# Next Week: Guest Lecture (via Zoom)

**Prof. Dr. Henning Femmer**

Alumnus of TUM and SE study program

Founder of Qualicen

Professor at Fachhochschule Südwestfalen

## Assisted Requirements Engineering

## What will remain in the hands of the Requirement Engineer in the future?

# Requirements Engineering

## Lecture 8

**Prof. Dr. Alexander Pretschner**

Chair of Software & Systems Engineering
TUM Department of Informatics
Technical University of Munich

## Orientation

### Recap:

- Lecture 7: Functional Requirements
- System Models, Use Cases, Function Hierarchies

### Coming up:

- Formalization: How to denote Functional Requirements?
- Application in the context of a model-based RE
- Tradeoff: Precision vs. Comprehensibility
- Tradeoff: Precision vs. Intentional Imprecision
- Requirements for Machine Learning

## Formalization

… by metrics
… by formulas
… by models with a formal semantics

## Discussion: Quality Metrics

*Can one set of quality metrics work for all kinds of software?*

# Interpretation of Statements on Properties

Statements on requirements serve:

- the professional communication between project participants
- the documentation
- for analysis by
  - People
  - Machines

Natural language is used for communication between people.

Essential prerequisites for Natural Languages are:

- Uniform understanding (same interpretation)
- Knowledge of the language or technical language

# Exemplary Statements on Requirements

- *The system should respond as soon as possible.*
- *Operating errors are largely excluded.*
- *The system is self-explanatory.*
- *The system is highly reliable.*
- *All processes in the system are traceable.*
- *The system is inexpensive.*

**Remember:**
Quality criteria for requirements!

# Problems: Precision and Explicitness

- *The system should respond as soon as possible.*
- *Operating errors are largely excluded.*
- *The system is self-explanatory.*
- *The system is highly reliable.*
- *All processes in the system are traceable.*
- *The system is inexpensive.*

**Remember:**
Quality criteria for requirements!

**Interpretation of the statements meaning:**

- Unambiguous: meaning independent of individual interpretation
- Comprehensible: target group can recognize and understand the meaning
- Precision: Clear identification of a system property
- Verifiable: For a system it can be observed / measured whether the formulated property is fulfilled

## Formalization - What is that?

**Definition Formalization:**

Formulation and Representation of content in a form independent of individual (subjective) interpretation and based on a given system of definitions (theory)

How to apply Formalization:

- Formal Language
- Semantics
  - Translation into reference systematics (logic, mathematics)
  - Rules for the transformation and deduction of further contents ("statements")
- Metrics

→ Typically, proven and prepared formalization concepts are used for formalization.

# Examples for Formalization Concepts

- Data modelling
  - Ontologies
  - Taxonomies
- **Logic**
  - Predicate logic
  - Temporal Logic
- **State Machines**
- Interfaces
- Architectures
- Metrics

# Formalization provides Precision

Natural Language:

- ambiguous, contextual (example: “Band")
- very complex

→ Informally described requirements are usually not precise and thus not free of subjective interpretation (Hence, the are ambiguous)

Formalization provides a more precise definition

- Formalization clearly defines the meaning, i.e., one interpretation of the requirement, which is at first only formulated in natural language, is chosen from a set of possible interpretations (Requires validation: ***Is this specification the desired one? OUCH!***)
- Choice of certain formalization concepts: This raises the question of comprehensibility and manageability.

→ Precise requirements can be better validated and analyzed.

**However:** Specification of system properties requires a precise definition of “system", or even a formal model for a System. And it requires precisely understood requirements!

## But once again: Do we really want this?

Formalization means precision, possibly unintended precision

Are requirements precise? Um.

## Discussion: Traffic Light State Machine

*A traffic light state machine is in the state “cyclic mode”, where it cycles between the 3 sub-states symbolizing the different lights. Now the traffic light switches to the second state “blinking”. When the traffic light now switches back again to the first state “cyclic mode” – in what sub-state is it now?*

# Formalization and Quality Requirements of IEEE 830

- Correct: better "valid", formulated vs. actual requirements (stakeholder expectations), requires precision
- Unambiguous: Formalization can help if it is understandable
- Complete: Formalization can help to uncover incompleteness
- Consistent: Consistency - formal logic consistency
- Verifiable: In mathematical sense only with formalization
- Modifiable: Change effects through formalization of visible
- Traceable: Formalization simplifies requirement relationships
- Ranked for Importance/Stability: Does not affect formalization

## Formulation of Statements

- Subjectivity: Related to the communication partners ("subjects")
  - Ability to formulate given statements precisely (for the producer) and to understand them (for the consumer)
- Objectivity: Related to a description system
  - Statements reduce subjectivity through methods of description (formalization) that are independent of subjective judgements
  - Attention: Objectivity leads to the problem of subjectivity by asking whether producers and consumers are capable of formulating and understanding in the description system.
- Intersubjectivity: An issue is equally recognizable and comprehensible for several viewers
  - There is agreement on how to perceive the facts, how to classify them and what they mean.

