



MODEL-BASED SOFTWARE DEVELOPMENT

LECTURE I.

INTRODUCTION

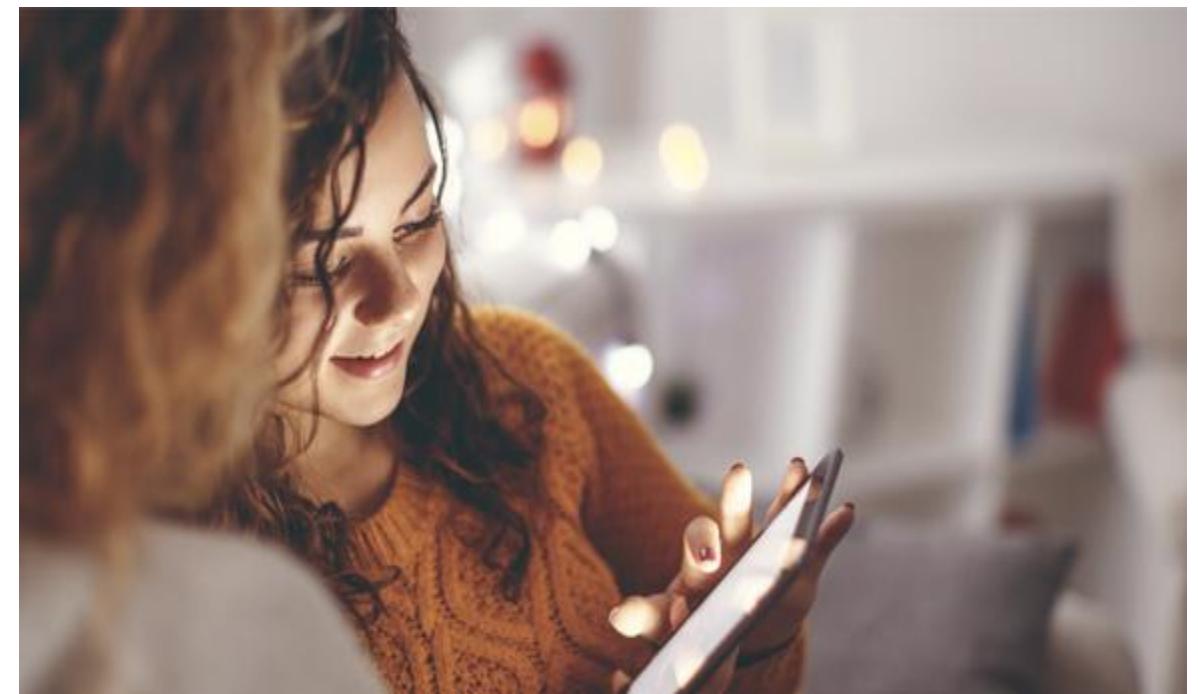
DR. GERGELY MEZEI

TODAY

Chapter I. Why?

Chapter II. About what?

Chapter III. How?

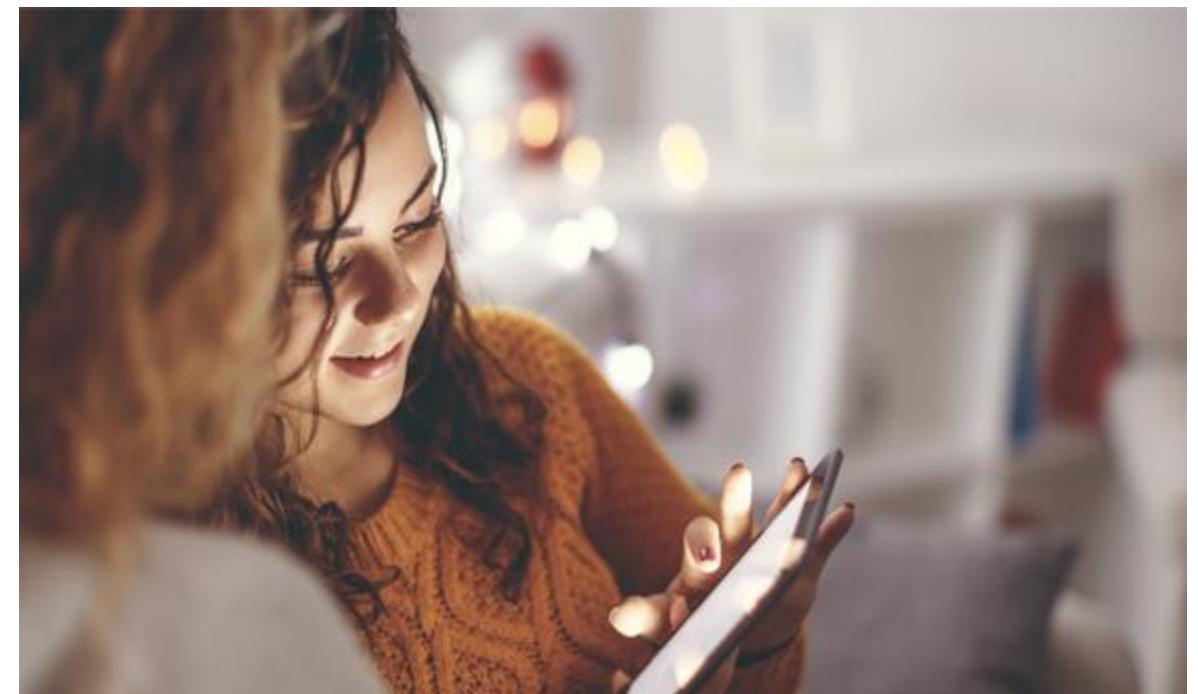


TODAY

Chapter I. Why?

Chapter II. About what?

Chapter III. How?



SOFTWARE DEVELOPMENT – THE BEGINNING

- Tom Kilburn - Manchester Small-Scale Experimental Machine - 1948
 - First program code stored in the machine
 - Program in the memory (later on punch card)
- Fortran - 1957
 - First high level programming language + compiler
- C Language - 1972
- Object oriented languages : Simula (1967), Smalltalk (1972)
- C++ - 1985
- Java - 1995
 - Java Virtual Machine - more portable code
- C# - 2002
 - Intermediate Language – language interoperability (VB, C#, Managed C++)
- Python, Swift, Go, Kotlin, ...

WHERE IS SOFTWARE DEVELOPMENT HEADED?

- What are the expectations?
 - Increasing application size
 - Decreasing development time
 - Less bugs, errors
 - Higher quality
- A solution... ?

CREATING SOFTWARE BECAME CHEAPER

- With AI: rapid prototyping, more code, more iteration
 - More variants → more integration and quality risks
- New challenges: specification + control + responsibility
- Demonstrable quality?

- What are the expectations?
 - Increasing application size
 - Decreasing development time
 - Less bugs, errors
 - Higher quality

AI = „Infinite junior developer”

- It's fast, but who is responsible for the mistakes?
- Strong
 - Boilerplate
 - Refactor
 - API usage
- Weak
 - Specific areas of expertise
 - Edge cases
 - Non-functional expectations
- Risk: "guessing" the missing parts
- Key: defined expectations + automatic checks



DEVELOPMENT:AI + HUMAN

- Human
 - Goals, limits
 - Architectural design
 - Acceptance criteria
- AI:
 - Implementation, alternatives
 - Documentation

Why is "I write down the task in the chat" not enough?

- Natural language is ambiguous (e.g., "fast," "safe," "handle well")
- Missing parts: exceptions, priorities, boundary conditions
- AI tends to "fill" gaps
- The specification is a contract — and it should be verifiable

TREND CHANGES IN SOFTWARE DEVELOPMENT

- Don't write something...
- ... that others have already written
 - There is a (semi) complete solution (library, component) for almost everything
 - Coding → installation and configuration management
- ... that can be written by others (with lower payment category)
 - AI, Code monkeys or rather: automation and code generation
 - Can the customer write it?

LOW CODE - NO CODE

- Motto: “Product development without programmers”
 - Graphic interface, "clickable" application logic
 - Rapid development
 - Limited area of usage
 - Business concepts, specific controls and processes

LOW CODE - NO CODE

Low code

- Minimal code + AI-assisted generation
- Fast training and development
- Usually graphic editor
- Partially limited expressive power
- For developers and for businessmen

No code

- Description / configuration instead of coding
- Almost zero training, immediate development
- Usually graphic editor
- Limited expressive power
- For businessmen

LOW CODE - NO CODE

- The secret of success
 - Let's talk about the problem at your language
 - Let's focus on the real task
 - Skip the repetitive parts
 - Be concise and transparent
 - Do not require programming skills (or only if its mandatory)

SOFTWARE DEVELOPMENT - TODAY

- Demand: quickly, well and a lot
- Solution:
 - Higher abstraction level
 - Assembly → C → C++ → Java/C# → ...
 - Configuration and integration instead of programming
 - *Use the solutions from others*
 - Generate everything
 - AI
 - *C++ template, generated constructor + destructor, property*
 - Ability of testing
 - *Not just line coverage, formal verification/validation*

