Critical Systems Labs
Strategic Insight for Safety

Safety Engineering for Software Intensive Systems

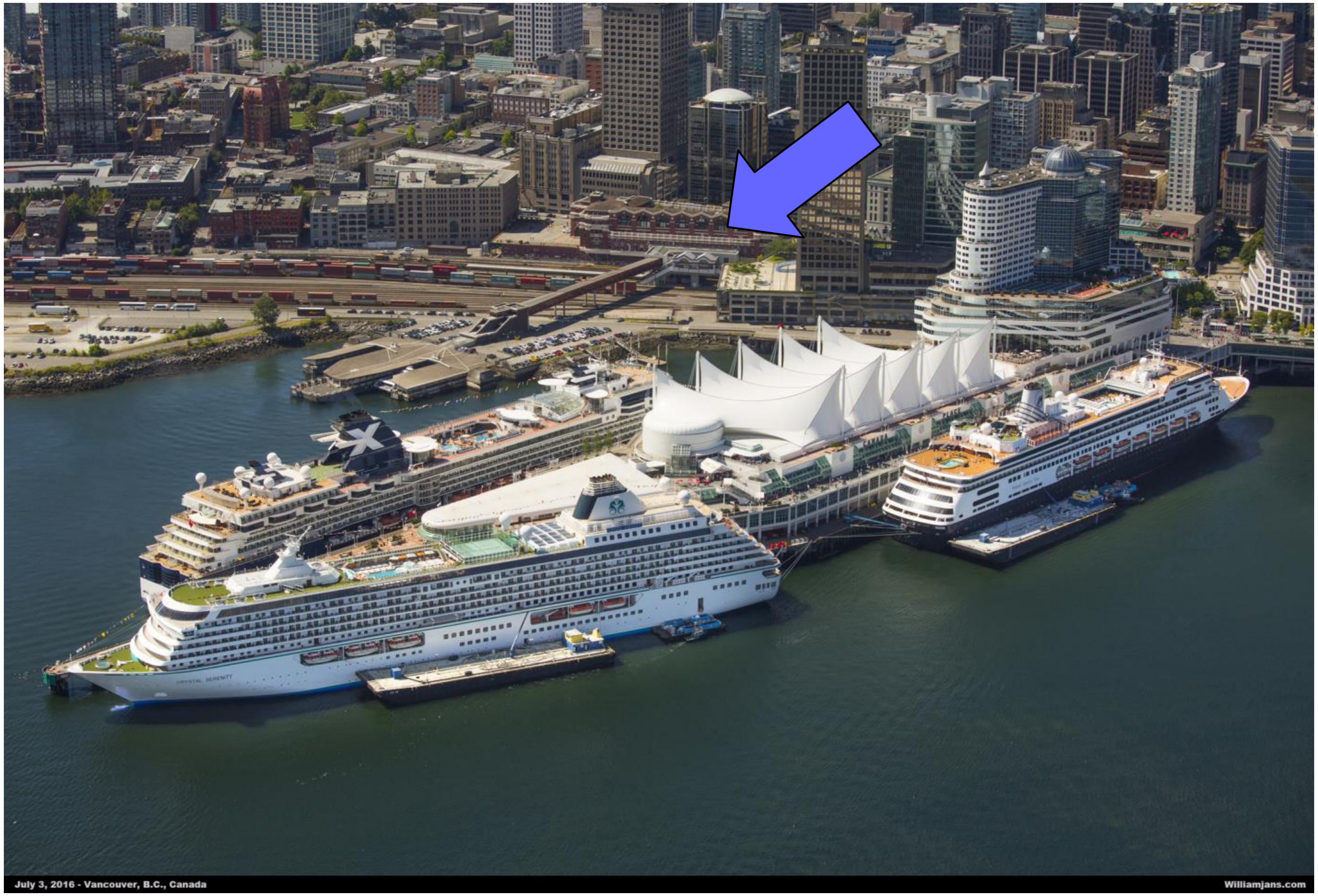
Guest Lecture

Simon Diemert, P.Eng., M.Sc.

Presenter - Simon Diemert

- Systems and Software Engineer - Critical Systems Labs Inc.
- Bachelors of Software Engineering – University of Victoria
- Masters of Computer Science – University of Victoria
- Professional Engineering – Engineers and Geoscientists of British Columbia

Critical Systems Labs



Systems engineering

Psychology

Statistics

Discrete math

Safety engineering *for* ***software intensive systems*** *is…*

A social activity

Software engineering

Safety engineering

Computer science

Lufthansa 2904 – September 1993

2 fatalities in Warsaw, Poland

High lateral winds made for adverse conditions.

Lufthansa 2904

- At least three different systems should have helped slow aircraft upon touch down

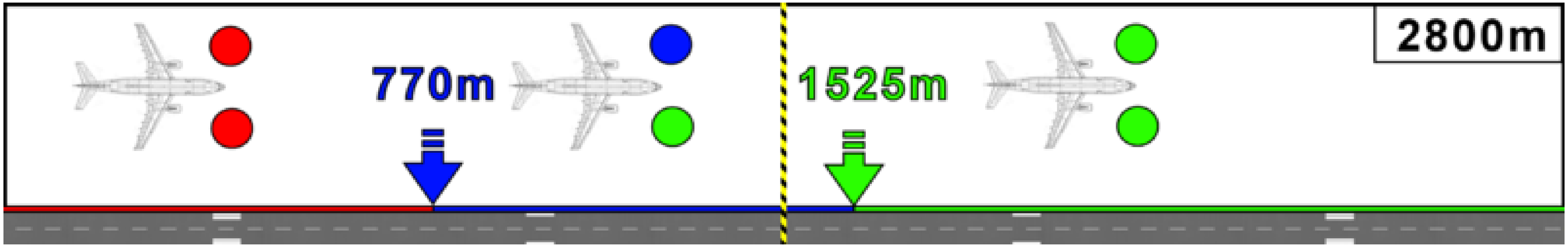
1. Reverse thrusters

2. Ground spoilers

3. Wheel brakes

- Left gear touched down 9 seconds after right gear
- Ground spoilers and reverse thrusters not automatically deployed until weight on both wheels

- Heavy rain result in hydroplaning
- Brakes not deployed until wheel rotation is greater than 72 knots
- “Computer did not actually know the aircraft had landed until it was already 125 meters beyond the half way point of runway 11. “



“To ensure that the thrust-reverse system and the spoilers are only activated in a landing situation, the software has to be sure the airplane is on the ground even if the systems are selected mid-air.” - Wikipedia

Every system on the aircraft functioned correctly, yet there was still an accident!

Reliability != Safety

?	Safe	Unsafe
Reliable		
Not Reliable		

	Safe	Unsafe
Reliable		LH 2904
Not Reliable		

Mars Polar Lander - 1999

- Intended to explore Mars south polar ice caps.

- Failed during landing phase of the mission due to a systems engineering error.