## The Tradeoff: Precision vs. Comprehensibility

**Precision** ("objective"):
How precise are Requirements formulated

→ Ultimately requires formalization

**VS.**

**Comprehensibility** ("subjective"):
How good and simple are requirements to understand

→ Depending on target group

**Caveat:**

- A very precisely formulated requirement can be difficult to understand (e.g., if the formalism used is not familiar to the target group)! A requirement can be misinterpreted or not understood.
- A requirement that is (apparently) very understandable from an individual point of view can be interpreted in different ways.
  → It can be misinterpreted and thus misunderstood.

## Empirically Formulated Statements on Properties

Statements about the properties of systems can also be formulated empirically:
→ Capturing the statements by observations of systems and their environment.
→ Verification by experiments and measurements possible.
→ Metrics!

**Examples:**

- *A system error occurs only once in 10 hours for a given operating situation.*
- *With 10 entries from a certain sample 9 usable answers are generated.*

**Difficulties / Problems:**

- Falsification of the properties by further refuting observations is always possible (except for static metrics, e.g., McCabe)

## Conclusion: Statements on Properties

**Formulation of Statements on Properties:**

- Subjective: No explicit model/description system
- Objective: Explicit model/description system
- Empirical: No mathematical prediction/explanatory model
  - Observations/Measurements
- Formalized: Mathematical prediction/explanatory model
- **Partially Combinable**

**Connection:**

- Statement about property used as a requirement for a system to be built
- Formalized requirements and agility?

# Formulation of Statements

**Informal:** Syntax and Semantics not formally defined

- Example: Natural language, sketches, informal diagrams, etc.

**Semi-formal:** Syntax formally defined

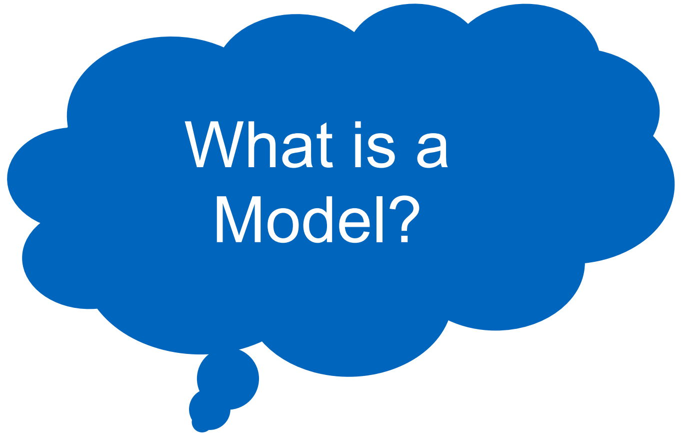
- Example: UML, SysML, Matlab Simulink

**Formal:** Syntax and Semantics formally defined

- Example: Propositional and predicate logic, temporal logic, Petri nets, $\lambda$-calculus, programming languages

Note: "Semantics" is a mapping of mathematical objects into other mathematical objects that seem to be more intuitively understandable

## The Term "Model" (Repetition)



**Attention:**

- Models are usually bound to a purpose, shortened images of a mental or real, existing or future reality.
- Models do not have to have a formal basis (i.e., defined with mathematical and computer tools), even if they are documented in a formally defined notation.

→ Both their meaning and their representation can also be handled informally.

# Formalization versus Modeling

Formalization Requires:

- A formal description language
  - A set of description/modeling concepts
  - Based on a specific system view

Models Require:

Implicit model building and is therefore

- An abstraction
- Only as correct as observations of reality correspond to those on the model (validity of a model)

Falsification of a model: Through observation, contradictions between model and reality are detected.

And sure: a language for modeling

## Model versus Reality

- A (formal) model is always an abstraction of reality

→ Formalization and modeling create an independent view of systems

→ Formalization only allows the formulation of certain properties that can be spoken about in the model.

**But:**

- Systems (even Software!) are ultimately part of our reality

→ Note difference between model and reality
(Danger of detachment from reality)

→ Always consider to what extent the model validly represents a property of the system.

→ Precision is sometimes unintentional and invalid!

