technische universität
dortmund

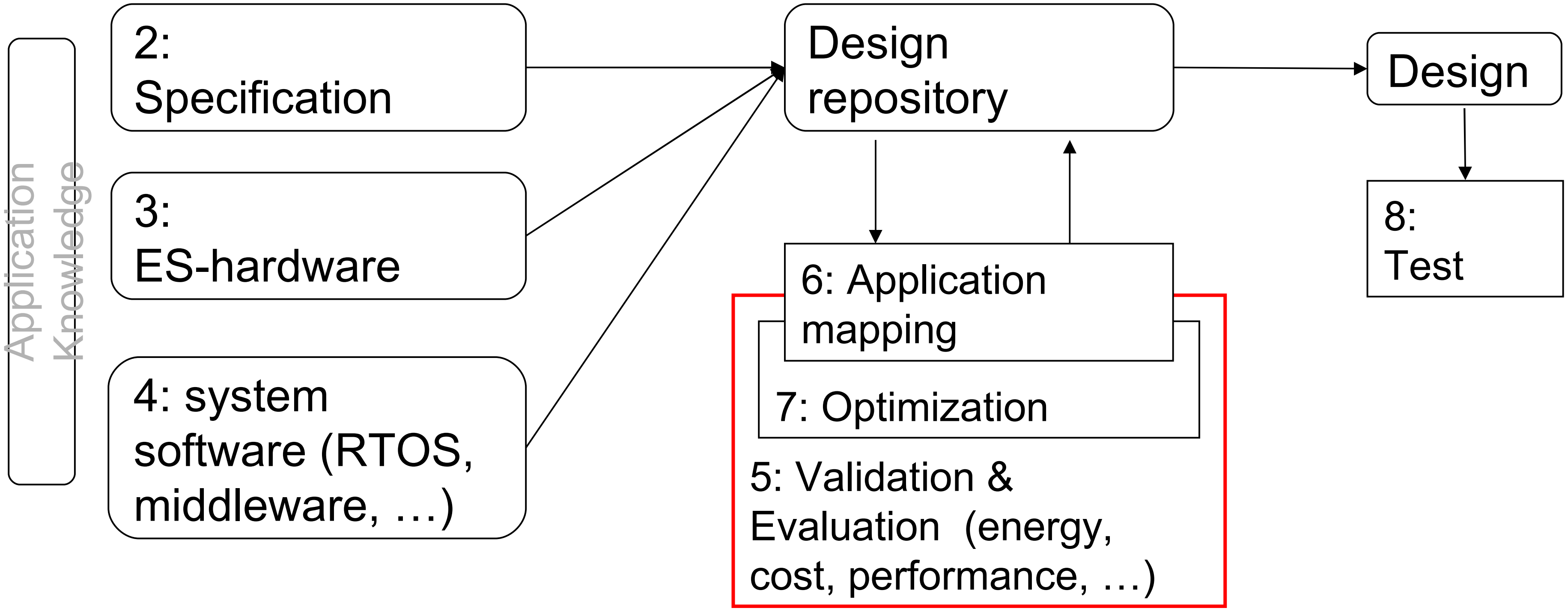
fakultät für informatik
informatik 12

Evaluation and Validation

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2009/12/01

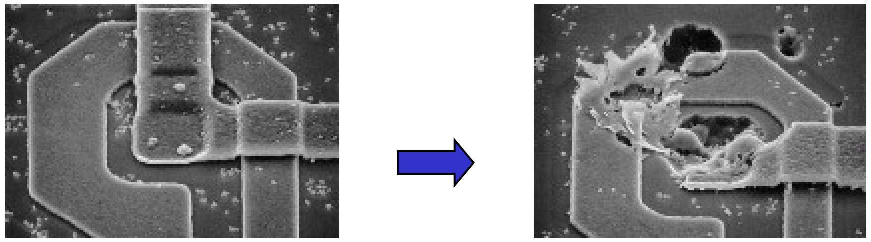
Structure of this course



Numbers denote sequence of chapters

Risk- and dependability analysis

Example : metal migration @ Pentium 4



www.jrwhipple.com/computer_hangs.html

“10^{-9}“: For many systems, probability of a catastrophe has to be less than 10^{-9} per hour $\equiv$ one case per 100,000 systems for 10,000 hours.

