

# Automated Software Engineering: Introduction

*Oszkár Semeráth, Kristóf Marussy,  
Attila Ficsor, Ármin Zavada*



**Critical Systems  
Research Group**

Welcome everyone to the specialization!

# Administration



# Teachers and communication channels

## Teachers:

- Department of Artificial Intelligence and Systems Engineering
- Critical Systems Research Group - <https://ftsrg.mit.bme.hu/en/>

## Communication:

- Teams
- Moodle + Github
- Email (semerath@mit.bme.hu)



Oszkár Semeráth  
(lead lecturer)



Attila Ficsor



Kristóf Marussy



Ármin Zavada

# Schedule

- Each week:
  - one **lecture**
  - one **practice / laboratory** session (lab is skipped in first week)
- Laboratory is **mandatory**, there will be an attendance check.  
Max. 3 lab sessions can be missed
- **Homework**
  - Assignment: 2 stages, published on week #5 and #8
  - Deadline: Week #13 end of Friday (midnight)
  - Homework retake: Retake week end of Friday (midnight)
- **Exam** during the exam period

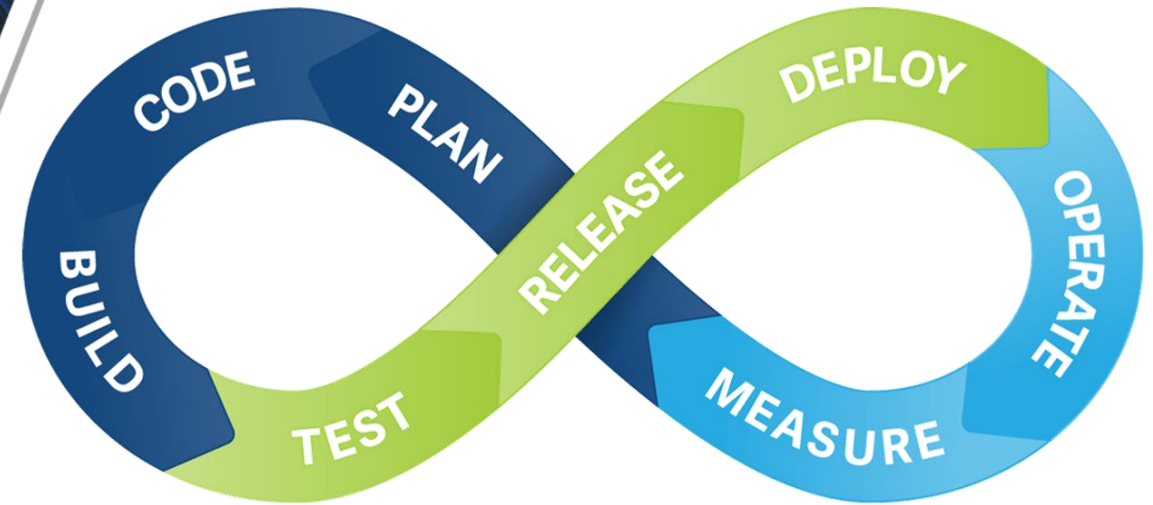
# Homework

- 3-person teams formed using a Moodle survey
- We ask everyone to create a Github account in advance!
- Topics: development / automation tasks connected to the laboratory materials
- Some degree of documentation is also required  
*(code comments, few paragraphs of markdown)*
- Different task are deliberately not interdependent  
→ Laboratory materials will have a more coherent 'story'
- You may improve your solution during the retake period if you have submitted until the regular deadline and received feedback

# Requirements

- Lab (practice) session attendance (3 can be missed)
- Accepted homework  
*(must obtain 40% of the possible score in both stages, constitutes 30% of the total score)*
- Exam  
*(must obtain 40% of the possible score, constitutes 70% of the total score)*
- Final mark is as usual (85%, 70%, 55%, 40%)

# Course overview





# Goals

**The aim** of the course is to introduce

**1. state-of-the-art**

**2. automated**

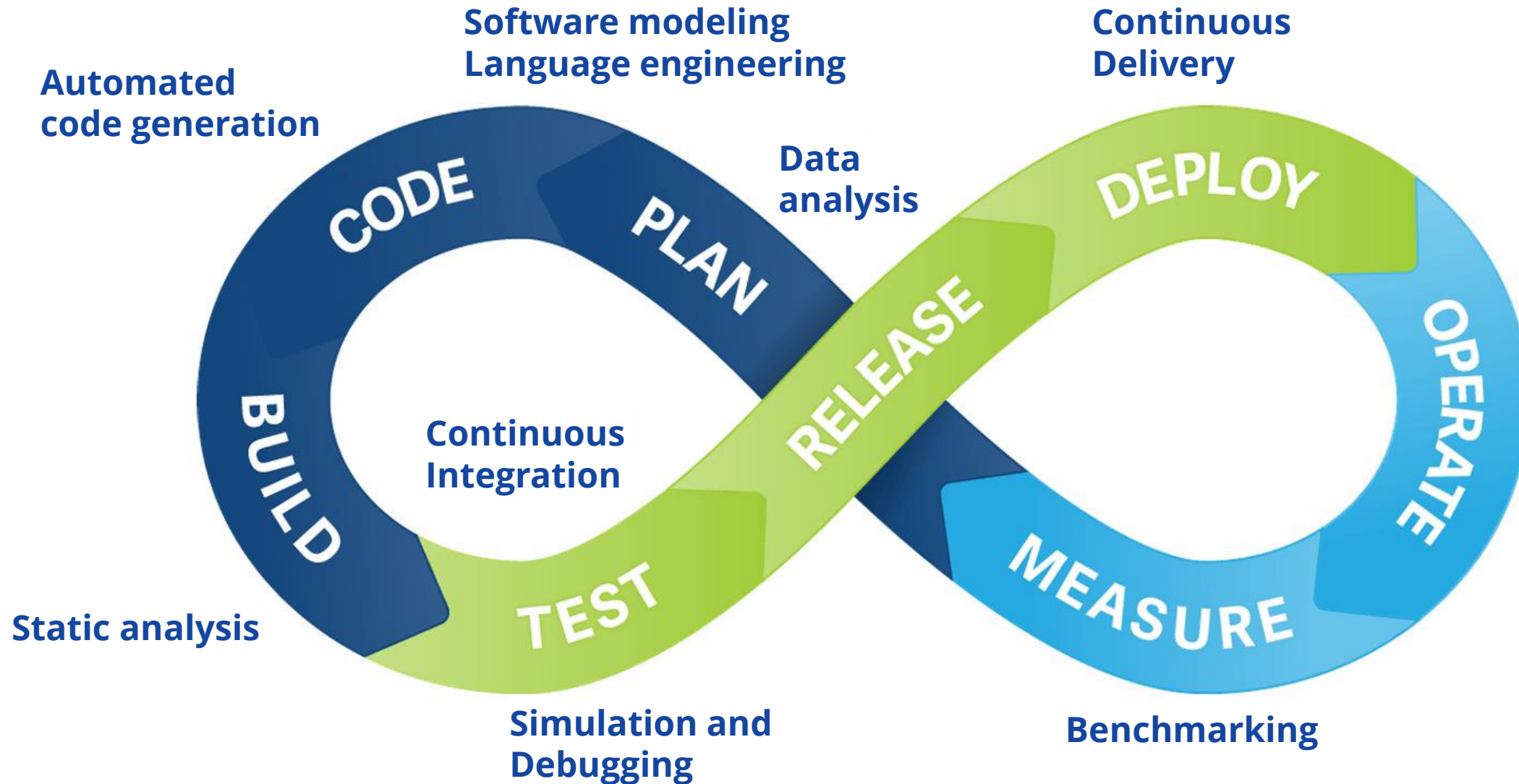
**3. technologies** in the field of software engineering.

## Outcomes:

Improved software **quality** → guarantees (formal methods)

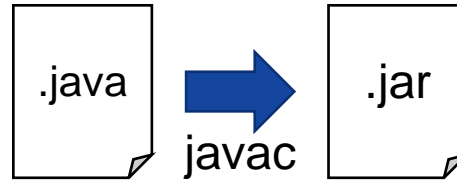
Higher **productivity** → automations (code generation)

# Areas of automation



# Development processes

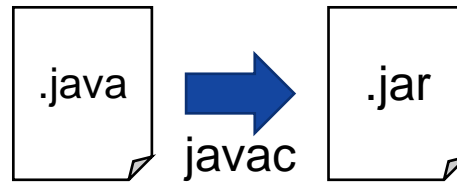
- What are the typical elements of software development?
  1. Source files
  2. Compilers
  3. Derived artifacts



# Development processes

- What are the typical elements of software development?

1. Source files
2. Compilers
3. Derived artifacts



Domain-specific models:  
Models and modelling languages  
created for solving a specific  
problem in a specific field

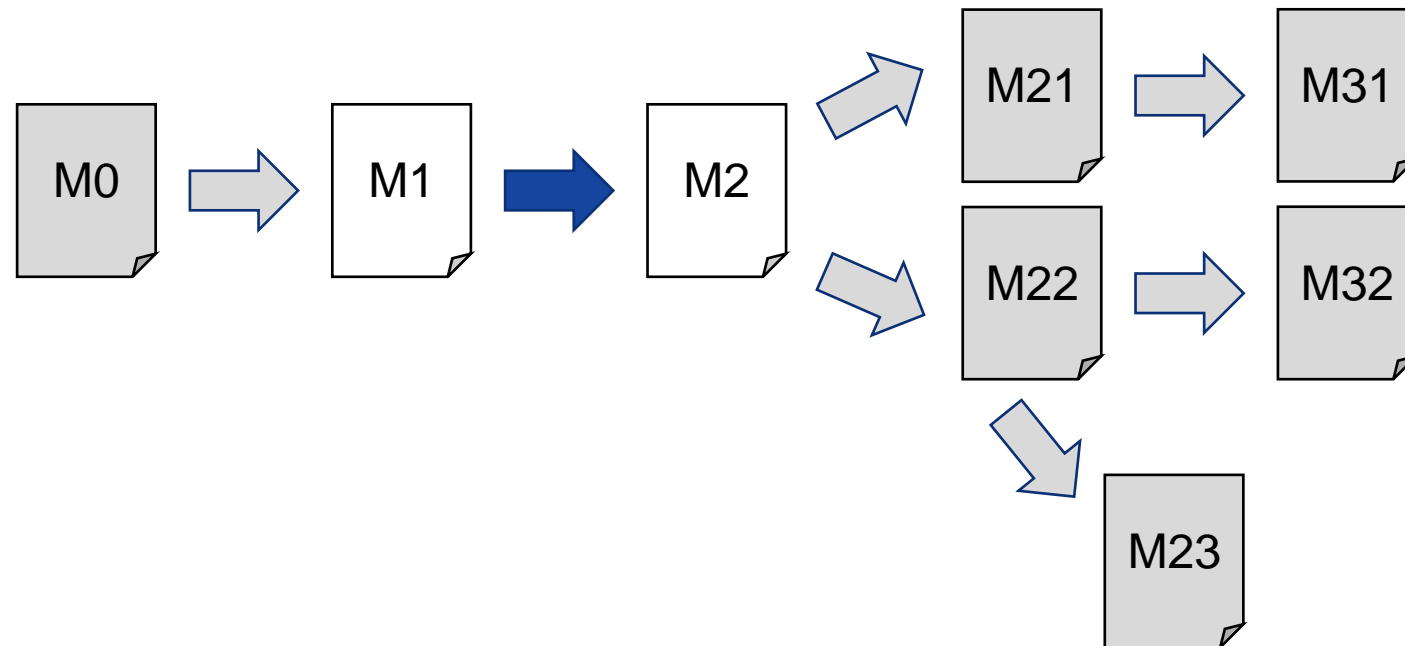
- More generally:

- Models = { source files, documentation, configuration, domain-specific models }
- Transformations : Models → Models

