Software Safety and Security in a world of Systems

•••

Franco Gasperoni

AdaCore

Disclaimer

I am a simple observer

Other people @ AdaCore know automotive much better than I do

Security is VERY important, tomorrow’s talk is about security

Safety & security are tightly connected

The point of this talk

Engineering requires

- Intuition / creativity → Humans only
- Formalism (model of the world) → Tools & humans

Complexity of engineering safe & secure systems keeps ↗

if formalization ↗ help from machines ↗ (tools)

Where are possible near-term / long-tem wins ?

AdaCore

Automotive: the most important transportation industry today

100 M vehicles produced in 2016

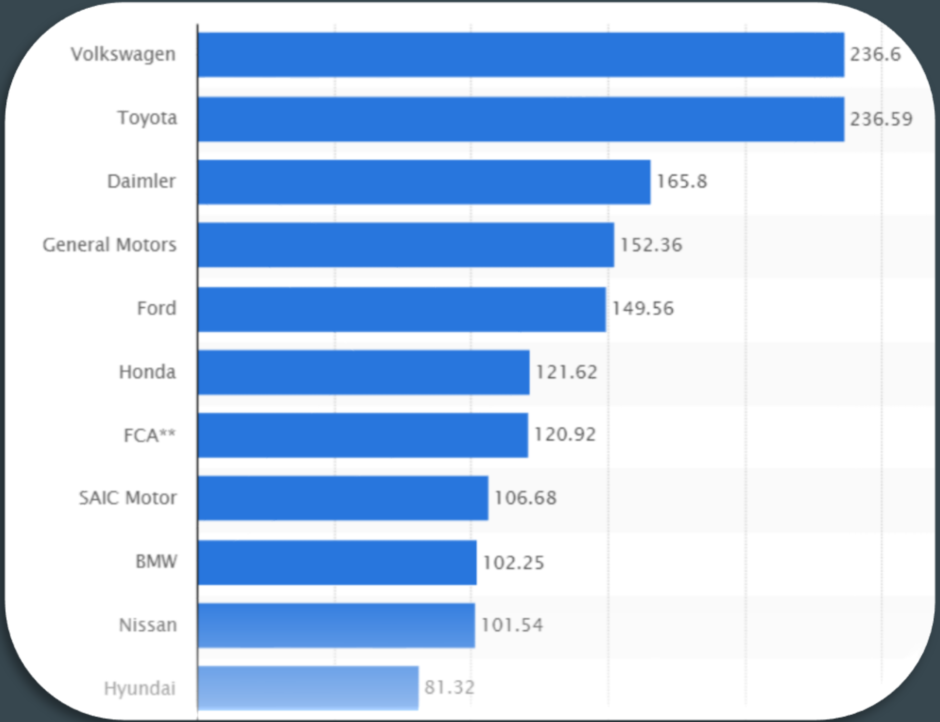
vs a TOTAL of 40,000 planes in the world now

Employees: 50M worldwide

10M in manufacturing

40M car dealers + car repair shops

Sales in 2015



AdaCore

Automotive trends …

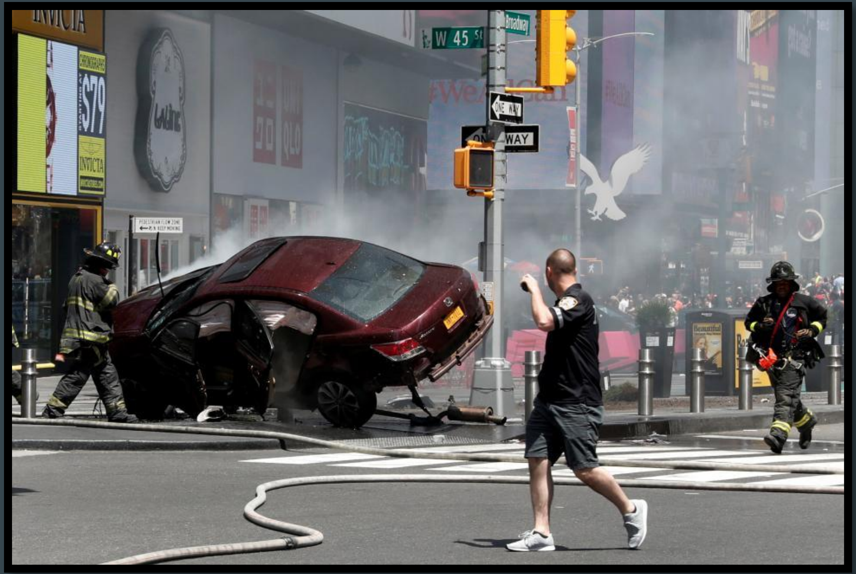
… shakeup coming ?

1. Safety & Convenience

What do you see?

… a weapon

Times Square 2017-May-19



Automotive Safety: road crashes

- 1.3 M/year die: 2+ deaths/hour
- 20-50 M/year injured or disabled
- 9th leading cause of death
- Leading cause of death among ages 15-29
- Cost of USD $518 billion globally, 1 to 2% of their annual GDP
- Unless action is taken, 5th leading cause of death by 2030

What if humans were not allowed to drive?

AdaCore

1. Safety & convenience

Level 0: Humans do all the driving

Level 1: one task automated (cruise control)

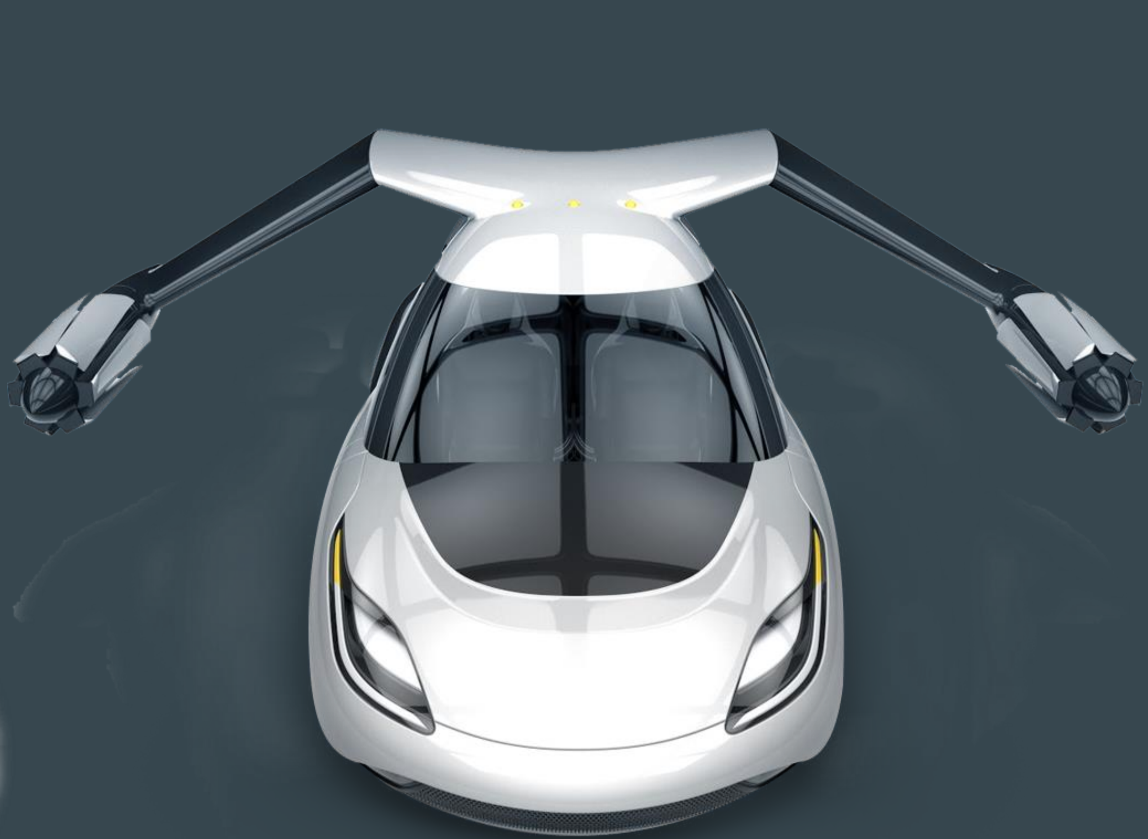
Level 2: a few tasks automated (L1 + slow down/break in front of obstacles, stay in lane)

Level 3: Some decisions (L2 + look around, decide to change lanes, pass, A7 prototype)

Level 4: Car handles many situations by itself in “safe” places (e.g. highway low-traffic)

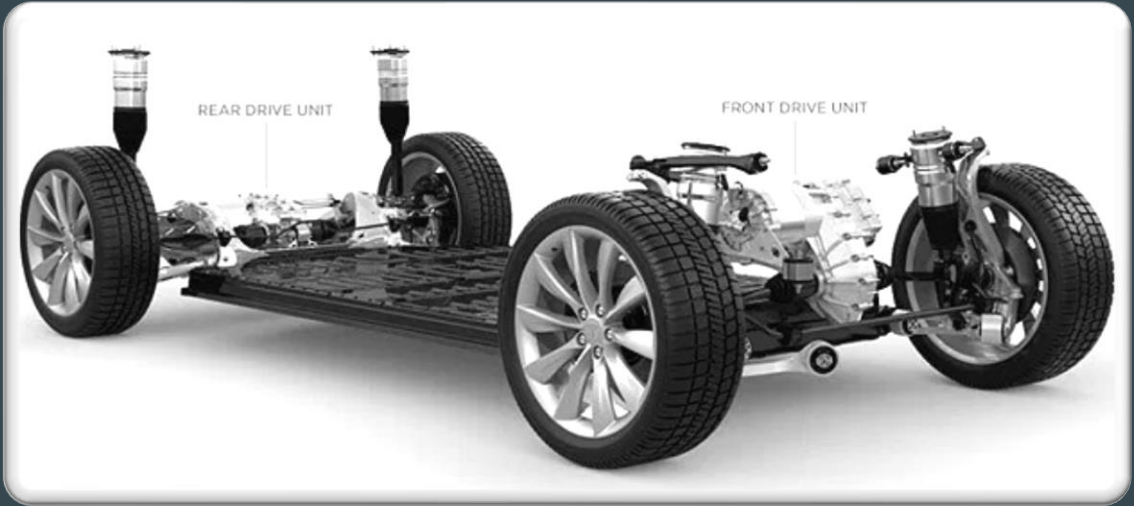
Level 5: Humans are NOT allowed to drive, the car does it all (Google car)