FIT: failure-in-time unit for failure rate (=1/MTTF$\approx$1/MTBF);

1 FIT: rate of 10^{-9} failures per hour

Damages are resulting from hazards.

For every damage there is a severity and a probability.

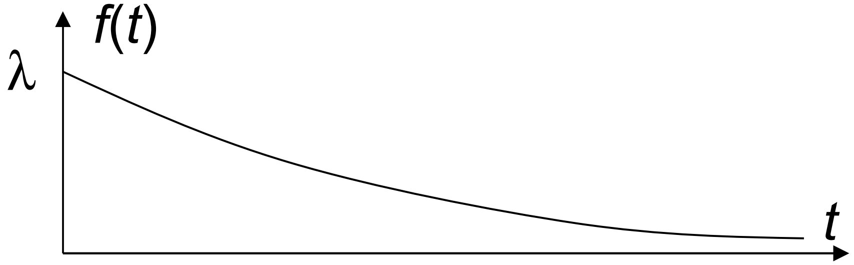
Several techniques for analyzing risks.

Reliability: f(t), F(t)

- Let *T*: time until first failure, *T* is a random variable
- Let *f*(*t*) be the density function of *T*

<u>Example</u>: Exponential distribution

$$f(t)=\lambda e^{-\lambda t}$$

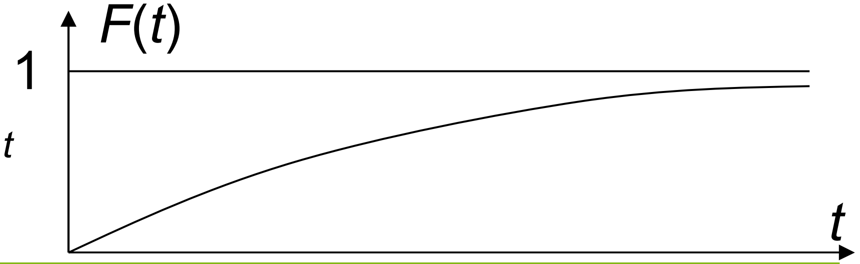


- *F*(*t*) = probability of the system being faulty at time *t*:

$$F(t) = \Pr(T \le t) \qquad F(t) = \int_0^t f(x)\,dx$$

<u>Example</u>: Exponential distribution

$$F(t) = \int_0^t \lambda e^{-\lambda x} dx = -[e^{-\lambda x}]_0^t = 1 - e^{-\lambda t}$$



Reliability: *R*(*t*)

- **Reliability** $R(t)$ = probability that the time until the first failure is larger than some time t:

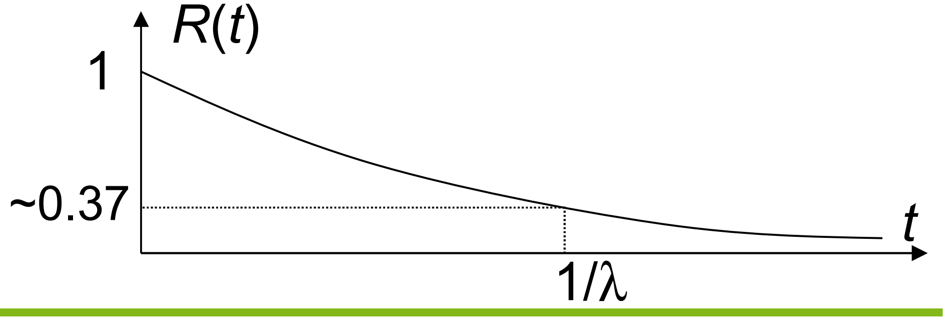
$R(t)=\Pr(T>t),\ t\geq 0$ $\qquad$ $R(t)=\int_t^{\infty} f(x)dx$

$$F(t)+R(t)=\int_0^t f(x)dx+\int_t^{\infty} f(x)dx=1$$

$R(t)=1-F(t)$ $\qquad$ $f(t)=-\frac{dR(t)}{dt}$

<u>Example</u>: Exponential distribution

$R(t)=e^{-\lambda t}$;



