

Extended from a  
version by Arjan Mooij

# Domain-Specific Languages (DSL)

Software Systems (Computer & Embedded Systems Engineering)

Rosilde Corvino

January 2023 (week 9)

Based on the DSL  
awareness training of ESI.

An initiative of industry, academia and TNO

# Objectives

## At the end of the course, you should be able to:

- Explain the purpose of Domain-Specific Languages, including several application areas
- Explain the basics of grammars and parsing
- Create basic textual Domain-Specific Languages, including editor support, validation and generators

## Assessment:

- Modeling assignment using Domain-Specific Languages (in groups of 2 students)
- Reflection document on Model-Based Development (individual)

# Agenda for Domain-specific language

(Each week the Software Systems course has 2 lecture hours + 4 lab hours)

- **Week 9 Lecture**
  - 15 minutes      Why DSL?
  - 30 minutes      Formal grammars
  - 15 minutes      Break
  - 30 minutes      Generator and Validator
  - 5 minutes      Application areas
  - 10 minutes      General conclusions
- **Week 9 Lab**
  - Follow the manual “Creating a Domain Specific Language (DSL) with Xtext” up to section 3.5
  - Modeling assignment (Define a formal grammar for your DSL)

# Motivation

## Domain-Specific Languages (DSL)

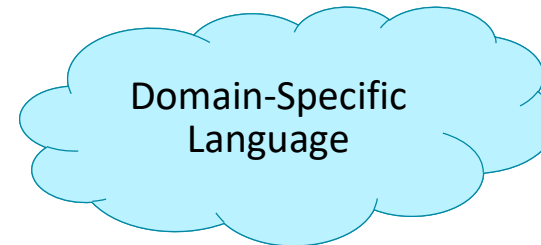
# What is Jargon?

## Oxford dictionaries:

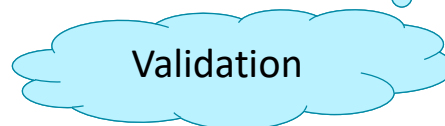
- **Special words or expressions**
  - used by a profession or group
  - that are difficult for others to understand

## Wikipedia:


- **Terminology defined in relationship to**
    - a specific activity, profession, group, or event**
      - ... a barrier to communication with those not familiar with the language
- **A standard term may be given a more precise or unique usage**



Domain-Specific  
Language



Validation



Code  
Generation

## What is a Domain-Specific Language?

- What are your associations with the term Domain-Specific Language?
- Do you know any Domain-Specific Languages?

Think → Pair → Share

# What is a Domain-Specific Language?

## General-purpose programming languages:

- C, C++, Java, Python, etc.

## Domain-specific languages:

- HTML for web pages
- SQL for relational database queries

## Domain-specific languages:

- Specifically designed for a specific application by a single company

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

```
SELECT *
FROM Book
WHERE price > 100
ORDER BY title;
```

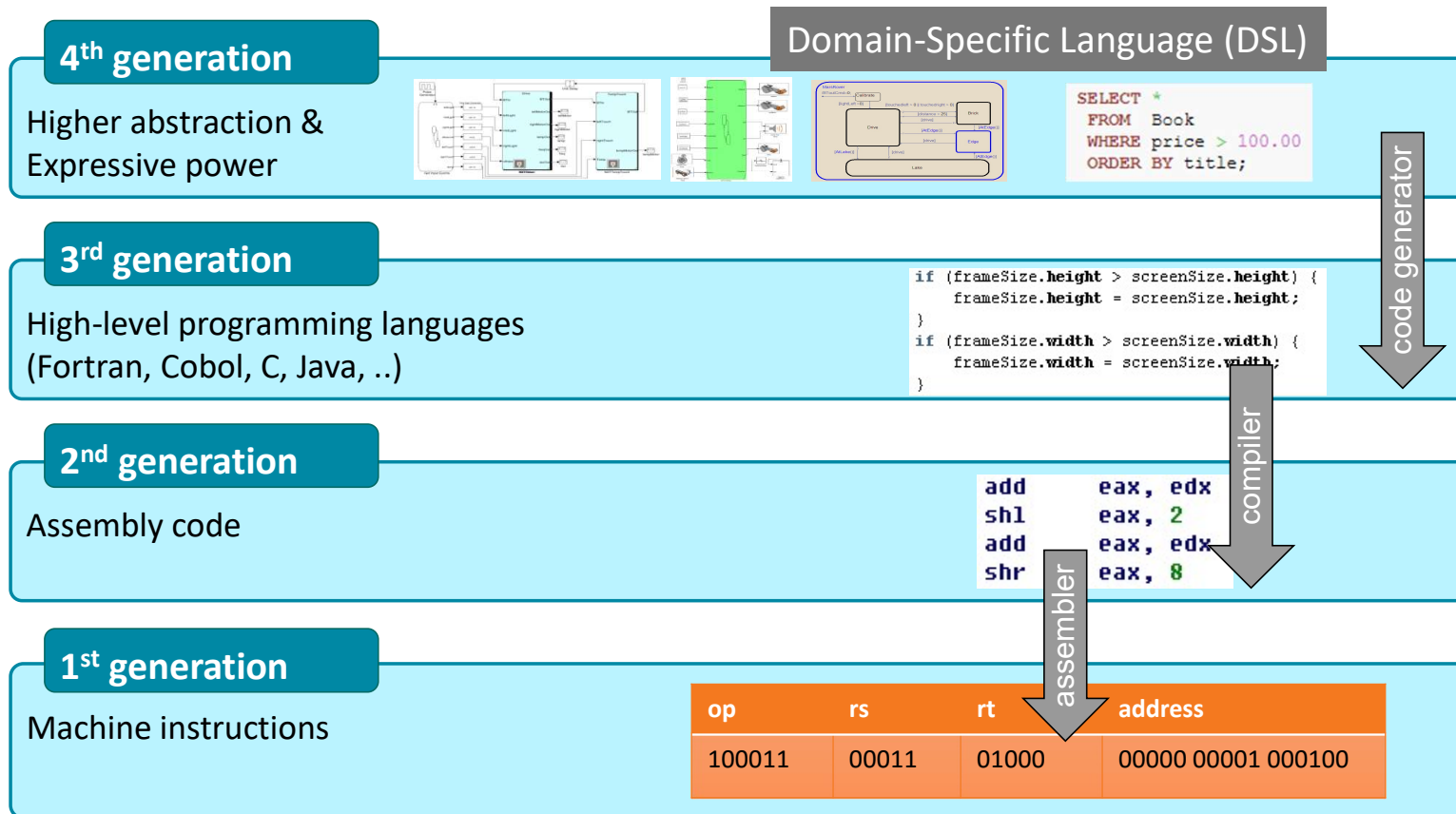
## What about the other model-based techniques from this course?

- **PlantUML** for **Unified Modeling Language (UML)**
- **CREATE Statechart Tools** for **Finite-State Machines (FSM)**

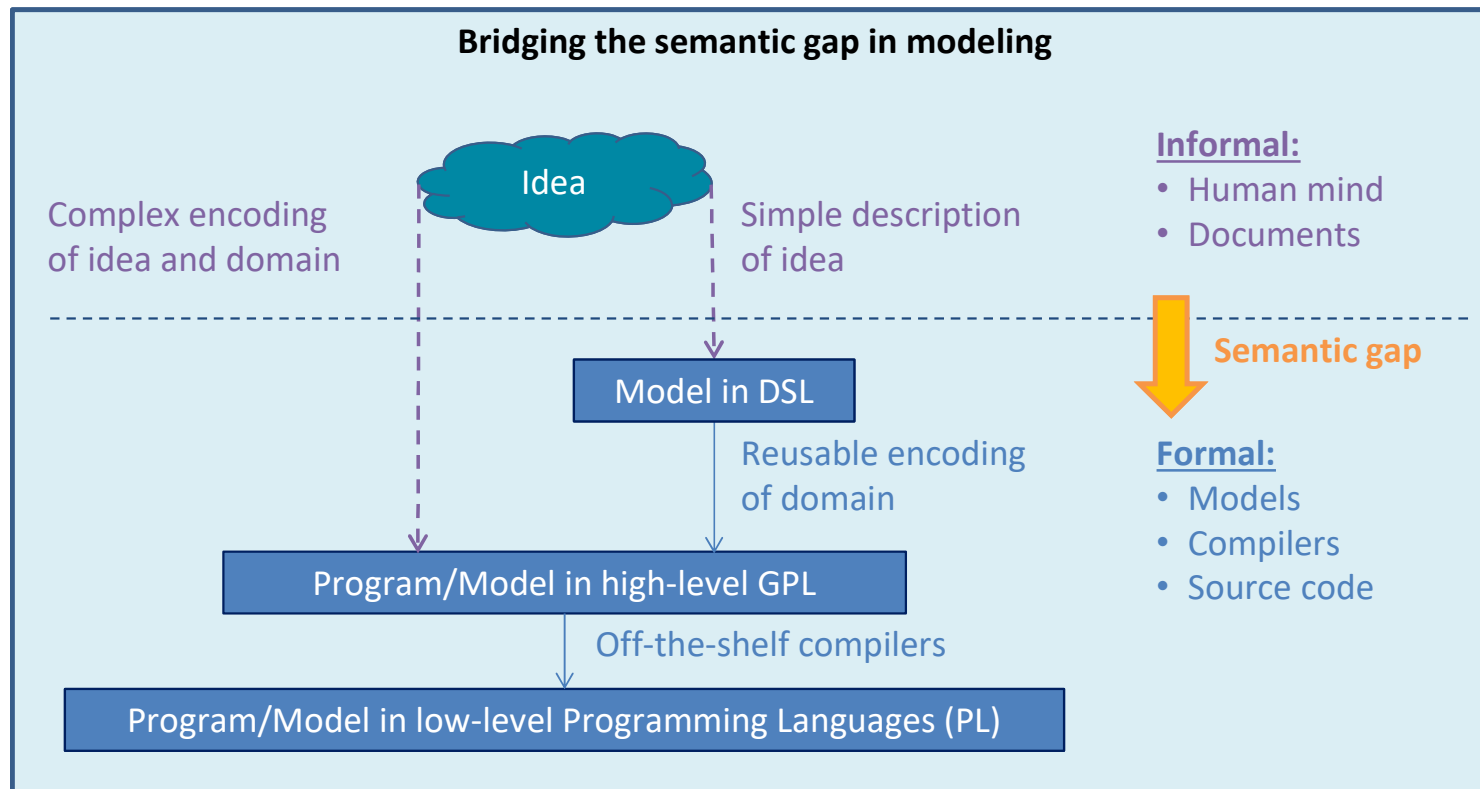
→ **Horizontal DSLs**



# Generations of programming languages



## Modeling Perspective on DSLs



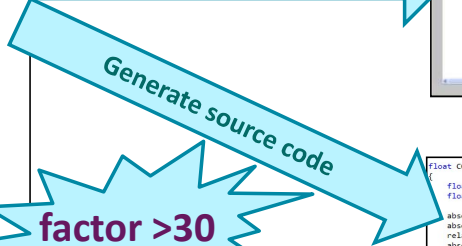
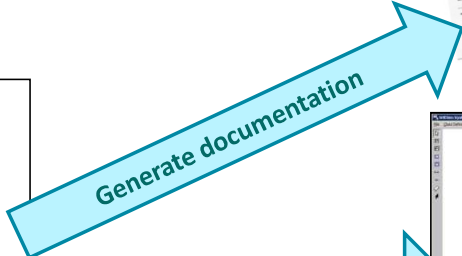