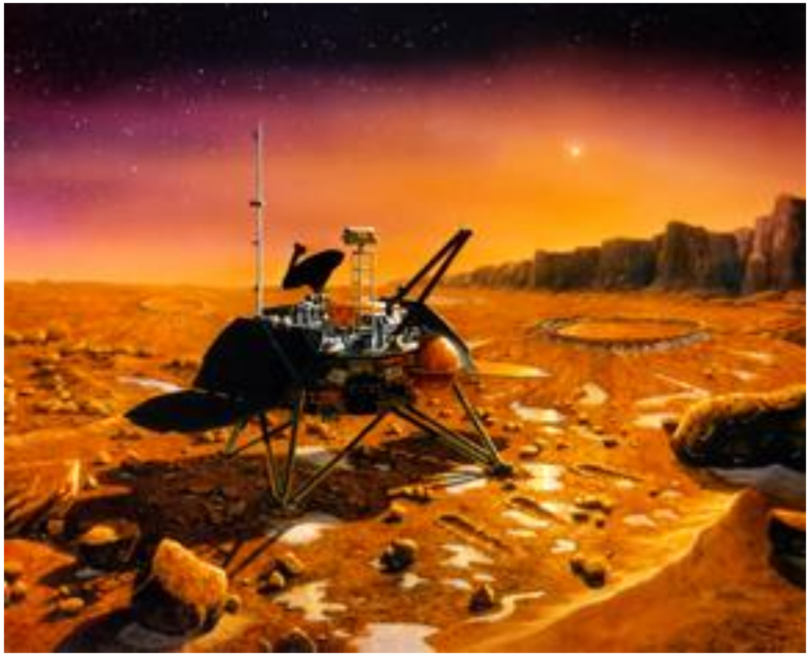
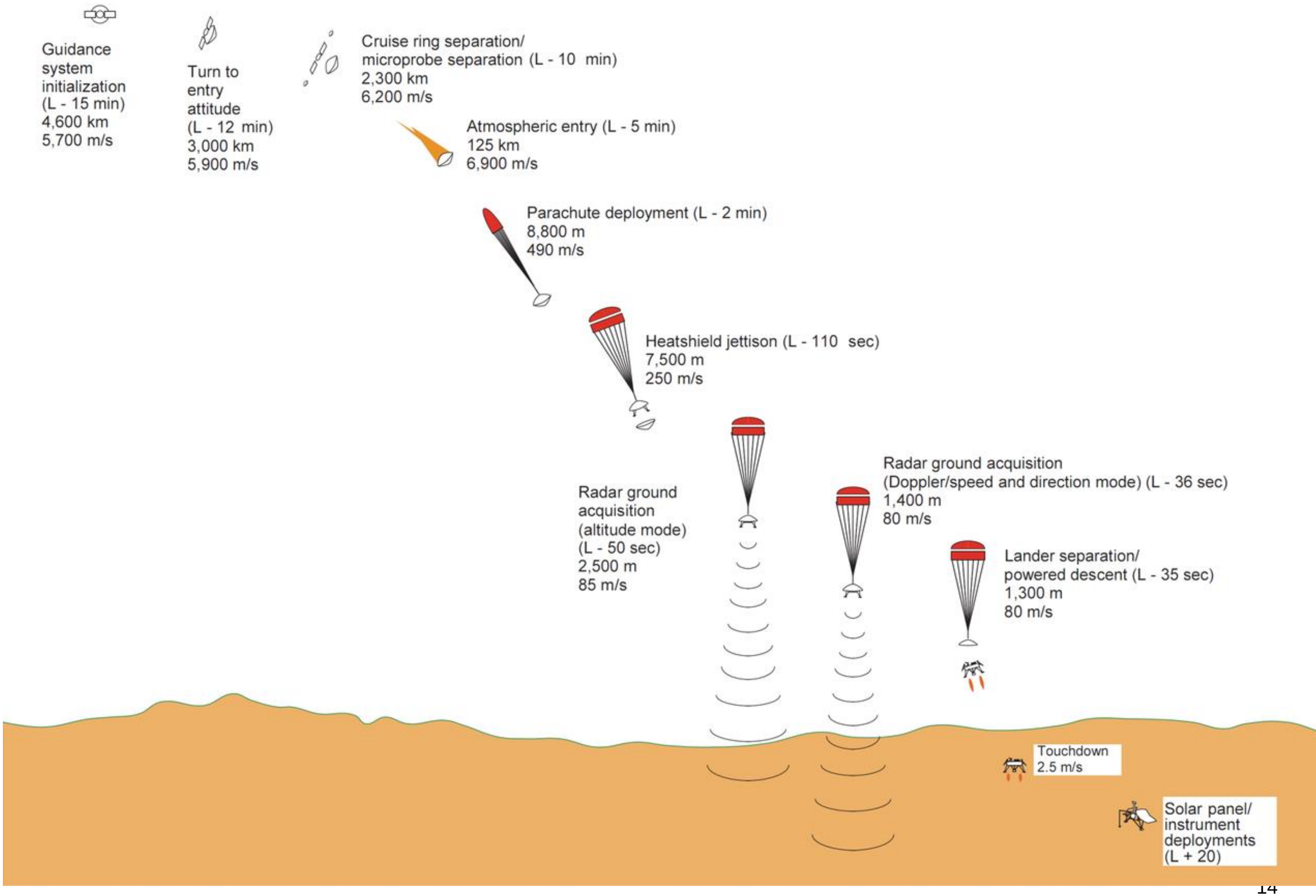
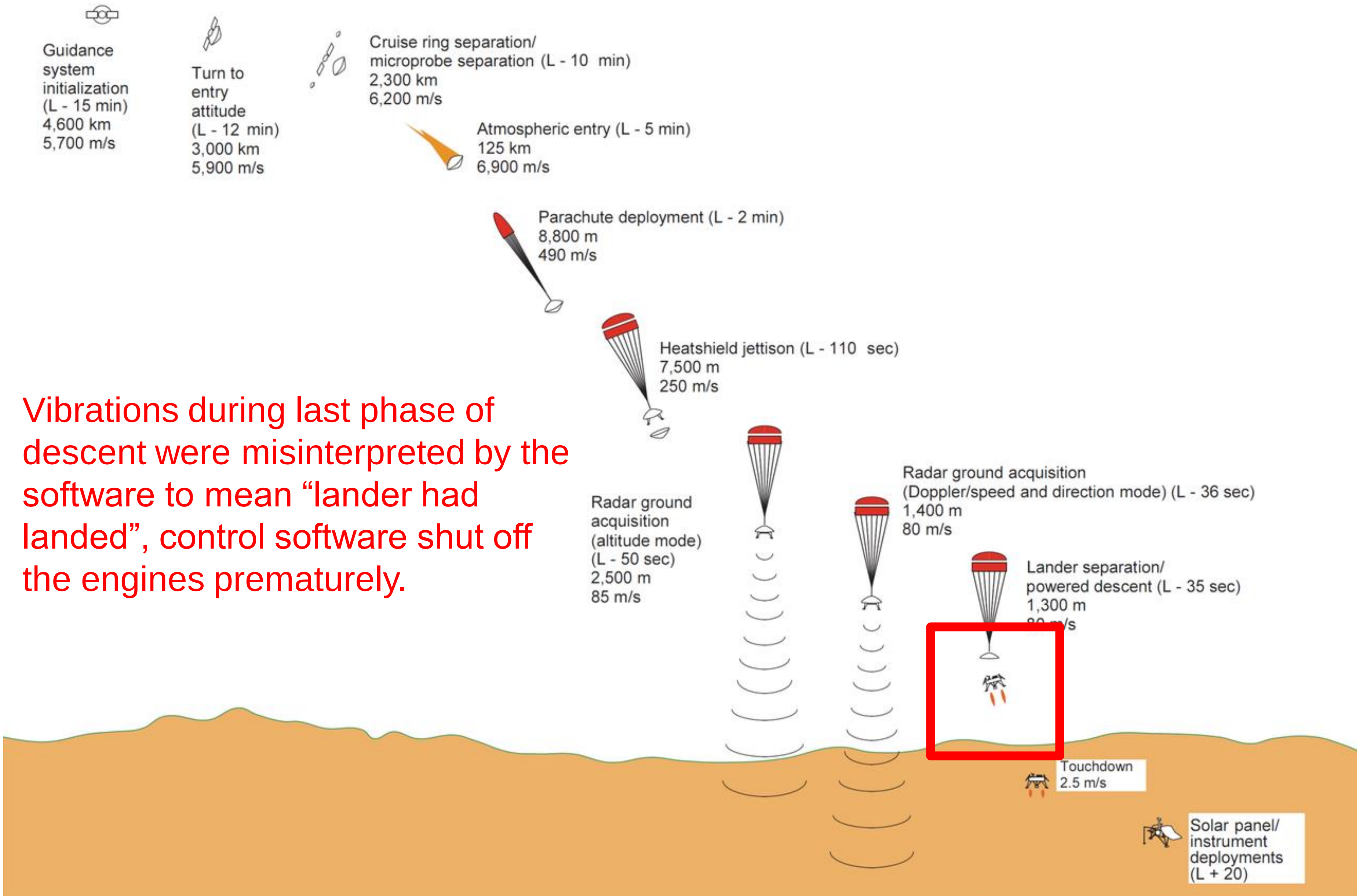


Photo: https://en.wikipedia.org/wiki/Mars_Polar_Lander



https://en.wikipedia.org/wiki/Mars_Polar_Lander

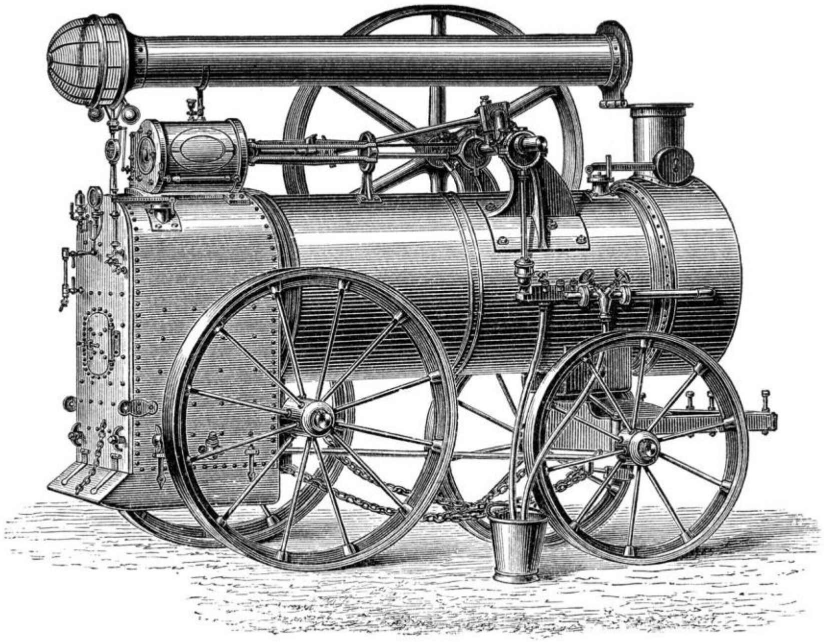


Vibrations during last phase of descent were misinterpreted by the software to mean “lander had landed”, control software shut off the engines prematurely.

What element of the system failed on the Mars Polar lander?