## Discussion: Informal Models

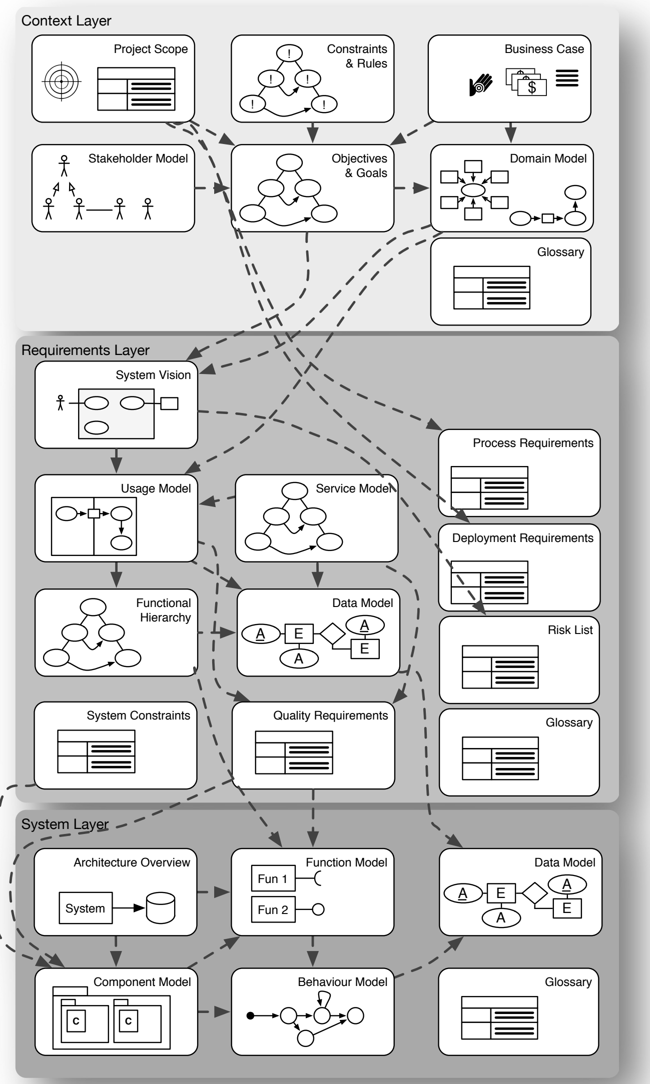
*Can you name models which are not formalized?*

*Are models already a formalization?*

# System Specification

Relevant results for the system specification of the RE are:

- Context model: description of the operating environment (neighboring systems, physical/technical environment, users)
- Data model
- Interface description
  - – Syntactic interface (based on the data model)
  - – Functionality (structured as function hierarchy)
  - – Interface behavior of the individual functions
    - as state machine
    - as interface assurances



## System Views

- States: Which states/transitions are there?
- Sequence/Interaction: What message/interaction sequences can there be?
- Data: What messages do the components exchange?
- Interfaces: Which messages go through which ports?
- Distribution: Which components are connected and how?

**Different formalisms exist for all views.**
**(See for example “formal lectures”)**

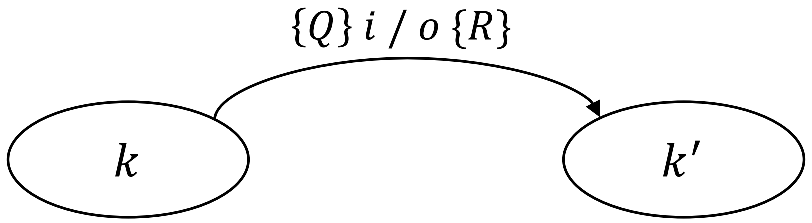
## State Machines (SM) with Inputs and Outputs

**Definition State Machine (SM):**

A nondeterministic finite SM $(\Delta, \Sigma_0)$ with *input alphabet* $I$ and *output alphabet* $O$ consists of a set of *states* $\Sigma$, *initial states* $\Sigma_0 \subseteq \Sigma$ (or one initial state $\sigma_0 \in \Sigma$) and a *state transition function*

$$\Delta: \Sigma \times I \rightarrow \wp(\Sigma \times O)$$

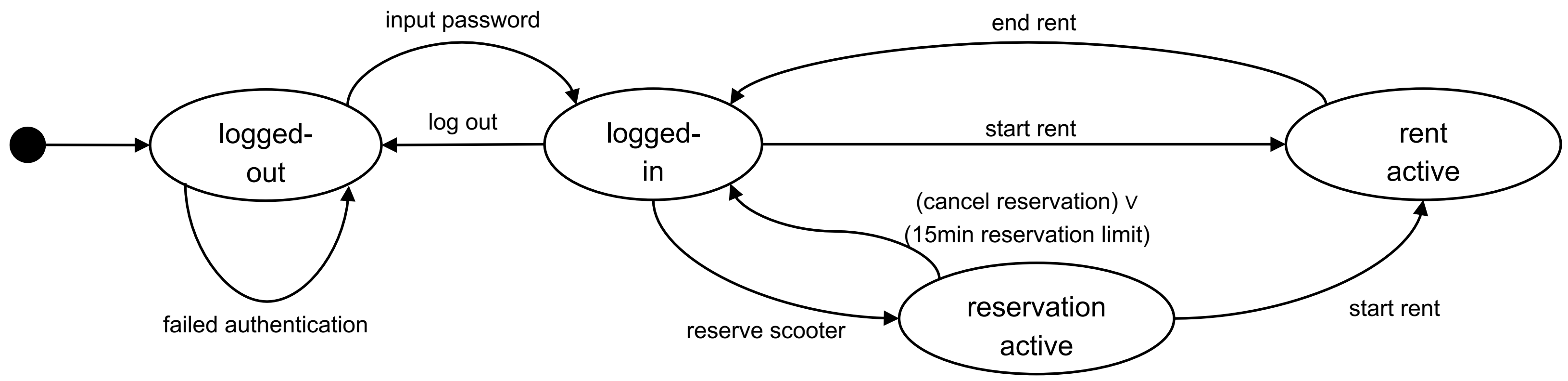
- Such SMs are also called *Mealy Machines*
  - Output depends on input and state
- Special case *Moore Machine*
  - Output depends only on the state
- Notation for transitions in diagrams



## From Requirements to State Machines (SM)

**Functional Requirement:**
A user needs to log into the App. After successfully logging in they can reserve a Scooter or directly start a rent. The reservation is limited to 15min and the user would need to reserve again after that time. If the user has an active reservation, they can also start the rent for that Scooter.