# DSL as Central Artifact

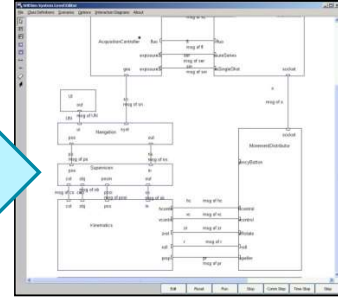
```

restriction VeryCloseTableTopAndBeam
  activation
    Distance(TableTop, Beam) < 20 mm
  effects
    userGuidance "TableTop and Beam very close"
    relative limit TableTop*[Rotation, Translation],
                  Beam*[Rotation, Translation]
    at 0

restriction ApproachingTableTopAndBeam
  activation
    Distance(TableTop, Beam) < 35 mm + 15 cm
  effects
    userGuidance "TableTop and Beam approaching"
    relative limit TableTop*[Rotation, Translation],
                  Beam*[Rotation, Translation]
    at (Distance(TableTop, Beam) - 35 mm) / 15 cm
  
```



factor >30



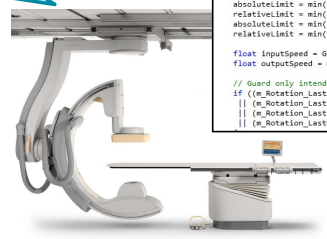
```

Float Object_Beam::GetRotationScaling()
Float absoluteLimit = GEN_t_float_max;
Float relativeLimit = 1.;

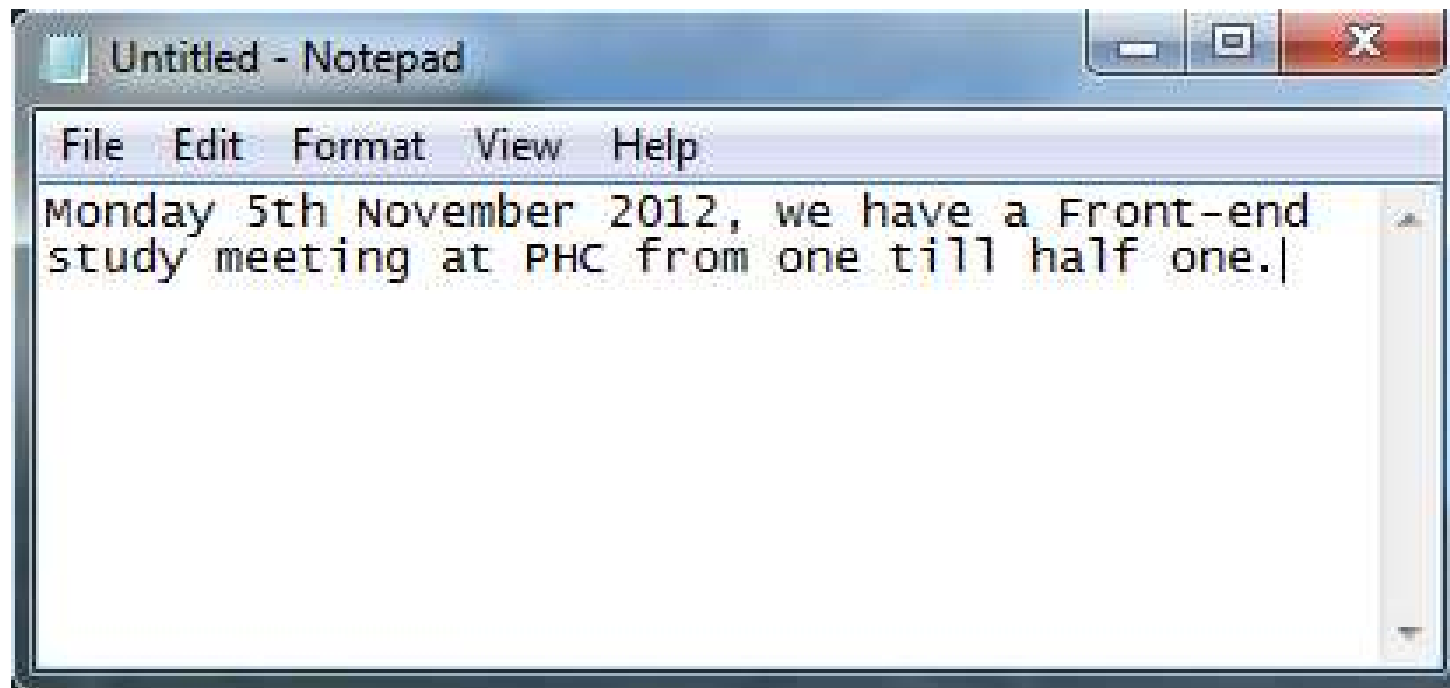
absoluteLimit = min(absoluteLimit, m_Rotation_AbsoluteLimit_DetectorRotationInBetween);
absoluteLimit = min(absoluteLimit, m_Rotation_AbsoluteLimit_VeryCloseTableTopAndBeam);
relativeLimit = min(relativeLimit, m_Rotation_RelativeLimit_ApproachingTableTopAndBeam);
absoluteLimit = min(absoluteLimit, m_Rotation_AbsoluteLimit_VeryCloseTableTopAndDetector);
relativeLimit = min(relativeLimit, m_Rotation_RelativeLimit_ApproachingTableTopAndDetector);
absoluteLimit = min(absoluteLimit, m_Rotation_AbsoluteLimit_VeryCloseTableBaseAndBeam);
relativeLimit = min(relativeLimit, m_Rotation_RelativeLimit_ApproachingTableBaseAndBeam);
absoluteLimit = min(absoluteLimit, m_Rotation_AbsoluteLimit_VeryCloseTableBaseAndDetector);
relativeLimit = min(relativeLimit, m_Rotation_RelativeLimit_ApproachingTableBaseAndDetector);

Float inputSpeed = GetRequestedRotSpeed();
Float outputSpeed = min(inputSpeed * relativeLimit, absoluteLimit);

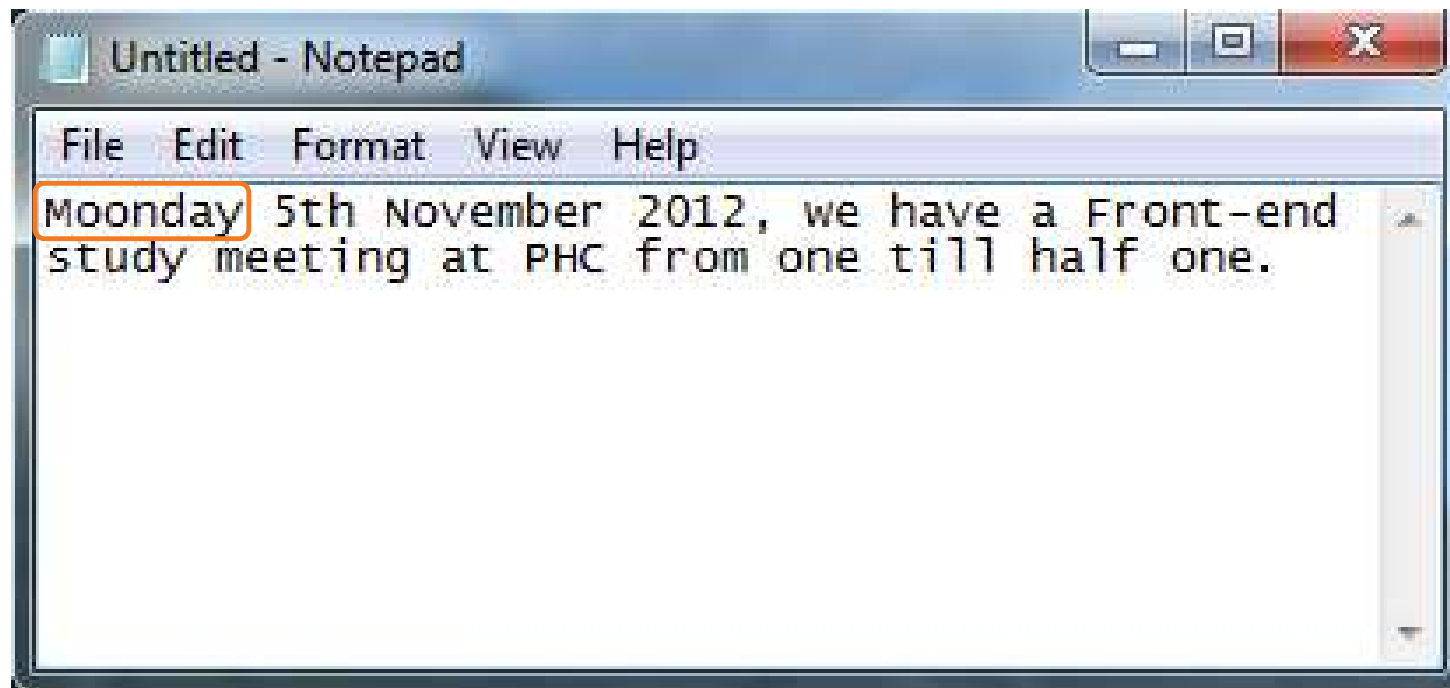
// Guard only intended to reduce calls to logging
if ((m_Rotation_LastAbsoluteLimit != absoluteLimit)
    || (m_Rotation_LastRelativeLimit != relativeLimit)
    || (m_Rotation_LastInputSpeed != inputSpeed)
    || (m_Rotation_LastOutputSpeed != outputSpeed))
  