?	Safe	Unsafe
Reliable		LH 2904
Not Reliable		

	Safe	Unsafe
Reliable		LH 2904 Mars Lander
Not Reliable		



High-Pressure Steam Engines and Computer Software*

Nancy G. Leveson
Computer Science & Eng. Dept., FR-35
University of Washington
Seattle, WA 98195

Even though a scientific explanation may appear to be a model of rational order, we should not infer from that order that the genesis of the explanation was itself orderly. Science is only orderly after the fact; in process, and especially at the advancing edge of some field, it is chaotic and fiercely controversial.

— William Ruckelshaus [33, p.108]

The introduction of computers into the control of potentially dangerous devices has led to a growing

Every great invention is really either an aggregation of minor inventions, or the final step of a progression. It is not a creation but a growth — as truly so as that of the trees in the forest. Hence, the same invention is frequently brought out in several countries, and by several individuals, simultaneously. Frequently an important invention is made before the world is ready to receive it, and the unhappy inventor is taught, by his failure, that it is as unfortunate to be in advance of his age as to be behind it. Inventions only become successful when they are not only

Photo: https://sciencestruck.com/steam-engine-invention

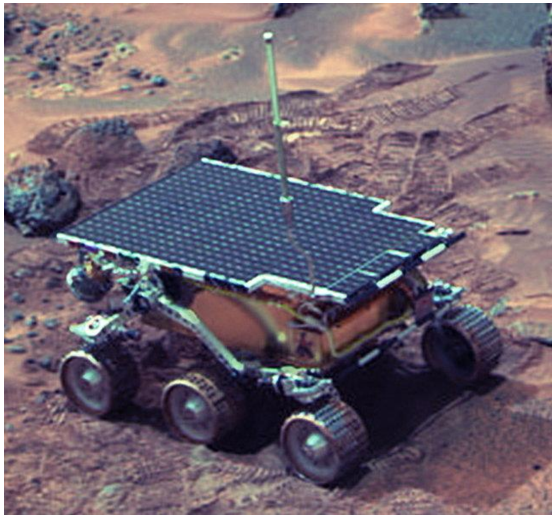
	Safe	Unsafe
Reliable		LH 2904 Mars Lander
Not Reliable		1800s steam engines

What Can We Learn from Steam Engines?

- The steam engine was at one time bleeding edge technology (~1800).
- At the time, there was little scientific and engineering knowledge about the behaviour of pressurized steam vessels.
- Steam engine operators we assumed to behave "rationally".
- Limited government enforced regulation of steam engines and boilers.

Mars Pathfinder, 1998

- The Pathfinder lander experienced a *priority inversion* in the software of the control computer.



- Resulted in repeated “resets” making the lander un-usable.
- This jeopardized the mission until teams were able to push a fix.
- The bug was known about prior to launch but deemed “low priority”

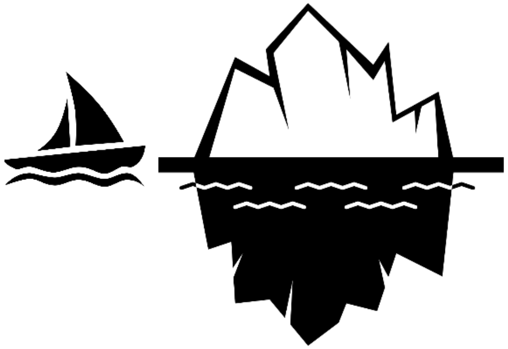
Photo: https://en.wikipedia.org/wiki/Mars_Pathfinder

?	Safe	Unsafe
Reliable		LH 2904 Mars Lander
Not Reliable		1800s steam engines

What constitutes "safe" or "unsafe" behaviour of the Pathfinder lander?

Safety

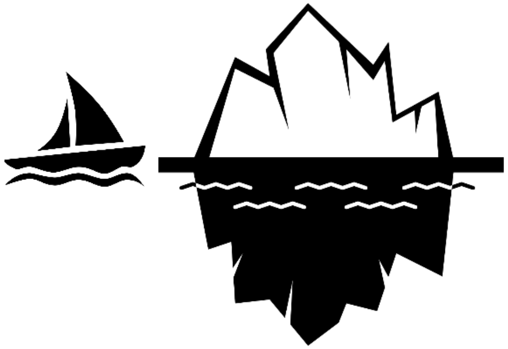
“freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property, or damage to environment.” – MIL 882E



	Safe	Unsafe
Reliable		LH 2904 Mars Lander
Not Reliable	Pathfinder(?)	1800s steam engines

Hazard

"A real or potential condition that could lead to an unplanned event or series of events (i.e. mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment"
– MIL 882E



Hazards v. Accidents

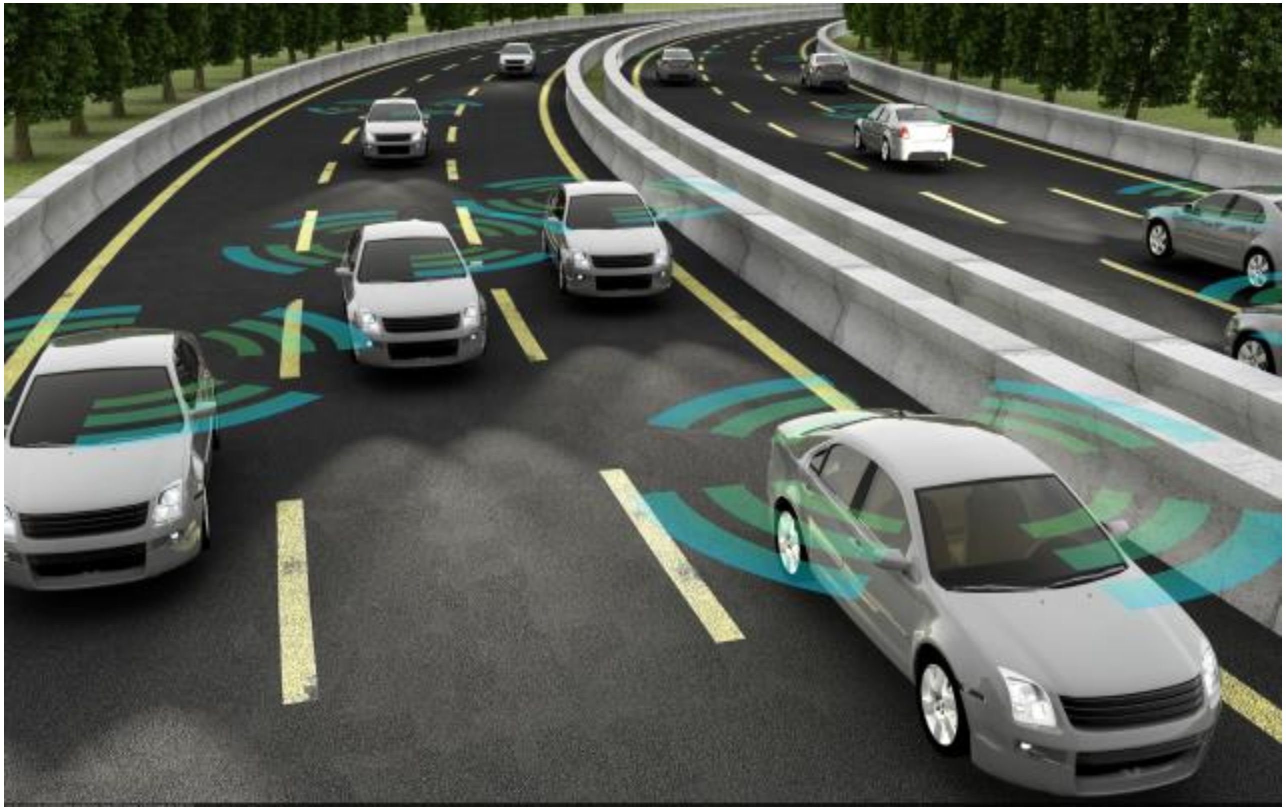
Suppose we have a laser system that is capable of emitting high-power light.

What **harm** could arise from this system?

What is a possible **hazard** for this system?

What is a **fail safe state** for this system?

