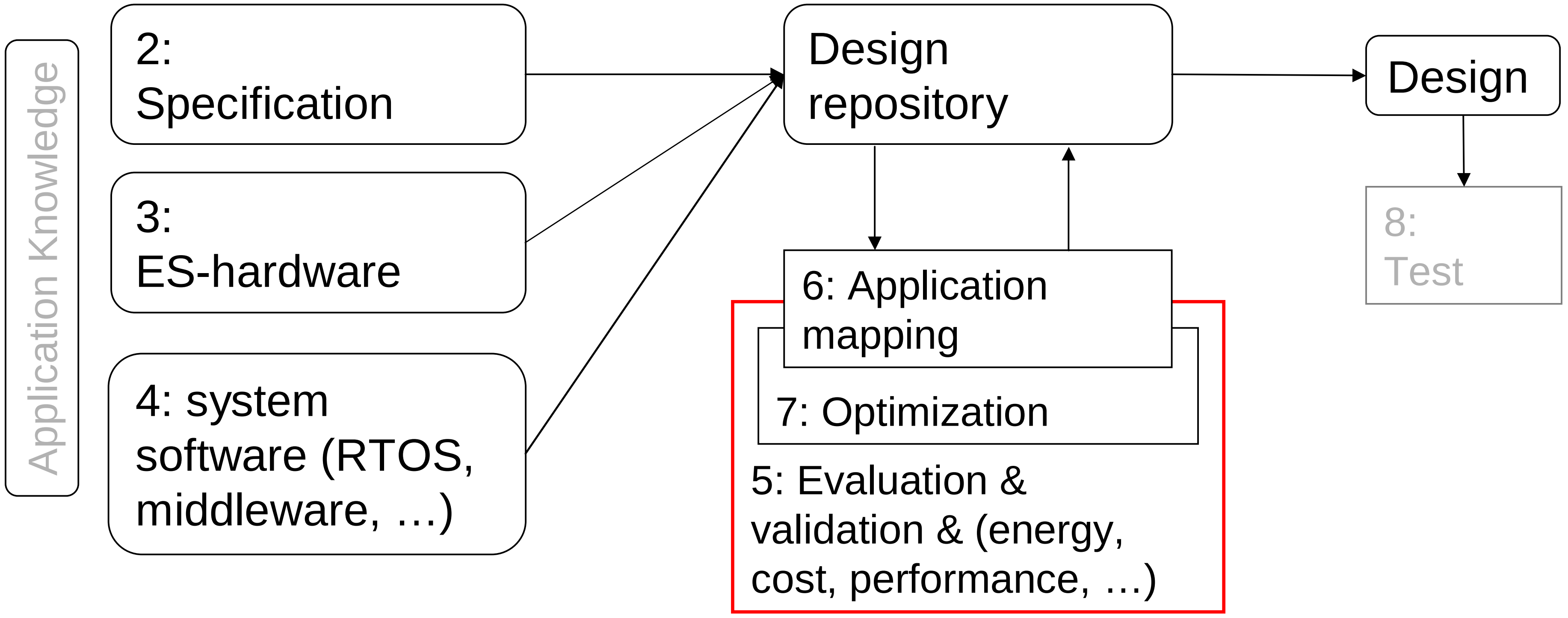
Evaluation and Validation

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2012 年 12 月 11 日

Structure of this course

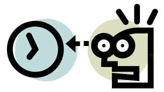


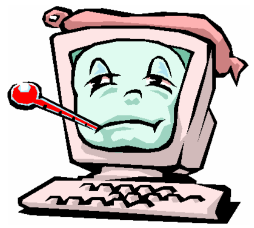
Numbers denote sequence of chapters

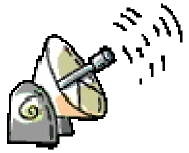
How to evaluate designs according to multiple criteria?

Many different criteria are relevant for evaluating designs:

- average & worst case delay
- power/energy consumption
- thermal behavior
- reliability, safety, security
- cost, size
- weight, numerical precision
- EMC characteristics
- radiation hardness, environmental friendliness, ..