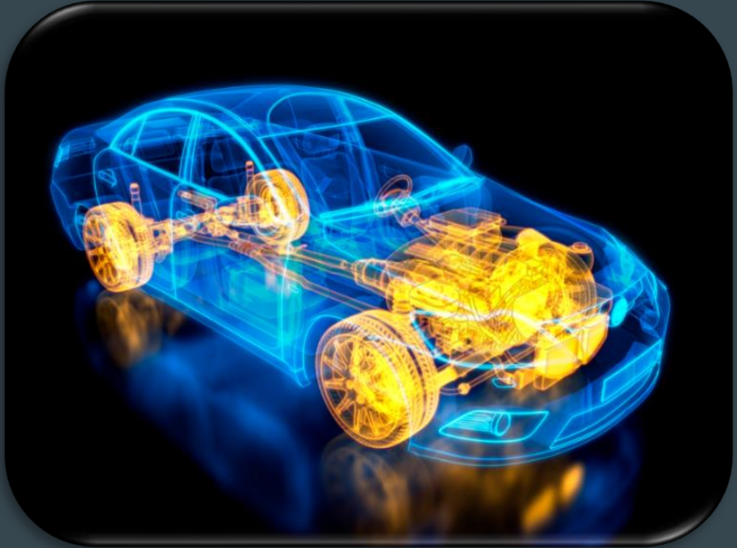
Level 6: Cars can fly or planes can drive (Terrafugia TF-X)



AdaCore

2. Shift in engineering complexity





AdaCore

Tesla to produce batteries for significantly less cost

General Motors Company

NYSE: GM - May 22, 3:59 PM EDT

32.93 USD ↑0.21 (0.64%)

1 day | 5 day | 1 month | 3 month | 1 year | 5 year | max

Open	32.97	Mkt cap	50.10B
High	33.25	P/E ratio	5.1
Low	32.74	Div yield	4.61%

Google Finance - Yahoo Finance - MSN Money Disclaimer

Tesla Inc

NASDAQ: TSLA - May 22, 4:20 PM EDT

310.35 USD ↓0.48 (0.15%)

After-hours: 310.34 ↓0.00%

1 day | 5 day | 1 month | 3 month | 1 year | 5 year | max

Open	312.80	Mkt cap	50.61B
High	314.37	P/E ratio	-
Low	306.80	Div yield	-

Google Finance - Yahoo Finance - MSN Money Disclaimer

AdaCore

3. Ownership Paradigm

Why do we need to own a car which will sit unused most of the time?

AdaCore

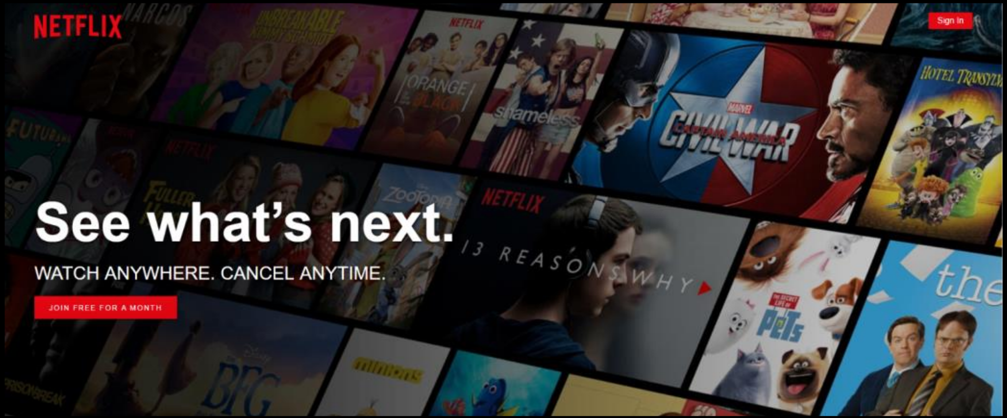
Music

I owned music records

My daughters not

AdaCore

Netflix does not own its servers



amazon
web services™

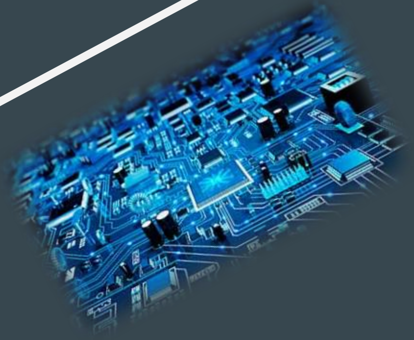
AdaCore

On Disruptive Innovation

Telephony

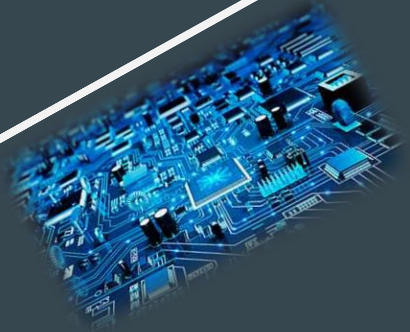
AdaCore

Electronics and software made this possible



Phones are still in the communication business, voice is just one many things communicated

Will electronics and software make this possible ?



What’s at stake: value creation

There is already a lot of SW in a car

Jet fighter: 10+ M SLoc

A380 - B787: 100+ M SLoc

Modern car: 100+ M Sloc

Much more software coming

OEMs we spoke to

SW development getting more complex

challenge to maintain quality at a reasonable cost.

life cycle is 5 years for a car

But software improvements are done every year

Self driving vehicles require a different attitude towards

Safety	(accidental crashes)
Security	(cars being used as weapons on a mass scale)

Avionics has similar challenges

Software Engineering Institute
Carnegie Mellon University

FAA RESEARCH PROJECT ON SYSTEM COMPLEXITY EFFECTS ON AIRCRAFT SAFETY: IDENTIFYING THE IMPACT OF COMPLEXITY ON SAFETY

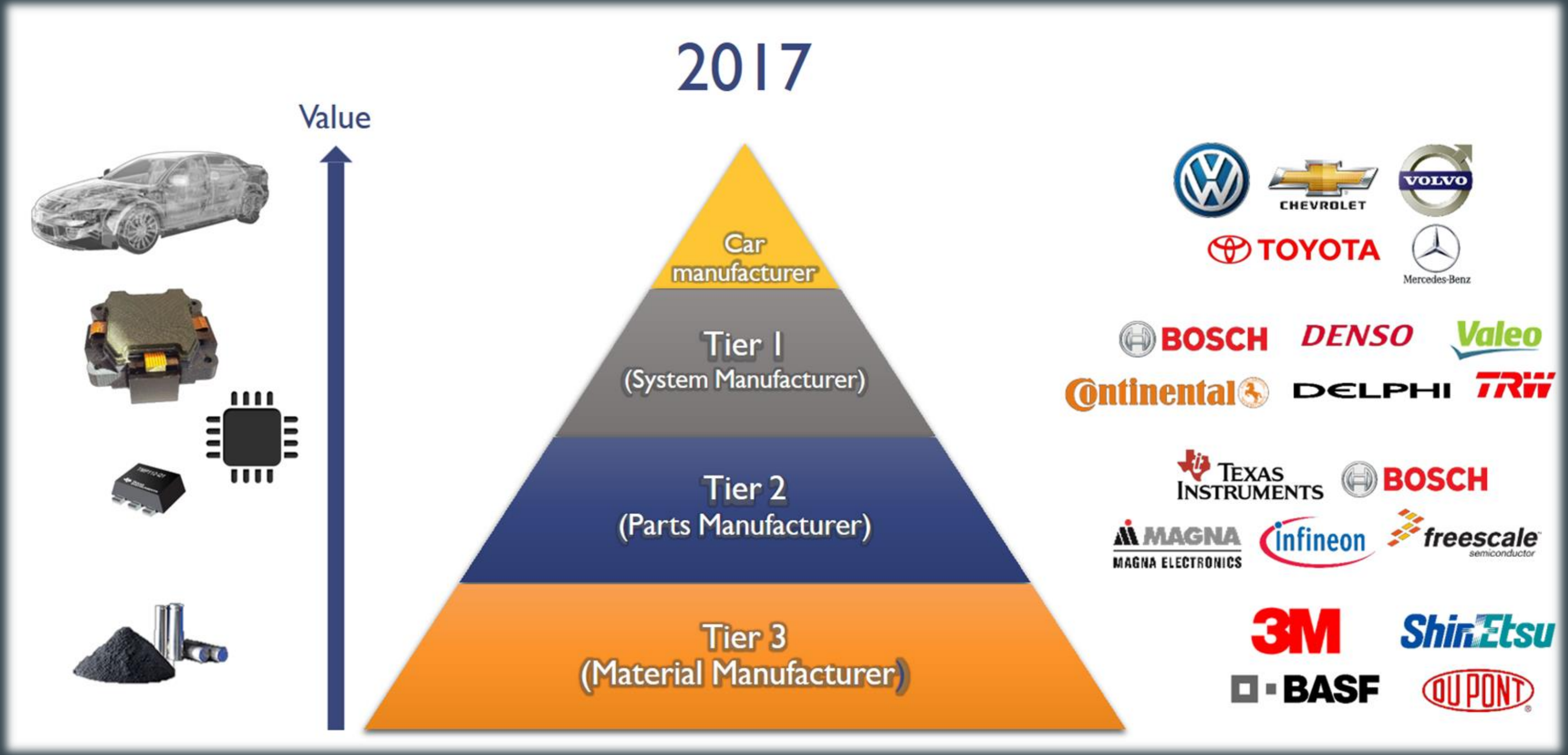
Sarah Sheard, Chuck Weinstock, Michael Konrad, and Donald Firesmith

