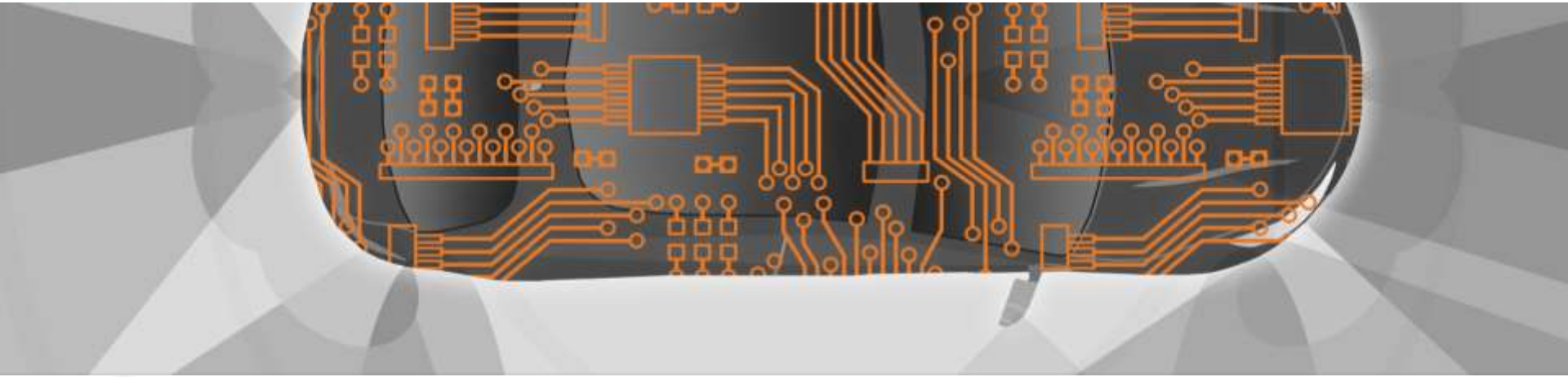
FUNCTIONAL SAFETY & QUALITY EXPERTS GmbH

MCG Management Consult GmbH



Dealing with pseudo-random software faults in complex critical driver assistance systems

Dr. Rüdiger Nortmann (MCG Management Consult GmbH)

Stephen Cobeldick (FSQ Experts GmbH)

Agenda

- Introduction
- Complex systems
- Non-linear and chaotic system behavior
- How to deal with random software faults

Introduction

Introduction

- ISO 26262: software faults are not random, but exclusively systematic

  → This holds only for low complex systems as is required by ISO 26262

- Highly complex software systems:
  e.g. automated driver assistance system according to SAE L3/L4

  →faults can occur that appear randomly

  → faults can only be detected by continuously monitoring safety mechanisms

Types of Randomness

Mild randomness

Subject to physics → restricted by physical constraints

Corresponds (in general) to the physical quantities

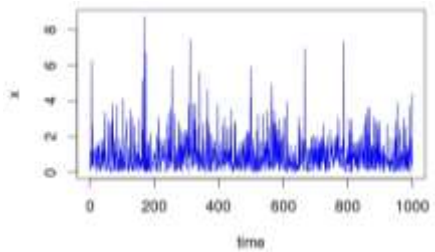
Total is not determined by a single case or observation

When you observe for a while, you can get to know what is going on

Easy to start from what you see and predict what you do not see

Better presented by probabilistic distribution

Robust design can be verified by stress testing or worst-case scenarios



Wild randomness

Dependent on number space → no physical constraints

Corresponds to calculated numerical values

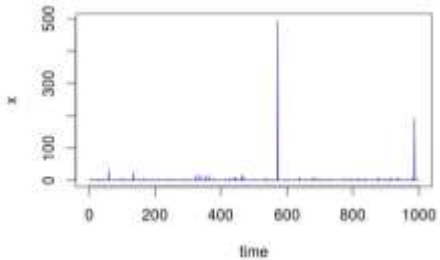
Total will be determined by a small number of extreme events

It takes a long time to know what is going on

Hard to predict from past information

Better presented by fractal distribution

Antifragile design can be verified by simulation



See
https://en.wikipedia.org/wiki/
Seven_states_of_randomness

Reason for Random Software Failures

Possible occurrences:

- Software in a complex system (e.g. chaotic system, stability)
- Software above a certain complexity (e.g. permutations not testable)
- Numeric computations (e.g. binary floating point, algorithm robustness)

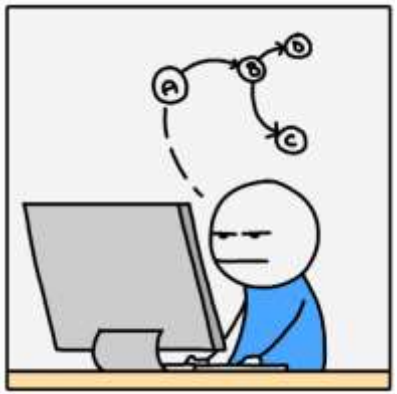
Not in scope:

- Software design, software bugs, cosmic rays, etc.