```



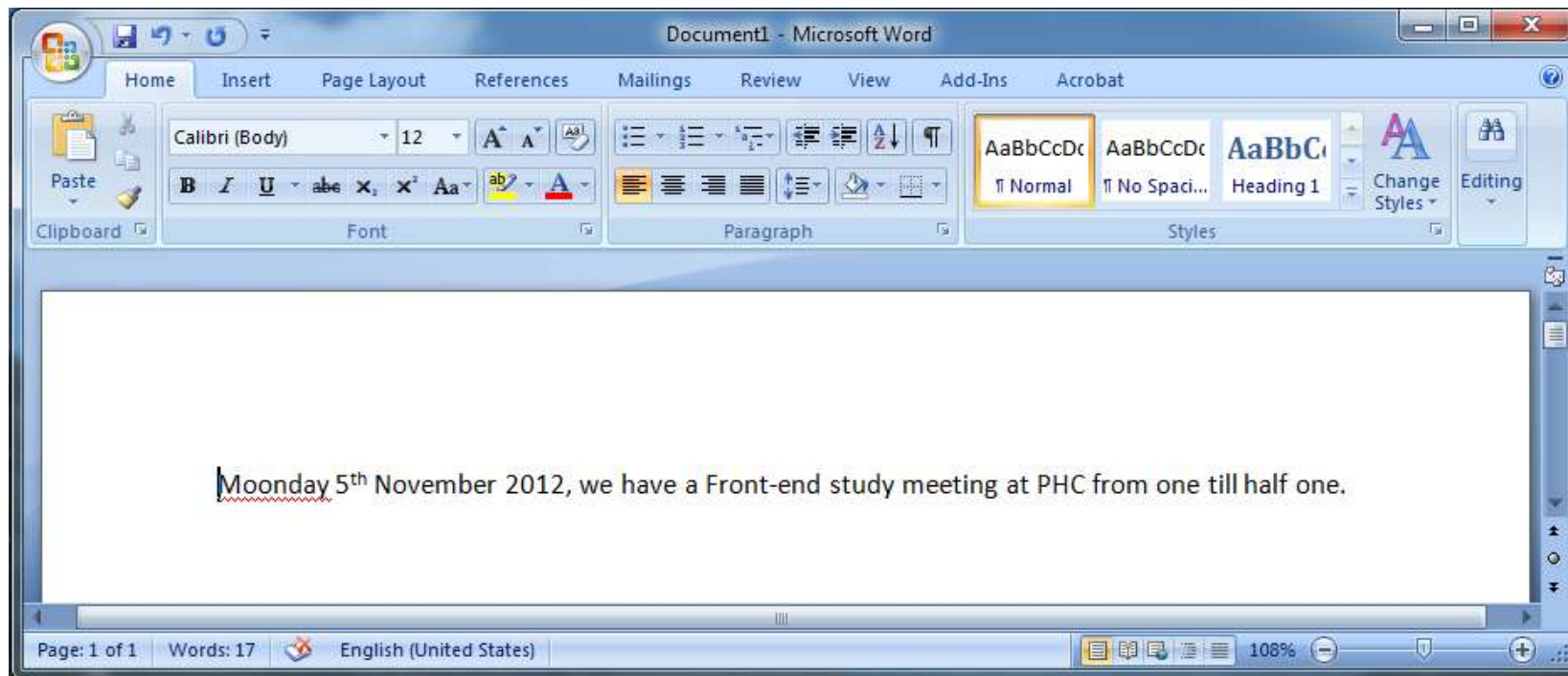
## Microsoft Notepad: A Correct Sentence



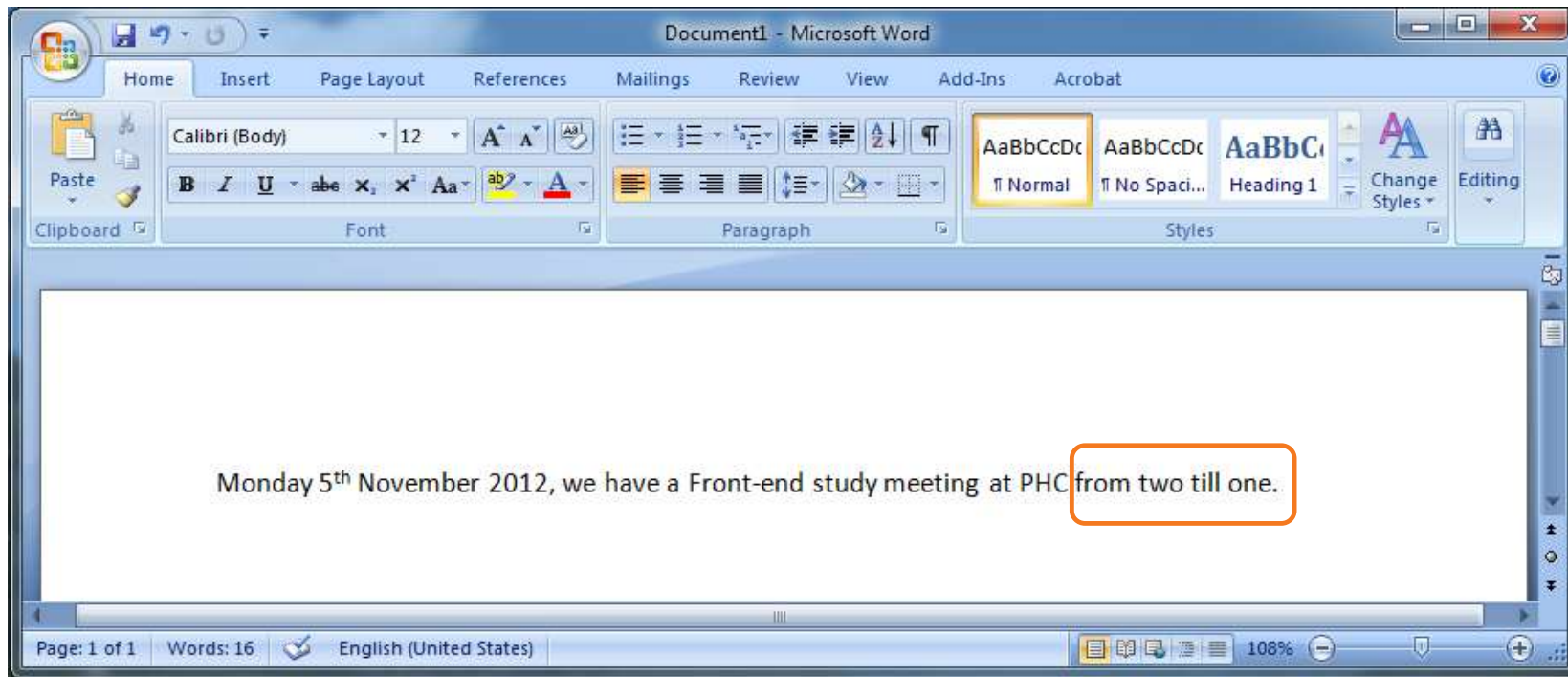
## Microsoft Notepad: No Problem Detected, but ...