Example of a Mealy Machines FSM that formalizes the mentioned requirement:



## Mathematical Logic

- Formal Syntax: Definition of the statement formulation
  - Textual, graphical (formal languages, metamodels)
- Formal Semantics/Significance: Model Theory
  - Definition of a universe
  - Determines the interpretation of formulas: Validity
- Formal Derivative Theory: Calculations
  - Rules for transforming formulas: Derivation rules
    → Correctness: create only "true" formulas from "true" formulas
  - Concept of derivation (proof): Derivation trees

**Important Terms:**

- Correctness: What is derivable is valid
- Completeness: What is valid can be derived
- Consistency/non-contradiction

## Discussion: Syntax and Semantics

*What is the difference between Syntax and Semantics?*

# Example: Behavioral Requirements Using Logic

- **Informal (natural language):**
  *If no target vehicle is detected, the ACC switches to the state in which the vehicle speed is equal to the set target speed is adjusted.*

- **Semi-formal (controlled grammar):**
  *IF "Target vehicle does not exist" THEN "ACC in constant speed control state" AND "ACC" MUST "Adjust vehicle speed to set target speed"*

- **Formal (temporal logic):**
  $G(target\_vehicle\_not\_exists$
  $\rightarrow \mathbf{O}\ (acc\_const\_speed\_control) \wedge \Diamond(ego\_vehicle\_speed = target\_speed))$

## Motivation: Temporal Properties

If a user has reserved a scooter and starts the rent in the next timestep, the rent will eventually end in some future timestep

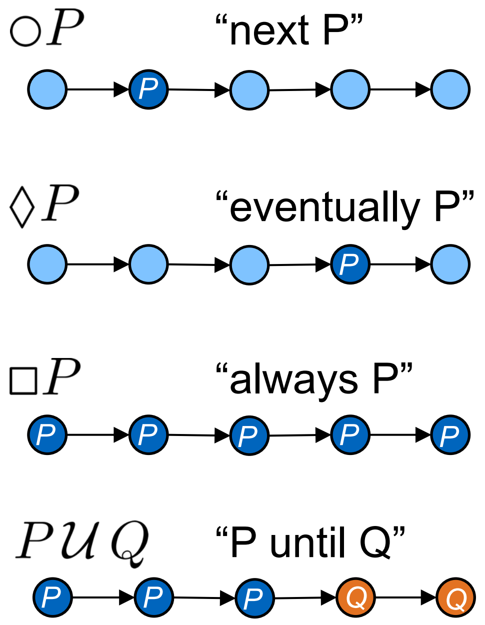
Temporal Logic is used to make statements about state-based systems:

- Statements about states
- Statements about executions

# Linear Time Temporal Logic (LTL)

Streams of states are sequentially ordered. Properties of streams can be described with predicates. Linear Time Temporal Logic (LTL) is a special method for formulating predicates over infinite state streams.

$$
\begin{aligned}
(\circ P)(s) \quad &\equiv \quad P(\mathrm{next}(s)) && \text{next} \\
(\Diamond P)(s) \quad &\equiv \quad \exists i \in \mathbb{N} : P\left(\mathrm{next}^i(s)\right) && \text{eventually} \\
(\Box P)(s) \quad &\equiv \quad \forall i \in \mathbb{N} : P\left(\mathrm{next}^i(s)\right) && \text{always} \\
(P \,\mathcal{U}\, Q)(s) \quad &\equiv \quad \exists i \in \mathbb{N} : \forall j \in \mathbb{N} : (j < i \Rightarrow && \text{until} \\
&\qquad P(\mathrm{next}^j(s))) \wedge Q(\mathrm{next}^i(s))
\end{aligned}
$$



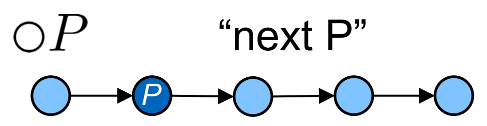
$$\text{where } \mathrm{next}^0(s) = s \text{ and } \mathrm{next}^{i+1}(s) = \mathrm{next}(\mathrm{next}^i(s))$$

# Concept: Next-Operator

Goal: Description of at the next time step valid properties

Concept: Next-Property

- Notation: $\bigcirc P$ ("Next P")
- Interpretation: Property P is valid at the next time step, P is valid in the next state.



Example: "The ACC constant speed control is activated in the next time step"

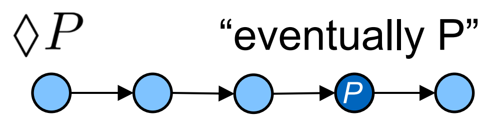
$\equiv \bigcirc\ \text{acc\_const\_speed\_control}$

# Concept: Eventually-Operator

Goal: Description of in the future valid properties

Concept: Eventually-Property

- Notation: $\Diamond P$ ("Eventually P", "Finally P")
- Interpretation: At one point in time after the current point Property P is valid, P is valid somewhere on the subsequent path.



