Challenges in the Modeling and Quantitative Analysis of Safety-Critical Automotive Systems

University of Konstanz

Florian Leitner-Fischer
florian.leitner@uni-konstanz.de

26.03.2011 @ ROCKS 2011

Joint work with

Matthias Kuntz and Bernd Reh
TRW Automotive

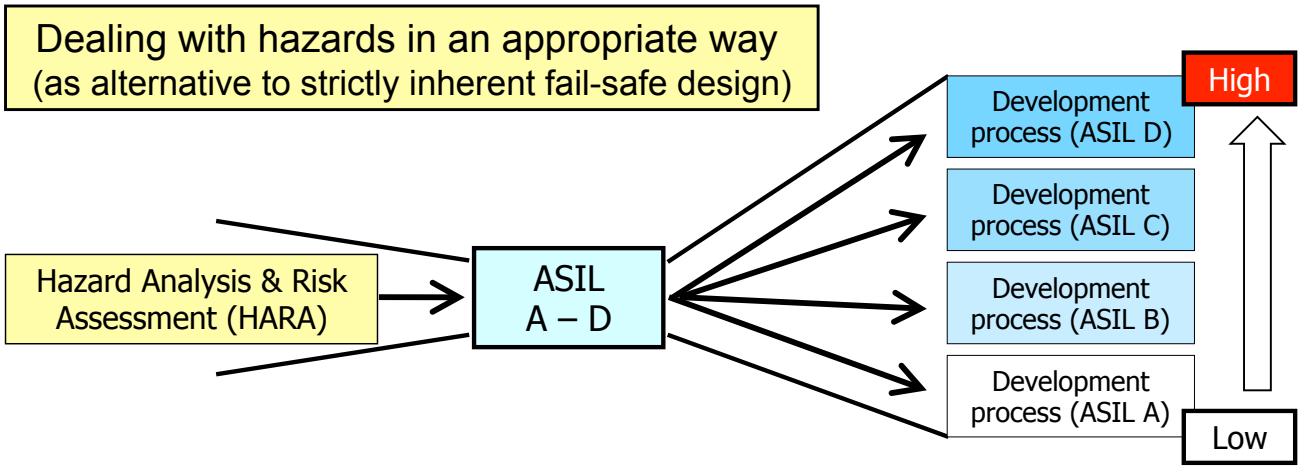
Stefan Leue
Univeristy of Konstanz

software engineering

Outline

- Safety-Critical Automotive Systems
- Safety Analysis Methods in the Automotive Domain
- Probabilistic FMEA
- Probabilistic Analysis of System Architectures
- Conclusion

ISO 26262: Road Vehicles – Functional Safety



- **Results of HARA:**
  - **Safety Goals (top level safety requirements)**
  - **Automotive Safety Integrity Level (ASIL)**

- **Demonstrate (ASIL-adjusted) compliance with safety requirements by adequate:**
  - procedures
  - architecture
  - design, …

ISO 26262 defines appropriate means for achieving sufficient functional safety

software engineering

ISO 26262: Automotive Safety Integrity Levels (ASIL)

- **Automotive Safety Integrity Levels (ASIL)**
  - ASIL is assigned to each function of the system based on hazard analysis and risk assessment
  - Defines target values for random hardware failures

Level	Random HW Failure
ASIL D	$10^{-8}\ h^{-1}$
ASIL C	$10^{-7}\ h^{-1}$
ASIL B	$10^{-7}\ h^{-1}$
ASIL A	$10^{-6}\ h^{-1}$

Safety Analysis Methods in the Automotive Domain

Qualitative Methods

„identify Failures“

- Qualitative FMEA
- Qualitative Fault Tree Analysis
- Event Tree Analysis

Quantitative Methods

„predict frequency of failures“

- Quantitative FMEA
- Quantitative Fault Tree Analysis
- Event Tree Analysis
- Markov models
- Reliability block diagrams