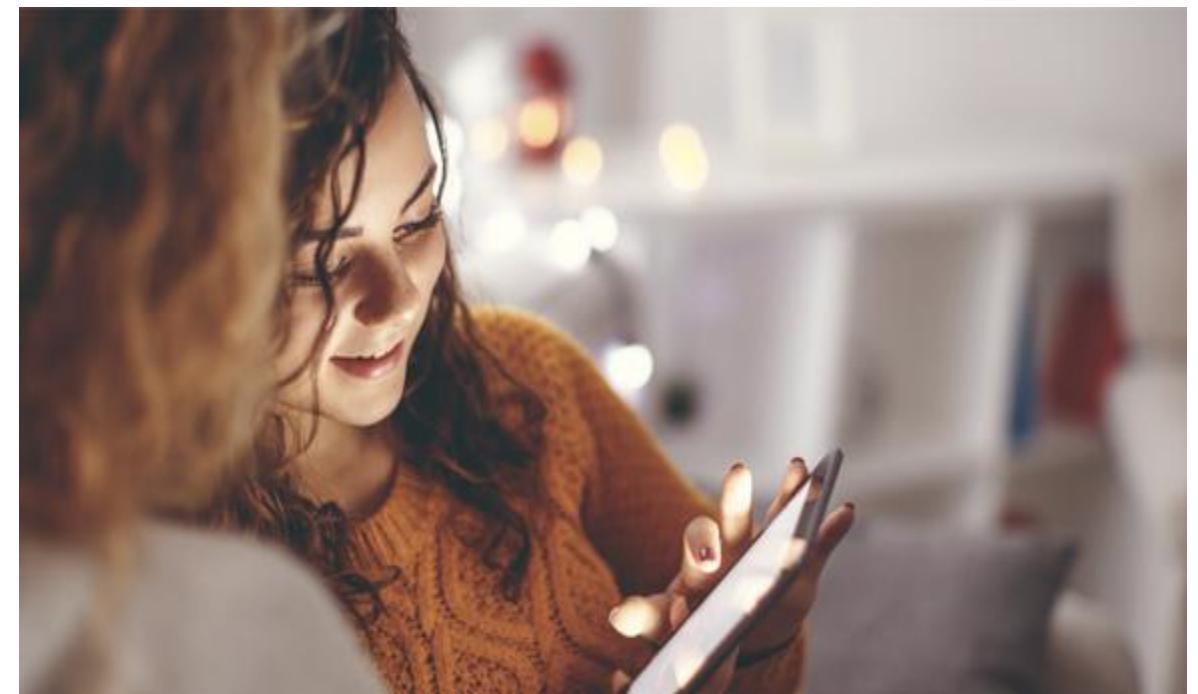
.... all of this is given by **Modeling**

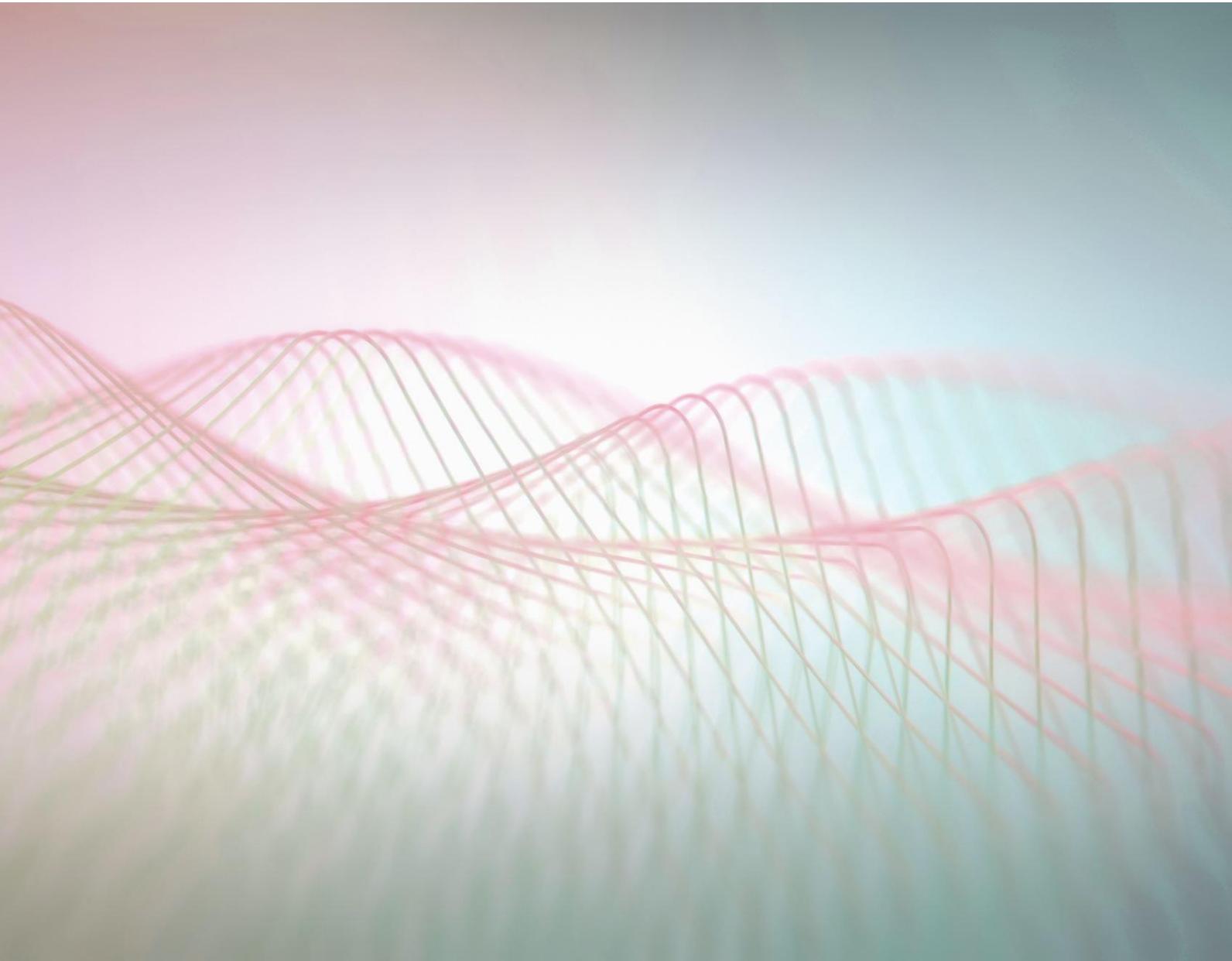
TODAY

Chapter I. Why?

Chapter II. About what?

Chapter III. How?

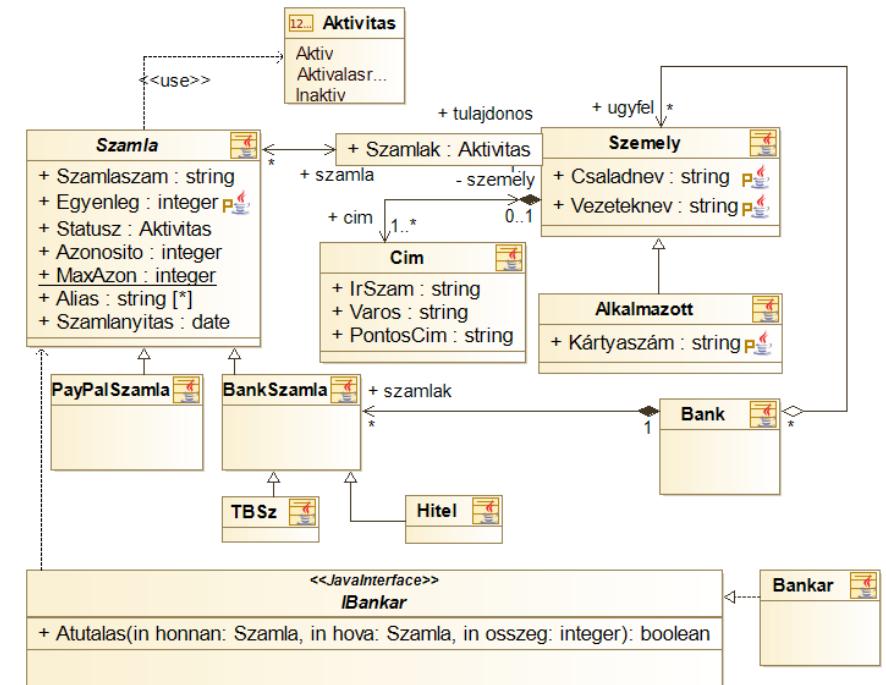




UNIVERSAL OR
CUSTOMIZED?