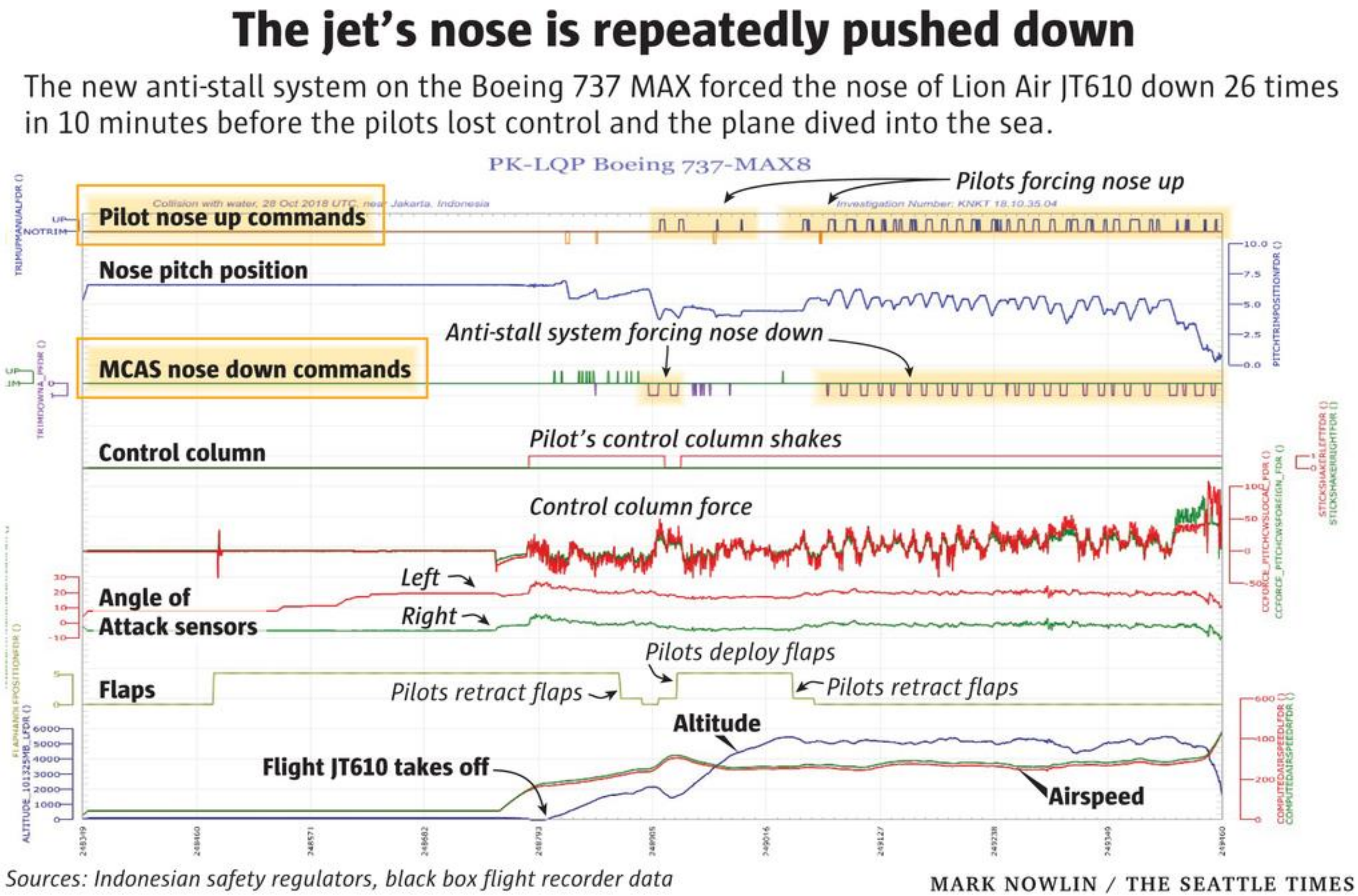
	Safe	Unsafe
Reliable		LH 2904 Mars Lander
Not Reliable	Pathfinder(?) Laser System	1800s steam engines



What is the "fail safe state" of a self-driving car brake control system?

	Safe	Unsafe
Reliable	Car Braking System	LH 2904 Mars Lander
Not Reliable	Pathfinder(?) Laser System	1800s steam engines

Lion Air 610 737 MAX, October 2018



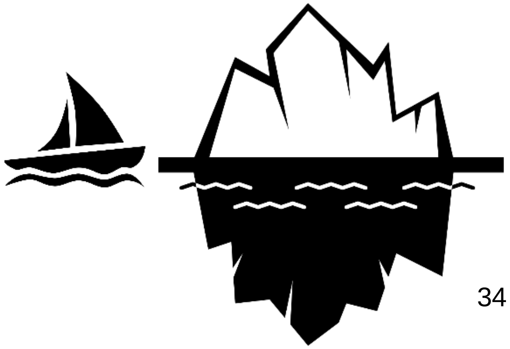
Critical Systems Labs
Strategic Insight for Safety

737-MAX MCAS?

	Safe	Unsafe
Reliable	Car Braking System	LH 2904 Mars Lander
Not Reliable	Pathfinder(?) Laser System	1800s steam engines

Watch out…

- Hazards **are** conditions **not** objects (nouns).
- Hazards are **not** casual factors.
- Hazards are **not** failures.
- Hazards **are** defined at the system boundary with its environment.



Risk

Classically **risk** is defined as:

$$\textit{Probability} \times \textit{Severity}$$

However, this does not work very well in software-intensive systems…. Why?

Risk

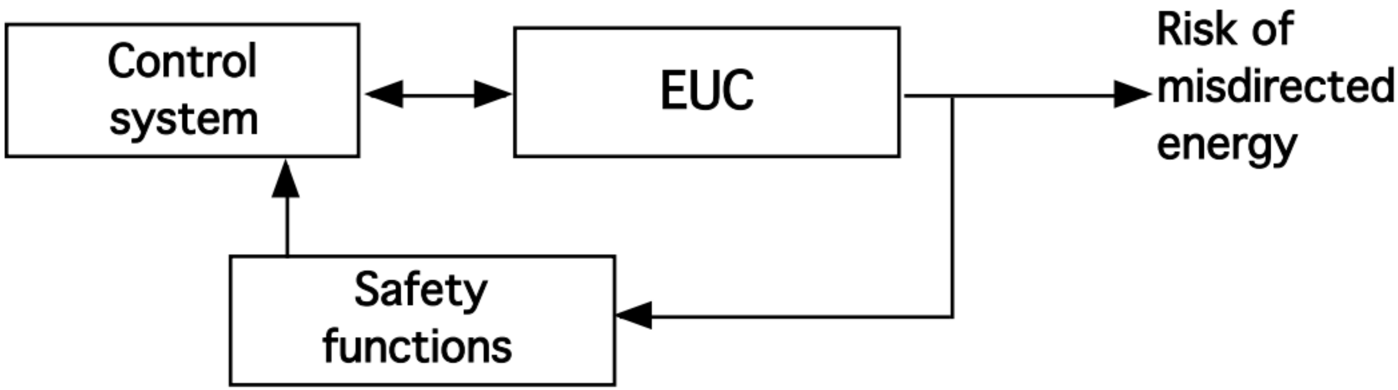
- How do you determine the probability of a hazardous event occurring?
- How do you quantify harm?
- Once you determine the level of risk, what do you do with it?

IEC 61508 – Numerical Risk

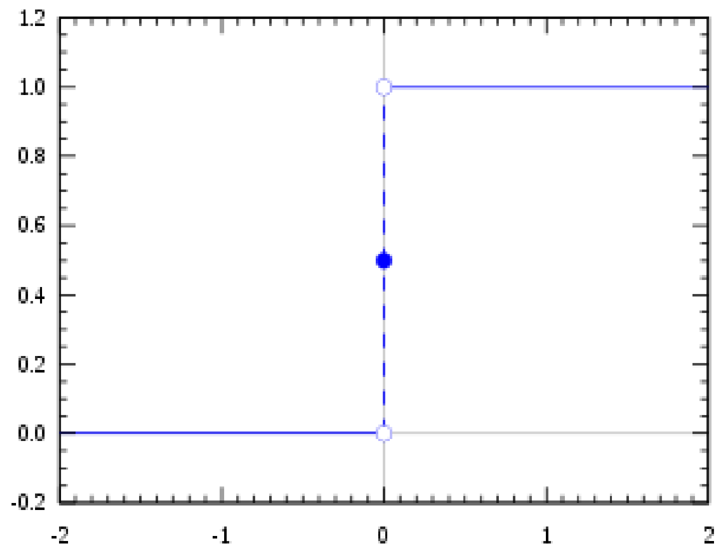
Table 5: Safety integrity levels

Safety Integrity Level	Low Demand Mode of Operation (Pr. of failure to perform its safety functions on demand)	Continuous/High-demand Mode of Operation (Pr. of dangerous failure per hour)
4	>= 10^{-5} to 10^{-4}	>= 10^{-9} to 10^{-8}
3	>= 10^{-4} to 10^{-3}	>= 10^{-8} to 10^{-7}
2	>= 10^{-3} to 10^{-2}	>= 10^{-7} to 10^{-6}
1	>= 10^{-2} to 10^{-1}	>= 10^{-6} to 10^{-5}

IEC 61508 – “Bolt on Safety”



Think of software you have written, can you determine a "dangerous failure rate" for that software?