How to compare different designs?
(Some designs are “better” than others)

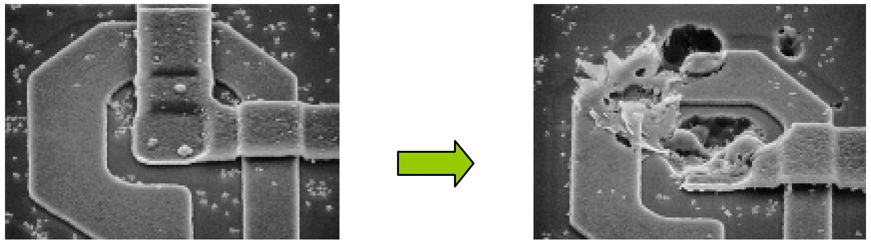
Impact of shrinking feature sizes

- Reduced reliability due to smaller patterns within semiconductor chips [ITRS, 2009]
- Transient & permanent faults

Types of faults: Example: Electro-migration

Example : metal migration @ Pentium 4

www.jrwhipple.com/computer_hangs.html



- Rate of faults expected to increase such that designs need to become fault-tolerant

Terms

- “*A **service failure**, often abbreviated here to **failure**, is an event that occurs when the delivered service of a system deviates from the correct service.”*
- *“The definition of an **error** is the part of the total state of the system that may lead to its subsequent service failure”.*
- “*The adjudged or hypothesized cause of an error is called a **fault**. Faults can be internal or external of a system.*”

Example:

- Transient **fault** flipping a bit in memory.
- After this bit flip, the memory cell will be in **error**.
- **Failure***:* if the system service is affected by this error.

We will consider **failure** rates & **fault** models.

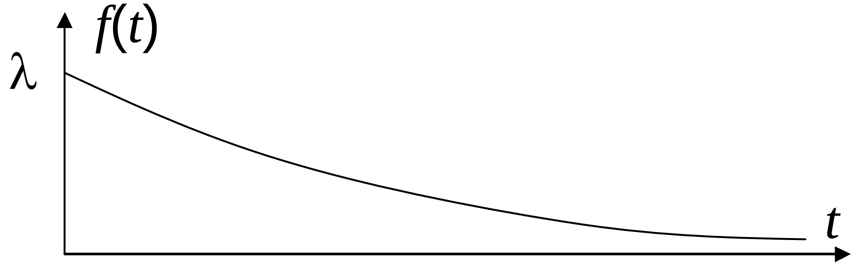
[Laprie et al., 1992, 2004]

Reliability: $f(t)$, $F(t)$

- Let T: time until first failure (random variable)
- Let $f(t)$ be the density function of T

Example: Exponential distribution

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



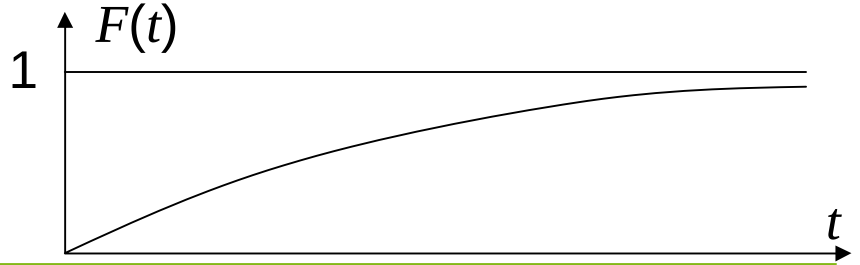
- $F(t)$ = probability of the system being faulty at time t:

$F(t) = \Pr(T \le t)$

$$F(t)=\int_0^t f(x)dx$$

Example: 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:

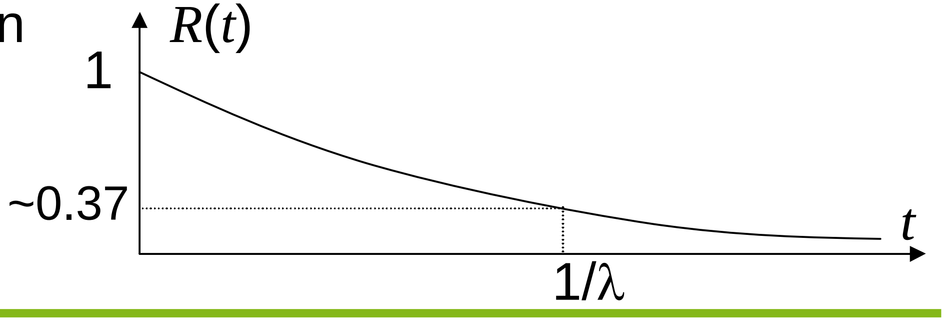
$R(t)=\Pr(T>t),\ t\geq 0$ $$R(t)=\int_t^\infty f(x)dx$$