Complex systems

Complex Systems: Types of Complexity

Accidental



© Monkey User

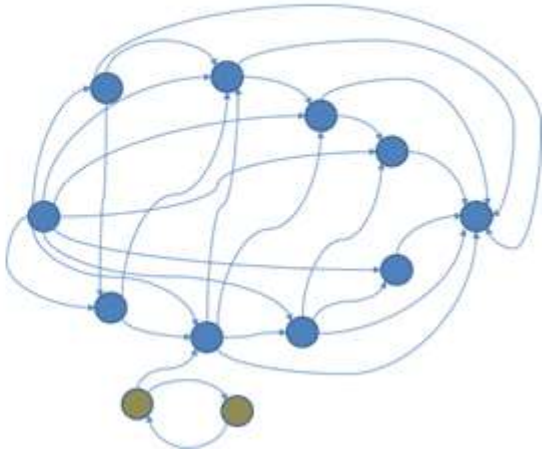
Non-essential complexity introduced into the design by mistake

Incidental

© Monkey User

Non-essential complexity introduced into the design out of ignorance or on purpose

Essential



Unavoidable complexity

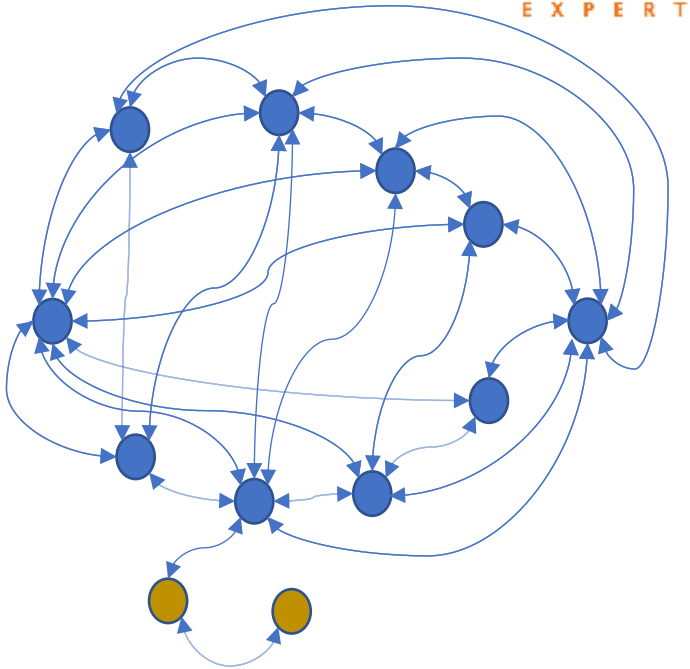
The automated driving system is essential complex

Complex Systems: Metric

Cyclomatic complexity

$$M = E - N + 2P$$

E => The no. of edges of the graph
N => The no. of nodes of the graph
P => The no. of connected components

