CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Architecture and Agility for complex systems

The case of automotive domain

Patrizio Pelliccione

Associate Professor (Docent), Chalmers|GU

www.patriziopelliccione.com

About myself…

WASP WALLENBERG AUTONOMOUS SYSTEMS PROGRAM

Autonomous and smart systems

- **Adaptation**
- **Evolution**

https://www.chalmers.se/sv/styrkeomraden/ikt/forskning/automatiserat-samhalle/wasp/Sidor/default.aspx

4Robots

Achieving complex Collaborative Missions via Decentralized Control and Coordination of Interacting Robots

http://www.co4robots.eu/

NGEA
Next Generation Electrical Architecture

Next Generation Electrical Architecture – Volvo Cars and many suppliers

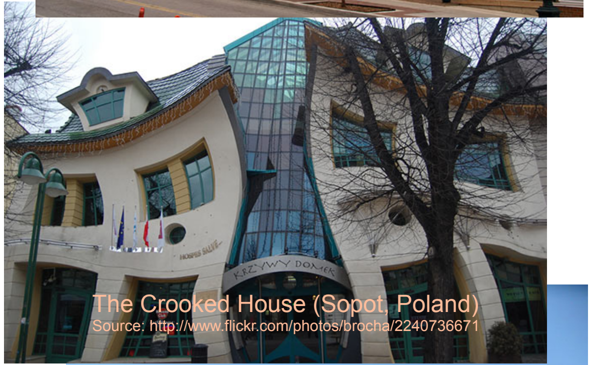
https://www.researchgate.net/project/Next-Generation-Electrical-Architecture-NGEA

Software architecture – building metaphor

- **No comparable intuition for software**
  - We must be more methodological and analytical in our approach
- **Software is intrinsically intangible**
  - More difficult to measure, analyze, and evaluate qualities
- **Software more malleable than physical building materials**
  - Types of changes unthinkable in a physical domain

Kansas City Public Library (Missouri, USA)

Source: http://www.flickr.com/photos/jonathan_moreau/418008212



The Crooked House (Sopot, Poland)

Source: http://www.flickr.com/photos/brocha/2240736671

Museum of Contemporary Art (Niteroi, Brazil)

Source: http://www.flickr.com/photos/howvin/7457617764/

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. 2009 ***Software Architecture: Foundations, Theory, and Practice***. Wiley Publishing.

Architecture Standard

- ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description – December 2011
    - Joint ISO and IEEE revision of IEEE Std 1471, first published in 2000
    - The standard is method-neutral: it is intended for use by architects employing various architecting1 methods.

Architecture: (system) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

http://www.iso.org/iso/catalogue_detail.htm?csnumber=50508

Some facts about architecture

- Every application has an architecture

Russian Embassy (Havana, Cuba)

http://upload.wikimedia.org/wikipedia/commons/d/d2/Russian_embassy_in_Havana.jpg

Some facts about architecture

- **Every application has an architecture**
  - The architecture of a system can be characterized by the principal design decisions made during its development

Architecture underlying command-line shell programs

```
ls invoices | grep -e August | sort
```

Architectural style pipe-and-filter

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. 2009 ***Software Architecture: Foundations, Theory, and Practice***. Wiley Publishing.

Some facts about architecture

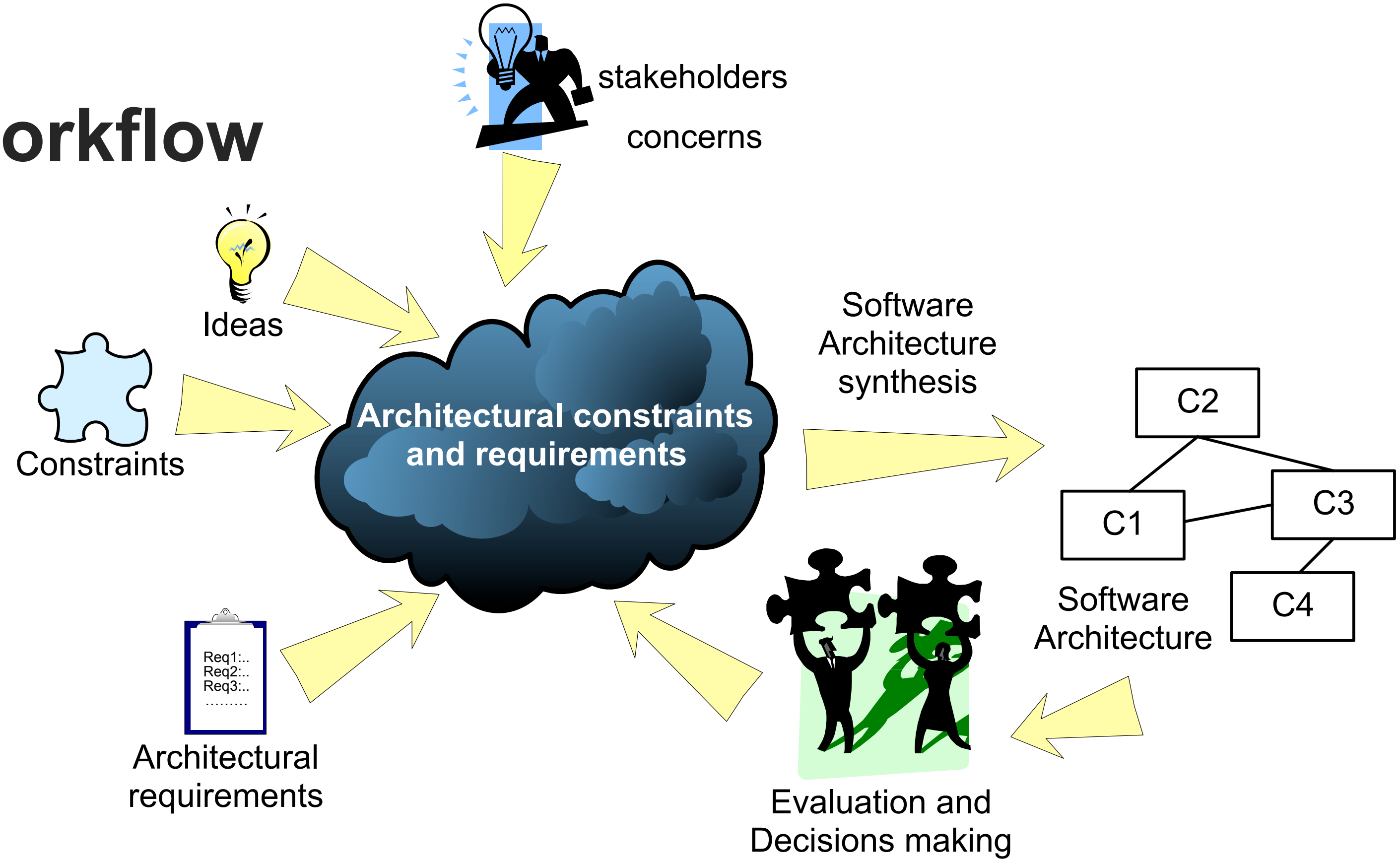
- **Every application has an architecture**
  - The architecture of a system can be characterized by the principal design decisions made during its development

- **Every application has at least one architect**
  - Perhaps not known by that title or recognized for what is done

- **Architecture is not a phase of development**

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. 2009 ***Software Architecture: Foundations, Theory, and Practice***. Wiley Publishing.

General workflow


stakeholders
concerns
Ideas
Constraints
Architectural constraints and requirements
Software Architecture synthesis
C2
C1
C3
C4
Software Architecture
Req1:..
Req2:..
Req3:..
....
Architectural requirements
Evaluation and Decisions making

Patrizio Pelliccione, Paola Inverardi and Henry Muccini, **CHARMY: A Framework for Designing and Verifying Architectural Specifications** (2009), in: IEEE Transactions on Software Engineering (TSE), 35:3(325 - 346)

Software architecture characteristics

- **Multitude of stakeholders**
  - Dealing with a broad variety of concerns and stakeholders, and has a multidisciplinary nature.
- **Separation of concerns**
  - Stakeholder concerns are addressed by modeling and describing the architecture from separate points of view associated with the various stakeholder concerns.
- **Quality-driven**
  - Architecture of a system is closely related to its quality attributes, such as fault-tolerance, backward compatibility, extensibility, reliability, maintainability, availability, security, usability, and other such –ilities.
- **Recurring styles**
  - Common terms for recurring solutions are architectural style, strategy or tactic, reference architecture and architectural patterns.
- **Conceptual integrity**
  - The architect assumes the role of “keeper of the vision”, making sure that changes to the systems are in line with the architecture, hence preserving conceptual integrity.

Software architecture benefits

- ***Basis for analysis of software systems' behavior before the system has been built.***
  - Substantial cost-saving and risk-mitigation.
- ***Basis for re-use of elements and decisions.***
  - Saving design costs and mitigating the risk of design mistakes.
- ***Support for early design decisions which have high impact on a system's development, deployment and maintenance life.***
  - Prevent schedule and budget overruns.
- ***Facilitate communication among stakeholders, contributing to a system that better fulfills their needs.***
  - Substantial cost-saving and risk-mitigation: communicate about design decisions before the system is implemented, when they are still relatively easy to adapt.

Agile architecture

1. A **system or software architecture** that is versatile, easy to evolve, to modify, flexible in a way, while still resilient to changes
2. An **agile way** to define an architecture, using an iterative lifecycle, allowing the architectural design to tactically evolve gradually, as the problem and the constraints are better understood

- The two are not the same
  - you can have a **non-agile development process** leading to a **flexible, adaptable architecture**, and *vice versa*,
  - an **agile process** may lead to a rather **rigid and inflexible architecture**.
  - One does not imply the other.
- In the best of worlds, we’d like to have an **agile process**, leading to a **flexible architecture**.

Taken from the Philippe Kruchten: http://philippe.kruchten.com/2013/12/11/agile-architecture/

Naïve thinking

- By being agile, an architecture will gradually emerge, out of bi-weekly refactorings.
- This belief was amplified by a rather poorly worded principle #11 in the agile manifesto[1], which states that:
  - *“The best architectures, requirements, and designs emerge from self-organizing teams.”*
- and cemented by profuse amount of repeated mantras like:
  - YAGNI (You Ain’t Gonna Need It) or
  - No BUFD (No Big Up-Front Design), or
  - “Defer decision to the last responsible moment”. (This principle is neither prescriptive, nor testable, as Séguin *et al.* showed in [2], so it is probably not a *principle*, but merely an observation or a wish.)

[1] Agile Alliance, ***Manifesto for Agile Software Development***, June 2001 http://agilemanifesto.org/.

[2] N. Séguin, G. Tremblay, and H. Bagane, **Agile Principles as Software Engineering Principles: An Analysis**, vol. 111, *Lecture Notes in Business Information Processing*, C. Wohlin, Ed. Berlin Heidelberg: Springer, 2012, pp. 1-15.

Taken from the Philippe Kruchten: http://philippe.kruchten.com/2013/12/11/agile-architecture/

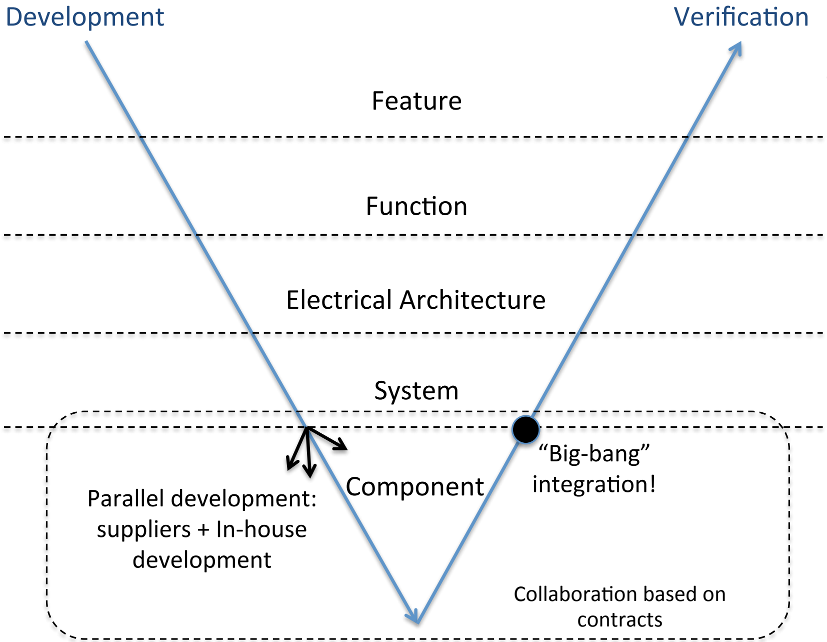
ARTWORK: MILLO, 2014, B.ART-ARTE IN BARRIERA, TURIN, ITALY

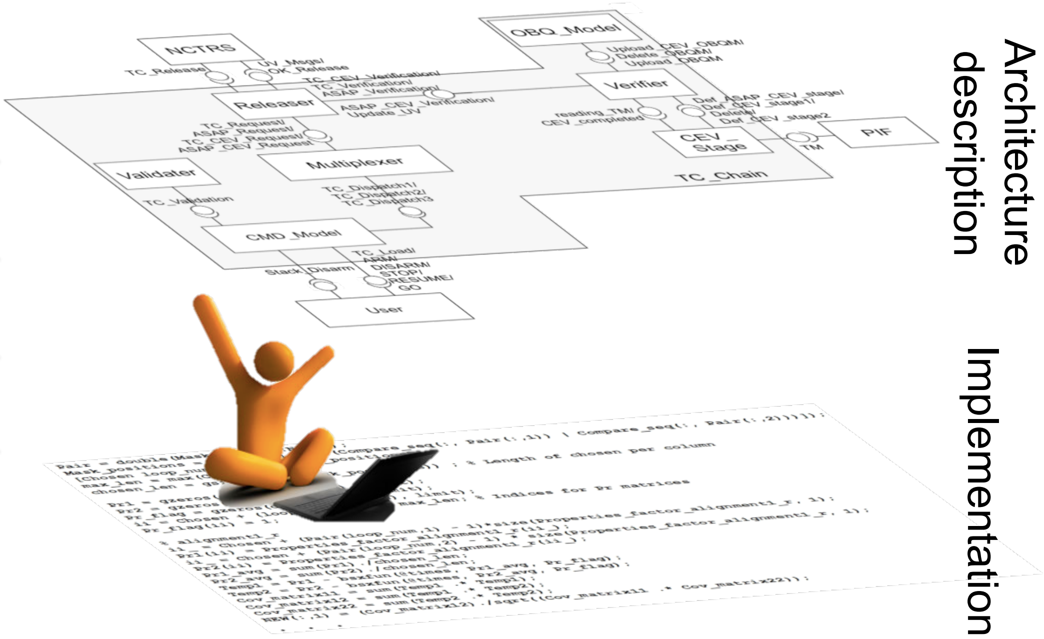
However…

- Key architectural choices **cannot be easily retrofitted** on an existing system by means of simple **refactoring**
- **Much of the architectural decisions have to be taken early**, although not all at once up front

