Introduction†

IN4390 Quantitative Evaluation of Embedded Systems

Koen Langendoen

†Original slides by Mitra Nasri, now at TU/e

Who is who

Teachers & Topics

course coordinator

Koen Langendoen
- Operational laws
- Queueing theory

Marco Zuniga
- Markov chains

flipped classroom

Lydia Chen
- Design of Experiments

George Iosifidis
- Petri nets

Who is who

Teaching assistants

Naram Mhaisen
- Lab assignments

Agrim Sharma
- Lab assignments

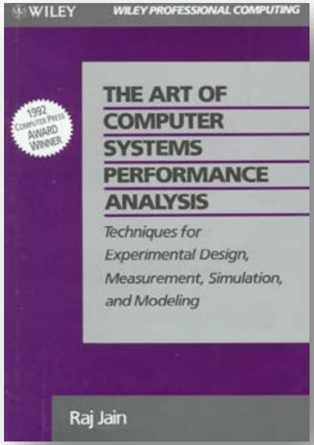
Fridays
08:45 – 12:30

Weeks
2.2 – 2.6, 2.8

Course setup

The first 80% – compulsory

- Lectures
  - Theory, instructions, examples, Q&A
- Exam
  - Written exam with open-ended questions
- Practicum (lab)
  - 3 main assignments
  - Tools: ROS and Petri-nets
  - Required: basic knowledge of C++ and Linux



ROS Robot Operating System

Course setup

The last 30% – elective

- In-lecture quizzes
- Take-home questions
- Extra lab questions
- Project

10% bonus

Customizable part

Grading scheme

The fine print

$$\text{Final grade} = \min\left(10, \frac{E + C}{100}\right) * M$$

M = {pass=1 | fail=0} (**mandatory** assignments)

E = **Exam**

C = **Customizable** points

$$400 \leq E \leq 800$$
$$0 \leq C \leq 300$$

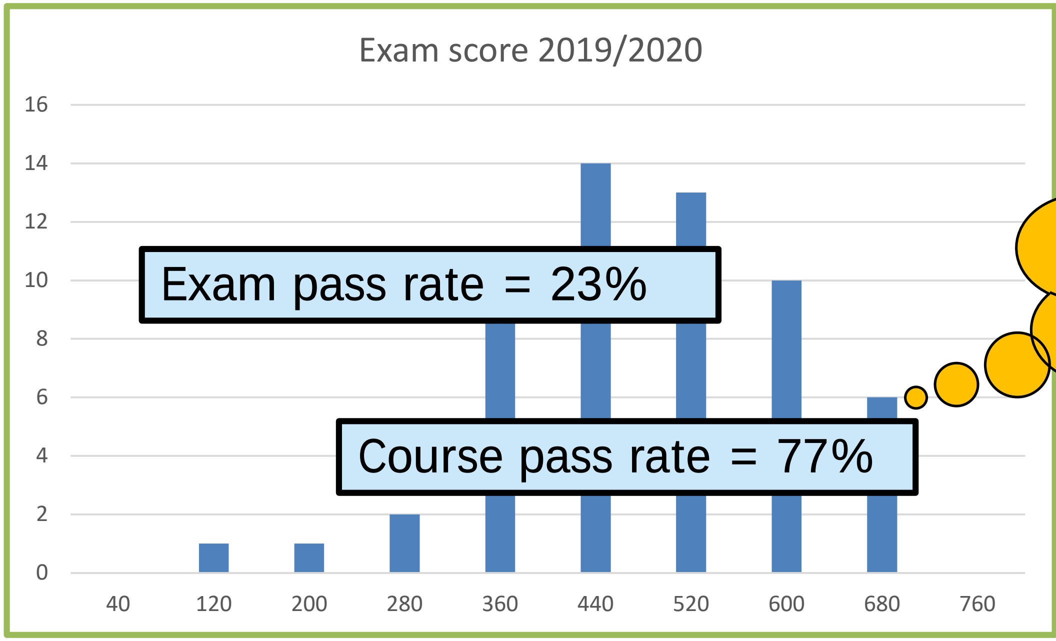
In-lecture quizzes (5x)	20 pts each
Take-home questions (2x)	10 pts each
Extra lab questions	≤ 120 pts
Project	80 - 120 pts

Grading scheme

A word of warning

$$\text{Final grade} = \min\left(10, \frac{E + C}{100}\right) * M$$

$$400 \leq E \leq 800$$
$$0 \leq C \leq 300$$



plan your customizable path now!

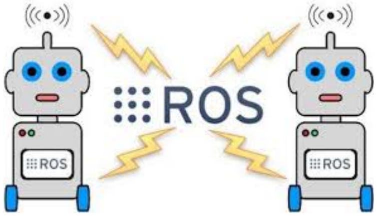
Mandatory assignments

With customizable extras

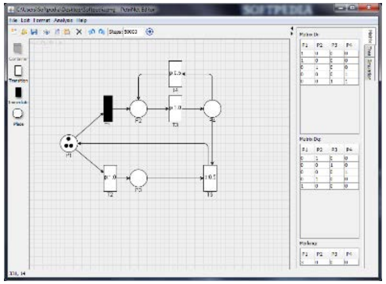
- Assignment 0
  - Paper reading; questions on BS

Pair programming

- Assignment 1 (Lab 1)
  - Measurement-based performance evaluation of ROS 2.0 communication



- Assignment 2 (Lab 2)
  - Behavior modeling and analysis using Petri-nets



- Assignment 3 (Lab 3)
  - Derive a petri-net model from a ROS application and analyze it

Projects

Customizable points

- Tool demo
  - pick an existing performance/modeling tool
  - evaluate it
  - report experience (in class, as report)

- Application study
  - pick existing application/software
  - model or evaluate it
  - report experience (in class, as report)

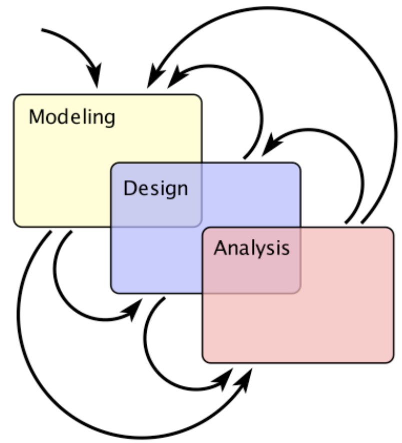
Get approval before starting!

Questions?

- Logistical issues ...

QEES

What is it about?



- Use models to design, analyze, and evaluate a system
- Compare alternatives
  - based on quantitative information
- Determine the impact of a feature on overall system performance
  - pin-point bottlenecks
- System tuning/optimization
  - find the best parameter settings