Failure rate

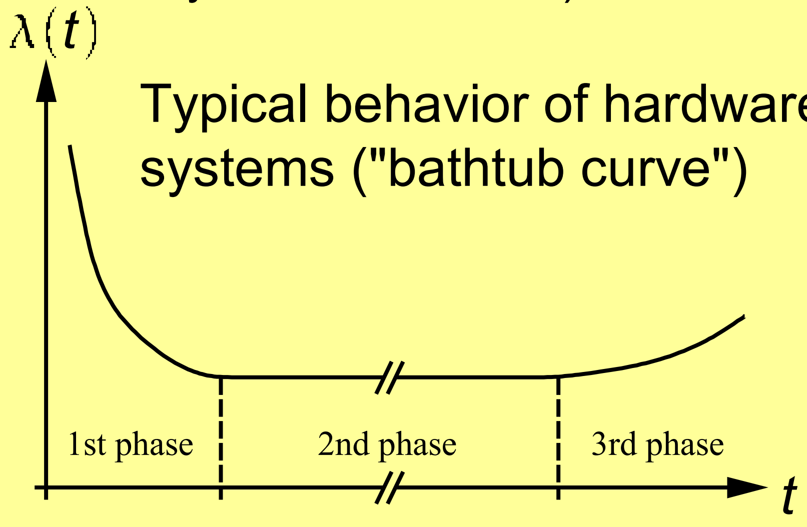
The failure rate at time *t* is the probability of the system failing between time *t* and time *t+Δt:*

$$\lambda(t) = \lim_{\Delta t \to 0} \frac{\Pr(t < T \le t + \Delta t \mid T > t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)}$$

Conditional probability ("provided that the system works at *t* ");

P(A|B)=P(AB)/P(B)

Typical behavior of hardware systems ("bathtub curve")



For exponential distribution:

$$\frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda$$

FIT = expected number of failures in 10^9 hrs.

MTTF = *E*{*T* }, the *statistical mean* value of *T*

$$\text{MTTF} = E\{T\} = \int_0^\infty t \cdot f(t)\, dt$$

According to the definition of the statistical mean value

Example: Exponential distribution

$$\text{MTTF}_{\text{exp}} = \int_0^\infty t \cdot \lambda e^{-\lambda t} dt = -\cancel{\left[t \cdot e^{-\lambda t}\right]_0^\infty} + \int_0^\infty e^{-\lambda t} dt$$

$$\int u \cdot v' = u \cdot v - \int u' \cdot v$$

$$\text{MTTF}_{\text{exp}} = -\frac{1}{\lambda}\left[e^{-\lambda t}\right]_0^\infty = -\frac{1}{\lambda}[0-1] = \frac{1}{\lambda}$$

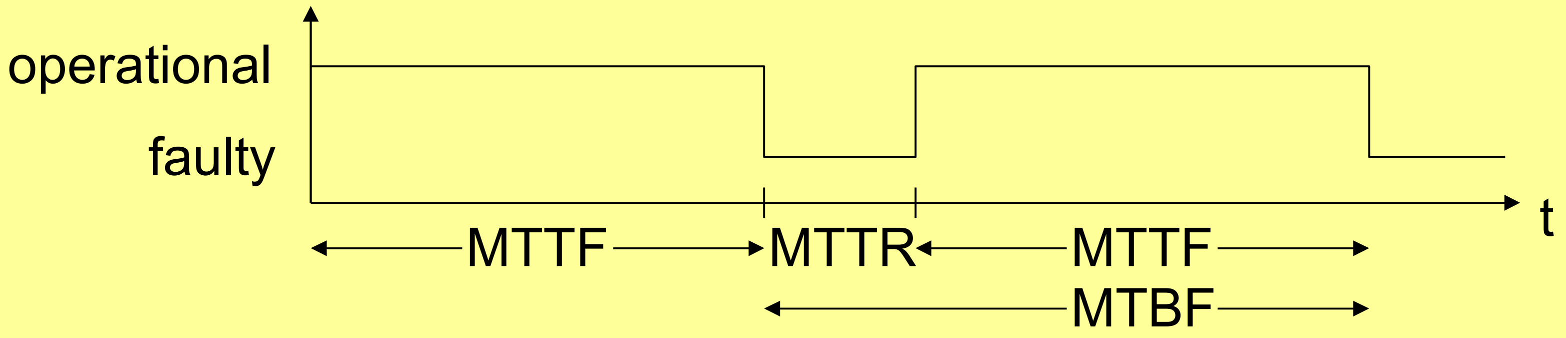
MTTF is the reciprocal value of failure rate.