Inspired by: http://philippe.kruchten.com/2013/12/11/agile-architecture/

Architecting in practice…

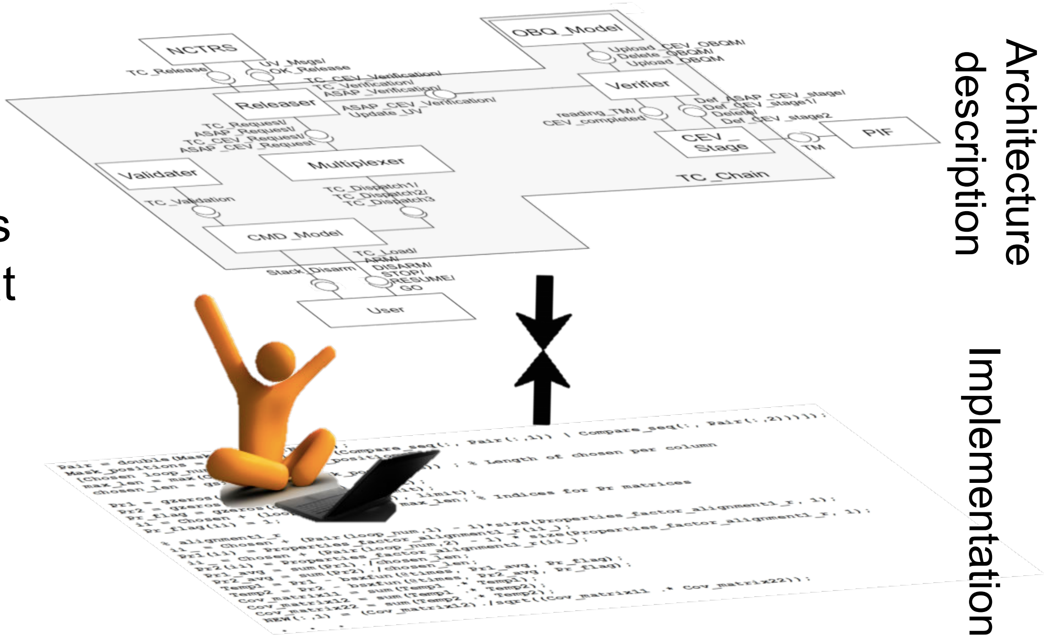




Architecting in practice…

Architecture degradation

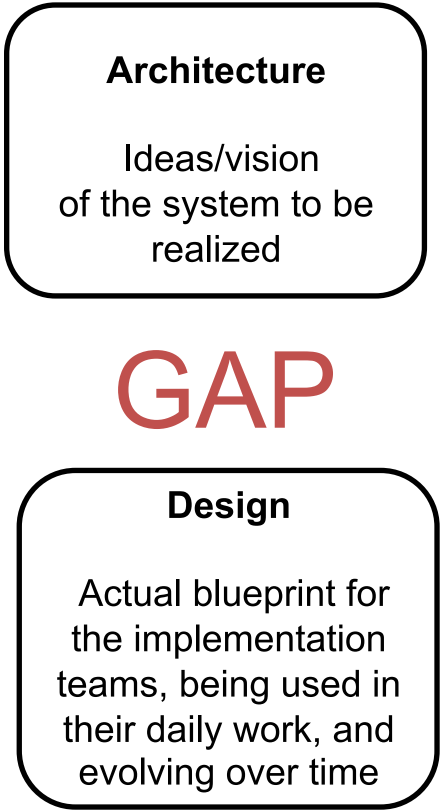
- **Architectural drift** – discrepancies that do not violate any decision that is documented in the as-intended architecture
- **Architectural erosion** – some decisions violate the as-intended architecture



Discrepancy between as-intended and as-implemented architectures !

Risk of architectural erosion

- The actual architecture of the car is not exactly the one conceived by the architects
  - The architecture is also emerging during development (bottom-up)
  - Some architectural decisions are made unconsciously
    - Which decisions have an impact on the architecture? – not easy
  - Some “actual” architects do not have the title of architect



Architecture

Ideas/vision of the system to be realized

GAP

Design

Actual blueprint for the implementation teams, being used in their daily work, and evolving over time

Limitations of the actual architecture description

State of Practice

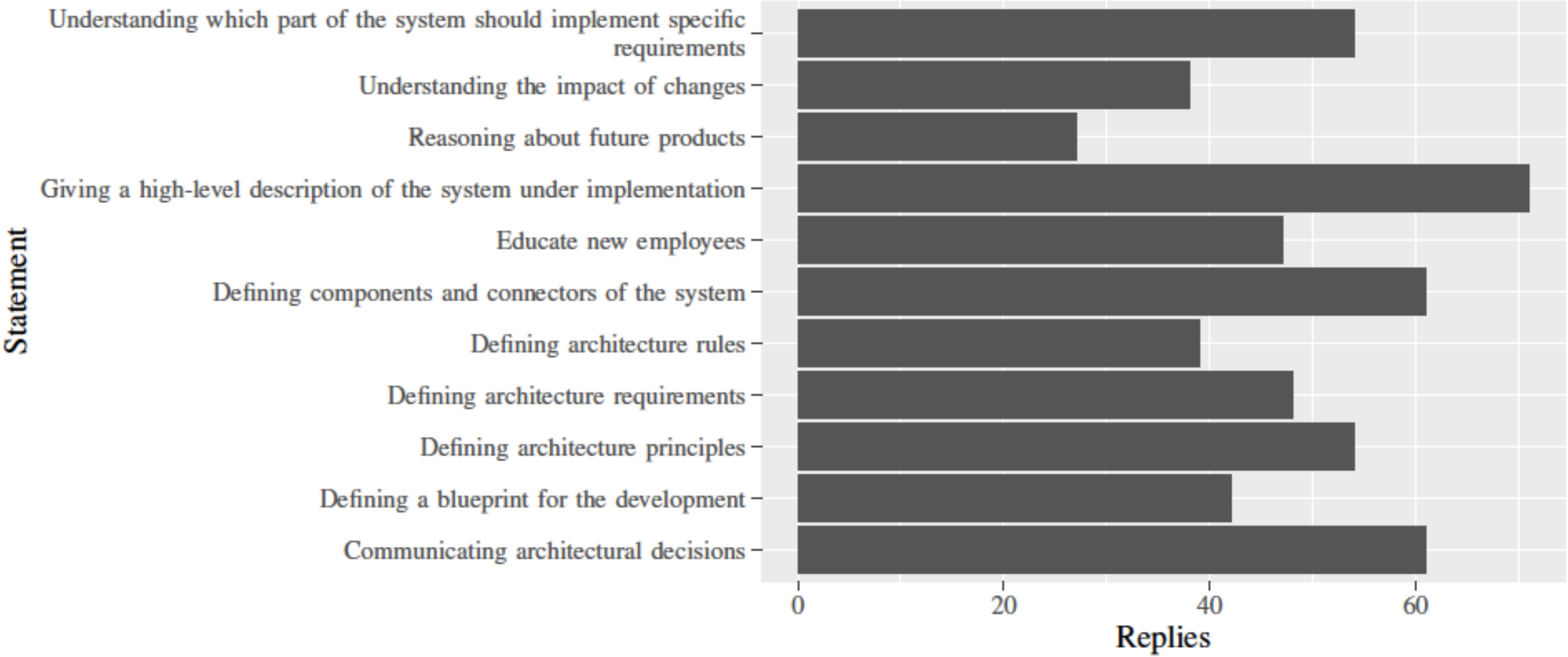
- Importance varies over time
- Easily becomes out of date
- Too many details
- Variability management
- Should better document the design decisions
- Should better document / make explicit the assumptions made
- Should be a living document connected with the other development phases
- Should handle different views and viewpoints of different stakeholders' concerns
- Present and Future mixed in the same document

U. Eliasson, R. Heldal, P. Pelliccione, J.Lantz (2015) **Architecting in the Automotive Domain: Descriptive vs Prescriptive Architecture** In: In Proceedings of 12th Working IEEE / IFIP Conference on Software Architecture (WICSA 2015), IEEE, Montreal, Canada.
R. Heldal, P. Pelliccione, U. Eliasson, J. Lantz, J. Derehag, J. Whittle, **Descriptive vs Prescriptive Models in Industry**, Models 2016, St-Malo, France, 2016

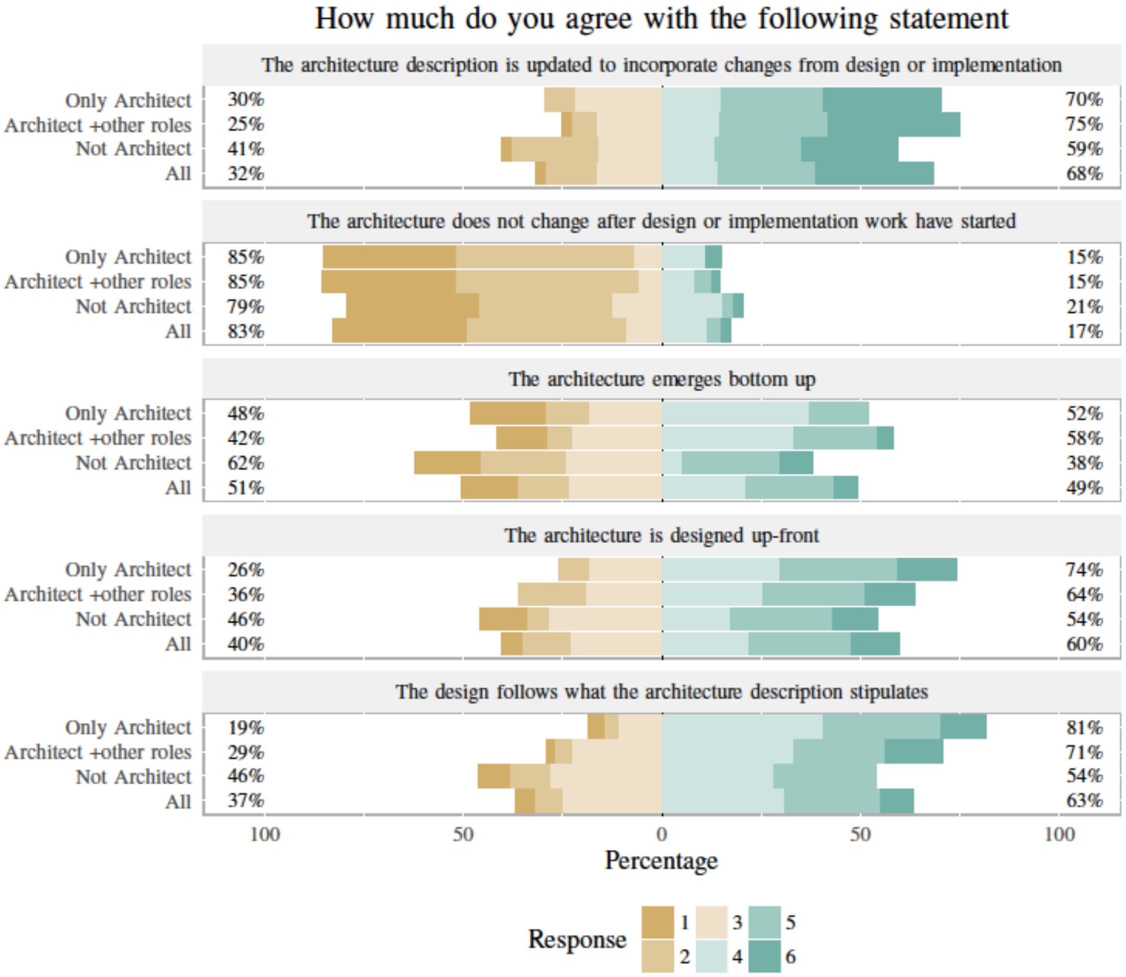
Recent work: Architecture Gap Survey

- Involved Volvo Cars (VCG), Volvo Group Truck Technology (VGTT), Ericsson, Jeppesen AB, plus many other companies around the world
- Research questions that we are investigating:
    - Is the architecture driving the development?
    - Is the architecture “emerging” from the development?
    - Is there any gap between what specified in the architecture and what is developed?
    - If so, what are the reasons for that and what are the consequences?
    - How could the architecture description be improved to be more useful during the development and maintenance phases?