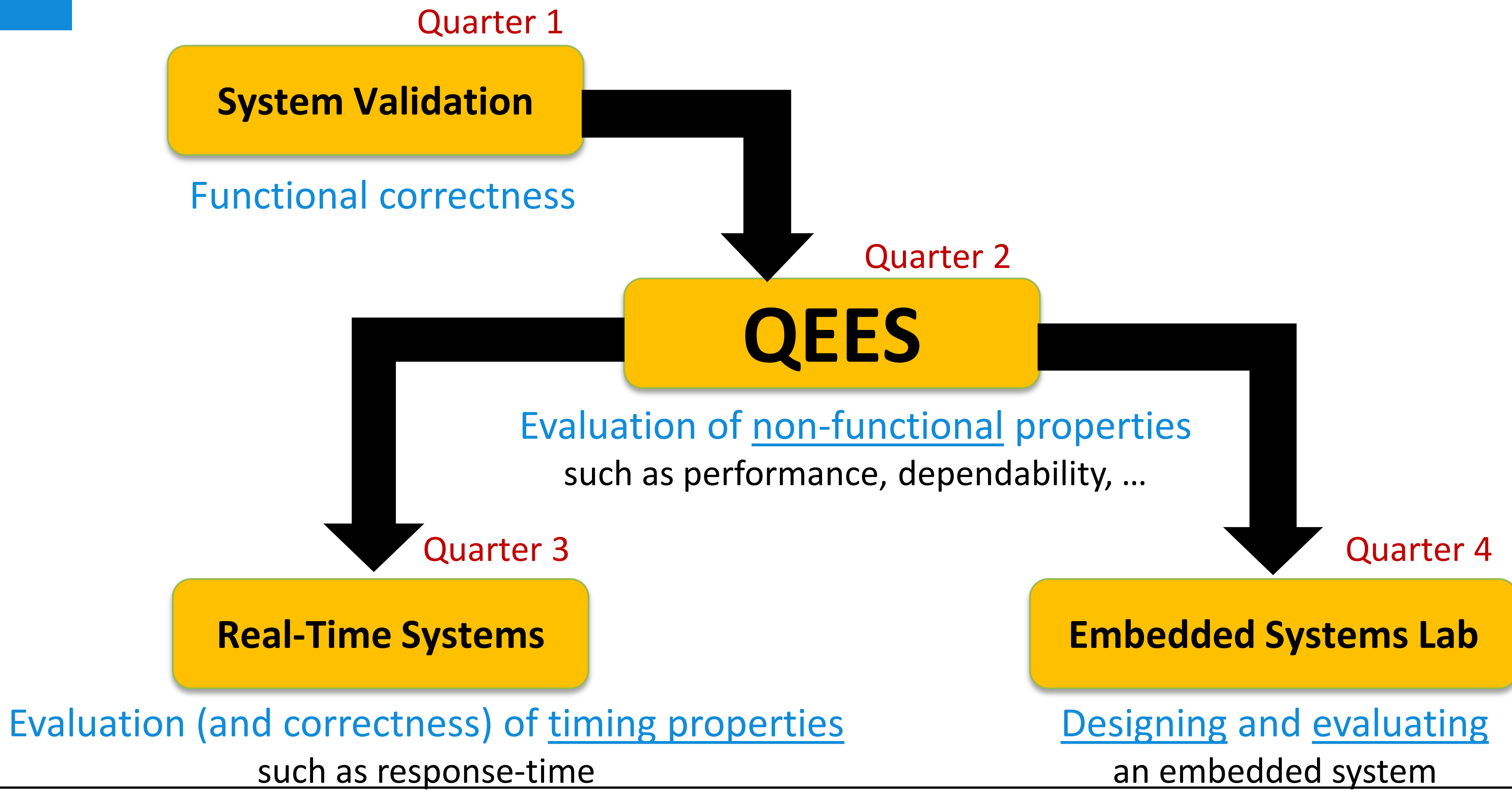
Example: The number of packets lost on two links was measured for our file sizes as shown below:

File Size	Link A	Link B
1000	5	10
1200	7	3
1300	3	0
50	0	1

Which link is better?

QEES

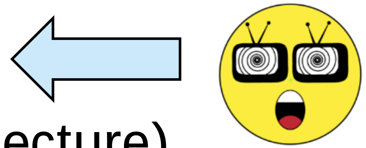
Where does it fit?



QEES

Which topics?

- Introduction to modeling and model-based design (1 lecture)
- Design of experiments (2 lectures)



- Measurement-based performance evaluation (1 lecture)
- Petri-nets and data-flow networks (2 lectures)
- Markov models (2 lectures + 1 Q&A)
- Queueing theory (2 lectures + 1 Q&A)

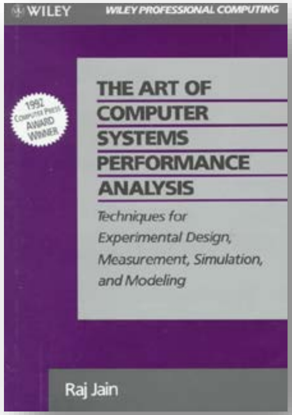
- Project presentations by students (1 lecture)

QEES

Course material

- Brightspace
  - videos
  - assignments
  - lab info + deadlines
  - old exams
  - reading list

- Books
  - The Art of Computer Systems Performance Analysis
  - Embedded System Design [Peter Marwedel]
  - Measuring Computer Performance: A Practitioner's Guide [David Lilja]



DEFINITIONS AND CONCEPTS

[Book]: Marwedel (chapter 1)

[Paper]: Basic Concepts and Taxonomy of Dependable and Secure Computing

What is an embedded system?

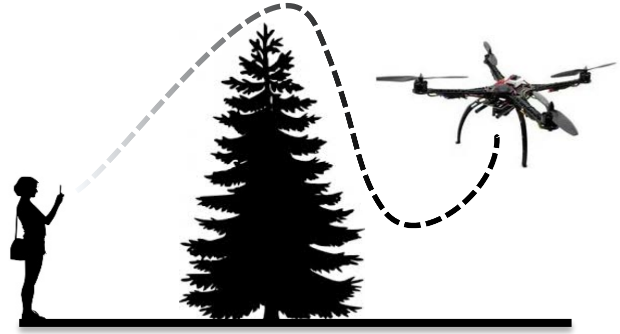
It is an information processing system that is embedded into an enclosing physical product.

Unlike a PC or servers, an embedded system "**interacts**" with its physical world.

Have you heard about Cyber-physical systems (CPS)?

[wiki] **CPS vs. ES:** A CPS is typically designed as a network of interacting elements with physical input and output instead of as standalone devices.

Concepts and definitions



System

A system is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, etc.

Function

The function of a system is what the system is intended to do and is described by the functional specification in terms of functionality and performance

Behavior

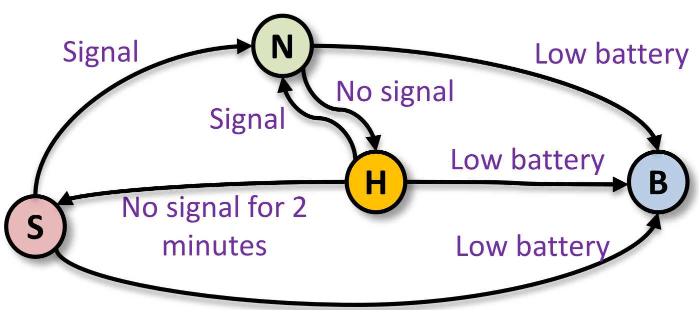
The behavior of a system is what the system does to implement its function. The behavior, for example, can be described by a sequence of states

Structure

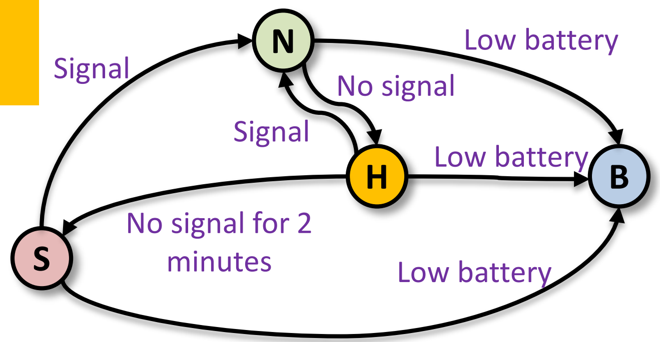
The structure of a system is what enables it to generate the behavior.

