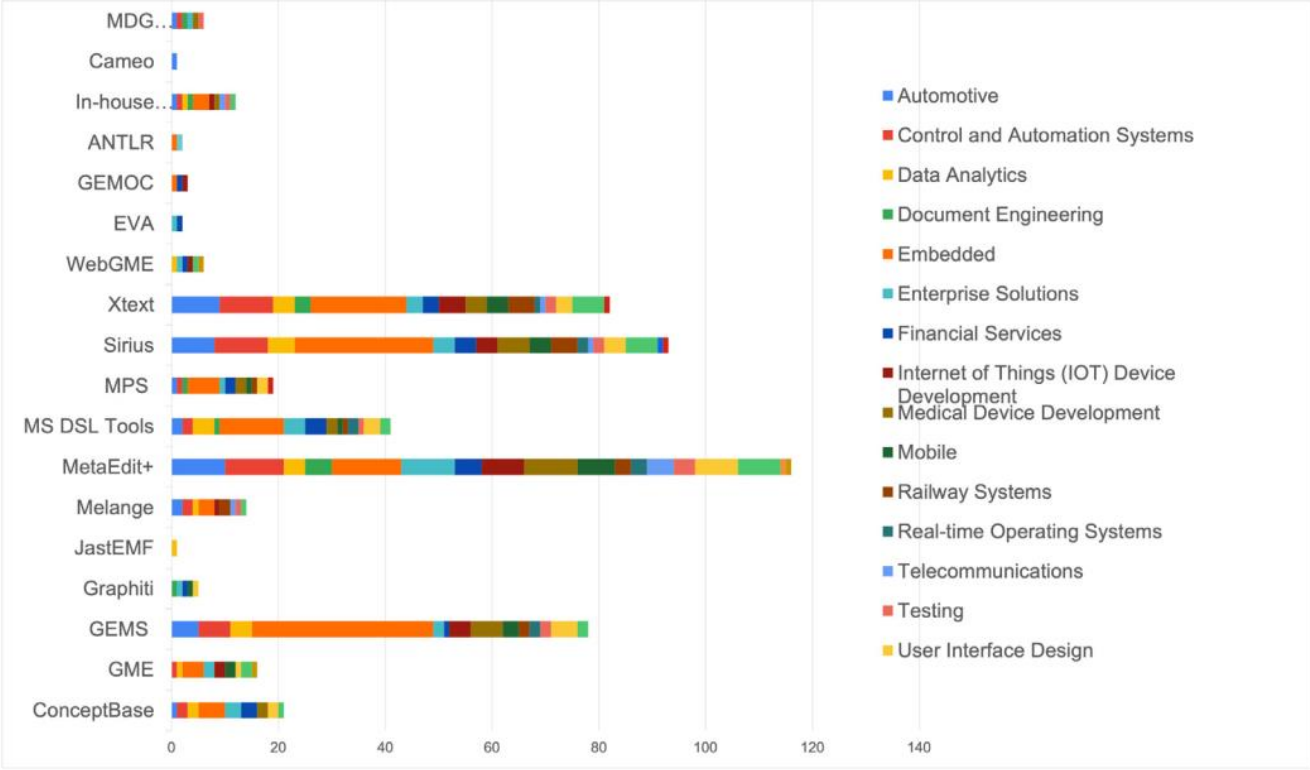
What about tools?

- 6 ways to get the tools we need for our language
  1. Write own modeling tool from scratch
  2. Write own modeling tool based on frameworks
  3. Metamodel, generate modeling tool skeleton, add code
  4. Metamodel, generate full modeling tool over a framework
  5. Metamodel, output configuration for generic modeling tool
  6. Integrated modeling and metamodeling environment
- Single-user, collaborative
- Versioning (not as traditional VCS), a domain-specific
- Easy to access and learn, supported, training etc.