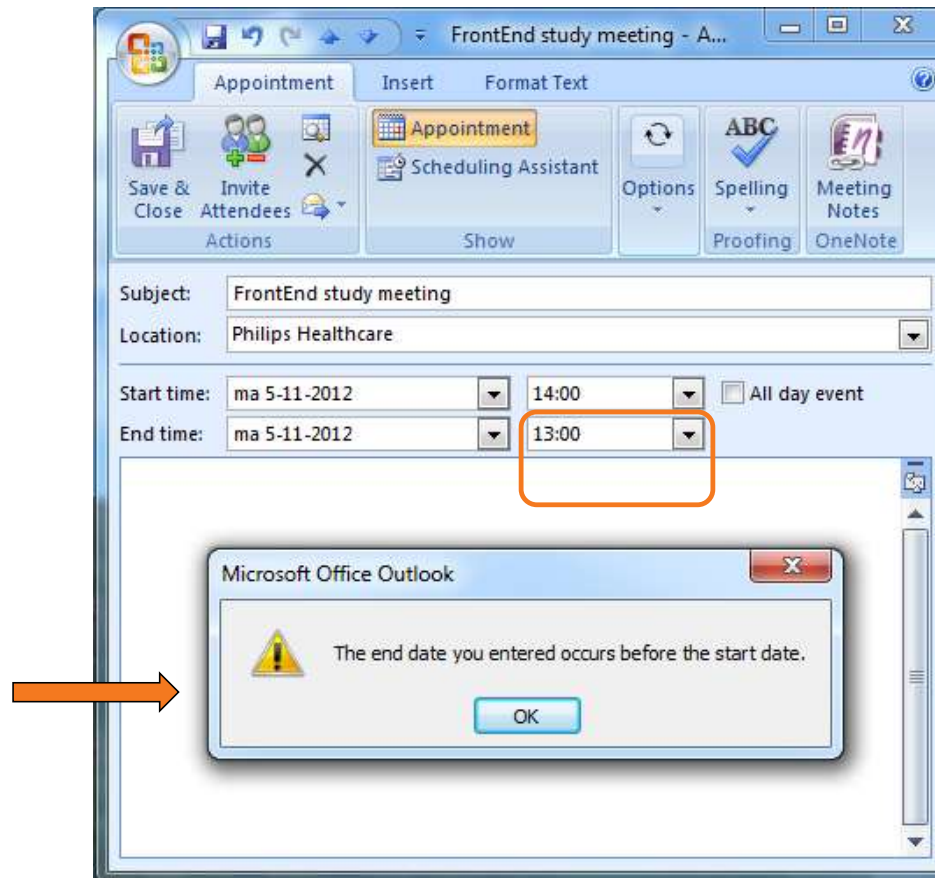
## Microsoft Word: Spelling Error



## Microsoft Word: No Problem Detected, but ...



## Microsoft Outlook: Wrong Times

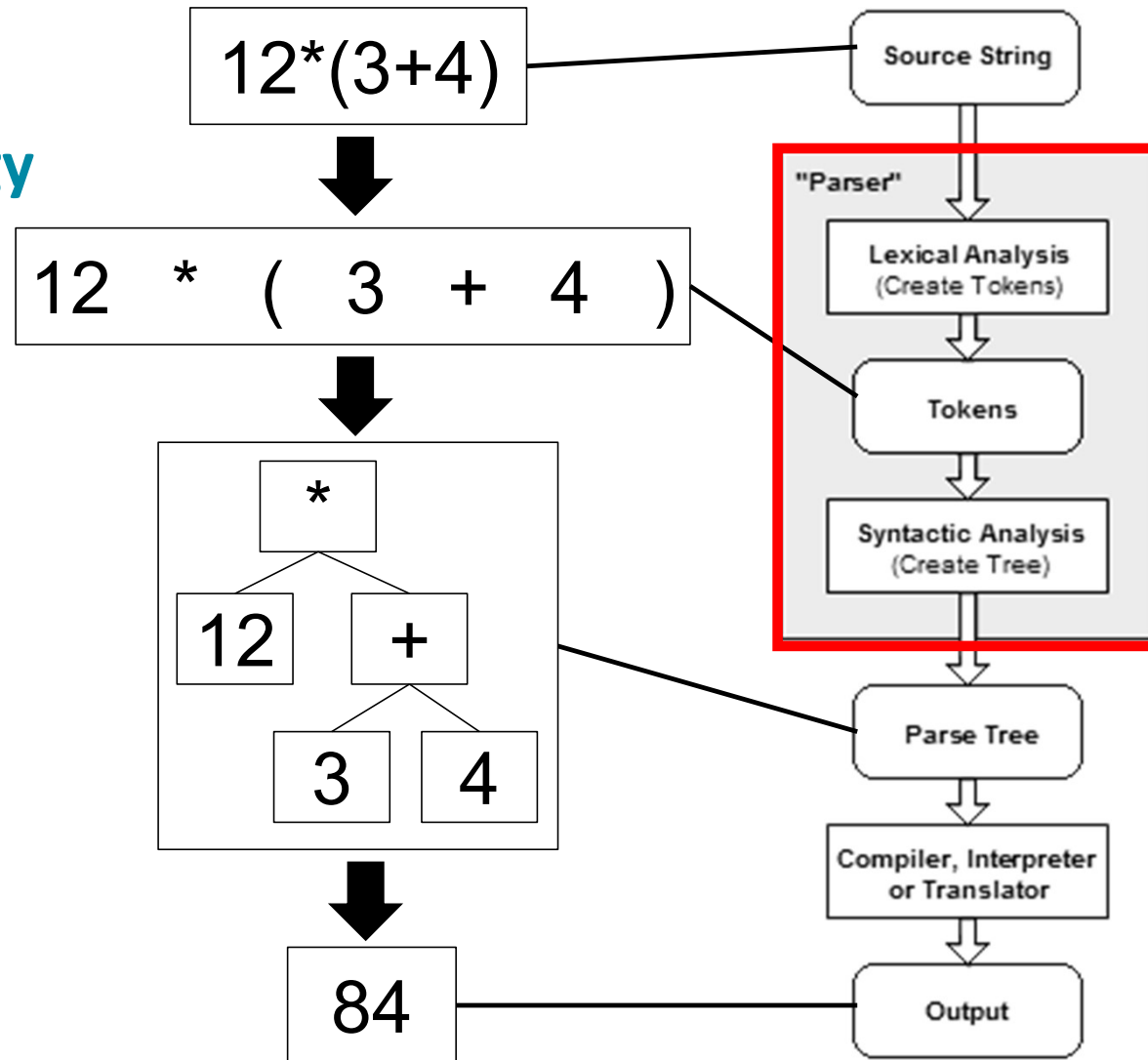
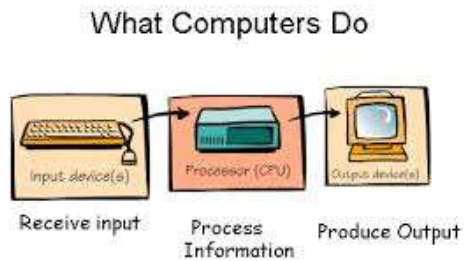




# Metamodels and grammars

## Domain-Specific Languages (DSL)

# Compiler Technology



# Formal Grammar

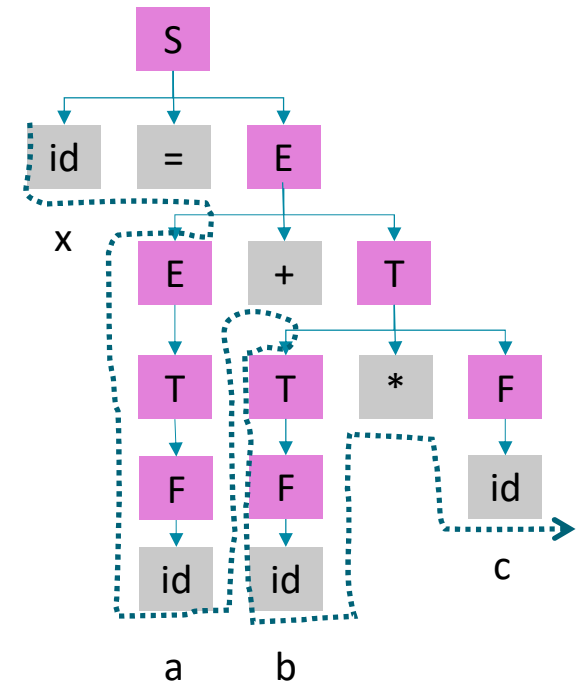
$$G = (N, T, P, S)$$

1	S	->	id = E
2	E	->	E + T   T
3	T	->	T * F   F
4	F	->	id

- N - Non terminals
- T - Terminals
- P - Productions
- S - Starting point

$$x = a + b * c$$

↑↑↑↑↑↑



$$7 = 1 + (3 * 2)$$

# Xtext / Xtend

## Extended Backus-Naur Form

The screenshot shows the Xtext website header with navigation links: Download, Documentation, Community, Support & Trainings, and Xtend. Below the header is the main heading "LANGUAGE ENGINEERING FOR EVERYONE!".

The main text on the website reads: "Eclipse Xtext™ is a framework for development of programming languages and domain-specific languages. With Xtext™ you define your language using a powerful grammar language. As a result you get a full infrastructure, including **parser**, **linker**, **typechecker**, **compiler** as well as editing support for **Eclipse**, **any editor that supports the Language Server Protocol** and your favorite **web browser**." Below this text is a link: [Learn more...](#)

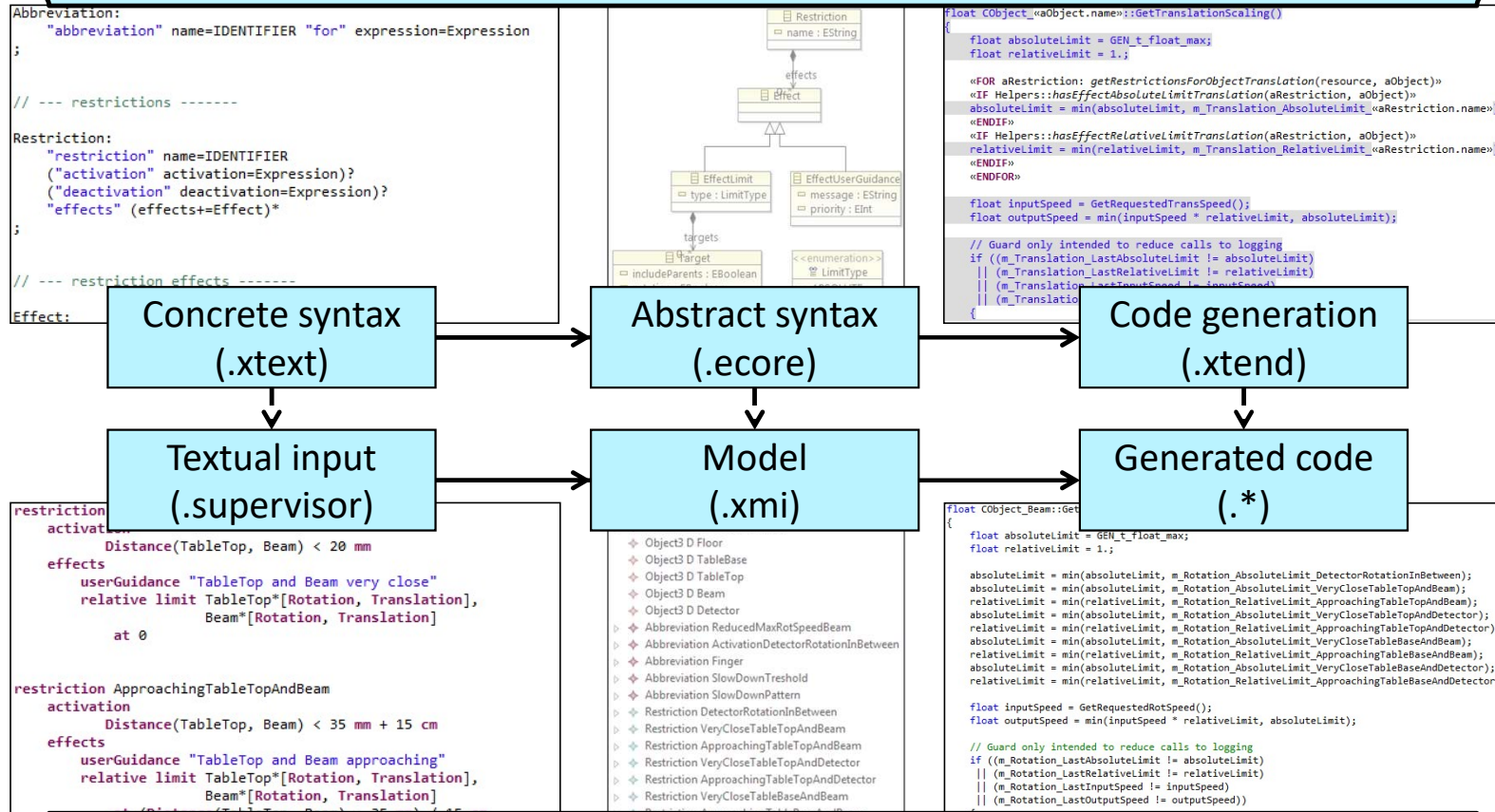
The bottom part of the screenshot shows the Eclipse IDE interface. The central editor displays a domain-specific language (DSL) for home automation rules, written in Xtend. The code includes:

```

1 Device Window can be OPEN, SHUT
2 Device Heating can be ON, OFF
3
4 Rule 'Close Window, when heating turned on'
5 when Heating_ON
6 then Window_SHUT
7
8 Rule 'Switch off heating, when windows gets opened'
9 when Window_OPEN
10 then Heating_OFF
    Heating_OFF
    Heating_ON
    Window_OPEN
    Window_SHUT
  
```

The IDE also shows a package explorer on the left with a project named 'my-home' containing 'Home.rules', 'JRE System Library', and 'src-gen'. A console window at the bottom indicates 'No consoles to display at this time.' The status bar at the bottom shows 'Writable', 'Insert', and '10 : 8'.