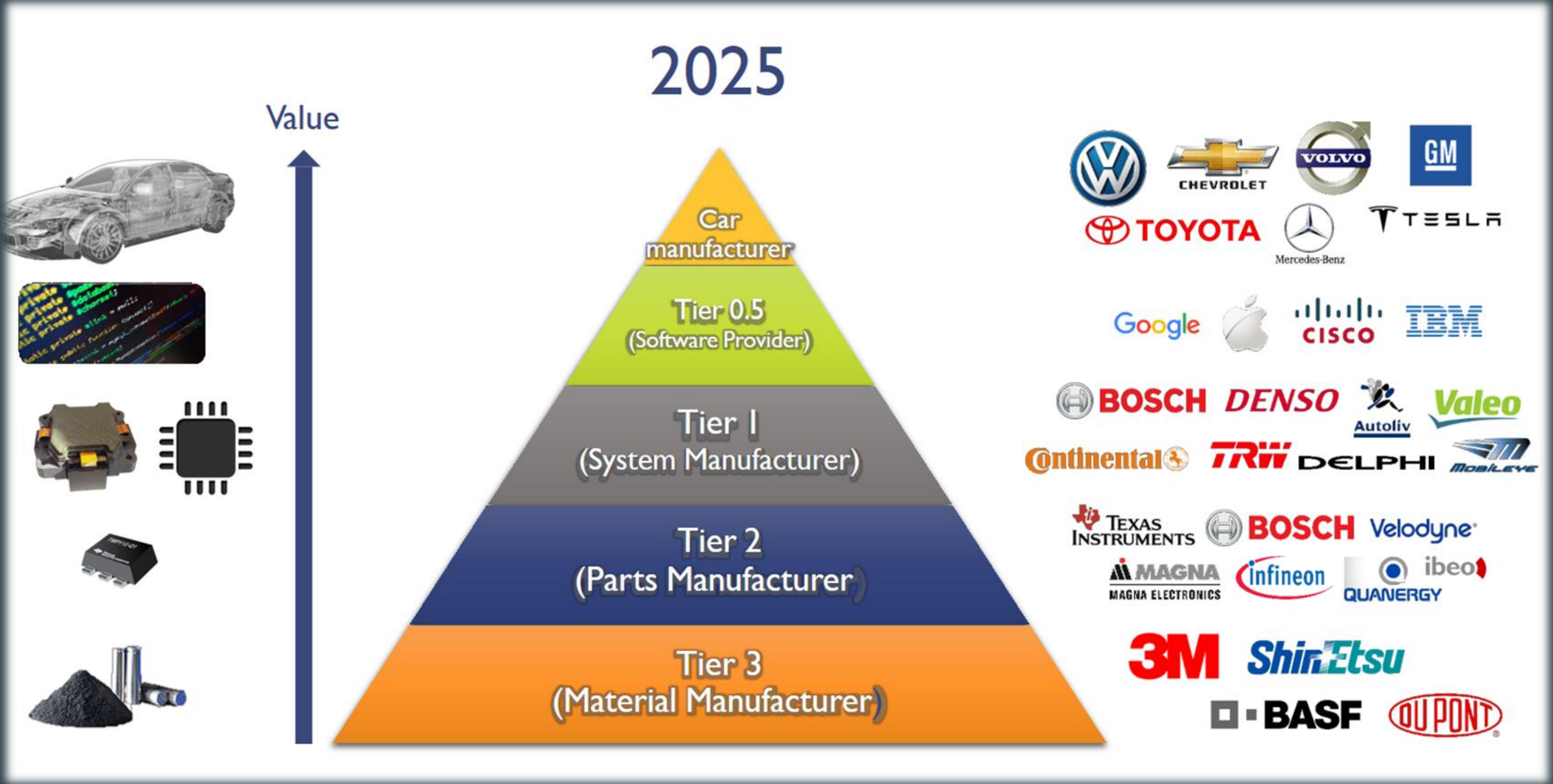
July 2015

Summary

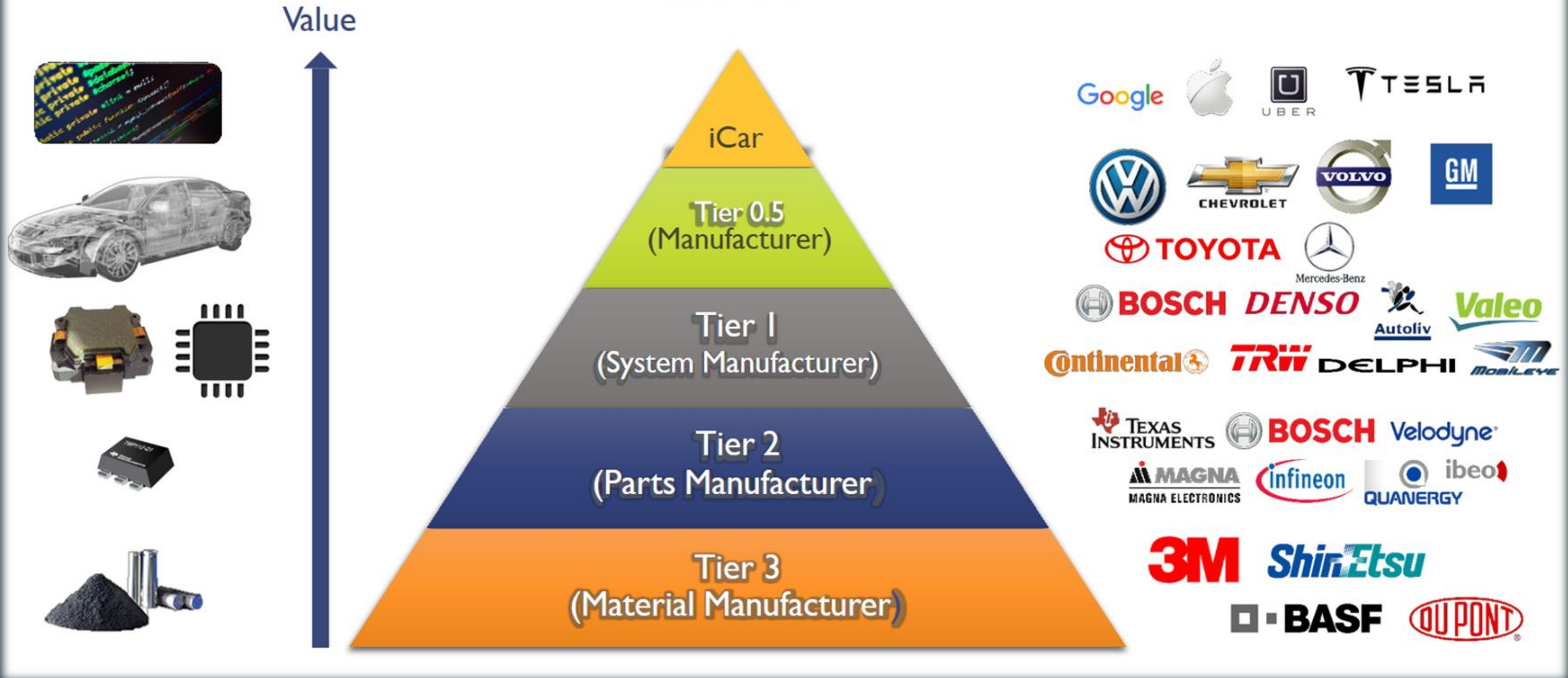
Current best practice— —is unable to cope with the exponential growth in size and interaction complexity of embedded software in today's increasingly software-reliant systems.

Who will solve the automotive software challenge?




2025
Value
Car manufacturer
Tier 0.5
(Software Provider)
Tier 1
(System Manufacturer)
Tier 2
(Parts Manufacturer)
Tier 3
(Material Manufacturer)
CHEVROLET
VOLVO
GM
TOYOTA
Mercedes-Benz
TESLA
Google
CISCO
IBM
BOSCH
DENSO
Autoliv
Valeo
Continental
TRW
DELPHI
MOBILEYE
TEXAS INSTRUMENTS
BOSCH
Velodyne
MAGNA
MAGNA ELECTRONICS
infineon
QUANERGY
ibeo
3M
ShinEtsu
BASF
DUPONT

2040



Value
iCar
Tier 0.5
(Manufacturer)
Tier 1
(System Manufacturer)
Tier 2
(Parts Manufacturer)
Tier 3
(Material Manufacturer)
Google
UBER
TESLA
CHEVROLET
VOLVO
GM
TOYOTA
Mercedes-Benz
BOSCH
DENSO
Autoliv
Valeo
Continental
TRW
DELPHI
MOBILEYE
TEXAS INSTRUMENTS
BOSCH
Velodyne
MAGNA
MAGNA ELECTRONICS
infineon
QUANERGY
ibeo
3M
ShinEtsu
BASF
DUPONT

Take away

Electronics & SW in the front seat

Mechanics → Mechatronics → Softchanics

This will take time, but when building automotive SW we have to master

time-to-market

complexity

safety & security

conflicting requirements

This is always a moving target ☺

AdaCore

Safety & security standards to the rescue?

Safety standards

DO-178 B/C or ED-12B/C	Avionics
IEC 61508	Industrial automation at large
CENELEC EN 50128	Railway
ISO 26262	Automotive

Prescribe “due diligence” during system construction

1,000s of artifacts produced and reviewed (requirements, design, ...)

TESTING is a significant part of the “due diligence”

AdaCore

The challenge with safety standards, DO-178 example

1982: DO-178

- Basic SW design assurance
- 3 DAL (design assurance levels) ie 3 levels of SW safety

1985: DO-178A

- Testing and configuration management

1992: DO-178B

- 5 DALs
- Focus moves from testing to requirements

2012: DO-178C

- Clarifies tool qualification
- Adds Model Based Development and Verification, OO technology, formal methods

1982 to 2012 - 30 years gone by

Technology evolves faster than Safety standards

Work on a new standard started when the technology is already there (example of Model Based Development, e.g. Simulink)

By the time the new standard is out technology is moving on.

So work has started in 2015-2016 on a meta approach

What are the overarching principles of a safety standard?

EASA, FAA, et al.