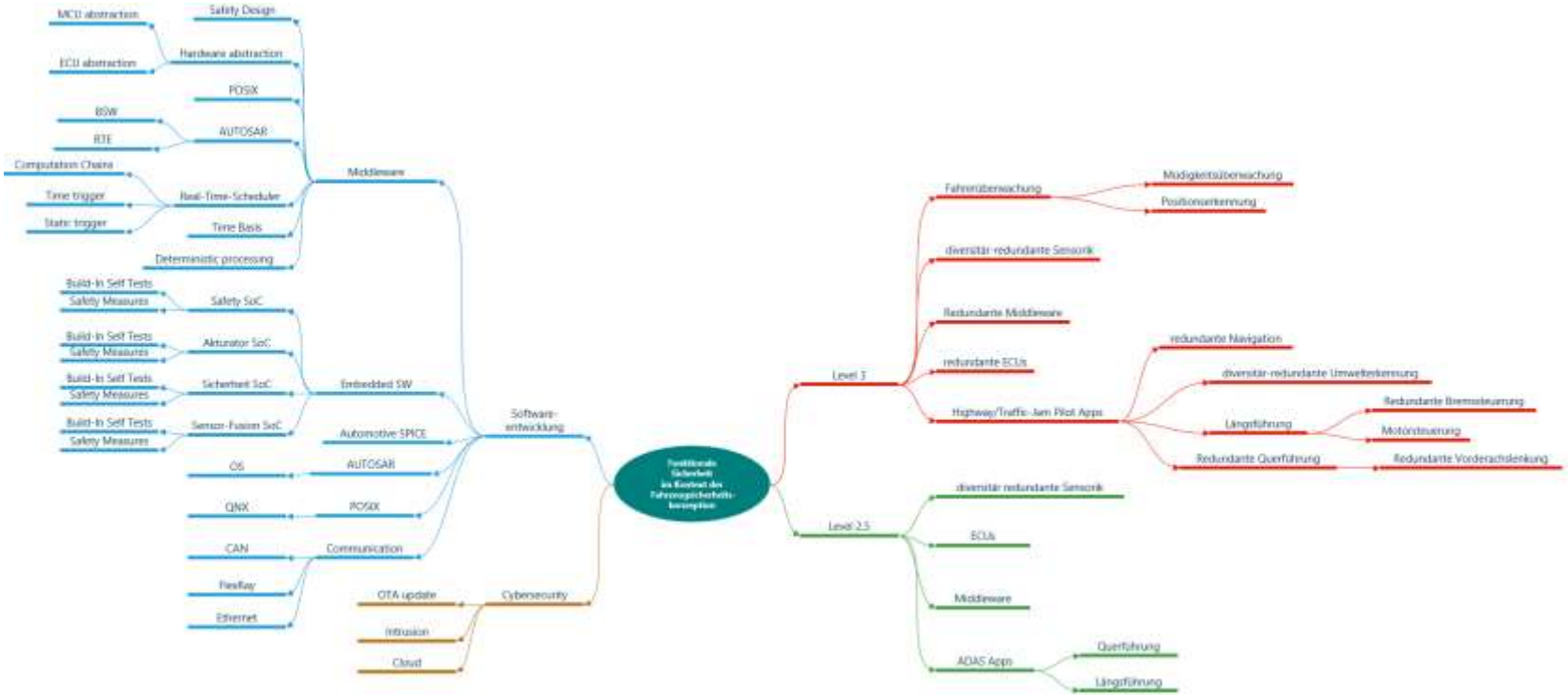


Tom McCabe's categorization to interpret cyclomatic complexity:

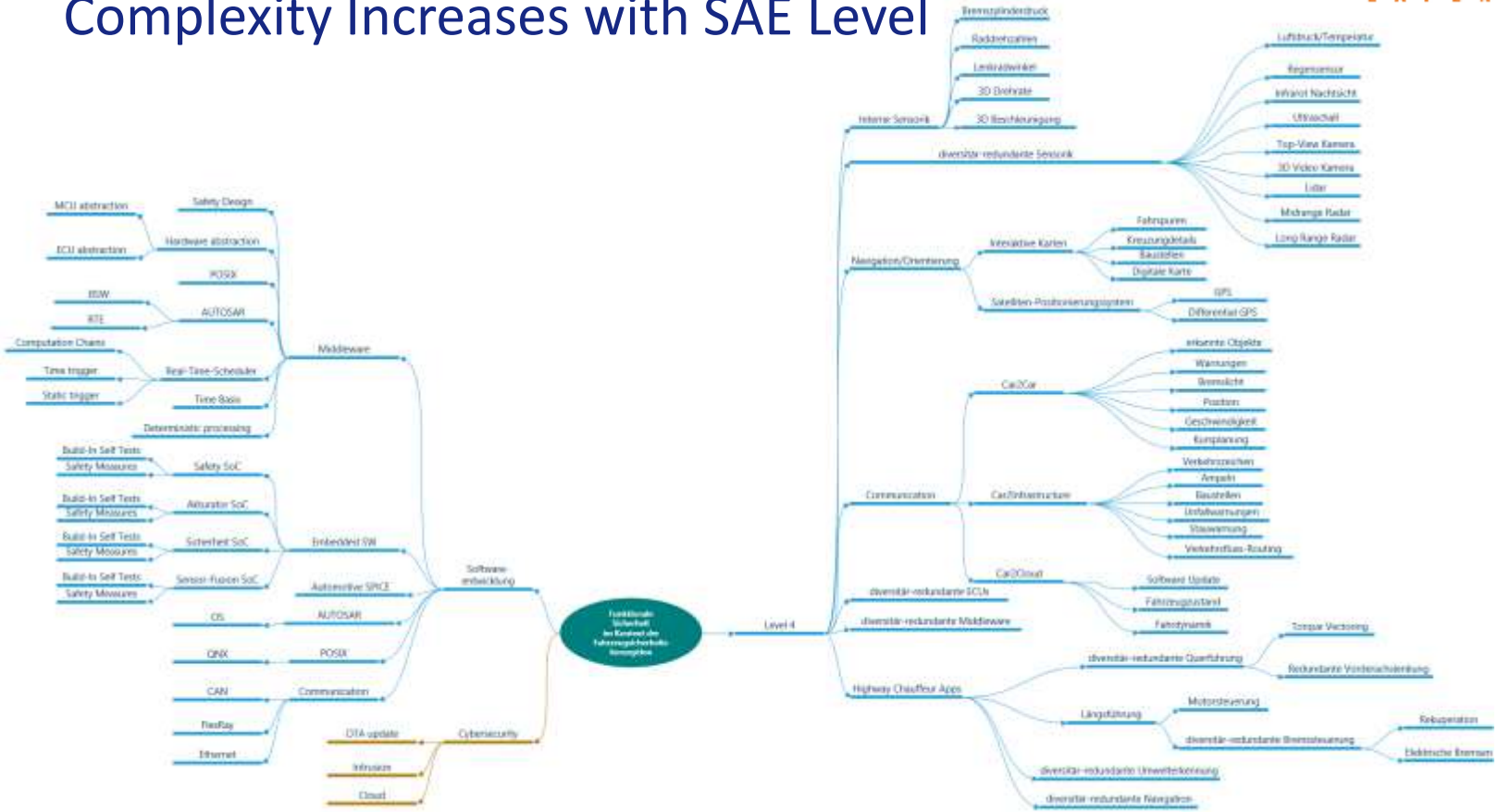
1 – 10 -> Simple procedure, little risk -> low complexity
11 – 20 -> More complex, moderate risk-> not acceptable for ASIL D
21 – 50 -> Complex, high risk -> not acceptable for ASIL
> 50 -> Untestable code, very high risk