What to do With Risk? (ISO 26262)

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++ ✓	++✓	++✓	++✓
1b	Use of language subsets	++✓	++✓	++✓	++✓
1c	Enforcement of strong typing	++✓	++✓	++✓	++✓
1d	Use of defensive implementation techniques	O	+✓	++✓	++✓
1e	Use of established design principles	+✓	+✓	+✓	++✓
1f	Use of unambiguous graphical representation	+✓	++✓	++✓	++✓
1g	Use of style guides	+✓	++✓	++✓	++✓
1h	Use of naming conventions	++✓	++✓	++✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Potential for efficiency gains through the use of test tools

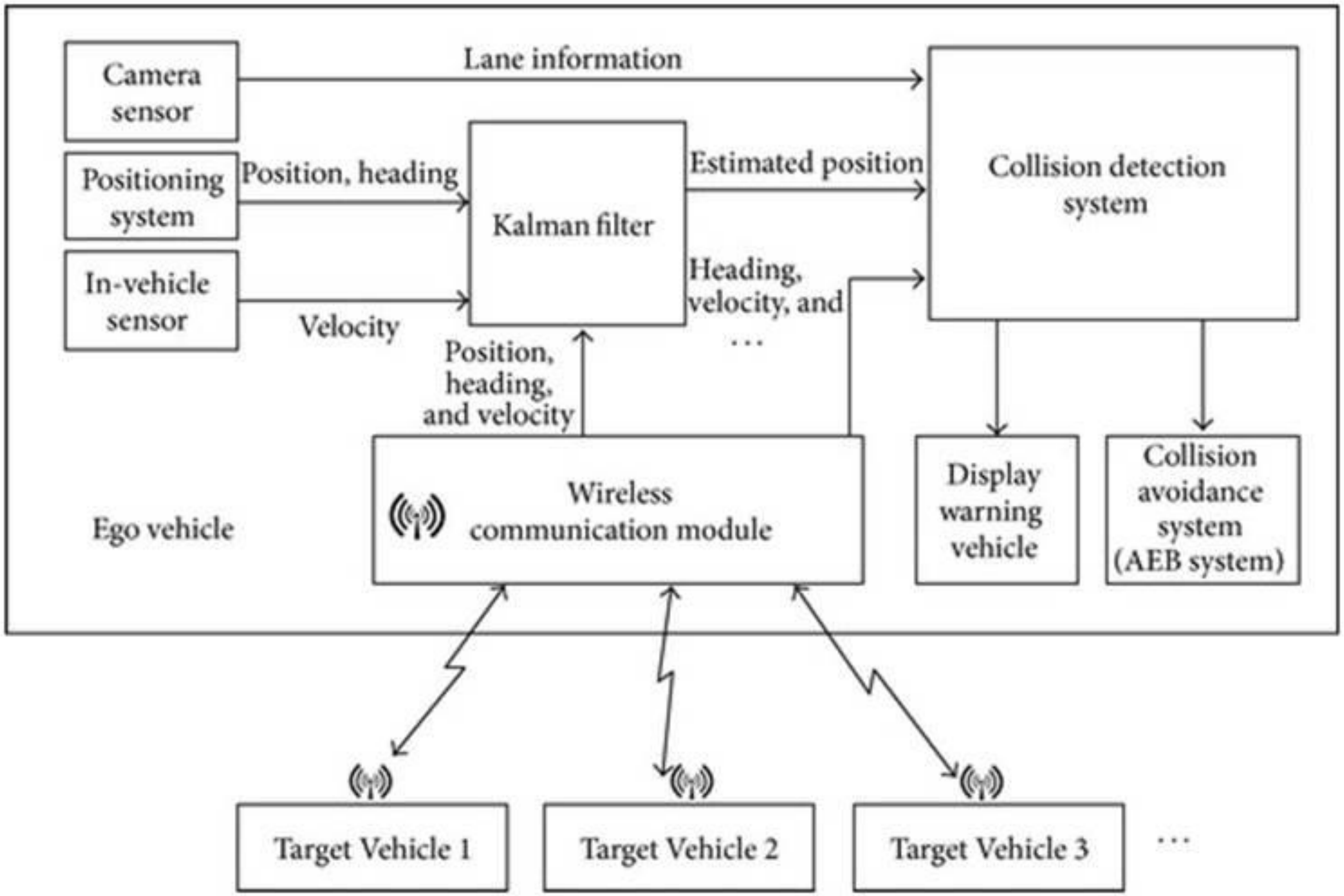
Critical Systems Labs
Strategic Insight for Safety

So what can do to make a “safe” system?

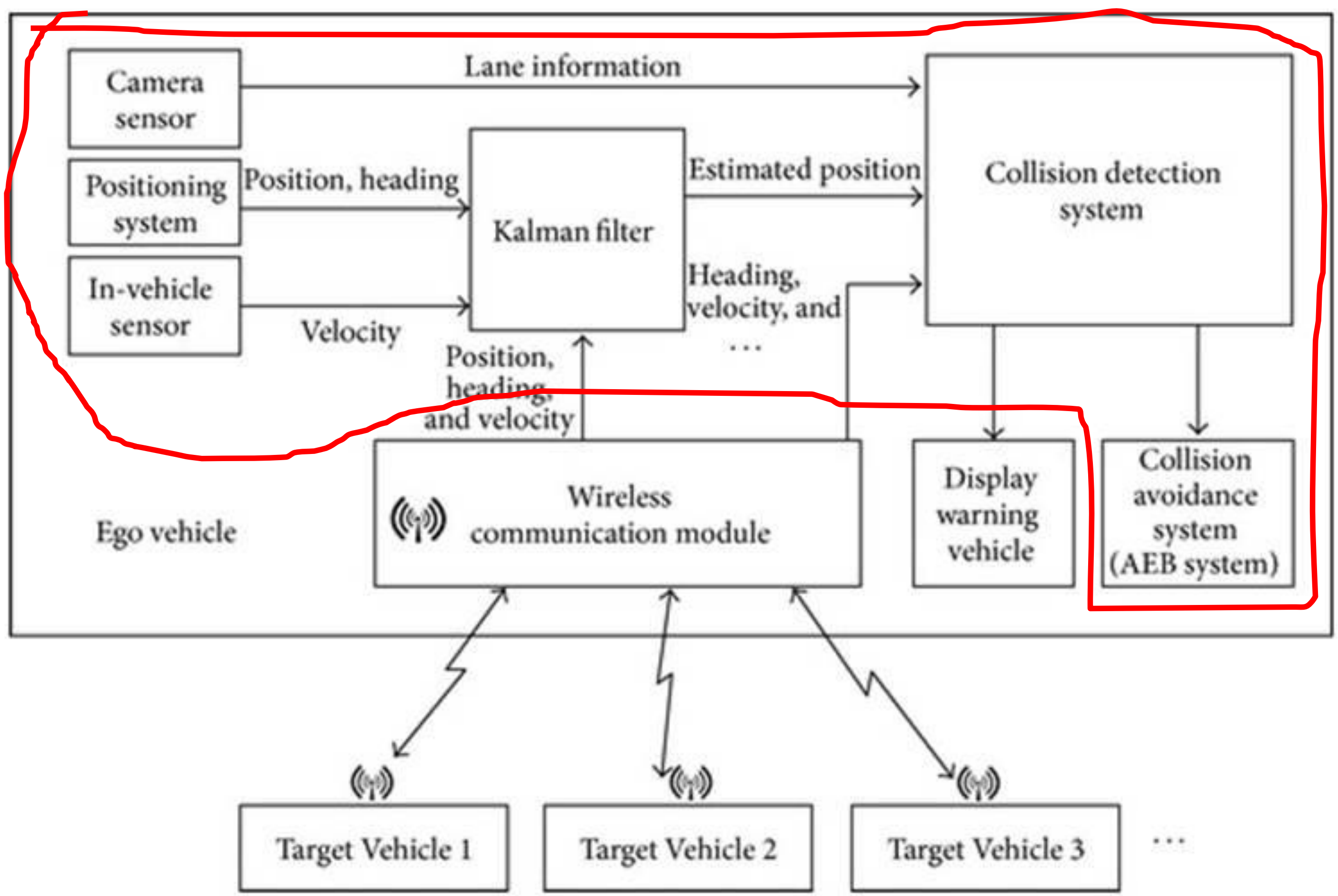
Steps to Safety …

1. Identify the system scope
2. Identify hazards
3. Identify level of criticality (risk)
4. **Identify casual factors**
5. **Design to mitigate causal factors.**
6. **Verify design/requirements.**
7. **Make an Argument**
8. Repeat…

Case Study - Automated Emergency Braking (AEB)



Step 1 – Identify System Scope



Step 2 – Identify Hazards

1. Hazard statement:
   - *“An occurrence of this hazard exists when…”*
2. Identify possible harm
   - What bad things can happen because of this hazard?
3. Make a simple hazard scenario
   - What reasonably foreseeable sequence of events could lead to an occurrence of this hazard?