> “overarching principles to streamline the overall certification process, looking at it from a system perspective rather than prescriptive rules”

Trying to combine

- system standard (ARP4754A)
- software standard (DO-178C)
- complex hardware standard (DO-254)

Meta-certification approach

AdaCore

Overarching properties

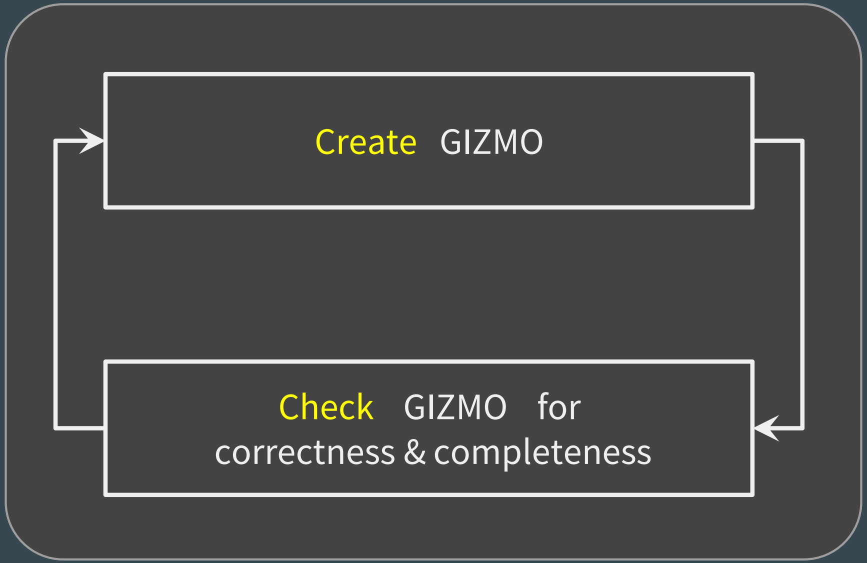
1. Intent *what is the system supposed to do and how does it address failures ?*

2. Correctness *is the implementation of the intent OK ?*

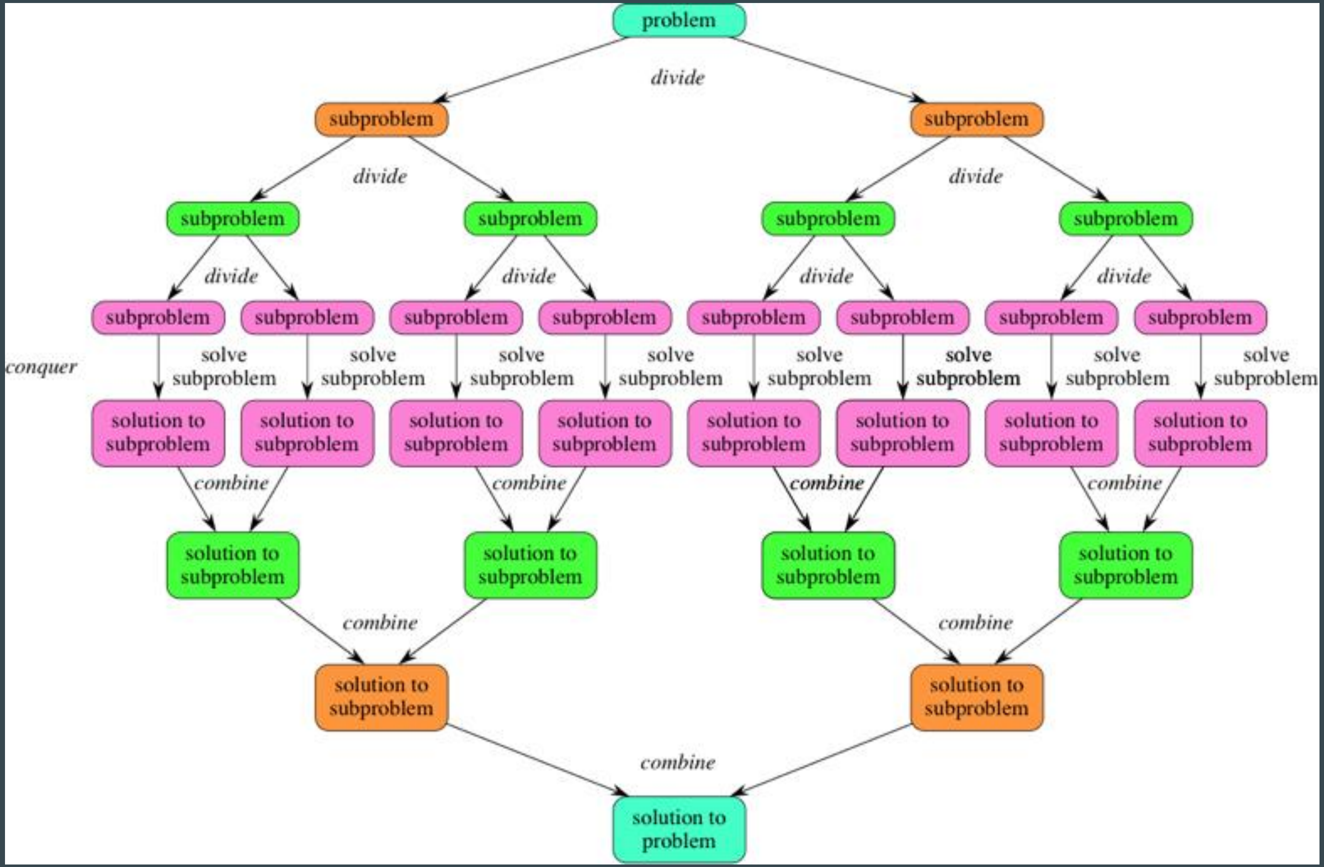
3. Necessity *are all the items in the implementation needed ?*
If not, can the additional items be shown not to affect safety ?

Some aspects of engineering systems

One aspect of engineering

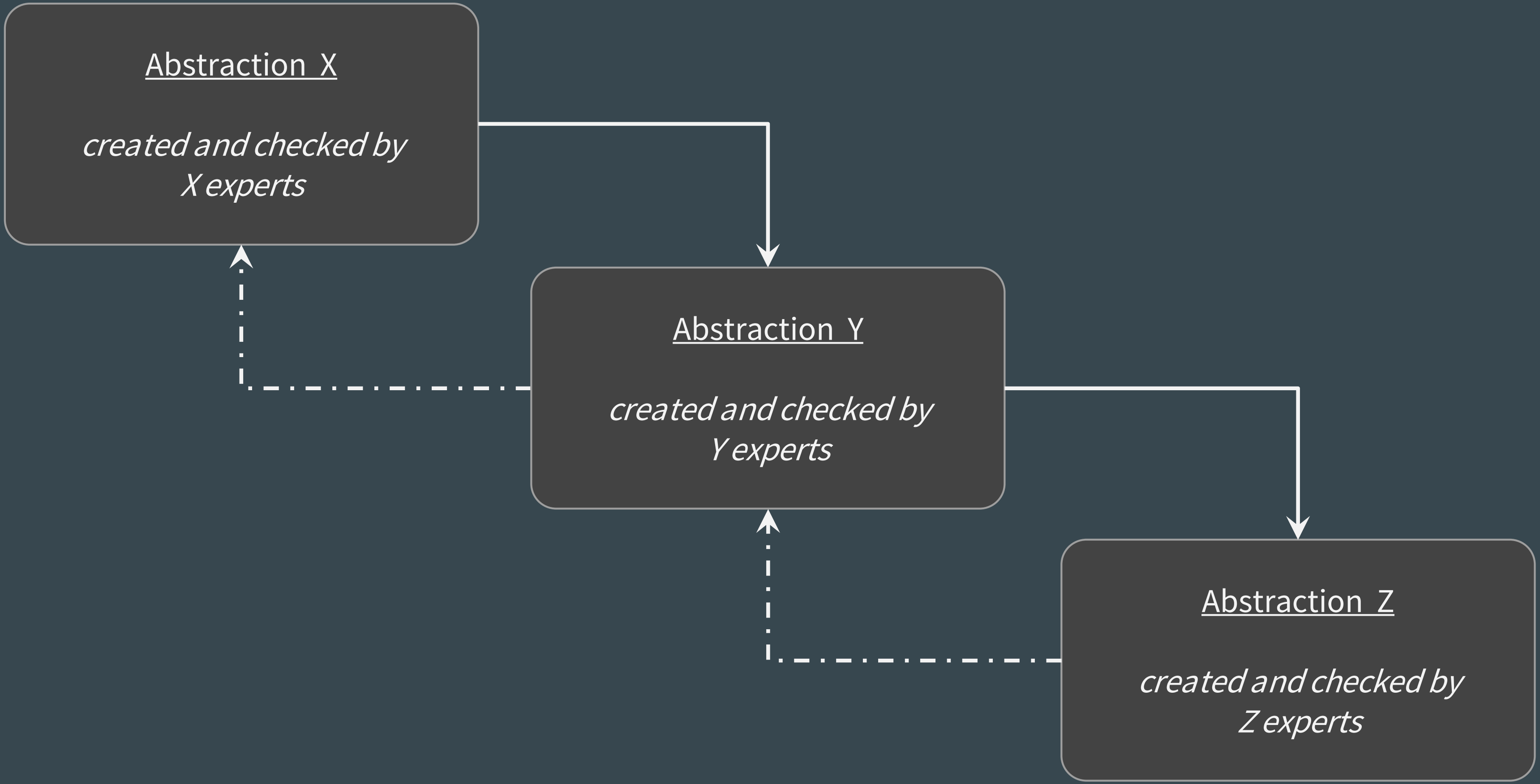


Another: divide, create, integrate



AdaCore

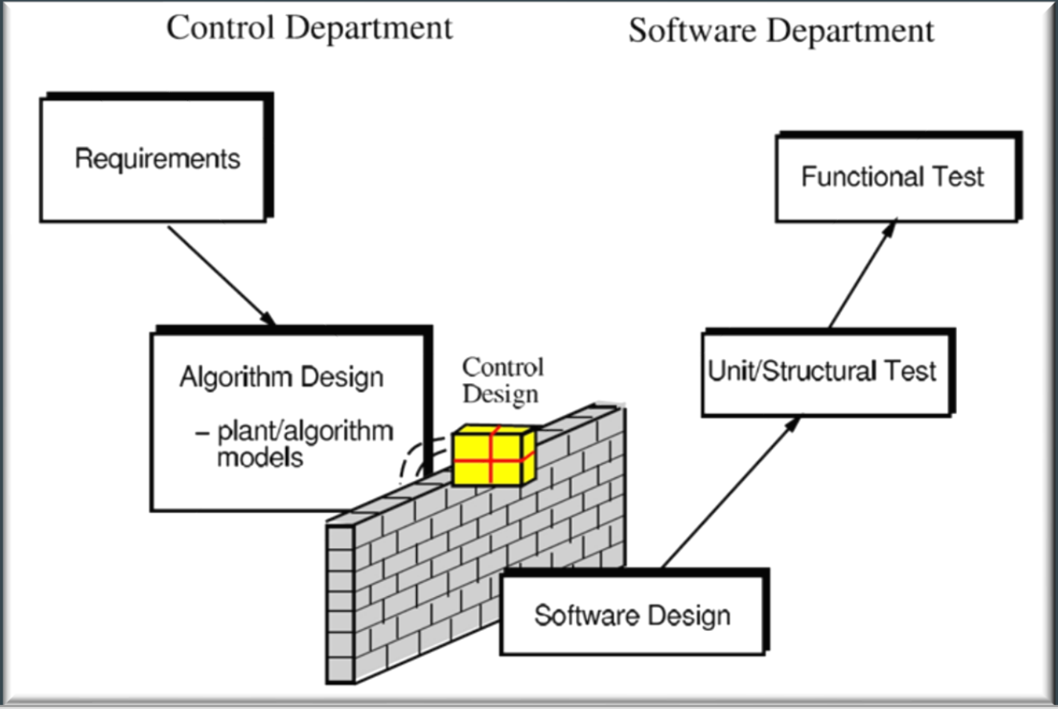
Tiers of decomposition = refinements of abstraction