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

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

Example: Exponential distribution

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



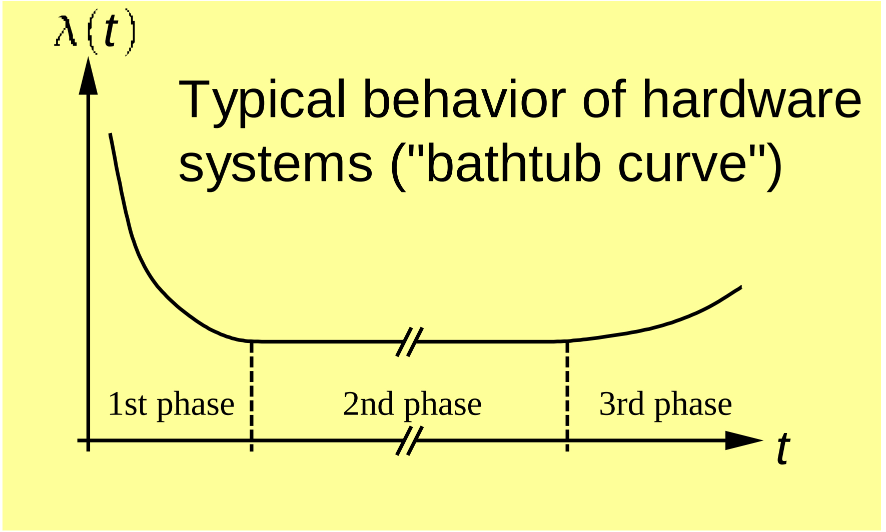
Failure rate

The failure rate at time t is the probability of the system failing between time t and time $t+\Delta t$:

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

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

$\Pr(A|B)=\Pr(AB)/\Pr(B)$



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.

FIT & “10^{-9}“ (per hour)

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

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

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

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

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

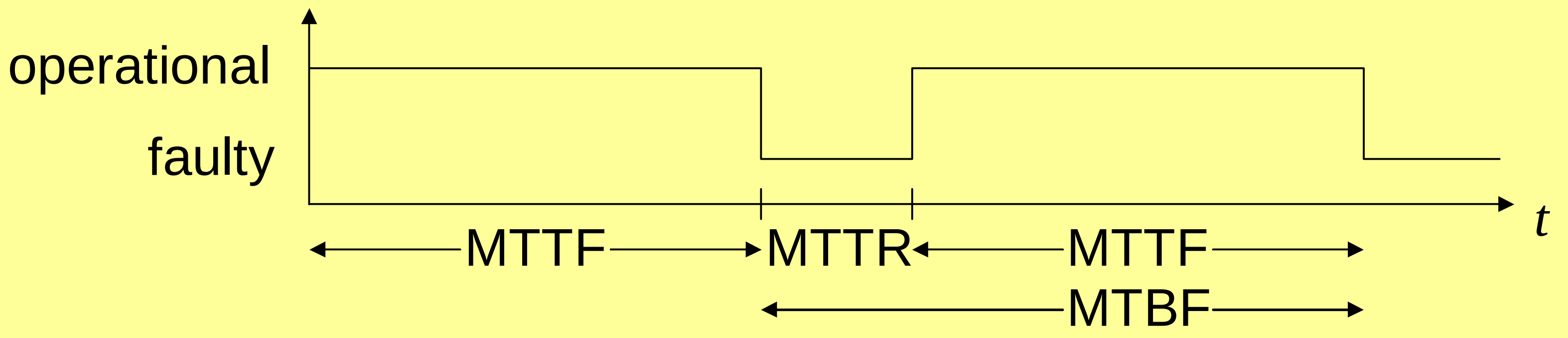
MTTF is the reciprocal value of failure rate.

MTTF, MTTR and MTBF

MTTR = mean time to repair (average over repair times using distribution $M(d)$)

MTBF* = mean time between failures = MTTF + MTTR

Ignoring the statistical nature of failures …



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

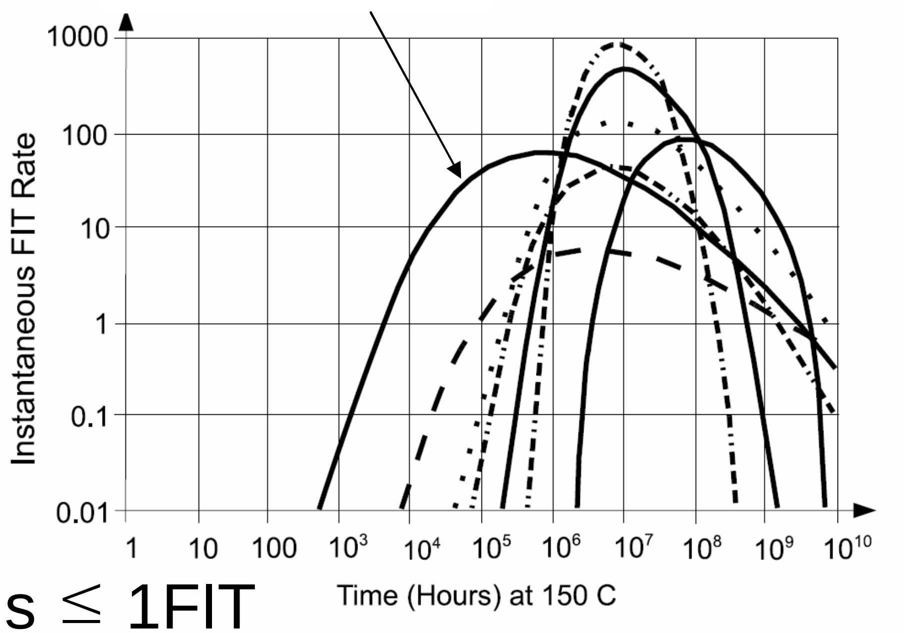
* Mixed up with MTTF, if starting in operational state is implicitly assumed