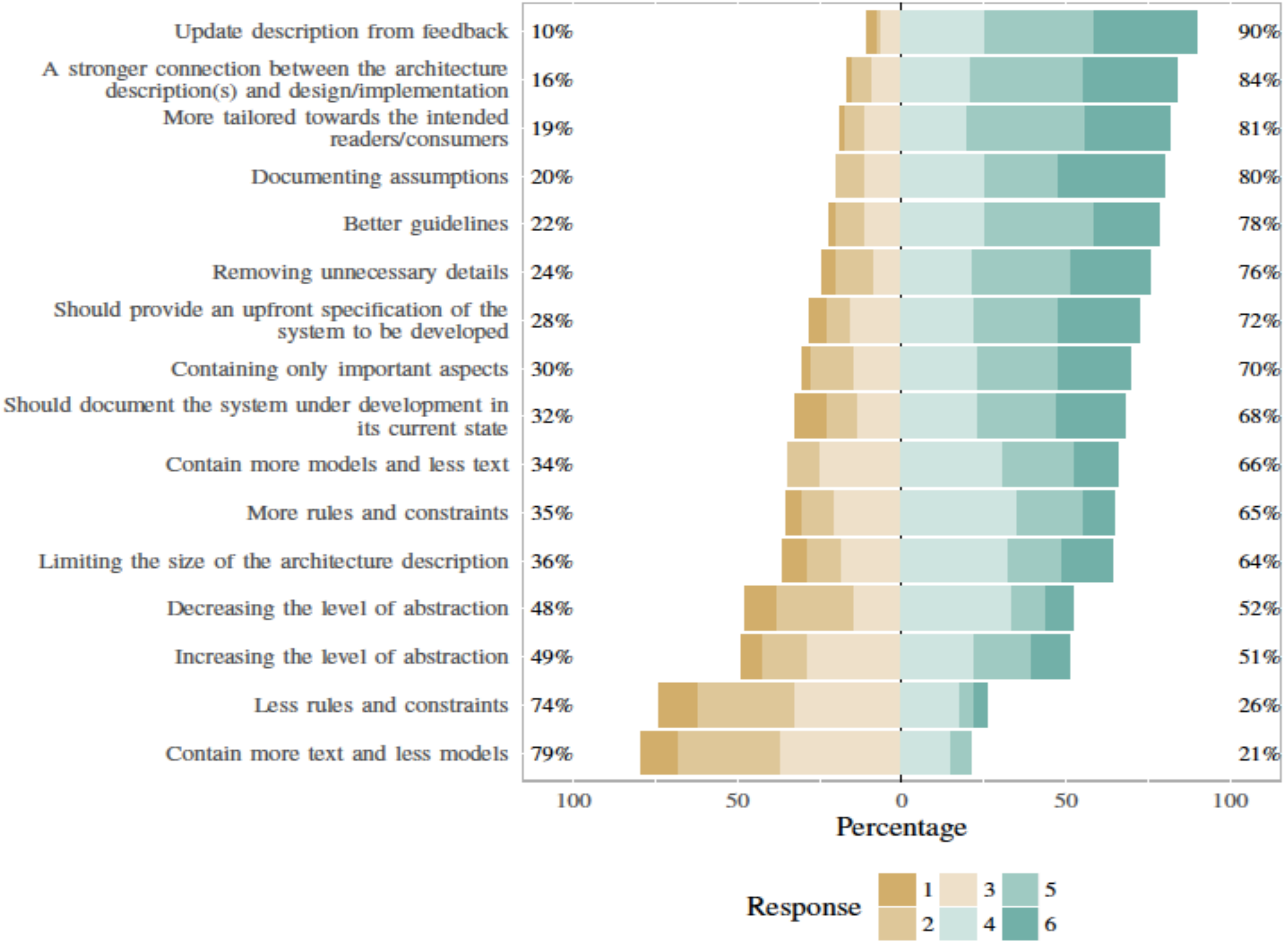
What is the purpose of an architecture description



Architecture driving the implementation or the other way round ?



How could the architecture description be improved?



Main findings of the study

Finding

- Typically, **more than one architecture description** (often at different levels of abstraction) exists for describing the architecture of a system

Implication

- Several reasons for representing architectures in architecture descriptions
- Need of investigating
  - how to ameliorate the creation and maintenance of architecture descriptions
  - languages and notations to describe architectures
  - how to maintain more than one architecture description for representing the architecture of complex systems

Main findings of the study

Finding

- Architecture descriptions serve **various purposes** and should be conceived for different types of stakeholders

Implication

- Need of multiple views and viewpoints
- Languages and notations to represent architecture descriptions should be conceived with the flexibility to satisfy various concerns of different stakeholders
- The different purposes might also be conflicting each other, and suitable tradeoff analysis should be put in place

Main findings of the study

Finding

- While it is important to have an upfront architecture and architecture description, the **architecture description should evolve** during the system development from input and feedback coming from stakeholders that are even different from architects

Implication

- How much information should be put in the upfront architecture description?
- Need to support “just in time” architecting, thus enabling stakeholders (even different from architects) to refine, add information, or provide feedback to the architecture description.

Main findings of the study

Finding

- Exist **inconsistencies** both among different architecture descriptions and between architecture descriptions and design/ implementation
- Some of the inconsistencies might have high impact

Implication

- This finding triggers the need of investigating causes and mechanisms to discover, avoid, and mitigate inconsistencies

Main findings of the study

Finding

- We identified some **discrepancy** between the architect team and other stakeholders

Implication

- There is the risk that within the same company will grow and will become established different cultures and beliefs.
- Innovative and more effective communication means are needed to enable communication among different stakeholders.

Different points of view

What the **design groups** think of the high-level architecture group

- High-level architects **lack an understanding** of the current situation and the **system under implementation**
- High-level architect group **focuses** too much on what might be good for the **future**, while neglecting a concrete vision of what is the best solution for the current situation

What the **architecture group** thinks of the working architecture group

- The design groups are very focused on everyday problems and then they **miss an overall picture**
- The design groups are too **focused on short-term solutions**: choosing the best solution in the short run might cause problems in the next future

“But sometimes you feel that the architecture-group thinks that we should change everything. While we [design group] are more focused on that we have to solve something to the project, and yes, what we have is maybe not the optimal solution but it is what we have.”

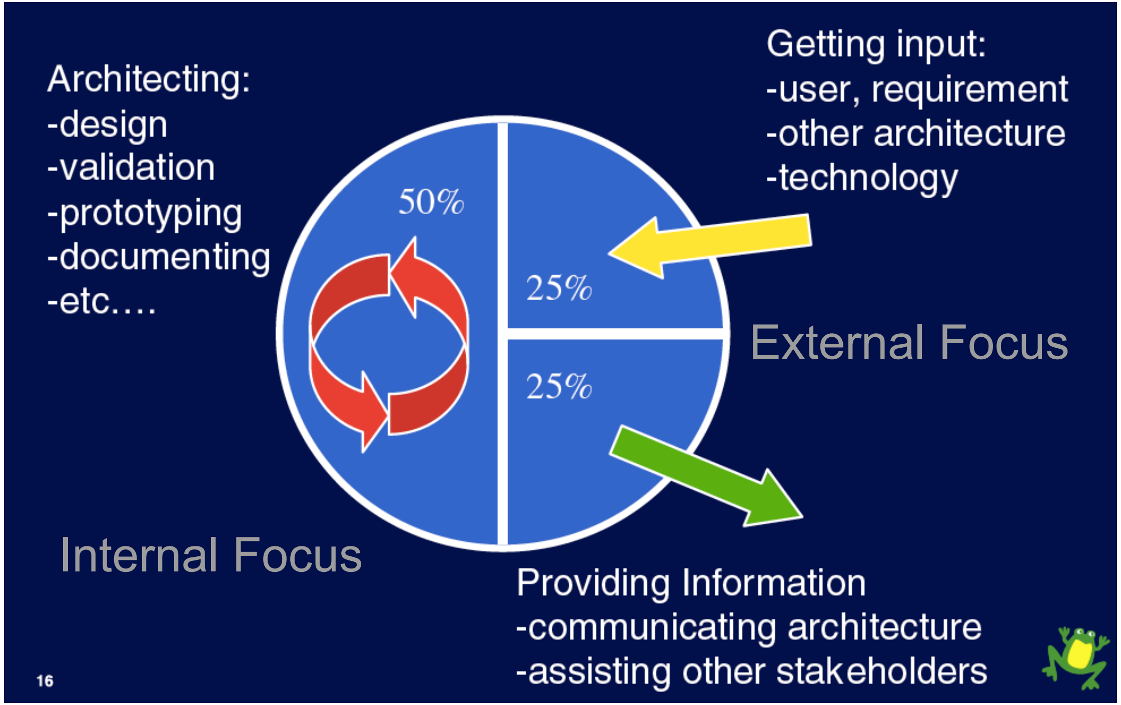
Ulf Eliasson, Rogardt Heldal, Patrizio Pelliccione, Jonn Lantz, “**Architecting in the Automotive Domain: Descriptive vs Prescriptive Architecture**”, WICSA 2015, 12th Working IEEE/IFIP Conf. on Software Architecture”, Montreal, Canada

Identified antipatterns

- **GoldPlating** – the architecture that has been created is a perfect architecture but it is describing the wrong system
- **Ivory tower** – the architect team is isolated from the other groups with few communication. They might experience rejection from developers
- **Architecture watch** – the group of architects is limited to a watching group. They provide recommendations without making any architectural decision.

P. Kruchten. **What do software architects really do?** Journal of Systems and Software, 81(12):2413 – 2416, 2008.

What do architect really do?



Architecting:
-design
-validation
-prototyping
-documenting
-etc.…

Getting input:
-user, requirement
-other architecture
-technology

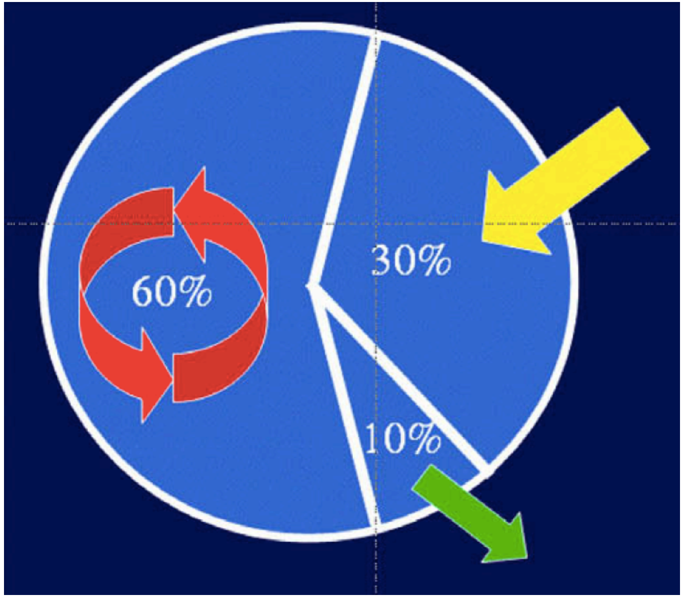
Providing Information
-communicating architecture
-assisting other stakeholders

P. Kruchten. **What do software architects really do?** Journal of Systems and Software, 81(12):2413 – 2416, 2008.

What do architect really do?

The [60:30:10] antipattern – goldplating

GoldPlating – the architecture that has been created is a perfect architecture but it is describing the wrong system

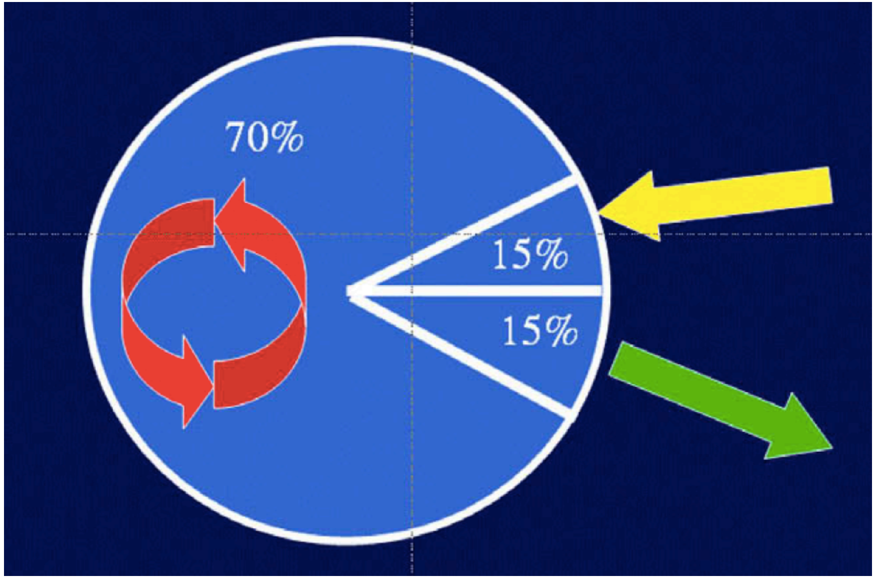


P. Kruchten. **What do software architects really do?** Journal of Systems and Software, 81(12):2413 – 2416, 2008.

What do architect really do?