## Meta level, for developing the general infrastructure



## Instance level, for developing a specific system

# Lexical Analysis

	choice
?	optional
*	zero or more times
+	one or more times

- **Regular expressions detecting tokens (tokens == terminals)**

- Literal character sequences

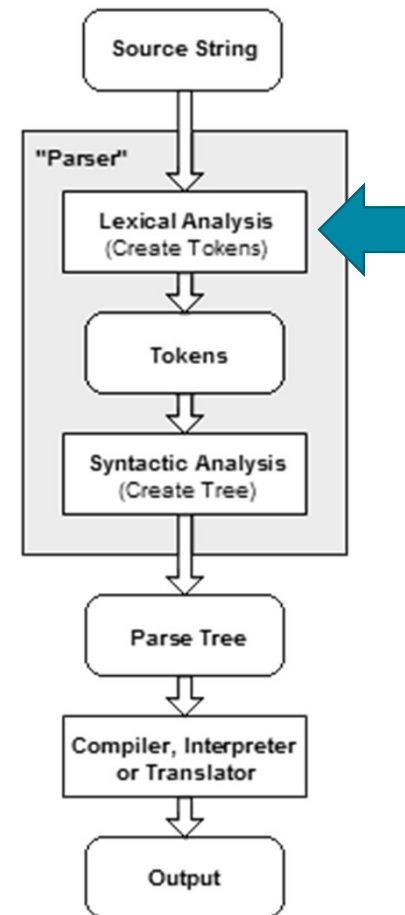
```
'while'
'('
```

- Custom terminal definitions

```
terminal INT: ('0'..'9')+;
terminal ID: '^?('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
```

- Terminals that are imported by default in Xtext:

- ID
- INT
- STRING
- ML\_COMMENT (= multi-line comment) ← Hidden by default
- SL\_COMMENT (= single-line comment) ← Hidden by default
- WS (= whitespace, tab, newline) ← Hidden by default



# Syntactic Analysis

	choice
?	optional
*	zero or more times
+	one or more times

- Context-free grammars using Extended Backus-Naur Form (EBNF)
  - More expressive than regular expressions, e.g., recursion to parse nested brackets

```

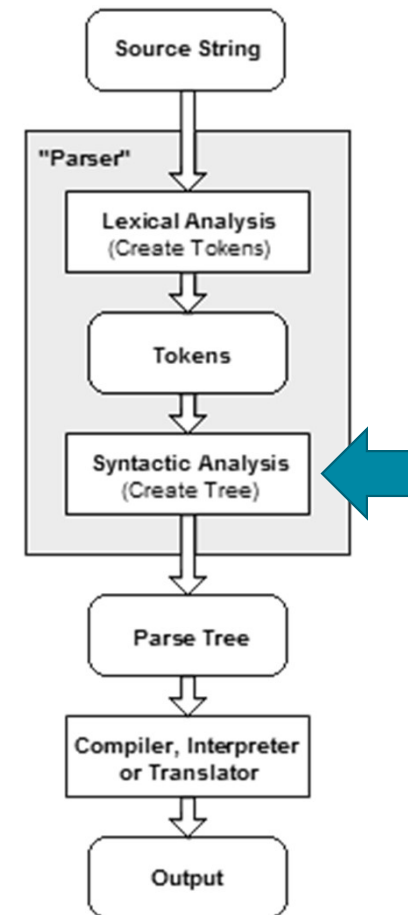
Task name AvoidBorder Task freq 100
Task moves
Move name Turn Start border End noBorder Speed 15 Rotation 90
Move name Back Speed 30
    
```

Robot: (Message | Task)\*

Message: 'Display' STRING

Task: 'Task' 'name' ID ('Task' 'freq' INT)?  
'Task' 'moves' (Move)+

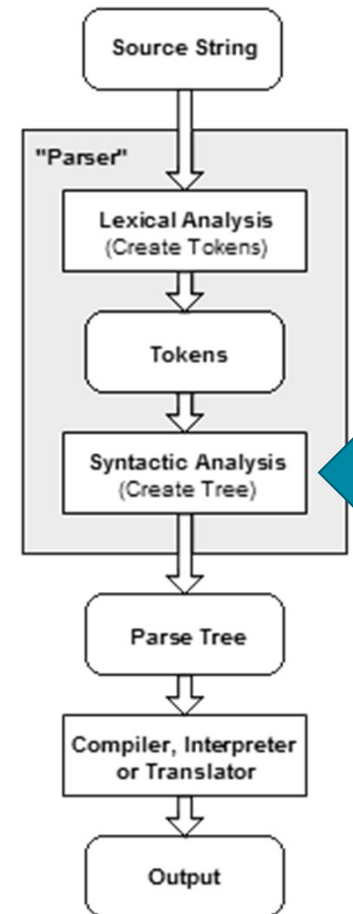
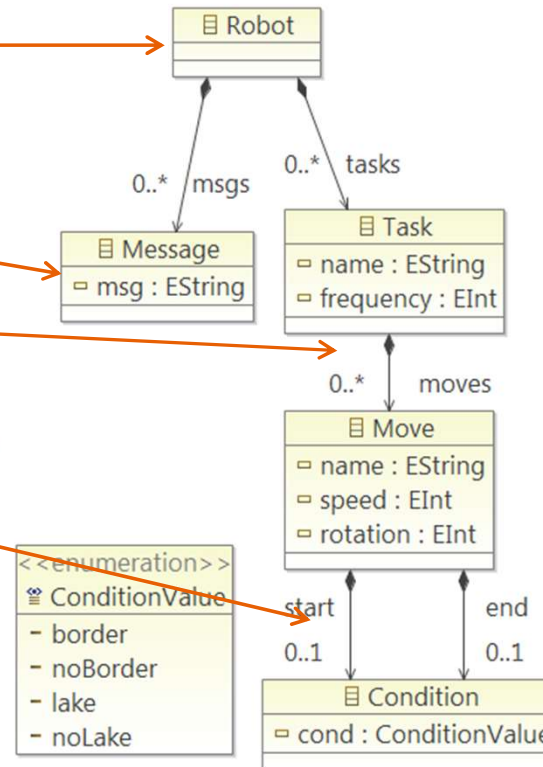
Move: 'Move' 'name' ID  
( 'Start' Condition)? ('End' Condition)?  
'Speed' INT ('Rotation' INT)?



# Abstract Syntax

```

Robot:
  (msgs += Message | tasks += Task)*
;
Message:
  'Display' msg = STRING
;
Task:
  'Task name' name = ID ('Task freq' frequency = INT)?
  'Task moves' (moves += Move)+
;
Move:
  'Move name' name = ID
  ('Start' start = Condition)? ('End' end = Condition)?
  'Speed' speed = INT ('Rotation' rotation = INT)?
;
Condition :
  cond = ConditionValue
;
enum ConditionValue:
  border | noBorder | lake | noLake
;
  
```





# Concrete and Abstract Syntax

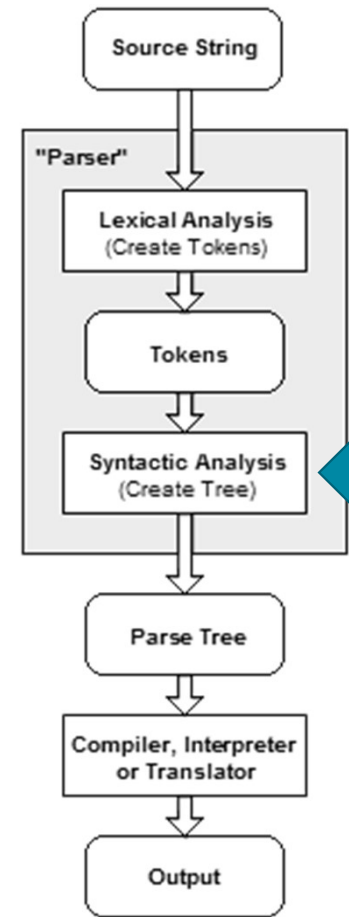
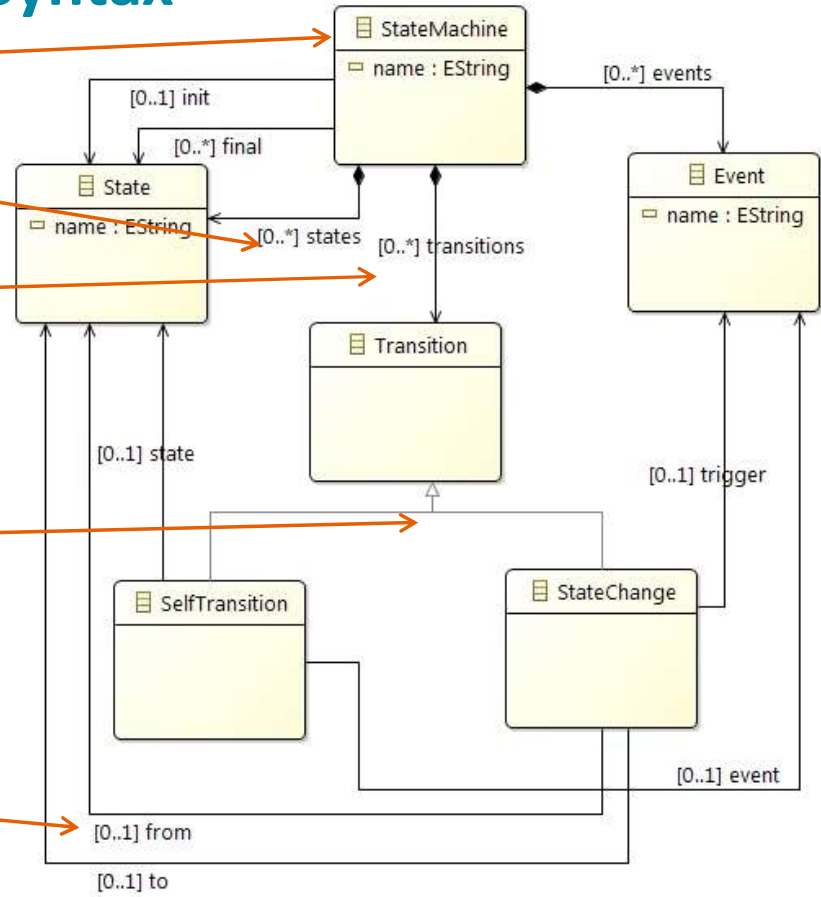
```

StateMachine:
  'StateMachine' name = ID
  'Init' init = [State]
  'States' states += State+
  'Events' events += Event+
  'Transitions'
    transitions += Transition*
  'Final' final += [State];

State: name=ID;
Event: name =ID;

Transition:
  SelfTransition | StateChange;

SelfTransition:
  'FromTo' state = [State]
  'Event' event = [Event]
;
StateChange:
  'From' from = [State]
  'Trigger' trigger = [Event]
  'To' to = [State];
  
```



## What is a valid state machine description according to this grammar?

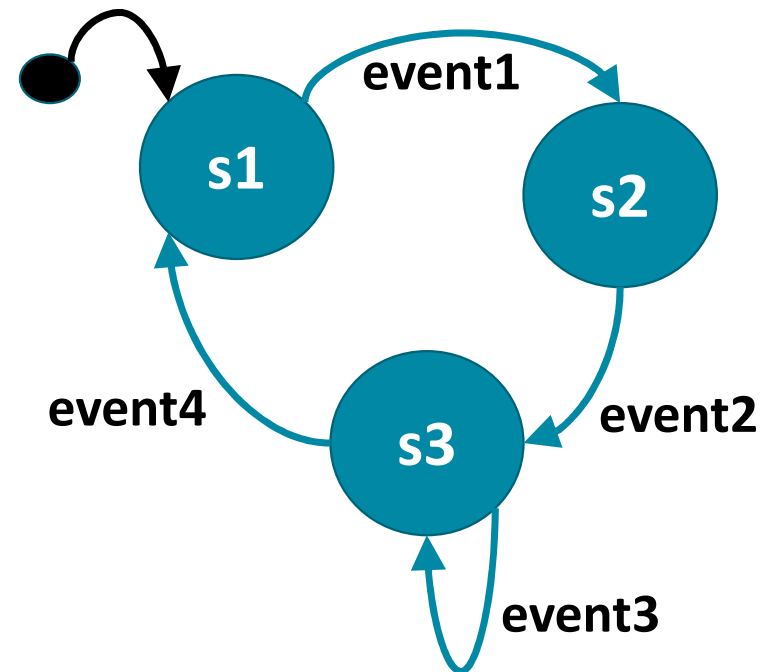
```
StateMachine:  
  'StateMachine' name = ID  
  'Init' init = [State]  
  'States' states += State+  
  'Events' events += Event+  
  'Transitions'  
    transitions += Transition*  
  'Final' final += [State];
```

```
State: name=ID;  
Event: name =ID;
```

```
Transition:  
  SelfTransition | StateChange;
```

```
SelfTransition:  
  'FromTo' state = [State]  
  'Event' event = [Event]  
  ;
```

```
StateChange:  
  'From' from = [State]  
  'Trigger' trigger = [Event]  
  'To' to = [State];
```