Step 3 – Assess Level of Criticality

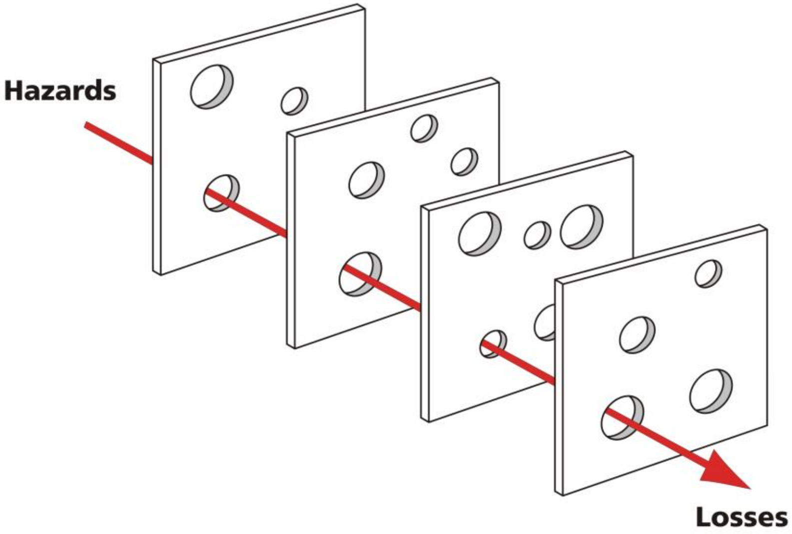
Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

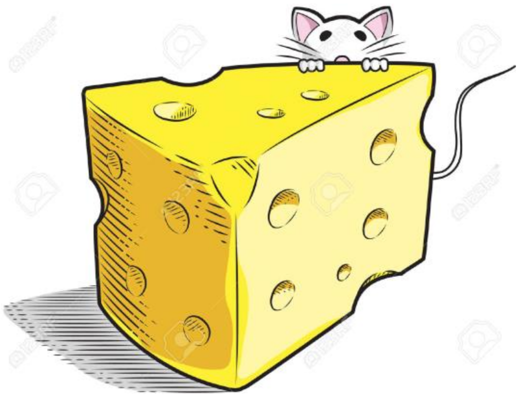
Severity – How bad is the harm (minor, major, death)?

Exposure – How likely are you in a situation where it matters?

Controllability – How much control does the driver have over the situation?

Step 4 - Identifying Casual Factors I

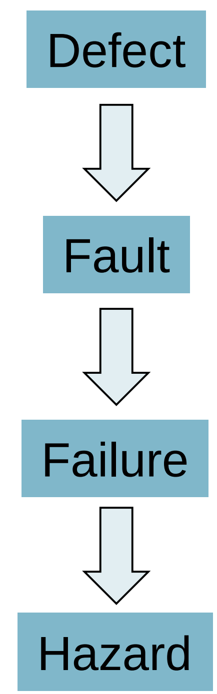




A number of techniques exist to detect casual factors…

Step 4 - Identifying Casual Factors II

- **Casual Factors** are conditions that contribute to the occurrence of a hazard:
  - Mechanical failure
  - Electrical fault
  - Software defect
  - Memory corruption
  - Scheduling delay



Step 4 - Identifying Casual Factors III

Systematic Faults/Failures

- Related to errors in the design of the system.
  - Beam insufficient for anticipated load.
  - Incorrect choice of resistor.
  - Bad logic/circuit design.
  - Incorrect functional requirements.

Random Faults/Failures

- Related to randomly occurring events.
  - Wear/tear on materials.
  - Environmental conditions (e.g., radiation).
  - Extremely rare external events (100 year storms).

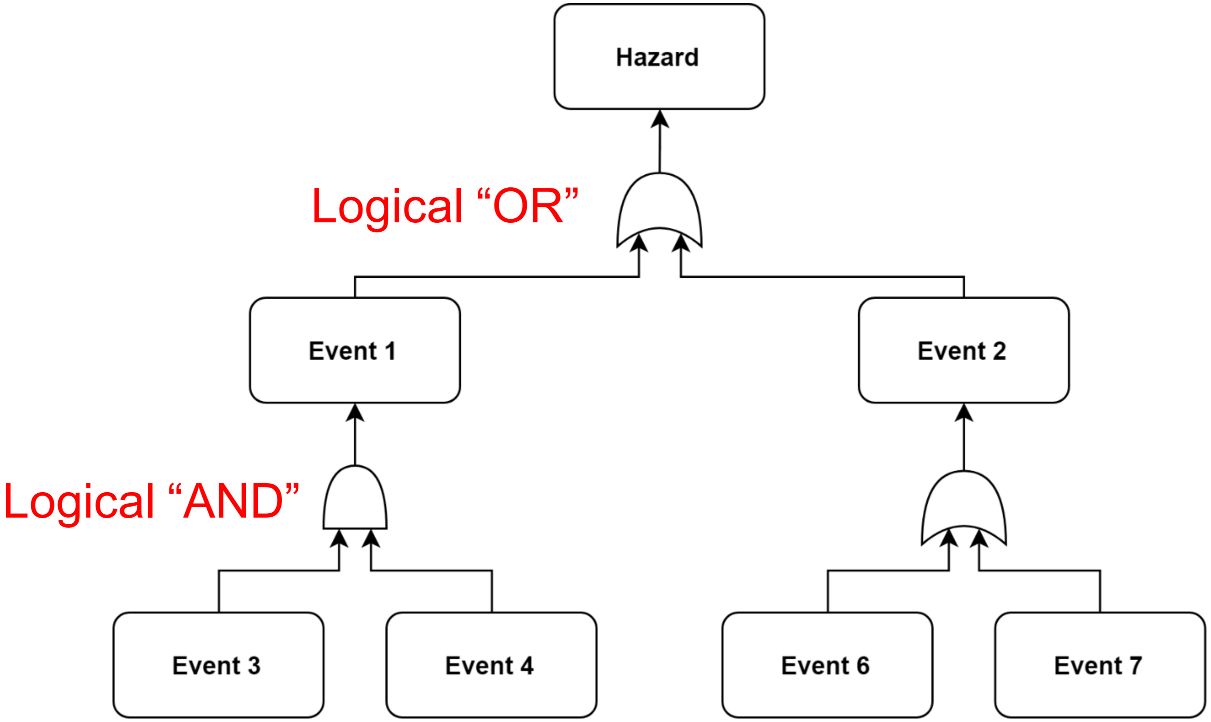
Where do concepts from software engineering fit?

- Software bugs/defect?
- Scheduling deadline missed?
- Statistical/probabilistic algorithms?
- Faults/failures in software dependencies?

Step 4 – Identifying Causal Factors – FTA I

- “top-down” search for potential hazard causes
- Logic Gates (AND, OR) are used for branches in the tree
- Sometimes annotated with probabilities in an effort to quantify likelihood of a hazard.

Step 4 – Identifying Causal Factors – FTA II



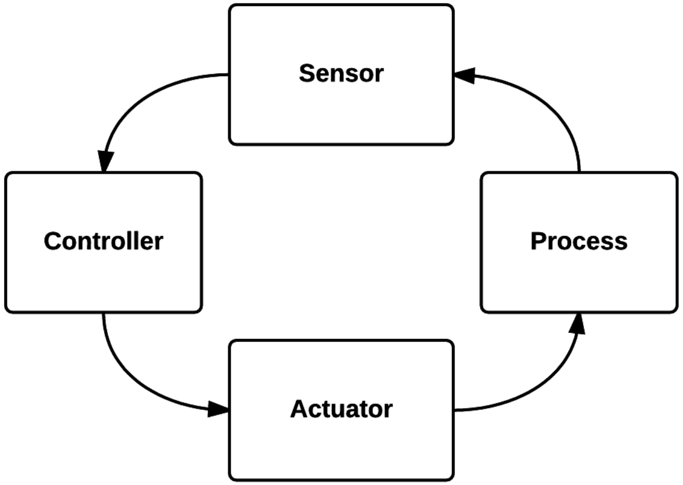
Step 4 – Identifying Causal Factors – FMEA I

- “bottom-up” for potential hazard causes.

Element	Failure Mode i.e., how does required behaviour fail ?	Effect i.e., what is the final state or output ?	Analysis i.e., is this hazardous ?

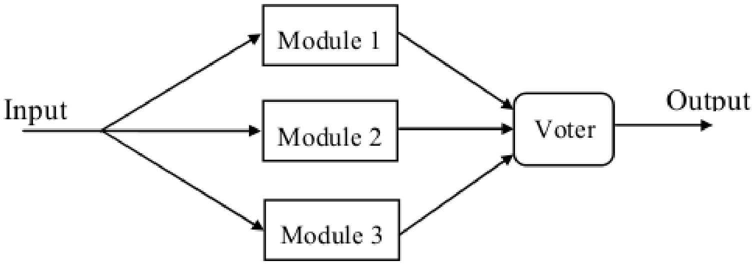
Step 4 – Identifying Causal Factors – STPA I

- **S**ystems-**T**heoretic **A**ccident **P**rocesses
- Model systems as series of "control loops".
- Controllers have *control actions* that are used to actuate processes.
- Systematically consider effects of control actions and whether they might lead to a hazard.



Step 5 – Eliminate Causal Factors – I

- Component redundancy:
  - Duplicate sensors
- Hardware redundancy:
  - Cold/warm/hot "spare" component.
  - Parallel hardware
- Software Redundancy:
  - N-Version Programming
  - Checkpointing
- Fail-safe state(s)



Designing for Failure - Ariane 5

Ariane 5 is a ESA rocket that experienced a catastrophic failure in 1996.

The rocket had two redundant flight control computers.

During take-off, the primary computer experienced an “overflow” issue due to a poorly written type cast.

Backup computer was switched to and experienced the same issue…

Aside…

What modern aircraft experienced a similar ”overflow” issue?

https://www.theguardian.com/business/2015/may/01/us-aviation-authority-boeing-787-dreamliner-bug-could-cause-loss-of-control

Aside Aside…. - Therac-25

Therac 25 was a radiation therapy machine developed in 1980’s.