The boomerang drone's functions:

- **Normal mode** (receives signals from user and battery is not low)
  - Move up, down, left, right
  - Increase or decrease speed
  - Land
  - Take picture
  - Send picture
- **Hold mode** (no signal)
  - Stay still and wait for signal
- **Safe-return mode** (no signal from user for 2 minutes)
  - Follow the path back if there is no signal
  - Detect obstacles on the way
  - Avoid obstacles
- **Low battery mode** (battery is low)
  - Land safely if the battery is low



Behavior

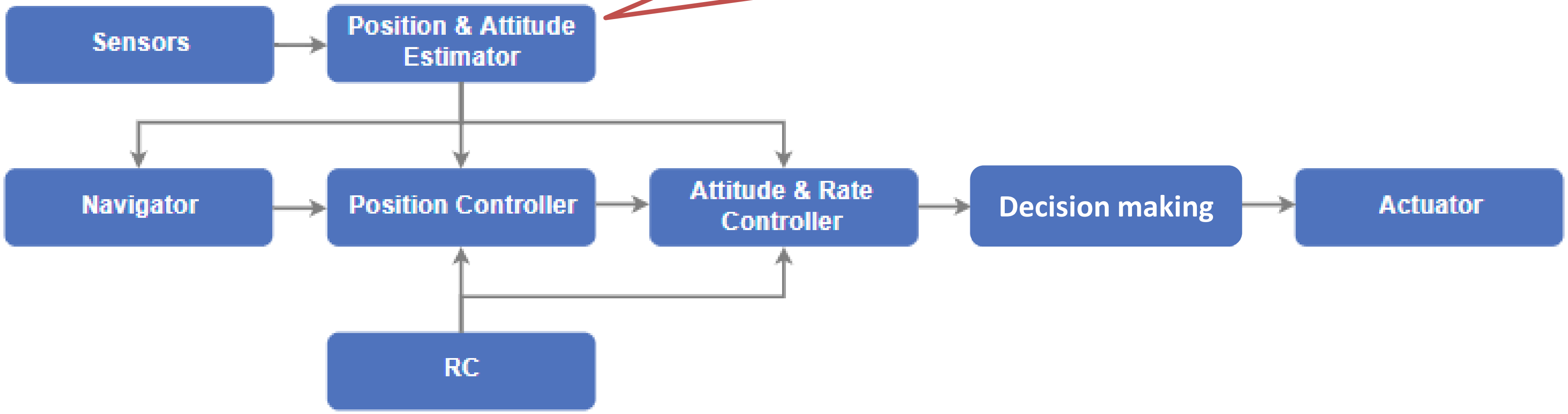


This is a **state diagram**, which is a "**model**" that describes how the system changes modes

Is this enough to describe the behavior?

This is a **component diagram**, which is a "**model**" to show the dependencies and interactions between SW/HW components

Flight control



Note: this diagram is symbolic and is not accurately model our prior example.

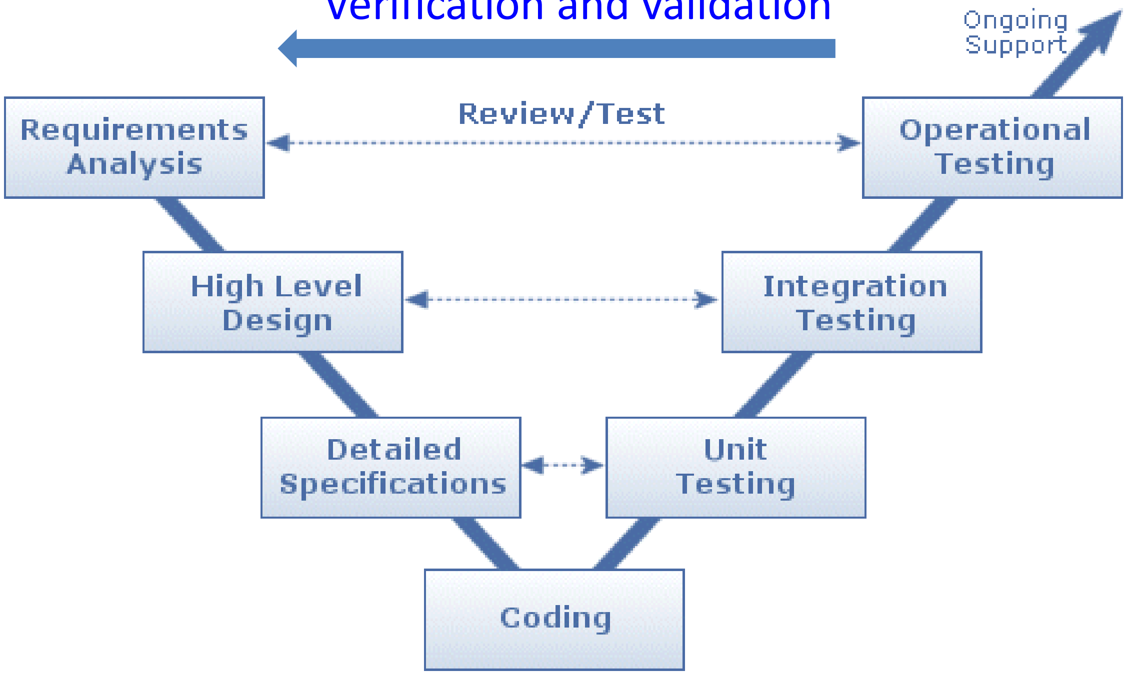
V-Model for system development

customer

requirements

End product

Verification and validation



Product development

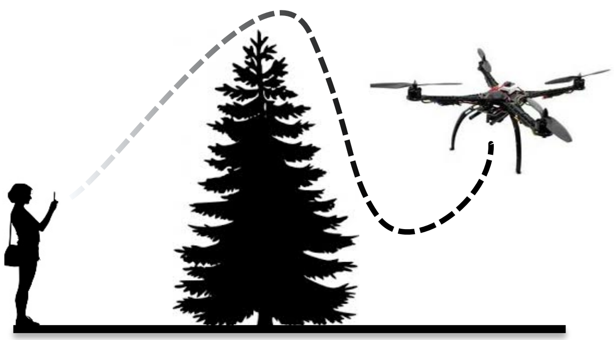
Specifications

Specifications
They describe the **functional** and **non-functional requirements** of the system

The boomerang drone's functions:

- **Normal mode** (receives signals from user and battery is not low)
  - Move up, down, left, right
  - Increase or decrease speed
  - Land
  - Take picture
  - Send picture
- **Hold mode** (no signal)
  - Stay still and wait for signal
- **Safe-return mode** (no signal from user for 2 minutes)
  - Follow the path back if there is no signal
  - Detect obstacles on the way
  - Avoid obstacles
- **Low battery mode** (battery is low)
  - Land safely if the battery is low

This is a (very informal) **functional specification**



Specifications

Specifications
They describe the **functional** and **non-functional requirements** of the system

[Wiki] A **non-functional requirement** (NFR) is a **requirement** that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

What examples do you have in mind for non-functional requirements?

Read more here: https://en.wikipedia.org/wiki/Non-functional_requirement

Quantitative properties

24h Active Use

97% Efficiency

3840x2160 pixels @ 100Hz

500m Range

9l per 100km

1Mh MTBF

Quantitative properties are key selling points
(next to the functional correctness)

Slide course (Bart Theelen): https://www.win.tue.nl/~pcuijper/QEES/Guest%20Lecture%20Bart%20Theelen.pdf

Quantitative properties in industry

- Print quality
- Throughput

- Engine performance
- Fuel consumption

- Overlay
- Throughput

- Image quality
- Responsiveness

Quantitative properties are key selling points
(next to the functional correctness)

Slide course (Bart Theelen): https://www.win.tue.nl/~pcuijper/QEES/Guest%20Lecture%20Bart%20Theelen.pdf

Examples of quantitative measures

- **Extrema** (worst/best case)
  - Maximum occupancy of a memory
  - Minimum time until next failure
  - Peak power consumption
  - Worst-case response time
  - Worst-case end-to-end delay
- **Reachability** (expected time until something happens)
  - Expected time until next failure
  - Expected time until first output is produced
- **Long-run average**
  - Processor load
  - Throughput of a communication network
  - Mean time between failures (MTBF)
  - Average power consumption

Slide course (Bart Theelen): https://www.win.tue.nl/~pcuijper/QEES/Guest%20Lecture%20Bart%20Theelen.pdf

Quantitative properties

Availability

Energy consumption

Response time

Dependability

Durability

Portability

Extensibility

Security

Performance

Privacy

Life time

Reliability

Safety

Understanding these requirements is what makes you a **valuable, highly-paid,** and **highly-wanted** engineer!