M = 56 - 12 + 2x2 = 48
N > 48x12 = 576 System testcases for binary decision

Complexity Increases with SAE Level



Complexity Increases with SAE Level



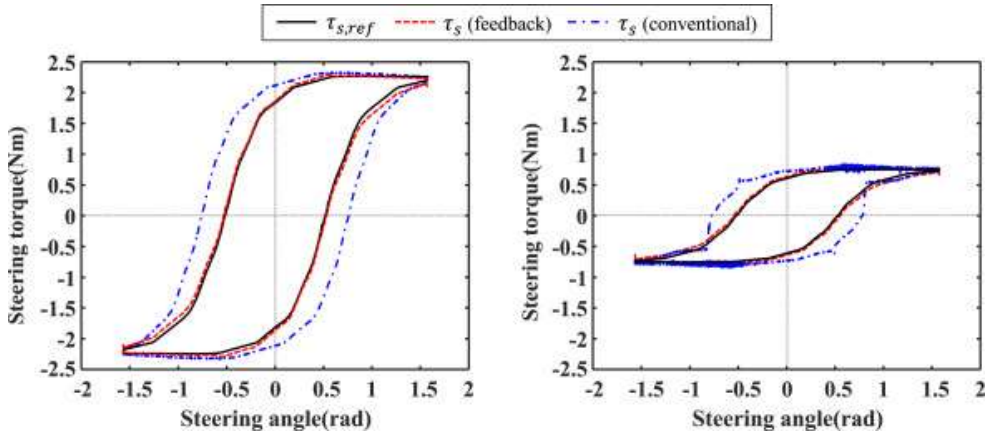
Complex Systems: Test is Not Enough

- Challenges for testing of complex systems
  - Design cannot be fully verified by testing
  - Equivalence classes do not help against black swans
  - Safety can only be proven by simulation:
    -> First simulate a critical situation and then test it
  - Tests are not sufficient to protect against random failures, only against systematic errors (even with endurance tests)
  - Btw: This applies to both hardware and software

Non-linear and chaotic system behavior

How to Deal with Non-Linearities

- If an algorithm is supposed to control non-linear behavior over several calculation steps, a correct result can no longer be expected after just a few steps (approx. 7) and the result is incorrect in an apparently random way (faulty).
- Complex motor control must be protected against accidental errors in the algorithms by dynamic limiters (e.g. dynamic torque limiters).



Deterministic, Yet Chaotic

- Simple systems may be deterministic, and yet demonstrate chaotic behavior.
- Systems which include feedback can be modelled by differential equations.
- Lorenz System was originally derived to model fluid convection system.
  - Ordinary differential equation (ODE)
  - Chaotic behavior for values around:
    - σ = 10
    - β = 8/3
    - ρ = 28

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x),$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y,$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z.$$

