Altreonic NV

www.altreonic.com

Zen and the art of safety engineering

From Deep Space to Deep Sea

Push Button High Reliability

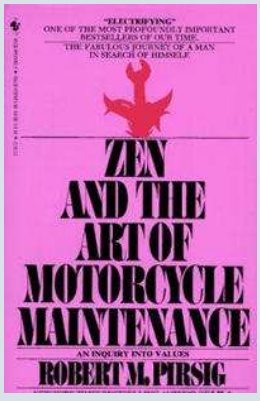
Content

- Zen and quality
- What is safety?
- Safety for mobility

- Analysis of the car pedal issue
- State explosion and safety features
- Conclusion

The safe car that doesn't move

Zen and the art of motorcycle maintenance



Book is about the "**metaphysics of Quality**"

To remember:

Quality is an emerging property of a system.
It requires a holistic approach.
It requires a deep understanding of the system, of how it works, how it is used and how it is to be maintained.

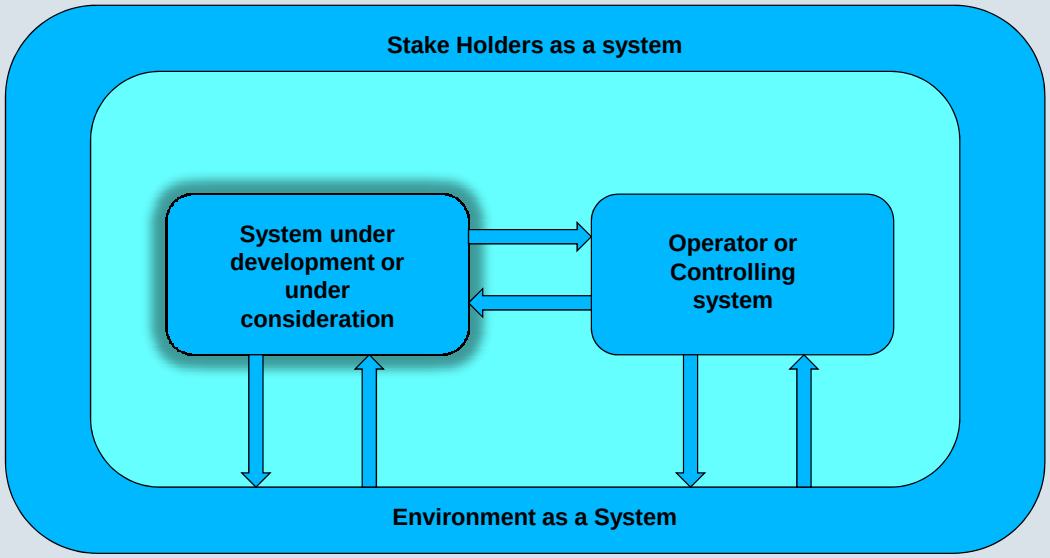
Two main concepts of Quality:
- Production focused (e.g. Deming)

More recent:
- Quality of design/development process

Safety (trustworthiness) links the two

What system?

Any system is part of a larger system



What system properties?

Any system has to meet different, often conflicting properties:

- Cost price
- Energy use
- Safety
- Security
- Size
- Ease of use
- Designed for “production”
- Must be produced by company X
- Designed to meet the requirements of the target market
- Life-cycle cost
- ….

⇒ **Trade-offs**

⇒ All properties are related

⇒ The best technical solution is not necessarily the one selected

What is safety?

- **Historical:**
  - Concept that is related to the loss of lives due to a malfunctioning of the system
  - Often post factum: what went wrong?
  - Hence safety engineering standards emphasize tracebility
- **The right safety view:**
  - **Safety is an emerging system property resulting from a quality engineering process**
  - **Reliability is a pre-condition, but not sufficient condition**
  - Safety can be improved by using feedback
- **The expanded view: Trustworthiness =**
  - **Safety**: preventing damage or the loss of lives due to unintentional failures or malfunctioning parts +
  - **Security**: preventing damage or the loss of lives due to maliciously injected failures of malfunctions +
  - **Useability**: preventing damage or the loss of lives due to improper operator interfaces +
  - **Privacy**: preventing loss or misuse of personal data.