Example: "The ego vehicle speed will become the selected target speed at a future time"

$\equiv \Diamond(\text{ego\_vehicle\_speed} = \text{target\_speed})$

# Concept: Always-Operator

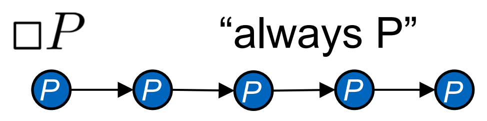
Goal: Description of always valid properties

Concept: Always-Property

- Notation: $\Box P$ ("Always P", "Globally P")
- Interpretation: From now on, P is always valid.

Example: "When a user has started a scooter rental, the rental always has to end."

$\equiv \Box(\text{rent\_active} \Rightarrow \Diamond \text{rent\_ends})$

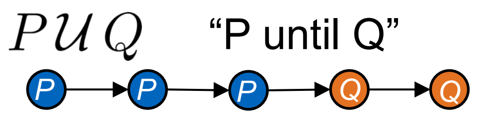


## Concept: Until-Operator

Goal: Binary Operator that describes relations between two properties followed by each other

Concept: Until-Property

- Notation: $P \mathcal{U} Q$ ("P until Q")
- Interpretation: P has to hold at least until Q becomes true.



Example: "If a user fails to authenticate, he is in the state "logged-out" until he correctly inputs his password and gets into the state "logged-in"

$\equiv \text{logged\_out} \; \mathcal{U} \; \text{logged\_in}$

## Discussion: Linear Time Temporal Logic (LTL)

*How would you denote "When the emergency brake is activated, then the scooter will stop at some future point"*

# Discussion: Eventuality Operator

*What other properties can be described with the unbounded eventuality (◊) operator?*

## Discussion: Why do all this?

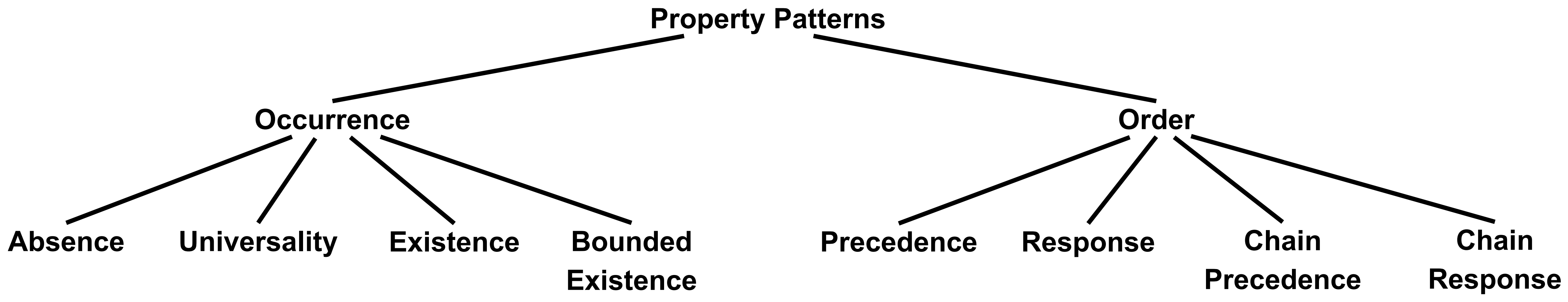
*Why would we want to formalize requirements at all? When would we want to formalize them in this way?*

# Forms of Formalized Requirements using Logic

- General predicate logic without reference to general system modeling concepts:
  - Example: Crash ⇒ Airbag
- Rudimentary system terms (time, space, sequence)
  - Example of temporal logic: $\Box(\mathrm{Crash} \Rightarrow \Diamond \mathrm{Airbag})$
    The following always applies: A crash leads to the airbag being triggered (at some point in the future).
- Formulated system model (system boundary, interface behavior)
  - Example:
    Be *crash_sensor_out* and *airbag_activation_in* channels and be
    the statement *m:c@t±s* the statement message *m* on channel *c* at time *t* (if s ≠ in interval [t-s, t+s]) measured in *msec*.

    crash_signal:crash_sensor_out@t ⇒
    act_airbag_Signal:airbag_activation_int@t+160±20