Safety Analysis Methods in the Automotive Domain

Qualitative Methods

„identify Failures“

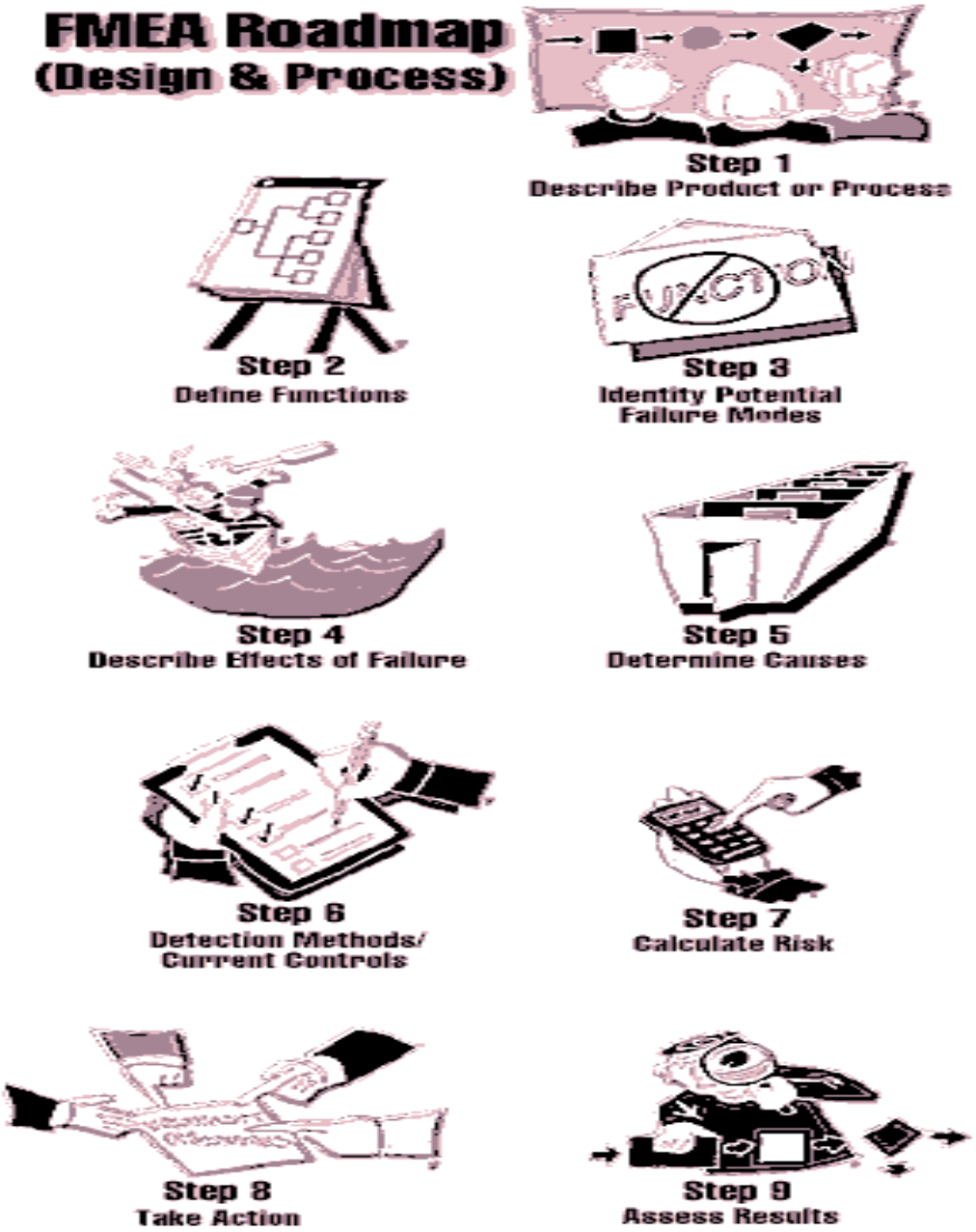
- Qualitative FMEA
- Qualitative Fault Tree Analysis
- Event Tree Analysis

Quantitative Methods

„predict frequency of failures“

- Quantitative FMEA
- Quantitative Fault Tree Analysis
- Event Tree Analysis
- Markov Models
- Reliability Block Diagrams

Failure Modes and Effects Analysis (FMEA)



software engineering

Failure Modes and Effects Analysis (FMEA)