Safety, reliability, predictability

"Safety and reliability are different properties. One does not imply nor require the other : A system can be reliable but unsafe. It can also be safe but unreliable. In some cases, these two properties even conflict, that is, making the system safer may decrease reliability and enhancing reliability may decrease safety."

(src: *Nancy* Leveson, Engineering a Safer World)

Predictability => is a higher level property of any other property. It reflects our control of the engineering process.

Use case: Mobility aids

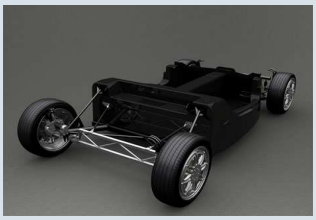
Future of transport is consumer-friendly

- Elderly customer base (mobility aid)
- Seamlessly Indoors ↔ Outdoors (other uses)
- Active safety (e.g. obstacle avoidance)
- Optimal use of road network

Intelligent Transport Systems , using cooperative Embedded Systems

- **100% trust-worthy**
- Fault Tolerance
- Heterogeneous network support
- Scalability
- Cost-Efficiency

Safe E-wheel control algorithms

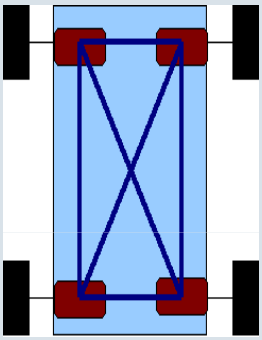


Key characteristics :

➢High Reliability (SIL3) → Fault Tolerance (SIL4)

All-in:

- Traction
- Braking
- Anti-slip
- Stability control
- Active suspension



➢ Distribute control => safety enabled SIL4 architecture

➢Software and Hardware redundancy enables

fault-tolerant controllers

1-, 2-, 3-, 4-, n-wheel platforms

Example of safety case

Risk of injury.

Under certain circumstances, due to a software issue, the product can unexpectedly apply reverse torque to the wheels, which can result in a rider falling and potentially suffering injuries. That this can occur in two situations: during a safety shutdown of the product, or when the rider exceeds the programmed speed limit.

Both situations involve specific sequences of events under narrow timeframes, and require that the handlebar be tilted back by the speed limiter and the rider come off and then back onto the rider detect switches on the riding platform within a short period of time combined with a traction control event. At least 6 incidents have been reported resulting in injuries to the head and wrist of users.

src: http://ec.europa.eu/consumers/dyna/rapex/rapex_archives_en.cfm

Is safety absolute?

1. **Safety can never be absolute:**
   - Always residual errors
   - Always residual risks
   - Design is always trade-off.
2. **Safety is a statistical property:**
   - Mean Time To Failure is what matters for malfuntions
   - => mean time to safety hazards
   - If MTTF >> life time, mainly external factors remain:
     - Operator
     - Environment
3. **Safety level must be selected on the basis of acceptable risks**
   - Has a cost tag (insurance) attached to it
   - Safety Integrity Level 3 (SIL3)
     - Fail-safe mode, but still safety hazard
   - Safety Integrity Level 4 (SIL4)
     - Fault tolerant, but still residual risks
       - Common mode failures
       - Common design mistakes