Actual failure rates

Failure rates derived from experiments at higher temperatures.

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]

Different devices



Target: Failures rates of systems $\leq$ 1FIT

Reality: Failures rates of circuits $\leq$ 100 FIT

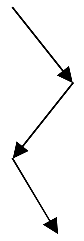
☞redundancy is required to make a system more reliable than its components

$\exists$ non-constant failure rates!

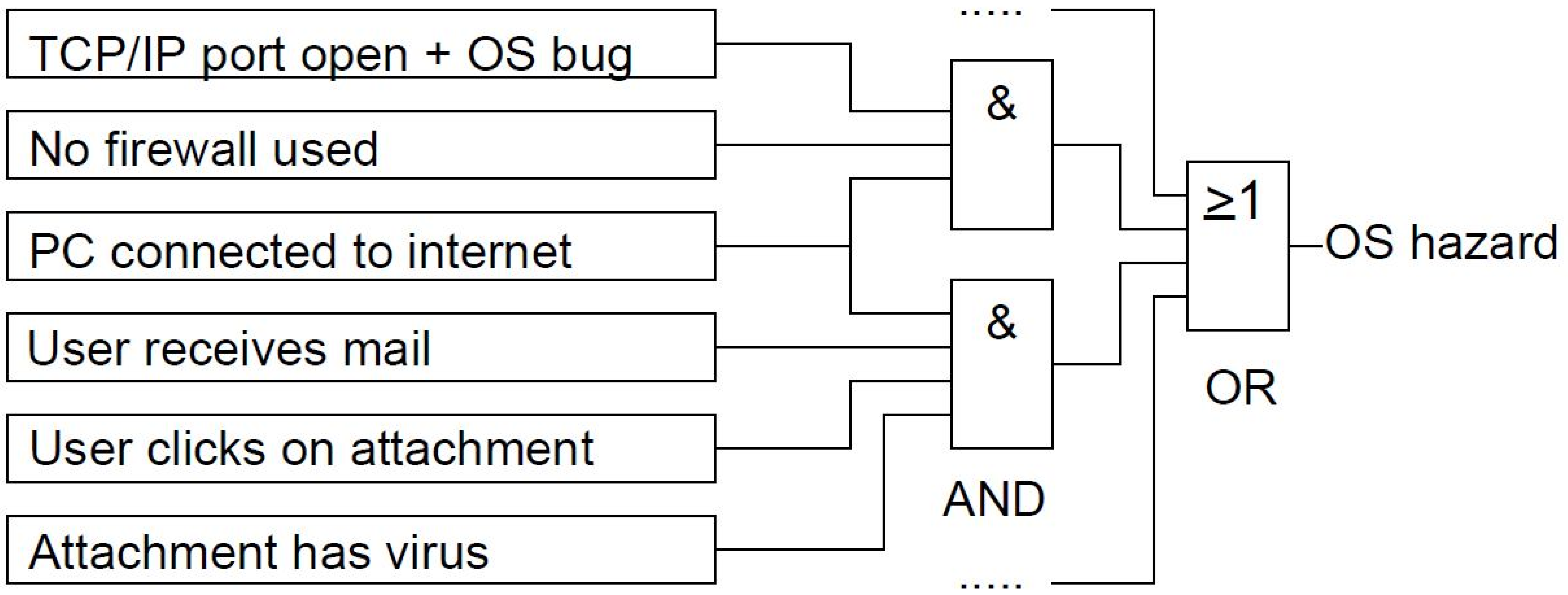
Fault tree Analysis (FTA)

Damages are resulting from hazards/risks.
For every damage there is a severity and a probability.
Several techniques for analyzing risks.

- 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


TCP/IP port open + OS bug
No firewall used
PC connected to internet
User receives mail
User clicks on attachment
Attachment has virus
.....
&
&
AND
.....
≥1
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.