Some engineering challenges

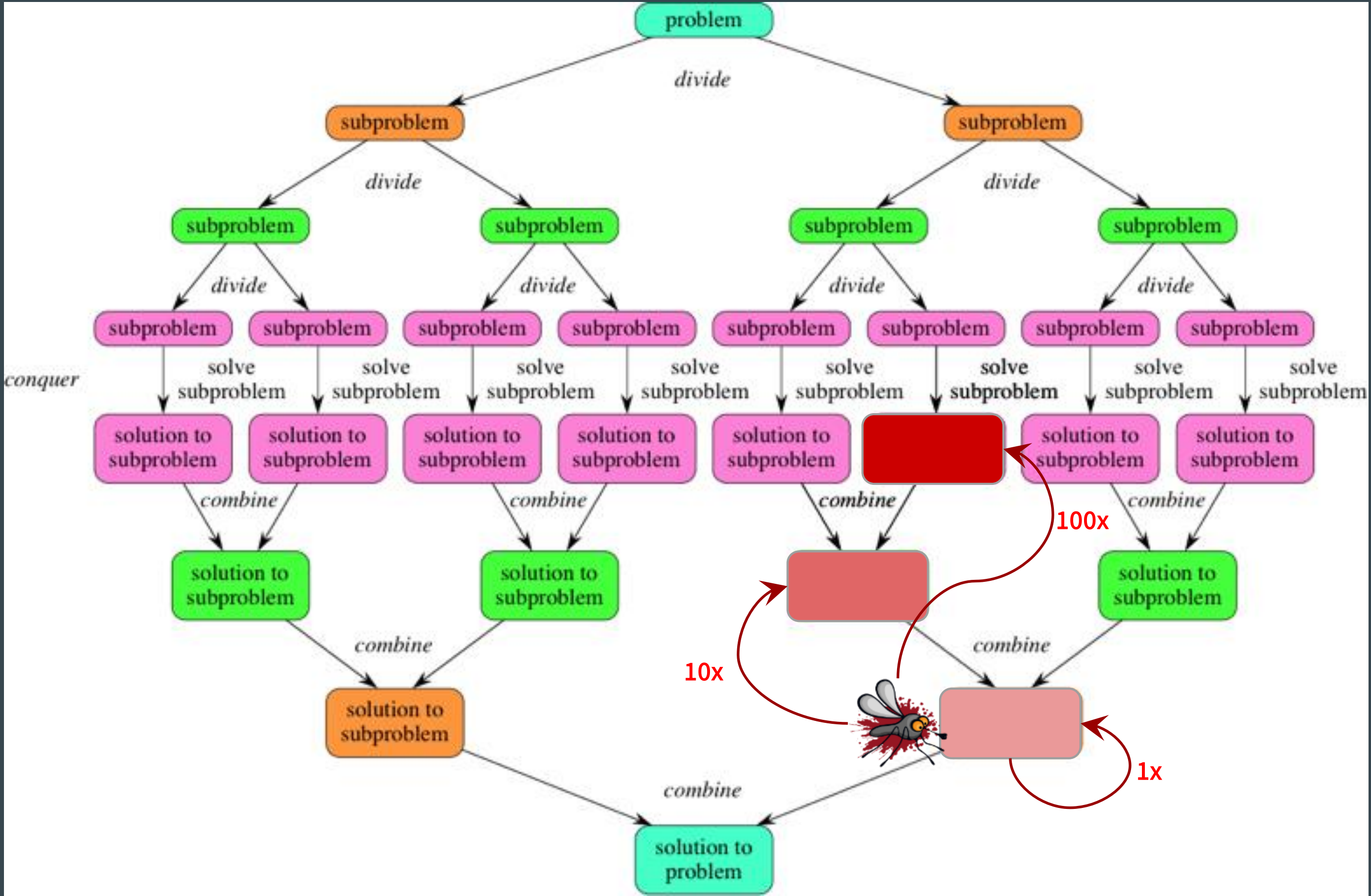
Challenge: Experts

Getting experts to work together and understand each other

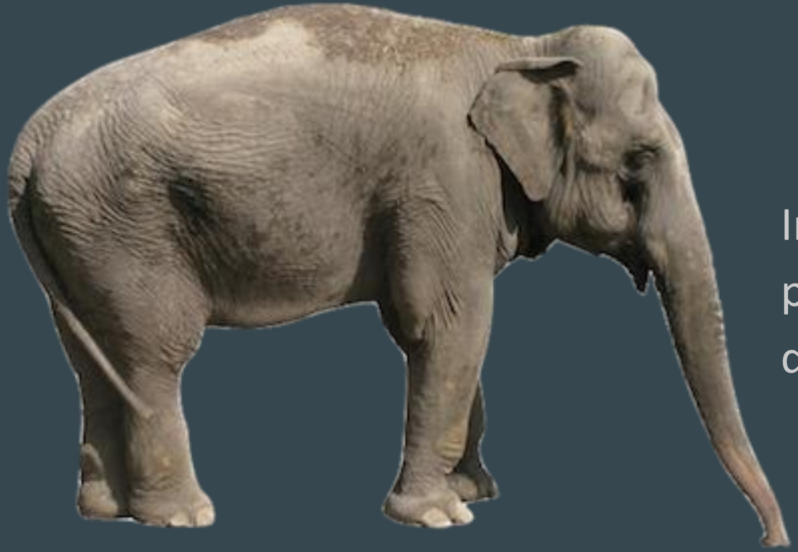


P. Albertos, Instituto de Automática e Informática Industrial, Universidad Politécnica de Valencia

Challenge: Cost of fixing problems



Challenge: The war on talent



In developed countries the need for engineerings professionals outpaces what local universities can deliver. This is particularly true in software.

How can businesses meet their engineering needs?

AdaCore

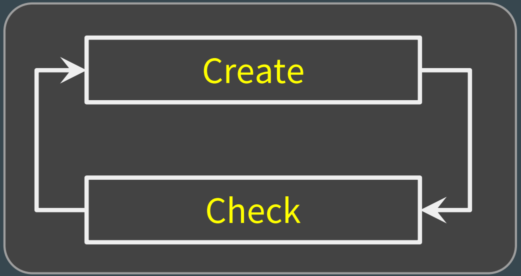
Solutions anyone?

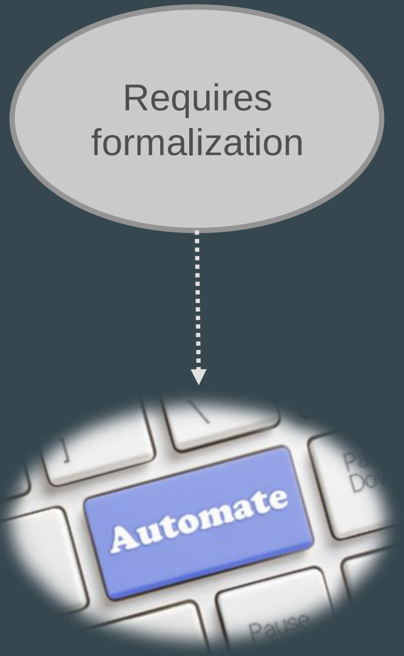
Getting experts to work together

Encourage cooperation across departments

“Bi-lingual” tools to help experts communicate and understand each other (an example later on)

Finding problems ASAP





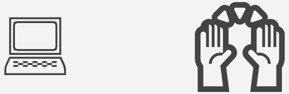
Humans are good at creating abstractions

Machines are good at execution (following orders)

checking

...

5X Sloc of tests per operational Sloc

Abstraction created has	Execution	Checking
No machine-readable semantics		
Potentially ambiguous semantics		
Clear semantics		
Clear semantics + ways to state intent		

If we have ...