**Development = Models + Transformations**

# Development processes

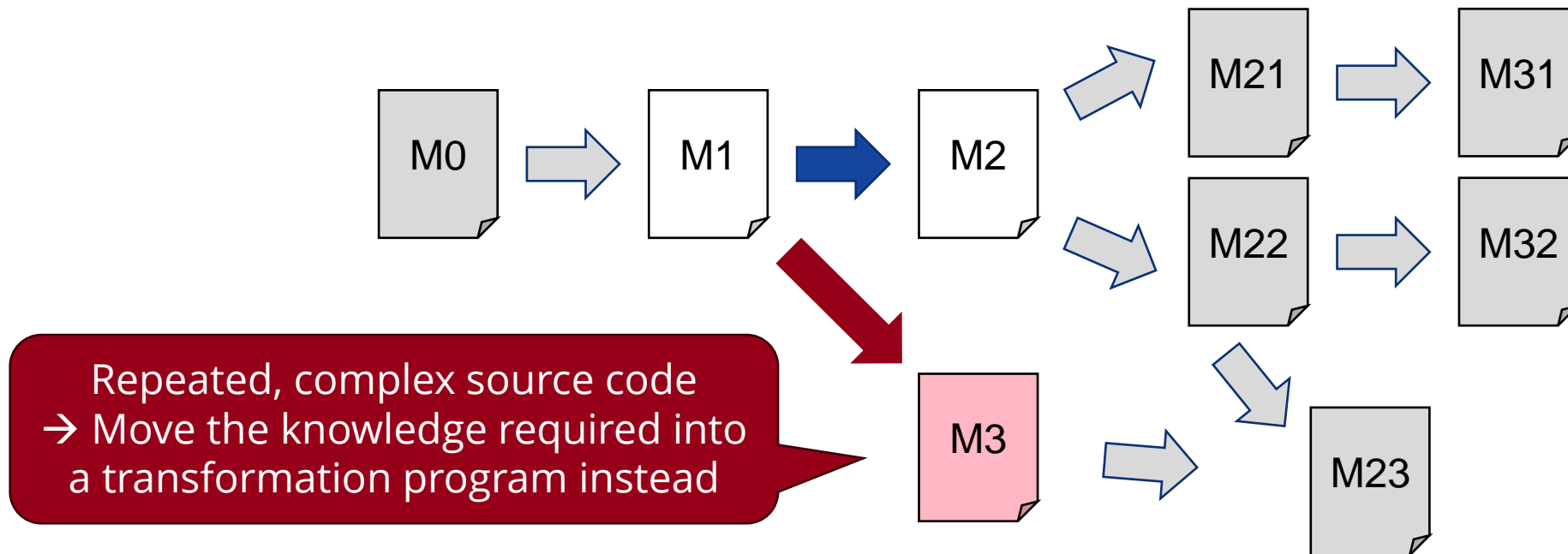
- We create models and automate their processing
- Software is constructed by a chain of transformations





# Development processes

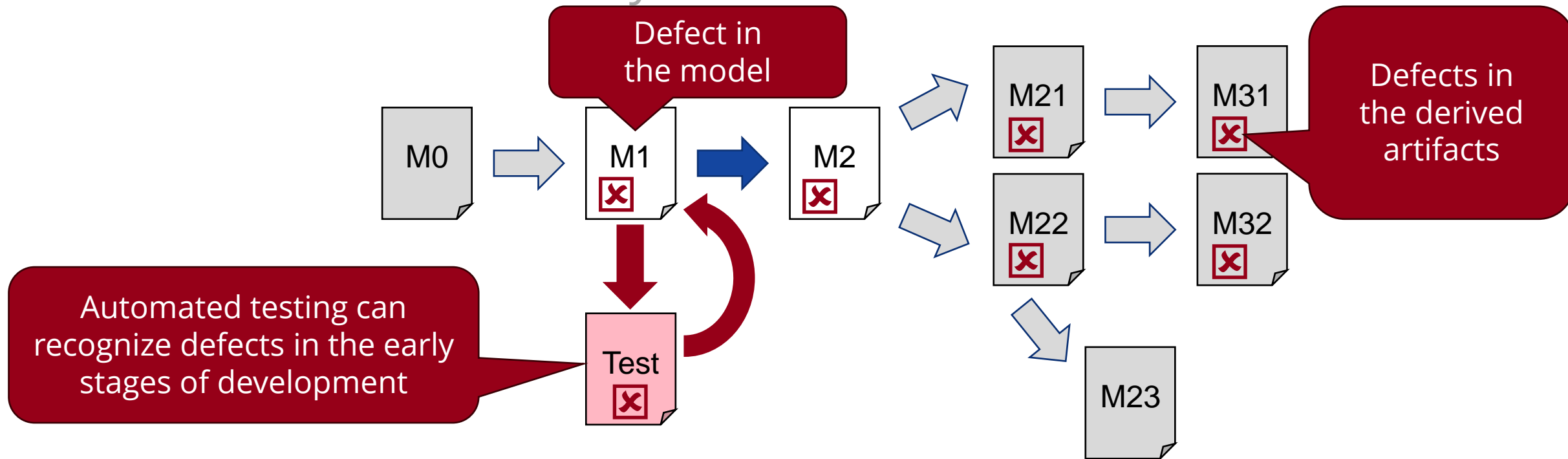
- We create models and automate their processing
- Software is constructed by a chain of transformations



- **Goal 1:** Automation of development tasks with transformations

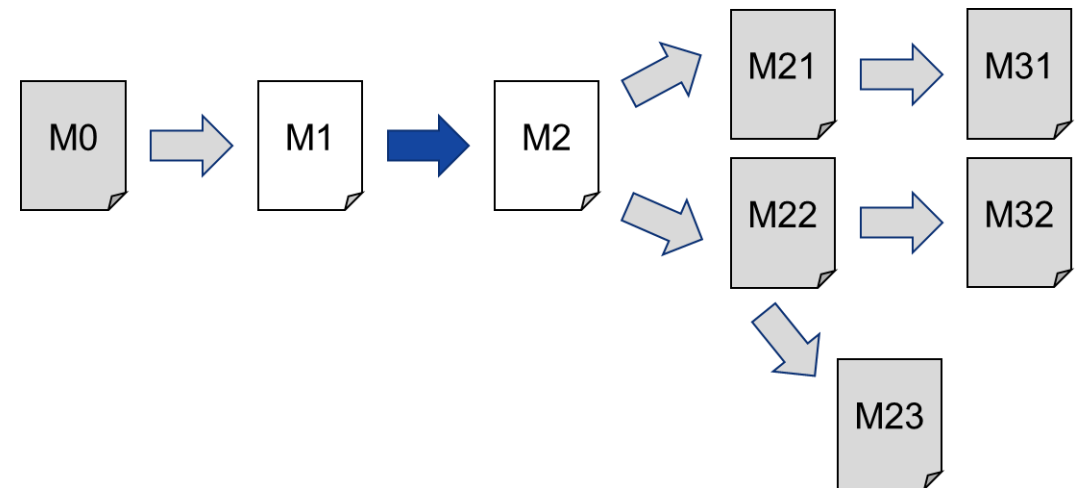
# Development processes

- We create models and automate their processing
- Software is constructed by a chain of transformations



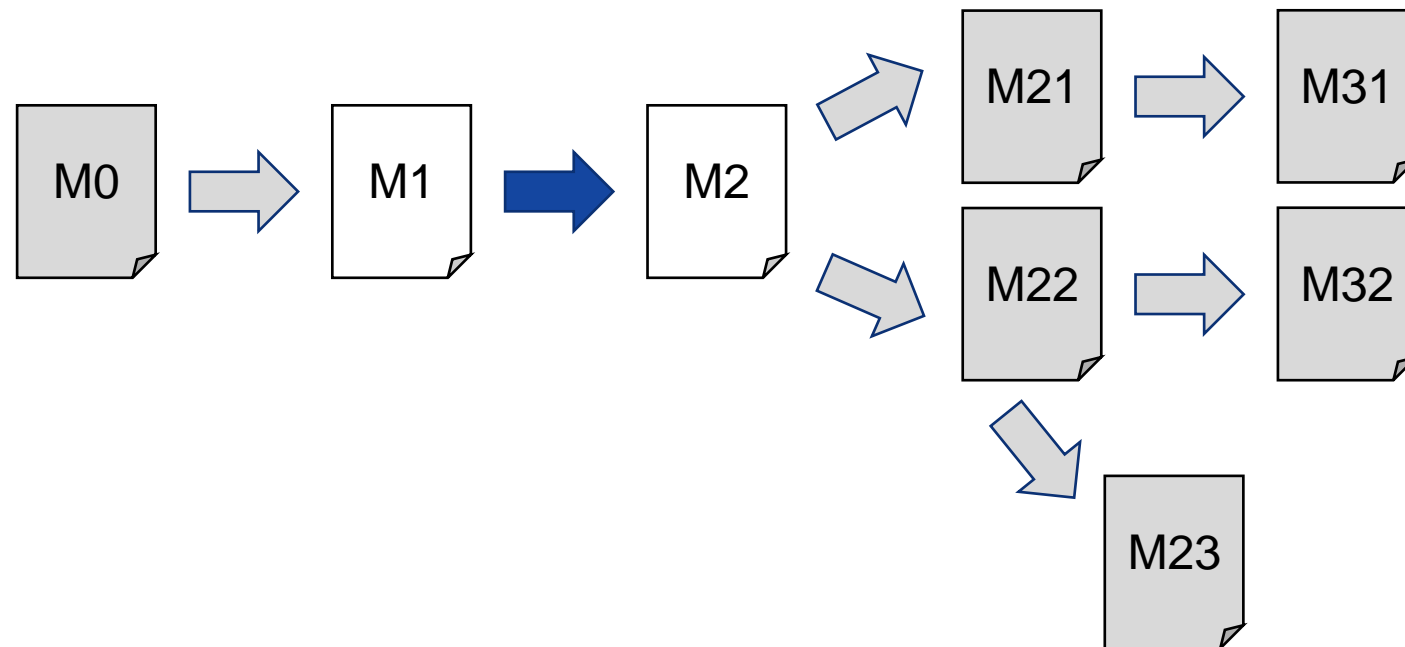
- **Goal 2:** Continuous correctness / performance evaluation

# Classification of transformations

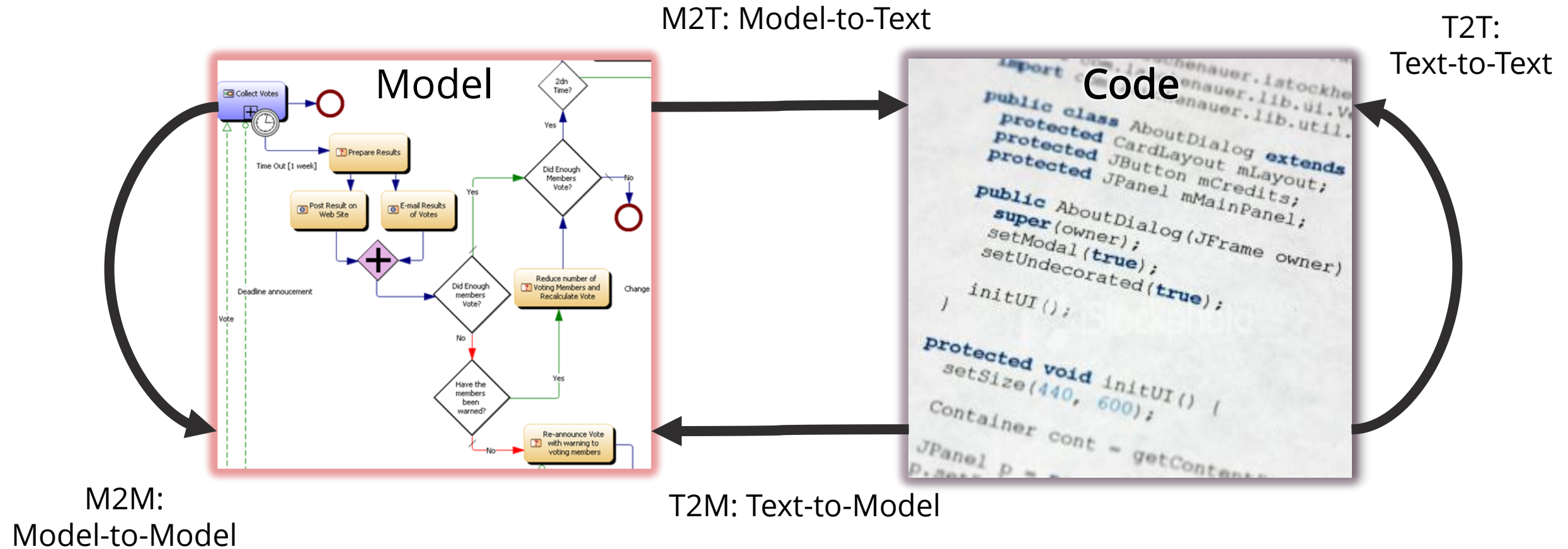


# Development processes

- We create models and automate their processing
- Software is constructed by a chain of transformations