The [70:15:15] antipattern – ivory tower

Ivory tower – the architect team is isolated from the other groups with few communication. They might experience rejection from developers



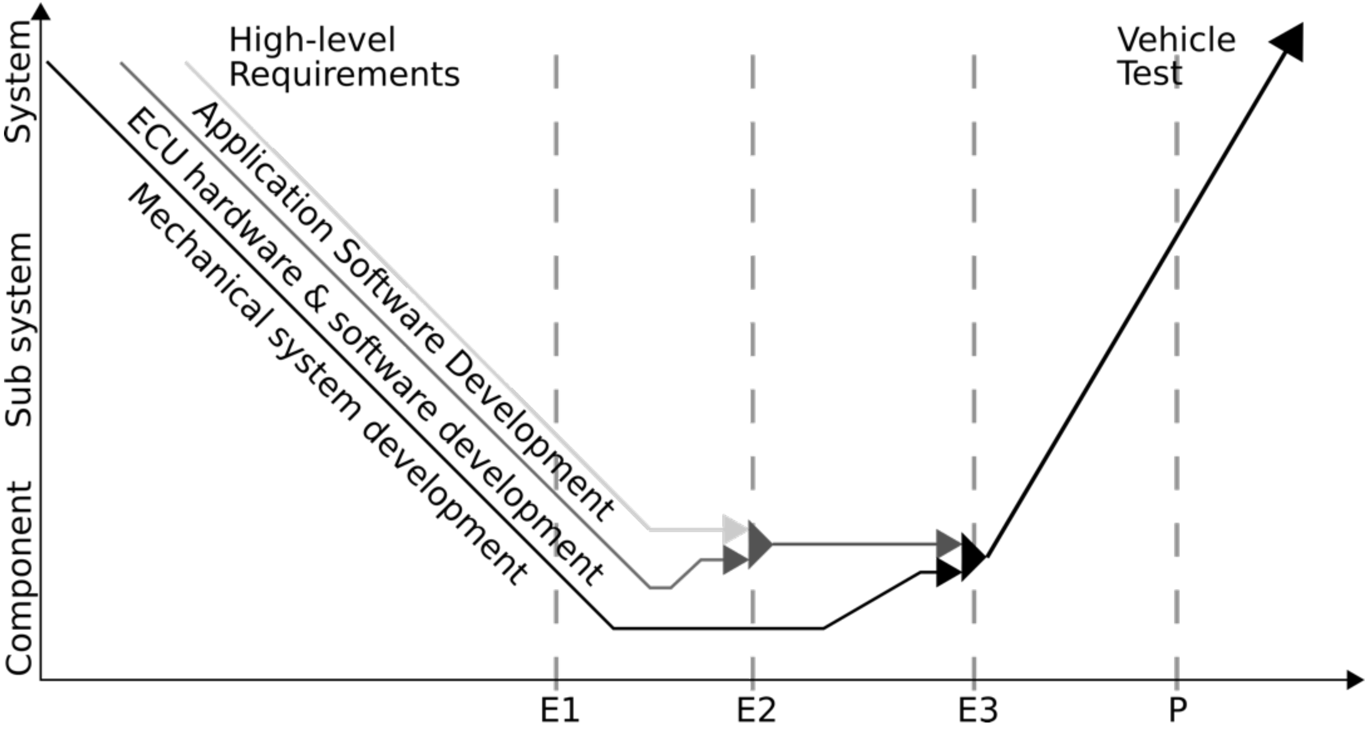
P. Kruchten. **What do software architects really do?** Journal of Systems and Software, 81(12):2413 – 2416, 2008.

NGEA
Next Generation Electrical Architecture

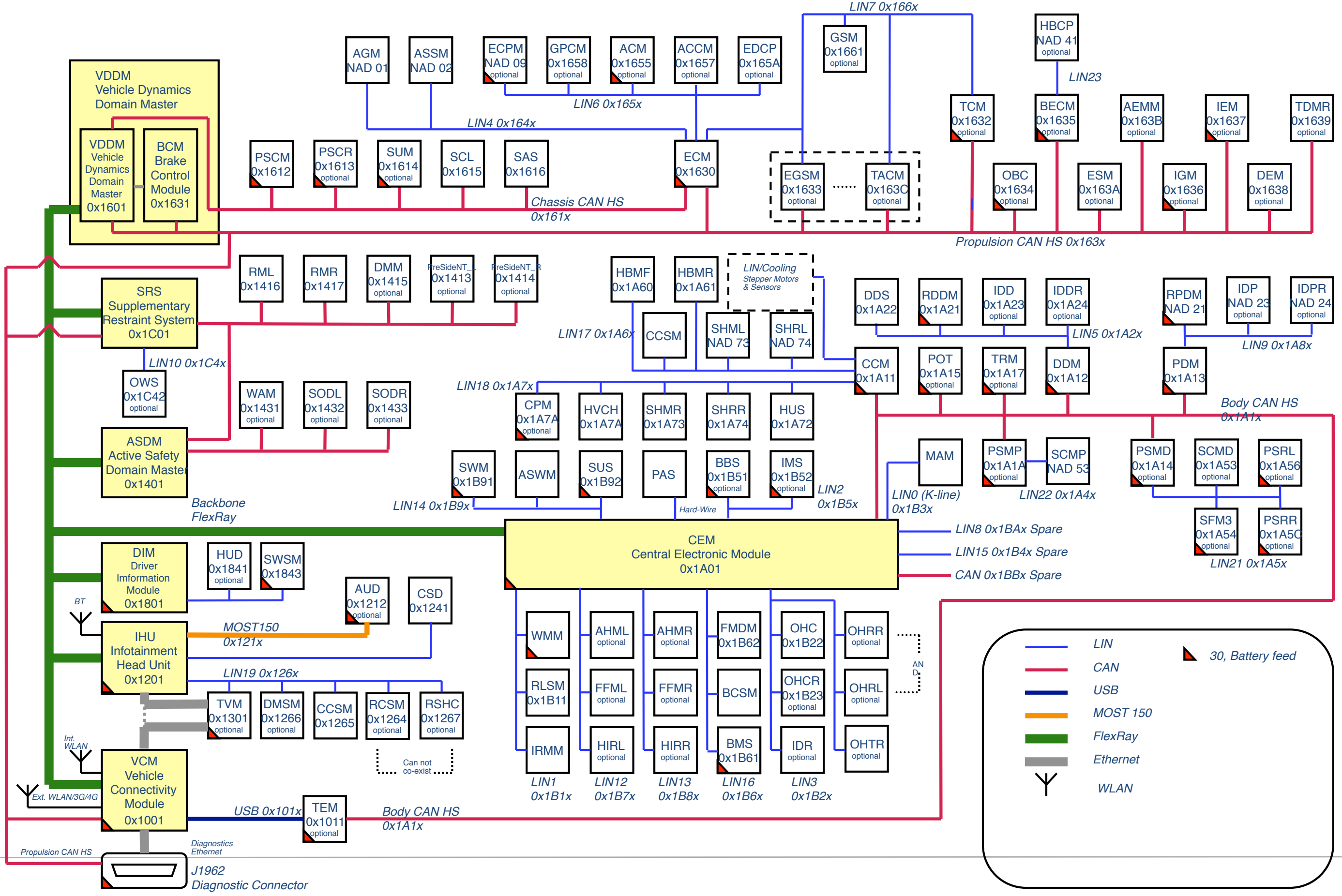
- How to reduce the time to market?
- How can a system respond quicker to changes in the market?
- How can we introduce CI&D practices in the automotive domain?

Just some of the research questions of the project…

Software, Hardware, and Mechanics


System
Sub system
Component
High-level
Requirements
Application Software Development
ECU hardware & software development
Mechanical system development
Vehicle
Test
E1
E2
E3
P

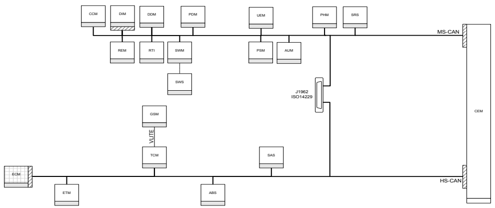
How many Electronic Control Units (ECUs) in a car?



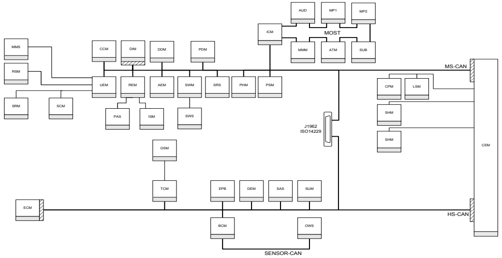
Kent Niesel http://slidegur.com/doc/173817/swc-reqs-test.v001

Volvo: 1998 – 2013 - ECU Growth

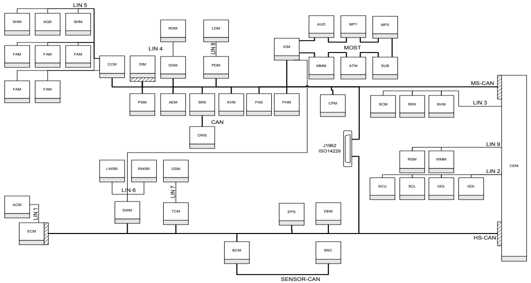
S80 1998 (19)



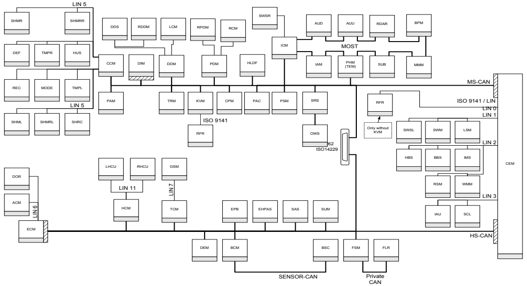
S/V 40 2002 (38)



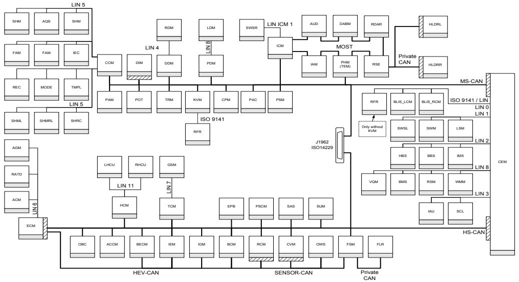
V40 2003 (49)



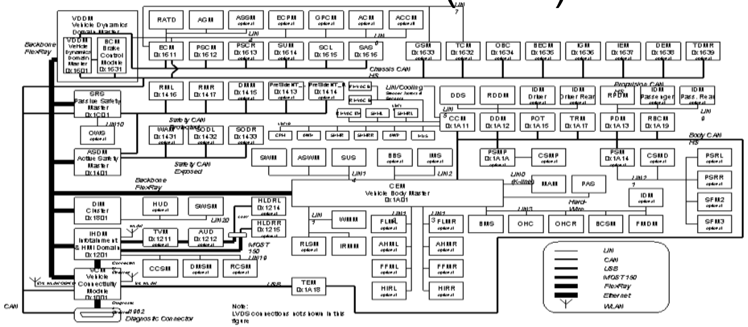
S80 2006 (68)



V60 PHEV 2012 (78)



XC90 2015 (>100)



Taken from: http://www.vinnova.se/PageFiles/751327324/Keynote%20Martin%20Nilsson%20presentation.pdf

A car is a complex system

Safety

Security

Qualities

Energy

Cost

NHV – Noise, Vibration, Harshness

Weight

Variability

Autonomous vehicle

Ecosystem & Transparency

Car in a SoS

CI&D

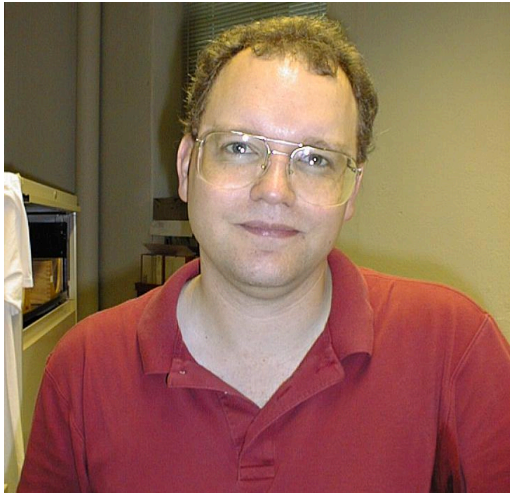
Technical architecture

Logical architecture

Functional architecture

In 10 years, about 10,000,000 cars have been recalled due to software-related problems

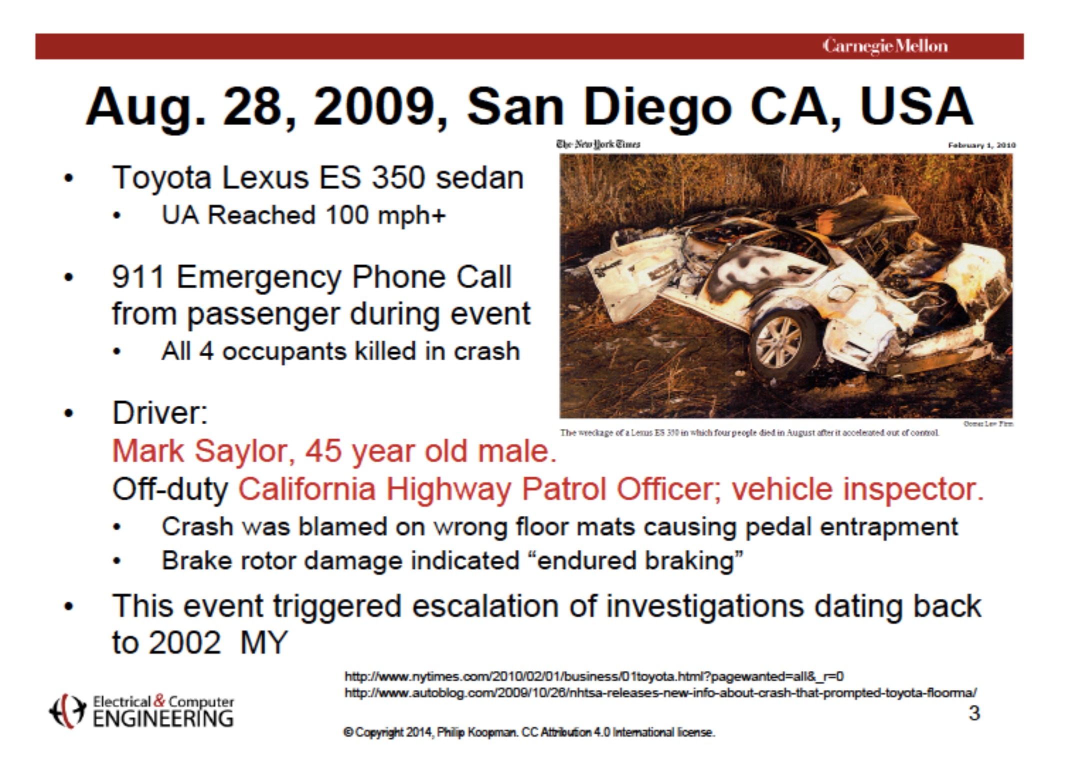
Infamous case: Toyota unintended acceleration



Prof. Philip Koopman has served as a Plaintiff expert witness on numerous cases in Toyota Unintended Acceleration litigation, and testified in the 2013 Bookout trial. Dr. Koopman is a member of the ECE faculty at Carnegie Mellon University, where he has worked in the broad areas of wearable computers, software robustness, embedded networking, dependable embedded computer systems, and autonomous vehicle safety. …

http://betterembsw.blogspot.hu/2014/09/a-case-study-of-toyota-unintended.html

Infamous case: Toyota unintended acceleration



Carnegie Mellon

Aug. 28, 2009, San Diego CA, USA

- Toyota Lexus ES 350 sedan
  - UA Reached 100 mph+
- 911 Emergency Phone Call from passenger during event
  - All 4 occupants killed in crash
- Driver:
  Mark Saylor, 45 year old male.
  Off-duty California Highway Patrol Officer; vehicle inspector.
  - Crash was blamed on wrong floor mats causing pedal entrapment
  - Brake rotor damage indicated “endured braking”
- This event triggered escalation of investigations dating back to 2002 MY

The New York Times February 1, 2010

Gomez Law Firm

The wreckage of a Lexus ES 350 in which four people died in August after it accelerated out of control.