## Specification Templates (According to set Patterns)

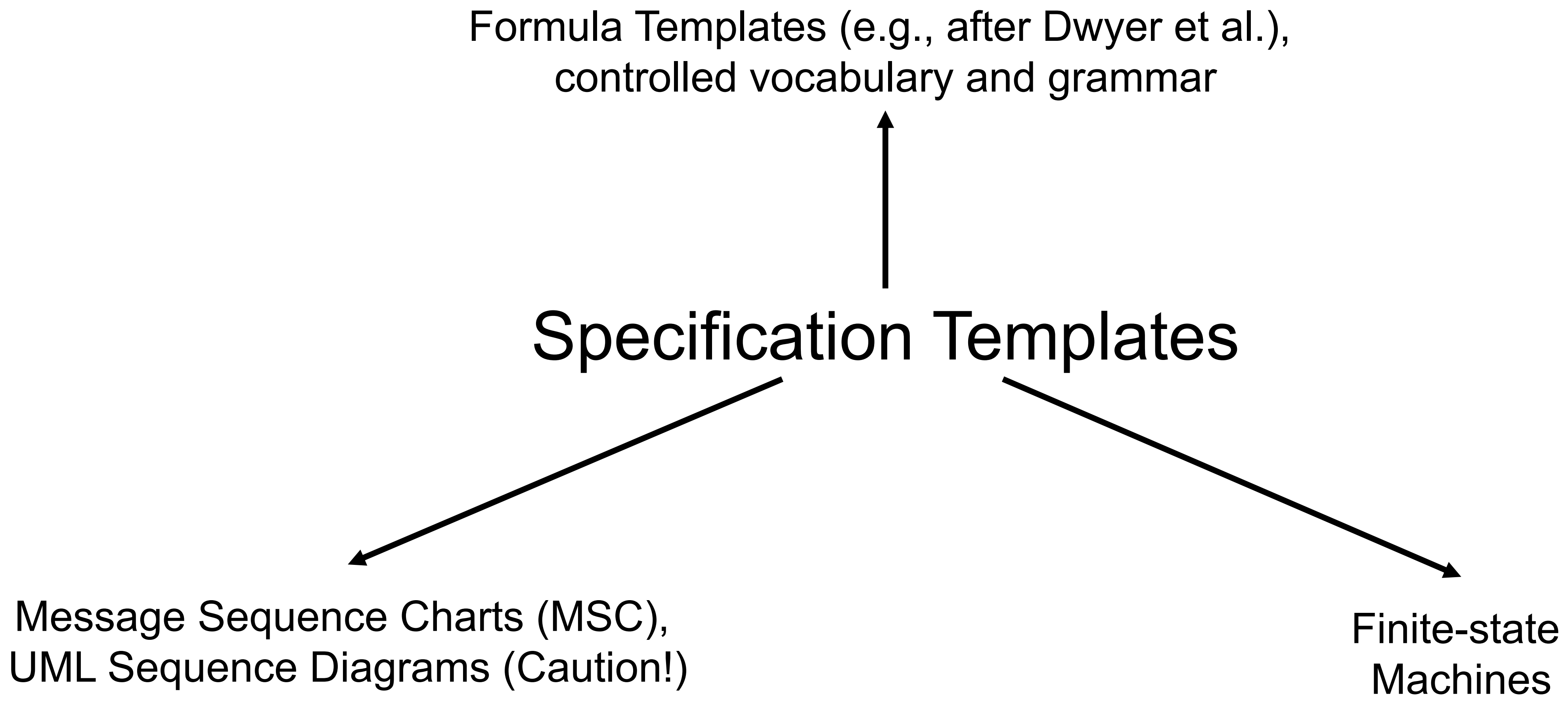


Dwyer et al. state:

Most of the specifications correspond to the following categories:

- Existence: Certain event must occur at least once
- Absence: Certain event must not occur
- Precedence: Event Y comes after event X
- Response: Event Y is the response to event X

## Alternatives for Templates

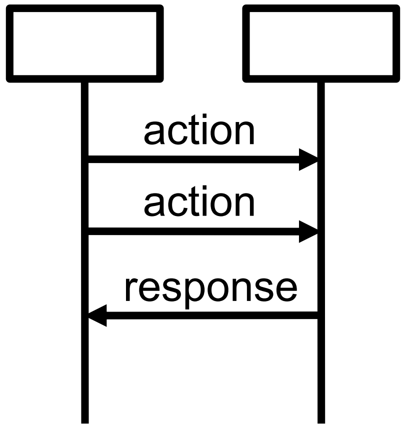

Formula Templates (e.g., after Dwyer et al.), controlled vocabulary and grammar
Specification Templates
Message Sequence Charts (MSC), UML Sequence Diagrams (Caution!)
Finite-state Machines

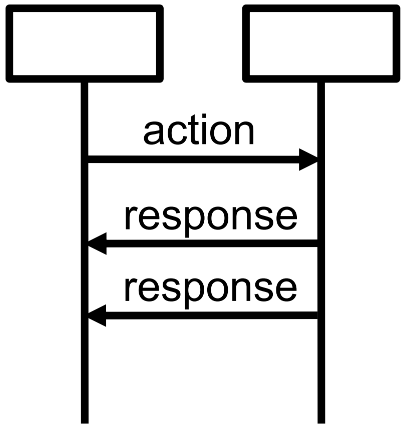
## Specification Pattern "if..., then..."