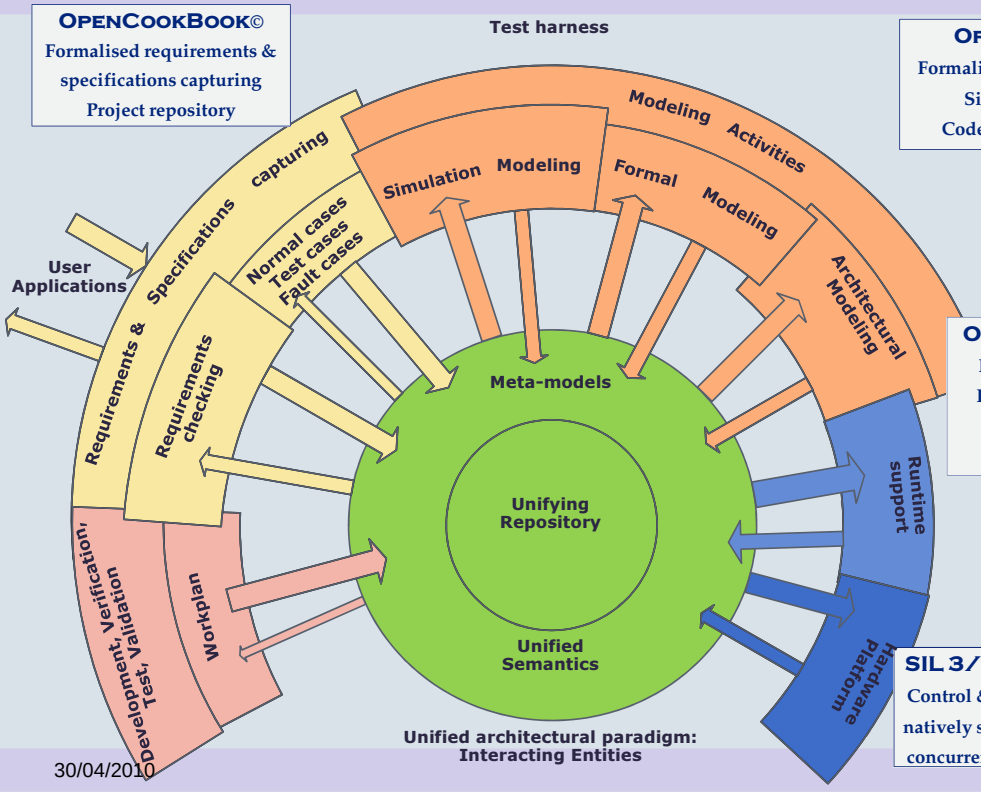
⇒ **A safety hazard can still happen ANY time!**

⇒ **Most people are optimistic and then become negligent**

How is safety reached?

1. Follow a formalised (safety) engineering process
   - Based on safety standard (often domain specific)
   - Organisation must be set up for it: mindset issue!
   - A good flow is iterative
     - Step1:
       - Requirements capturing
       - Many stakeholders, nice-to-haves, must haves
       - Normal case, Fault cases, Test cases
     - Step2:
       - Select requirements and write up specifications
     - Step3:
       - Model: (virtual) prototypes, simulations, formal models of critical properties, implementation models
     - Step4:
       - Verify the process
       - Test against specifications
       - Integrate and validate against requirements
2. Release
3. Certify

Unified Systems/Software engineering



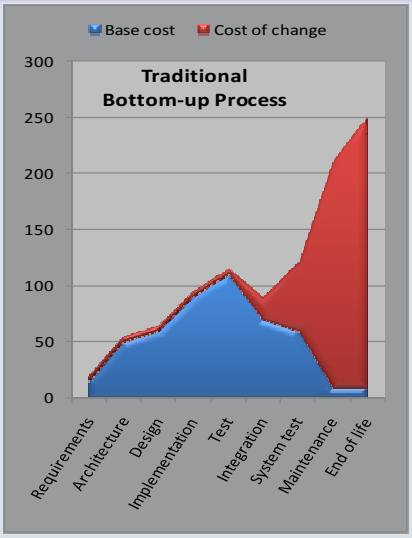
Why a unified and formalised approach is needed

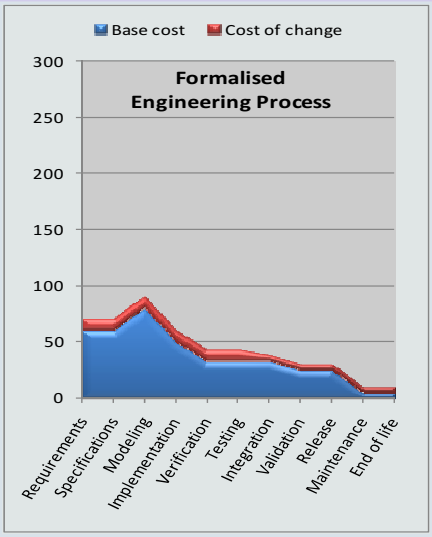
- **Many stakeholders:**
  - Political
  - Financial
  - Marketing
  - Engineering
  - Users
- **Many domains** => many domain-specific languages
  - Requires unified semantics (“ontology”)
  - Even in the technical domain!