Older versions (< 25) relied upon a “hardware interlock” safety mechanism to control the beam strength.

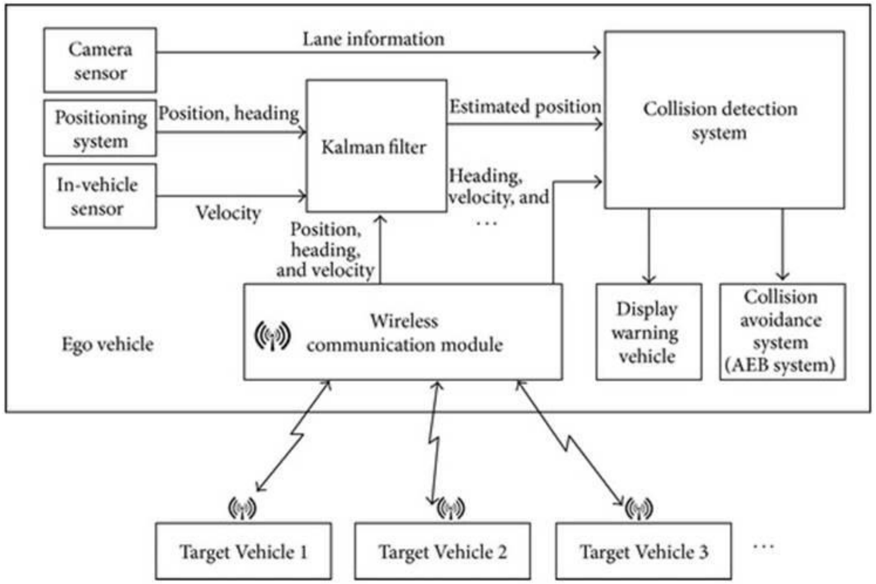
Version 25 replaced hardware interlocks with software control.

Software failed due to a race condition and integer overflow.

At least 6 incidents (serious burns or death) were reported (though probably more occurred).

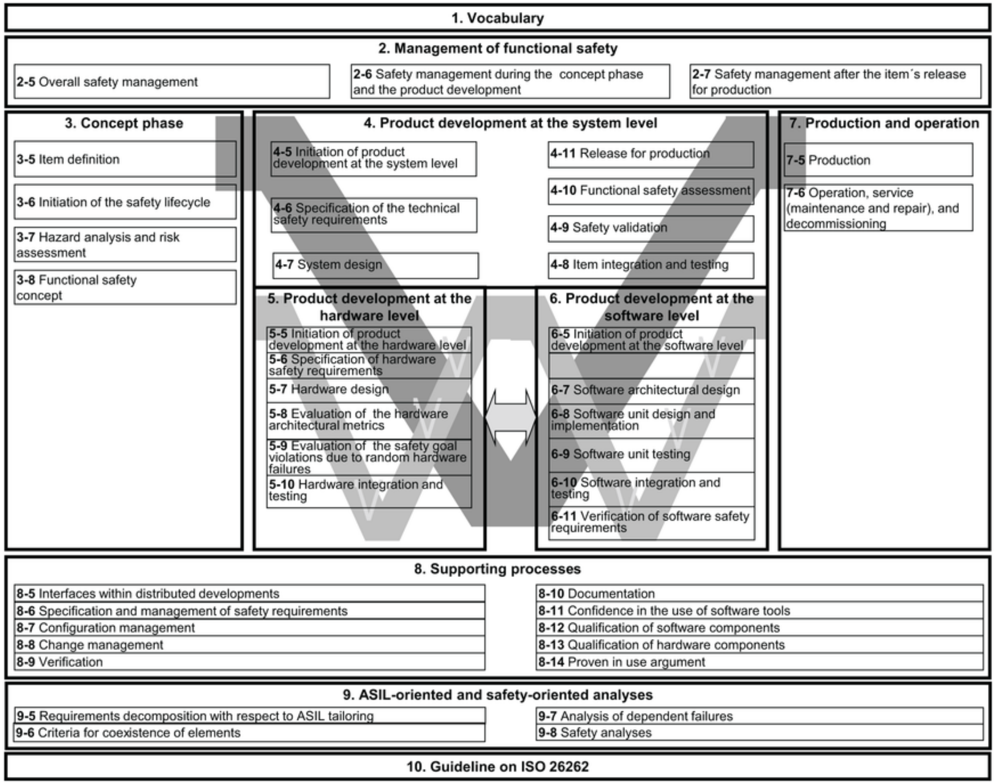
Step 5 – Eliminate Causal Factors – II

- What is a good fail-safe state for the AEB system?
  - Is this acceptable in all operational conditions?
  - How do you notify the user?
  - Can the AEB ever transition back to an operational state?
  - Are there degraded operational modes?



Step 5 – Eliminate Causal Factors – III

- Capture safety related designs as “safety requirements”.
- After requirements definition, the engineering team is focused on implementing the requirements.



Step 6 – Verify Requirements – I

- Testing (unit, integration, HIL, etc.)
  - Measure Coverage (is that enough?)
- Analytical Methods
  - WCET
- Statistical Methods
  - What is the probability of a defect existing in the software?
  - What is the probability of the scheduler meeting the deadline?
- Formal Methods
  - Model checking, Hoare Logic, Theorem Proving

Step 6 – Verify Requirements – Testing

Table 13 — Methods for software integration testing

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	++	++
1d	Resource usage test[cd]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[e]	+	+	++	++

[a] The software requirements at the architectural level are the basis for this requirements-based test.

[b] This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting software or hardware components).

[c] To ensure the fulfilment of requirements influenced by the hardware architectural design with sufficient tolerance, properties such as average and maximum processor performance, minimum or maximum execution times, storage usage (e.g. RAM for stack and heap, ROM for program and data) and the bandwidth of communication links (e.g. data buses) have to be determined.

[d] Some aspects of the resource usage test can only be evaluated properly when the software integration tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

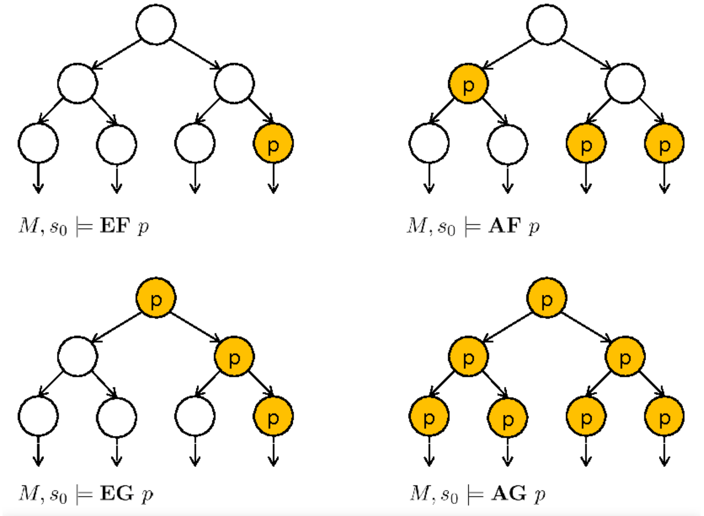
[e] This method requires a model that can simulate the functionality of the software components. Here, the model and code are stimulated in the same way and results compared with each other.

Table 12 — Structural coverage metrics at the software unit level

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

Step 6 – Verify Requirements – Formal Methods I

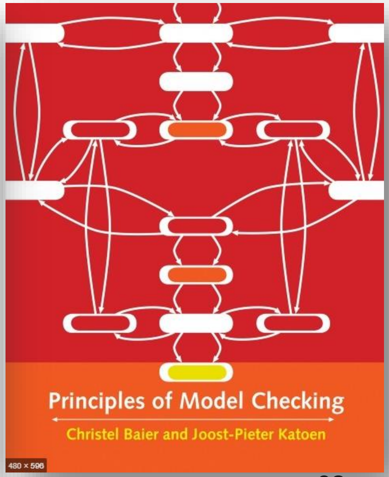
- Model checking systematically explores the state space of a “linear transition system”.
- Model system as a state machine.
- Express temporal properties.
- Widely used in FM community.
- Suffers from state space explosion problem.




contributed articles

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

How Amazon Web Services Uses Formal Methods



Step 6 – Verify Requirements – Formal Methods II

- Use “program verification” to prove that snippets of code are “bug free”.
- Covers all possible executions of code.
- Requires a formal semantics for the programming language.
- Uses Hoare Logic as underlying theory.
- VERY EXPENSIVE..

```
function Make_State( NS, SN, EW, WE : Light_State) return Traffic_State
  with
    Pre => True,
    Post => (
                Make_State'Result.Light_NS = NS and
                Make_State'Result.Light_SN = SN and
                Make_State'Result.Light_EW = EW and
                Make_State'Result.Light_WE = WE
            );

function NS_Green( State : Traffic_State ) return Traffic_State
  with
    Pre => True,
    Post => (
                NS_Green'Result.Light_NS = GREEN and
                NS_Green'Result.Light_SN = GREEN and
                NS_Green'Result.Light_EW = State.Light_EW and
                NS_Green'Result.Light_WE = State.Light_WE
            );


function NS_Red( State : Traffic_State ) return Traffic_State
  with
    Pre => True,
    Post => (
                NS_Red'Result
```