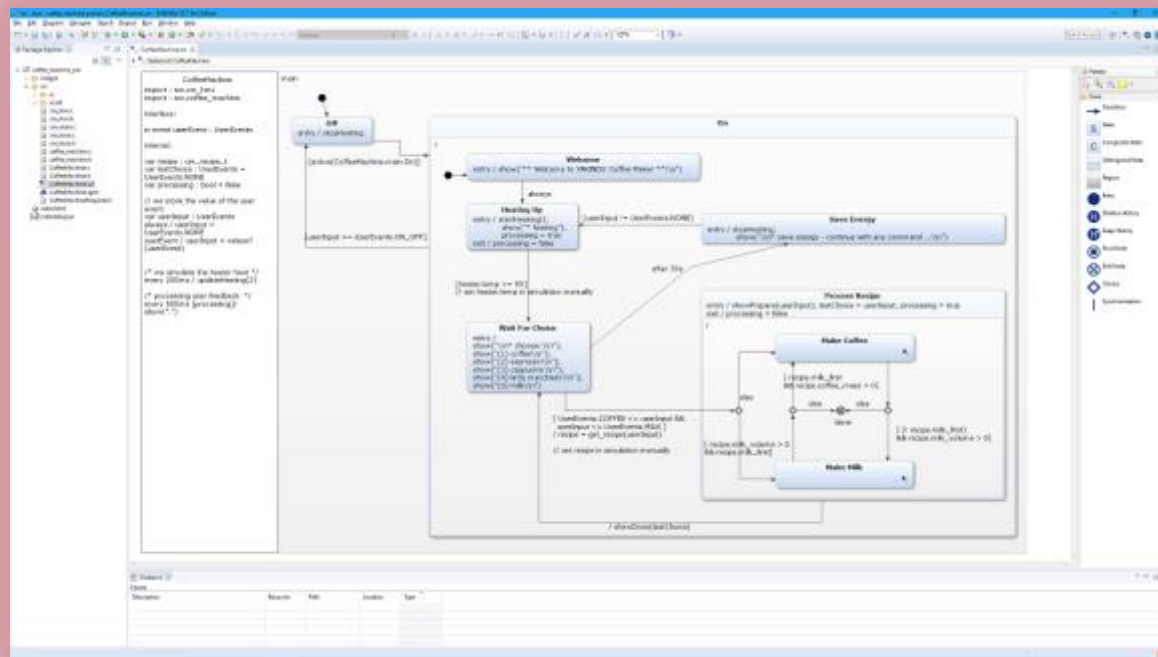
# Classification of transformations





# M2T example

- Code generation

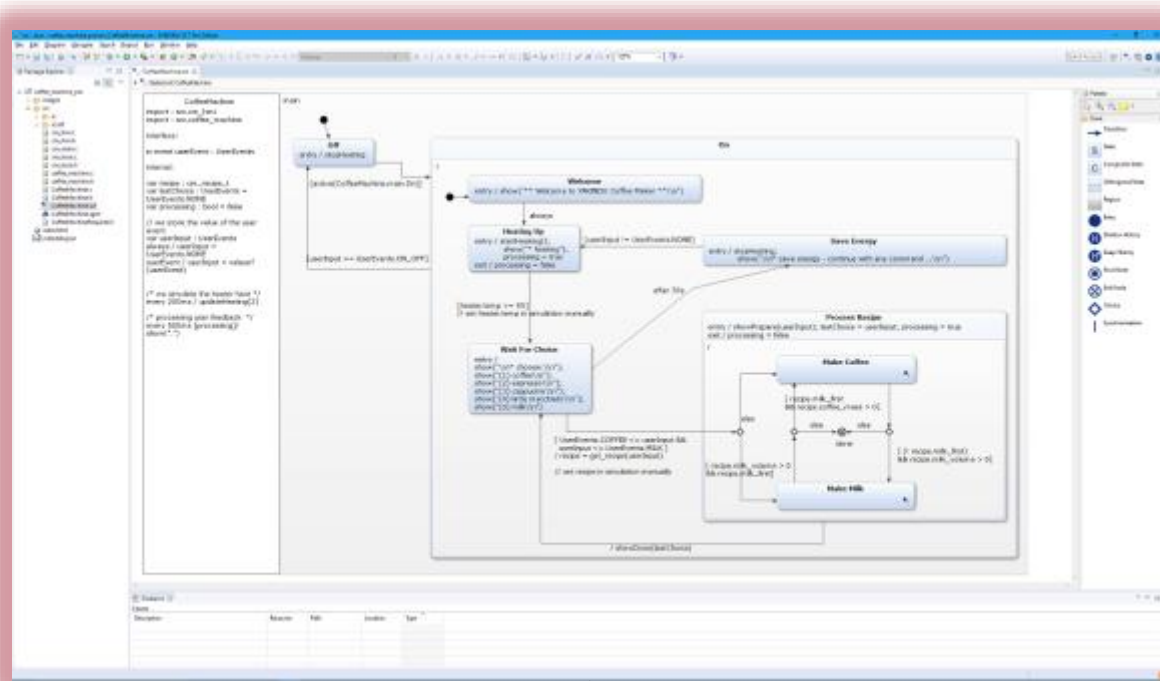


```
#ifndef DEFAULTSM_H_
#define DEFAULTSM_H_
#include "sc_types.h"
#include "StateMachineInterface.h"

class DefaultSM : public StateMachineInterface
{
public:
    DefaultSM();
    ~DefaultSM();
    /*! Enumeration of all states */
    typedef enum
    {
        main_region.MyState,
        DefaultSM_last_state
    } DefaultSMStates;
    /*! Inner class for Sample interface scope.
    class SCI_Sample
    {
    public:
        /*! Gets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
        sc_boolean get_a();
        /*! Sets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
        void set_a(sc_boolean value);
        /*! Raises the in event 'evA' that is defined in the interface scope 'Sample'. */
        void raise_evA(sc_boolean value);
        /*! Checks if the out event 'evB' that is defined in the interface scope 'Sample' has been raised. */
        sc_boolean isRaised_evB();
        /*! Gets the value of the out event 'evB' that is defined in the interface scope 'Sample'. */
        sc_integer get_evB_value();
    private:
        friend class DefaultSM;
        sc_boolean a;
        sc_boolean evA_raised;
        sc_boolean evA_value;
        sc_boolean evB_raised;
        sc_integer evB_value;
    };
    /*! Returns an instance of the interface class 'SCI_Sample'. */
    SCI_Sample* getSCI_Sample();
    void init();
    void enter();
    void exit();
    void runCycle();
    sc_boolean isActive();
    sc_boolean isFinal();
    sc_boolean isStateActive(DefaultSMStates state);
private:
    static const sc_integer maxOrthogonalStates = 1;
    DefaultSMStates stateConfVector[maxOrthogonalStates];
    sc_ushort stateConfVectorPosition;
};
```

# T2M example

- Import source code into a model



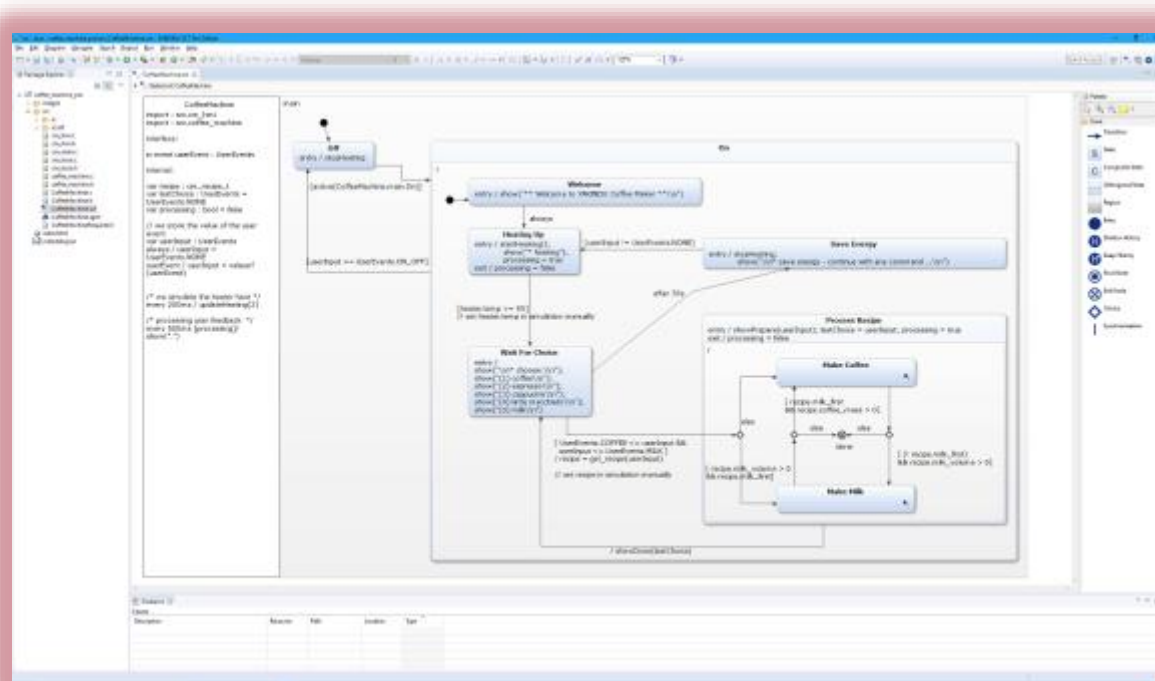
```
class Point
{
    public:
        int32_t get_x();
        void set_x(int32_t x);
        int32_t get_y();
        void set_y(int32_t y);
    private:
        int32_t x;
        int32_t y;
};
```