Quantitative properties are key selling points (next to the functional correctness)

Read more here: https://en.wikipedia.org/wiki/Non-functional_requirement

Quantitative properties

Availability

Energy consumption

Response time

Dependability

Durability

Portability

Extensibility

Security

Performance

Privacy

Life time

Reliability

Safety

Understanding these requirements is what makes you a **valuable, highly-paid,** and **highly-wanted** engineer!

Quantitative properties are key selling points
(next to the functional correctness)

Read more here: https://en.wikipedia.org/wiki/Non-functional_requirement

Dependability

Definition 1

The ability to deliver service that can justifiably be trusted

- The stress is on the need for justification (e.g., through rigorous evaluation or proof) of trust

Definition 2

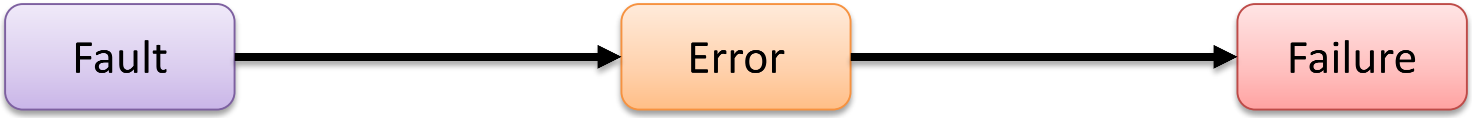
The ability to avoid service failures that are more frequent and more severe than is acceptable

"The dependability and security specification of a system must include the requirements for the attributes in terms of the acceptable frequency and severity of service failures for specified classes of faults and a given use environment. One or more attributes may not be required at all for a given system" [TPDS00].

[TPDS00] "Basic Concepts and Taxonomy of Dependable and Secure Computing", 2004.

Service/function failure

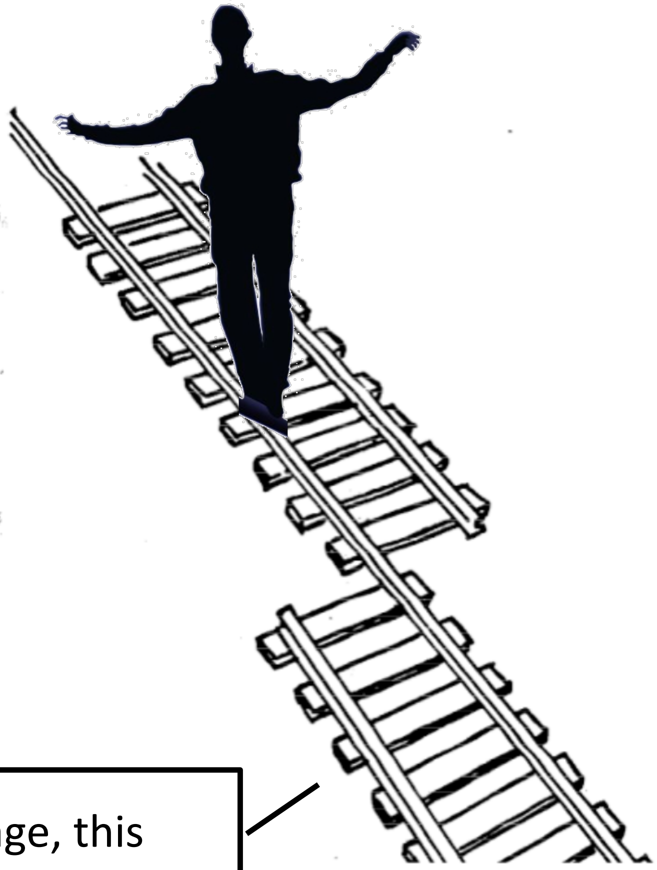
- **Fault** -- often referred to as Bug [TPDS00]
  - A static defect in software (incorrect lines of code) or hardware
  - A fault is the adjudged or hypothesized cause of an error
- **Error**
  - An incorrect internal state (**unobserved**)
  - An error is the part of total state of the system that may lead to its subsequent service failure
- **Failure**
  - External, incorrect behavior with respect to the expected behavior (**observed**)



[TPDS00] "Basic Concepts and Taxonomy of Dependable and Secure Computing", 2004.

Fault, error, and failure

What is this?
A fault? An error? Or a failure?



First we need to know the desired behavior

"A design without specifications cannot be right or wrong, it can only be surprising!"

[Lee and Seshia]

Under this usage, this implementation might not be a fault :-)

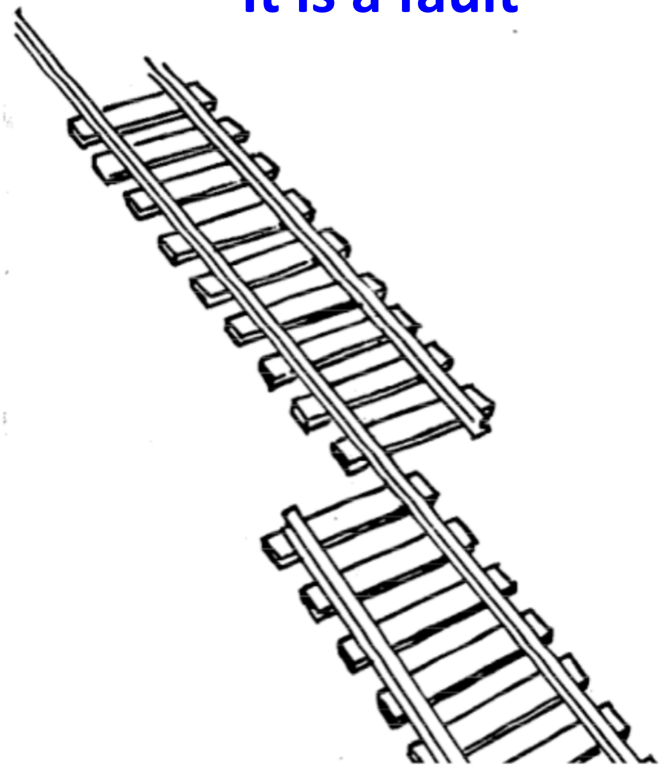
image: Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

Fault, error, and failure

What is this?
A fault? An error? Or a failure?