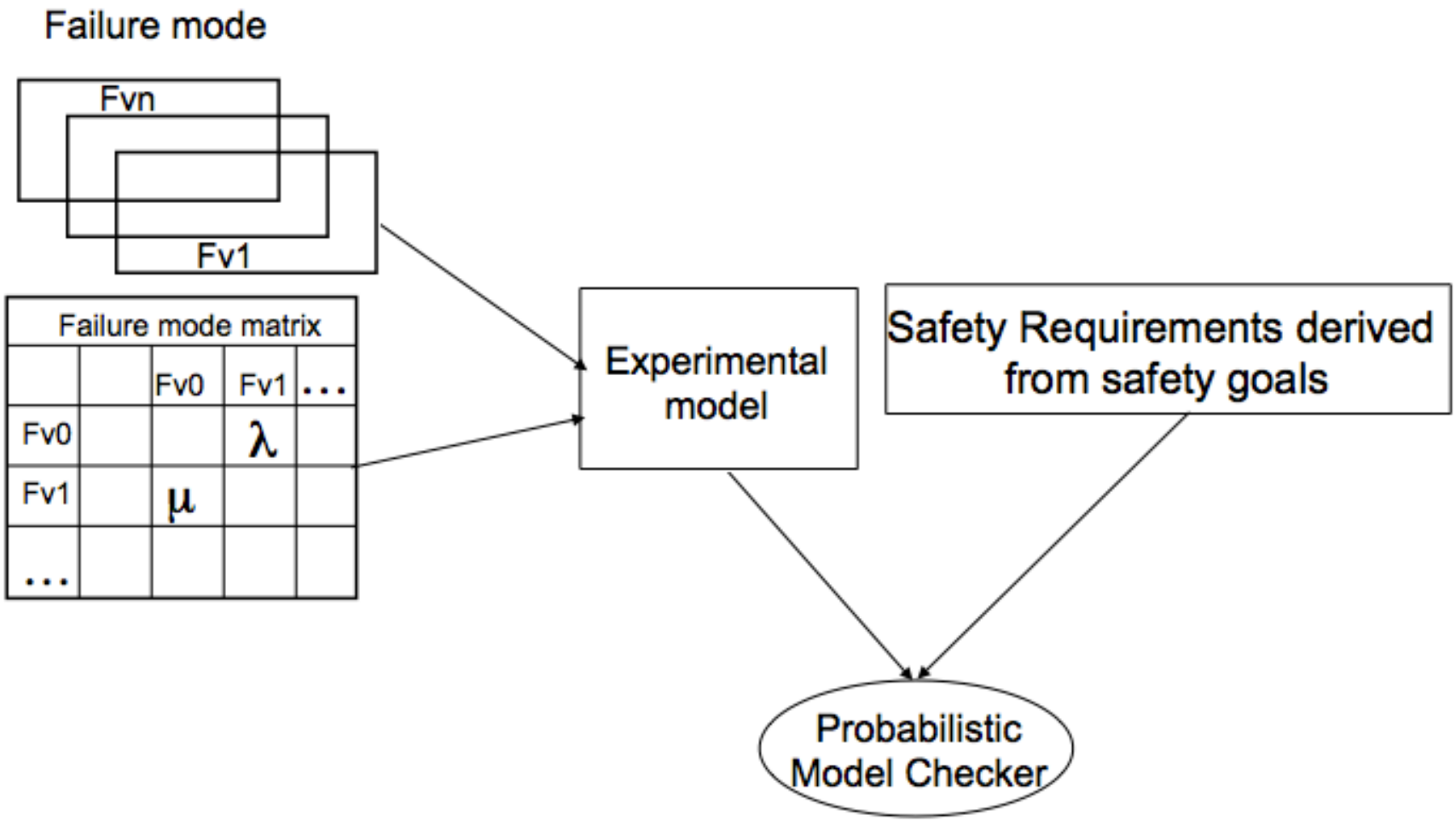
Disadvantage of FMEA

- Manual Process

- Considers only single sources of failures, i.e.,
  - does not take into account that a system can fail only in presence of several (independent) failures

- Does not take advantage of system architecture / behavioural system description:
  - fault injection!

Probabilistic FMEA



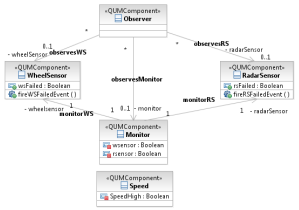
- **QEST 2009:** Husain Aljazzar, Manuel Fischer, Lars Grunske, Matthias Kuntz, Florian Leitner-Fischer, Stefan Leue **"Safety Analysis of an Airbag System using Probabilistic FMEA and Probabilistic Counterexamples"**

Probabilistic FMEA

- **Automated approach**
  - Multiple Faults can be taken into account
  - Check which failure modes can lead to violation of safety goal
  - System architecture in PRISM
  - Counterexamples as sets of paths:
    - graphical representation possible
- **Disadvantages:**
  - PRISM model is input of analysis:
    - SysML / UML are industrial standard, used for real-world application

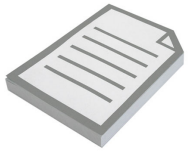
State of the Art: Analysis of SysML / UML Models