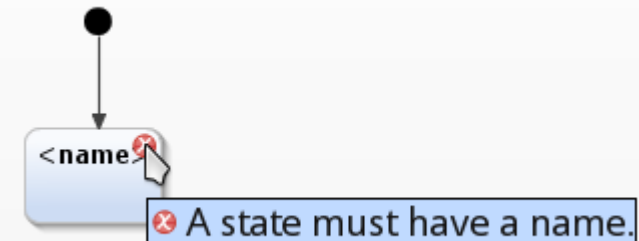
YAKINDU STATECHART TOOLS

# M2M example 1

- M2M: Model validation: error pattern → error messages

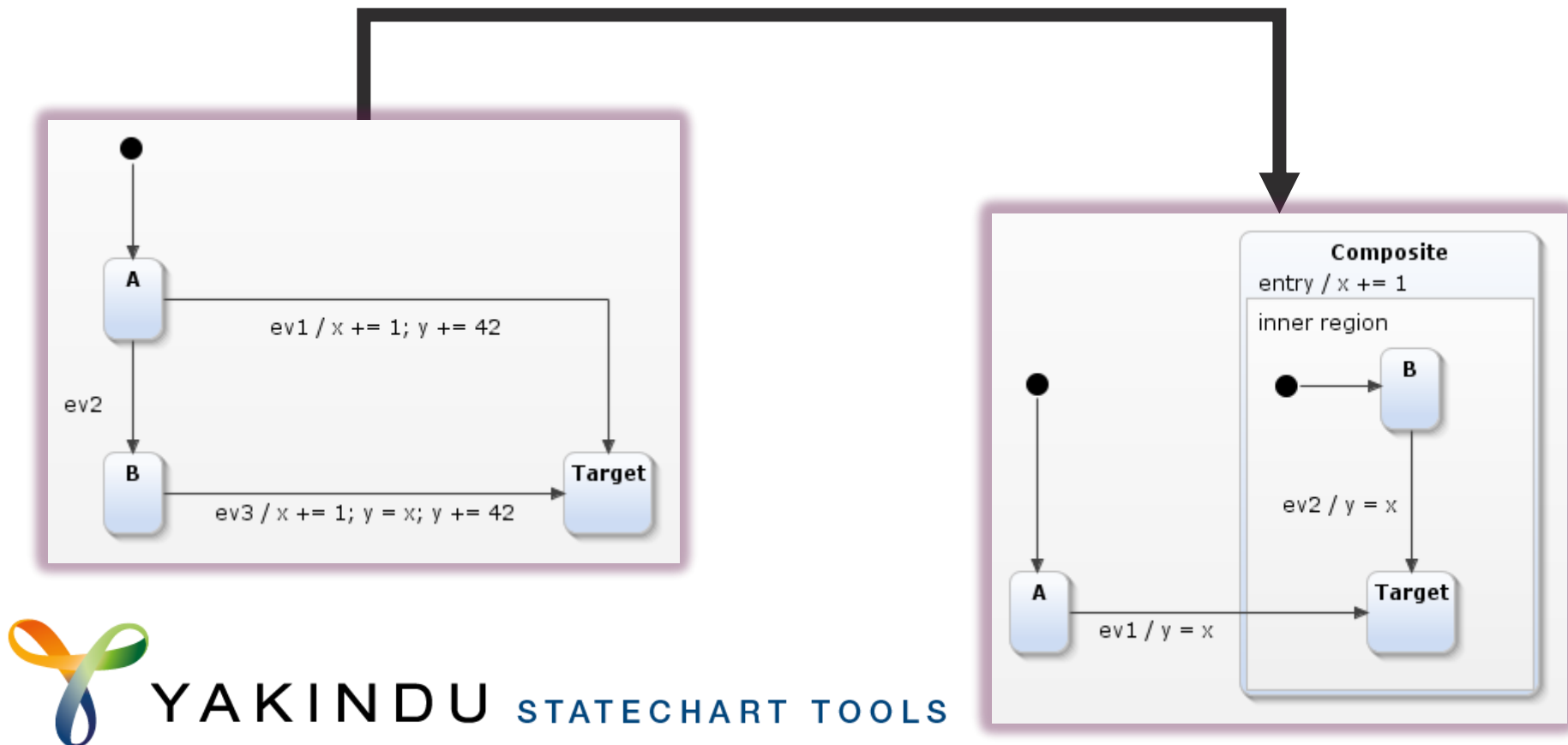


YAKINDU STATECHART TOOLS



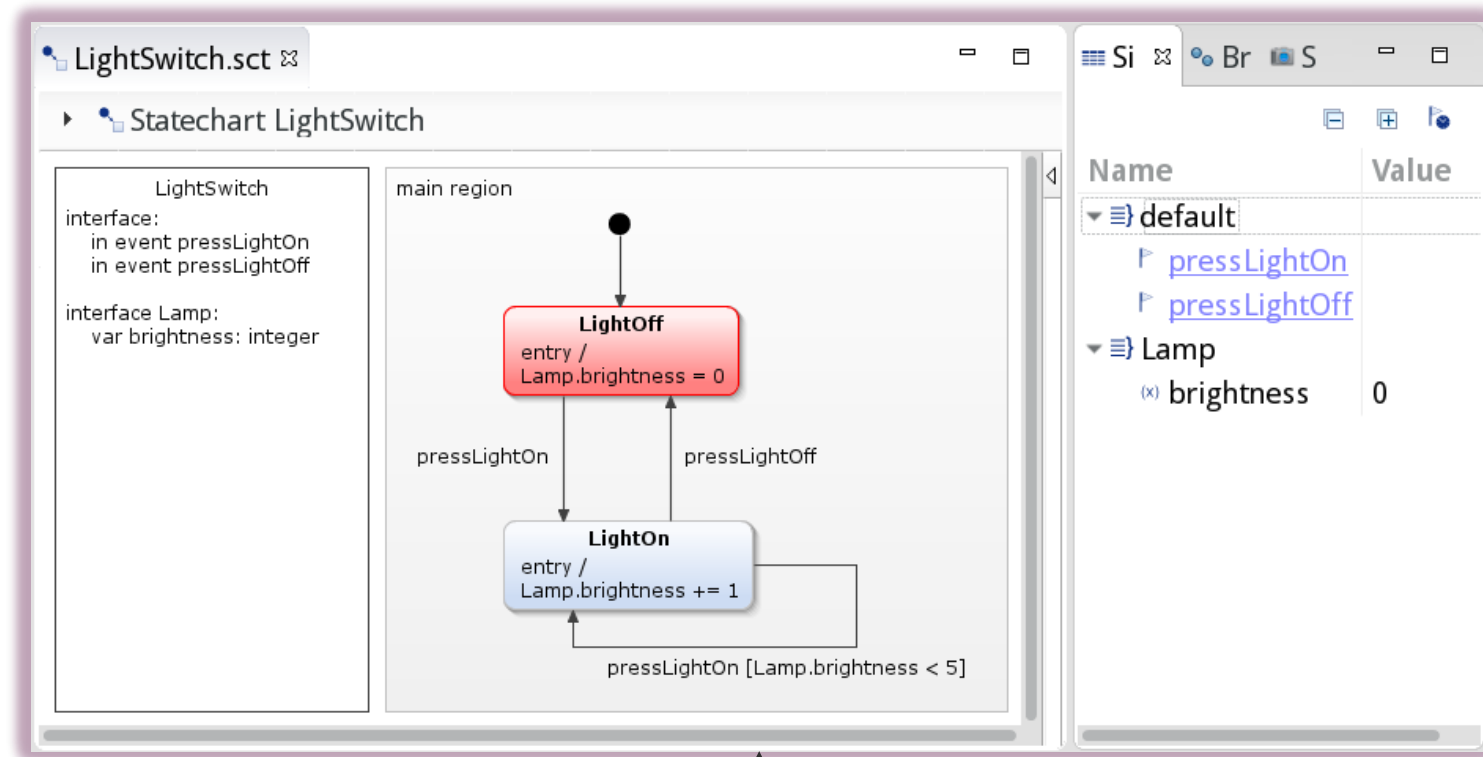
# M2M example 2

- Model refactoring



# M2M example 3

- Simulation
- Semantics



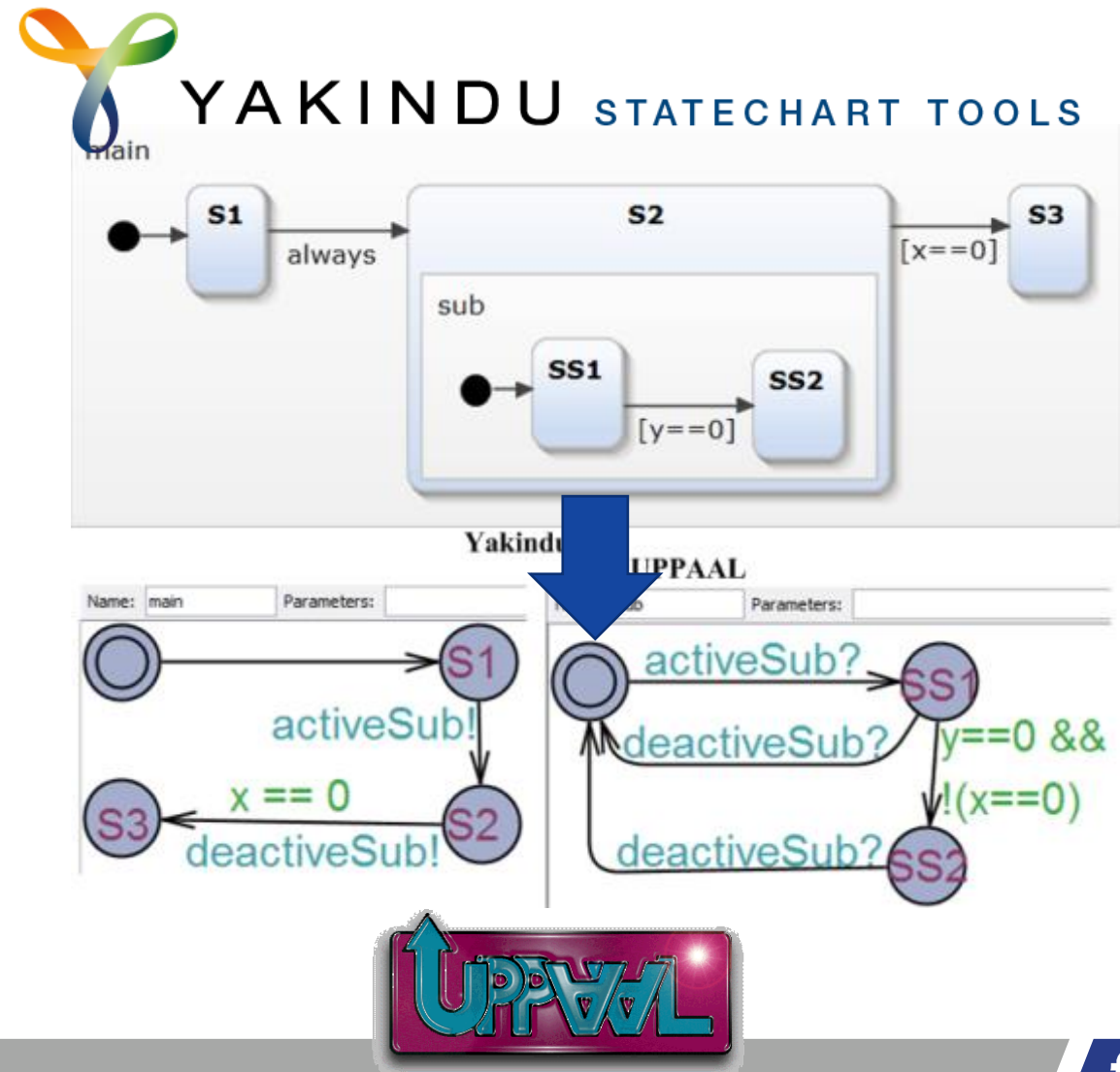


# M2M example 4

- Formal methods
- Hidden formal methods:

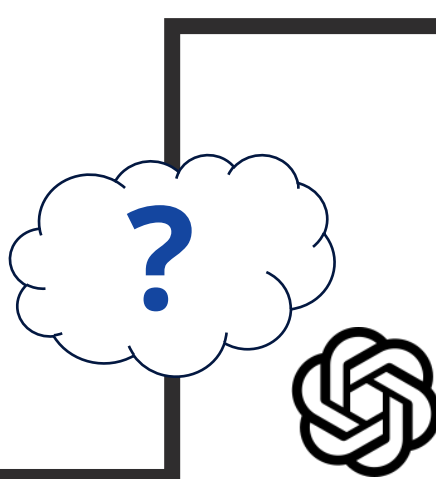
Algorithms than can be applied without verification expertise

- Tool support
- Back-annotation of results



# T2T example?

```
class Point
{
    public:
        int32_t get_x();
        void set_x(int32_t x);
        int32_t get_y();
        void set_y(int32_t y);
    private:
        int32_t x;
        int32_t y;
};
```



```
#ifndef DEFAULTSM_H_
#define DEFAULTSM_H_
#include "sc_types.h"
#include "StateMachineInterface.h"

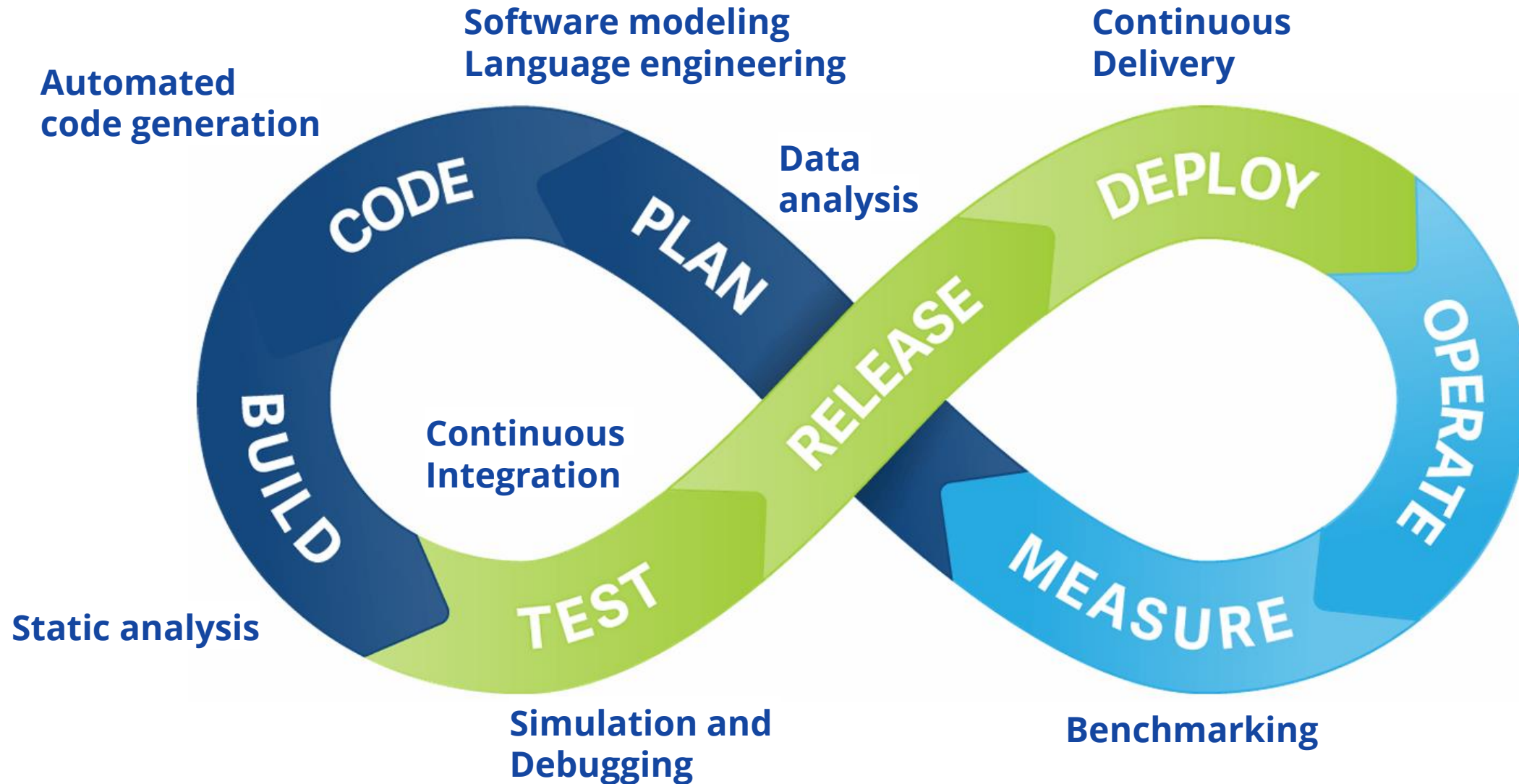
class DefaultSM : public StateMachineInterface
{
    public:
        DefaultSM();
        ~DefaultSM();
        /*! Enumeration of all states */
        typedef enum
        {
            main_region_MyState,
            DefaultSM_last_state
        } DefaultSMStates;
        /*! Inner class for Sample interface scope.
        class SCI_Sample
        {
            public:
                /*! Gets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
                sc_boolean get_a();
                /*! Sets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
                void set_a(sc_boolean value);
                /*! Raises the in event 'evA' that is defined in the interface scope 'Sample'. */
                void raise_evA(sc_boolean value);
                /*! Checks if the out event 'evB' that is defined in the interface scope 'Sample' has been raised. */
                sc_boolean isRaised_evB();
                /*! Gets the value of the out event 'evB' that is defined in the interface scope 'Sample'. */
                sc_integer get_evB_value();
            private:
                friend class DefaultSM;
                sc_boolean a;
                sc_boolean evA_raised;
                sc_boolean evA_value;
                sc_boolean evB_raised;
                sc_integer evB_value;
        };
        /*! Returns an instance of the interface class 'SCI_Sample'. */
        SCI_Sample* getSCI_Sample();
        void init();
        void enter();
        void exit();
        void runCycle();
        sc_boolean isActive();
        sc_boolean isFinal();
        sc_boolean isStateActive(DefaultSMStates state);
    private:
        static const sc_integer maxOrthogonalStates = 1;
        DefaultSMStates stateConfVector[maxOrthogonalStates];
        sc_ushort stateConfVectorPosition;
};
```