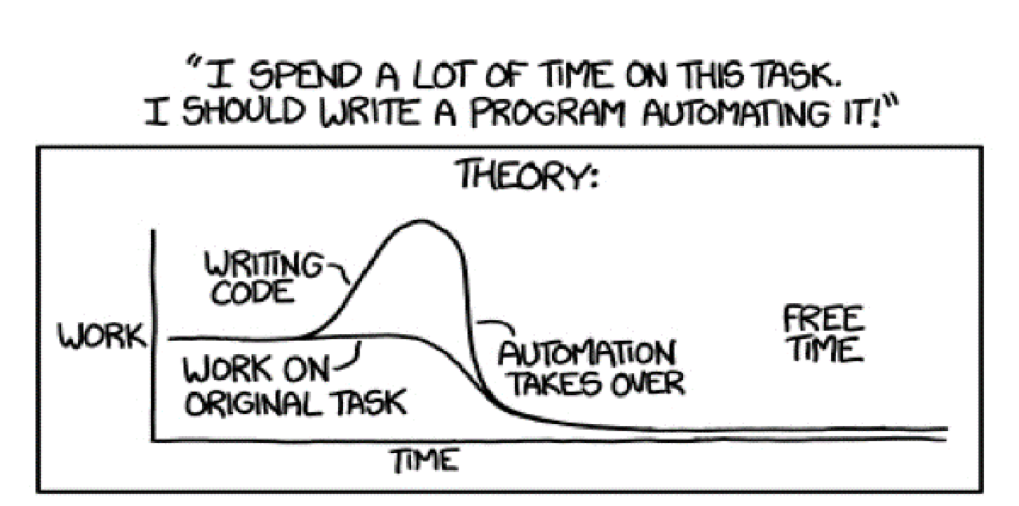
Formalized semantics

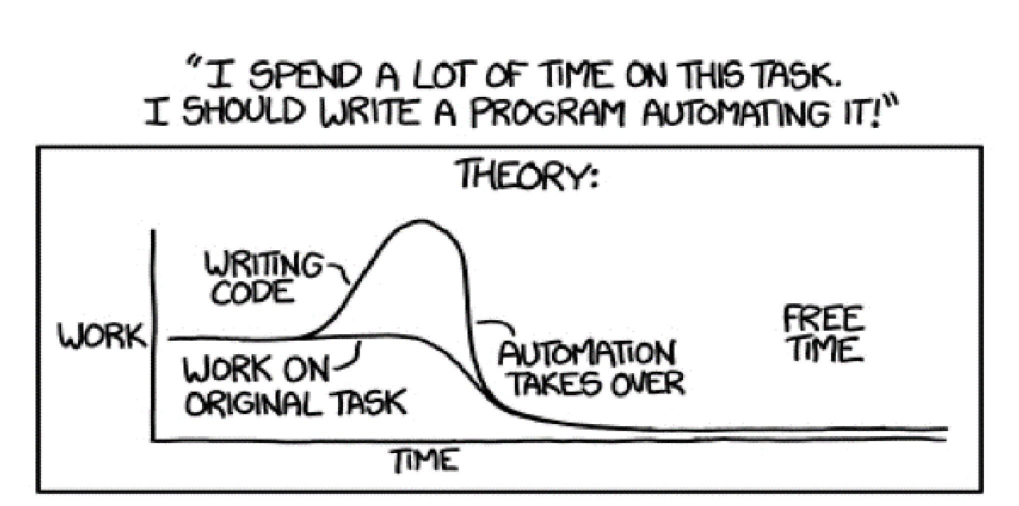
→ Machines can do part of the work

→ can check some inconsistencies

Formalized semantics + ways to state intent

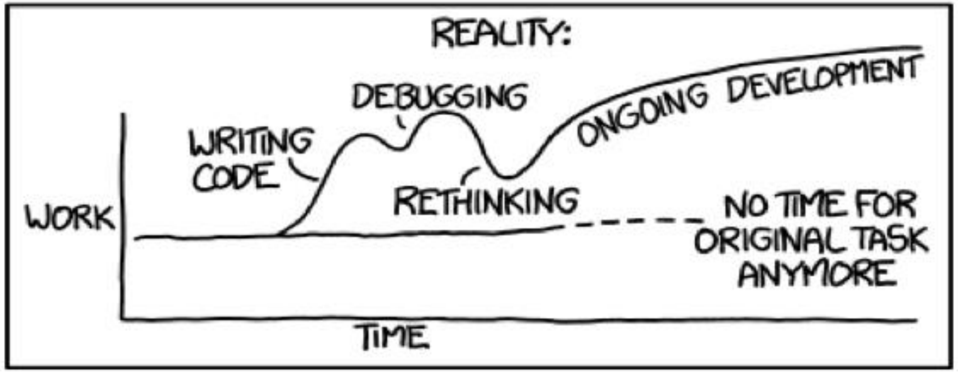
1. *State properties of your abstractions*
2. Machines can check property-preservation *(ideally across abstraction layers)*


"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"
THEORY:
WRITING CODE
WORK
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME



... formalizing & automating takes time and is a tradeoff

AdaCore



An obligatory XKCD

AdaCore

Intent and formalization

Specifying & checking intent today

Intent: function of many things

- what we can implement, cost, time-to-market, know-how, hiring ...

Some aspects of intent clarified during implementation

Specifying intent: creative activity, hard to formalize as a whole

Checking intent: by hand as part of the whole process (we should do better)

AdaCore

Checking safety & security properties

Use formalisms (likely domain-specific) to

Specify key safety & security properties of the intent

Machines check their consistency (completeness checked by humans)

Machines check properties hold across abstraction layers

AdaCore

Example: UAV Mission Management System

Ranges [a:p:b] (u): from min a, to max b, with increments p, physical unit u.

Climb safety constraints:

- max take-off speed: 75kt,
- climb rate guaranteed in [0.3, 3] m/s,
- precision of flight level capture: +/- 50ft,
- precision of speed capture: +/- 5kt.

Cruise safety constraints:

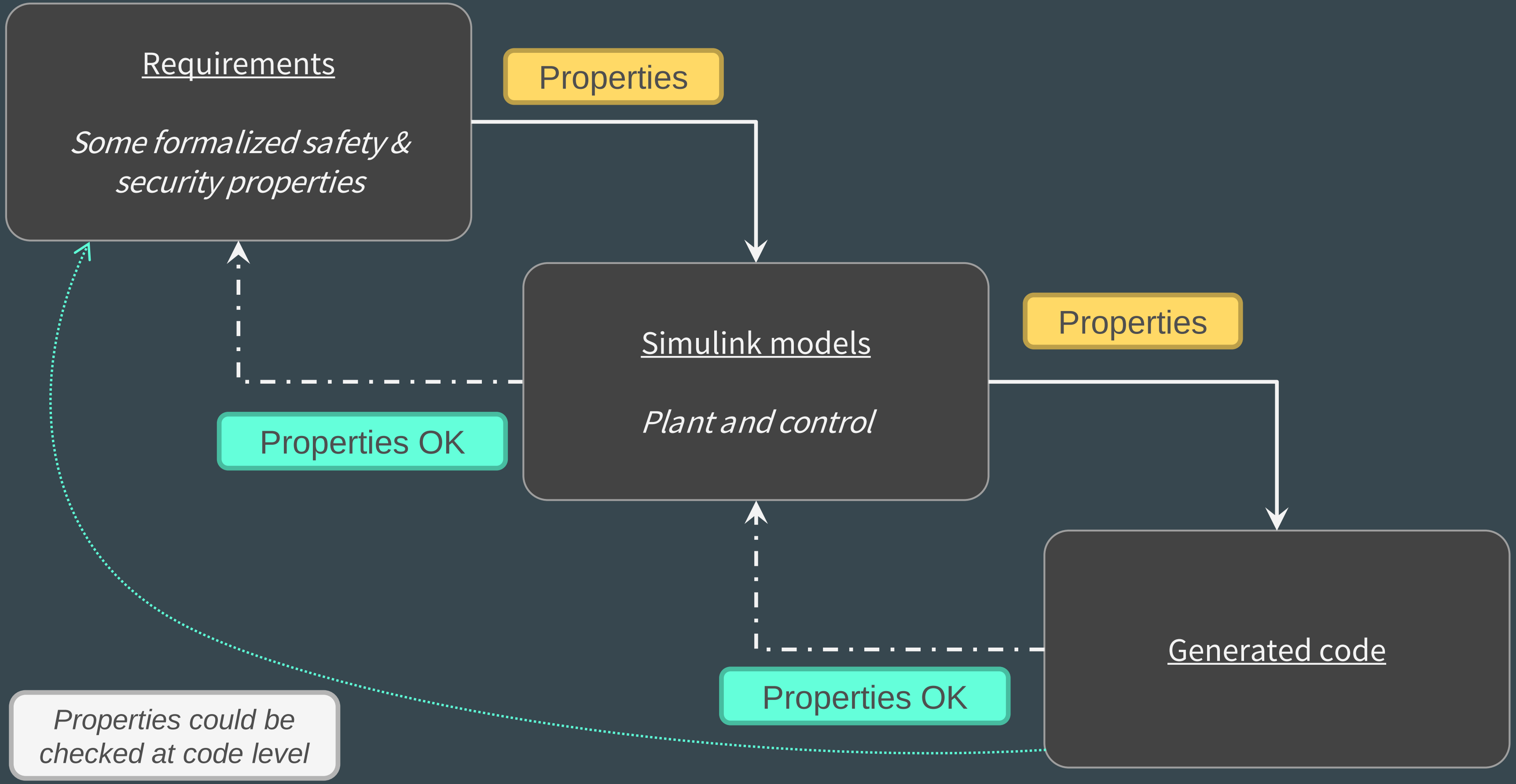
- Minimum flight level: 500ft,
- Maximum flight level: 1500ft,
- Flight level precision: +/- 50ft,
- Maximum speed: 125kt,

Descent safety constraints:

- descent rate guaranteed in [0.1, 1] m/s,
- maximum landing speed: 25kt.