- If no clear and common understanding is reached, there will be too many conflicting requirements, misunderstood requirements and hence the system will have hidden flaws.
- **Selecting the right system is the first step to develop it right.**

- **This work has to be done up front.**
  - This is also the cheapest phase for correcting mistakes
  - The further in the process, the more expensive
  - Risk of stopped projects
  - Risk of run-way costs

Cost benefits of Formalisation

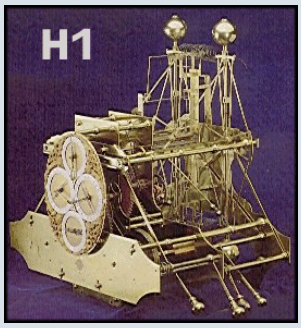




- Life cycle cost calculation
- Formalised approach leads also to more cost-efficiency and controlled reuse
  - Cleaner architectures leads to smaller size (especially for software)
    => up to 10x!

The biggest gain: the architecture

- Complexity is the enemy
- KISS: Keep It Simple but Smart
  - If a solution is complex, it means the problem is not well understood.
  - Simple solutions require a lot of thinking first.
- Technical notion of “elegance”: where engineering becomes an art.
- Examples:
  - NASA 1 million dollar space pen vs. 1 euro russian pencil
  - Harrison’s (1693-1776) time-keepers allowing navigation on sea:
    - “It has to be “**practical**” => small and simple
    - Besides saving many lives, it made the British Empire possible.



Concrete case: Pedal issues (1)

- Hybrid (=complex) vehicle with drive-by-wire throttle, brake, automatic gears
- Drivers claim unintended and uncontrollable acceleration
- Drivers claim that hitting the brakes doesn’t help
- Drivers find that braking can be lacking
- Many car manufacturers have similar issues
- But reproducing the hazard situation is (often) impossible
- Earlier Toyota Prius never had serious problems

- What is going on?
  - Mass hysteria? Toyota bashing to help GM?
  - Some high publicity cases were proven to be a hoax
  - Drive-by-wire issues? Likely some were real safety hazards
    - 1) Floor-mat wrong placement (Lexus) => mechanical issue
      2) Accelator pedal (Made in USA) mechanical issue (BIG recall #1)
      3) Prius ABS setting issue (recall #2) => recognised design issue
      4) Severe Cruise control issues (discussion is ongoing)

    - More info at: http://www.toyota.com/recall/

Concrete case: Pedal issues (2)

- Gas pedal issues:
  - Not reported with older Prius models (> 10 years!)
  - Cannot be reproduced
  - Pedal was accepted as specified by Toyota
  - Claimed cause ranges from:
    - Floor mat
    - Dirty and wet environment
    - Interference with cruise control
    - Interference with stability control
    - Software issue
    - Hardware (electronic) issue
    - EMI: external EM disturbance => logic lock-up
    - Mechanical: spring gets stuck
    - Mechanical: plastic used absorbs water
    - Driver doesn't know what to do
- => many opinions are mainly guesses, not based on facts.
- => nevertheless, there is clearly room for improvement
- => these are issues for ALL car manufacturers

What can automotive learn from it?

- Types of errors
  - Permanent: when things break
  - Intermittent: e.g. bad contact or external disturbance (e.g. RF, power)
  - Design errors: not all use- and fault-cases have been considered
    - **IS SIL3 (fail safe) acceptable for gas and brake systems?**
      - => NO! We need SIL4 (fault tolerance)
- Conclusions:
  - Black box will be needed in cars
    - A simple flash card would already help a lot
    - But is the car designed for it? (architecture) => no
  - Drive-by-wire SIL4 needed:
    - SIL3 assumes the fault can be detected
    - SIL3 (e.g. high idle) is not always fail-safe (e.g. highway@200 Km/hr)
    - But are all possible faults detected?
    - => triplication and voting architectures
    - => heterogeneous sensors
    - => n-version programming
    - Is the car designed for it? (architecture) => partially
    - What is the impact on the reliability?

What can automotive learn from it?

- Safety must really become TRUSTWORHTY
  - Safety: system operates as specified, always
    - Common mode failures mitigation by triplication
  - Security: integrity of system is assured
    - Freedom of external disturbances
  - Useability:
    - The driver is part of the system
    - What were his intentions as a driver?
- How can we keep the system simple while increasing ROI (more functionality, more active safety features, less fuel consumption, …)
- Publication of all safety issues can help the whole industry:
  - Cfr. Safety culture in aviation industry
- Car must become smart enough to deal with emergency situations rather than driver, who cannot be expected to act like a test pilot.

