

Evaluation and Validation

Peter Marwedel
TU Dortmund, Informatik 12
Germany

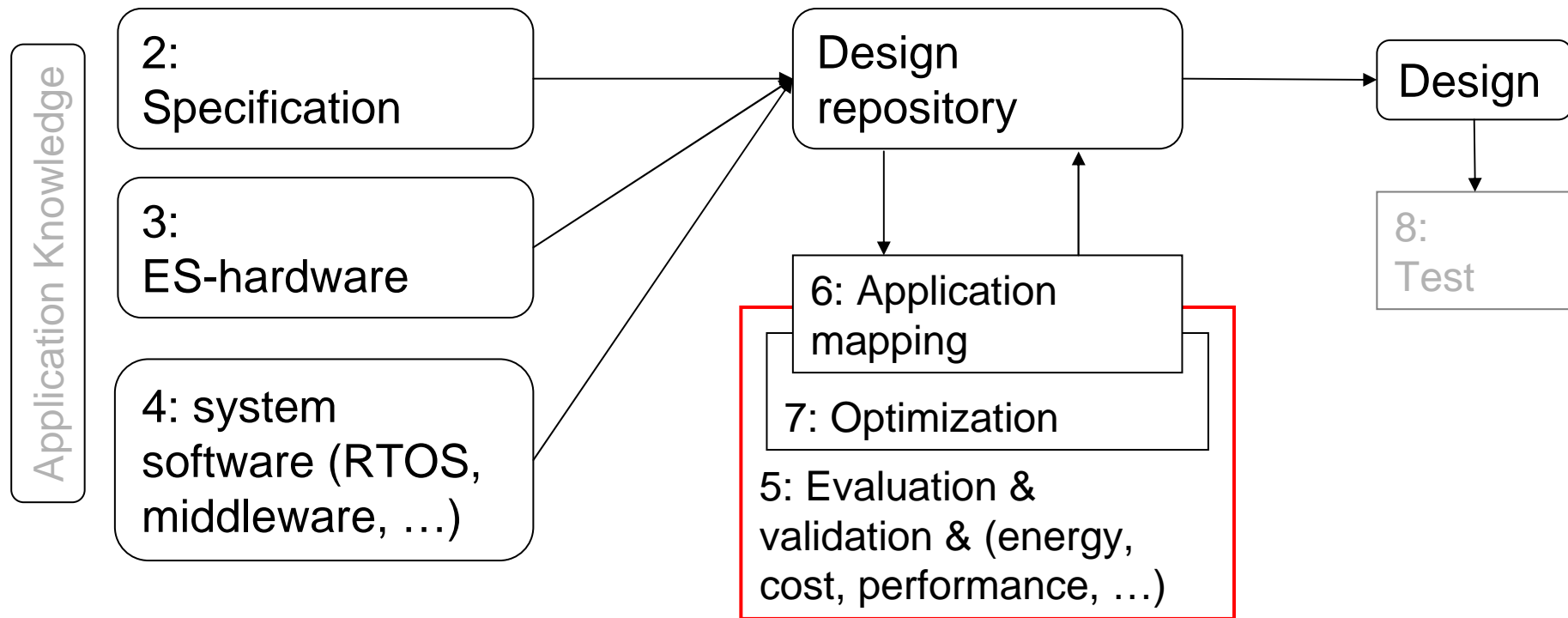


Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

2010年 12 月 05 日

These slides use Microsoft clip arts.
Microsoft copyright restrictions apply.


Structure of this course



Numbers denote sequence of chapters

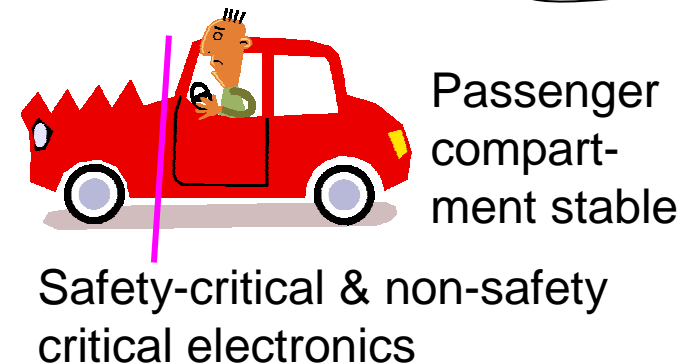
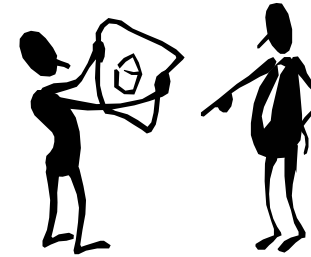
Evaluation of designs according to multiple objectives