MTTF, MTTR and MTBF

MTTR = mean time to repair
(average over repair times using distribution *M*(*d*))
MTBF* = mean time between failures = MTTF + MTTR

$$\text{Availability } A = \lim_{t \to \infty} A(t) = \frac{\text{MTTF}}{\text{MTBF}}$$

Ignoring the statistical nature of faults …

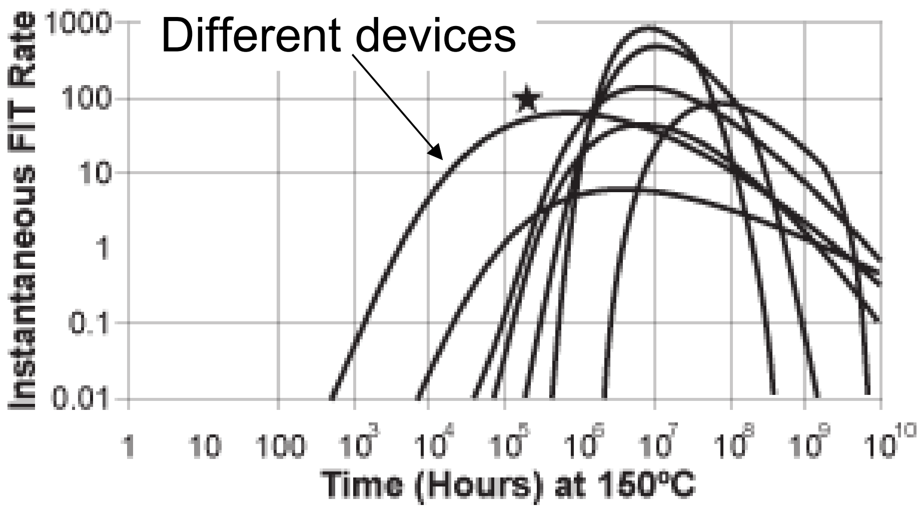


* Mixed up with MTTF, if starting in operational state is implicitly assumed

Actual failure rates

Example: failure rates less than 100 FIT for the first 20 years (175,300 hrs) of life at 150°C @ TriQuint (GaAs)

[www.triquint.com/company/quality/faqs/faq_11.cfm]



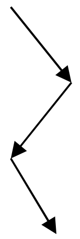
Target: Failures rates of systems ≤ 1FIT