SysML / UML Model

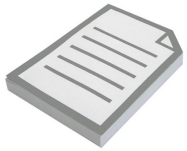


+

Failure Mode Specification



(Safety-) Requirements

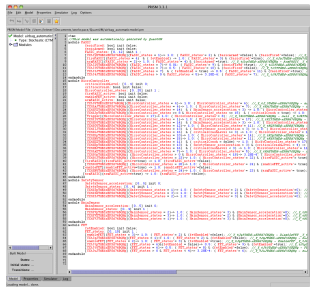


manual

manual

Error prone
+ Time consuming

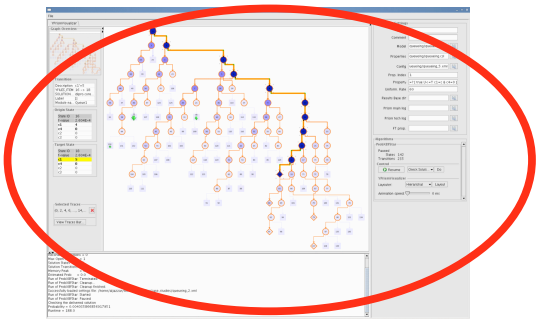
PRISM Model



CSL Property

$$P_{=?}[(\text{true})U^{<=T}(inadvertent\_deployment)]$$

Prob. counterexample

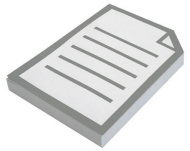


Large number of paths in counterexample,
Interpretation on level of PRISM model

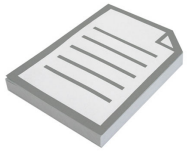
software engineering

The QuantUM Approach

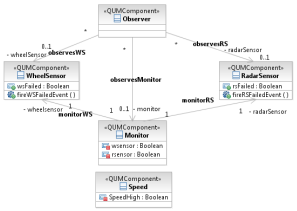
Failure Mode Specification



(Safety-) Requirements

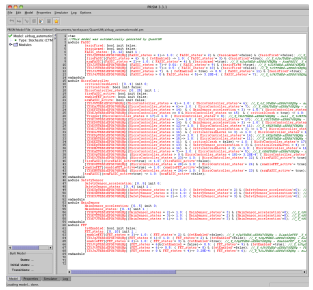


QuantUM UML Model



automatic

PRISM Model

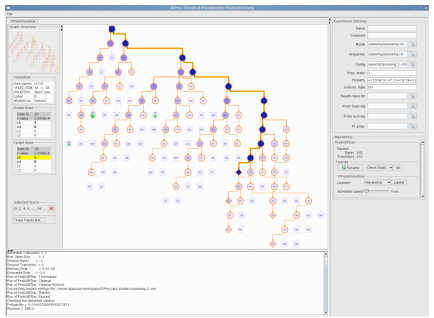


CSL Property

$$P_{=?}[(\mathrm{true})U^{<=T}(inadvertent\_deployment)]$$

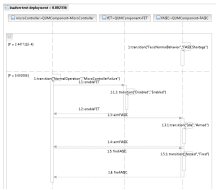
automatic

Prob. Counterexample

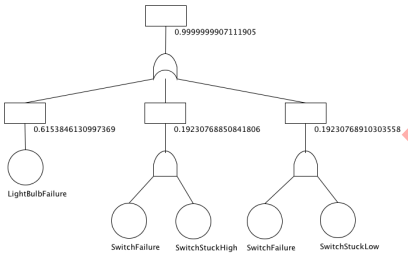


automatic

UML Sequence Diagrams



Fault Trees

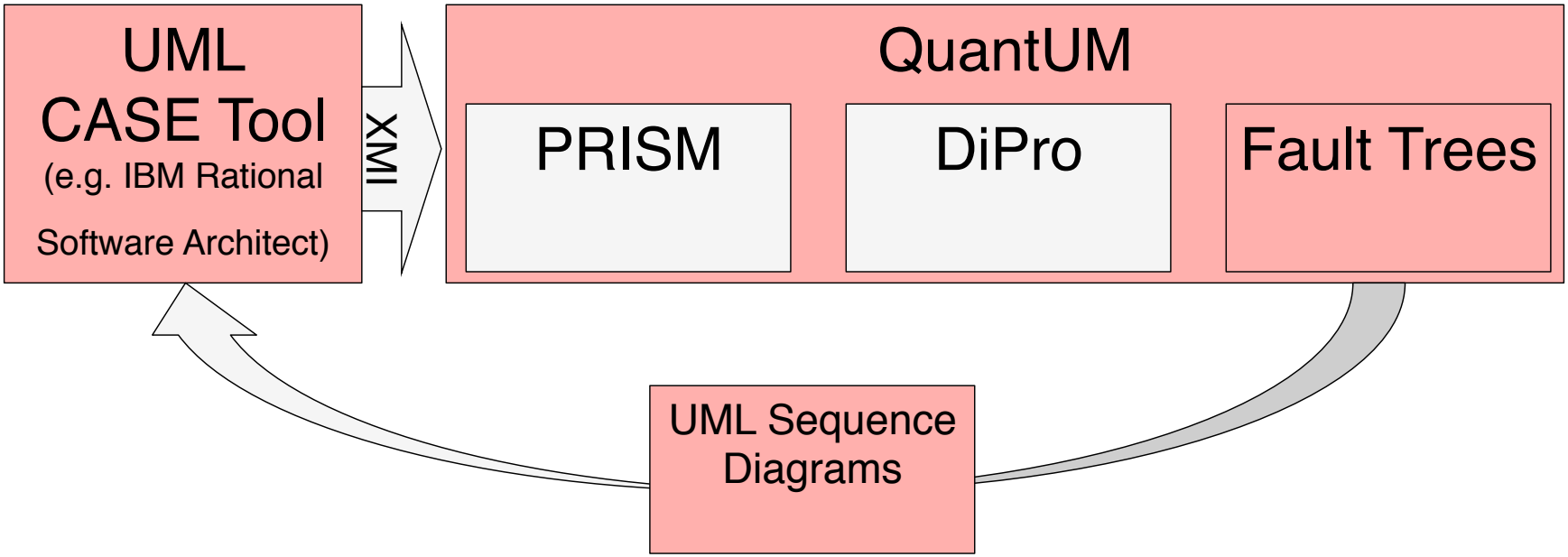


automatic

QAPL 2011:
QuantUM: Quantitative Safety Analysis of UML Models

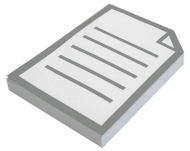
software engineering

The QuantUM Tool

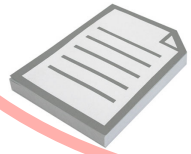


The QuantUM Approach

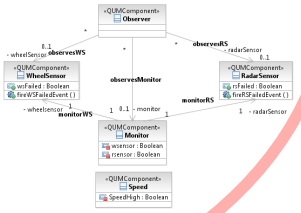
Failure Mode Specification



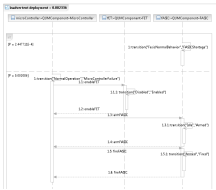
(Safety-) Requirements



QuantUM UML Model

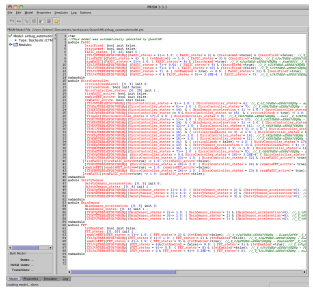


UML Sequence Diagrams



automatic

PRISM Model

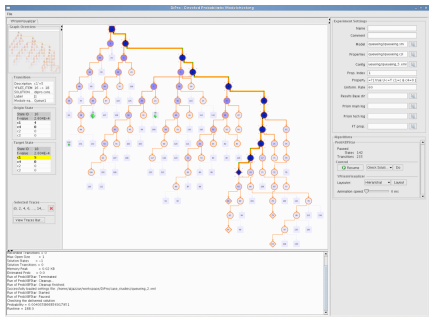


CSL Property

$$P_{=?}[(\mathrm{true})U^{<=T}(inadvertent\_deployment)]$$

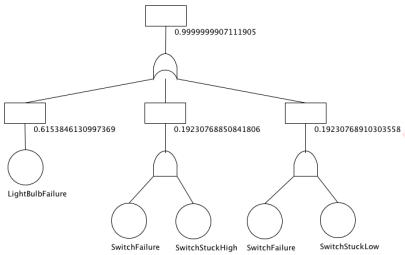
automatic

Prob. Counterexample



automatic

Fault Trees



automatic

software engineering

The QuantUM Model

- **Requirements for the QuantUM Extensions:**
  - Based on
    - „Basic concepts and taxonomy of dependable and secure computing“ Avizzienis et al. (2004)
    - „Reliability Engineering: Theory and Practice“ Birolini (2007)
  - Derived requirements
    - Applicable to system and software architectures
    - Specification of
      - Safety requirements