Different design objectives/criteria are relevant:

- Average performance
 - Worst case performance
 - Energy/power consumption
 - Thermal behavior
 - Reliability
 - Electromagnetic compatibility
 - Numeric precision
 - Testability
 - Cost
 - Weight, robustness, usability, extendibility, security, safety, environmental friendliness
- 

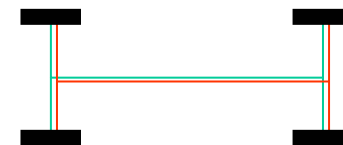
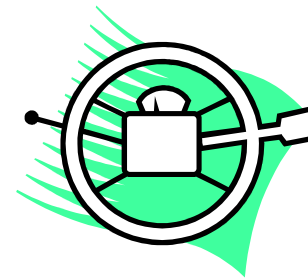
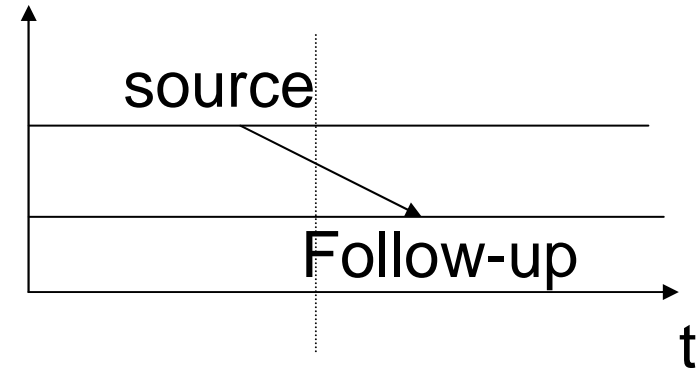
Kopetz's 12 design principles (1-3)

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.



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.

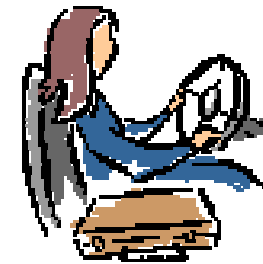


one of the systems
sufficient for braking



Kopetz's 12 design principles (10)

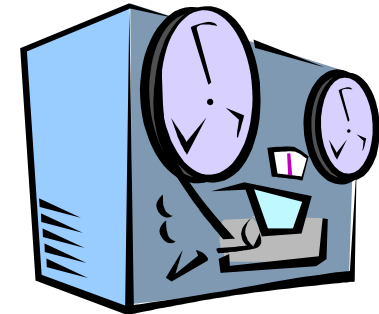
10. The man-machine interface must be intuitive and forgiving. Safety should be maintained despite mistakes made by humans



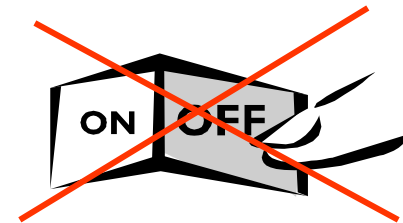
airbag

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.



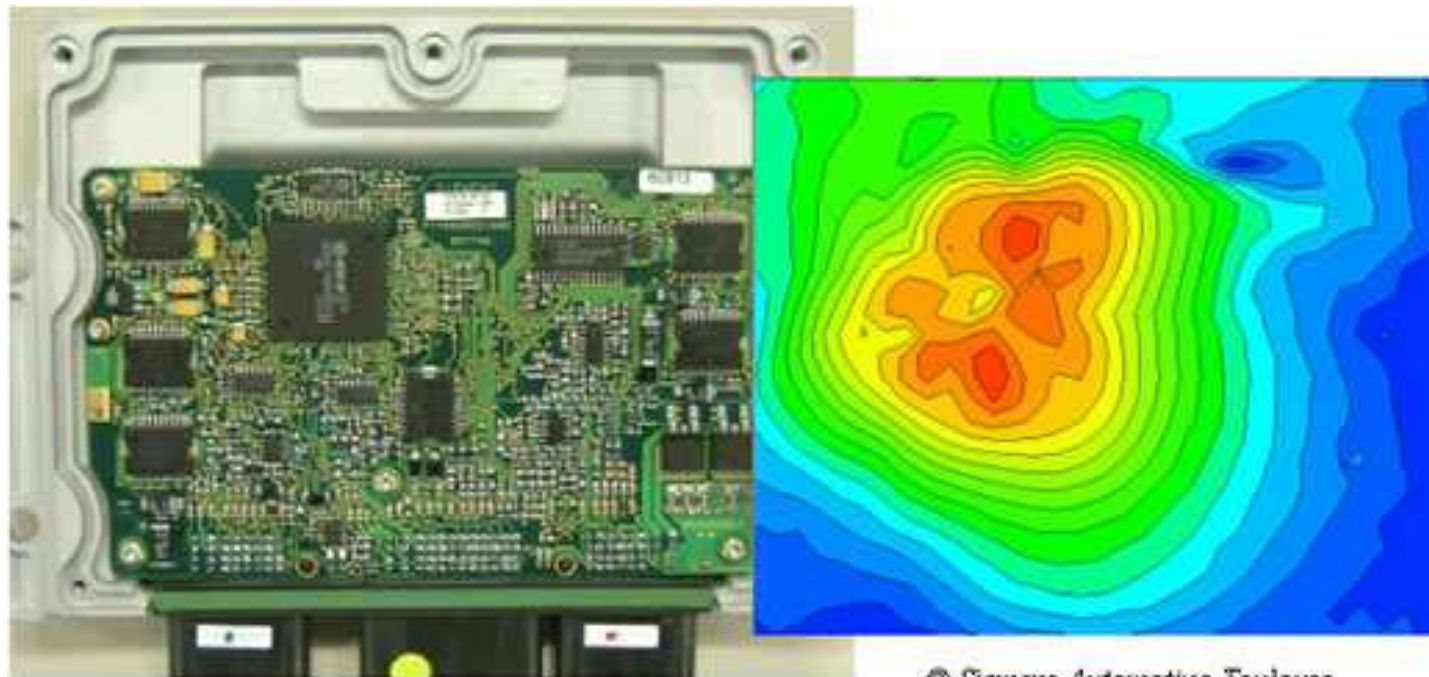
Evaluation of designs according to multiple objectives