Reality: Failures rates of circuits ≤ 100 FIT

☞ redundancy is required to make a system more reliable than its components

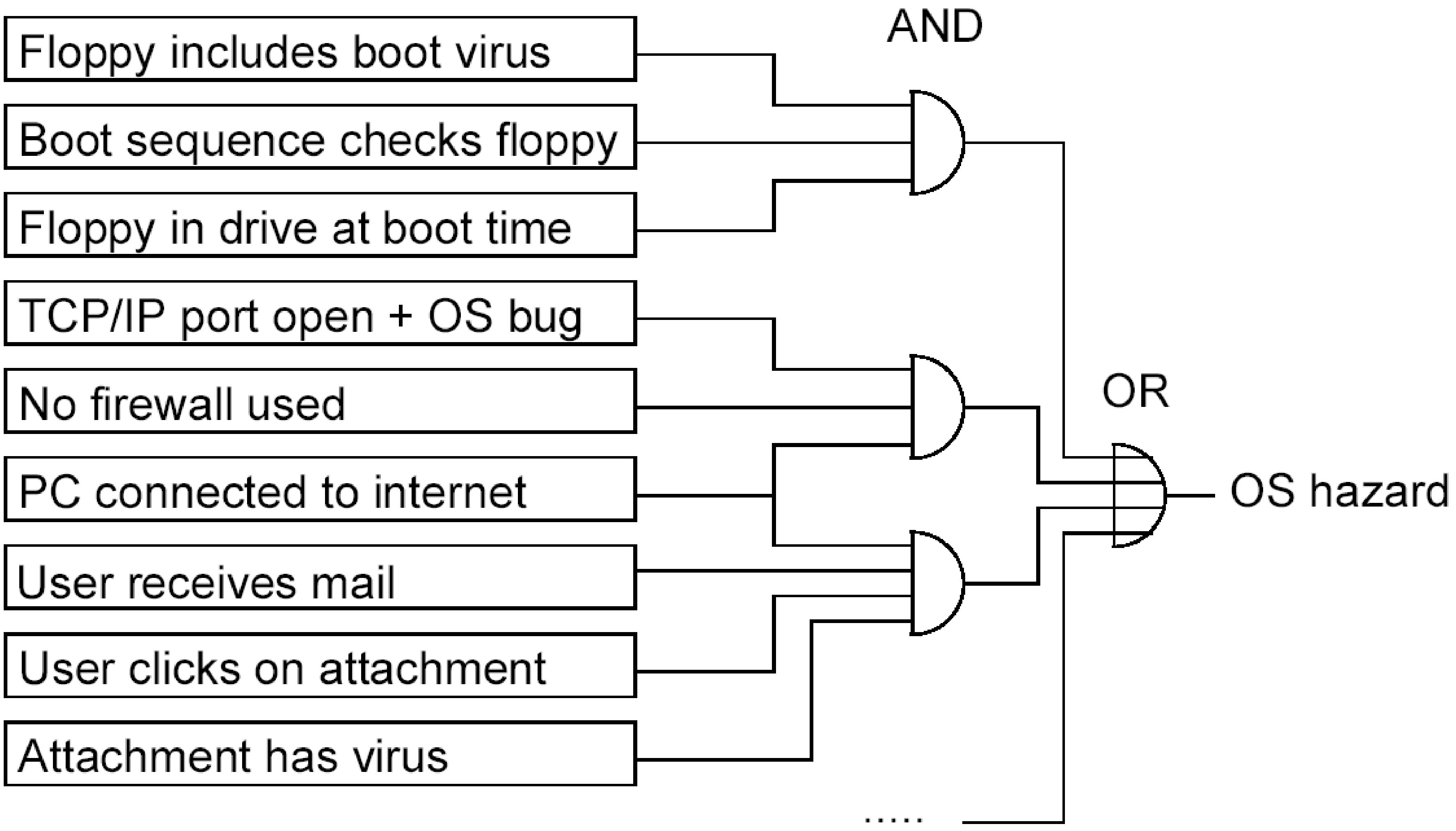
Analysis frequently works with simplified models ☞

Fault tree Analysis (FTA)



- FTA is a top-down method of analyzing risks. Analysis starts with possible damage, tries to come up with possible scenarios that lead to that damage.
- FTA typically uses a graphical representation of possible damages, including symbols for AND- and OR-gates.
- OR-gates are used if a single event could result in a hazard.
- AND-gates are used when several events or conditions are required for that hazard to exist.

Example


AND
Floppy includes boot virus
Boot sequence checks floppy
Floppy in drive at boot time
TCP/IP port open + OS bug
No firewall used
PC connected to internet
User receives mail
User clicks on attachment
Attachment has virus
OR
OS hazard
.....

Limitations

The simple AND- and OR-gates cannot model all situations. For example, their modeling power is exceeded if shared resources of some limited amount (like energy or storage locations) exist.

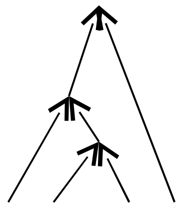
Markov models may have to be used to cover such cases.

Failure mode and effect analysis (FMEA)

- FMEA starts at the components and tries to estimate their reliability. The first step is to create a table containing components, possible faults, probability of faults and consequences on the system behavior.

Component	*Failure*	*Consequences*	*Probability*	*Critical?*
Processor	metal migration	no service	10^{-6} /h	yes
...	...	...	...	...

- Using this information, the reliability of the system is computed from the reliability of its parts (corresponding to a bottom-up analysis).



Safety cases

Both approaches may be used in “safety cases”. In such cases, an independent authority has to be convinced that certain technical equipment is indeed safe.
One of the commonly requested properties of technical systems is that no single failing component should potentially cause a catastrophe.

Fault injection

Fault simulation may be too time-consuming

☞ If real systems are available, faults can be injected.

Two types of fault injection:

1. local faults within the system, and
2. faults in the environment (behaviors which do not correspond to the specification). For example, we can check how the system behaves if it is operated outside the specified temperature or radiation ranges.

Physical fault injection

Hardware fault injection requires major effort, but generates precise information about the behavior of the real system.
3 techniques compared in the PDCS project on the MARS hardware [Kopetz]:

Injection Technique	**Heavy-ion**	**Pin-level**	**EMI**
Controllability, space	Low	High	Low
Controllability, time	None	High/medium	Low
Flexibility	Low	Medium	High
Reproducibility	Medium	High	Low
Physical reachability	High	Medium	Medium
Timing measurement	Medium	high	Low

Software fault injection

Errors are injected into the memories.