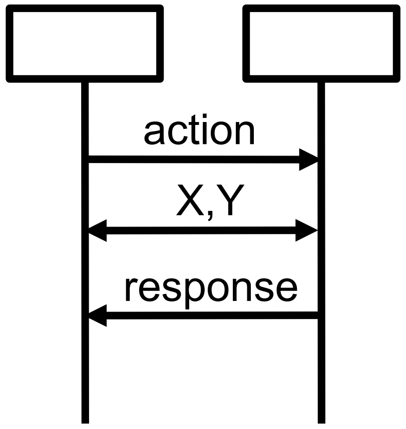
**Template:** If *A*, then *B* - discussion as regular expressions

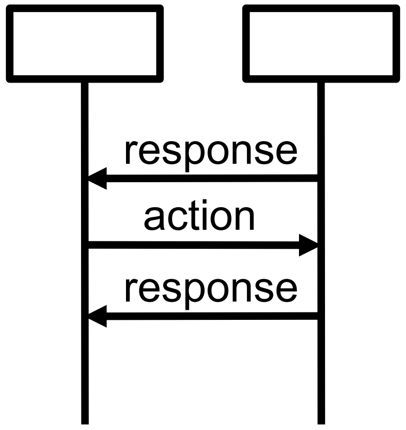
- Is the sequence *(A+)B* allowed?
- Is the sequence *AB+* allowed?
- Is *B* allowed without *A?*
- Is A required for B: Is the sequence *A*B* allowed?
- Is the sequence *BAB* allowed?
- Does a second *A* have to be followed by a second *B*?

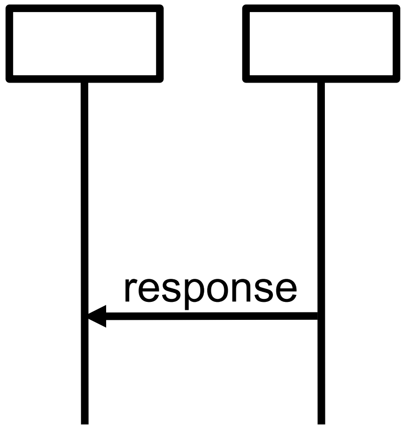
# Further Specification Templates as Message Sequence Charts

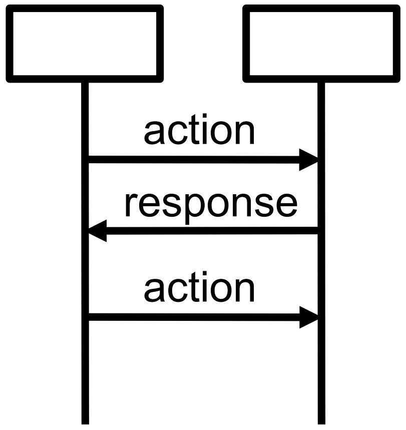












Which processes are allowed?

# Advantages and Disadvantages of Formalization

**Pro/Strengths:**

- Precise, clear description
- Automation
  - Model testing
  - Test case generation
  - Program generation
- Clear terms for
  - Consistency
  - Completeness
- Rules for derivation
- Traceability
- Verifiability with formal proof

**Contra/Weaknesses:**

- Limited expressiveness
  - Abstraction bound to a purpose
- Poor intelligibility
  - Fear of contact
- If necessary unintended precision
- High initial outlay
  - Costs for training
  - Experience
- Modification may be difficult
- Poor scalability

## Discussion: System Formalization

*Given the "Juicer" subsystem of the scooter App, where a person collects several out of charge scooters, brings them to a charging station, and redistributes charged scooters on positions calculated by the app, would you formalize this system?*

*Which kind of formalization would you choose?*

## Requirements for Machine Learning

**Definition Machine Learning (ML):**

Machine Learning (ML) generates outputs based on statistical correlations to training data. The input-output pairs of a ML System are thus not specifically programmed but learned through sample data.

Machine Learning:

- Used in different applications like computer vision, speech recognition, medical diagnosis
- Different ML algorithms exist such as SVM, Decision Trees, Linear Regression, Bayesian Networks, Genetic Algorithms, Neural Networks, Deep Learning
- Learning through optimization: Different training algorithms like evolutionary algorithms, gradient descent, information gain, expectation–maximization

**Problem:** How to define Functional and Non-Functional Requirements for ML Systems?

## Repetition: Problem and Solution

Machine Learning is the solution, not the problem

**Requirements for ML Systems are hard to specify:**

- Functional Requirement Example: “Detect all pedestrians in the image”
- If we could formally (rule based) specify the output of an ML system, why would we need a ML system in the first place?
- What is the contradiction between the usage of ML and the demand of precise and unambiguous system requirements?

→ Machine Learning has no formal and exact specification

## Non-Functional Requirements for Machine Learning

Different types of non-functional requirements for ML exist:

- Fairness
- Transparency
- Privacy
- Security
- Testability

Other qualities of ML models:

- Correctness/Performance: Metrics like Precision, Recall, Average Precision, Intersection over Union
- Computational Efficiency; energy: Is the model lightweight or do we need a GPU server to run inference?
- Interpretability: How does the model work? Why did the model come to that conclusion?
- Robustness: Is the model prone to attacks and does it work in new/unusual environments?

# Performance Requirements for Machine Learning

How can we show that a specific functional requirement is fulfilled?

- Test for performance metrics (90% accuracy)
- Test on the collected dataset might be problematic
- Need to avoid overfitting on training data with validation techniques (validation set, cross validation)

How can we show that a specific non-functional requirement is fulfilled?