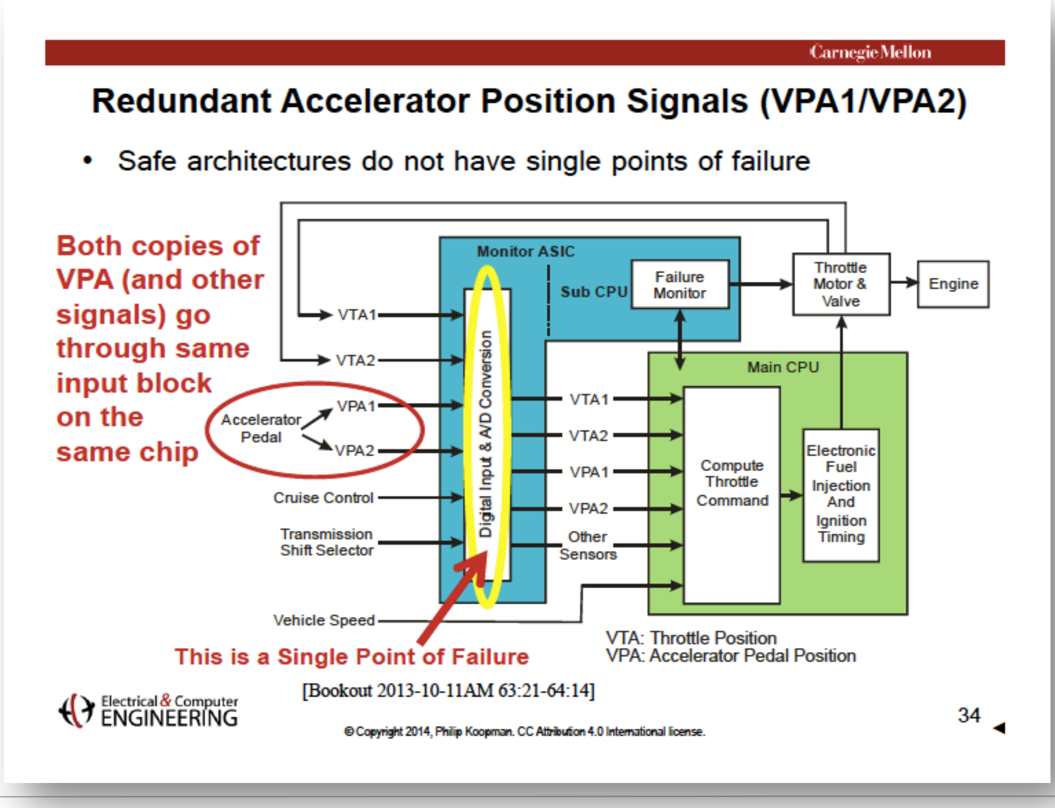
Electrical & Computer ENGINEERING

http://www.nytimes.com/2010/02/01/business/01toyota.html?pagewanted=all&_r=0
http://www.autoblog.com/2009/10/26/nhtsa-releases-new-info-about-crash-that-prompted-toyota-floorma/

3

http://betterembsw.blogspot.hu/2014/09/a-case-study-of-toyota-unintended.html

Infamous case: Toyota unintended acceleration



Redundant Accelerator Position Signals (VPA1/VPA2)

- Safe architectures do not have single points of failure

Both copies of VPA (and other signals) go through same input block on the same chip

This is a Single Point of Failure

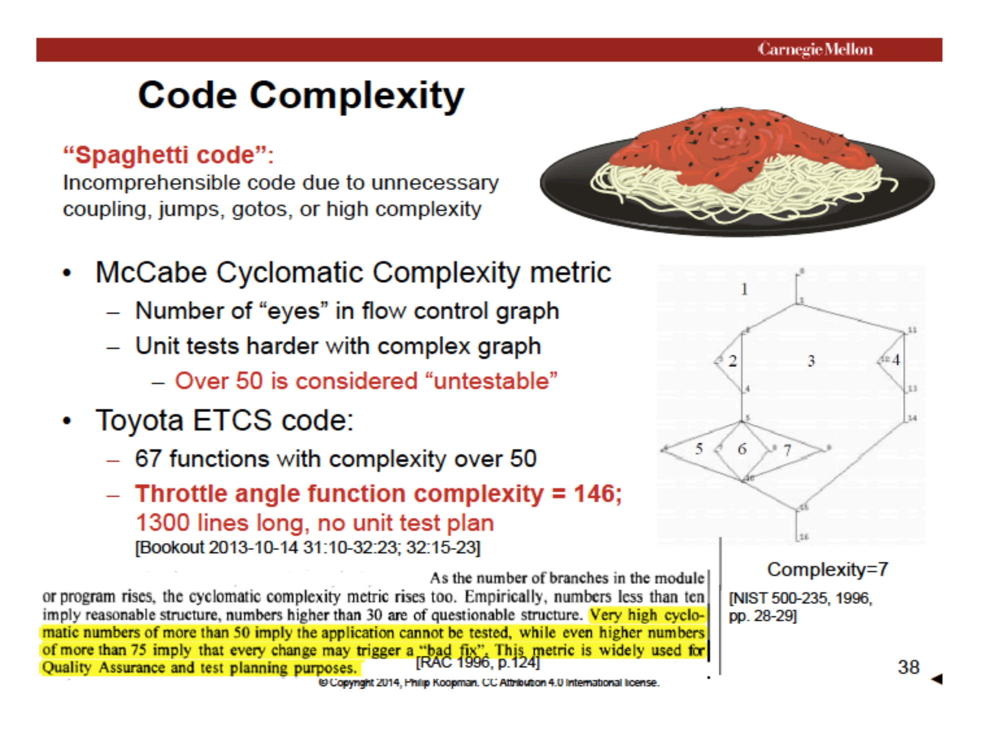
VTA: Throttle Position
VPA: Accelerator Pedal Position

[Bookout 2013-10-11AM 63:21-64:14]

Electrical & Computer ENGINEERING

http://betterembsw.blogspot.hu/2014/09/a-case-study-of-toyota-unintended.html

Infamous case: Toyota unintended acceleration



Code Complexity

"Spaghetti code":
Incomprehensible code due to unnecessary coupling, jumps, gotos, or high complexity

- McCabe Cyclomatic Complexity metric
  - Number of "eyes" in flow control graph
  - Unit tests harder with complex graph
    - Over 50 is considered "untestable"
- Toyota ETCS code:
  - 67 functions with complexity over 50
  - **Throttle angle function complexity = 146;** 1300 lines long, no unit test plan [Bookout 2013-10-14 31:10-32:23; 32:15-23]

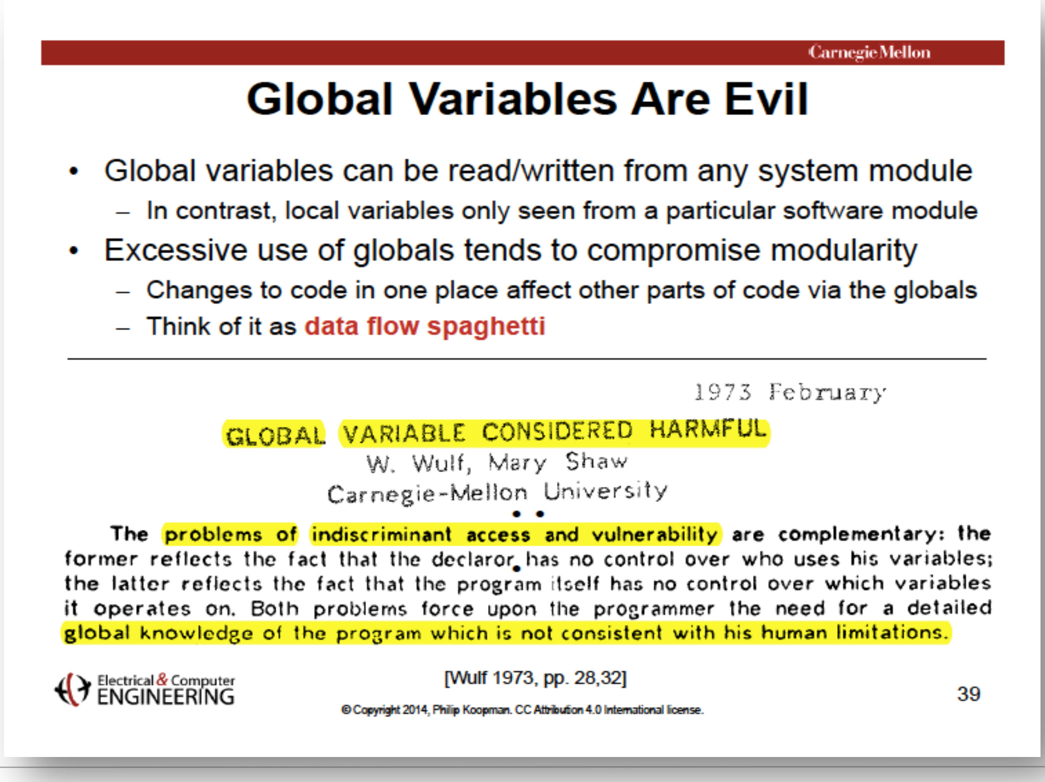
Complexity=7

[NIST 500-235, 1996, pp. 28-29]

As the number of branches in the module or program rises, the cyclomatic complexity metric rises too. Empirically, numbers less than ten imply reasonable structure, numbers higher than 30 are of questionable structure. Very high cyclomatic numbers of more than 50 imply the application cannot be tested, while even higher numbers of more than 75 imply that every change may trigger a "bad fix". This metric is widely used for Quality Assurance and test planning purposes. [RAC 1996, p.124]

http://betterembsw.blogspot.hu/2014/09/a-case-study-of-toyota-unintended.html

Infamous case: Toyota unintended acceleration



Carnegie Mellon

Global Variables Are Evil

- Global variables can be read/written from any system module
  - In contrast, local variables only seen from a particular software module
- Excessive use of globals tends to compromise modularity
  - Changes to code in one place affect other parts of code via the globals
  - Think of it as **data flow spaghetti**

1973 February

GLOBAL VARIABLE CONSIDERED HARMFUL

W. Wulf, Mary Shaw

Carnegie-Mellon University

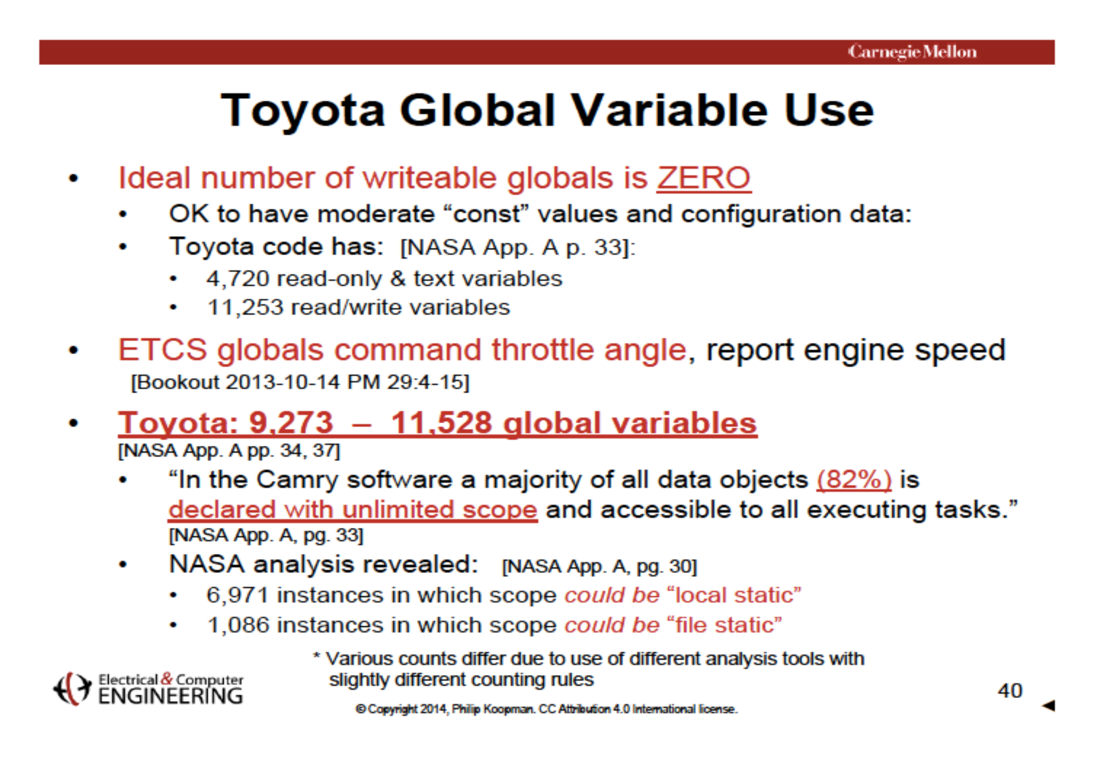
The problems of indiscriminant access and vulnerability are complementary: the former reflects the fact that the declaror has no control over who uses his variables; the latter reflects the fact that the program itself has no control over which variables it operates on. Both problems force upon the programmer the need for a detailed global knowledge of the program which is not consistent with his human limitations.