Think → Pair → Share

```

StateMachine:
  'StateMachine' name = ID
  'Init' init = [State]
  'States' states += State+
  'Events' events += Event+
  'Transitions'
    transitions += Transition*
  'Final' final += [State];

```

```

State: name=ID;
Event: name =ID;

```

```

Transition:
  SelfTransition | StateChange;

```

```

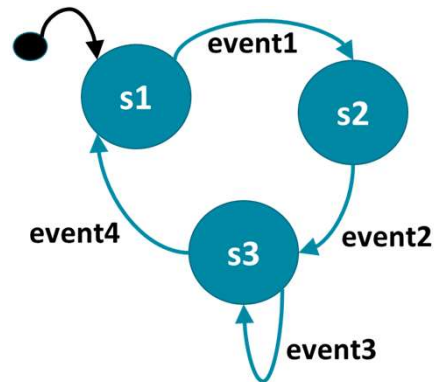
SelfTransition:
  'FromTo' state = [State]
  'Event' event = [Event]
;

```

```

StateChange:
  'From' from = [State]
  'Trigger' trigger = [Event]
  'To' to = [State];

```



### StateMachine machine

```

Init s1
States s1 s2 s3
Events event1 event2 event3 event4
Transitions
From s1
Trigger event1
To s2
From s1
Trigger event1
To s2
From s2
Trigger event2
To s3
From s3
Trigger event4
To s1
FromTo s3
Event event3
Final s1

```

15 minutes break

# Let's build a grammar example

## Domain-Specific Languages (DSL)

## General Tips and Tricks

- **It may help to first create an example instance, and afterwards create a grammar.**
  - “test-driven”
- **Look at the abstract syntax!**
  - E.g., check missing attribute names
- **Don't be too restrictive in the grammar; validation can be used for extra checks.**
- **Focus on specifying (not on executing)**
- **A DSL is not a general-purpose programming language**
- **Use enumeration types when appropriate:**

```
enum ChangeKind :  
    ADD      = 'add'  
    | MOVE   = 'move'  
;
```

## Write a grammar for the following DSL

**Planning** planningA

**Person** Mary

**Person** John

**Person** Pascal

**Task** task1: Mary Pascal

**Planning** planningB

**Person** John

**Task** task1: John

	choice
?	optional
*	zero or more times
+	one or more times

Robot:

Message:

Task:

(Message | Task)\*

'Display' STRING

'Task' 'name' ID ('Task' 'freq' INT)?

Think → Pair → Share

Planning:

```
"Planning" name=ID  
(persons += Person | tasks += Task)*  
;
```

Person:

```
"Person" name=ID  
;
```

Task:

```
"Task" name=ID ":" persons+=[Person]+  
;
```

**Planning** planningA

**Person** Mary

**Person** John

**Person** Pascal

**Task** task1: Mary Pascal

**Planning** planningB

**Person** John

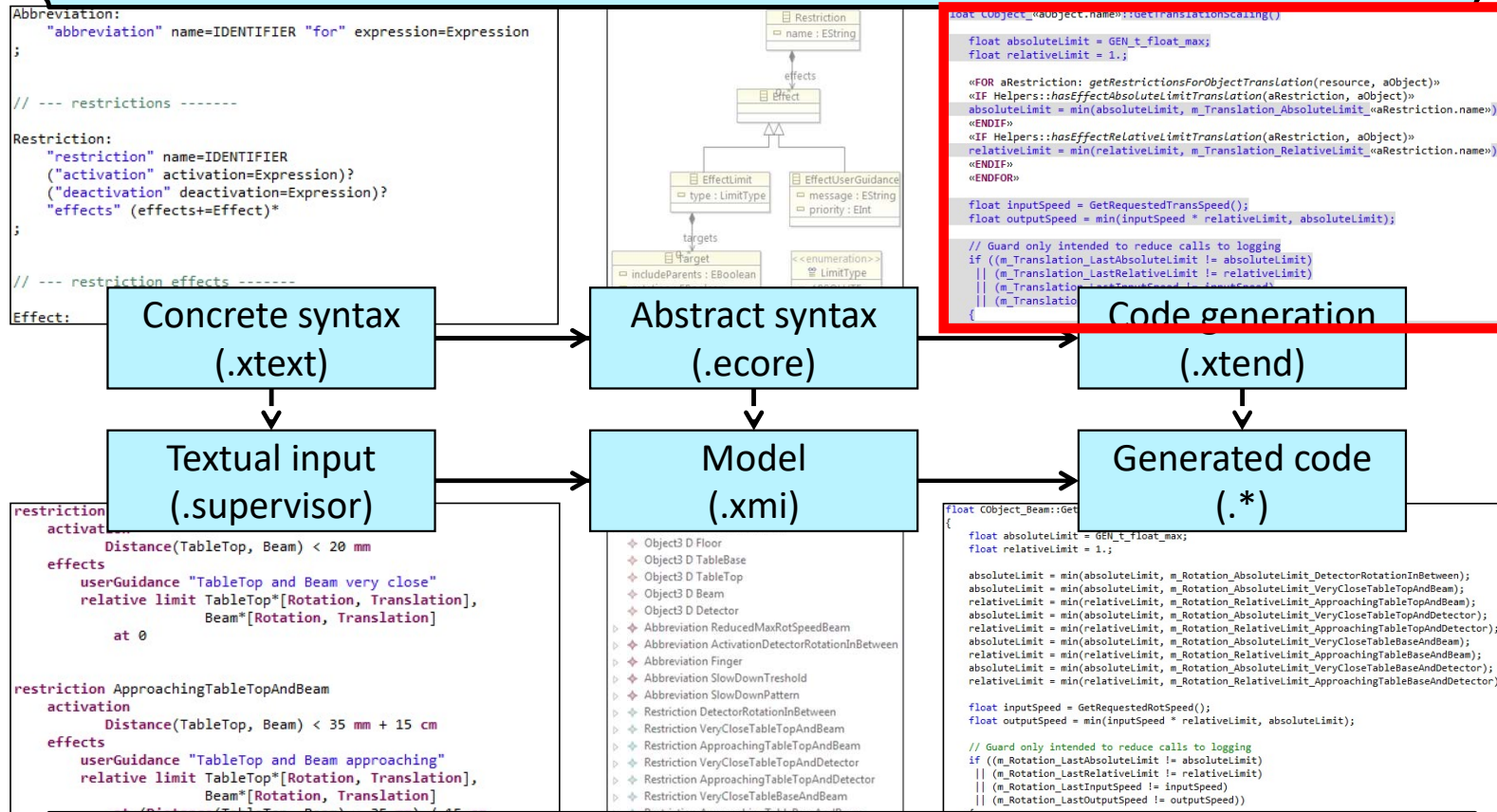
**Task** task1: John



# Editor support, validation and generators

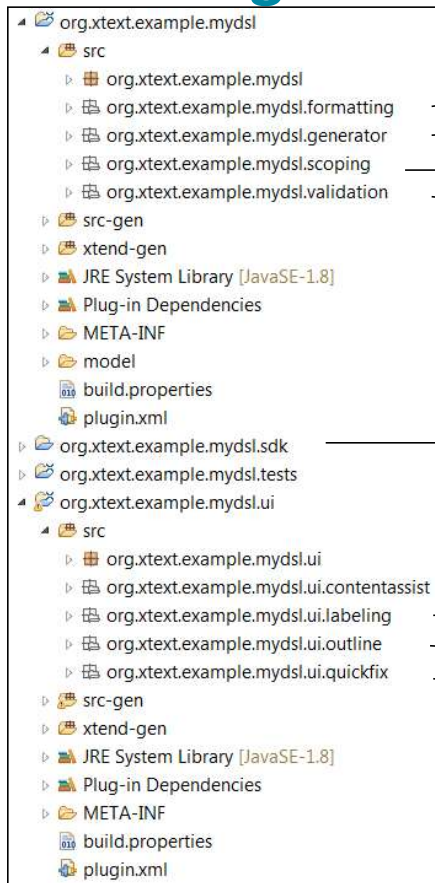
## Domain-Specific Languages (DSL)

## Meta level, for developing the general infrastructure



## Instance level, for developing a specific system

# Xtext Starting Points for Advanced Features



- Auto-formatting (<CTRL>-<SHIFT>-<F>)
- **Code generation**
- Scoping rules (for references)
- **Validation**
  
- Feature for creating a plugin
  
- Content assist (<CTRL>-<SPACE>)
- Labels for hovers
- Outline tree
- Quickfixes (for validation results)

# Model-to-Text Generation – Example Manual

Grammar

```

Planning: 'Planning' name = ID
        (persons += Person | tasks += Task)*
;
Person: 'Person:' name=ID
    
```

Generator

```

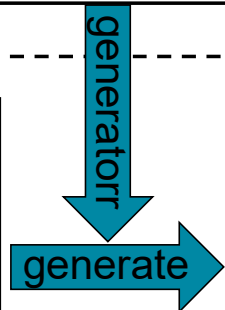
class TextGenerator {
    def static toText(Planning root)'''
        Info of the planning «root.name»
        All Persons:«"\n"»
        «FOR p : root.persons»«"\t"»«p.name»«"\n"»«ENDFOR»
        All actions of tasks:
        «FOR t : root.tasks BEFORE "=====" \n"
    '''
}
    
```

Meta-level  
workspace

```

Planning Managers
Person: Alice
Person: Bob
Person: Carol

Task: Meeting "strategy" persons: Alice priority: 4
Task: Lunch Canteen persons: Alice priority: 8 durat
    
```



```

Info of the planning Managers
All Persons:
    Alice
    Bob
    Carol
All actions:
    =====
    
```