Advantages:

- **Predictability:** it is possible to reproduce every injected fault in time and space.
- **Reachability:** possible to reach storage locations within chips instead of just pins.
- **Less effort** than physical fault injection: no modified hardware.

Same quality of results?

Dependability requirements

Allowed failures may be in the order of 1 failure per 10^9 h.

~ 1000 times less than typical failure rates of chips.

☞ For safety-critical systems, the system as a whole must be more dependable than any of its parts.

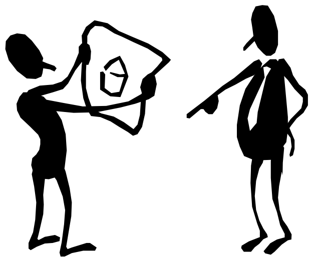
☞ fault-tolerance mechanisms must be used.

Low acceptable failure rate → systems not 100% testable.

☞ Safety must be shown by a combination of testing and reasoning. Abstraction must be used to make the system explainable using a hierarchical set of behavioral models. Design faults and human failures must be taken into account.

Kopetz‘s 12 design principles (1-3)

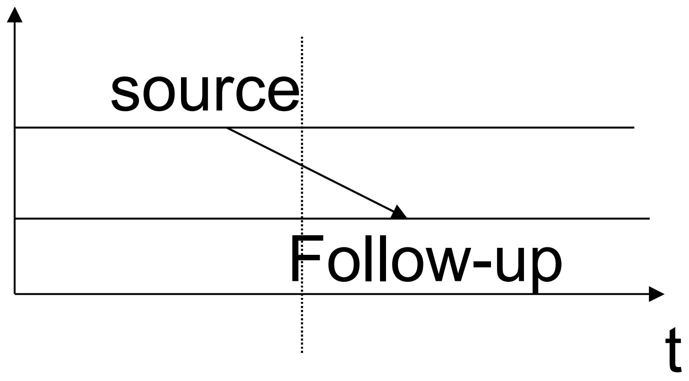
1. Safety considerations may have to be used as the important part of the specification, driving the entire design process.
2. Precise specifications of design hypotheses must be made right at the beginning. These include expected failures and their probability.
3. Fault containment regions (FCRs) must be considered. Faults in one FCR should not affect other FCRs.

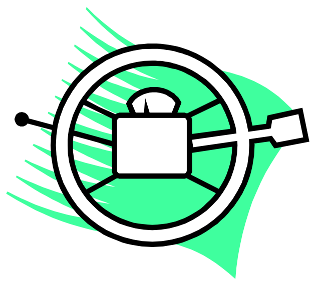


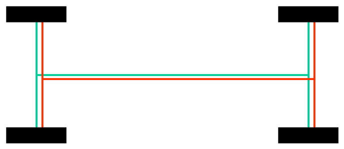
Safety-critical & non-safety critical electronics

Kopetz's 12 design principles (4-6)

4. A consistent notion of time and state must be established. Otherwise, it will be impossible to differentiate between original and follow-up errors.
5. Well-defined interfaces have to hide the internals of components.
6. It must be ensured that components fail independently.





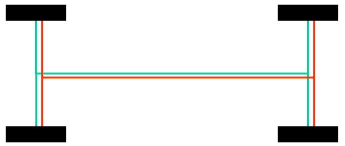


2 independent brake hose systems

technische universität dortmund
fakultät für informatik

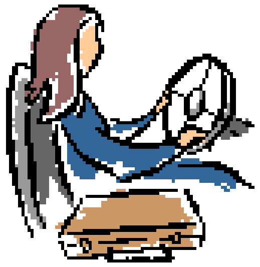
Kopetz‘s 12 design principles (7-9)

7. Components should consider themselves to be correct unless two or more other components pretend the contrary to be true (principle of self-confidence).
8. Fault tolerance mechanisms must be designed such that they do not create any additional difficulty in explaining the behavior of the system. Fault tolerance mechanisms should be decoupled from the regular function.
9. The system must be designed for diagnosis. For example, it has to be possible to identifying existing (but masked) errors.


one of the systems sufficient for braking

Kopetz‘s 12 design principles (10)