      - dependability characteristics (failure modes, …)
      - failure propagation and component dependencies
      - safety mechanisms (repair management, redundancy structures)
    - Experienced user shall be able to use QuantUM with minimal training
    - Additional modeling cost shall be kept minimal

The QuantUM Model

- **QuantUM Profile**
  - Extension of the UML with Stereotypes
  - Allows for the specification of
    - Safety Requirements
    - Dependability characteristics (failure modes, …)
    - Failure propagation and component dependencies
    - Safety mechanisms (repair management, redundancy structures)
  - Used within the UML CASE Tool
    - hence low training costs
  - Annotation directly in the architecture model
    - hence low additional modeling costs

The QuantUM Model

- **QuantUM Profile: QUMComponent**
  - Stereotype that can be used to tag
    - Classes
    - Components
  - Associates UML elements with
    - A state machine representing normal behavior
    - (Multiple) state machines representing failure modes
    - A list of (failure) rates that can be used in the state machines

The QuantUM Model

- **QuantUM Profile: Normal Behavior and Failure Modes**
  - State machines are used to describe the
    - Normal behavior of QUMComponents
    - Failure Modes of QUMComponents
  - Each QUMComponent is associated with
    - 0..1 normal behavior state machines
    - 0..n failure pattern state machines describing failure modes
  - Transitions are extended by stereotypes to specify
    - Exponential distributed rates

The QuantUM Model

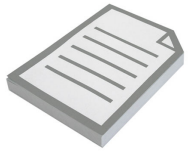
QuantUM Profile: QUMStateConfiguration

- Goal
  - What is the probability of reaching certain states of the UML model?
  - States need to be somehow identified

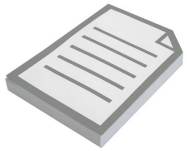
- Solution: QUMStateConfiguration
  - States of the UML state machines can be associated with a name and the “and” or “or” operator
  - All states tagged with the same name are forming a state formula connected by the define operator
    - E.g. state1 and state2, state2 or state3
  - The state formula can then be used in the CSL formula