UML

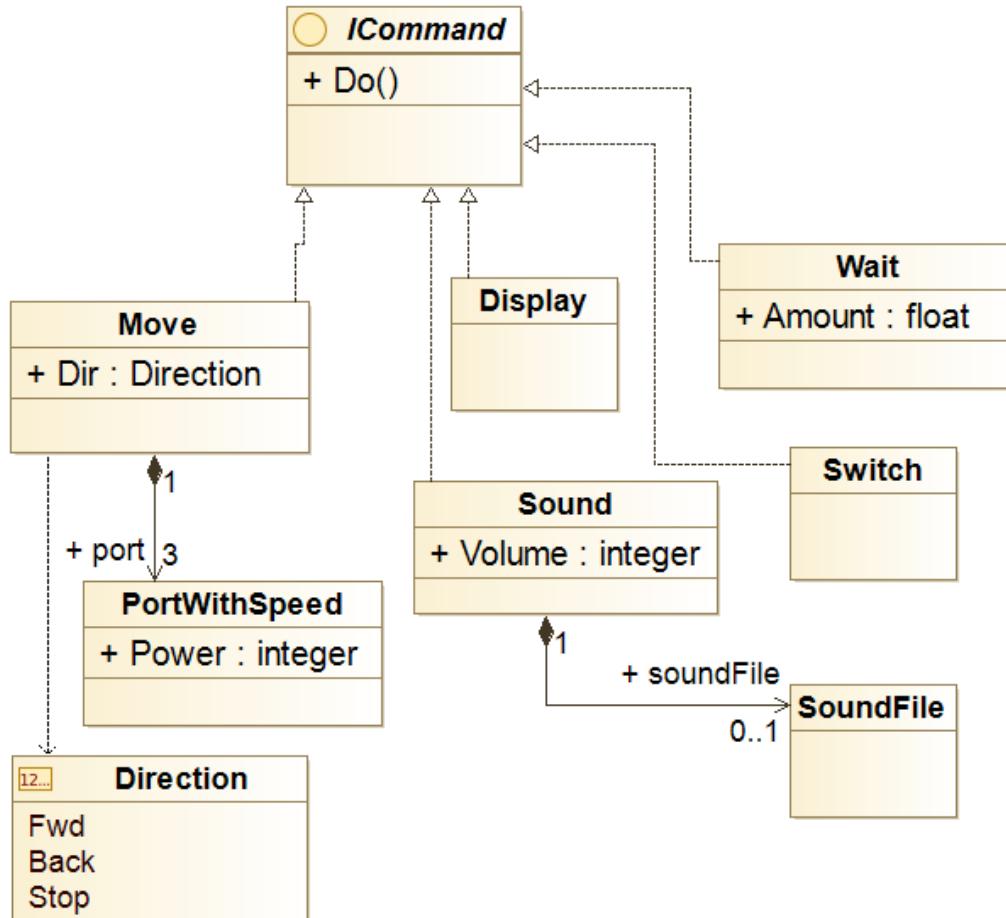
- Modeling - as you may know: UML
 - Software engineers common modeling language
 - High abstraction level
 - Standard notations
 - Supported by many tools
 - Limited customization
 - Partial code generation