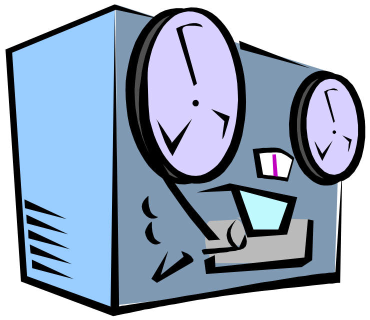
10. The man-machine interface must be intuitive and forgiving. Safety should be maintained despite mistakes made by humans



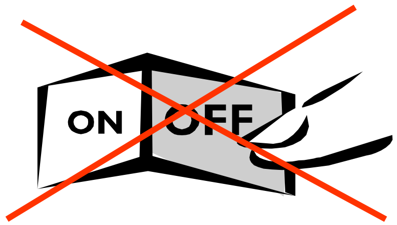
airbag

Kopetz‘s 12 design principles (11-12)

11. Every anomaly should be recorded. These anomalies may be unobservable at the regular interface level. Recording to involve internal effects, otherwise they may be masked by fault-tolerance mechanisms.



12. Provide a never-give up strategy. ES may have to provide uninterrupted service. Going offline is unacceptable.



Chapter 6.8

Formal verification

- Formal verification = formally proving a system correct, using the language of mathematics.
- Formal model required. Obtaining this cannot be automated.
- Model available ☞ try to prove properties.
- Even a formally verified system can fail (e.g. if assumptions are not met).
- Classification by the type of logics.

Ideally: Formally verified tools transforming specifications into implementations („*correctness by construction*“).

In practice: Non-verified tools and manual design steps ☞ validation of each and every design required

Unfortunately has to be done at intermediate steps and not just for the final design ☞ Major effort required.

Propositional logic (1)

- Consisting of Boolean formulas comprising Boolean variables and connectives such as $\vee$ and $\wedge$.
- Gate-level logic networks can be described.
- Typical aim: checking if two models are equivalent (called **tautology checkers** or **equivalence checkers)**.
- Since propositional logic is decidable, it is also decidable whether or not the two representations are equivalent.
- Tautology checkers can frequently cope with designs which are too large to allow simulation-based exhaustive validation.

Propositional logic (2)

- Reason for power of tautology checkers: Binary Decision Diagrams (BDDs)
- Complexity of equivalence checks of Boolean functions represented with BDDs: *O*(*number of BDD-nodes*) (equivalence check for sums of products is NP-hard).
  #(BDD-nodes) not to be ignored!
- Many functions can be efficiently represented with BDDs. In general, however, the #(nodes) of BDDs grows exponentially with the number of variables.
- Simulators frequently replaced by equivalence checkers if functions can be efficiently represented with BDDs.
- Very much limited ability to verify FSMs.

First order logic (FOL)

FOL includes quantification, using ∃ and ∀.
Some automation for verifying FOL models is feasible.
However, since FOL is undecidable in general, there may be cases of doubt.

Higher order logic (HOL)