Deterministic, Yet Chaotic

$$\frac{dx}{dt} = \sigma(y - x),$$

$$\frac{dy}{dt} = x(\rho - z) - y,$$

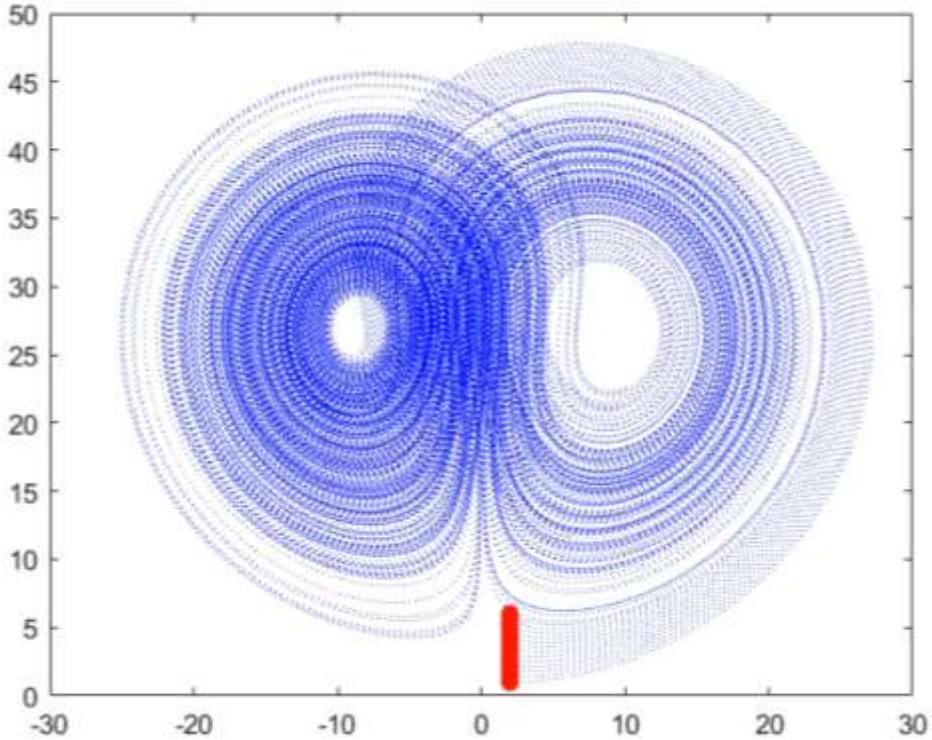
$$\frac{dz}{dt} = xy - \beta z.$$

```
sigma = 10;
 beta = 8/3;
 rho = 28;
 % ODE function:
 fun = @(t,a)[…
      sigma*(a(2)-a(1));
      a(1)*(rho-a(3))-a(2);
      a(1)*a(2)-beta*a(3)];
 % Runge-Kutta 4th/5th order ODE
 solver:
 t = linspace(0,23,2345);
 [~,a] = ode45(fun,t,[1,2,3]);
```

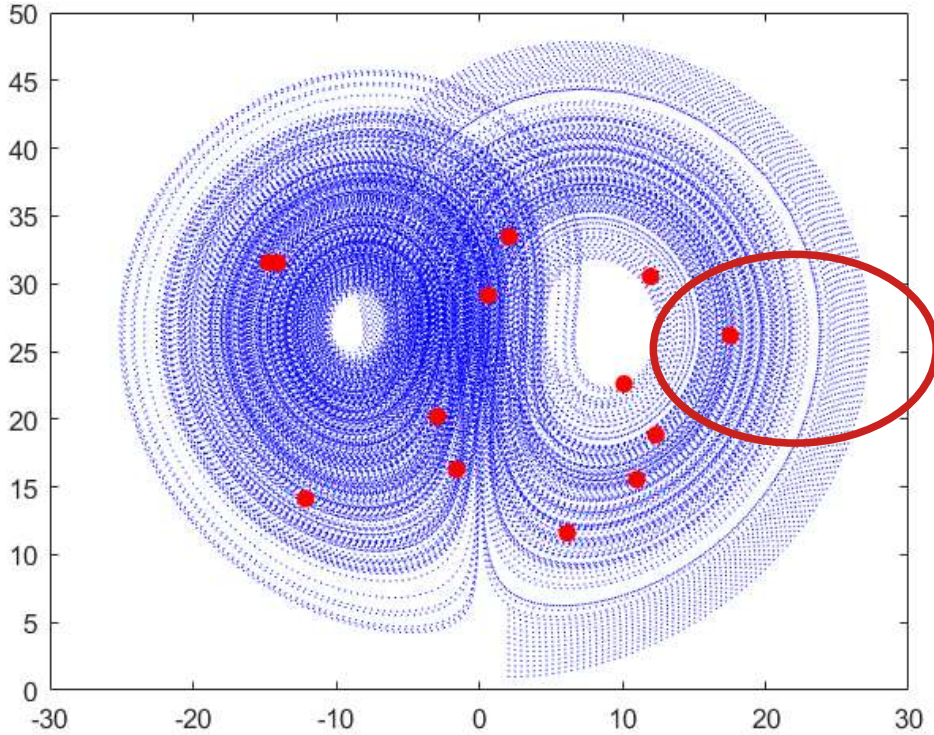
Deterministic, Yet Chaotic

- Differential equations can be used to describe many systems which utilize feedback, for example:
  - Actuators,
  - Cameras, LIDAR, etc which compensate for lighting levels,
  - Decision-making algorithms based on prior states, etc.
- Small perturbations in the initial conditions can lead to very large differences in the later state.
- Impossible to simulate or test all permutations of any non-trivial system:
  - Later states may be sampled by distribution
  - Chaotic behavior may occur

Deterministic, Yet Chaotic



Deterministic, Yet Chaotic



Note the density, i.e. probability distribution

Deterministic, Yet Chaotic

- Chaotic behavior may be sampled over repeated periods of time.
- Sufficient samples can fit a statistical probability distribution.
- Compare ISO 26262-1:2018 3.118:

3.118
random hardware failure
failure (3.50) that can occur unpredictably during the lifetime of a hardware *element* (3.41) and that follows a probability distribution

Note 1 to entry: Random hardware failure rates can be predicted with reasonable accuracy.