Assume it is a rail for a normal train

It is a fault

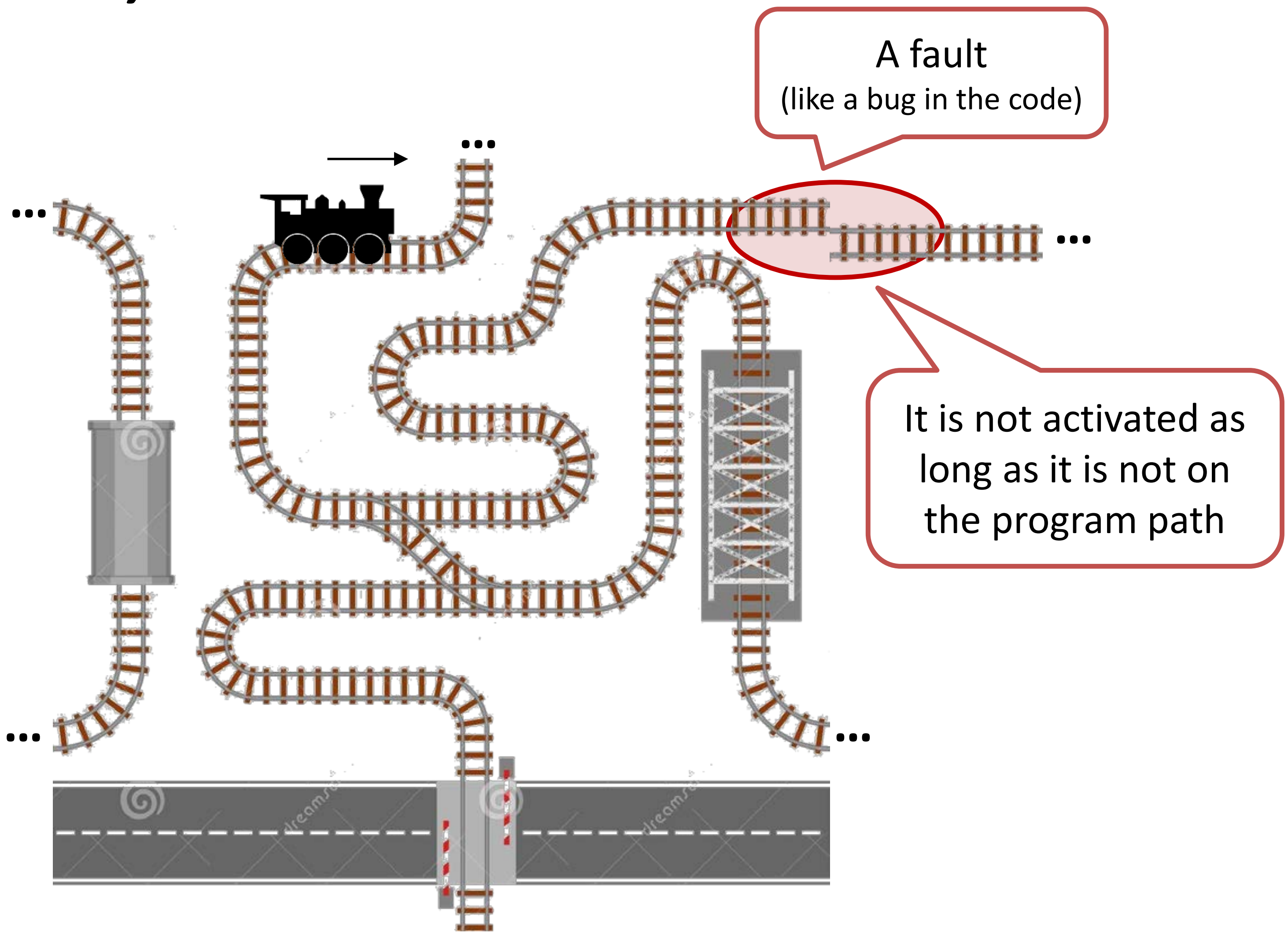


Fault examples:

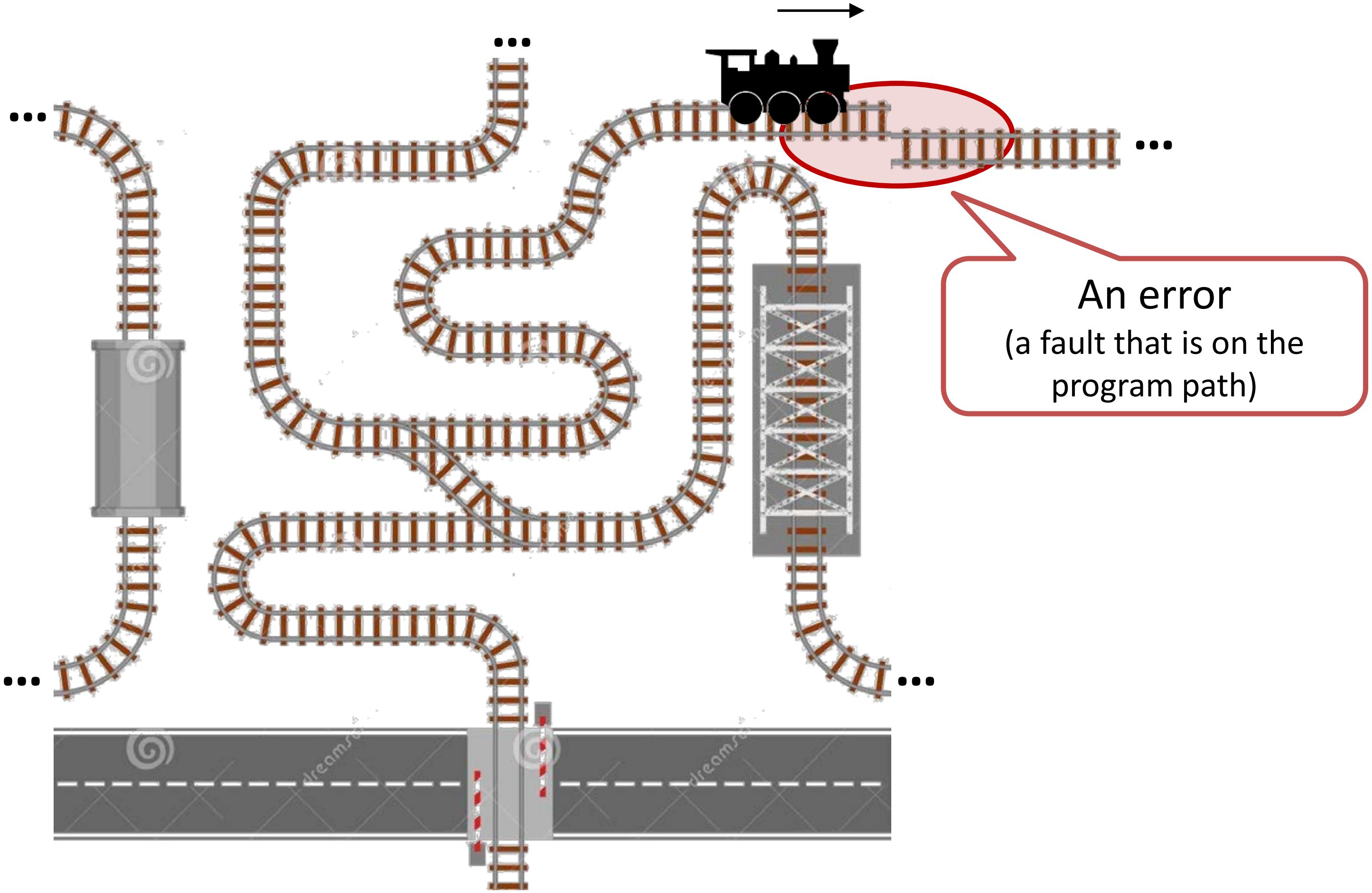
- Misconfiguration (e.g., misconfigured user permissions)
- Hardware faults (a memory cell that is always zero)
- Physical faults (caused by the environment)
  - At high radiation, memory bits may randomly flip
  - At high temperature, pressure sensor produces noisy data
  - In the tunnels, the GPS does not work