[Wulf 1973, pp. 28,32]

Electrical & Computer ENGINEERING

http://betterembsw.blogspot.hu/2014/09/a-case-study-of-toyota-unintended.html

Infamous case: Toyota unintended acceleration



Toyota Global Variable Use

- Ideal number of writeable globals is ZERO
  - OK to have moderate "const" values and configuration data:
  - Toyota code has: [NASA App. A p. 33]:
    - 4,720 read-only & text variables
    - 11,253 read/write variables
- ETCS globals command throttle angle, report engine speed [Bookout 2013-10-14 PM 29:4-15]
- Toyota: 9,273 – 11,528 global variables [NASA App. A pp. 34, 37]
  - "In the Camry software a majority of all data objects (82%) is declared with unlimited scope and accessible to all executing tasks." [NASA App. A, pg. 33]
  - NASA analysis revealed: [NASA App. A, pg. 30]
    - 6,971 instances in which scope *could be* "local static"
    - 1,086 instances in which scope *could be* "file static"

* Various counts differ due to use of different analysis tools with slightly different counting rules

Electrical & Computer ENGINEERING

http://betterembsw.blogspot.hu/2014/09/a-case-study-of-toyota-unintended.html

How to provide evidence that all system safety objectives are satisfied?

Drive Towards Zero

Vision : To develop cars that don’t crash.

Zero killed or badly injured in a Volvo car 2020

Johan Konnberg - Volvo Car Electrification Strategy

www.fkg.se/wp/wp-content/uploads/2013/05/Volvo-Cars.ppt

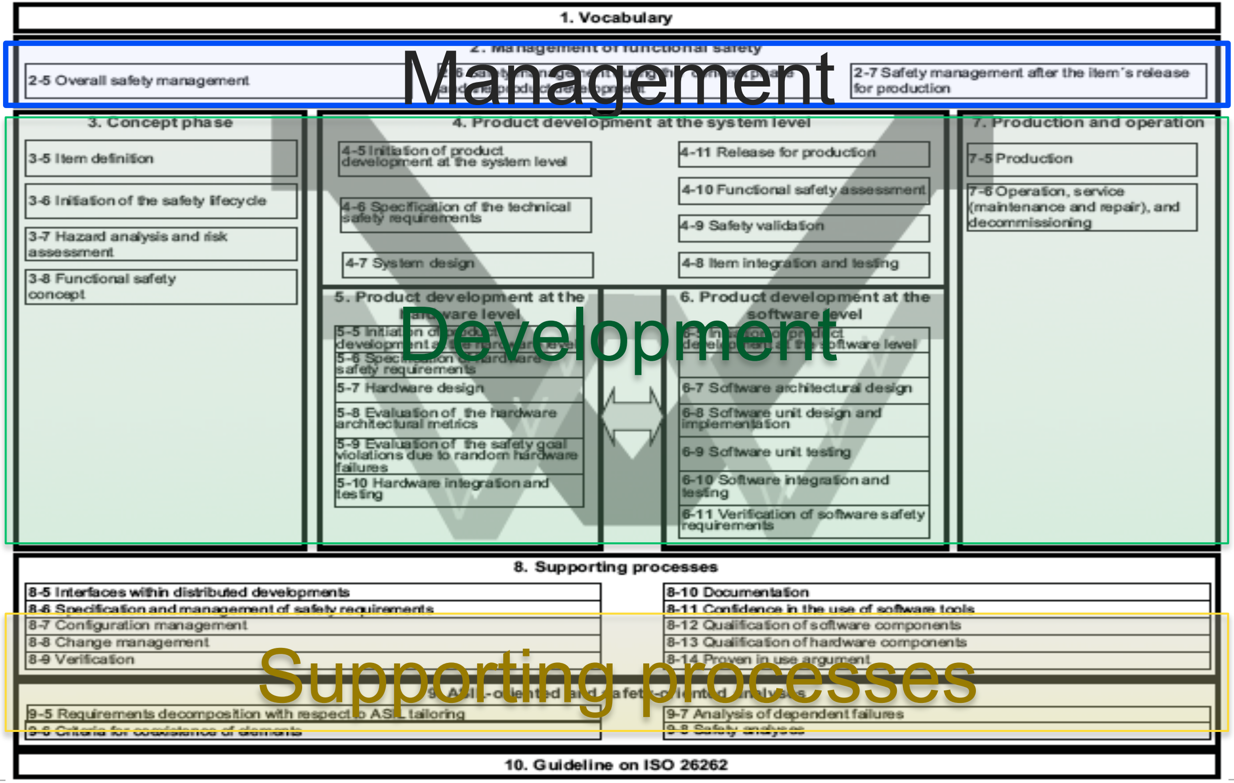
Focus on product or on the development process?

Question: What is a quality process?

Is it a process which …

- … leads to quality software?
- … is planned and controlled?
- … is predictable?
- … contains all activities necessary to deliver a quality product?
- … leads to the minimal effort possible for producing the product?
- … will ensure and maintain quality during the system life-time and evolution?

International Standard – ISO 26262



1. Vocabulary

2. Management of functional safety

- 2-5 Overall safety management
- 2-6 Safety management during the concept phase and the product development
- 2-7 Safety management after the item´s release for production

Management

3. Concept phase

- 3-5 Item definition
- 3-6 Initiation of the safety lifecycle
- 3-7 Hazard analysis and risk assessment
- 3-8 Functional safety concept

4. Product development at the system level

- 4-5 Initiation of product development at the system level
- 4-6 Specification of the technical safety requirements
- 4-7 System design
- 4-8 Item integration and testing
- 4-9 Safety validation
- 4-10 Functional safety assessment
- 4-11 Release for production

5. Product development at the hardware level

- 5-5 Initiation of product development at the hardware level
- 5-6 Specification of hardware safety requirements
- 5-7 Hardware design
- 5-8 Evaluation of the hardware architectural metrics
- 5-9 Evaluation of the safety goal violations due to random hardware failures
- 5-10 Hardware integration and testing

6. Product development at the software level

- 6-5 Initiation of product development at the software level
- 6-7 Software architectural design
- 6-8 Software unit design and implementation
- 6-9 Software unit testing
- 6-10 Software integration and testing
- 6-11 Verification of software safety requirements

7. Production and operation

- 7-5 Production
- 7-6 Operation, service (maintenance and repair), and decommissioning

Development

8. Supporting processes

- 8-5 Interfaces within distributed developments
- 8-6 Specification and management of safety requirements
- 8-7 Configuration management
- 8-8 Change management
- 8-9 Verification
- 8-10 Documentation
- 8-11 Confidence in the use of software tools
- 8-12 Qualification of software components
- 8-13 Qualification of hardware components
- 8-14 Proven in use argument

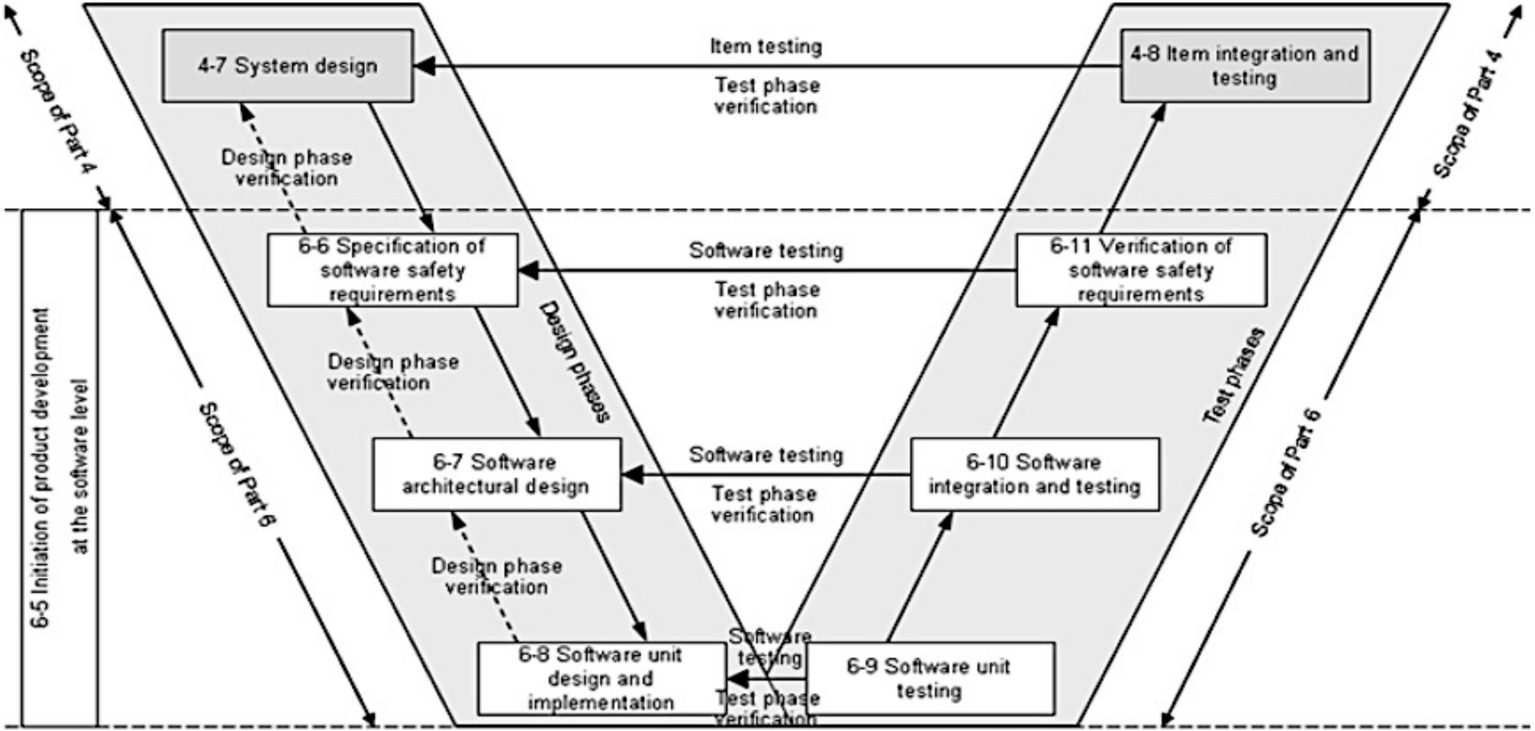
9. ASIL-oriented and safety-oriented analyses

- 9-5 Requirements decomposition with respect to ASIL tailoring
- 9-6 Criteria for coexistence of elements
- 9-7 Analysis of dependent failures
- 9-8 Safety analyses

Supporting processes

10. Guideline on ISO 26262

Product Development Software Level


Scope of Part 4
Scope of Part 4
4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Design phase verification
6-5 Initiation of product development at the software level
Scope of Part 6
6-6 Specification of software safety requirements
Software testing
Test phase verification
6-11 Verification of software safety requirements
Design phase verification
Design phases
Test phases
Scope of Part 6
6-7 Software architectural design
Software testing
Test phase verification
6-10 Software integration and testing
Design phase verification
6-8 Software unit design and implementation
Software testing
Test phase verification
6-9 Software unit testing

ISO 26262: If you did it well…

You are Able to Show:

- Completeness:
  - Everything accounted for
  - Requirements under Control
  - Everything tested – pass
  - Used the toolsets
- Traceability:
  - Structured Process Model
  - Documents linked
  - Evidence for Everything
  - Understandable for external
- Consistency
  - This is visible for external auditor even when project members have left
- Documentation:
  - All activities planned
  - Execution documented
  - Inspected - Archived
  - For a life-time (15year?)

Slide taken from “ISO 26262 Introduction” Singapore, 17 October 2012, Koen Leekens

ISO 26262: If you did it well…

You are Able to Show:

- Completeness:
  - Everything accounted for
  - Requ
  - Every
  - Used
- Traceabil
  - Structured Process Model
  - Documents linked
  - Evidence for Everything
  - Understandable for external

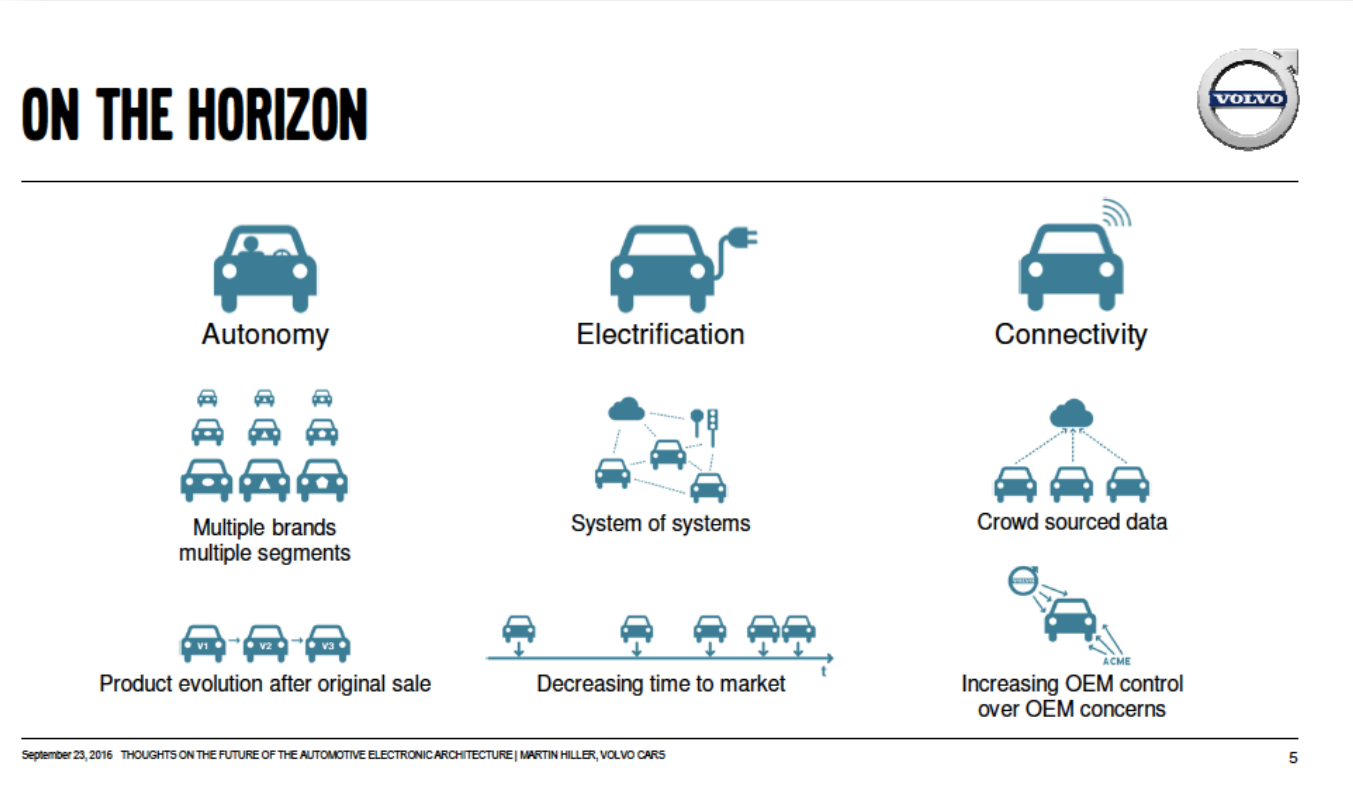
- Consistency
  - This is visible for external
    roject
  - All activities planned
  - Execution documented
  - Inspected - Archived
  - For a life-time (15year?)