Note 2 to entry: Physical hardware *failures* (3.50) as defined by the *PoF* (3.111) methodology (SAE J1211, JEDEC JEP122, or similar) can be considered as random hardware failures for the purpose of this document.

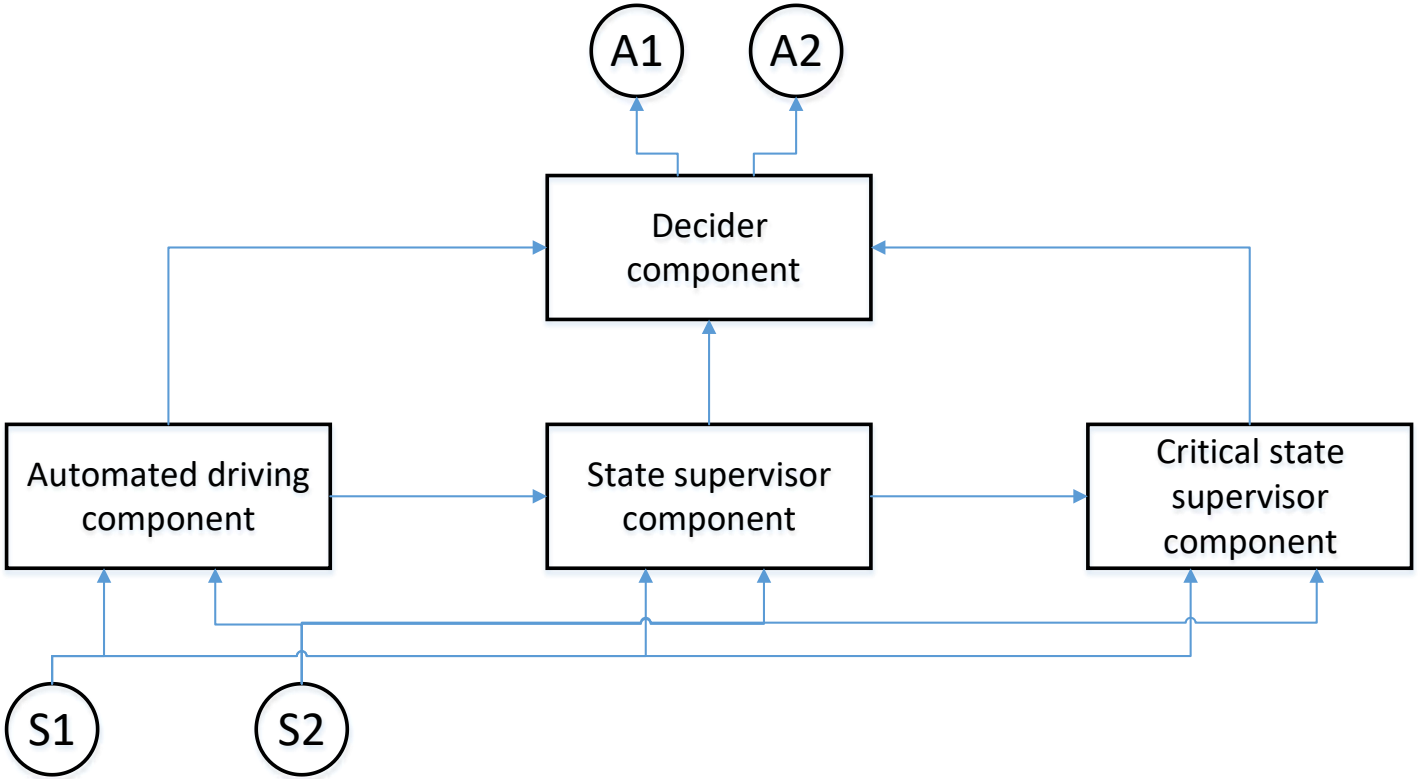
3.119
random hardware fault
hardware *fault* (3.54) with a probabilistic distribution

How to deal with random software faults

Complex Systems: Dealing with Random SW Faults

- Modelling
  - Robust design (control theory)
  - Antifragile design
  - Fault tolerant design
  - STAMP Model
- Analysis
  - State transition model
  - Markov-Chain Analysis
  - Petri Net Analysis
  - STPA Analysis
- Simulation
  - Simulink
  - Rhapsody
  - dSPACE