image: Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

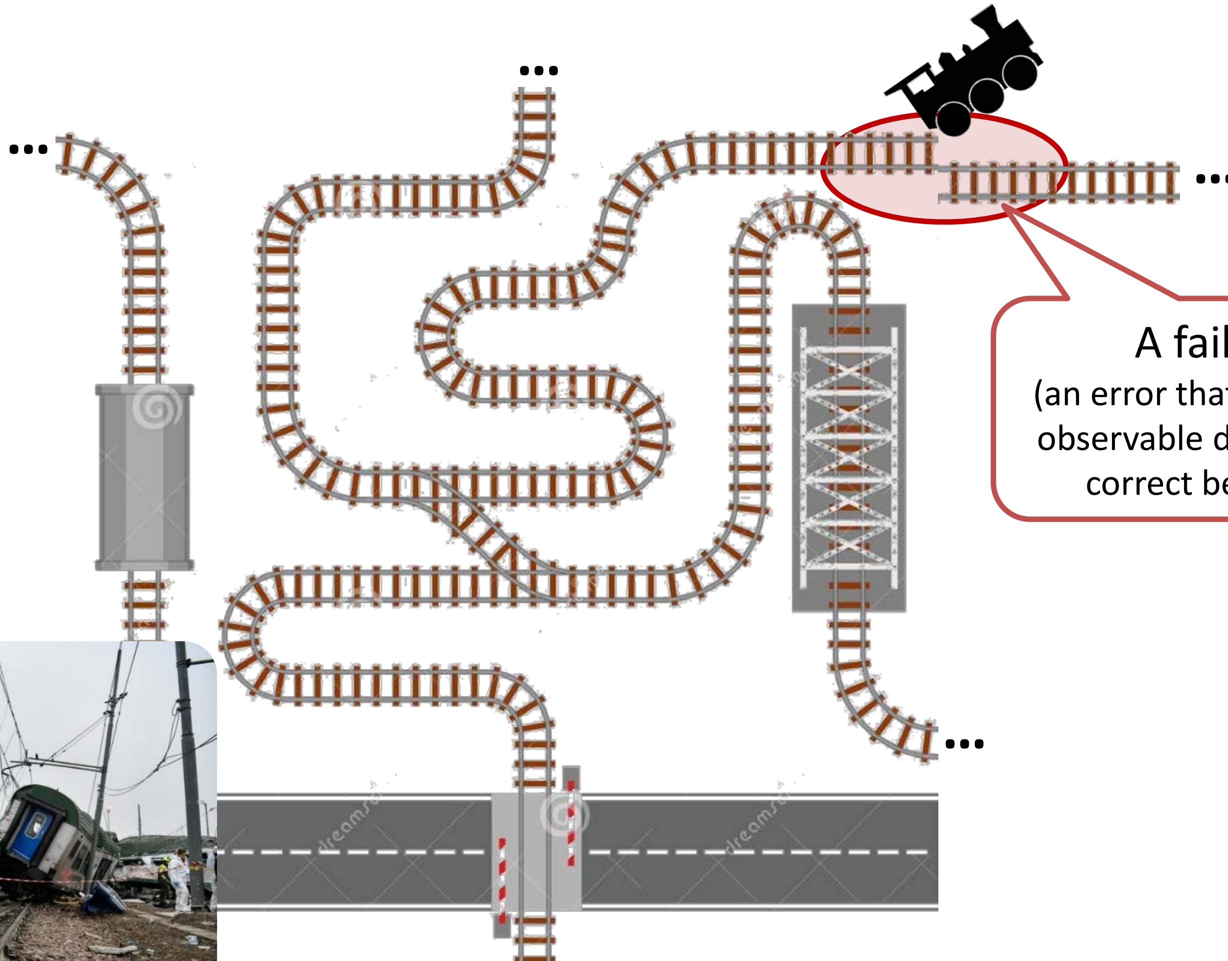
Fault, error, and failure



Fault, error, and failure



Fault, error, and failure



A failure
(an error that caused an observable deviation of correct behavior)

Addressing faults at different stages

Fault prevention
Better design, better tools, ….

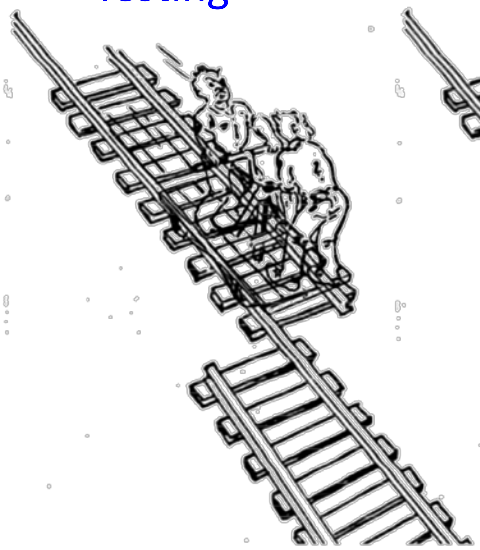
Fault detection
Testing, debugging, …

Fault removal
Fixing, patching, …

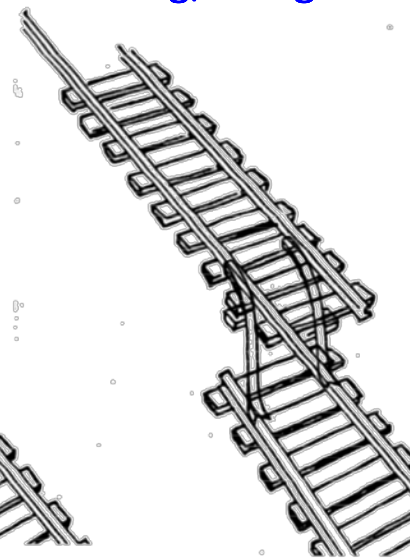
Fault tolerance
Redundancy, isolation, …

Fault forecasting
Estimating how many faults might still be there

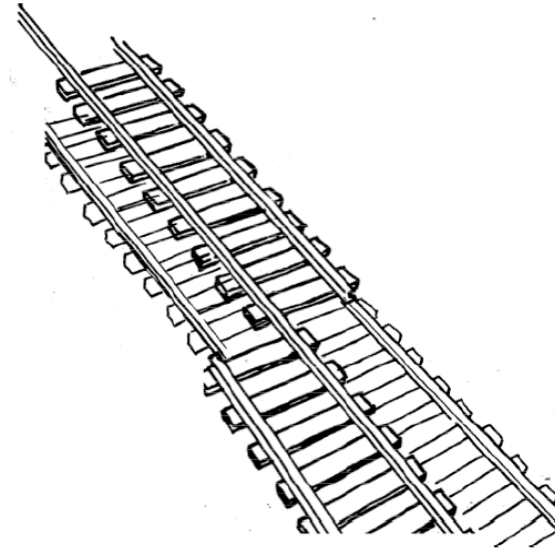
Testing



Patching/fixing



Redundancy



Bernd Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering: Using UML, Patters, and Java

MODELING AND MODEL-BASED DESIGN

Modeling, design, analysis

Modeling is the process of gaining a deeper understanding of a system through imitation.

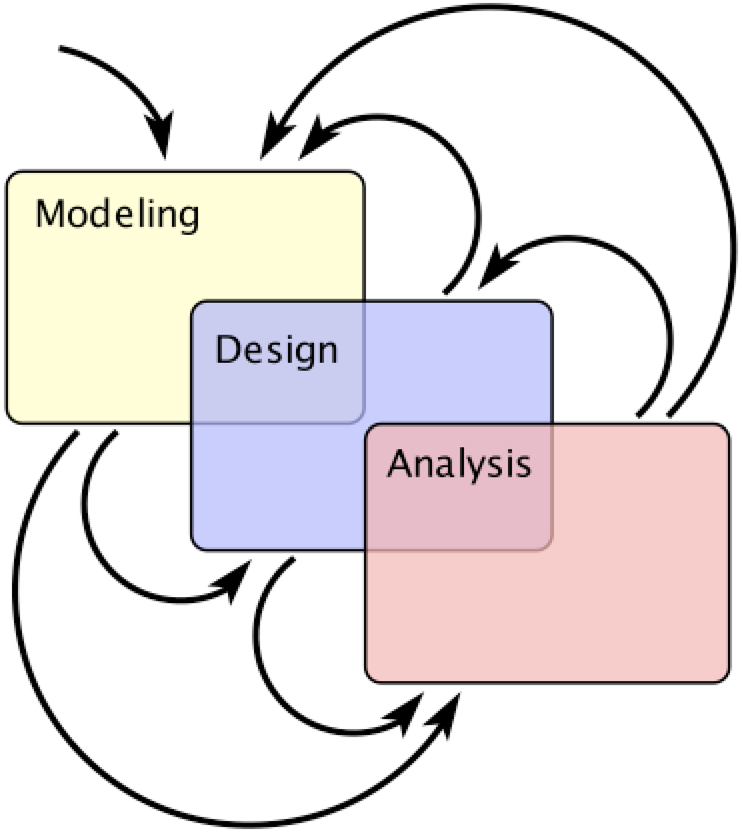
Models specify **what** a system does.