- Usually not *directly* applicable: the text must be transformed into a **structured** model (T2M, parser) before programmatic processing
- But recently **Large Language Models** (LLM) became available, which can process text without and intermediate step
  - Challenges: **finite context window**, **hallucinations**, ensuring **correctness**

Even  
**natural language**  
input /output

# Commonly automated tasks

# Areas of automation



# Continuous Integration (CI)



Martin Fowler

<https://martinfowler.com/articles/continuousIntegration.html>

- „a software development *practice* where members of a team integrate their work *frequently*, usually each person integrates *at least daily*”
- „Each integration is verified by an *automated build* (including test) to detect integration errors as quickly as possible.”

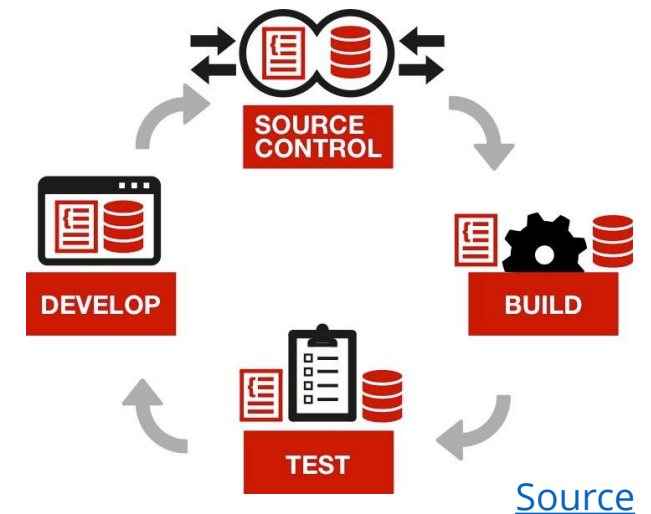
**Maven™**



**Gradle**



JUnit 5

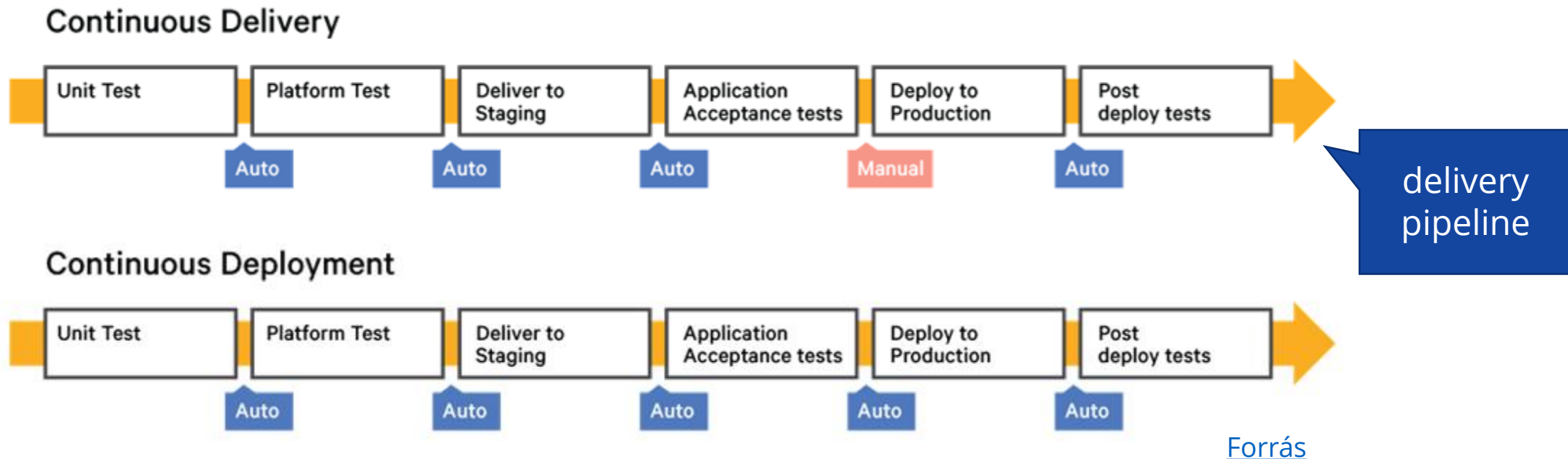




# Continuous Delivery (CD)

„build software so that it is always in a state where it could be put into production”

Source: <https://martinfowler.com/bliki/ContinuousDelivery.html>

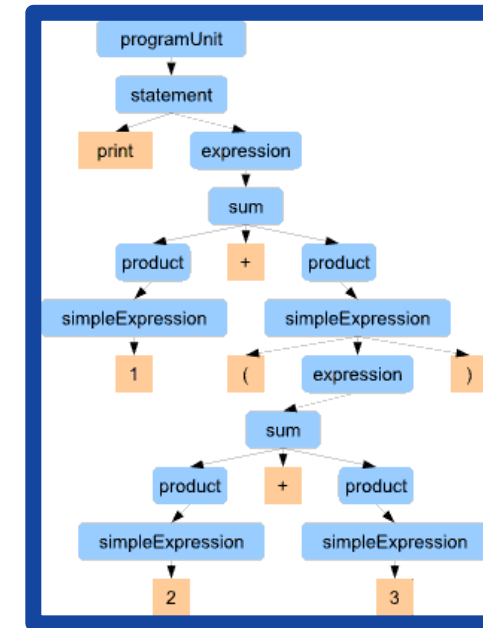


GitHub Actions



# Languages and compilers

- How to create our own (programming) languages?
- How to express grammatical rules?

A screenshot of Java source code, specifically a class named `AboutDialog`. The code includes imports for `com.lauchenauer` packages, class declarations for `CardLayout`,  `JButton`, and  `JPanel`, and method definitions like `initUI()` and `setSize(440, 600)`. The code is highlighted with a red border.

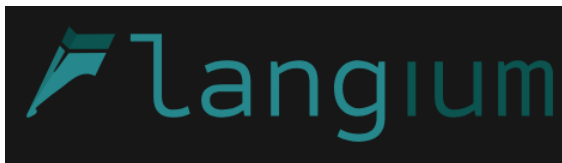
Xtext

Langium

Refinery

# Modern development environments

- **Intelligent functions** in development environments
  - Common helper functions: auto-completion (content assist), just-in-time validation, refactoring
  - More recently: **Large Language Models** (LLM) supporting coding and software modeling
- Browser-based development tools
  - No need for local installation
  - Easily reproducible environment

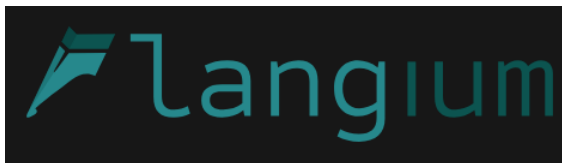
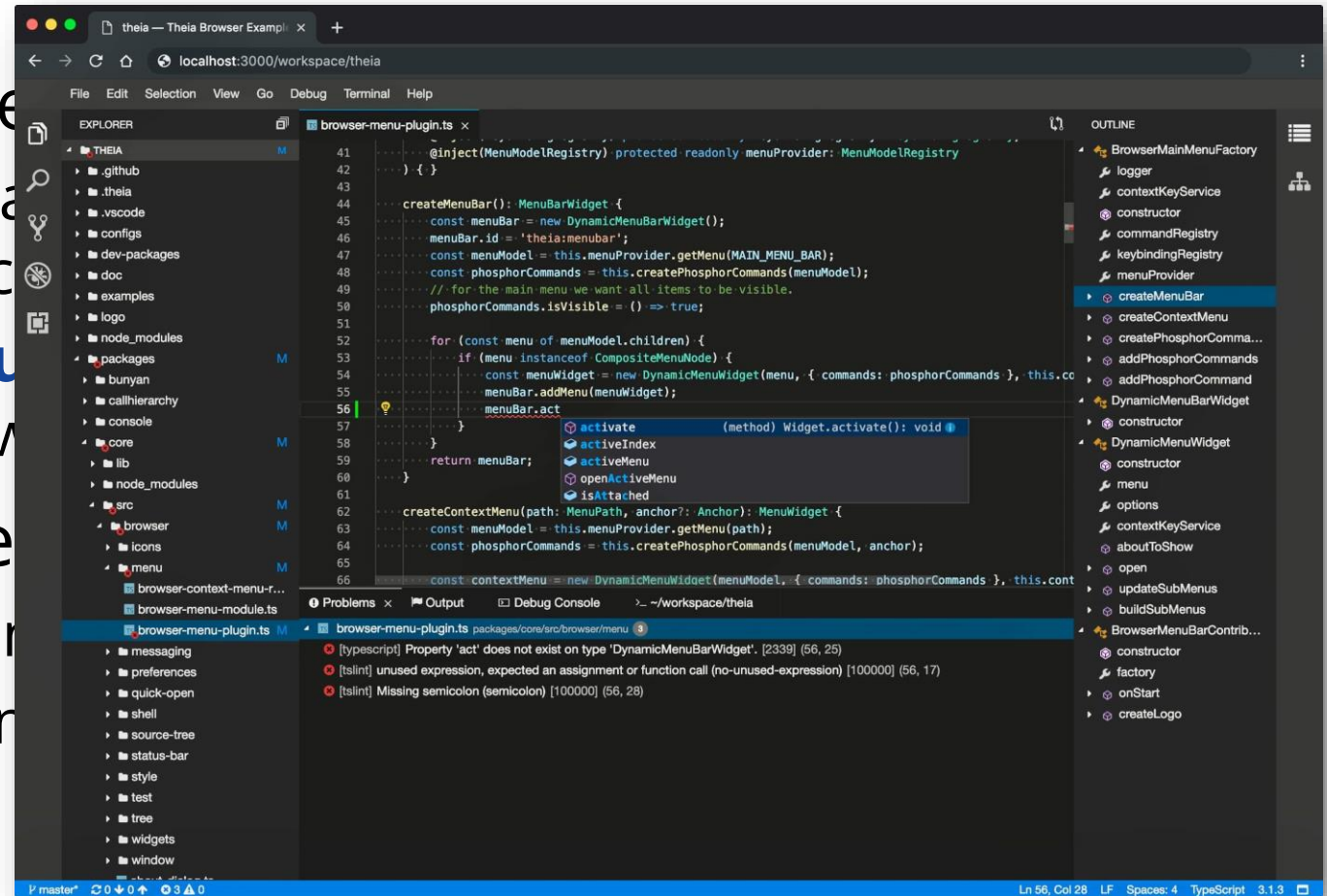