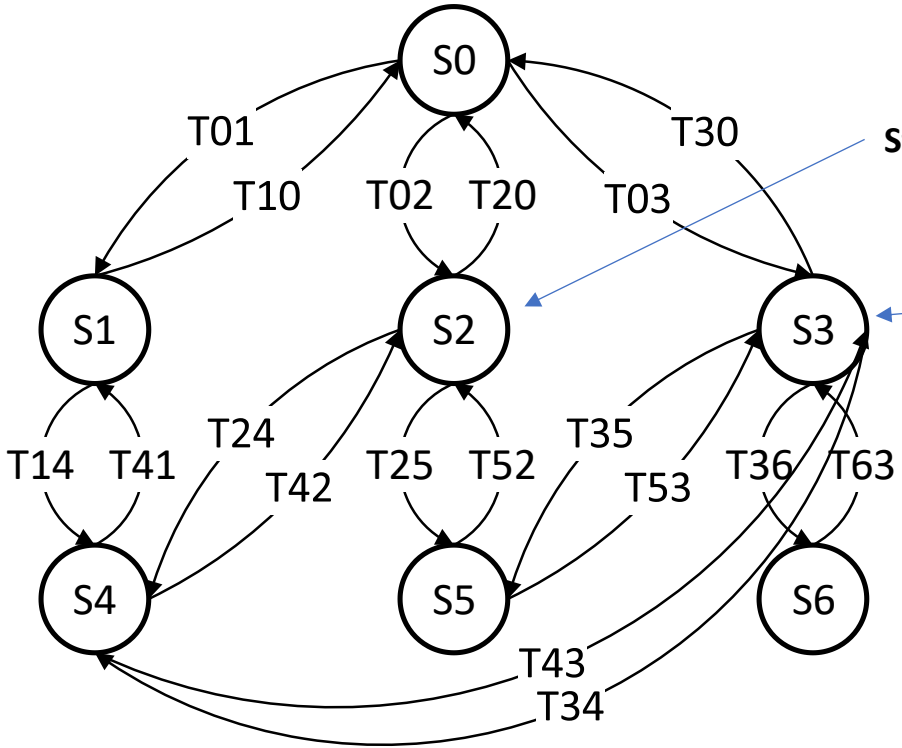
Automated Driving – 3-Components and 2-Channels



Complex Systems: Build Up Safety Mechanism

- Understand state transitions and their safety relevance
- Define signal supervision
- Define **State Machine Supervisor** to check correct state transition at runtime
  → Reduce complexity with respect to safety CC < 20
- Define **Critical State Supervisor** to check correctly taken safe states at runtime
  → Reduce complexity with respect to safety CC < 10
- Analyze the probability of faults and ISO 26262 compliance with **Markov Chain and Petri Net analyses**
- **Simulate** the state machine with simulation tool
- **Test** critical state transitions