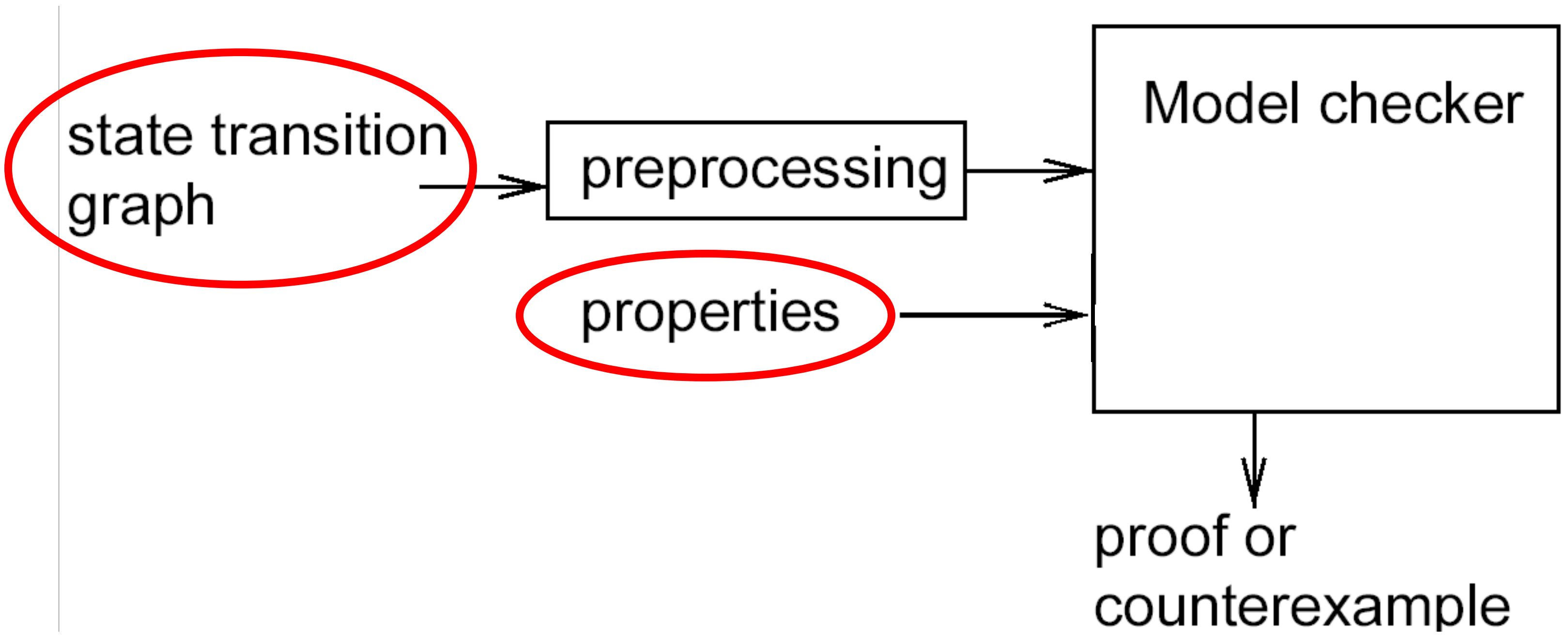
Higher order logic allows functions to be manipulated like other objects.
For higher order logic, proofs can hardly ever be automated and typically must be done manually with some proof-support.

Model checking

Aims at the verification of finite state systems.
Analyzes the state space of the system.
Verification using this approach requires three stages:

- generation of a model of the system to be verified,
- definition of the properties expected, and
- model checking (the actual verification step).

2 types of input



Verification tools can prove or disprove the properties.
In the latter case, they can provide a counter-example.
Example: Clarke's EMC-system

Computation tree logic (CTL)

Let V be a set of atomic propositions
CTL formulas are defined recursively:

1. Every atomic proposition is a formula
2. If f_1 and f_2 are CTL formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, AX f_1, EX f_1, A[f_1 U f_2] and E[f_1 U f_2]

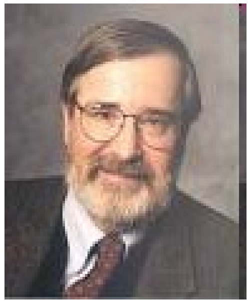
- AX f_1 means: holds in state $s°$ iff f_1 holds in all successor states of $s°$
- EX f_1 means: There exists a successor such that f_1 holds
- A[f_1 U f_2] means: always until.
- E[f_1 U f_2] means: There exists a path such that f_1 holds until is f_2 satisfied.

Christoph Kern and Mark R. Greenstreet: Formal Verification In Hardware Design: A Survey, ACM Transactions on Design Automation of Electronic Systems, Vol. 4, No. 2, April 1999, Pages 123–193.

Computational properties

- Model checking is easier to automate than FOL.
- In 1987, model checking was implemented using BDDs.
- It was possible to locate several errors in the specification of the *future bus* protocol.
- Extensions are needed in order to also cover real-time behavior and numbers.

ACM Turing award 2008 granted for basic work on model checking



Edmund M. Clarke, CMU, Pittsburgh

E. Allen Emerson, U. Texas at Austin

Joseph Sifakis, VERIMAG, Grenoble

Summary

Evaluation and Validation

- WCET estimation
  - Example: aiT (based on abstract interpretation)
- Energy models
  - Examples: Steinke’s instruction set-based model, CACTI
- Risk and dependability analysis
  - Failure rates, reliability, MTBF, MTTF, MTTR
  - Fault tree analysis, FMEA
- Formal verification
  - Propositional, first order, higher order based techniques,
  - model checking
- Fault injection
  - Software and hardware-based techniques