
TNO ESI
Powered by industry
and academia

MODEL-BASED DEVELOPMENT: INTRODUCTION

Software

Software Systems (Computer & Embedded System Engineering)

Rosilde Corvino

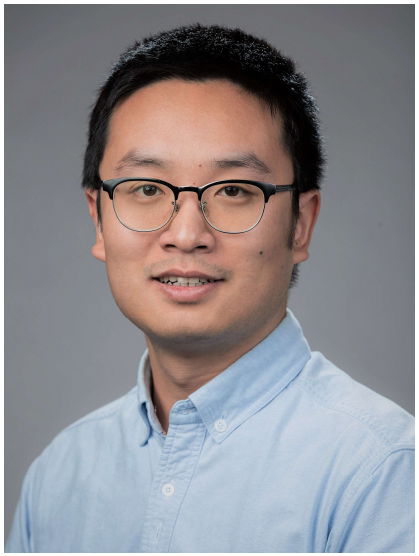
17-12-2024

TU Delft Delft University of Technology

TNO ESI Powered by industry and academia

LECTURERS AND TEACHING ASSISTANTS

Rosilde Corvino (TNO-ESI)

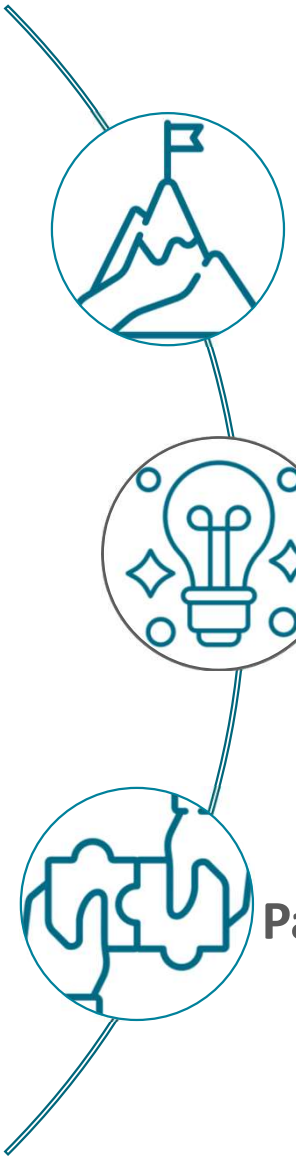


Guohao Lan (TU Delft)

Pepijn Kremers (Teaching assistant)

TNO ESI

Powered by industry and academia



Mission: Embedding cutting-edge methodologies into the Dutch high-tech systems industry to cope with the ever-increasing complexity of their products

Philosophy: *"the industry as a lab"*

Partners:

ASML, Canon Production Printing, Vanderlande, Radboud University Nijmegen, Philips, University of Amsterdam, TU/e Eindhoven University of Technology, University of Twente., TU Delft, Thales, itec, ICT Group Strypes, Capgemini engineering

OBJECTIVES OF THE COURSE

- At the end of the course, you should be able to:
  - Explain some complexity challenges of software-intensive high-tech systems
  - Explain the purpose of Model-Based Development, some of its principles and approaches
  - For 3 specific modeling techniques: explain their purpose and the basic concepts and create basic models
  - Compare Model-Based Development with other methodologies

- Assessment:
  - Modeling assignments for 3 modeling techniques (in groups of 2 students)
    - Maximum of 2 points per assignment
  - Reflection document on Model-Based Development (individual)
    - Maximum of 4 points

AGENDA OF THE COURSE

	Week 6 (17-12)	Week 6 (19-12)	Week 7 (7-1)	Week 7 (9-1)	Week 8 (14-1)	Week 8 (16 -1)	Week 9 (21-1)	Week 10
Lectures on Tuesdays (2 hours)	Introduction		StateChart 1		DSL 1			
	UML 1		StateChart 2		DSL 2			
Labs on Thursdays (4 hours)		1 UML Lect 3 labs		StateChart		1 DSL lec + conclusion 3 DSL labs		
Assignment due on Friday				UML		Statechart		DSL + Reflection

QUIZ GAMES

- Identified by the banner: **Think/Write → Pair → Share**

- Instructions:
  1. Divide into teams of two students
  2. Discuss the solution to the quiz together
  3. Volunteer or be asked to share
  4. Points will be awarded for participation:
     1. Every time you share during a game, you earn 0.3 points
     2. You can earn up to a maximum of 1 additional point on the final note for this part of the course
     3. Do not forget to write your name on the winners' sheet after the lecture

OUTLINE OF THIS LECTURE

10 minutes	Opening
15 minutes	Complexity challenges in large-scale software systems
20 minutes	Model-based development to manage the complexity

COMPLEXITY CHALLENGES IN LARGE-SCALE SOFTWARE SYSTEMS

WHAT TYPES OF SYSTEMS ARE WE TALKING ABOUT?