A simple set-up

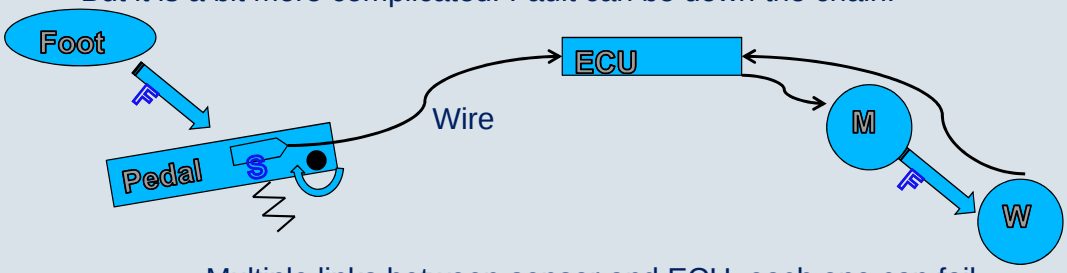
- Two pedals:
  - One for throttle, using one sensor
  - One for brake, using one sensor

4 states	Brake pressed	Brake not presses
Gas pressed	11 => brake => ?	10 => accelerate
Gas not pressed	01 => brake	00 => brake or idle?

- Cases 01 and 10 are clear but assume that:
  - The sensors work properly => see next slide
  - The driver had the intention
- Problem cases are 11 and 00
  - Priority brake > priority gas => brake when in doubt
  - What is the intention of the driver?

When can we trust the sensors?

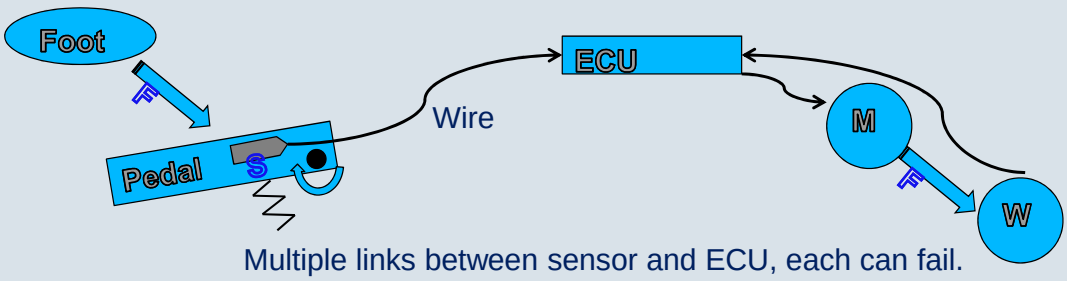
- One sensor (low cost cars):
  - If no fault => output reflects intention of driver
  - If fault => 3 cases: random output, stuck at 1, stuck at 0
- Two sensors (more expensive cars):
  - If different output => one of them is faulty
  - Maximum SIL is SIL3 (switch to fail safe mode, e.g. brake or engine at high idle)
  - Deadly at high speed
- Three sensors with triplication, voting, heterogeneous => SIL4 possible
  - Still very small risk of common mode failure
- But it is a bit more complicated. Fault can be down the chain!



Multiple links between sensor and ECU, each one can fail.

When can we trust safety features?

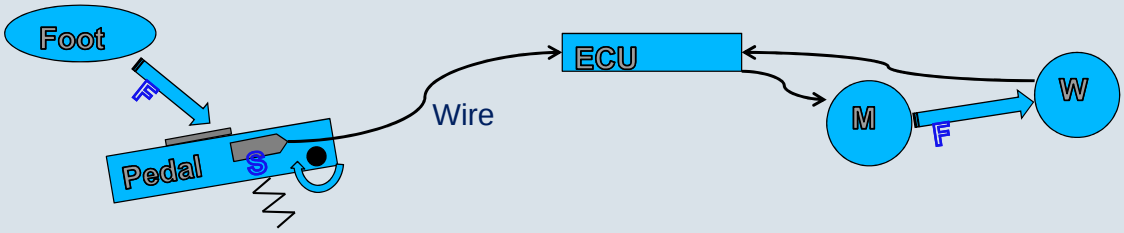
16 states	B=1 S=OK	B=1 S=Not OK	B=0 S=OK	B=0 S=Not OK
G=1 S=OK	1111	1011	0111	0011
G=1 S=Not OK	1110	1010	0110	0010
G=0 S=OK	1101	1001	0101	0001
G=0 S=Not OK	1100	1000	0100	0000



Multiple links between sensor and ECU, each can fail.