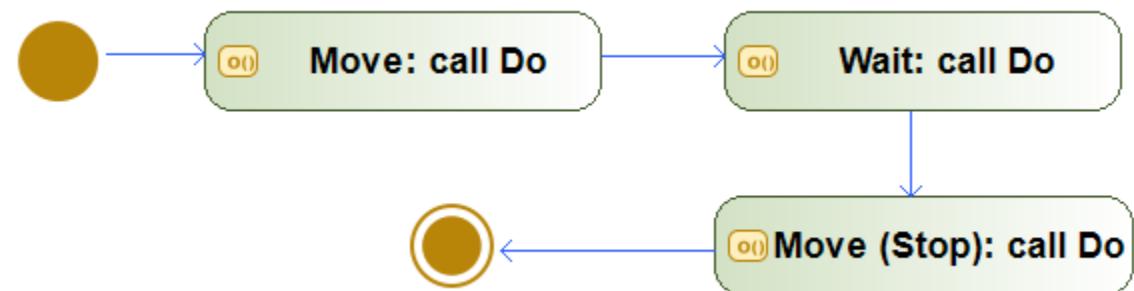
AN EXAMPLE : LEGO MINDSTORMS



MINDSTORMS - UML

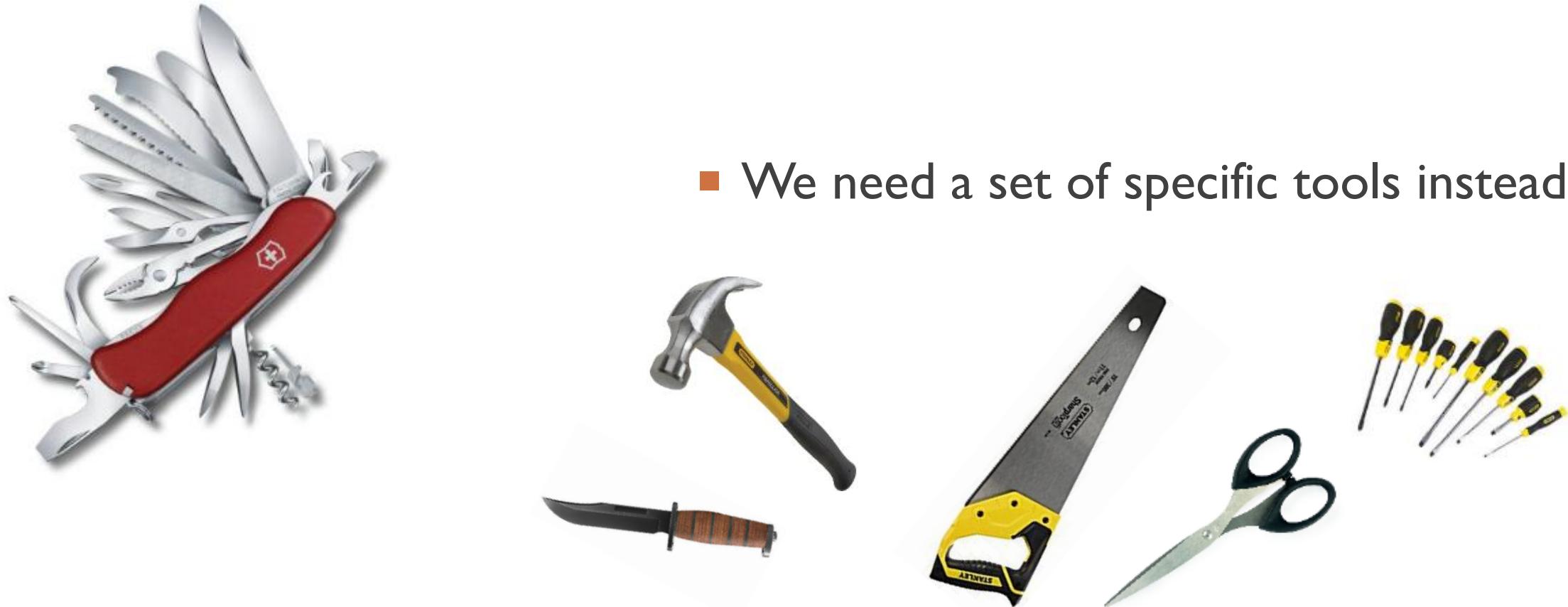


- Go forward for 1 second, then stop



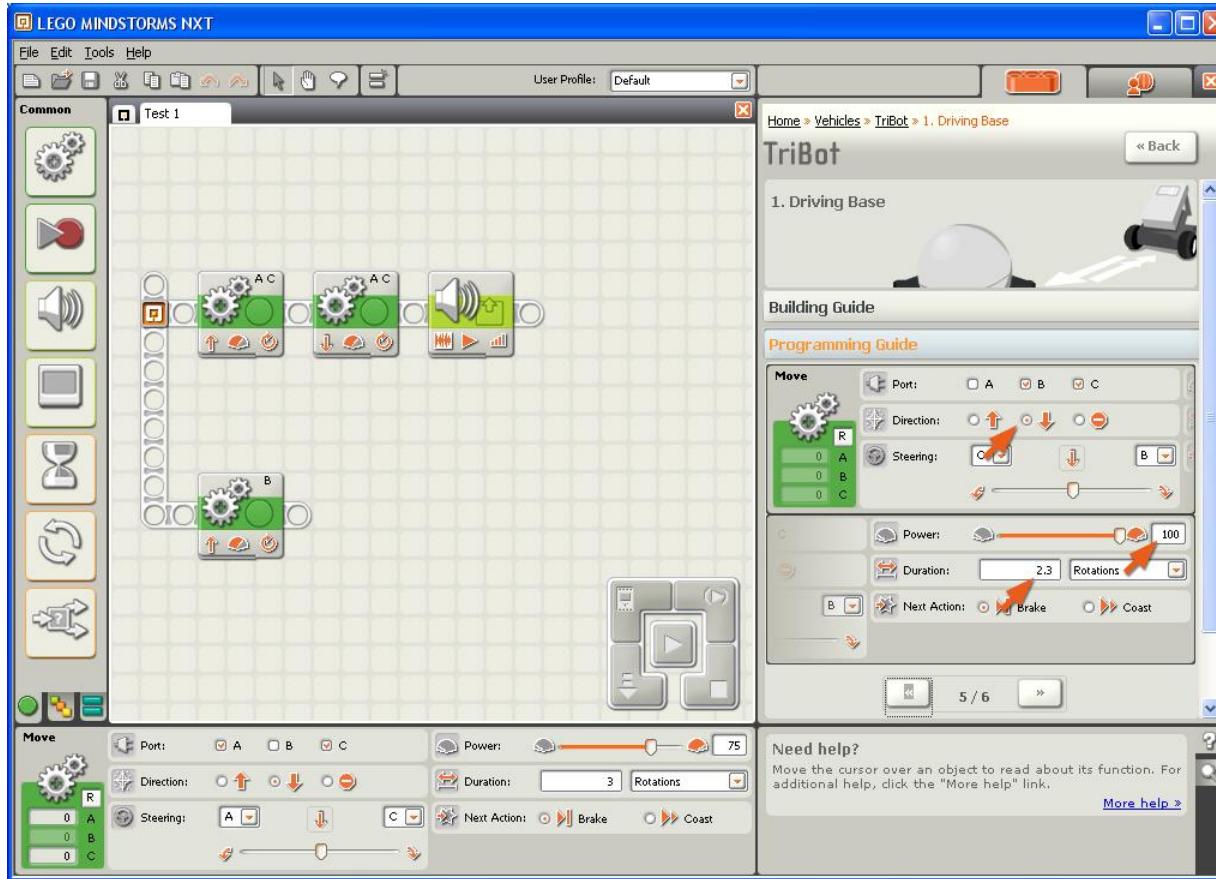
ABOUT UML

- UML – “ Swiss knife ”: good for everything ... more or less

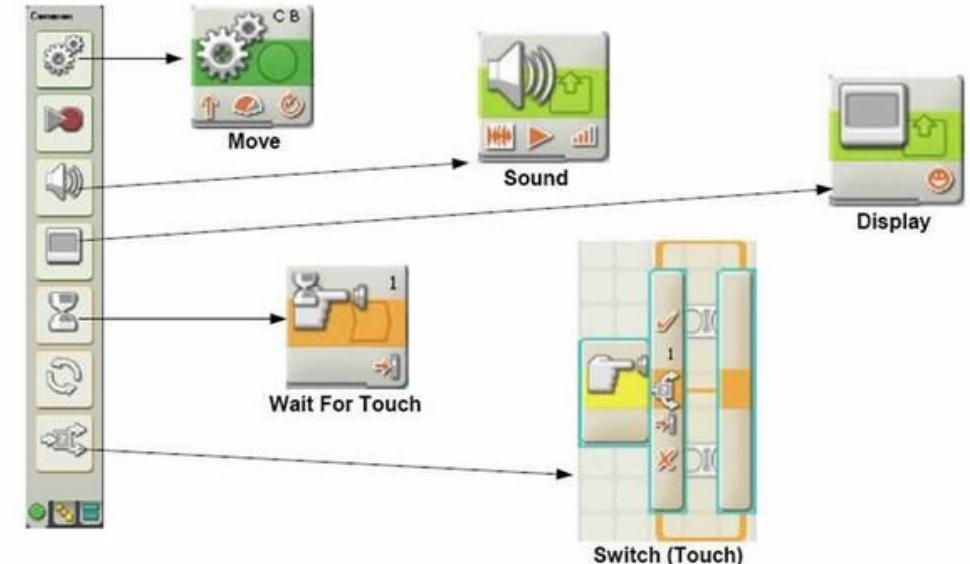


- We need a set of specific tools instead

MINDSTORMS - NO CODE GRAPHIC EDITOR

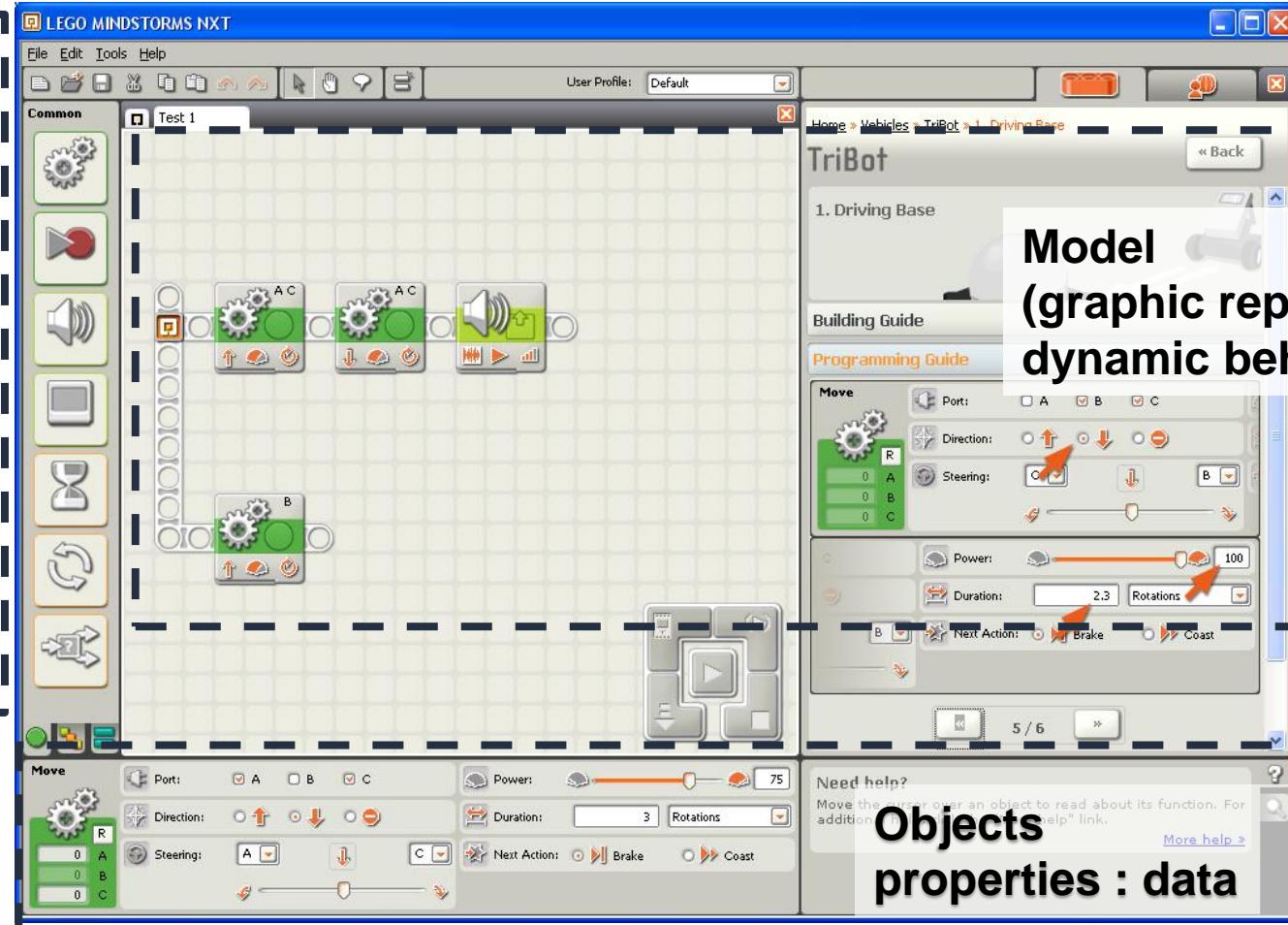


Common Blocks



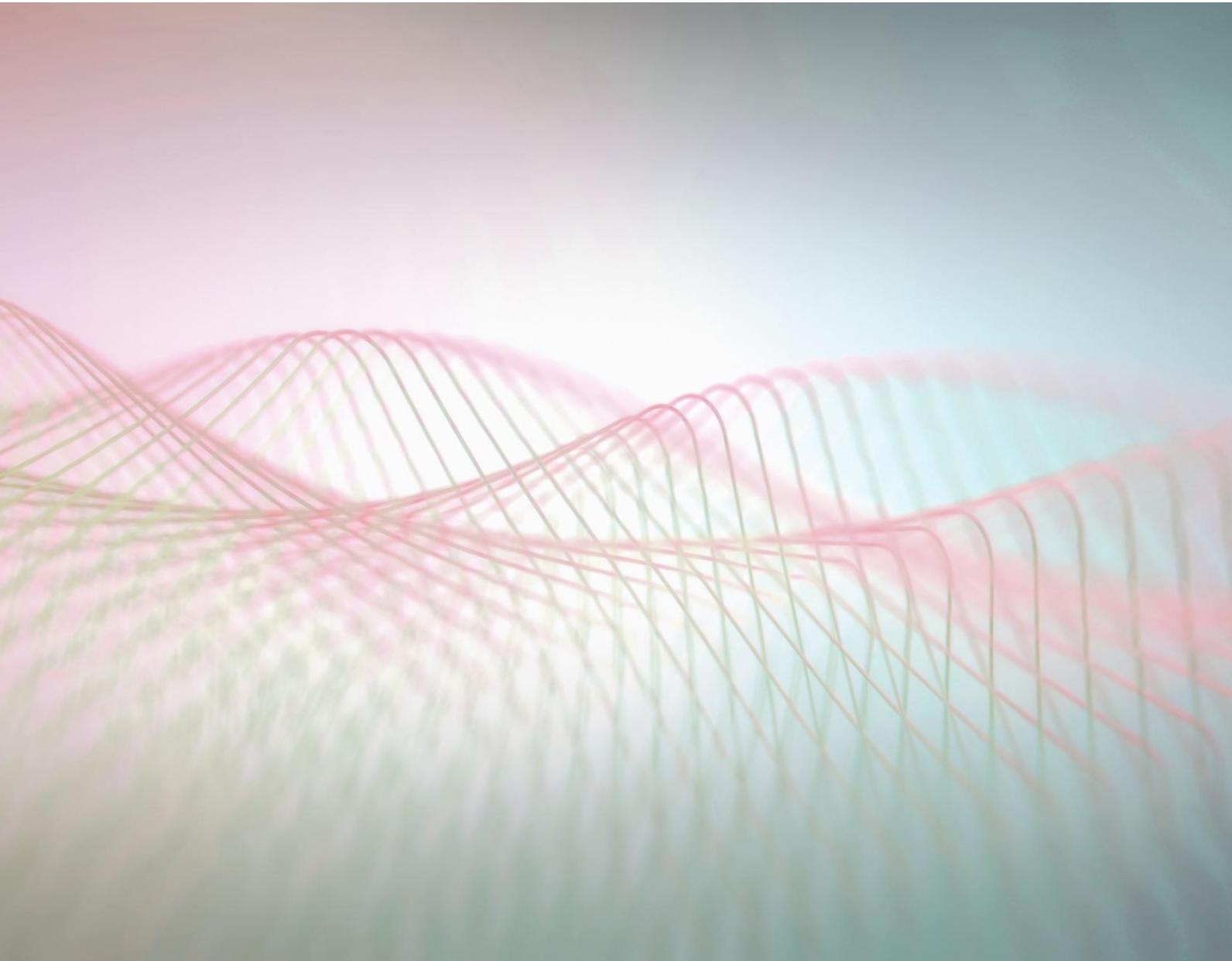
MINDSTORMS - PROGRAMMING ENVIRONMENT

Modeling primitives



Model
(graphic representation):
dynamic behavior

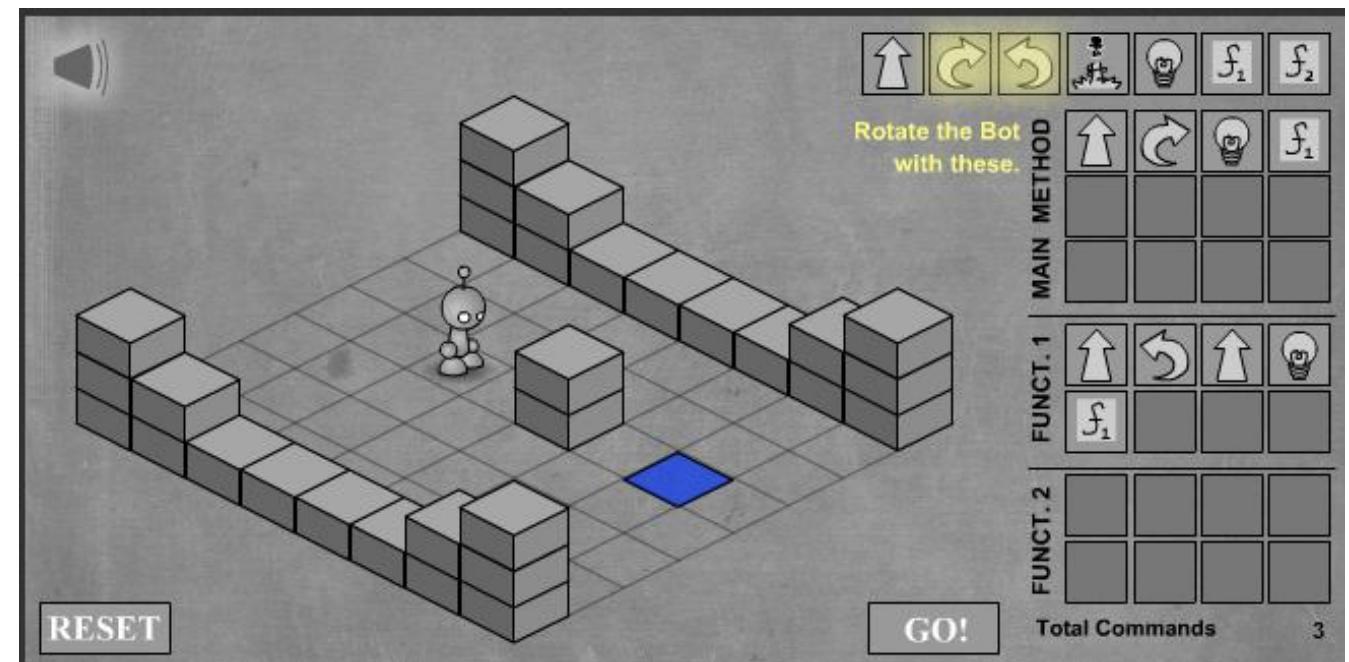
Objects
properties : data