Translation to PRISM

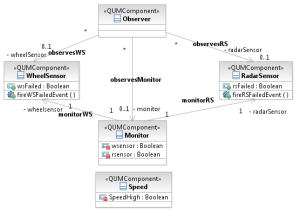
Failure Mode Specification



(Safety-) Requirements



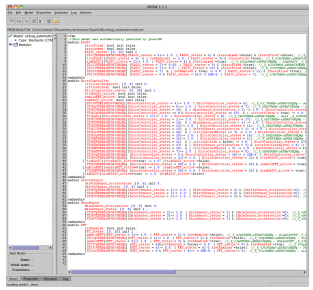
QuantUM UML Model



UML Sequence Diagrams

automatic

PRISM Model

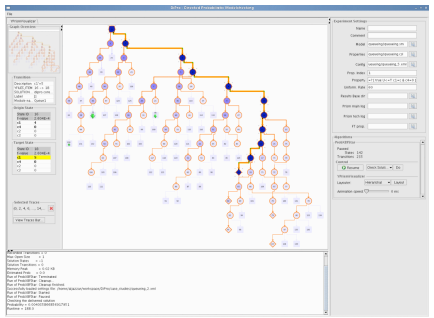


CSL Property

$$P_{=?}[(\mathrm{true})U^{<=T}(inadvertent\_deployment)]$$

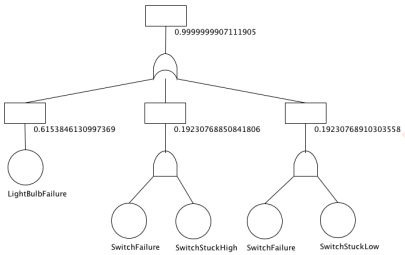
automatic

Prob. Counterexample



automatic

Fault Trees

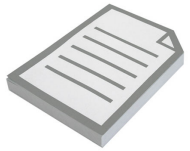


automatic

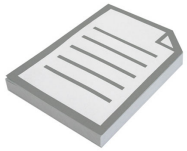
software engineering

Fault Tree Generation

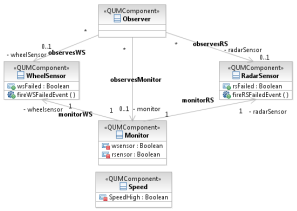
Failure Mode Specification



(Safety-) Requirements

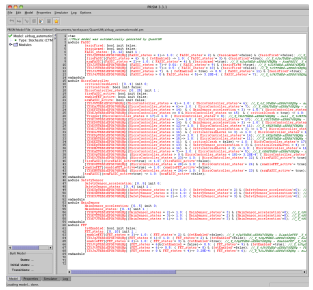


QuantUM UML Model



automatic

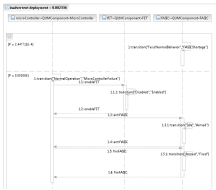
PRISM Model



CSL Property

$$P_{=?}[(\mathrm{true})U^{<=T}(inadvertent\_deployment)]$$

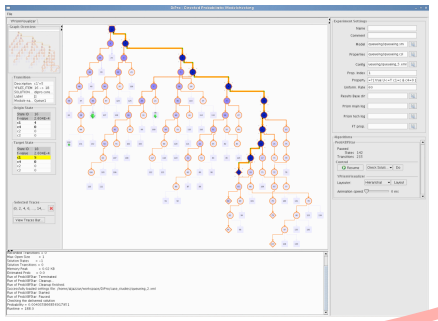
UML Sequence Diagrams



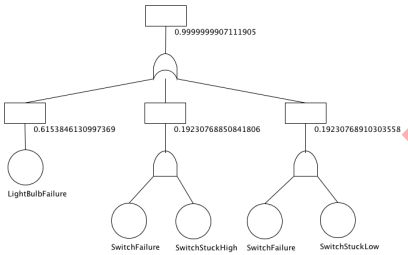
automatic

automatic

Prob. Counterexample



Fault Trees



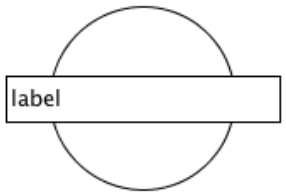
automatic

software engineering

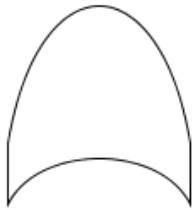
Fault Trees

Fault Trees

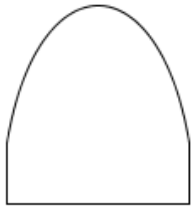
- Used to illustrate error causality
- Engineering practice method (IEC61508, ISO26262, DO-178B)
- Fault Tree Elements



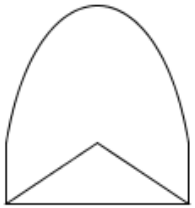
Basic Event



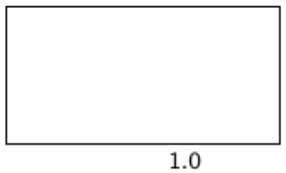
OR-Gate
(Disjunction)



AND-Gate
(Conjunction)



Priority AND-Gate
(Conjunction with Order)