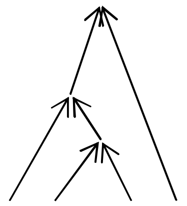
Markov models may have to be used to cover such cases.

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^{-7} /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.

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.

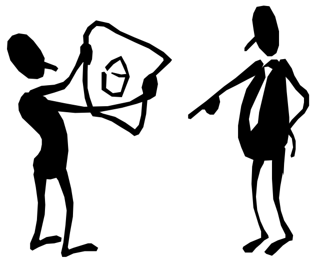
☞ fault-tolerance mechanisms must be used.

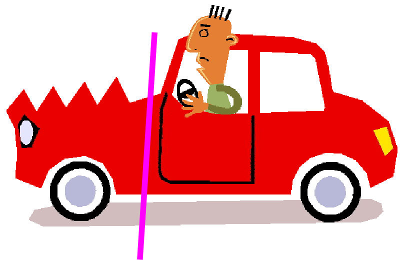
Low acceptable failure rate → systems not 100% testable.

☞ 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.



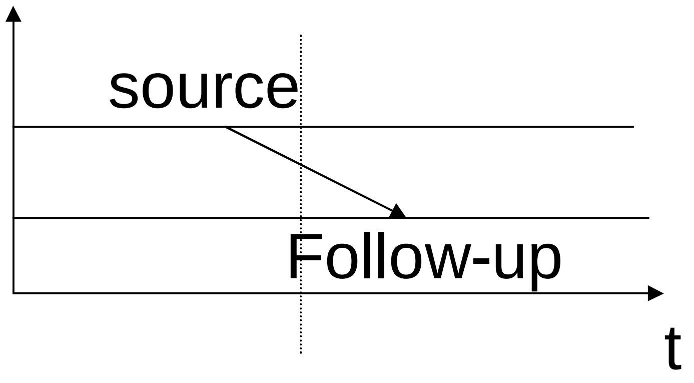


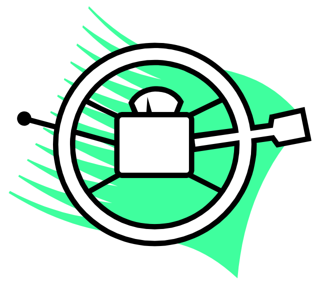
Passenger compart-ment stable

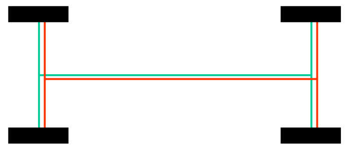
Safety-critical & non-safety critical electronics

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.



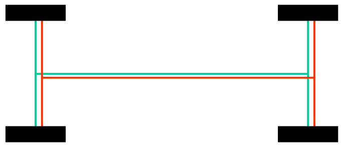




2 independent brake hose systems

Kopetz‘s 12 design principles (7-9)

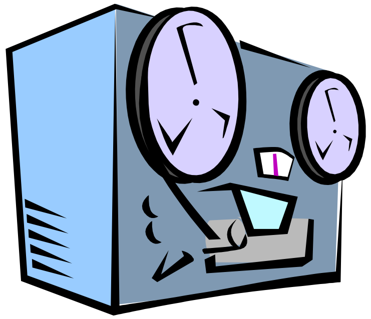
7. Components should consider themselves to be correct unless two or more other components pretend the contrary to be true (principle of self-confidence).
8. Fault tolerance mechanisms must be designed such that they do not create any additional difficulty in explaining the behavior of the system. Fault 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

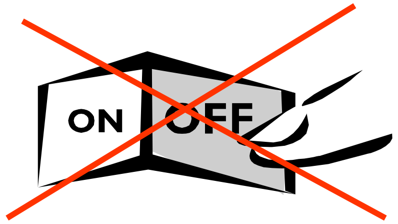
Kopetz‘s 12 design principles (10-12)

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

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.



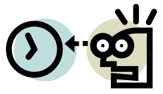
12. Provide a never-give up strategy. ES may have to provide uninterrupted service. Going offline is unacceptable.

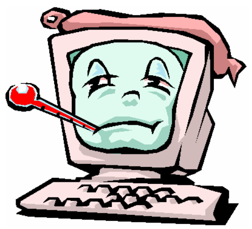
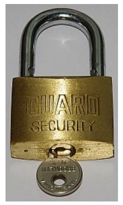


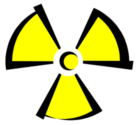
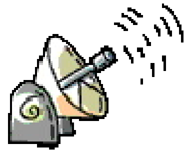
How to evaluate designs according to multiple criteria?