**A clear,
comprehensive and defensible argument
that a system is acceptably safe to operate
in a particular context**

(Tim Kelly / Rob Weawer University of York)

Slide taken from "ISO 26262 Introduction" Singapore, 17 October 2012, Koen Leekens

A car is a complex system



ON THE HORIZON

Autonomy

Electrification

Connectivity

Multiple brands
multiple segments

System of systems

Crowd sourced data

Product evolution after original sale

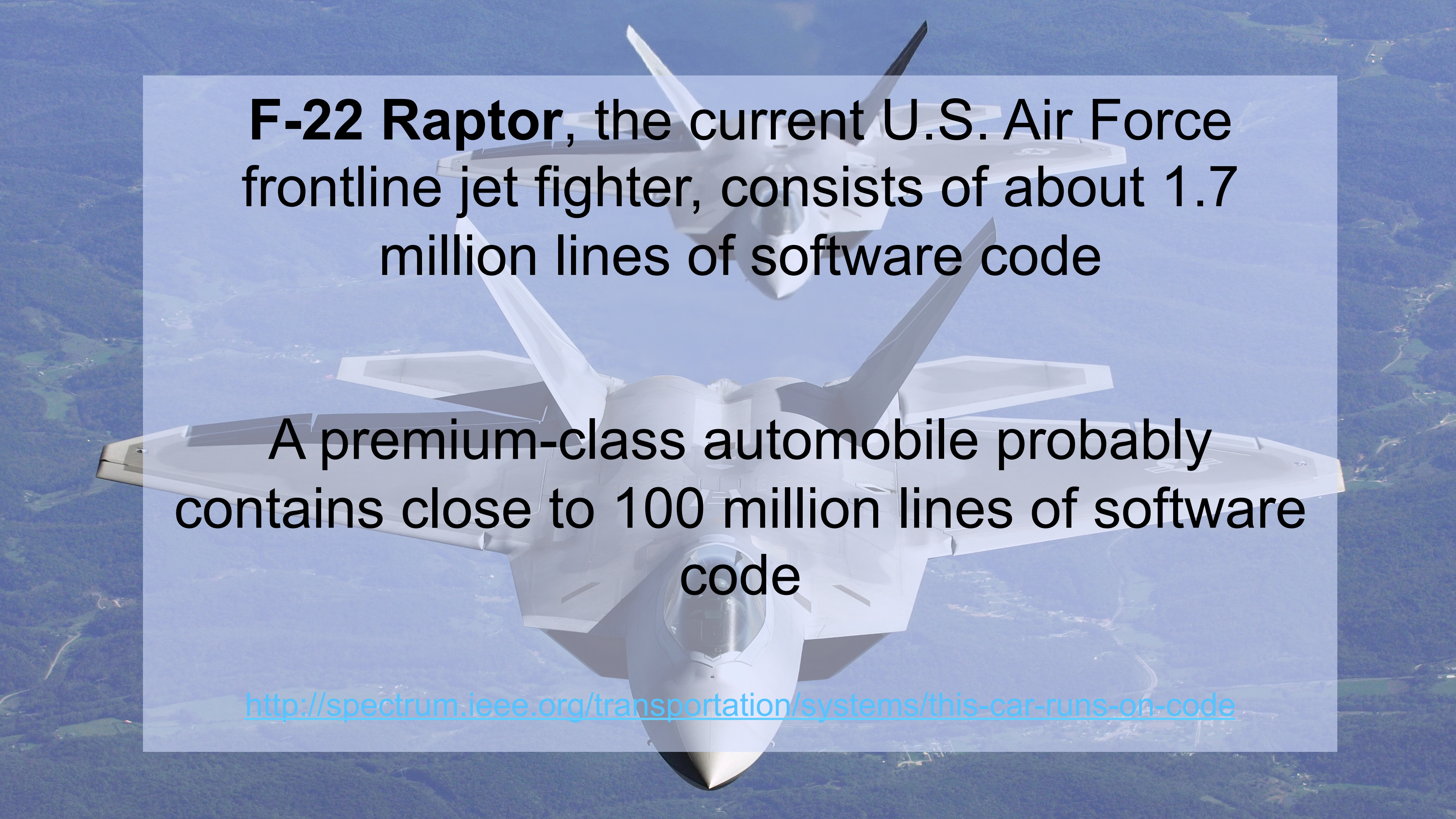
Decreasing time to market

Increasing OEM control
over OEM concerns

September 23, 2016 THOUGHTS ON THE FUTURE OF THE AUTOMOTIVE ELECTRONIC ARCHITECTURE | MARTIN HILLER, VOLVO CARS 5

VOLVO
V1
V2
V3
t
ACME

Thanks to Martin Hiller, Fuse meeting - September 23, 2016

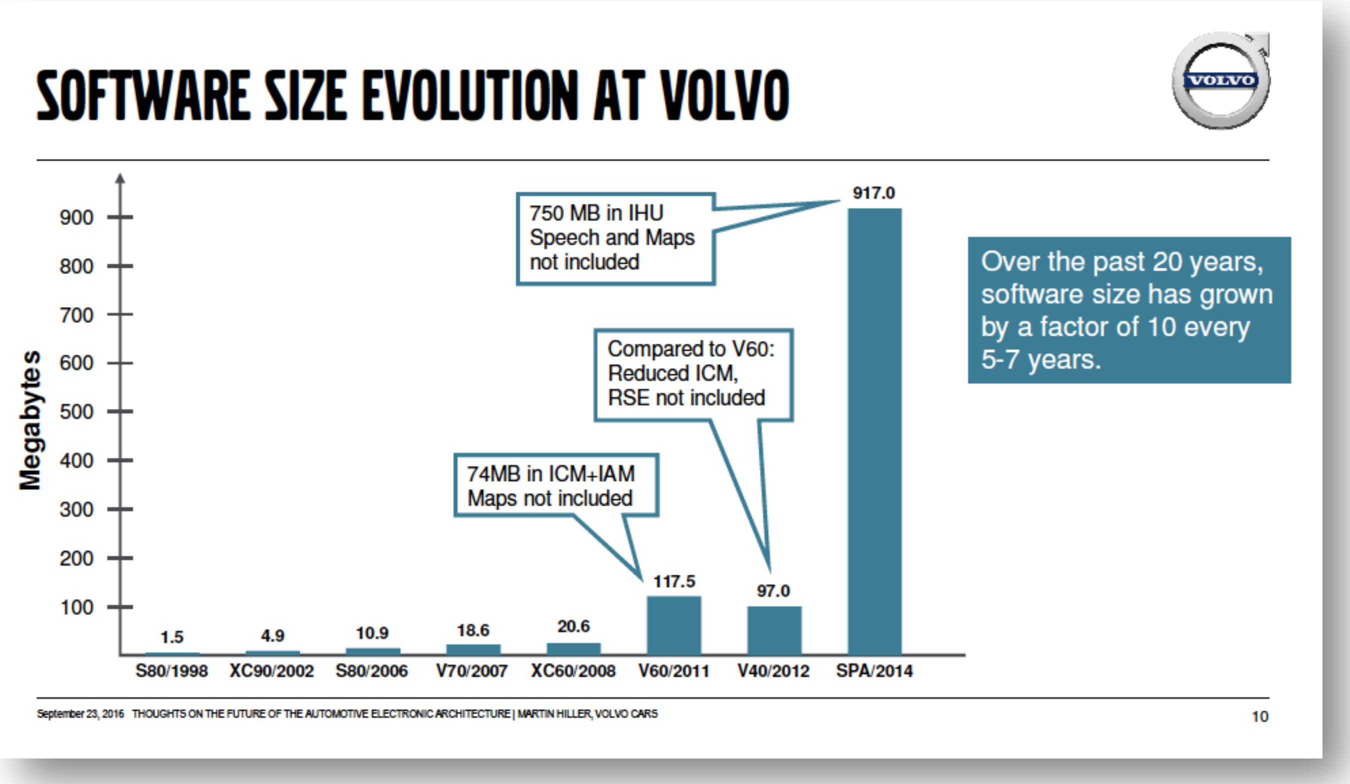


F-22 Raptor, the current U.S. Air Force frontline jet fighter, consists of about 1.7 million lines of software code

A premium-class automobile probably contains close to 100 million lines of software code

http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

A car is a complex system



SOFTWARE SIZE EVOLUTION AT VOLVO

750 MB in IHU
Speech and Maps
not included

Compared to V60:
Reduced ICM,
RSE not included

74MB in ICM+IAM
Maps not included

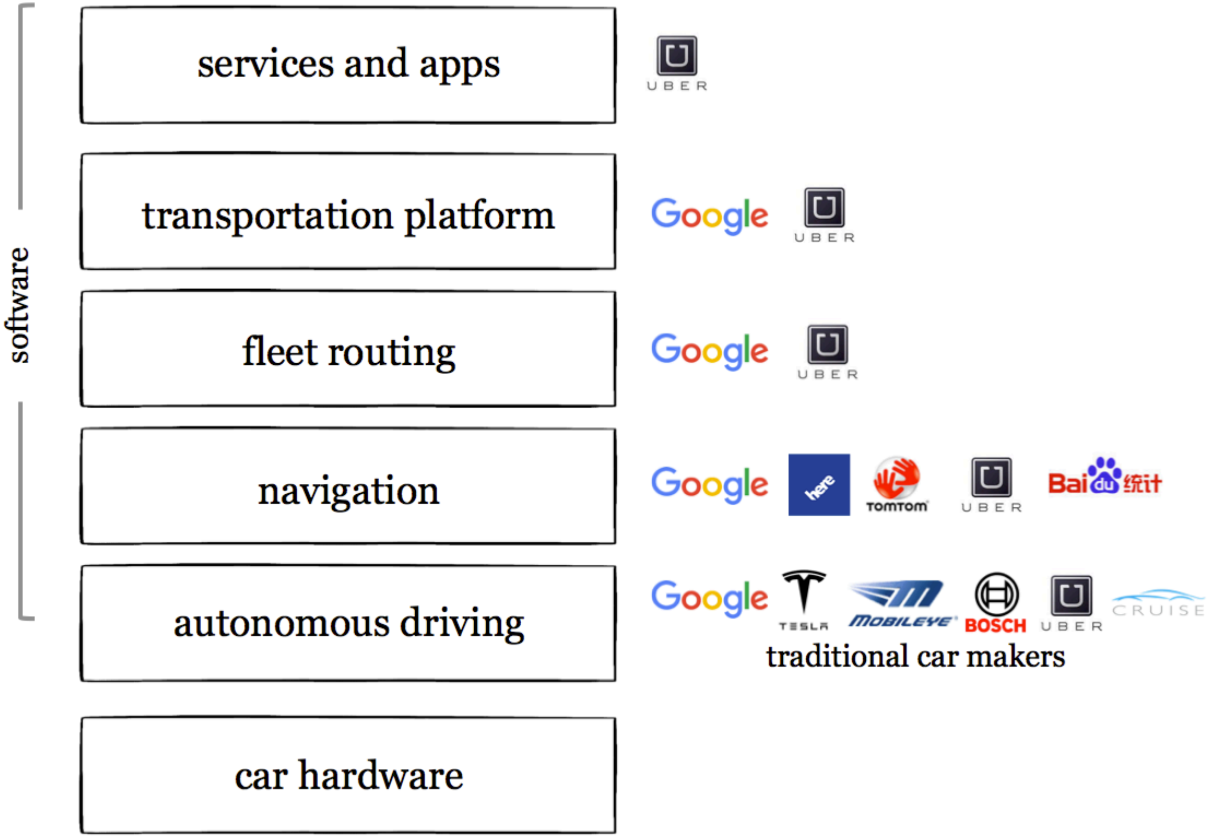
Over the past 20 years, software size has grown by a factor of 10 every 5-7 years.