Design is the structured creation of artifacts.

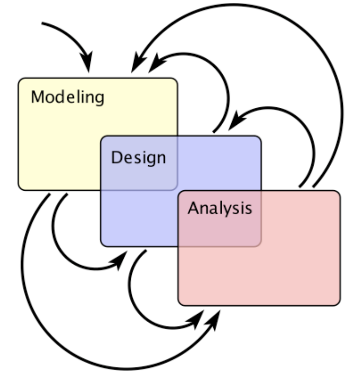
It specifies **how** a system does what it does. This includes optimization.

Analysis is the process of gaining a deeper understanding of a system through dissection.

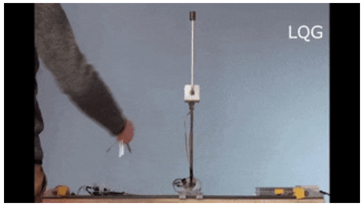
It specifies **why** a system does what it does (or fails to do what a model says it should do).



Modeling, design, analysis



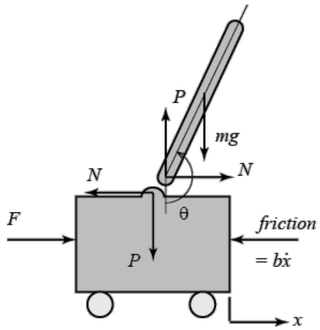
Real system



Models can be used to describe specification

Models can be used to design the system

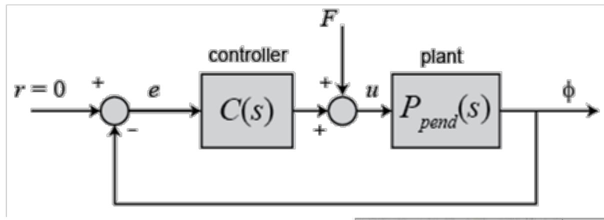
Models can be used to analyze the system

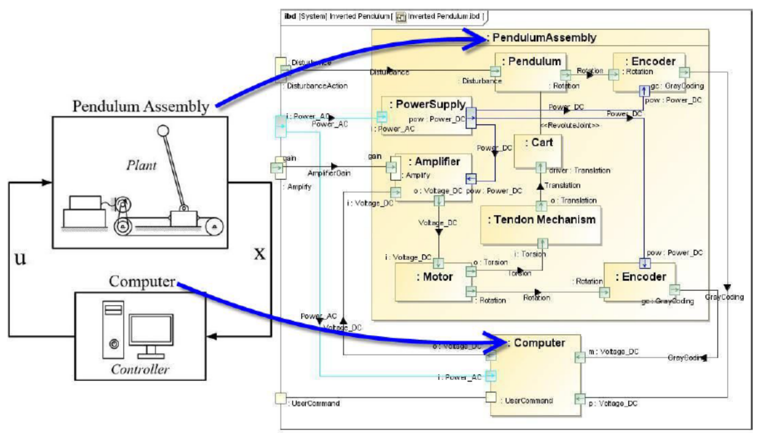


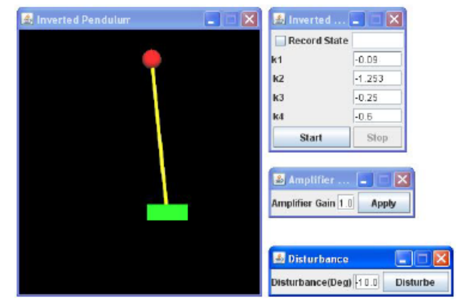
$$\ddot{x} = \frac{1}{M}\sum_{cart} F_x = \frac{1}{M}(F - N - b\dot{x})$$

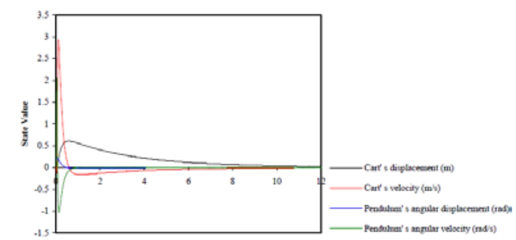
$$\ddot{\theta} = \frac{1}{I}\sum_{pend} \tau = \frac{1}{I}(-Nl\cos\theta - Pl\sin\theta)$$

This mathematical model expresses the physics of the plant







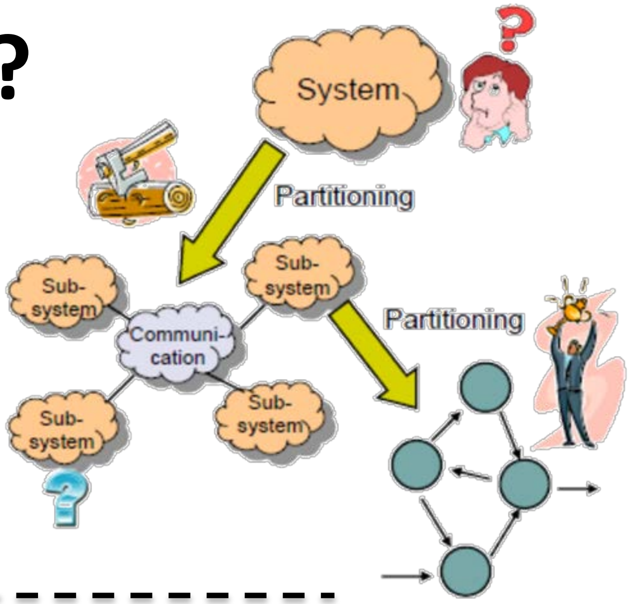


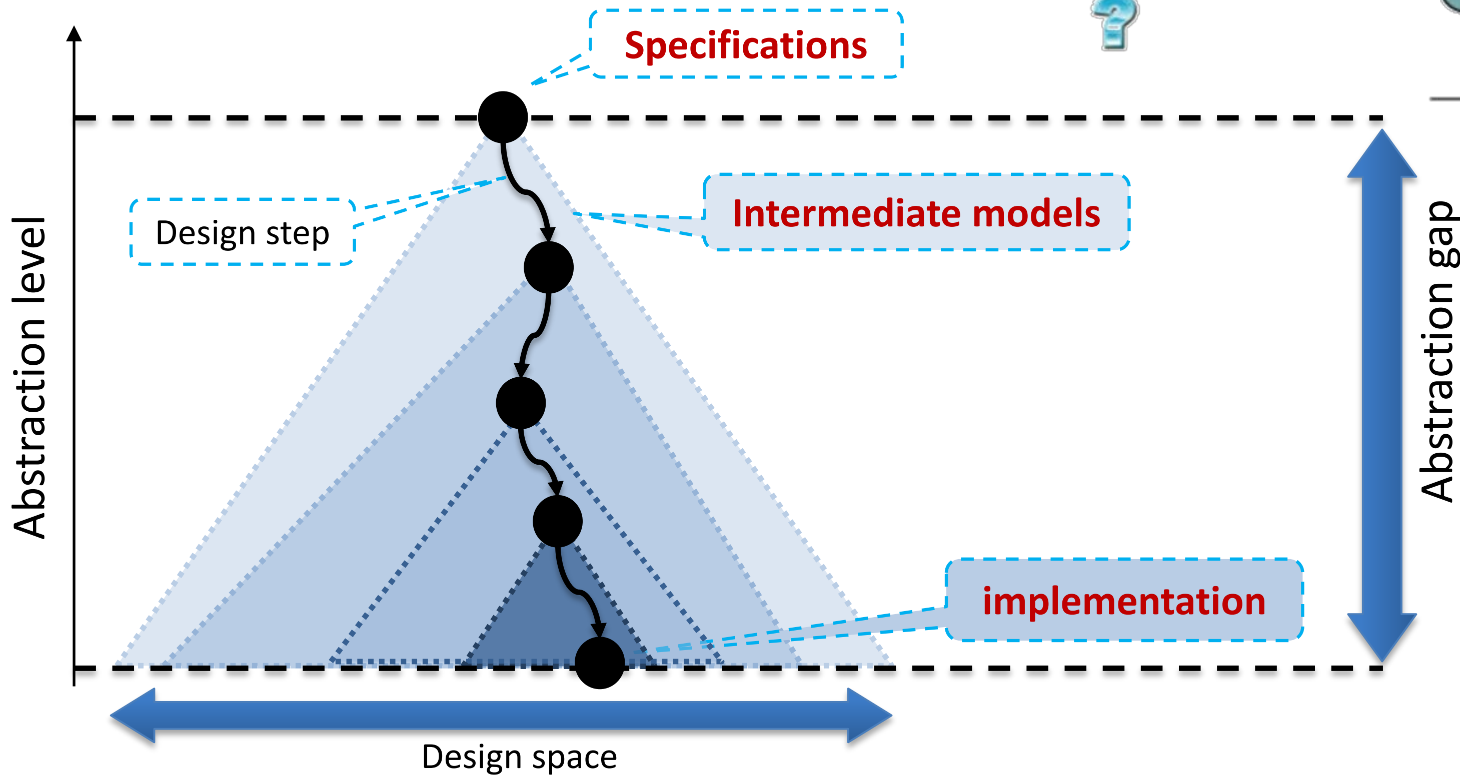
http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SimulinkModeling