Many different criteria are relevant for evaluating designs:

- average & worst case delay
- power/energy consumption
- thermal behavior
- reliability, safety, security
- cost, size
- weight, numerical precision
- EMC characteristics
- radiation hardness, environmental friendliness, ..



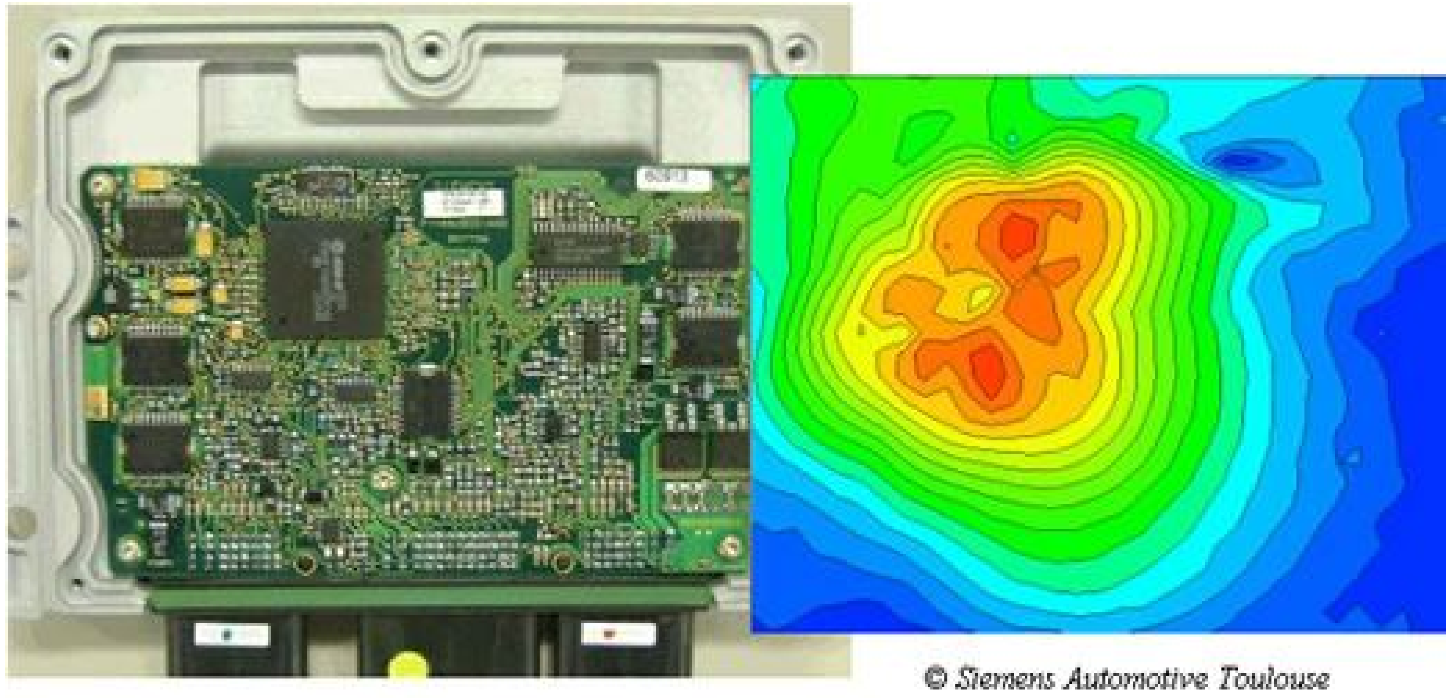



How to compare different designs?
(Some designs are "better" than others)

Electro-magnetic compatibility (EMC)

Example: car engine controller



Red: high emission; Validation of EMC properties often done at the end of the design phase.

Source: http://intrage.insa-tlse.fr/~etienne/emccourse/what_for.html

Simulations

- Simulations try to imitate the behavior of the real system on a (typically digital) computer.
- Simulation of the functional behavior requires executable models.
- Simulations can be performed at various levels.
- Some non-functional properties (e.g. temperatures, EMC) can also be simulated.
- Simulations can be used to **evaluate** and to **validate** a design

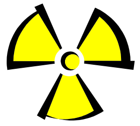
Validating functional behavior by simulation

Various levels of abstractions used for simulations:

- High-level of abstraction: fast, but sometimes not accurate
- Lower level of abstraction: slow and typically accurate
- Choosing a level is always a compromise

Simulations: Limitations

- Typically slower than the actual design.
  ☞ **Violations of timing constraints** likely if simulator is connected to the actual environment
- Simulations in the real environment may be **dangerous**
- There may be huge amounts of data and it may be impossible to simulate enough data in the available time.
- Most actual systems are too complex to allow simulating all possible cases (inputs).
  Simulations can help finding errors in designs, but they cannot guarantee the absence of errors.