Tooling



Tools in different domains*



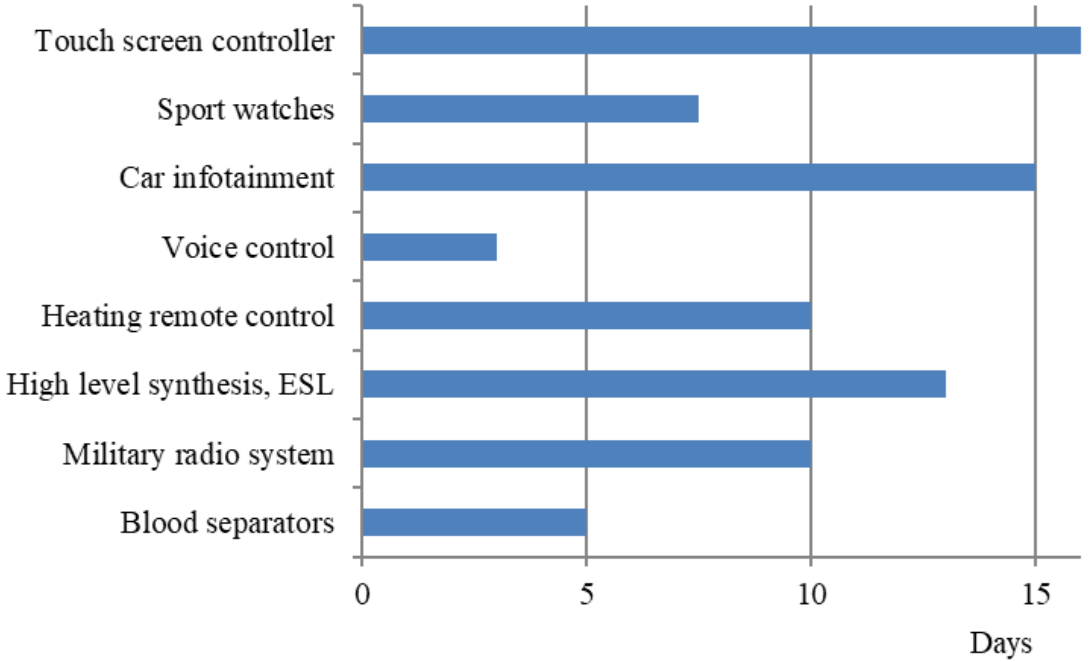
* Ozkaya, M., Akdur, D., What do practitioners expect from the meta-modeling tools? A survey, Journal of Computer Languages, Vo 63, 2021

Where to apply? Where not?

- Timing is crucial
  - At certain times organizations are ready for the change
- Repetition
  - Product line, configurable product, many similar features
- Domain knowledge is substantial
  - Business/domain rules have a big role, domain experts/subject matter experts needed

- Not if:
  - No repetition, new domain, unstable domain, multiple organizations involved, no resources to create languages

Cost of language creation: industry cases*



* Tolvanen and Kelly. Effort Used to Create Domain-Specific Modeling Languages. ACM/IEEE Conference on Model Driven Engineering Languages and Systems, 2018

Concluding remarks

- Languages for non-developers allow wider range of people to participate in developing software systems
  - Define, check, validate, collaborate, test etc.

- Languages for domain-experts must:
  - Raise abstraction above code, close to the problem domain
  - Apply rich knowledge representations (maps, diagrams etc.)
  - Provide more than just spec creation features, like guidance, checks, animation, simulation…

- Modern tools assist in creating and using languages

Thank you

Questions?

Comments?

Counterarguments?

Experiences?

Contact: jpt@metacase.com

About me: Juha-Pekka Tolvanen

- Works for MetaCase
  - Provider of modeling and code generation tool MetaEdit+
- Acts as a consultant for creating modeling languages
  - 100+ DSL solutions
- Co-author of a book on Domain-Specific Modeling, IEEE-Wiley
- PhD in computer science, adjunct professor
- Enjoys sailing and skiing