- Agree on the definition of the following terms:
  - Embedded Systems
  - Cyber-Physical Systems
  - Software-Intensive Systems

- What kind of **software-intensive cyber-physical systems** do you know that contain **embedded systems** and are from companies that are leaders in the Dutch high-tech systems industry?

- Duration: 5 minutes (3 minutes thinking and discussion with your partner)

Think/Write → Pair → Share

WHAT TYPES OF SYSTEMS ARE WE TALKING ABOUT?

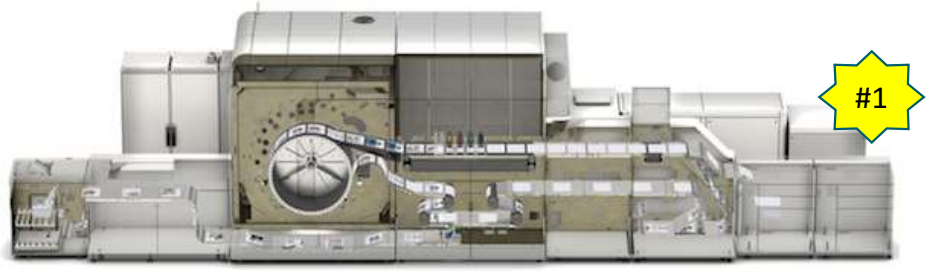
#1

#1

#1

#1

#1

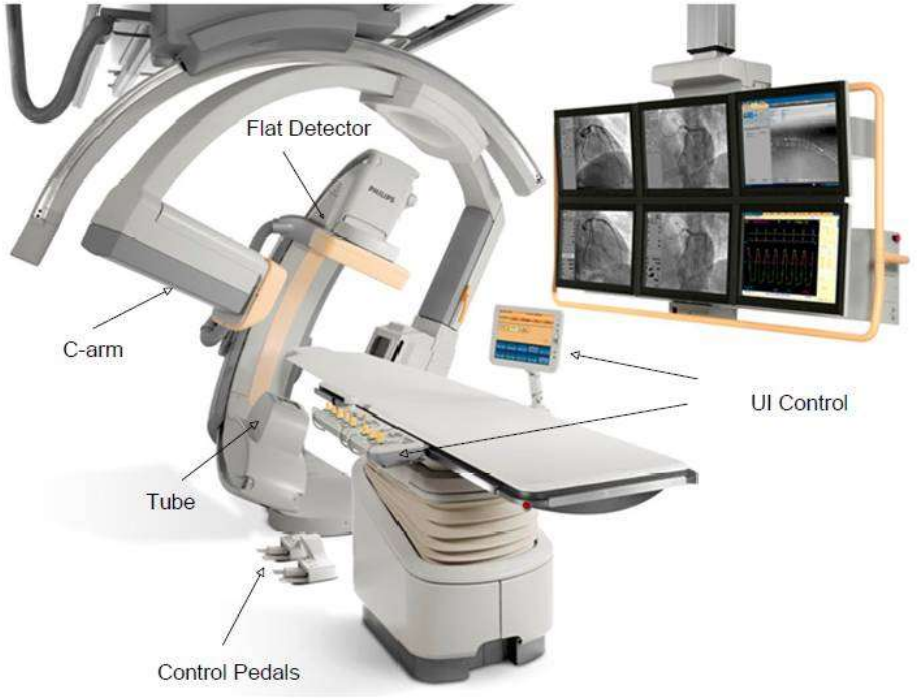


#1

MODELING ASSIGNMENT IS INSPIRED BY INTERVENTIONAL X-RAY SYSTEMS

- Systems used during medical interventions:
  - Minimally-invasive surgery
  - Cardiac, vascular and neurological
  - X-ray is used as “eyes of the surgeon”

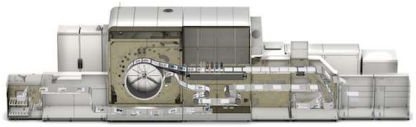
- 1 or 2 X-ray planes (connected by a C-arm), consisting of:
  - Tube: generates X-ray
  - Detector: receives X-ray
- Table at which the patient lays
  - Patient is positioned between the tube and detector
- User interface for the surgeon:
  - Tablet to select the medical procedure
  - Joysticks to move the table and C-arms
  - 3-6 pedals to control the system during a medical procedure
  - Screen to display visual images and video



WHAT TO CONSIDER WHEN DEVELOPING SOFTWARE FOR COMPLEX CYBER-PHYSICAL SYSTEMS?

When you compile your list, consider that:

- Complexity means being challenging to grasp or not straightforward



- You can get some ideas by answering these questions:
  - What makes such systems complex?
  - What makes the software of such systems complex?
  - What makes the design of such systems complex?
  - What makes the design of the software inside such systems complex?

- Duration: 5 minutes (3 minutes thinking and discussion with your partner)

Think/Write → Pair → Share

WHAT TO CONSIDER WHEN DEVELOPING SOFTWARE FOR COMPLEX CYBER-PHYSICAL SYSTEMS?