Rapid prototyping/Emulation

- Prototype: Embedded system that can be generated quickly and behaves very similar to the final product.
- May be larger, more power consuming and have other properties that can be accepted in the validation phase
- Can be built, for example, using FPGAs.

Example: Quickturn Cobalt System (1997), ~0.5M$ for 500kgate entry level system

Source & ©: http://www. eedesign. com/editorial/1997/toolsandtech9703.html

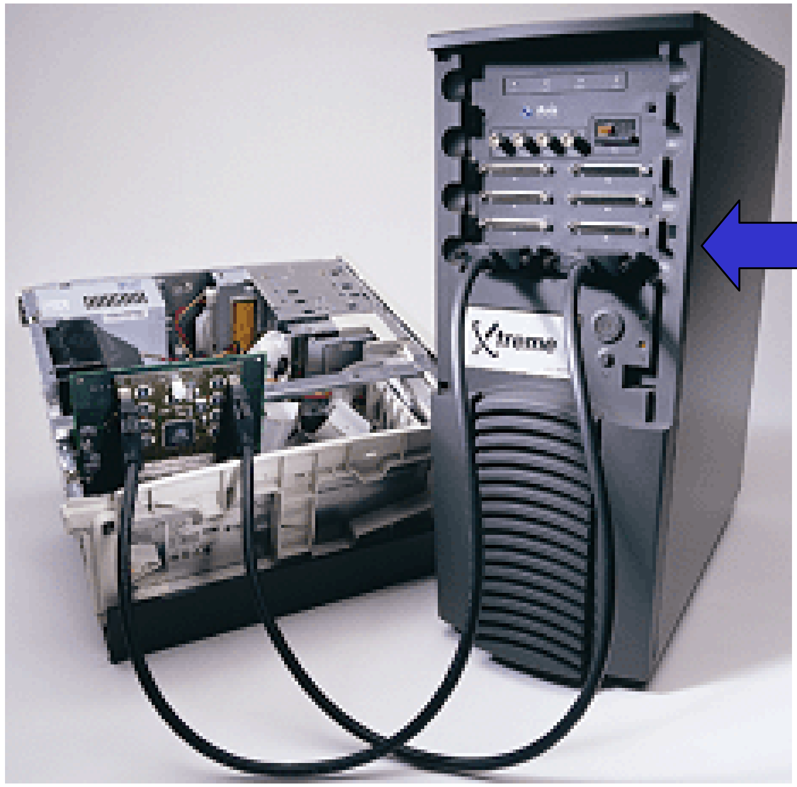
Emulation

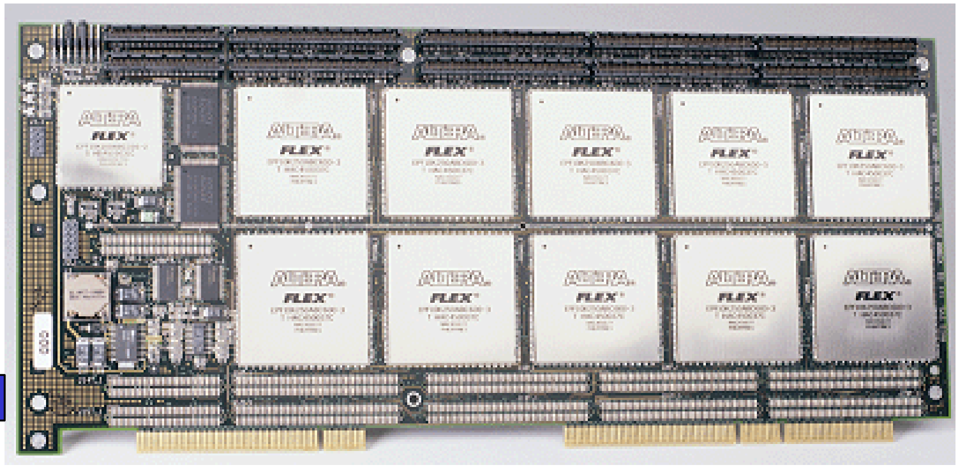
- Simulations: based on models, which are approximations of real systems.
- In general: ∃ difference between real system and model.
- Reduce gap by implementing parts of SUD more precisely!

Definition: Emulation is the process of executing a model of the SUD where at least one component is **not** represented by simulation on some kind of host computer.