MODELS
EVERYWHERE

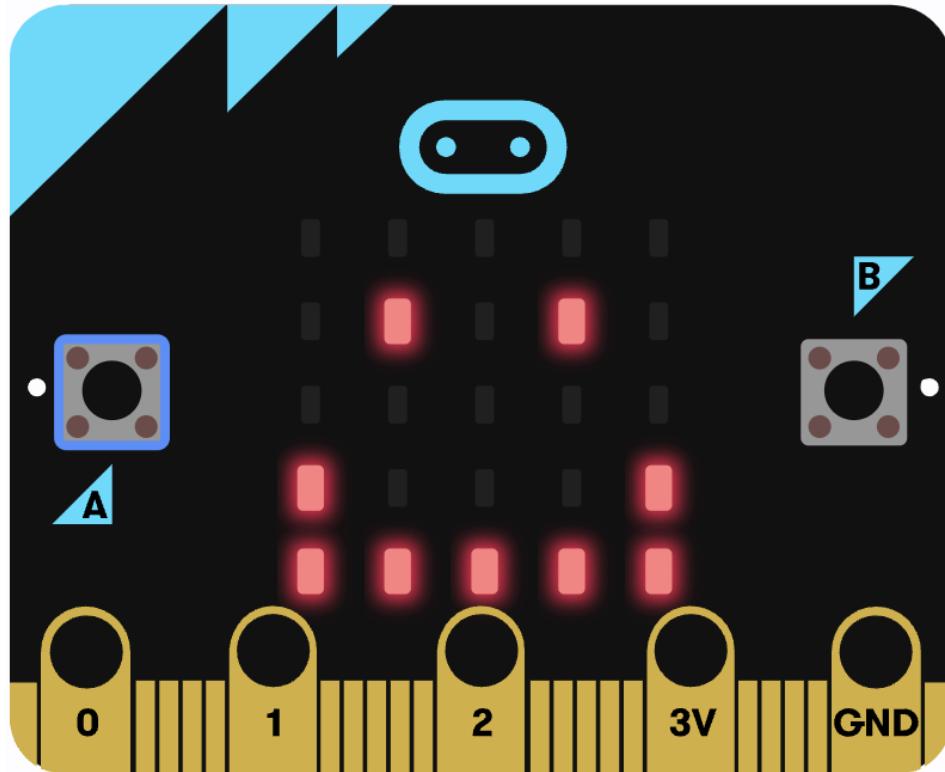
LIGHTBOT

- Game: robot controlled by simple commands
 - Graphical programming interface
 - Graphical "debugger"



MICRO:BIT

- Programming embedded systems in an easy way



CCG

■ Collectible card game – customized



```
CARD {  
    Name: "Kvatch Solder";  
    Type: Creature;  
    Attack: 2;  
    Health: 3;  
    Cost: 3;  
    Guard: true;  
}
```



FORM EDITOR

Select SELECT ▾

Name (?)
Select

Description (?)

Options (?)

Option 1 ⊖ +

Option 2 ⊖ +

Option 3 ⊖ +

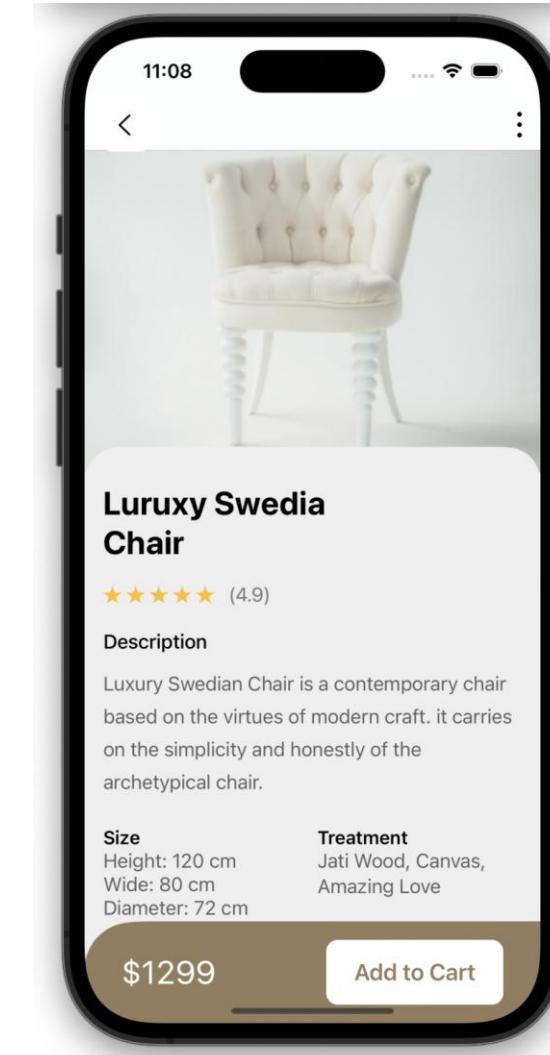
Add Option ⊕

Validation (?) Required (?)
 No

Size (?) Field Layout (?)
Medium ⊖ Default ⊖

CSS Classes (?)