- Multi-disciplinarity: closely related to physical systems and application domains
- Team effort: multiple development teams that evolve over time

- Highly-specialization: not many identical systems in the field
- Integration: interaction with other systems and humans (outside of your control)

- Large code bases: tens of thousands, or even millions, of lines of code (LOC)
- Long-lived code bases: embedded software evolves over decades

- Performance: quantitative performance criteria (hard real-time, competition)
- Dependability: non-functional requirements like availability, reliability, maintainability

MODEL-BASED DEVELOPMENT TO MANAGE THE COMPLEXITY

WHAT IS A MODEL?

- A model is a simplified representation of a system used to:
  - **understand**,
  - **analyze**,
  - or **predict its behavior**,
  - often serving as a basis for **code generation**

MODEL BASED DEVELOPMENT

MBD is a methodology, i.e., a structured set of methods, practices, and procedures used to achieve a specific objective.

It provides a systematic way to plan, execute, and manage software development.

It ensures consistency, efficiency, and quality.

It combines principles, approaches, and techniques.

- Principles: fundamental truth guiding the methodology
- Approaches: strategies to deal with complexity
- Techniques: methods or procedures used to accomplish a particular task

MBD PRINCIPLES

- “Keep it simple, stupid” (KISS)
  - https://en.wikipedia.org/wiki/KISS_principle
  - Most models work best if they are kept simple rather than made complicated
  - Simplicity should be a key goal in design, and unnecessary complexity should be avoided

- “Don't repeat yourself" (DRY)
  - https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
  - Replace repetition of software patterns with abstractions and use data normalization (no variations)

APPROACHES FOR DEALING WITH COMPLEXITY

A. Abstraction: Identify high-level concepts that hide low-level details
   - Architects design a building in terms of walls (instead of bricks)
   - Aerospace engineers reason about processor instructions instead of individual transistors

B. Boundedness: Impose acceptable restrictions on the considered problem space
   - Architects design a building for a specific set of usage scenarios (residential, industrial, retail)
   - Aerospace engineers assume that airplanes do not need to be usable in outer space

C. Composition: Divide one problem into multiple independent smaller problems
   - Architects design a new district in terms of buildings with separate foundations
   - Aerospace engineers separate flight control from cabin control and in-flight entertainment

D. Duplication: Use multiple overlapping approaches for the same problem
   - Architects use blueprints with multiple perspectives on the same building
   - Aerospace engineers introduce fallback systems for safety-critical functionality

EXAMPLES OF MODELING TECHNIQUES ARE

- **Unified Modeling Language** (UML): A standardized modeling language that includes various diagrams like class diagrams, sequence diagrams, and use case diagrams to visualize the design of a system.
- **Entity-Relationship Diagrams** (ERD): Used to model the data structure of a system, showing entities, attributes, and relationships.
- **Statecharts**: Diagrams that represent the states and transitions of a system, useful for modeling dynamic behavior.
- **Data Flow Diagrams** (DFD): Illustrate how data moves through a system, showing inputs, processes, and outputs.
- **Flowcharts**: Visual representations of the sequence of steps in a process or system.
- **Domain-Specific Languages** (DSL): Specialized languages tailored to specific aspects of a system, providing higher abstraction and efficiency.
- **Petri Nets**: Used for modeling concurrent systems, showing transitions and states in a graphical form.

SOFTWARE DEVELOPMENT LIFE-CYCLE

- The software development lifecycle (SDLC) is a structured process for planning, creating, testing, and deploying software applications.

Requirements analysis	Design	Implementation	Testing	Deployment	Maintenance
•Stakeholder Identification •Requirements Elicitation •Documentation •Analysis and Negotiation •Validation and Verification	•High-Level Design: components and interactions •Detailed Design: algorithms, data structures and interfaces •Design documentation	•Coding •Code Reviews •Unit Testing	•Integration Testing •System Testing •User Acceptance Test	•Deployment Planning •Deployment Execution •Post-Deployment Verification	•Monitoring •Bug Fixes •Updates and Enhancements •Documentation Update

SDLC: WHERE ARE MODELING TECHNIQUES USED AND HOW?

- Duration: 5 minutes (3 minutes thinking and discussion with your partner)

Think/Write → Pair → Share

SDLC: WHERE ARE MODELING TECHNIQUES USED AND HOW?

Requirements analysis	Design	Implementation	Testing	Deployment	Maintenance
• **DSL** to capture domain-specific requirements	• **UML** to capture system architecture: components and interactions • **Statechart**: system behavior	• **UML**: class diagrams • **DSL**: code generation for domain-specific modules, e.g. middleware	• **UML**: use case diagram for scenario-based testing • **Statechart**: verify the behavior of state dependent modules	• **UML**: deployment diagram	• **DSL**: for scripting code restructuring

MODELING FOR A SPECIFIC PURPOSE