State Transition Model for 3-Component System

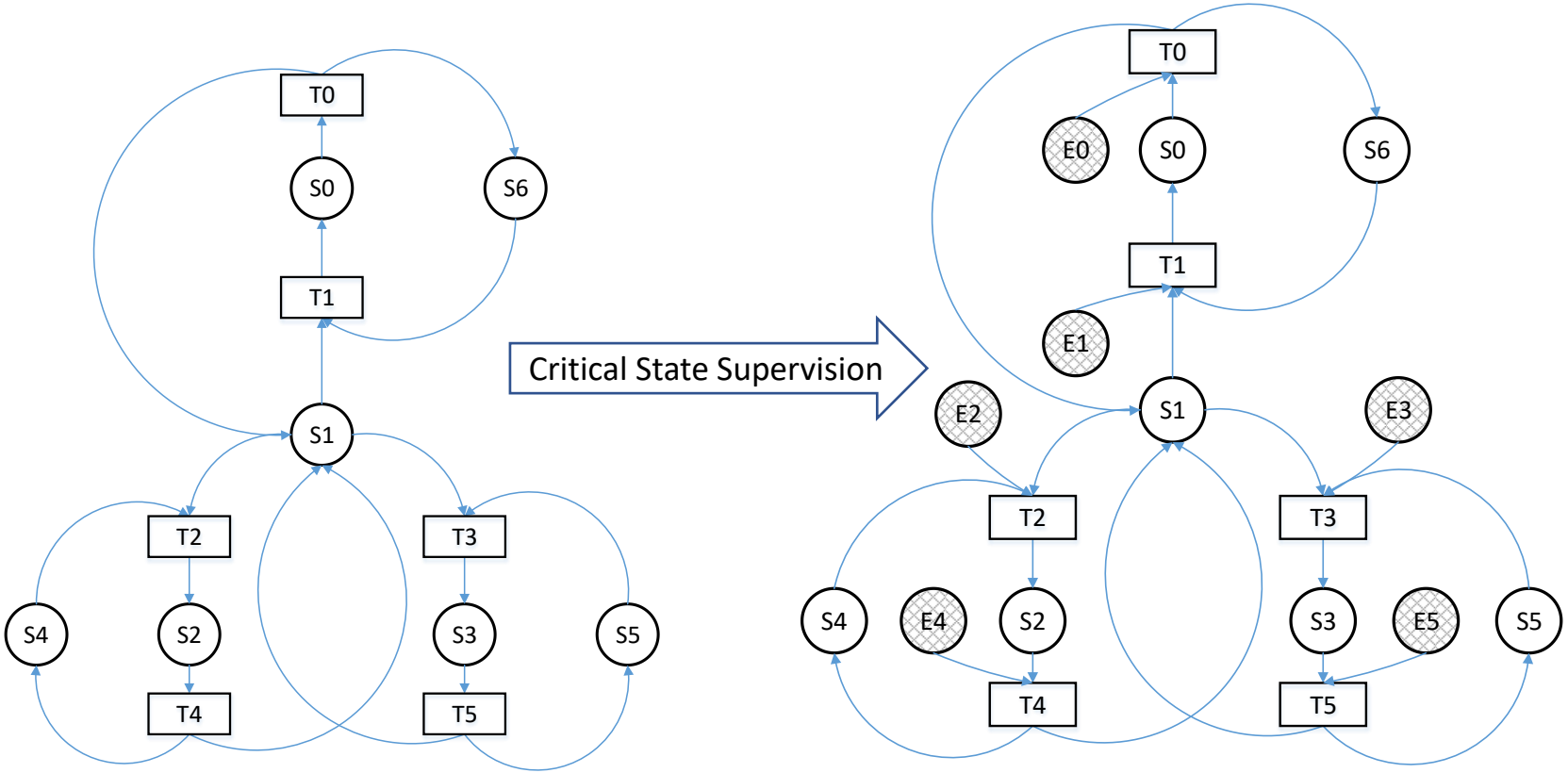


State Machine Supervisor

Critical State Supervisor

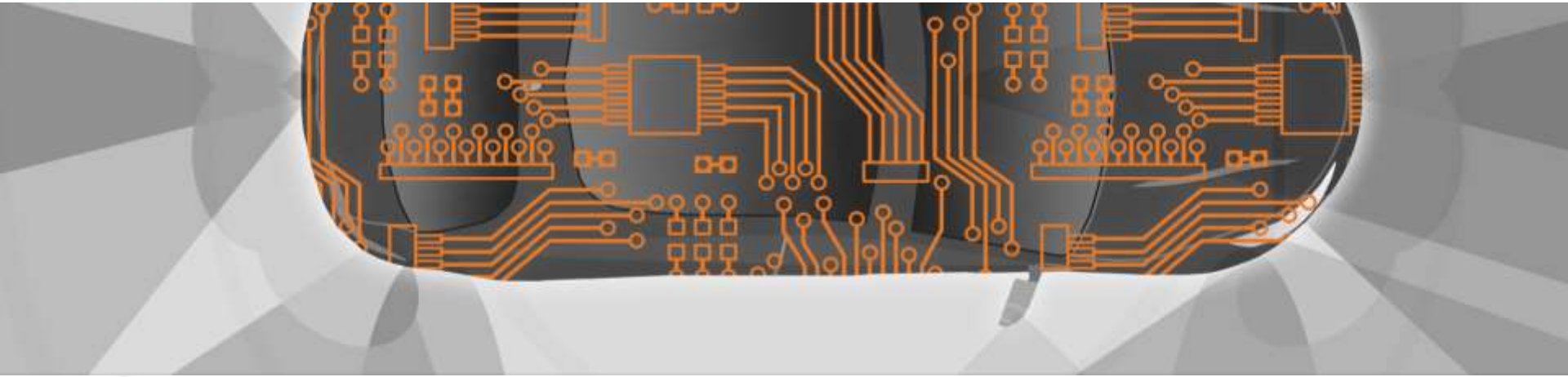
Use state transitions for Markov Chain Analysis

Petri Net for Control of 3-Component System



Conclusion

- Software can behave randomly although we might call this pseudo-random
- The random behavior of software is due to
  - design exceeding the reasonable values of complexity metrics
  - non-deterministic algorithms
  - non-linearity of the control problem
- Compliance of random fault-prone software with safety standards is rarely ensured by testing
- The compliance can be ensured by
  - Robust, antifragile and fault tolerant design, STAMP
  - Appropriate safety analysis methods like State Transition, Markov Chain, Petri Net, STPA
  - Simulation in Simulink, Rhapsody, dSPACE
  - Extensive test of critical situations identified in simulation



Thank you for your attention

FSQ Experts GmbH
Balanstrasse 14
81669 München
+49 89 588087571
info@fsq-experts.com
www.fsq-experts.com

MCG Management Consult GmbH
Waldstrasse 25
30890 Barsinghausen
+49 5105 585919
mcg@mcgnet.de
www.mcgnet.de