Delete ⊖



SQL / NOSQL

SQL: general-purpose declarative language specialized in relational database definition and querying

- Independent from database implementation
- Can be used instead of programming
- Adds an abstraction layer

```
SELECT Book.title ,  
       count(*) AS Authors  
  FROM Book  
 JOIN Book_author YOU Book.isbn =  
       Book_author.isbn  
 GROUP BY Book.title ;  
SQL query
```

NoSQL: Specialized databases and query languages

- New languages to use new search algorithms
- Languages highlight the strengths of algorithms

```
(: Person {name: string })  
-[:ACTED_IN {roles: [ string ]}]->  
(: Movie {title: string , released: number  
})
```

Cypher query

Neo4j promises an efficient **join algorithm**, so it uses this as a language element :
$$(x) -[] -> (y)$$

FURTHER EXAMPLES

- Markup languages: HTML, CSS, Latex
- First steps of programming: Logo, Scratch
- Game engine programming: UnrealScript
- Hardware description: VHDL, Verilog
- Financial software: HR rule system, Drools
- Embedded systems: Yakindu, AUTOSAR

SPECIALIZED LANGUAGE VS. GENERAL PURPOSE LANGUAGE

Specialized language	General purpose languages
Uses the concepts of the domain (e.g. bicycle , HTML input form)	Uses general concepts (e.g. class , function , XML tag)
For experts of the domain	For programmers
Special goals	General goals
Free syntax	Strict, rigid syntax
Custom processing and environment	Supported by development environments

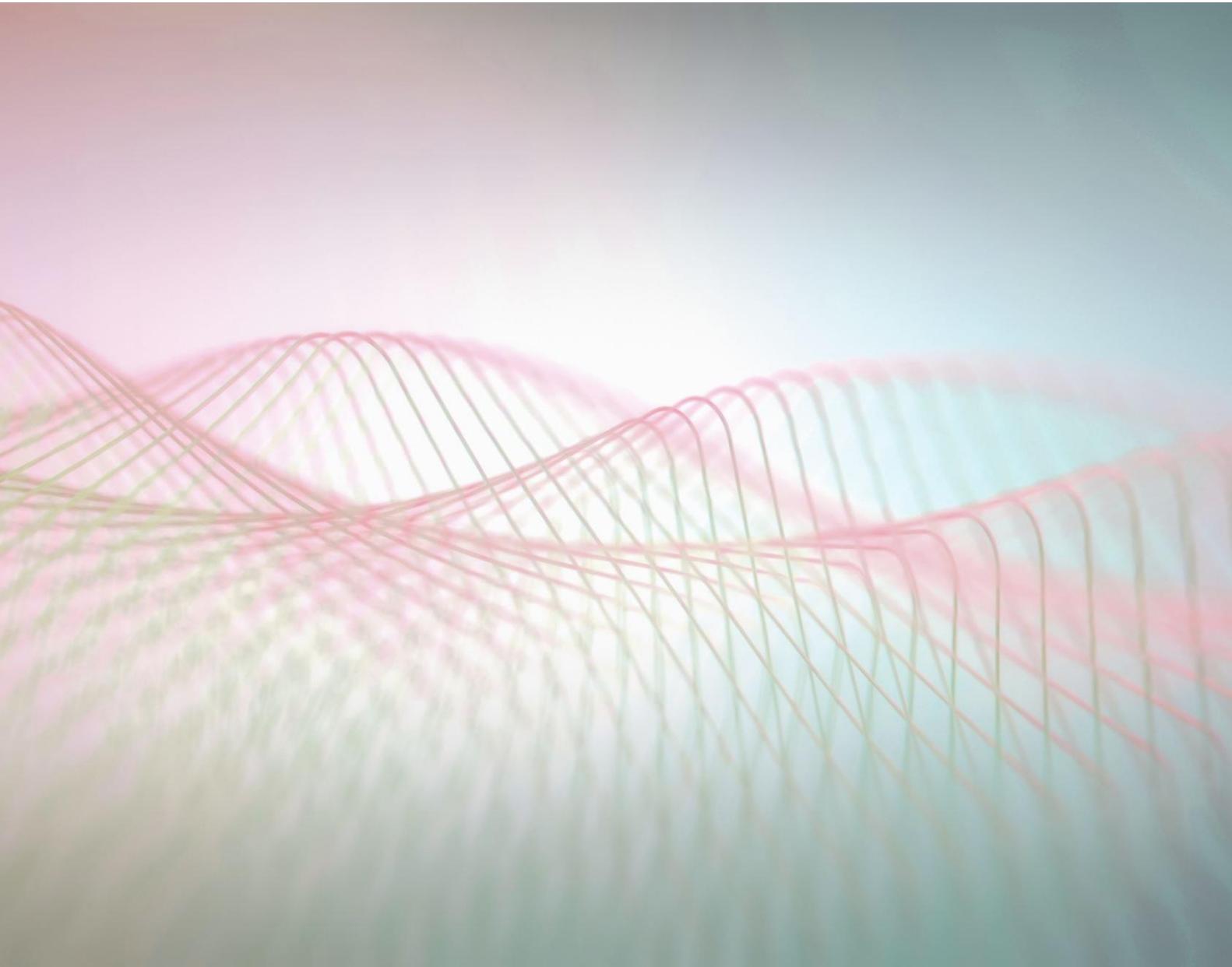
- Specialized language = Domain-Specific Language (DSL)
- General purpose language = General-Purpose (Programming) Language (GPL)
 - There exist general purpose languages not created for programming: XML, JSON

DOMAIN SPECIFIC LANGUAGES

- Domain-Specific Language, DSL
 - Special language applicable only to the domain
 - Limited item set
 - Strongly specialized rules and notation
 - Made for a given product(family)
 - Full code generation can be supported
 - Low code – No code

LANGUAGES COMPONENTS

- Language components
 - Syntax (structure + appearance)
 - Abstract syntax (building blocks, relationships)
 - Textual language : grammar, derivation rules
 - Graphic languages : metamodeling
 - Specific syntax (appearance)
 - Semantics (meaning)

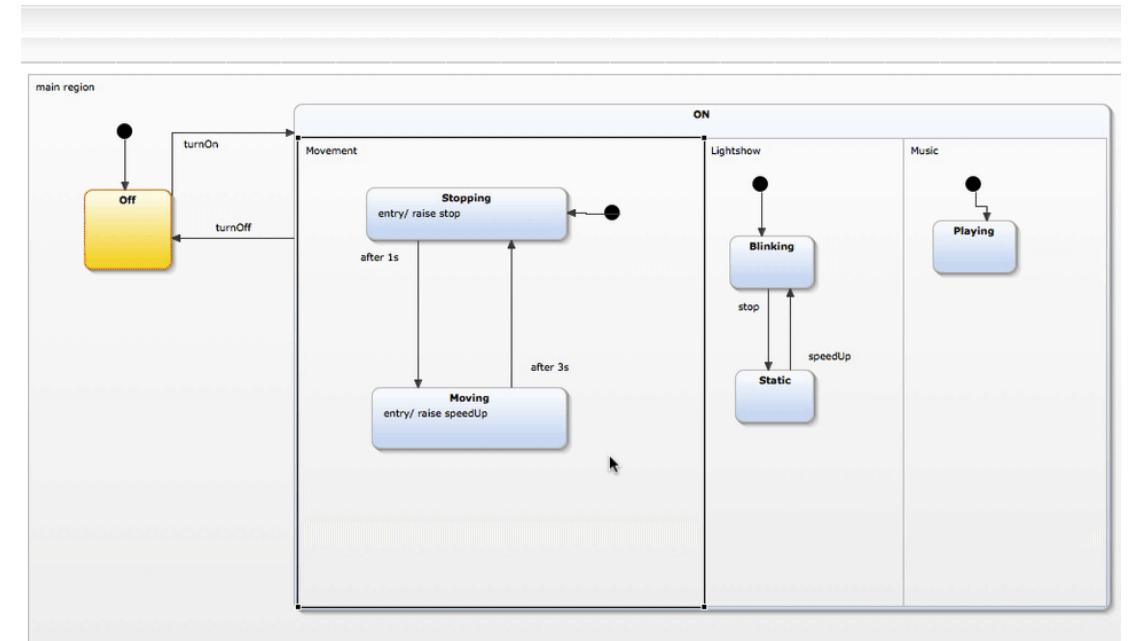


MODEL-BASED DEVELOPMENT

LANGUAGE != MODELING

- A specialized language in itself is usually not enough!
- We need:
 - Editor environment
 - Debugger / simulator
 - Model processor (e.g. code generator)
 - Additional functions (e.g. correctness check)