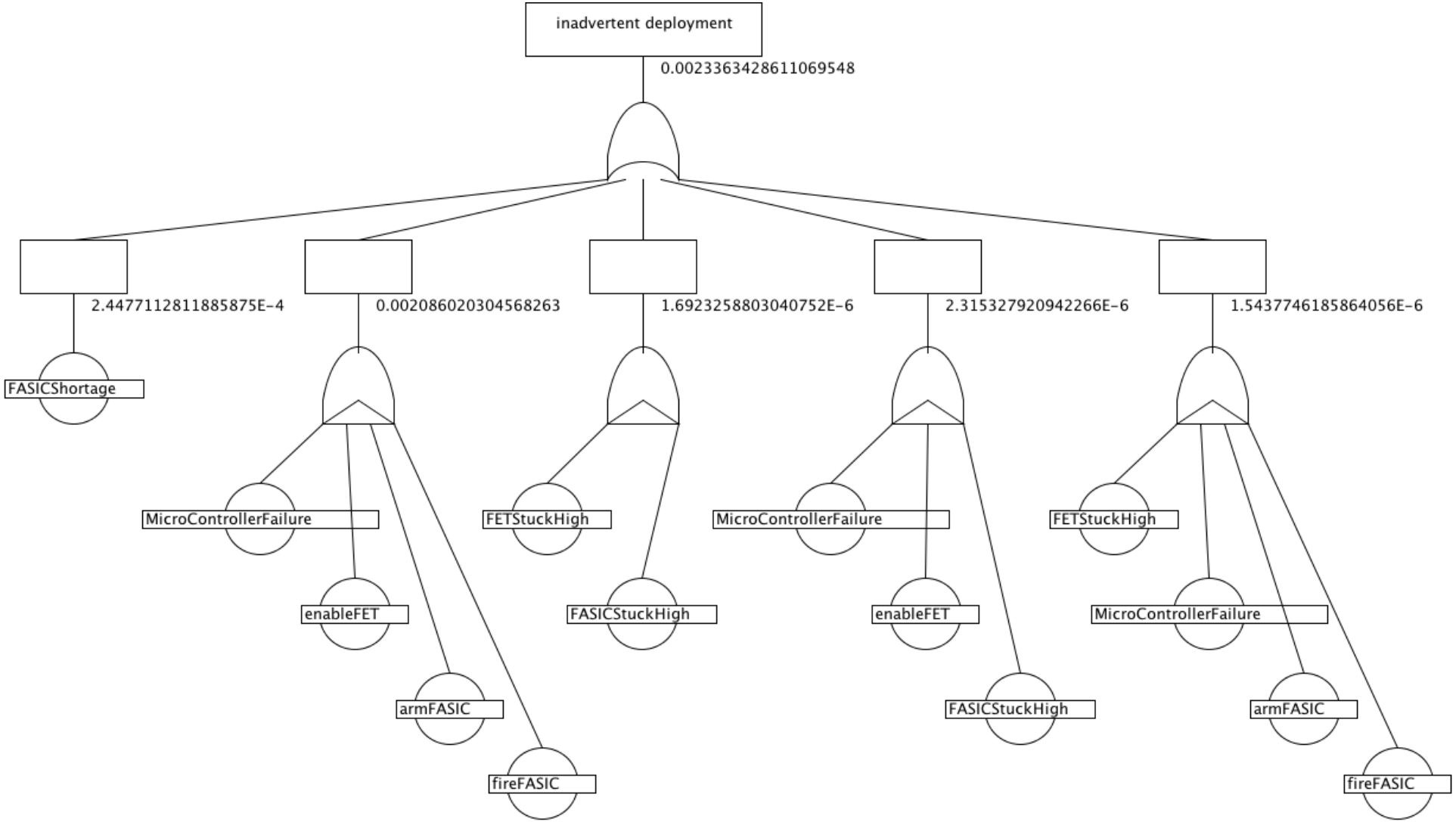
Intermediate Event

Fault Tree Generation

- **Mapping of Counterexamples to Fault Trees**
  - Computation of the fault tree
    - Computation of combinations of basic events causing the hazard
    - Represents events that are causal for the hazard
    - Computational complexity
      - NP-complete
      - Therefore: over approximation preserving all causality
  - Result: compact representation preserving causal information
    - Hundreds of paths in the counterexample
    - Few gates in the fault tree
  - More details: technical report “soft-11-02”
    www.se.inf.uni-konstanz.de > Publications > „From Probabilistic Counterexamples via Causality to Fault Trees“