September 23, 2016 THOUGHTS ON THE FUTURE OF THE AUTOMOTIVE ELECTRONIC ARCHITECTURE | MARTIN HILLER, VOLVO CARS

10

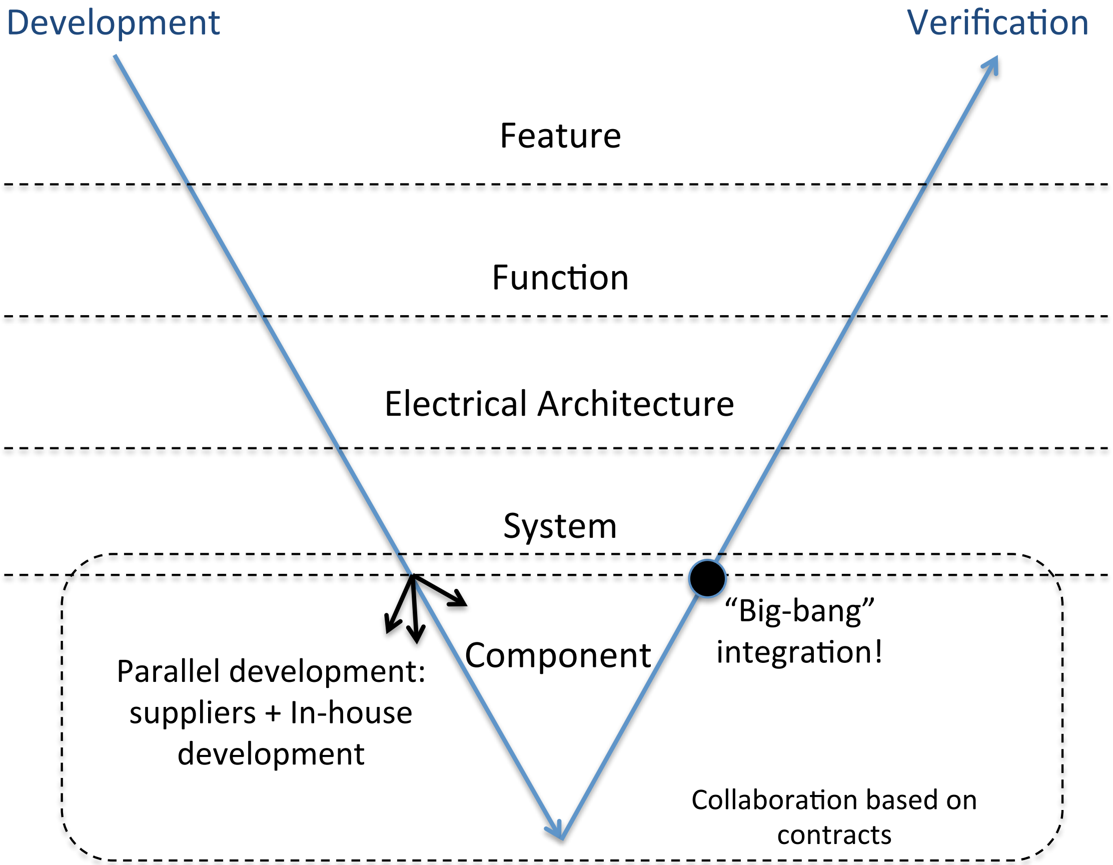
Thanks to Martin Hiller, Fuse meeting - September 23, 2016

Self-driving cars are about platforms, not about cars

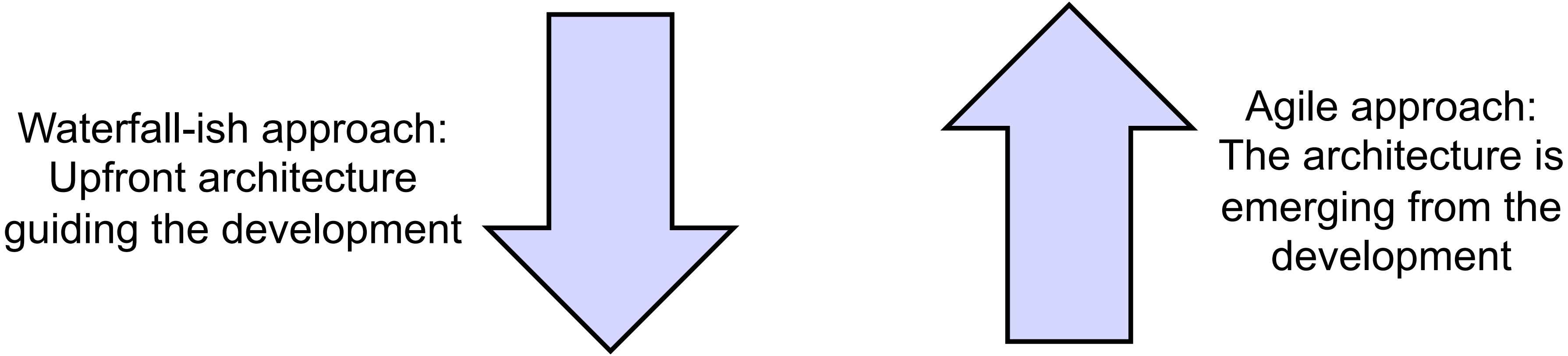


https://www.visionmobile.com/blog/2015/11/self-driving-cars-are-about-platforms-not-about-cars

CI&D and Agility, where?



CI&D: Agile architecture



Waterfall-ish approach:
Upfront architecture
guiding the development

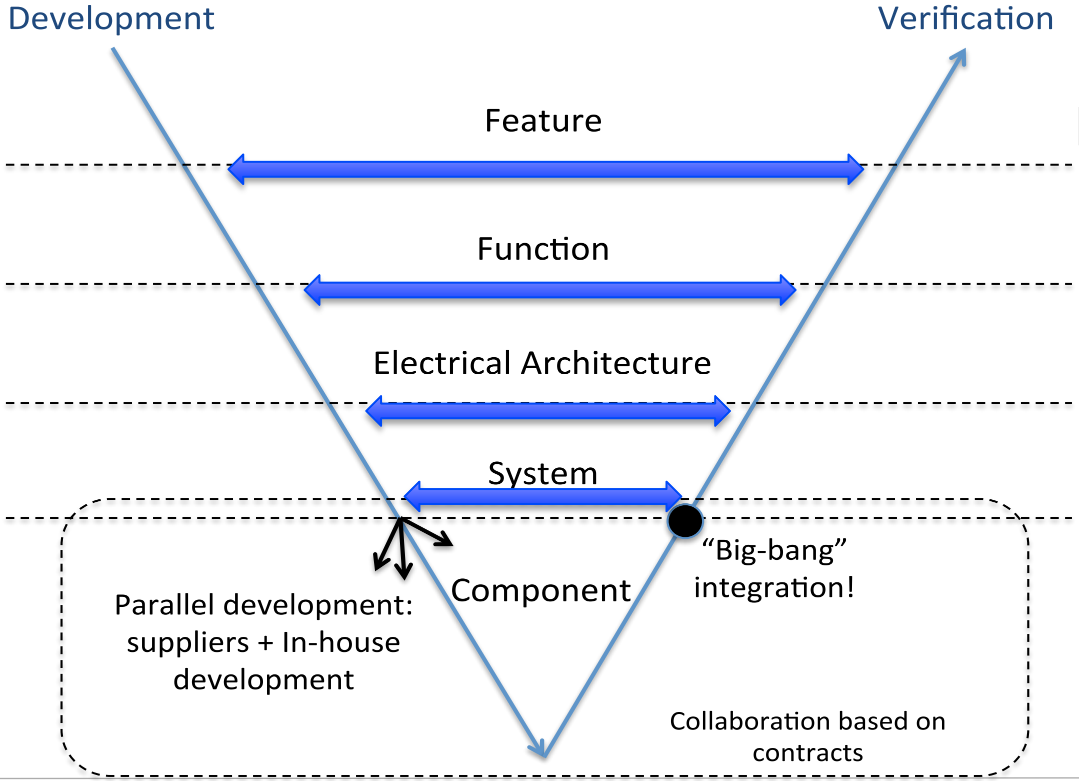
Agile approach:
The architecture is
emerging from the
development

In medio stat virtus

- Architectural design and the gradual building of the system (i.e., its user visible functionality) must go **hand-in-hand**, in **subsequent iterations**

- **Open questions**
  - Which changes will impact on the architecture?
  - How we address architectural issues, and make decisions over time in a way that will lead to a flexible architecture, and enable developers to proceed?
  - How do we keep everything synchronized?
  - How this affects the organization?

Short feedback loop through the use of models

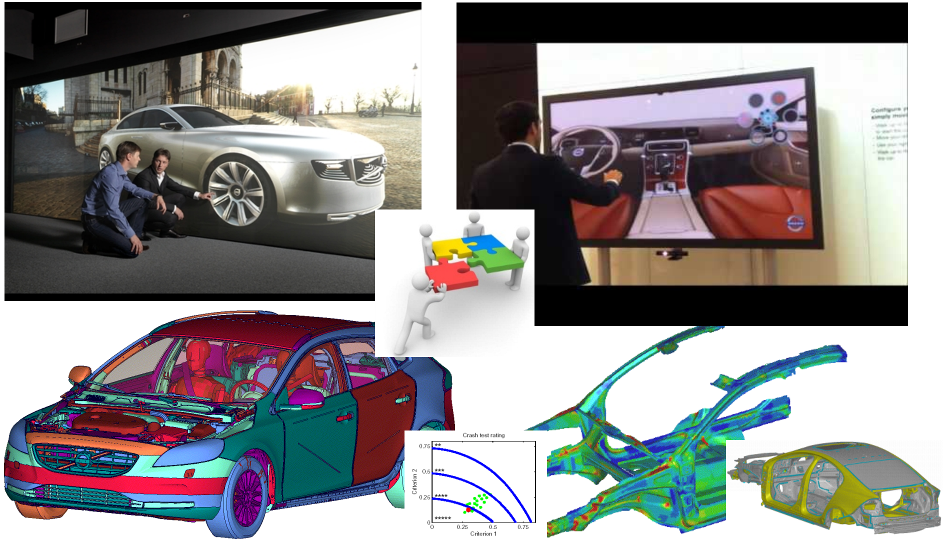


Executable models – simulation

Volvo Cars example

- **MIL** = Model in the Loop (that is, test with the model itself in a modeled environment).
- **SIL** = Software in the Loop. Replace the Model above with its generated code. Verify the same behavior.
- **HIL** = Hardware in the loop - Integrate the generated code in the real ECU, but model everything outside the ECU.

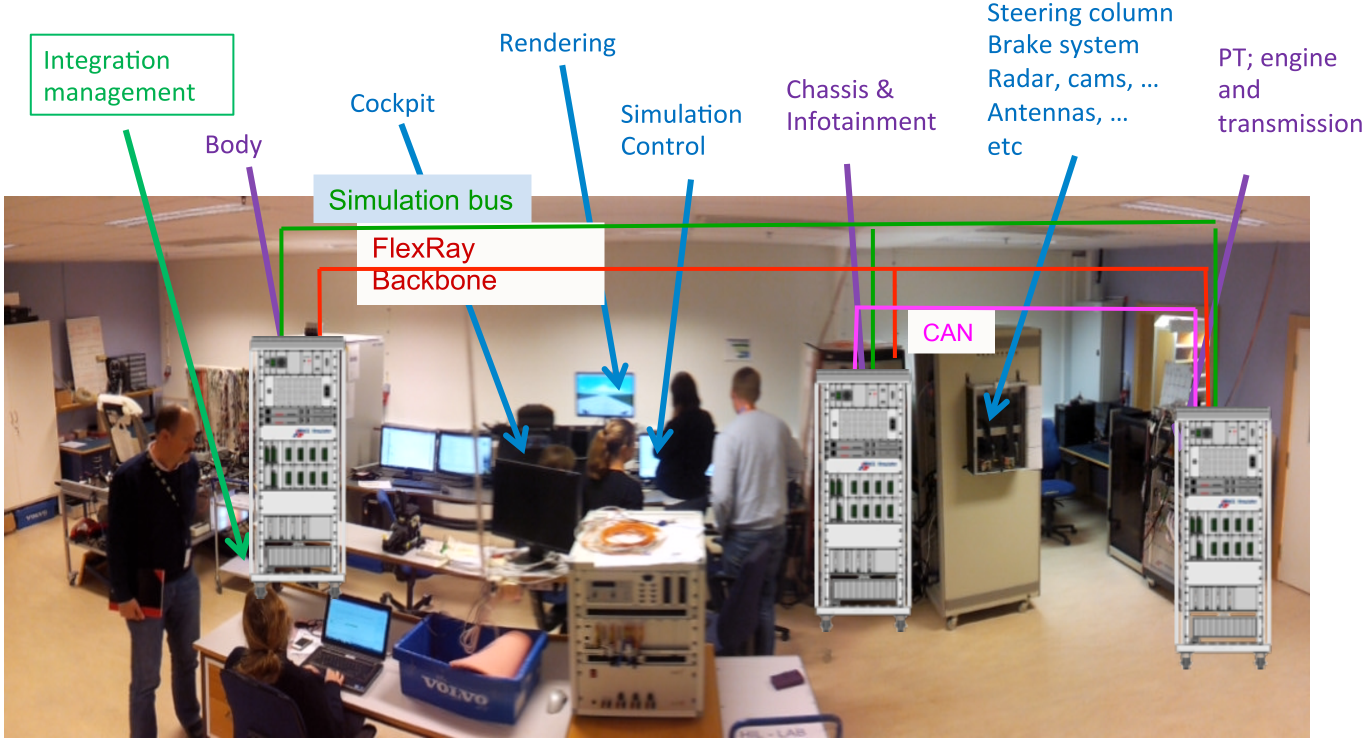
Executable models – simulation



One common virtual vehicle

Made By Göteborg - The software inside the all new XC90, Martin Nilsson, Volvo Cars

Executable models – simulation



Complete vehicle HIL – Func int. env XC90

Made By Göteborg - The software inside the all new XC90, Martin Nilsson, Volvo Cars

Executable models – simulation

- User stories
- Connectivity integration
- HMI user load evaluation
- Active Safety HMI development

- Eye tracking
- Distraction measurements
- Notification responses

Digital user experience lab

Executable models – simulation

VTI simulator

Takeaways

- Software (or system) Architecture is an important artifact for the development complex systems
  - There is the need for some **upfront** (only what it is needed and will stay stable as much as possible)
  - The architecture description should be a **living artifact** that should evolve according to the feedback coming from the development
  - There is the need to shift towards **“just in time” architecting**

- Defining an architecture for a complex and real system is much more than just modeling
  - Besides technicalities we need to consider also the **business**, **process**, and **organization** dimensions

- **Executable models** might be exploited to have early feedback even when (part of) the system, both hardware or software is not yet developed