https://www.nomagic.com/mbse/images/whitepapers/SysML_Inverted_Pendulum_System.pdf

Why is modeling important?

- Systems getting more complex
  - Abstraction is needed to understand the system or to design it
  - Partitioning into subsystems is needed





Why is modeling important?

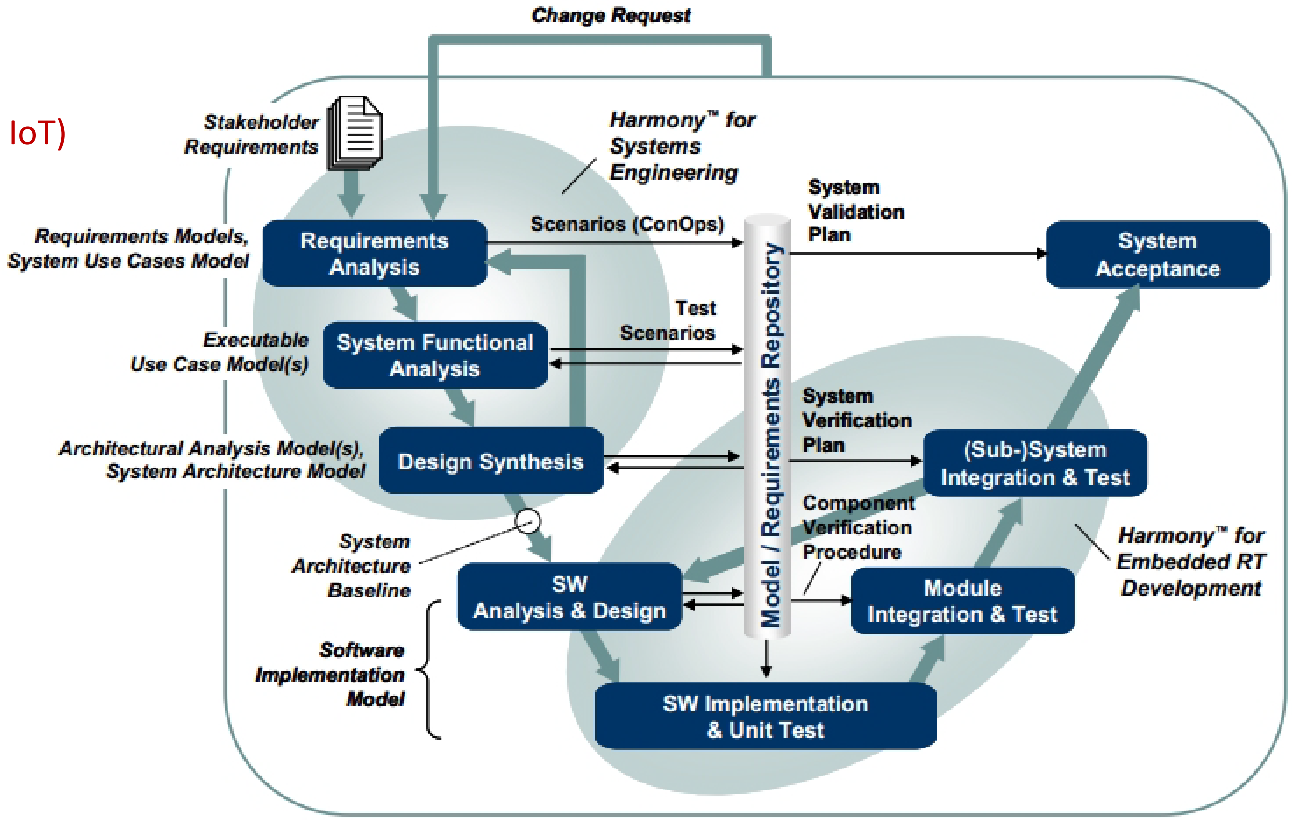
- **Models are a good base for communication between engineers**
  - Engineers think in diagrams
- **Models are important for documentation**
  - Standardized semantics
  - Formal language with (hopefully) only one meaning
- **Models abstract from the very detailed implementation**
  - Allow focusing on most important aspects
- **Models can be used when the system architecture is not yet ready at early design stages in order to:**
  - Evaluate functional and non-functional requirements
  - Find design faults

Model-driven v.s. model-based design

Model-driven design:
use models to automate the arrows

Model-based design:
use models to get a grip on the boxes

Example:
The IBM Harmony process (for IoT) is a model-driven design.



An interesting read

IBM Harmony: https://www.ibm.com/support/knowledgecenter/SSB2MU_8.3.0/com.btc.tcatg.user.doc/topics/atgreqcov_SecSysControllerHarmony.html

Other source: http://www.win.tue.nl/~pcuijper/docs/QEES/DF/QEES%20introcollege%202013-2014.pptx

When do we create a model?

To design a given specification

End
user

Development
team

Use modeling to design the system

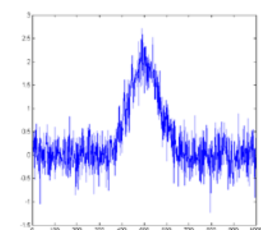
To understand, evaluate, analyze, or optimize a given system

End
user

Development
team

Use modeling to understand, analyze, or optimize the system

Quantitative evaluation and optimization



QEES

Modeling is key

- Introduction to modeling and model-based design (1 lecture)
- Design of experiments (2 lectures)
- Measurement-based performance evaluation (1 lecture)
- Petri-nets and data-flow networks (2 lectures)
- Markov models (2 lectures + 1 Q&A)
- Queueing theory (2 lectures + 1 Q&A)
- Project presentations by students (1 lecture)

Statistical modeling

Modeling concurrent programs
(design and performance analysis)

Modeling state-based programs and queues
(performance analysis)

Assignment 0

- Read the following paper (all sections)
  - "Basic Concepts and Taxonomy of Dependable and Secure Computing"
  - https://ieeexplore.ieee.org/document/1335465
- Take the quiz on Brightspace
  - Due date **Monday Nov. 15th**
- Notes.
  - There will be a **quiz** (with customizable points) from the first four chapters of the paper on **Thursday Nov. 18th**