Different design objectives/criteria are relevant:

- Average performance
- Worst case performance
- Energy/power consumption
- Thermal behavior
- Reliability
- Electromagnetic compatibility 
- Numeric precision
- Testability
- Cost
- Weight, robustness, usability, extendibility, security, safety, environmental friendliness

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://intrade.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

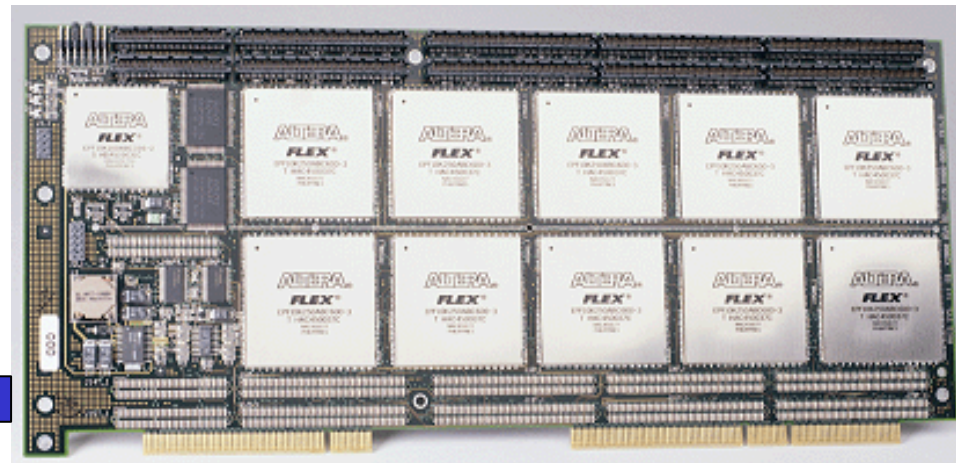
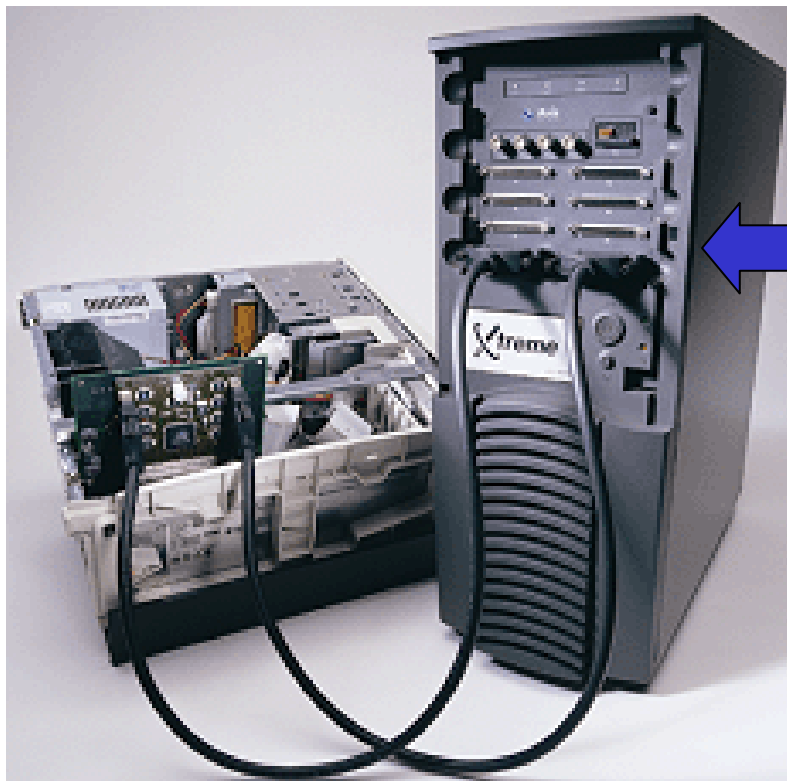
- Simulations: based on models, which are approximations of real systems.
- In general: some difference between the real system and the model.
- Reduce gap by implementing some parts of our SUD more precisely!