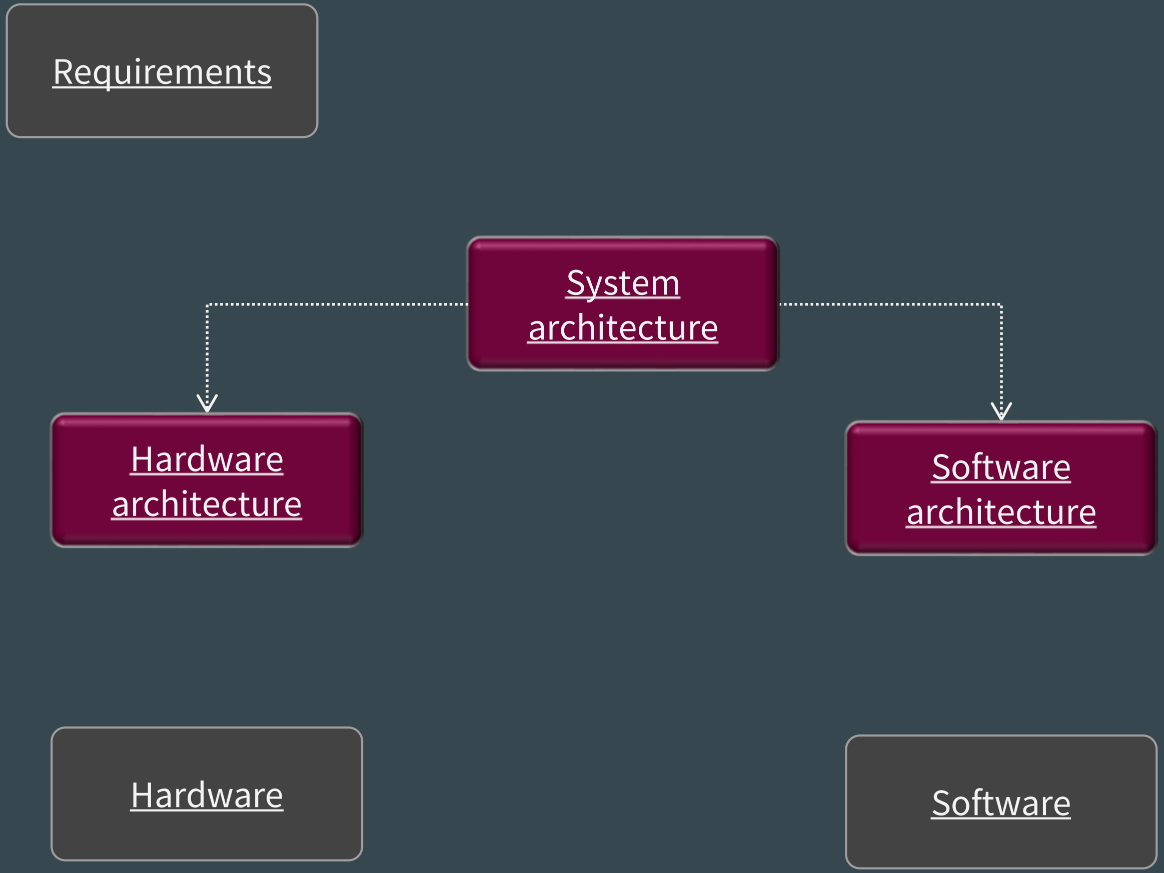
AdaCore

Example: Property preservation across ALs



Correctness of implementation

The architecture abstraction



The Architecture abstraction

How do we understand the impact of architectural decisions?

- Components cannot be safe & secure in isolation
- Need tools to identify mismatched assumptions in system interactions
- Work on secure architectures is an important topic

Today most diagrams (Sysml) don't have clear semantics.

Is AADL better?

Can asynchronous events interfere? Can SW components interfere?

In automotive FFI is a requirement

In the 100 M Sloc in a car

Plenty of non critical SW (e.g. audio/video) in theory

How do we know this SW does not interfere with the rest?

Automotive has a word for this FFI

Even this is not enough: if hack car entertainment system + cranks up the volume + puts unbearable frequency + cannot turn off the radio …

Windshield sprays hacked + start spraying your windshield + disabling wipers

if something can be hacked it will

Secure architectures is an important area

Researchers' car hacking demonstrations show need for secure architecture

keeping cars' control circuits separate from the Internet ones

Not sharing a common bus between critical + non critical components

Software & correctness

Major classes of software as I understand them in automotive

Control laws: Simulink ®

OS, drivers, ...: Hand-written

Other: Hand-written

Pattern matching: Deep learning techniques

Did I miss something?

Correctness & Deep learning

Deep learning: the key is in the data & learning

Universal Adversarial Perturbations Against Semantic Image Segmentation

Jan Hendrik Metzen
Bosch Center for Artificial Intelligence, Robert Bosch GmbH
janhendrik.metzen@de.bosch.com

Mummadi Chaithanya Kumar
University of Freiburg
chaithu0536@gmail.com

Thomas Brox
University of Freiburg
brox@cs.uni-freiburg.de

Volker Fischer
Bosch Center for Artificial Intelligence, Robert Bosch GmbH
volker.fischer@de.bosch.com

Abstract

While deep learning is remarkably successful on perceptual tasks, it was also shown to be vulnerable to adversarial perturbations of the input. These perturbations denote noise added to the input that was generated specifically to fool the system while being quasi-imperceptible for humans. More severely, there even exist universal perturbations that are input-agnostic but fool the network on the majority of inputs. While recent work has focused on image classification, this work proposes attacks against semantic image segmentation: we present an approach for generating (universal) adversarial perturbations that make the network yield a desired target segmentation as output. We show empirically that there exist barely perceptible universal noise patterns which result in nearly the same predicted segmentation for arbitrary inputs. Furthermore, we also show the existence of universal noise which removes a target class (e.g., all pedestrians) from the segmentation while leaving the segmentation mostly unchanged otherwise.

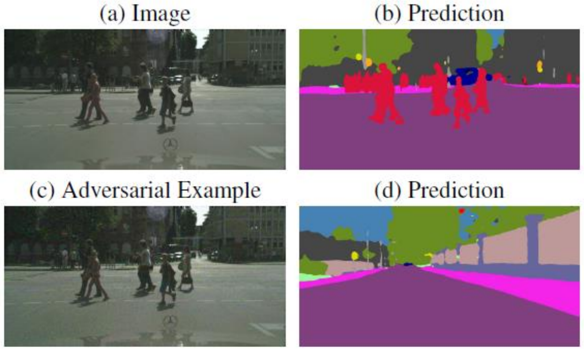
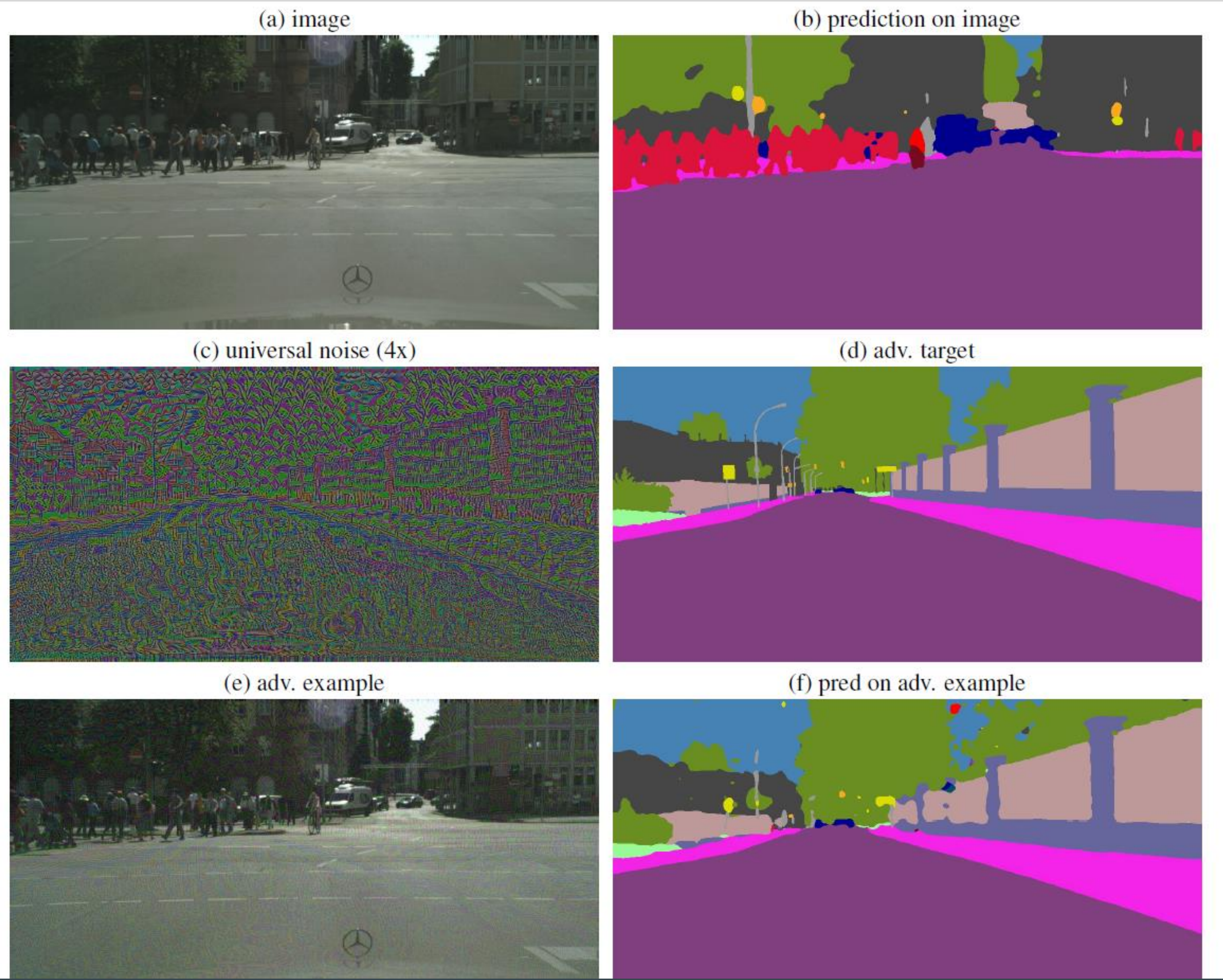


Figure 1. The upper row shows an image from the validation set of Cityscapes and its prediction. The lower row shows the image perturbed with universal adversarial noise and the resulting prediction. Note that the prediction would look very similar for other images when perturbed with the same noise (see Figure 3).


(a) image
(b) prediction on image
(c) universal noise (4x)
(d) adv. target
(e) adv. example
(f) pred on adv. example

Correctness & hand-written code

Testing challenges

Software Engineering Institute
Carnegie Mellon University

FAA RESEARCH PROJECT ON SYSTEM COMPLEXITY EFFECTS ON AIRCRAFT SAFETY: IDENTIFYING THE IMPACT OF COMPLEXITY ON SAFETY

Sarah Sheard, Chuck Weinstock, Michael Konrad, and Donald Firesmith

July 2015

Summary

The complexity and the nondeterministic nature of software interaction requires formal static analysis methods to complement testing, with consistency across analysis models.