Was the pedal stuck or did the driver push full force?

- => Sensor on pedal to detect presence of foot
- But: state space now becomes 32
- If we take into account that sensor can fail as well: 64
- Introduces new limit case:
  - Driver puts foot but puts no pressure on pedal
- Issues:
  - Sensor should “sense” intention of operator
  - Operator should sense result of actions:
    - Force feedback => more complexity
  - What about the spring?



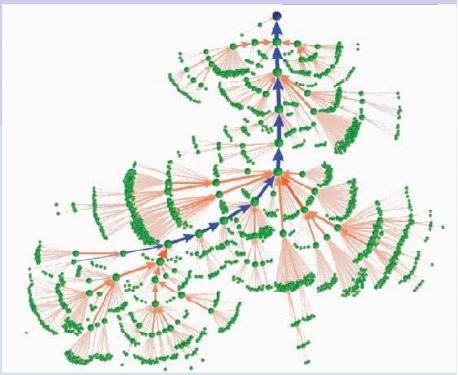
First conclusions for safety

- When there is a risk (“hazard”) possible:
  - Best try to isolate by applying orthogonal architecture
    - Else state space explodes even more
  - Assume that **everything** can malfunction of give a wrong output
  - Malfunction cause can be in three domains:
    - System itself
    - Operator
    - Environment
- Safety measures can make it worse:
  - => simple and clean architectures are best
  - Let the system keep a history log

The impact of electronics and software

- Why electronics and software?
  - Programmable => easy to change => also risk
  - Cheap, small
  - Allow more sophisticated functionality
    - Modern planes can't fly anymore without
    - Also key for lower energy and cleaner operation
- BUT:
  - Mechanical predecessors fail gracefully in the continuous domain
  - Electronic and software are clocked in the discrete domain
    - Fail within one clock cycle (typically 20 nanoseconds)
    - State space is huge (100 millions of states) because of data dependencies (1 integer = 2**32 states)
    - 10E-23 bit error rates => bit error becomes a certainty with time

The impact of complexity



- More functionality:
  - => complexity increases
  - => state space explodes exponentially
- More dynamic behaviour means more complexity:
  - Feedback loops needed to guarantee stability
  - Creates however difficult to find transition states
    - Clearly an issue with Toyota Prius: older models have no issues, latest issues seem to be related to interplay of more advanced features like dual engine, ABS + ESP, regenerative braking, rough road, etc.
    - Is an issue for all cars!
    - Might require use of active suspension to sense state of wheel vs. road
- But, where does the complexity come from?
  - Reusing old Harrison 1 type architectures ?
  - Layering of complexity?
- Conclusion: tackle complexity (and get safety) by cleaner architecture
  - Validate design and verify using formal approaches

Static vs. dynamic safety

- Current safety design is often based on rigorous static analysis and static implementation
  - Benefits: can be formally analysed
  - Drawbacks: fails catastrophically
- Real systems are more and more dynamic and are becoming difficult to analyse up-front
- Real systems can often tolerate a few intermittent or aperiodic misses (e.g. ABS system)

  => dynamic resource scheduling based on QoS

  =>Fault tolerance is a limit case
- Requires feedback control systems

Can cars become driver less?

- Can traffic flow improve through automatisation?
  - Or should we remove the bottlenecks first?
- Examples:
  - Driverless busses exist
- Big difference with trains:
  - much more decentralised, individual transport mode
- Will require adaption to:
  - Cars: multi-sensor fusion, comm links
  - Infrastructure: road beacons, comm links
  - Traffic separation: cars vs busses vs trucks
- Can only work if standardisation is applied
- Driver must remain in driver seat: he is the fail-safe mode
- Global view is transport and communication:
  - Do we still need trains or do we need cars that connect and act like trains?

Altreonic SIL 3/4 Controller project

•**Key characteristics :**

- High Reliability (SIL3) → Fault Tolerance (SIL4)

• **Target market :**

- Robotics, Automotive, Transport, Aerospace, Machine Control.

• **Technological competencies/partners sought:**

- Input from Use cases / Application scenarios
- Control systems design competence
- System Simulation

Thank You for your attention

"If it doesn't work, it must be art.
If it does, it was real engineering"