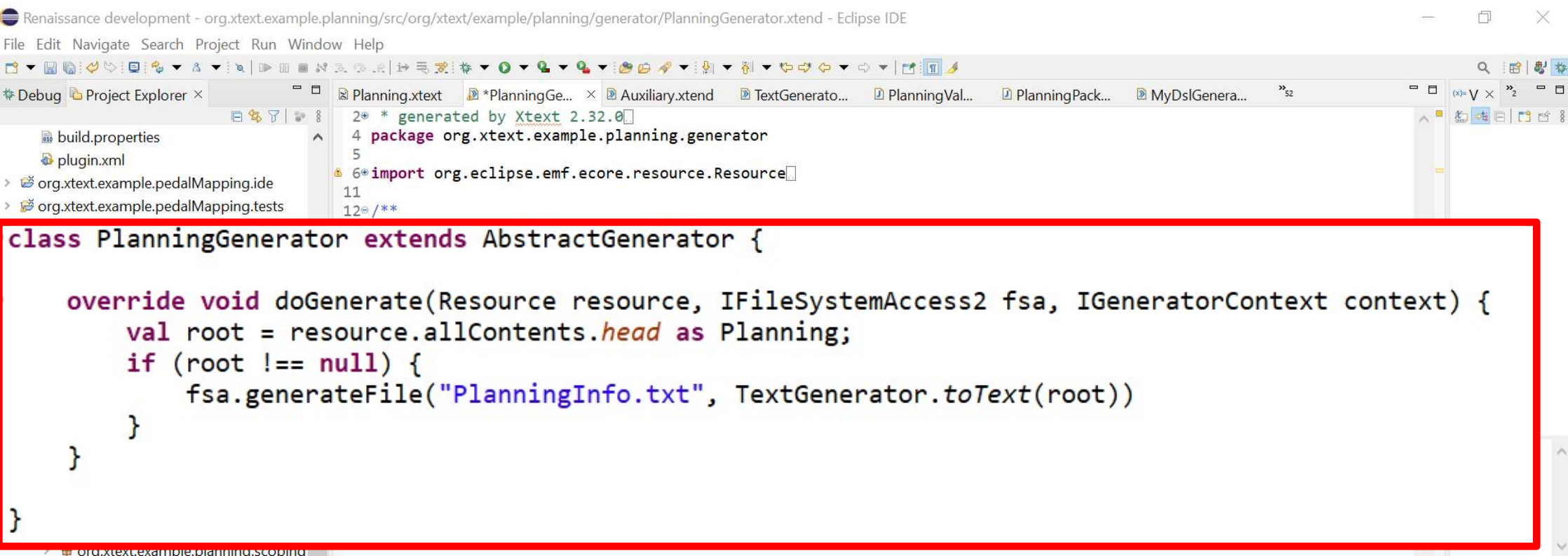
Runtime  
workspace

## Xtend: “JAVA with spice” ( <http://www.eclipse.org/xtend/> )

Flexible, expressive dialect of Java, which compiles into readable Java 8 compatible source code

- **Extension methods** - enhance closed types with new functionality
- **Lambda Expressions** - concise syntax for anonymous function literals
- **Operator overloading** - make your libraries even more expressive
- **Powerful switch expressions** - type-based switching with implicit casts
- **Multiple dispatch** - a.k.a. polymorphic method invocation
- **Template expressions** - with intelligent white space handling
- **No statements** - everything is an expression
- **Properties** - short-hands for accessing and defining getters and setters
- ...

## Starting Point for Code Generation



```
Renaissance development - org.xtext.example.planning/src/org.xtext.example.planning.generator/PlanningGenerator.xtend - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Debug Project Explorer ×
build.properties
plugin.xml
org.xtext.example.pedalMapping.ide
org.xtext.example.pedalMapping.tests
Planning.xtend *PlanningGe... Auxiliary.xtend TextGenerato... PlanningVal... PlanningPack... MyDslGenera...
2 * generated by Xtext 2.32.0
4 package org.xtext.example.planning.generator
5
6 import org.eclipse.emf.ecore.resource.Resource
11
12 /**
class PlanningGenerator extends AbstractGenerator {
    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        val root = resource.allContents.head as Planning;
        if (root != null) {
            fsa.generateFile("PlanningInfo.txt", TextGenerator.toText(root))
        }
    }
}
```

## Definition of Text Generator

```

11 class TextGenerator {
12     def static toText(Planning root) '''
13         Info of the planning «root.name»
14         All Persons:«"\n"»
15     ....
16     «FOR p : root.persons»«"\t"»«p.name»«"\n"»«ENDFOR»
17     All actions of tasks:
18     «FOR t : root.tasks BEFORE "=====\n" SEPARATOR " &" AFTER "====="»
19         «action2Text(t.action)»«infoAction(t)»
20     «ENDFOR»
21     ....
22     Other way of listing all tasks:
23     «FOR a : Auxiliary.getActions(root) SEPARATOR " , "»
24         «action2Text(a)»
25     «ENDFOR'''
26
27     def static dispatch action2Text(LunchAction action) '''
28         Lunch at location «action.location'''
29
30     def static dispatch action2Text(MeetingAction action) '''
31         Meeting with topic «action.topic'''
32

```

```

class Auxiliary {
    def static List<Action> getActions(Planning root) {
        var List<Action> actionlist = new ArrayList<Action>()
        for (Task t : root.tasks) {
            actionlist.add(t.action)
        }
        return actionlist;
        // return root.tasks.map[t|t.action]
    }
}

```

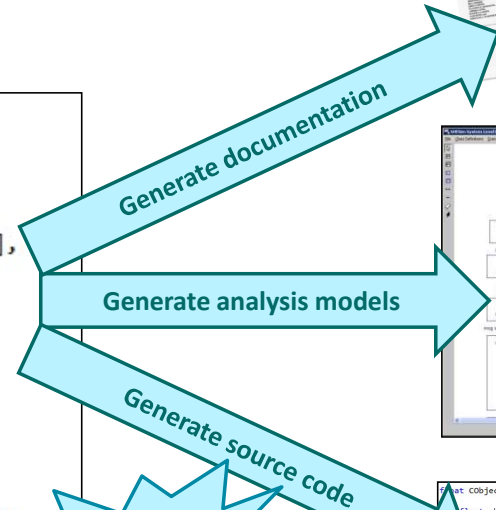
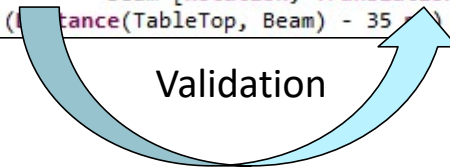


# DSL as Central Artifact

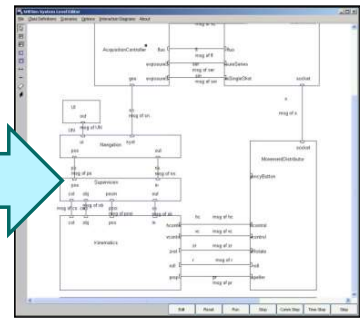
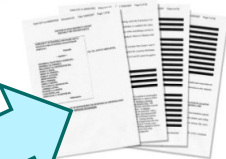
```

restriction VeryCloseTableTopAndBeam
  activation
    Distance(TableTop, Beam) < 20 mm
  effects
    userGuidance "TableTop and Beam very close"
    relative limit TableTop*[Rotation, Translation],
                  Beam*[Rotation, Translation]
    at 0

restriction ApproachingTableTopAndBeam
  activation
    Distance(TableTop, Beam) < 35 mm + 15 cm
  effects
    userGuidance "TableTop and Beam approaching"
    relative limit TableTop*[Rotation, Translation],
                  Beam*[Rotation, Translation]
    at (Distance(TableTop, Beam) - 35 mm) / 15
  
```



factor >30



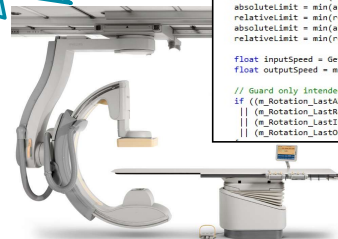
```

at CObject Beam: getRotationScaling()
float absoluteLimit = getFloat_max();
float relativeLimit = 1.;

absoluteLimit = min(absoluteLimit, m_Rotation.AbsoluteLimit_DetectorRotationInBetween);
relativeLimit = min(absoluteLimit, m_Rotation.AbsoluteLimit_VeryCloseTableTopAndBeam);
relativeLimit = min(relativeLimit, m_Rotation.RelativeLimit_ApproachingTableTopAndBeam);
absoluteLimit = min(absoluteLimit, m_Rotation.AbsoluteLimit_VeryCloseTableTopAndBeam);
relativeLimit = min(relativeLimit, m_Rotation.RelativeLimit_ApproachingTableTopAndBeam);
absoluteLimit = min(absoluteLimit, m_Rotation.AbsoluteLimit_VeryCloseTableBaseAndBeam);
relativeLimit = min(relativeLimit, m_Rotation.RelativeLimit_ApproachingTableBaseAndBeam);
absoluteLimit = min(absoluteLimit, m_Rotation.AbsoluteLimit_VeryCloseTableBaseAndBeam);
relativeLimit = min(relativeLimit, m_Rotation.RelativeLimit_ApproachingTableBaseAndBeam);

float inputSpeed = getRequestedRotSpeed();
float outputSpeed = min(inputSpeed * relativeLimit, absoluteLimit);

// Guard only intended to reduce calls to logging
if ((m_Rotation_LastAbsoluteLimit != absoluteLimit)
    || (m_Rotation_LastRelativeLimit != relativeLimit)
    || (m_Rotation_LastInputSpeed != inputSpeed)
    || (m_Rotation_LastOutputSpeed != outputSpeed))
  
```





# Consistency between Generated Artifacts

**Artifacts generated from a single model are consistent with model by construction**

- No more manual synchronization of code, diagrams, analysis models, and documentation!