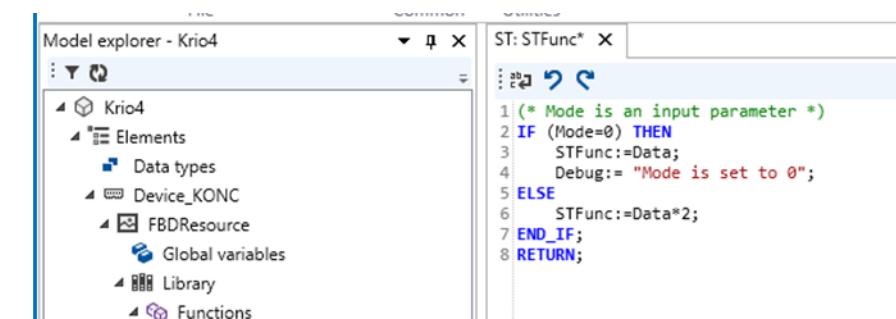
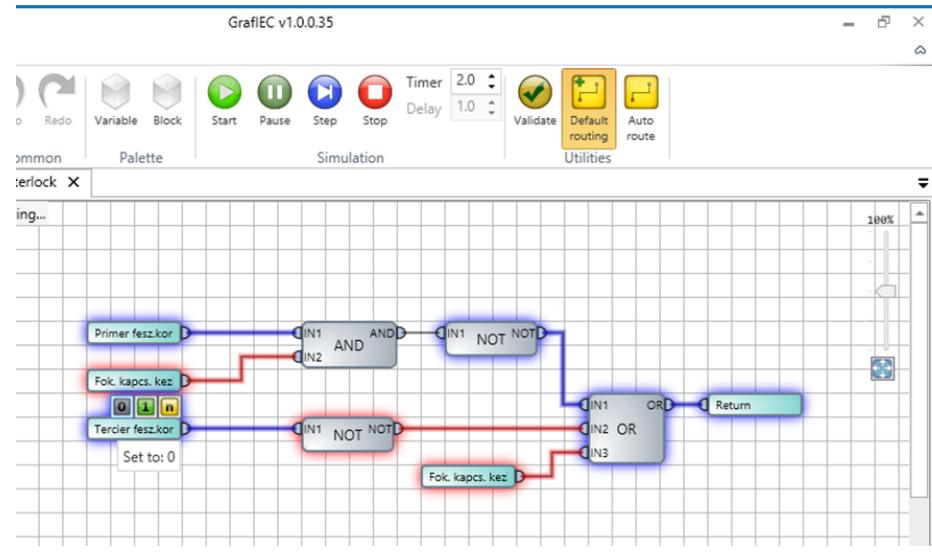
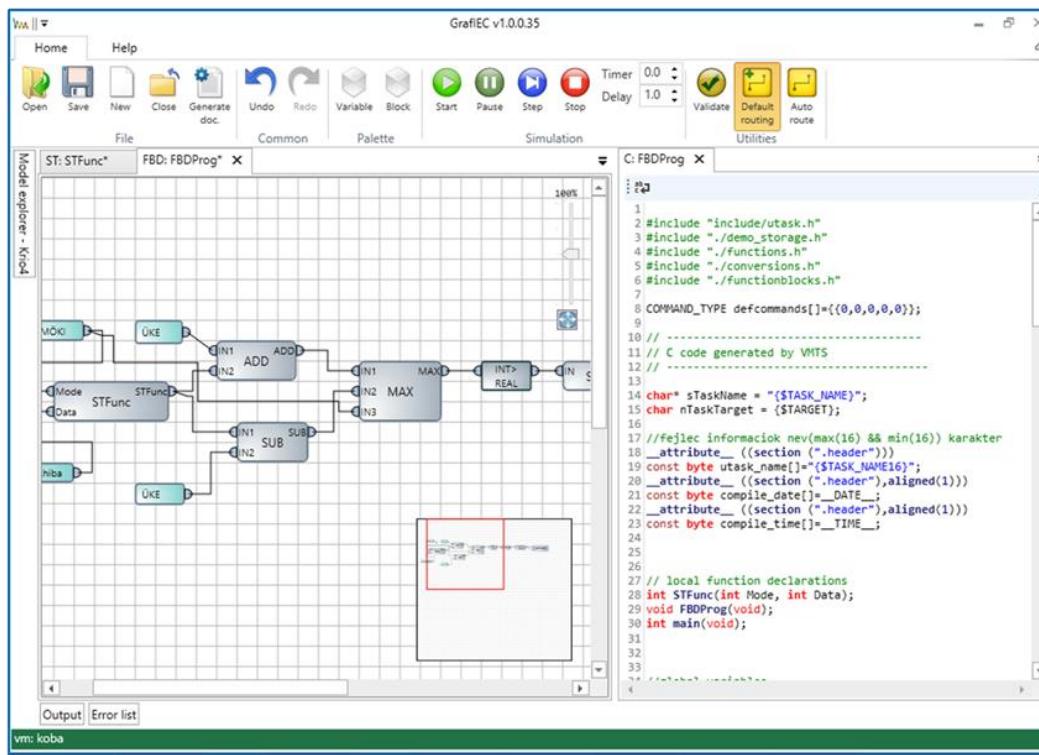
- Modeling environment
- Simulator
- Code generator to several languages
- Mathematical correctness check



<https://blogs.itemis.com/en/how-to-simulate-a-statechart-model>
<https://github.com/ftsrcg/gamma>

GRAPH IEC

■ IEC 61131 industrial standard



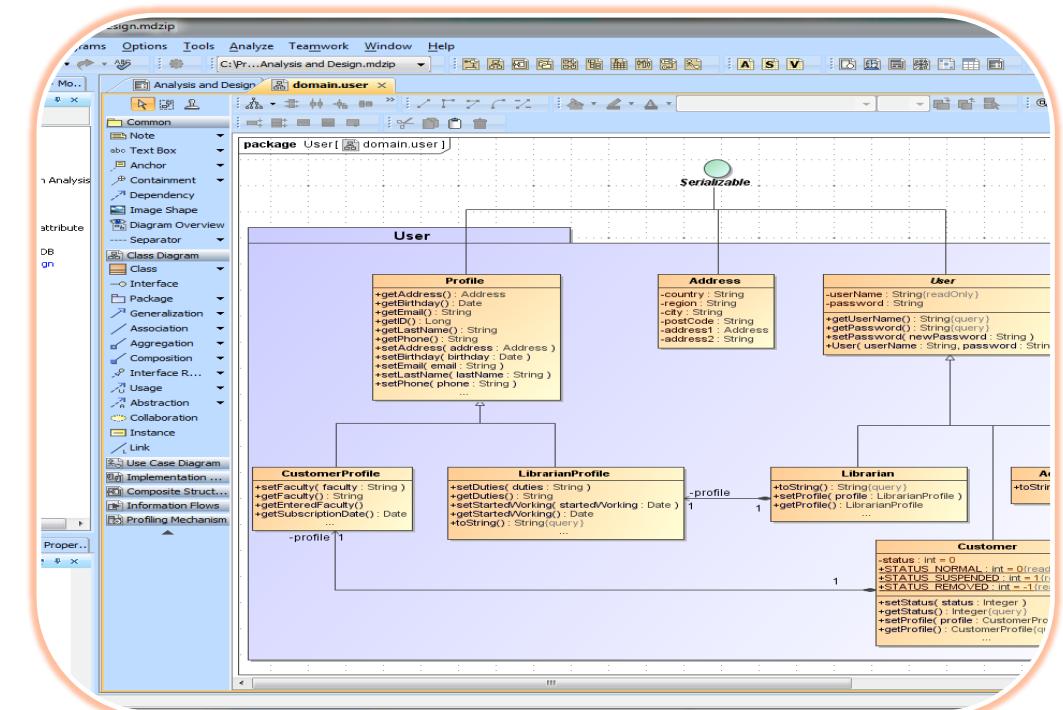
INDUSTRIAL MODELING TOOLS

- Modeling languages and modeling tools are dominant in many fields
 - Developers work exclusively on these tools
 - The standard prescribes the way of use

DO-178C, Software Considerations in Airborne Systems and Equipment Certification. SG4: Model Based Development and Verification

- It is mandatory to cooperate with these tools
 - SysML ,AUTOSAR, MATLAB, ...

Examples:



SysML : MagicDraw

MODEL-BASED DEVELOPMENT

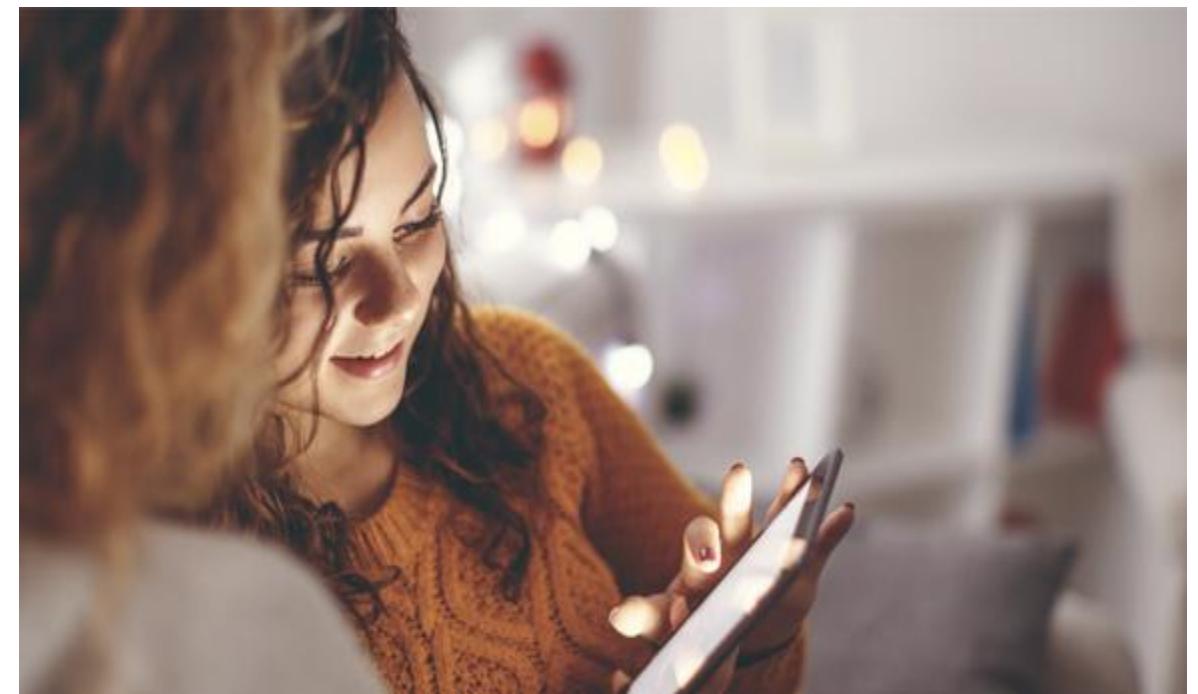
- Productivity and quality
 - Familiar linguistic elements and concepts to domain users
 - Minor changes can be accomplished even without developers
 - Domain rules are automatically enforced
 - Unimportant information is hidden
 - Targeted mathematical analysis
 - Multiplatform development
- Drawback: initial cost may be significant
 - Development and maintenance of the languages and tools

TODAY

Chapter I. Why?

Chapter II. About what?

Chapter III. How?



THE SUBJECT

- Data sheet: <https://portal.vik.bme.hu/kepzes/targyak/VIAUMA22>
- Education - cooperation of three department
 - Gergely Mezei, Ferenc Somogyi, Norbert Somogyi
(Dep. Of Automation and Applied Informatics)
 - Balázs Simon (Dep. of Control Engineering and Information Technology)
 - Oszkár Semeráth (Dep. of Measurement and Information Systems)
- Lectures - each week
 - Theory, technology overview
- Practice - every second week
 - Theory illustration in practice
 - Demos, practical examples, case studies

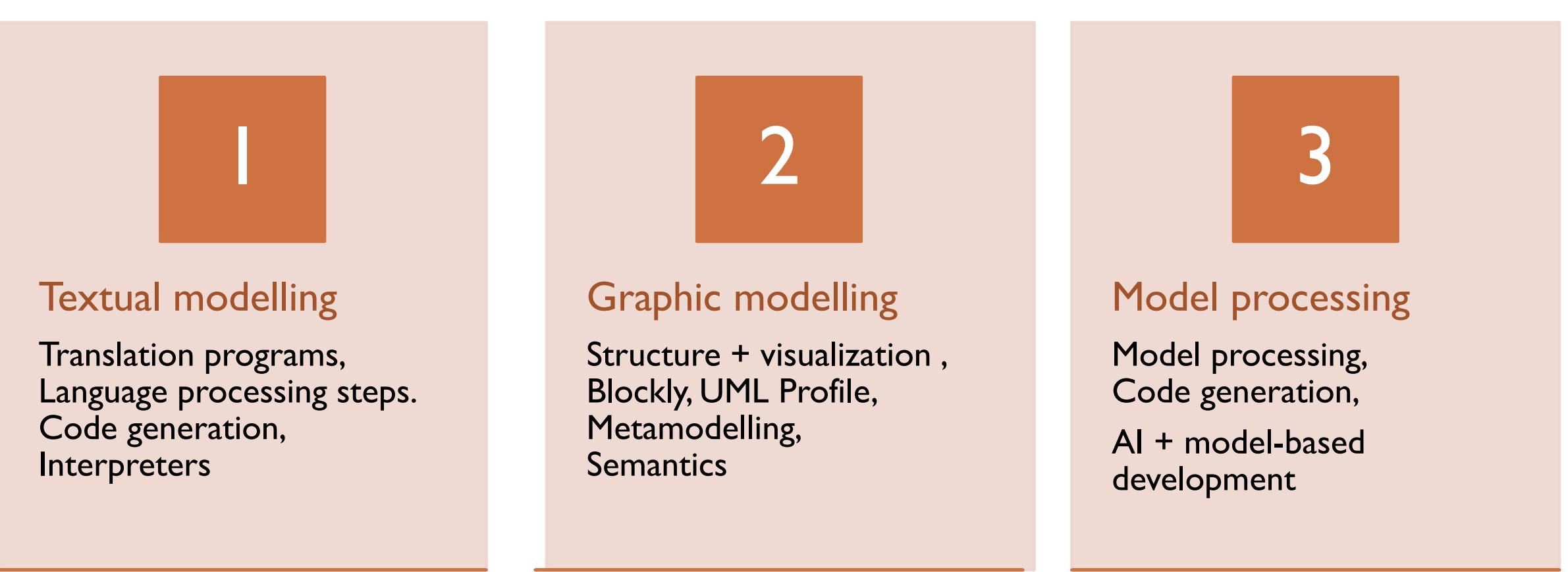


REQUIREMENTS

- 5 homemade tasks
 - Github
 - Based on the demos of practices
 - At least 3 of 5 must be successful
- Mid-term (2025. 04. 20. 18-20h)
- Exam
 - Written exam
 - The midterm results influences the final grade (48p + 10p + 14p)

SAVE POINTS

- Save points (SP)!
 - Helps to pass the midterm and the exam
 - Can gained by attending to lectures
 - 1 point – presence
 - 1 point – answer the question of the lecturer or added automatically if no questions are asked
 - 1 SP = 1 point in Midterm/exam
 - 9 lectures before mid-term -> max 18 SP for mid-term
 - 4 lectures after mid-term -> max 8 SP for exam (mid-term SPs cannot be used)
 - Can be used to pass, but not to further improve the grade



1

Textual modelling

Translation programs,
Language processing steps.
Code generation,
Interpreters

2

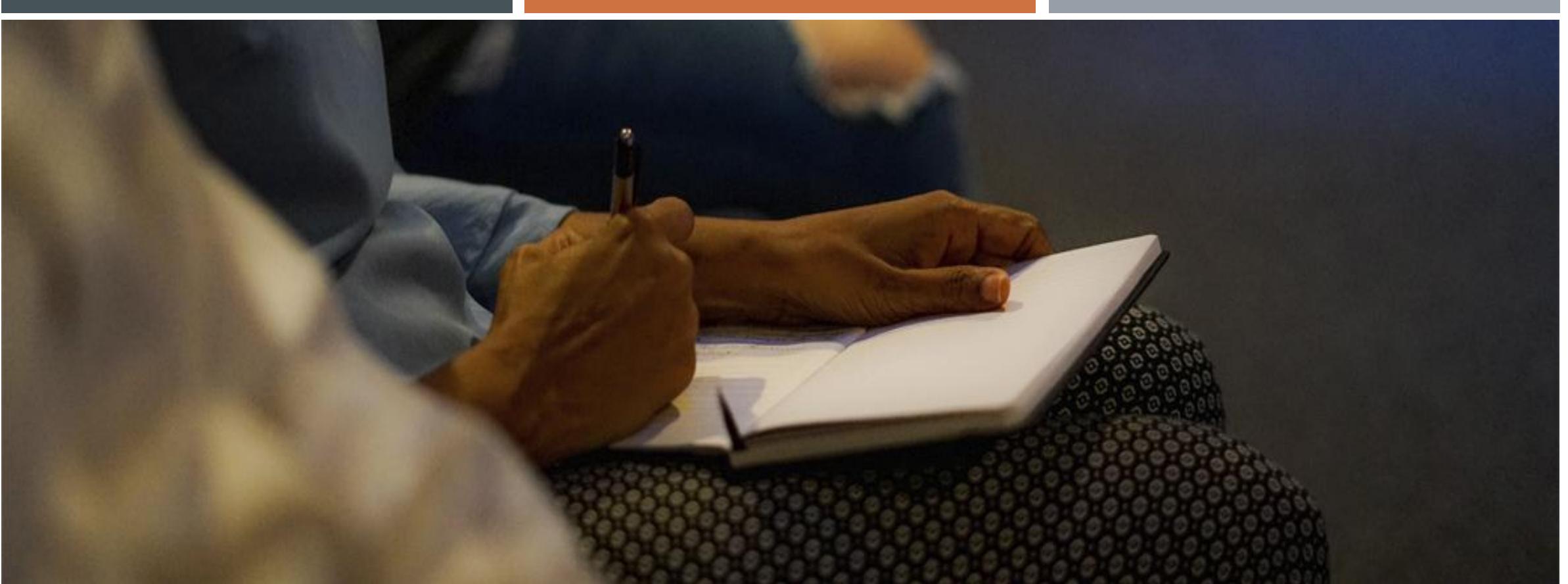
Graphic modelling

Structure + visualization ,
Blockly, UML Profile,
Metamodelling,
Semantics

3

Model processing

Model processing,
Code generation,
AI + model-based
development



THANK YOU FOR YOUR ATTENTION