- Runtime performance in Deep Learning influenced by model size (trade off between requirements → slow response means high accuracy)
- Training != Using

# Fairness Requirements for Machine Learning

As ML decisions are often difficult to understand fairness is a recently emerging non-functional Requirement:

- Fairness Through Unawareness: An Algorithm is fair if it doesn’t use "protected attributes" for decision making. (Delete protected attributes from train/test datasets. Correlations?)
- Counter-Factual Fairness: A ML model is fair if the output remains the same if the protected attribute is flipped to a counter-factual value
- Individual Fairness: A ML model should give similar predictions among similar individuals according to some distance measure.

Some sensitive features in the data are also present through Proxies and fairness cannot be ensured just by not using sensitive features. (e.g., the race of a person might correlate to the neighborhood it lives in)

## Performance Metrics for ML models

**Example Binary Classification:**

A binary classifier is trained to predict weather an item is class A or class B. Therefore, it predicts for each item a value between 0 and 1. By choosing different threshold values (e.g., $\beta = 0.3$ or $\beta = 0.4$) the output of the classifier can be altered.

A classifiers performance for a certain $\beta$ can be described with a confusion matrix:

| | | True Class | |
|---|---|---|---|
| | | Class A | Class B |
| Predicted Class | Predicted Class A | True Positive | False Positive |
| | Predicted Class B | False Negative | True Negative |

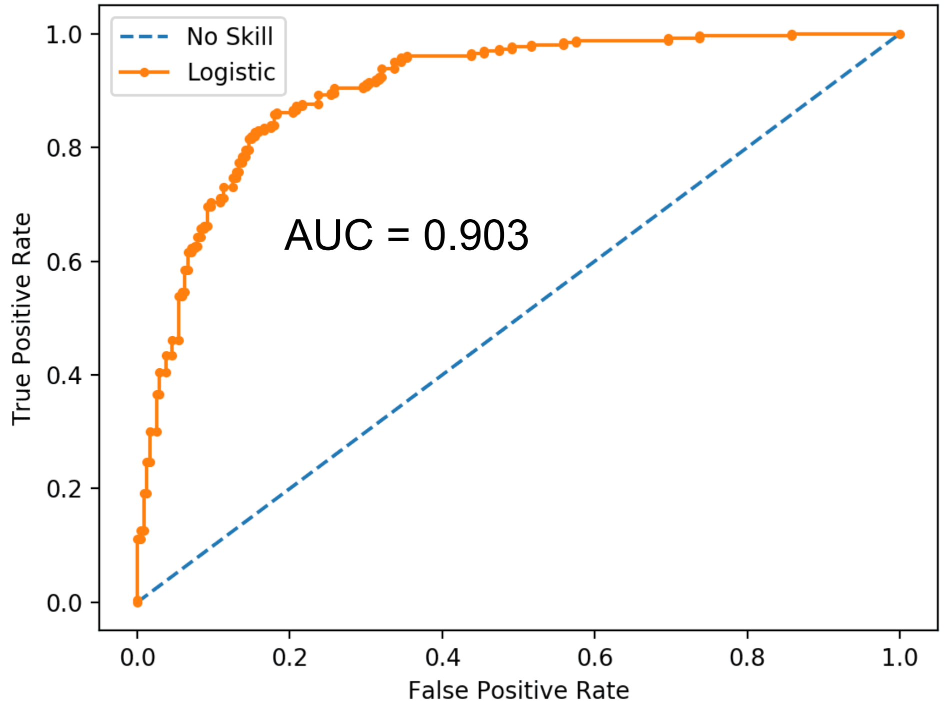
True Positive Rate: $TPR = \frac{TP}{TP+FN}$

False Positive Rate: $FPR = \frac{FP}{FP+TP}$

Accuracy: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$

# Performance Metrics for ML models

- With changing the classification threshold $\beta$ we can adjust the trade of between the true positive rate and the false positive rate of our model.
- For obtaining a general metric, independent of the classifiers setting $\beta$, of "how good" the model is, we measure the model's performance for many $\beta$ values.
- For each setting we note $TPR$ and $FPR$ values and plot them in a graph.
- The area under the curve (AUC) of this graph is a more general ML performance metric.
- Similarly, one can plot Precision and Recall values.



## Formulating Machine Learning Requirements

**Clear Separation of Problem and Solution:**

- Should we specify requirements differently just because it is for ML?
- High level natural language requirements work for ML
- Detailed, specific rule-based requirements not possible for ML
- Detailed requirements over input, output pairs possible (data set)

→ Having this Separation of Problem and Solution would be good
→ However, this doesn't always work

## Summary

**Formalization in model-based RE supported:**

- Precise formulation and interpretation of statements on required system properties
- Clear terminology and Traceability
- Possibilities for automation
- Verifiability with formal proof

**However, the following applies to formalization**

- Necessity of an underlying system model and associated with it possibly a limited expressiveness
- High effort for creation and modification of formal models

→ Trade-off between costs and benefits necessary!

**Machine Learning doesn’t have a specification**

## Outline and Outlook

Terms and Definitions

Context-specific nature of SE and RE

Quality models for requirements

Engineering Models

Stakeholder and Requirements Elicitation

Goals and Goal-oriented RE

Non-functional Requirements

Functional Requirements

Formalization

Agile Processes

Requirements Management and Quality Assurance

Trends in Research