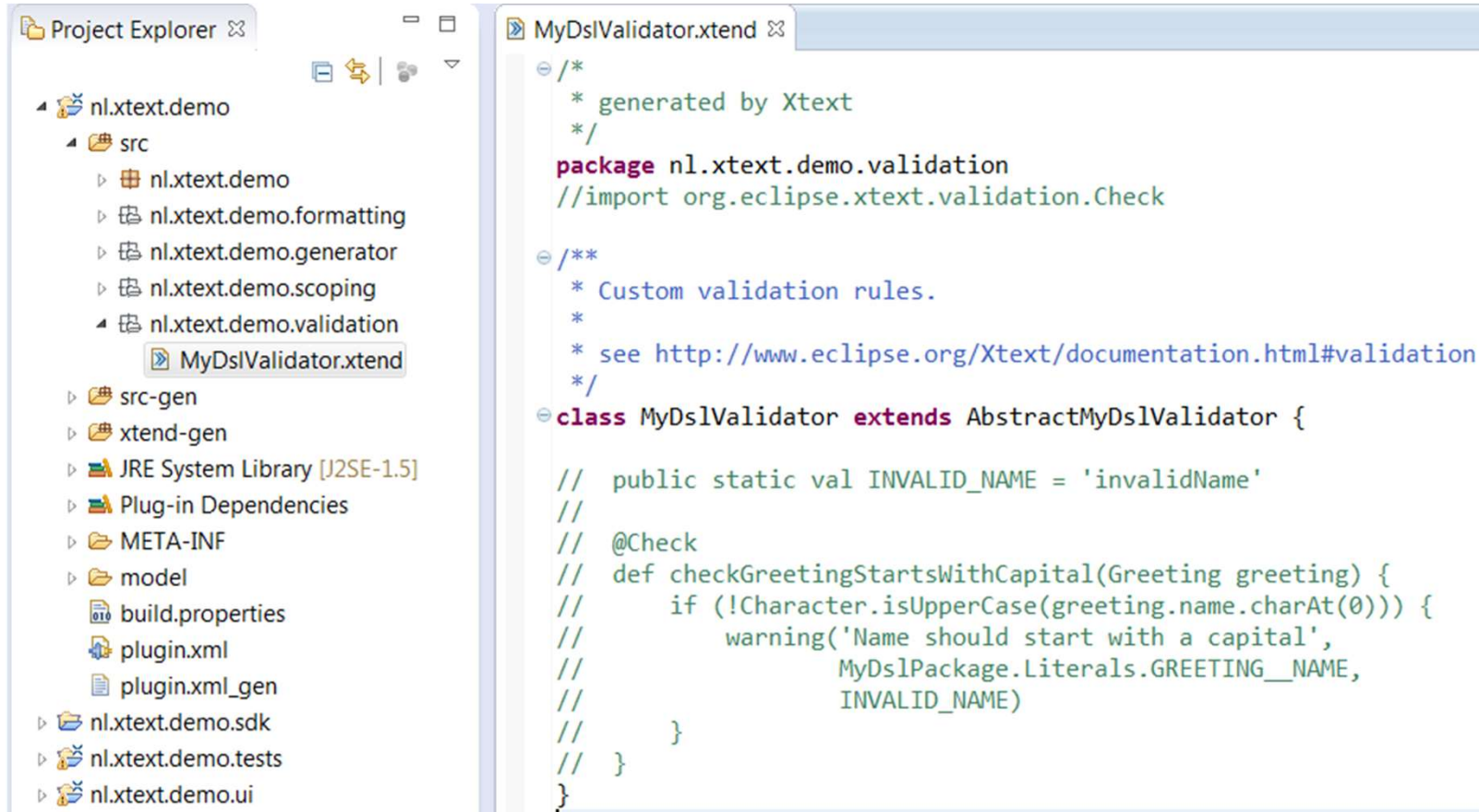
**Ensuring consistency between generated artifacts is very challenging**

- **Consistent implementation** of the DSL semantics required in all generators
- Artifacts may cover **different aspects** or are at **different level of abstraction**
- No single mitigation technique works for all artifacts

**Some approaches**

- |                                  |   |
|----------------------------------|---|
| 1. Formalizing the DSL semantics | independently of the details of any code generator  |
| 2. Model checking                | equivalence of generated artifacts and/or semantics |
| 3. Testing                       | equivalence of generated artifacts and/or semantics |

## Starting Point for Model Validation



```
Project Explorer
└─ nl.xtext.demo
   └─ src
      ├── nl.xtext.demo
      ├── nl.xtext.demo.formatting
      ├── nl.xtext.demo.generator
      ├── nl.xtext.demo.scoping
      └─ nl.xtext.demo.validation
         └─ MyDslValidator.xtend
      ├── src-gen
      ├── xtend-gen
      ├── JRE System Library [J2SE-1.5]
      ├── Plug-in Dependencies
      ├── META-INF
      └─ model
         ├── build.properties
         ├── plugin.xml
         └─ plugin.xml_gen
      ├── nl.xtext.demo.sdk
      ├── nl.xtext.demo.tests
      └─ nl.xtext.demo.ui

MyDslValidator.xtend
/*
 * generated by Xtext
 */
package nl.xtext.demo.validation
//import org.eclipse.xtext.validation.Check

/**
 * Custom validation rules.
 *
 * see http://www.eclipse.org/Xtext/documentation.html#validation
 */
class MyDslValidator extends AbstractMyDslValidator {

    // public static val INVALID_NAME = 'invalidName'
    //
    // @Check
    // def checkGreetingStartsWithCapital(Greeting greeting) {
    //     if (!Character.isUpperCase(greeting.name.charAt(0))) {
    //         warning('Name should start with a capital',
    //                 MyDslPackage.Literals.GREETING__NAME,
    //                 INVALID_NAME)
    //     }
    // }
}
```

## Validation: Example from Manual

```

@Check
def checkTimeUnit(Task task) {
  if (task.duration != null){
    switch(task.duration.unit){
      case TimeUnit::MINUTE: if (task.duration.dl > 1000)
        {warning("Rewrite to other unit",null)}
      case TimeUnit::HOURL: null
      case TimeUnit::DAY: if (task.duration.dl > 150)
        {info("Maybe rewrite to weeks",null)}
      case TimeUnit::WEEK: if (task.duration.dl > 52)
        {error("Deadline longer than 1 year not allowed",null)}
    }
  }
}

@Check
def checkDoublePersons(Task task){
  var plist = task.persons // lists start at position 0
  for (var i= 0; i < plist.size ; i++){
    for (var j = i+1; j < plist.size ; j++){
      if (plist.get(i).equals(plist.get(j))) {
        error("Double name",null)
      }
    }
  }
}

```

```

Task: Meeting "planning"persons: Carol priority: 5 duration: 1023 min
Task: Rewrite to other unitro persons: Alice priority: 1
Task: ns: Carol priority: 4 duration: 52 week
Task: Lunch Office persons: Alice Bob Carol priority: 6

```

```

Task: Lunch Canteen persons: Alice priority: 8 duration: 2 hour
i Maybe rewrite to weeksJournal persons: Bob priority: 2 duration: 153 day
Task: Meeting "planning"persons: Carol priority: 5 duration: 123 min

```

```

Task: Lunch Office persons: Carol priority: 4 duration: 53 week
Task: Deadline longer than 1 year not allowed Carol priority: 6
Task: ol priority: 3 duration: 1 hour
Task: priority: 4

```

# Validation Properties

## Basic

- Parsing
- Naming
- Referencing
- Type checking
- Structural
- Domain-specific

**(using Xtext/Xtend technologies, while editing)**

correct syntactic structure (keywords, grammar)

elements with unique names (usually per element type and scope)

references refer to elements that have been defined

expressions have a well-defined and correct type (and/or measurement unit)

no unused or unconnected elements, no cyclic dependencies between elements

e.g., length of messages

## Advanced

- Ranges
- Safety
- Deadlock

**(using external analysis tools, after editing)**

e.g., relative limits between 0 and 1, positive distances, no division by zero

e.g., low speeds if objects are close and approaching

object movements cannot be blocked completely

# Application areas

## Domain-Specific Languages (DSL)

## Where to Introduce DSLs?

*“The narrower the domain,  
the easier it becomes  
to build a good, high-level language  
and make generators produce first class code”*

[Tolvanen 2010]

**Restricted domain**      **with some variability**

### Focus on:

- Essential domain concepts      (INSTEAD OF      implementation details)
- Structured, reusable solution      (INSTEAD OF      ad-hoc implementation)

## Where to Introduce DSLs?

### Purpose of domain-specific modelling

**Abstraction:** from low-level implementations to readable requirements

- especially if the complexity is high

**Configuration:** from custom systems to reusable components with clear variability points

- especially if the product family is large

**Independence:** from platform-dependent software to technology-independent solutions

- especially if framework changes are expected

**Understanding:** from descriptions in natural text to formal, well-defined terminology

- especially if the main concepts and their relations are unclear

# Closing remarks

## Domain-Specific Languages (DSL)



# Objectives

## At the end of the course, you should be able to:

- Explain the purpose of Domain-Specific Languages, including several application areas
- Explain the basics of grammars and parsing
- Create basic textual Domain-Specific Languages, including editor support, validation and generators

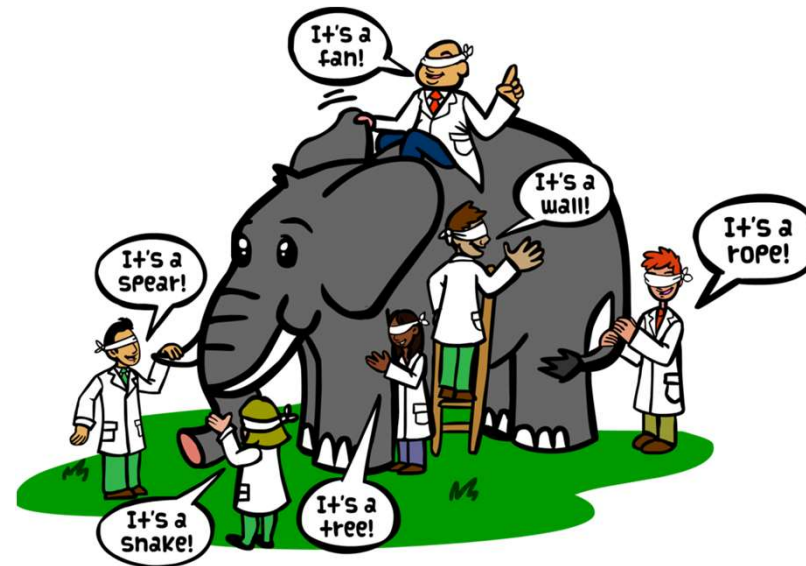
## Assessment:

- Modeling assignment using Domain-Specific Languages (in groups of 2 students)
- Reflection document on Model-Based Development (individual)

# Closing remarks

## Model-Based Development

## Modeling for a specific purpose



- In this course we have focused on the following 3 modeling techniques:
  - Unified Modeling Language (UML)
  - Finite-State Machines (FSM)
  - Domain-Specific Languages (DSL)

# Model-Based Development

- **Models-based development uses all four techniques for dealing with complexity:**
  - Abstraction: Identify high-level concepts that hide low-level details
  - Boundedness: Impose acceptable restrictions on the considered problem space
  - Composition: Divide one problem into multiple independent smaller problems
  - Duplication: Use multiple overlapping approaches for the same problem
- **General modeling goals:**
  - Speeding up software development of large complex systems
    - Human understanding
    - Early validation
    - Code generation
    - Automated testing
  - Bridging the gap between application domain expertise and technical system realization
- **Notes:**
  - Modeling is for a specific purpose; there exist many different types of models
  - Modeling often helps you to detect important unclarities

# Objectives

## At the end of the course, you should be able to:

- Explain some complexity challenges of software-intensive high-tech systems
- Explain the 3 modelling techniques UML – FSM and DSL
- Explain the purpose of Model-Based Development
- Compare Model-Based Development with other techniques you know

## Assessment:

- Modeling assignments for 3 modeling techniques (in groups of 2 students)
- Reflection document on Model-Based Development (individual)

# Reflection document

## Contents:

- **Formulate your informed view on Model-Based Development for Software Systems**
- **Motivate this view based on your experiences in this course**
  - (Optional) You may relate it to other (properly-referenced) experience/information sources
  - (Optional) You may relate it to your prior software development experiences

## Grading criteria:

- **Showing understanding of model-based development for software systems**
- **Providing an overarching view with supporting arguments (including your experiences in this course)**
- **Referencing all used sources (facts, experiences, etc.) in an appropriate way**

## Note:

- **Individual assignment, to be submitted as PDF**
- **Length: 1 – 2 pages A4 (= 500-1000 words)**

See you at the lab 😊