SPARK – robustness & programming by contract

Checks inconsistencies — out-of range values, dead exec paths…

Checks absence of run-time errors — buffer overflows, divide by 0, …

Robustness is there by construction, NO “conventional” robustness testing necessary

Specify properties & have them verified — FFI

SPARK

```
procedure Stabilize (Mode: in Mode_Type; Success: out Boolean)
```

AdaCore

SPARK – a more complete spec

```
procedure Stabilize (Mode: in Mode_Type; Success: out Boolean)
   with Global => (Input => (Accel, Giro), In_Out => Rotors),
```

AdaCore

SPARK – programming by contract

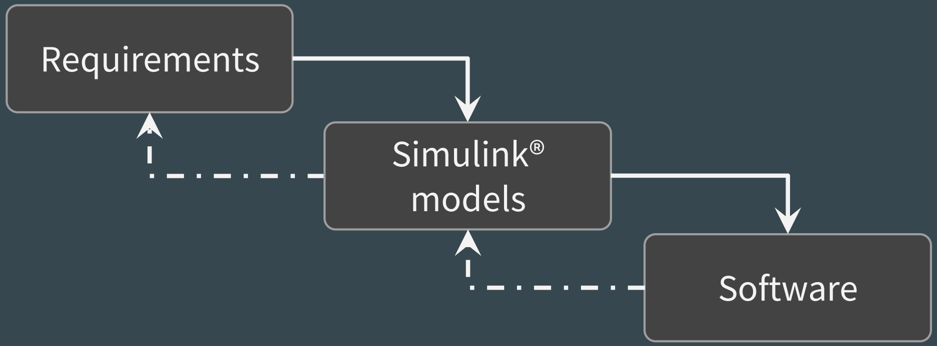
```
procedure Stabilize (Mode: in Mode_Type; Success: out Boolean)
   with Global => (Input => (Accel, Giro), In_Out => Rotors),
        Pre  => Mode /= Off,
        Post => (if Success then Delta_Change (Rotors'Old, Rotors));
```

Spark-2014.org & libre.adacore.com

AdaCore

Correctness & MBE

MBE in Automotive



Modeling the laws of physics

Creating a controller

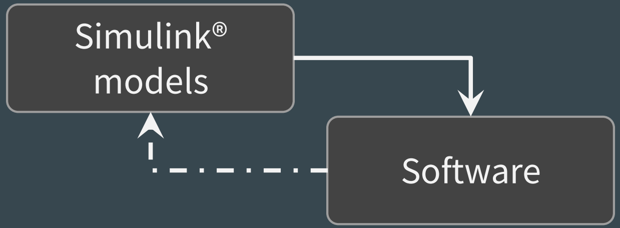
Checking (by simulation) the controller in its physical context (the “plant”)

Translating the controller into software

“Plant and controller models” written & simulated with Simulink®

Machines (autocoding) translate: what is the state of the art ?

Autocode generation



Generated code

- Consistent with simulation
- Customizeable
- Can integrate & seamlessly debug hand-written code

Trusted

How do you check if the controller contains runtime errors ?

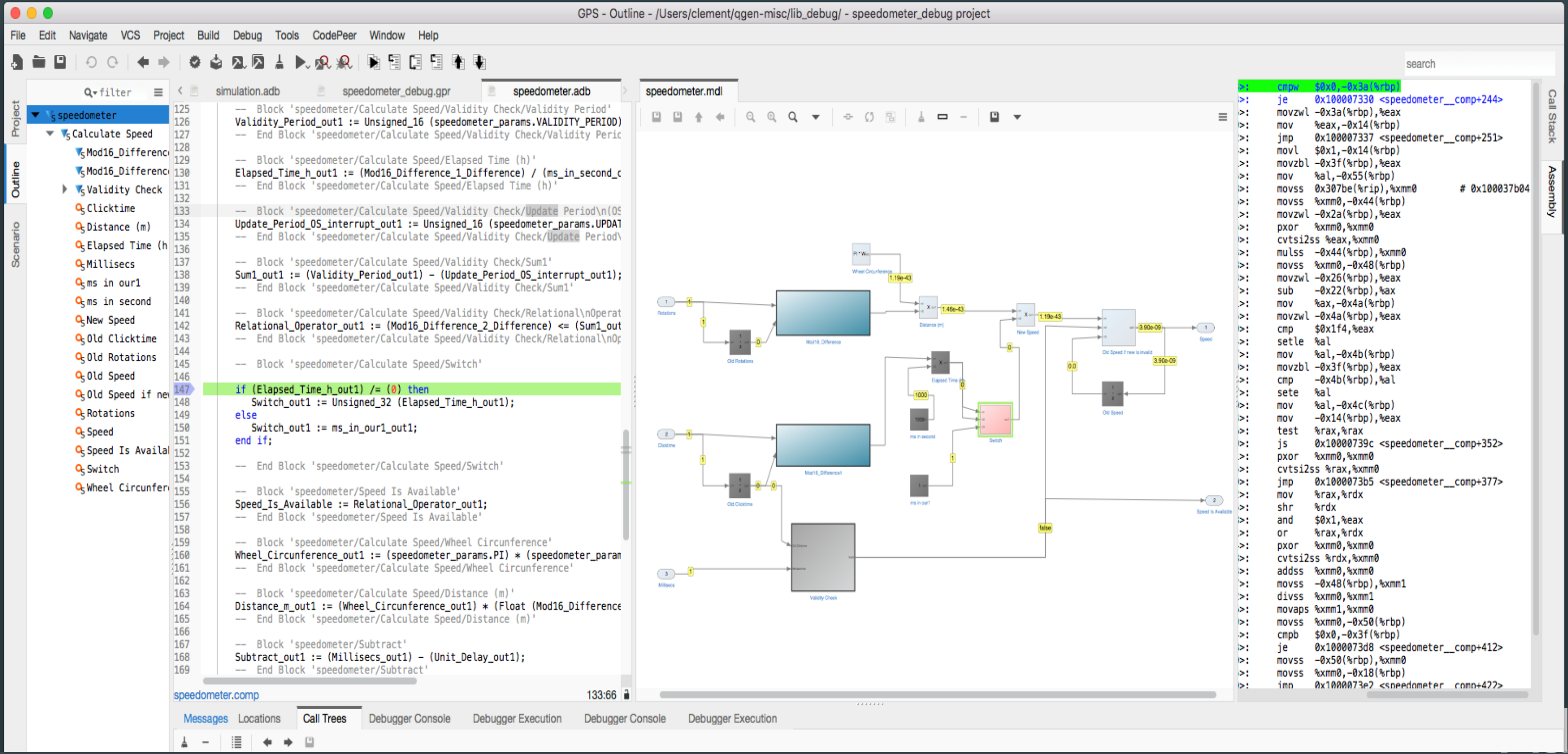
- Code generator comes with a static verifier

What happens when upgrading to a new version of Simulink® ?

- Code generator produces the same code for the same models

A joint model & code debugger helps experts work together

SIL + PIL debugging, what-if scenarios *(test difficult behaviors on the system by specifying signals that might be hard to generate by conventional testing)*



Conclusion

Engineering is the art of compromise

Complexity of engineering safe & secure systems ↗

if formalization ↗ help from tools ↗

but

too much formalism work ↗↗

AdaCore

Use the right balance of formalism to

Specify key safety & security properties of the intent

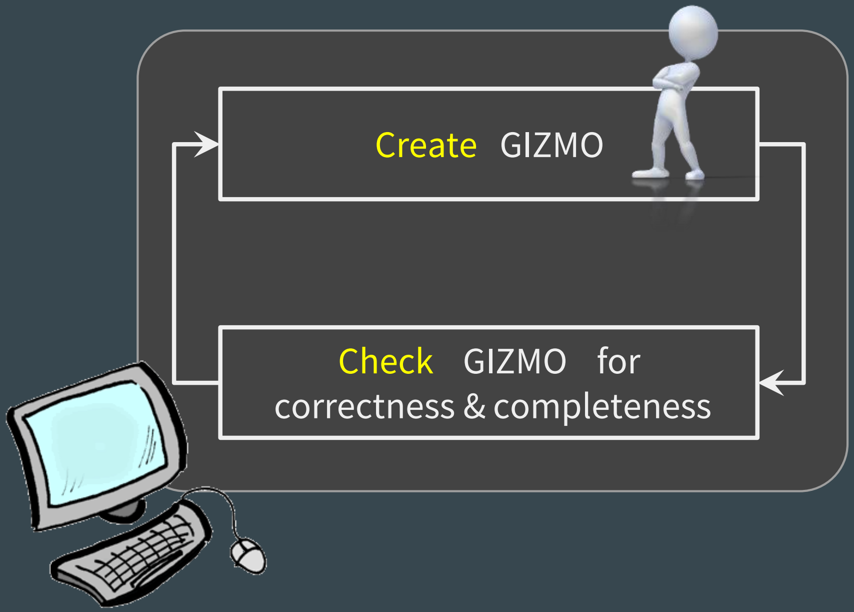
Have tools check these properties through layers of abstraction

AdaCore

Keep humans creative

Use technologies that

- Allow to specify safe & secure properties
- Check them
- Limit the introduction of flaws
- Detect errors early (e.g. inconsistencies)



Each OP is stated in 1 page

1. Statement
2. Definitions
3. Pre-requisites (which must exist to allow OP satisfaction to be shown)
4. Constraints (on how OP satisfaction must be demonstrated)
5. Assumptions (which need only be stated, not justified)

Definitions

- <u>Desired system behavior</u>: System needs and constraints expressed by the stakeholders
- <u>Defined intended functions (DIF)</u>: The record of the system needs and constraints as expressed by stakeholders
- <u>Failure Condition(s)</u>: A condition having an effect on the aircraft and/or its occupants, either direct or consequential, which is caused or contributed to by one or more failures or errors, considering flight phase and relevant adverse operational or environmental conditions or external events (from ARP 4754A)
- <u>Foreseeable operating conditions</u>: External and internal conditions in which the system is used, encompassing all known normal and abnormal conditions
- <u>Unacceptable Safety Impact</u>: An impact which compromises the system safety assessment
- <u>Implementation</u>: *Item* or collection of items contributing to system realization, for which acceptance or approval is being sought
    - <u>Item</u> (from ARP 4754A) is a hardware or software element having bounded and well-defined interfaces

Constraints applying to all OPs

- The process to satisfy each OP must be defined and conducted as defined
- Criteria for evaluating the artifacts are defined and shown to be satisfied individually and collectively
- All artifacts required to establish the OP are under configuration management and change control

Intent

The *defined intended functions* are correct and complete with respect to the *desired system behavior*.

Constraints

- The *defined intended functions* must address the *failure conditions*

Correctness

The *implementation* is correct with respect to its *defined intended functions*, under *foreseeable operating conditions*

Constraints

- When tiers of decomposition are used, the means of showing correctness among the tiers and to the *defined intended functions* must be defined and conducted as defined
- The *implementation* must be correct when functioning as part of the integrated system or in environment(s) representative of the integrated system

Necessity

All of the <u>implementation</u> is either required by the <u>defined intended functions</u> or is without <u>unacceptable safety impact</u>

Constraints

- The system safety assessment must address all of the implementation