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 more 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: $O(\text{number of BDD-nodes})$ (equivalence check for sums of products is NP-hard). $\#(\text{BDD-nodes})$ not to be ignored!
- Many functions can be efficiently represented with BDDs. In general, however, the $\#(\text{nodes})$ 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 \exists and \forall .

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.

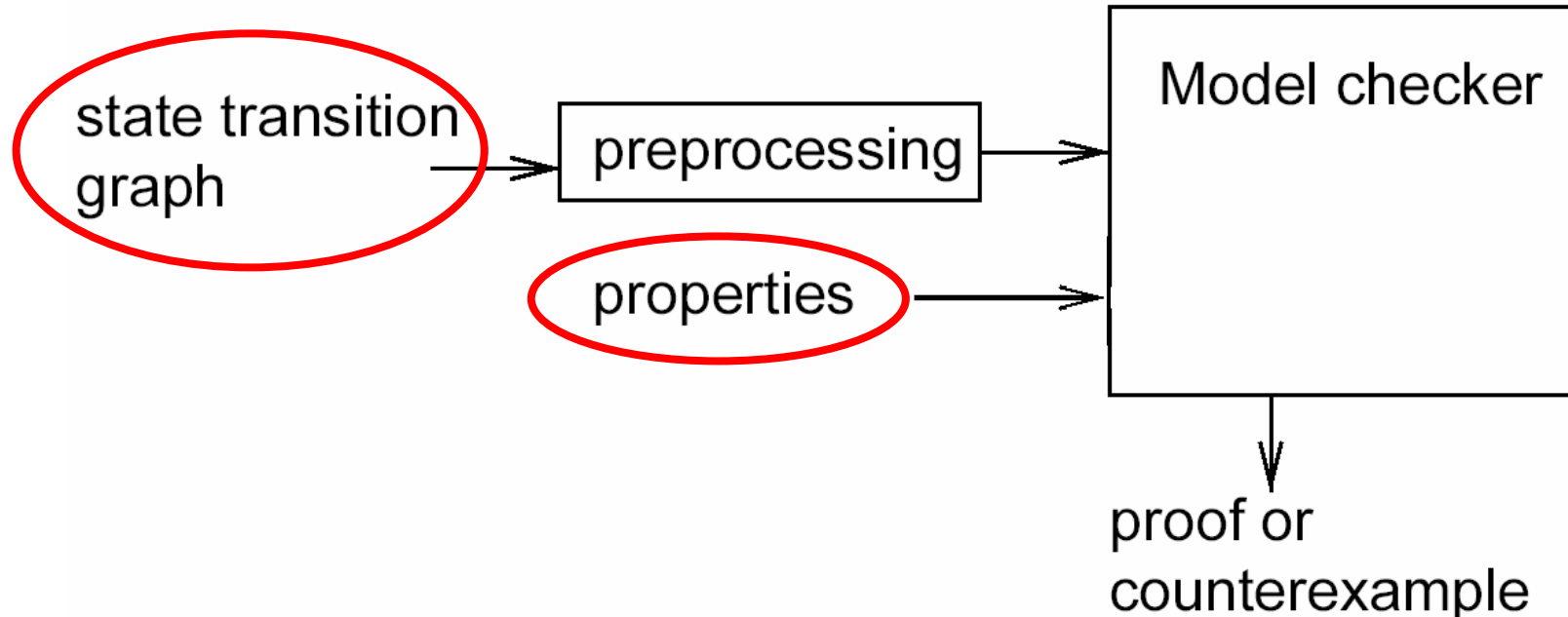
Model checking

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.

$$M, s \models AGg$$

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.

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

- Reliability
 - Kopetz' 12 principles
- Simulation
- Emulation
- Formal verification
 - Propositional,
 - first order,
 - higher order based techniques,
 - model checking