"*Bridging the credibility gap is not the only reason for a growing interest in emulation—the above definition of an emulation model remains valid when turned around— an emulation model is one where part of the real system is replaced by a model. Using emulation models to test control systems under realistic conditions, by replacing the* "real system" *with a model, is proving to be of considerable interest …* [McGregor, 2002]

Example of a recent commercial emulator





[www.verisity.com/images/products/xtremep{1|3}.gif]

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)

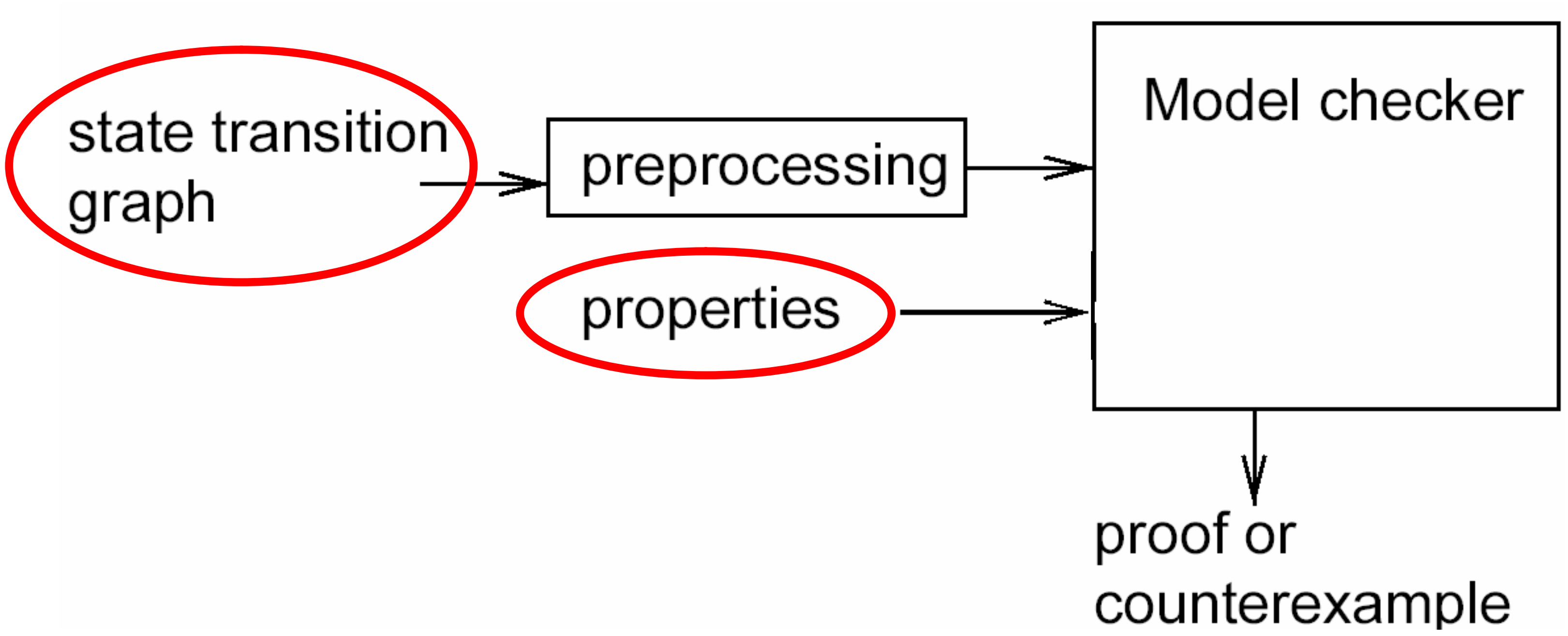
Higher order logic allows functions to be manipulated like other objects.
For higher order logic, proofs can hardly ever be automated and typically must be done manually with some proof-support.

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

Examples

1.

$M,s \vDash AGg$

means:

in the transition graph M, property g holds for all paths (denoted by A) and all states (denoted by G).

2.

For the Thalys example, we could prove that the number of trains is indeed constant.

Computational properties

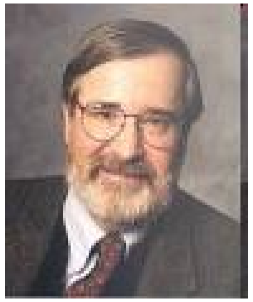
- Model checking is easier to automate than FOL.
- In 1987, model checking was implemented using BDDs.
- It was possible to locate several errors in the specification of the *future bus* protocol.
- Model checking becoming very popular
- Extensions are needed in order to also cover real-time behavior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

Summary

Evaluation and Validation:

- Reliability
  - Definitions
  - Failure rates
  - MTBF, MTTF, MTTR
  - Fault tree analysis, FMEA
  - Kopetz' 12 principles
- Electro-magnetic compatibility (briefly)
- Simulation, Emulation
- Formal verification
  - Propositional,
  - first order, higher order based techniques,
  - model checking

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