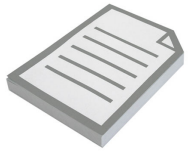
Fault Tree Generation

- **Example: Airbag System - „Inadvertent Deployment“**

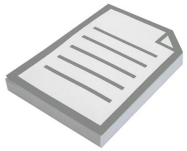


Mapping Counterexamples to UML

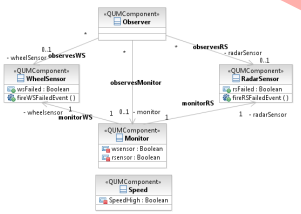
Failure Mode Specification



(Safety-) Requirements

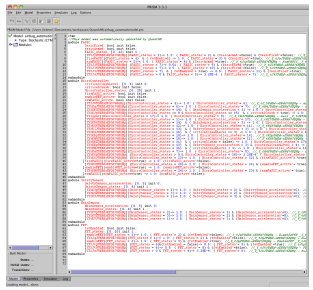


QuantUM UML Model



automatic

PRISM Model

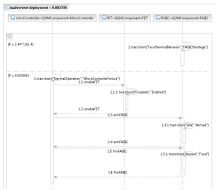


CSL Property

$$P_{=?}[(\mathrm{true})U^{<=T}(inadvertent\_deployment)]$$

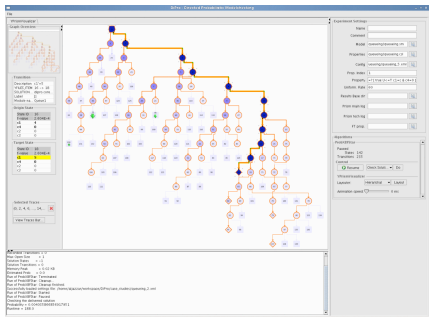
automatic

UML Sequence Diagrams

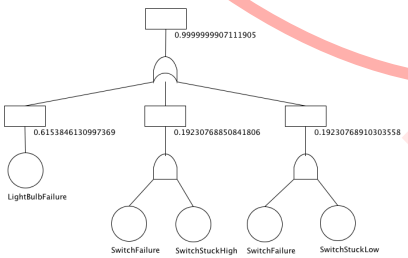


automatic

Prob. Counterexamples



Fault Trees

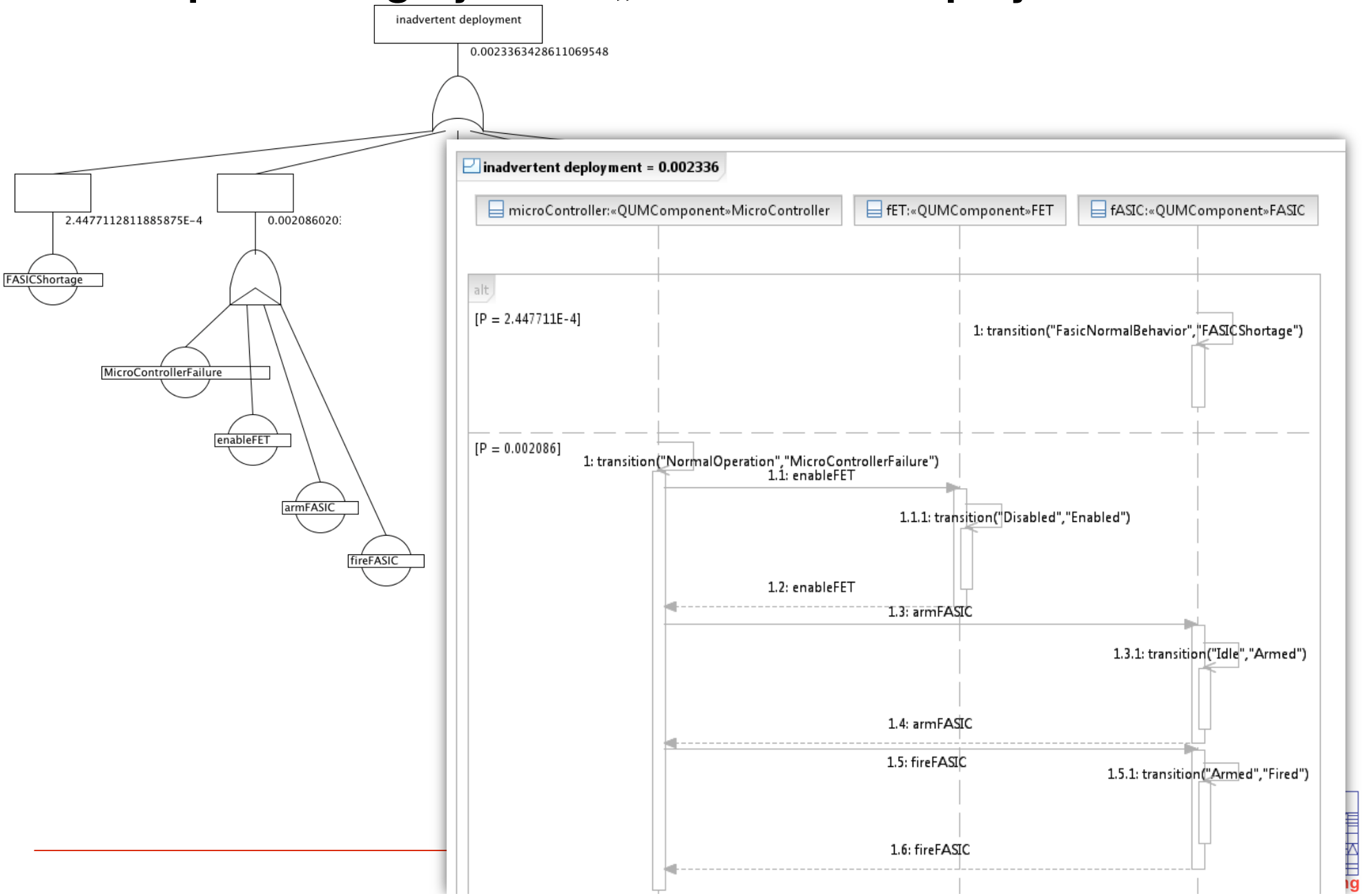


automatic

software engineering

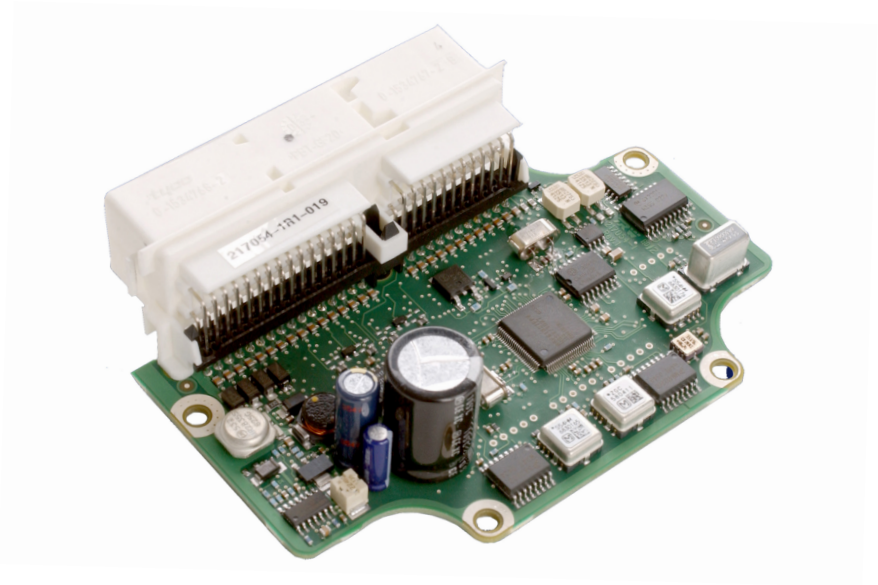
Mapping Counterexamples to UML

- **Example: Airbag System - „Inadvertend Deployment“**



Evaluation

- **Airbag Control System**



TRW Automotive

- **and other case studies outside of the automotive domain.**

software engineering

Evaluation of the QuantUM Approach

- **Results:**
  - Model translation from UML to PRISM
    - Before: several man-days of highly trained engineers
    - Now: automatic translation in a few seconds
  - Mapping the Counterexample to fault trees and UML sequence diagrams aids interpretation
    - Counterexamples with 500-750 paths can be reduced to fault trees with 5-10 gates
    - Patterns can be found immediately
      - Before comparison of several hundred paths was required
    - UML Sequence diagrams
      - Good representation for failure propagation by method calls

Conclusion

- **QuantUM**
  - Automatic safety analysis of UML / SysML models
    - Automatic translation to PRISM
    - Transformation of probabilistic counterexamples to fault trees
    - Representation of probabilistic counterexamples as UML sequence diagrams

- **Future Work**
  - Extension to other architecture description languages

  - Tighter integration in industrial tool chains
    - Requirements Management
    - Configuration and Version Management

Questions ...