GitHub Copilot



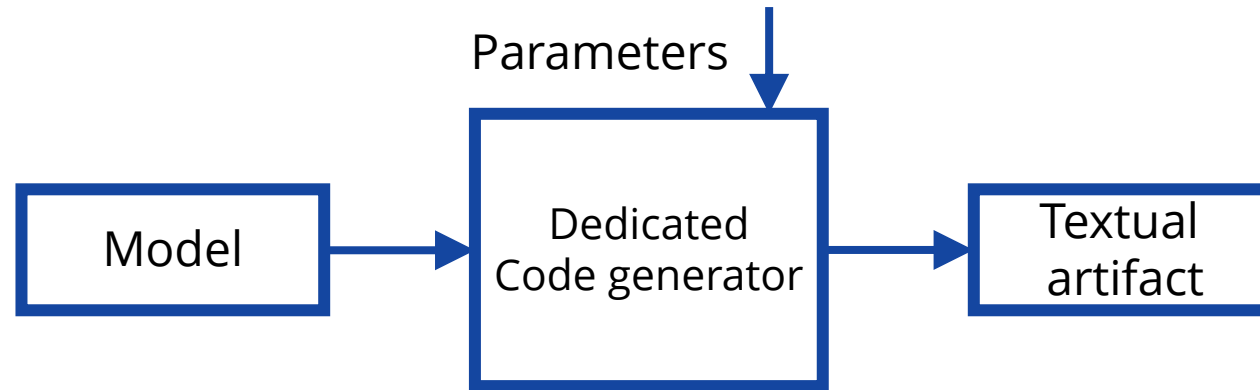
# Modern development environments

- **Intelligent functions** in development environments
  - Common helper functions: auto-completion, just-in-time validation, refactoring
  - More recently: **Lange Language** supporting coding and software development
- Browser-based development environments
  - No need for local installation
  - Easily reproducible environments



# Code generators

- How to transform our models into source code?
- How to tackle programming tasks automatically?



- By automating common coding tasks we can
  - incorporate **mathematical algorithms** into the development process
  - make tasks requiring **boring/long/complex** code feasible

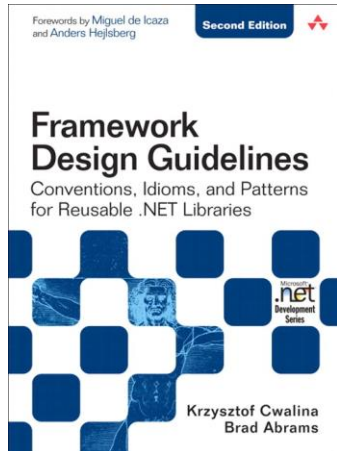
# Static analysis

- Verification of an artifact (documentation, model, code) **without executing** it
- Approaches for evaluating code quality
  - Syntactic correctness: checked by compiler



# Static analysis

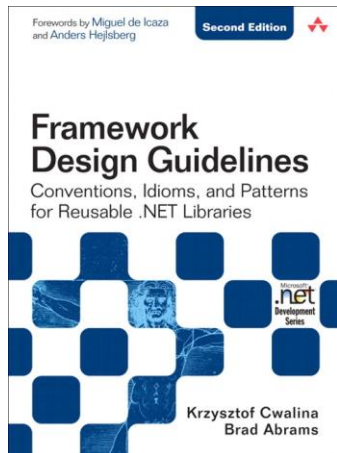
- Verification of an artifact (documentation, model, code) **without executing** it
- Approaches for evaluating code quality
  - Syntactic correctness: checked by compiler
  - Coding conventions: 'linter'



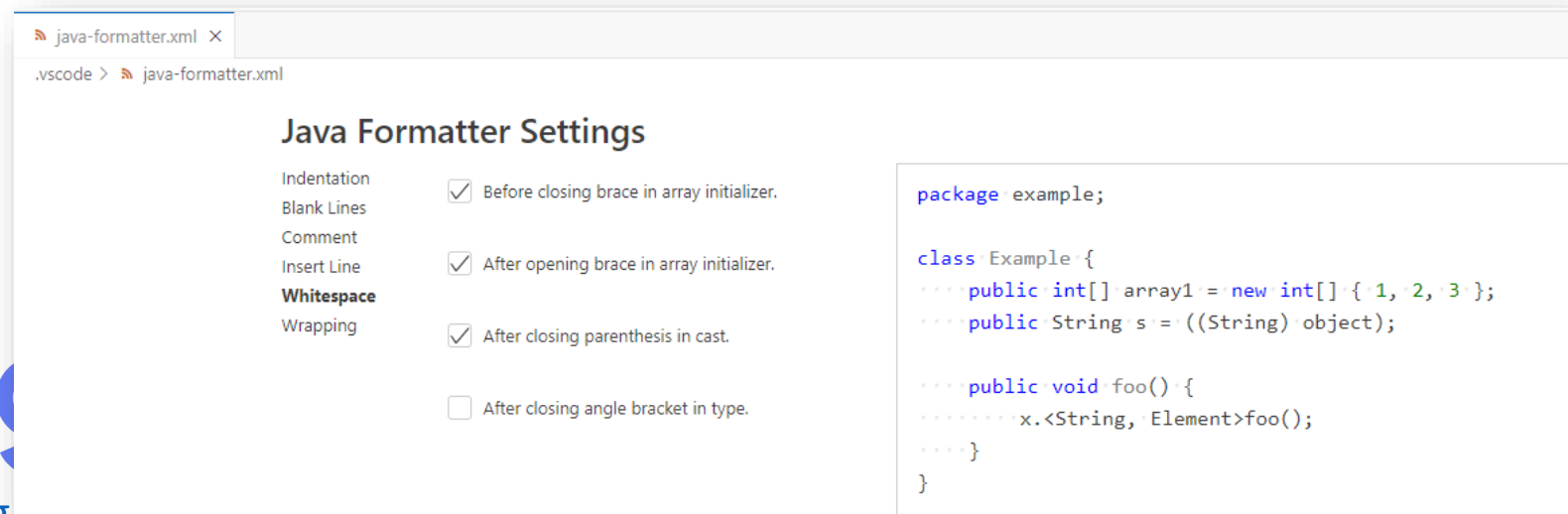
<https://google.github.io/styleguide/javaguide.html>

# Static analysis

- Verification of an artifact (documentation, model, code) **without executing** it
- Approaches for evaluating code quality
  - Syntactic correctness: checked by compiler
  - Coding conventions: 'linter'



<https://google.github.io/styleguide/javaguide.html>



# Statikus analízis

- Verification of an artifact (documentation, model, code) **without executing** it
- Approaches for evaluating code quality
  - Syntactic correctness: checked by compiler
  - Coding conventions: 'linter'
  - Correctness: static analysis (e.g. null pointer analysis)

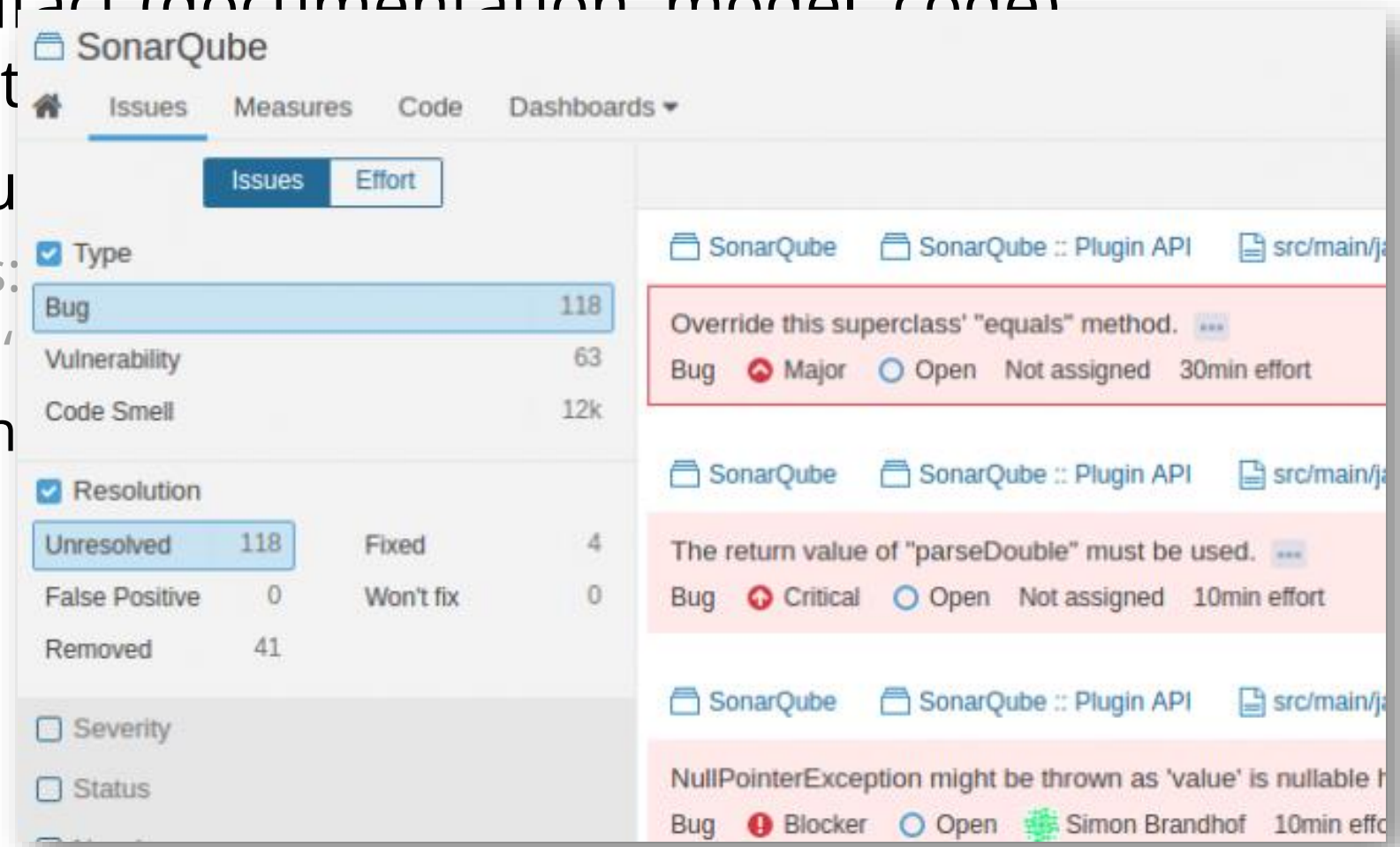


# Statikus analízis

- Verification of an artifact (documentation, model, code) **without executing** it
- Approaches for evaluation
  - Syntactic correctness:
  - Coding conventions:
  - Correctness: static analysis