SPARK 2014
Expanding the boundaries of safe and secure programming.

$$\frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ C_1; C_2\ (\!|\psi|\!)}\ \text{Composition}$$

$$\frac{}{(\!|\psi[E/x]|\!)\ x = E\ (\!|\psi|\!)}\ \text{Assignment}$$

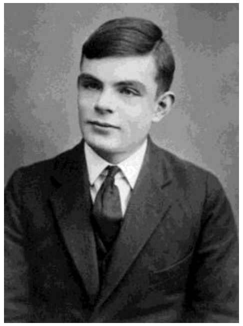
$$\frac{(\!|\phi \wedge B|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi \wedge \neg B|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-statement}$$

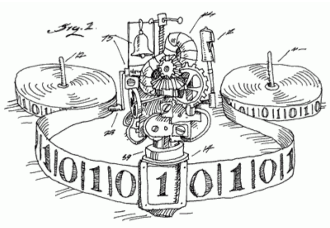
$$\frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\psi \wedge \neg B|\!)}\ \text{Partial-while}$$

$$\frac{\vdash_{\mathrm{AR}} \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{\mathrm{AR}} \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\ \text{Implied}$$

Step 6 – Verify Requirements – Formal Methods III

- Formal methods usually tools rely on Satisfiability Solvers (SAT Solvers) to prove correctness.
- SAT is an NP-Complete (computationally hard) problem.
- In general, we cannot prove properties of all programs (Entscheidungsproblem).
- Yet… we can still make proofs for many programs of interest.





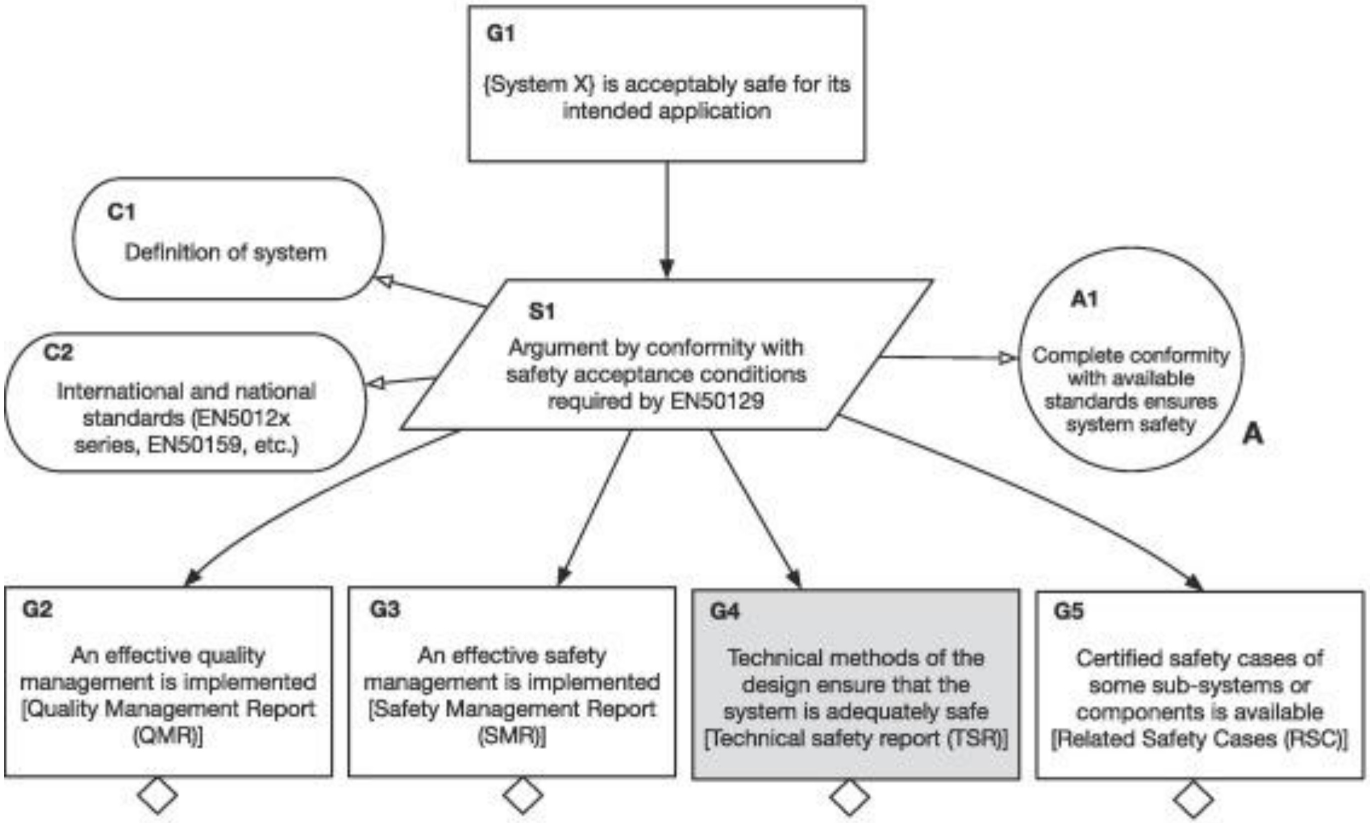
Step 6 – Verify Requirements – Statistical Methods

- What problems can we apply statistical methods to?
  - Scheduling
  - Stochastic inputs/signals
  - Random algorithms
  - Machine learning
- How do we translate statistical results into safety guarantees?

The AEB system shall identify vehicle's in front of the ego vehicle within 10 meters.

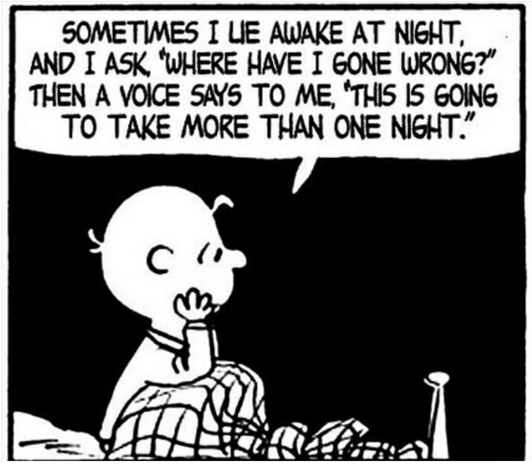
Step 7 – Make an Argument

Step 7 – Make an Argument



The Big Questions…

- How do I know when I have done enough testing?
- How do I know I haven't missed anything in my analysis?
- How do I know when to stop analyzing?
- …



What have we missed in our discussion thus far?

Humans!

Two Roles for Humans

- Engineers, Designers, etc.
  - Requires minimum technical skill.
  - Requires experience.
  - Requires judgement.
- Users
  - Trained users
  - Untrained users
  - Technicians/maintenance.

Journal of the American Medical Informatics Association Volume 12 Number 4 Jul / Aug 2005 377

Case Report ■

Comprehensive Analysis of a Medication Dosing Error Related to CPOE

JAN HORSKY, MA, MPHIL, GILAD J. KUPERMAN, MD, PHD, VIMLA L. PATEL, PHD, DSC

Abstract This case study of a serious medication error demonstrates the necessity of a comprehensive methodology for the analysis of failures in interaction between humans and information systems. The authors used a novel approach to analyze a dosing error related to computer-based ordering of potassium chloride (KCl). The method included a chronological reconstruction of events and their interdependencies from provider order entry usage logs, semistructured interviews with involved clinicians, and interface usability inspection of the ordering system. Information collected from all sources was compared and evaluated to understand how the error evolved and propagated through the system. In this case, the error was the product of faults in interaction among human and system agents that methods limited in scope to their distinct analytical domains would not identify. The authors characterized errors in several converging aspects of the drug ordering process: confusing on-screen laboratory results review, system usability difficulties, user training problems, and suboptimal clinical system safeguards that all contributed to a serious dosing error. The results of the authors' analysis were used to formulate specific recommendations for interface layout and functionality modifications, suggest new user alerts, propose changes to user training, and address error-prone steps of the KCl ordering process to reduce the risk of future medication dosing errors.

■ **J Am Med Inform Assoc.** 2005;12:377–382. DOI 10.1197/jamia.M1740.

Scenario

- A patient received a massive overdose of Potassium Chloride (KCl).
- This was delivered slowly over 42 hours through a series of human errors.
- Causal Factors:
  - Electronic medical record user interface poorly designed.
  - Tired medical professionals.
  - Incomplete communication between people.

“Experts estimate that as many as 98,000 people die in any given year from medical errors that occur in hospitals.” – To Err is Human, IOM 2000

98,000 / 365 = ~270

	Safe	U afe
Reliable		4 Lander
Not Relia		1800s steam engines

Where do we put socio-technical systems????

Questions?

Simon Diemert, M.Sc., P.Eng.

simon.diemert@cslabs.com