SpotBugs

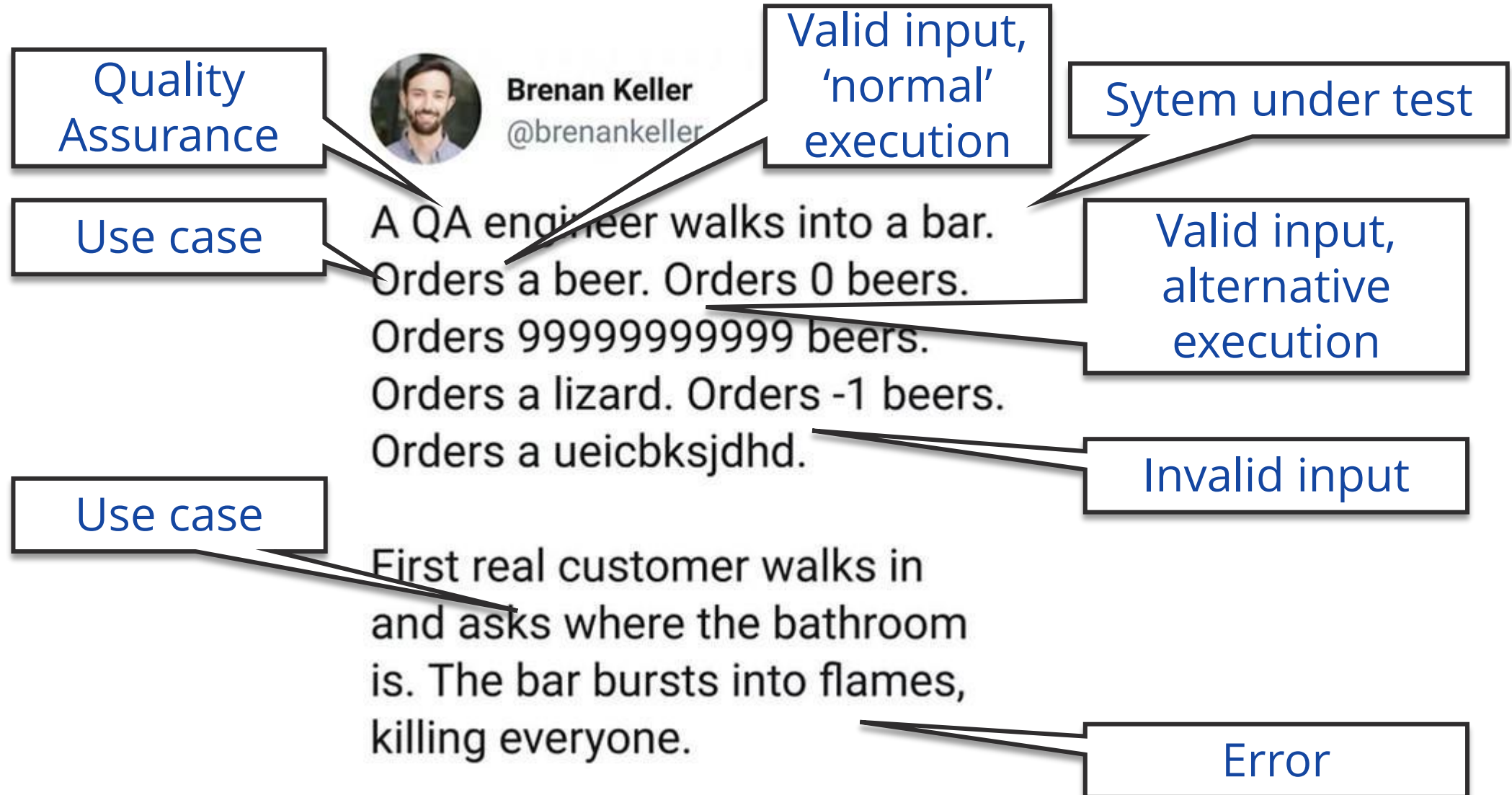
sonarcloud 



The screenshot shows the SonarQube interface with the 'Issues' tab selected. The left sidebar shows filters for 'Type' (Bug: 118, Vulnerability: 63, Code Smell: 12k) and 'Resolution' (Unresolved: 118, Fixed: 4, False Positive: 0, Won't fix: 0, Removed: 41). The main area displays a list of bugs with details such as severity, status, and effort.

Issue Key	Message	Severity	Status	Assignee	Effort
SonarQube :: Plugin API	Override this superclass' "equals" method.	Major	Open	Not assigned	30min
SonarQube :: Plugin API	The return value of "parseDouble" must be used.	Critical	Open	Not assigned	10min
SonarQube :: Plugin API	NullPointerException might be thrown as 'value' is nullable	Blocker	Open	Simon Brandhof	10min

# Test design



# Automated test design

- **Model-based** testing, test generation (M2T, M2M)
- **Structure-based** techniques:

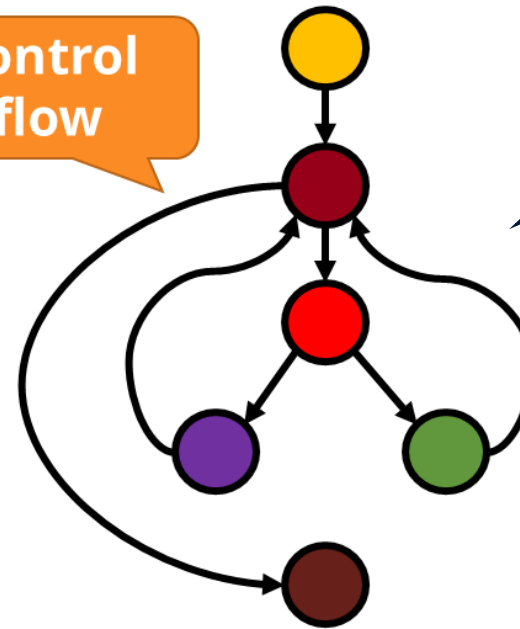
Representing  
the **structure**  
of source code

Source code:

```
int a = read();  
while(a < 16) {  
    if(a < 10) {  
        a += 2;  
    } else {  
        a++;  
    }  
}  
a = a * 2;
```

Control-flow graph (CFG):

control  
flow



Code coverage  
evaluation

# Automated test design

- **Model-based** testing, test generation (M2T, M2M)
- **Structure-based** techniques:

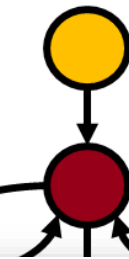
Representing the **structure** of source code

Source code:

```
int a = read();
while(a < 16) {
    if(a < 10) {
        a += 2;
    } else {
        a++;
    }
}
a = a * 2;
```

Control-flow graph (CFG):

control flow



Code coverage evaluation

JaCoCo > org.jacoco.report

org.jacoco.report

Element	Instruction Coverage	Missed Classes	Missed Methods	Missed Blocks	Missed Lines
org.jacoco.report.html	<div><div></div></div> 63%	10 / 20	54 / 128	84 / 214	117 / 385
org.jacoco.report.csv	<div><div></div></div> 20%	8 / 9	35 / 43	52 / 68	100 / 126
org.jacoco.report	<div><div></div></div> 75%	3 / 7	6 / 25	12 / 68	26 / 98
org.jacoco.report.xml	<div><div></div></div> 90%	2 / 10	9 / 42	13 / 68	17 / 146
org.jacoco.report.html.resources	<div><div></div></div> 87%	1 / 3	1 / 7	9 / 40	2 / 35
Total	64%	24 / 49	105 / 245	170 / 479	262 / 790

Code Coverage Report for JaCoCo 0.1.0.20091027174426

Created with JaCoCo 0.1.0.20091027174426

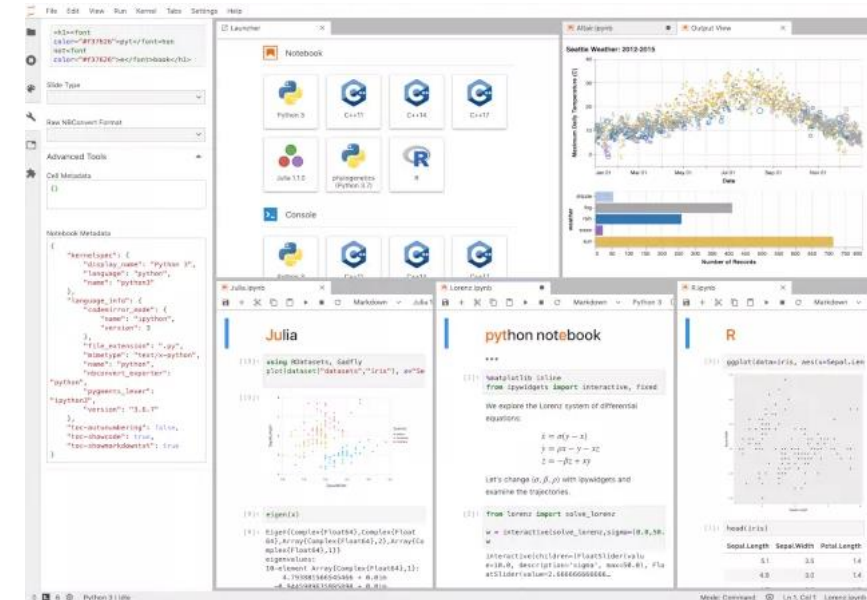


# Benchmarking, data analysis

- How to evaluate software performance?
  - Measurement setup
  - Software benchmarking best practices
  - Tools: JMH, VisualVM
- How to evaluate measurement results?
  - Data processing
  - Data visualizations
  - Hypothesis testing



<https://openjdk.org/projects/code-tools/jmh>



# Where should we use these methods?

Increasing levels of technical investment



## Nearly all projects

- Automated build
- Dependency management
- IDE warnings
- Unit tests
- Debugging

## Larger, long-term projects

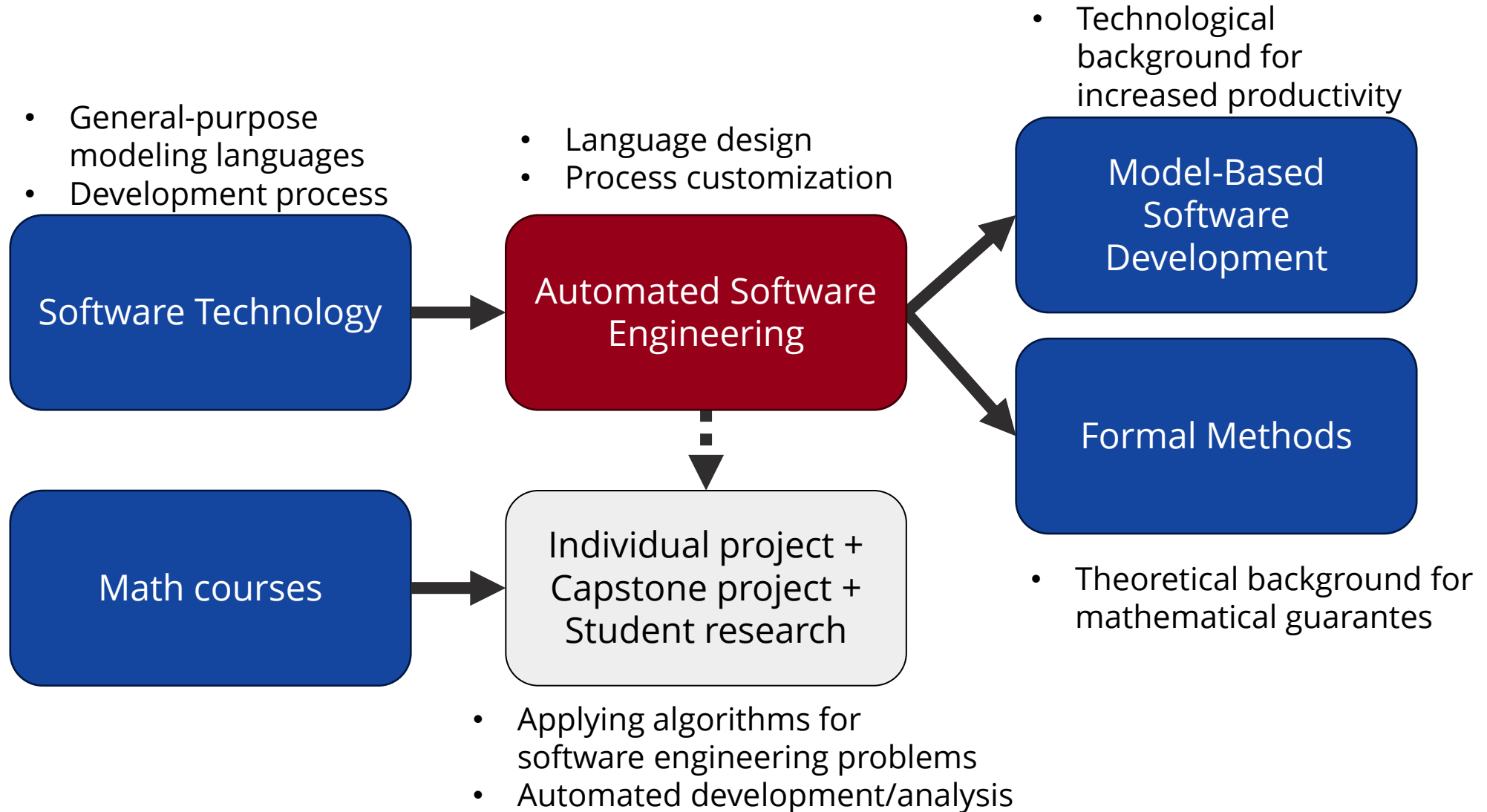
- Continuous Integration & Delivery
- Code generation
- Static analysis
- Performance evaluation
- Data analysis

## Critical systems

- Domain-specific languages
- Custom static analysis rules
- HW/SW simulation
- Formal verification

# Outlook

# Related courses



# Industry relations

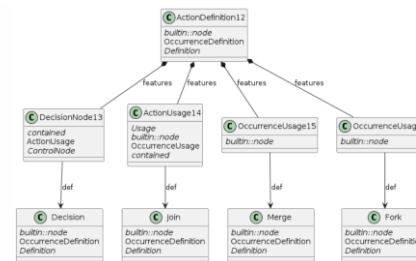
- Automated software development techniques enable us to create novel **development environments**
- These may comprise
  - Mathematical **algorithms** / physical **simulators**
  - **Artificial Intelligence** models
  - Evaluation of **safety rules**
- This lets us work with **domain experts**

In the following, we will show some industrial collaborations and recent results of our research group. Everyone is welcome to participate if they wish to do their capstone project at our group!

# Next-Generation MBSE – SysML v2

## Participation

- Authorship in the specification
- SMC Leadership
- Formal Methods WG (leading)
- Conformance WG (leading)
- Execution WG (core member)
- Semantics WG (core member)
- Reference Implementation WG
- Certification WG



```
verification def CrossroadVerification (  
  subject crossroads : Crossroads {  
    >>> controller.trafficDuration := 5;  
    >>> controller.stoppingDuration := 2;  
  }  
  
  objective ProveSafetyObjective {  
    verify requirement {  
      require constraint {  
        AG (! (crossroads.trafficLightA.greenOn  
          && crossroads.trafficLightB.greenOn))  
      }  
    }  
  }  
  
  return verdict : VerdictKind;  
}
```

## Recent results

Several new  
**partners** and  
**cooperations**

First SysML v2 formal  
**verification** and  
**model generator tool**

Coordinating first  
**academic paper** about  
SysML v2 verification

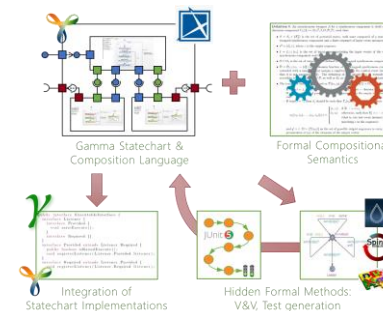
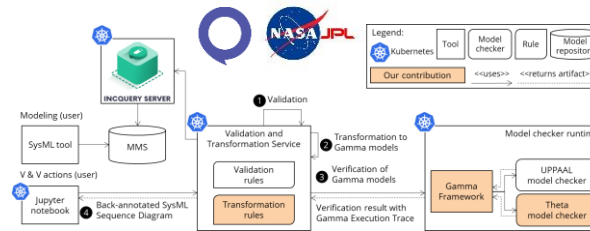
# Automated V&V in Systems Engineering

## “Use your models!”

- Semantic analysis
- Early verification of critical requirements
- Code/Test/Monitor generation

## Gamma: Bridge MBSE and FM

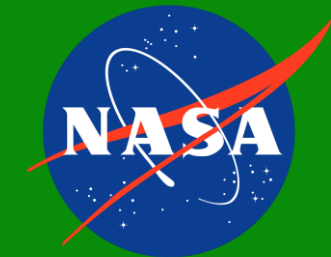
- Formal verification integrated into SysML, SysML v2 and DSLs



## Recent results

Automated analysis and generation for **railway systems**

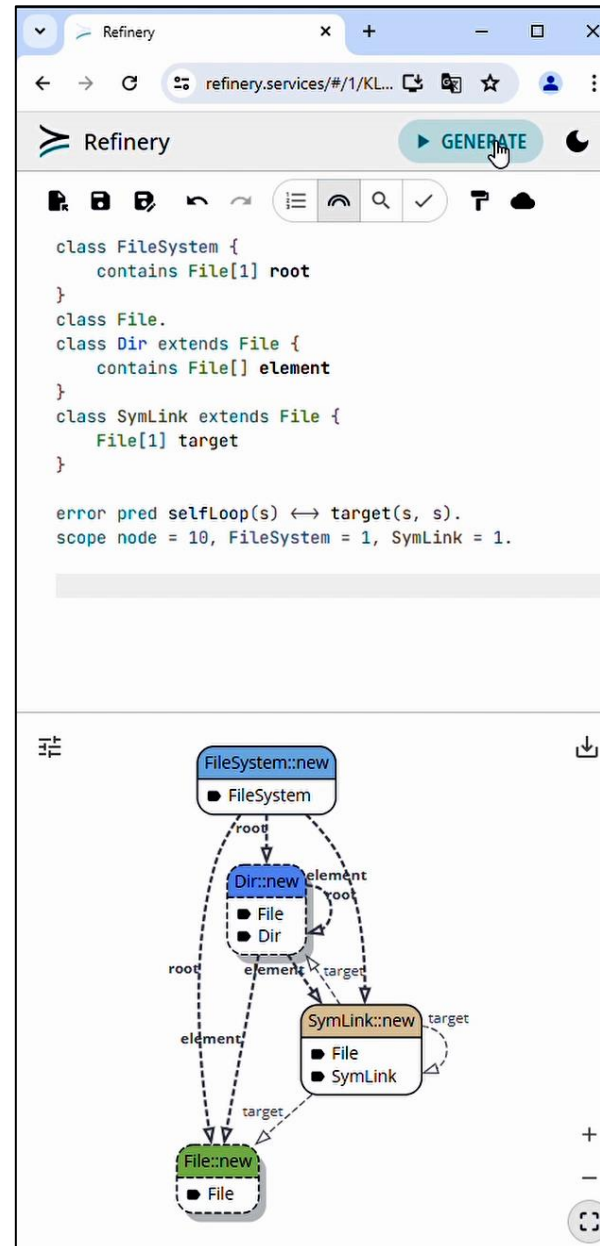
Automated formal methods in **engineering workflows** (NASA, IQL)





# Modern tools

- **Tool development**  
with modern technologies
- Interactive state-of-the-art **web-based editors**
- Generation algorithms  
**running on server**
- Various domains  
*blockchain architecture | chemical reaction  
railway topology | modeling environment,  
satellite network | video game maps*
- **Goal:** support **engineers** and **experts** solving problems



## Recent Results

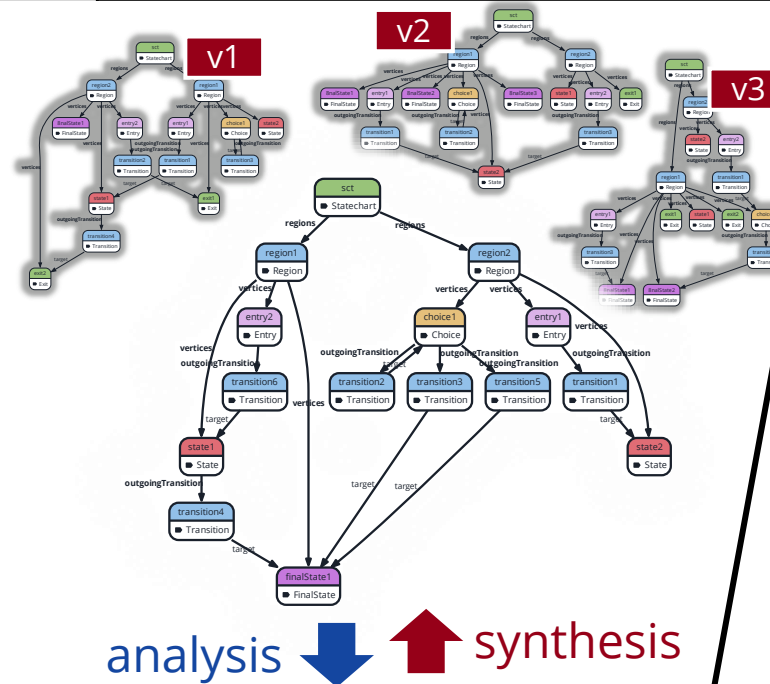
-  Refinery  
*Continuously deployed:*  
<https://refinery.tools>

- Amazon  
Research Award  
*First in the region*

**amazon** | science

# Graph analysis and synthesis

- Powerful mathematical **analysis techniques** for models
- Novel graph-based logic solver for the **automated synthesis** of design alternatives
- **Precision + Scalability**
- **Goal:** solve problems with complex structure



❌	✅	validation
\$\$\$		cost
▲▲		performance
%%		coverage

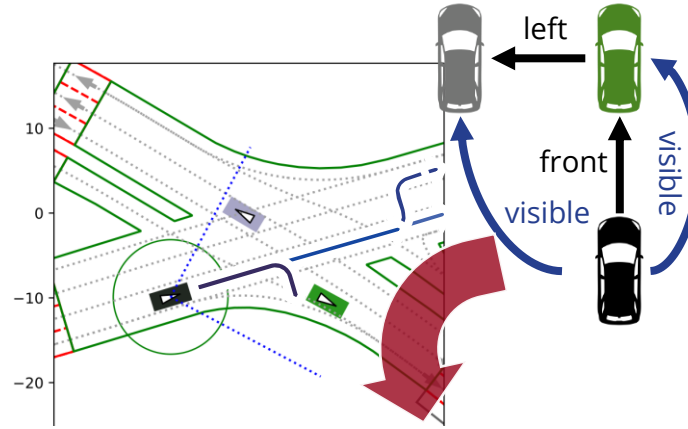
## Recent Results

- Research project  
*VERIFIABLE AI/ML  
TECHNIQUES FOR PNT  
APPLICATIONS*



# Verification/Testing of AI/ML Applications

- AI applications are **data-oriented** systems
- **Complex, dynamic** environment
- Novel **generation** + Advanced **simulators**  
→ Diverse **tests**
- Systematic testing of **AI applications**



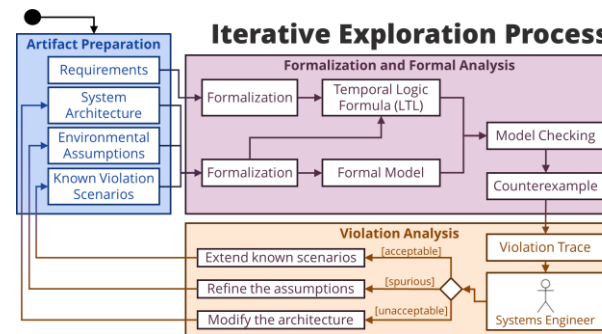
## Recent Results

- R&D project with Knorr-Bremse
- Research project with USA Navy  
*Office of Naval Research Global*



# Finding Critical Design Errors (Verification)

- **Methodology** to find **errors** in system design
- **State-of-the-art** verification algorithms
  - (Concurrent) software verification
  - Real-time systems
  - High-level engineering models
  - Component-based systems



## Winners

LIA-Lin	LIA	LIA-Lin-Arrays	LIA-Arrays	ADT-LIA	ADT-LIA-Arrays
Eldarica	Eldarica&Golem	Theta	Eldarica	Cata	Eldarica
Golem	U.Unihorn	Eldarica	U.Unihorn	Eldarica	
Loat	U.Tree.Automizer	U.Un	Theta		



## Recent results

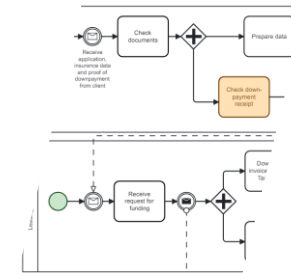
Found **real problems** in industrial use-cases (automotive, railway)

Last 5 years: 17 **first prizes** in university level TDK competitions

Recent theta **awards**:  
🏆 Distinguished artifact  
🏅 Horn close competition

# Advancing novel applications – key activities

- **Hungarian Blockchain Coalition**
  - Prof. Pataricza – member of the board
  - I. Kocsis: Education WG lead, L. Gönczy: FinTech WG
- **Supporting the EMAP project (PM/NAV)**
  - “Even-based Data-sharing Platform” pilot
  - Employer data provisions: event-based, single-channel
  - Blockchain-based implementation in preparation
- **CBDC research cooperation with MNB**
  - Mapping out: blockchain ↔ Central Bank Digital Currency
  - Payment, car leasing, energy support, industrial cooperation
  - Currently: “ecosystem” research
- **EDGE-Skills: data veracity in EU data spaces**
  - Blockchain-backed Verifiable Credentials



## Recent results

Energy price  
support CBDC  
prototype: **BIS**  
**Rosalind finalist**

Fabric ↔ Ethereum  
**CBDC bridge** in  
Hyperledger Cacti

**Smart gas meters**  
and readings – in  
production

Welcome everyone to the specialization!

We look forward to also working with you in  
the project courses of our research group!