fortiss

March 2015

View Integration in Model-Based Development

McGill NECSIS Workshop 2015

Bernhard Schätz, Sebastian Voss and Vincent Aravantinos

fortiss GmbH
An-Institut Technische Universität München

Model-Based Engineering of Embedded Systems

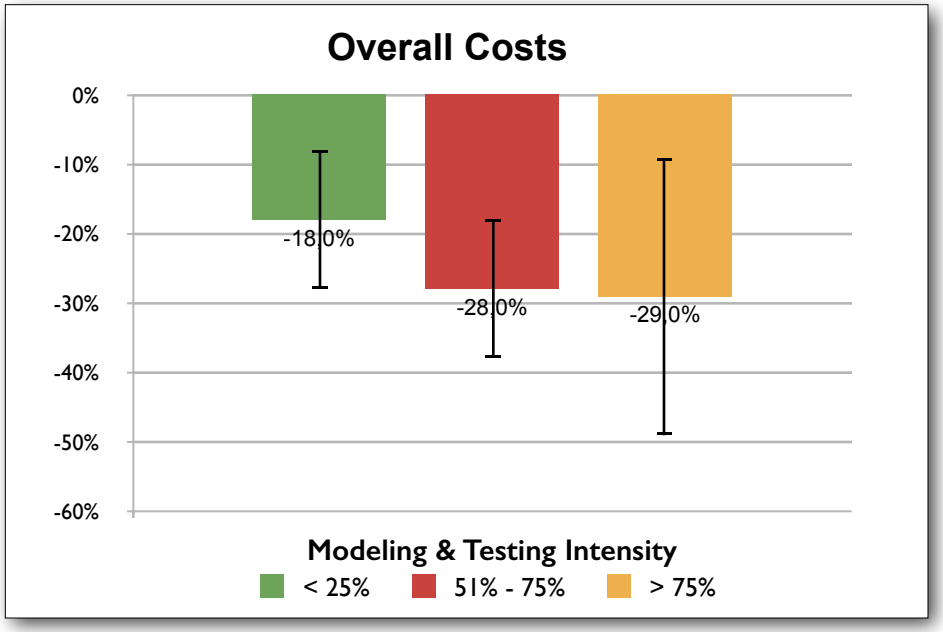
What is the benefit of a model-based design of embedded software systems in the car industry?

Manfred Broy
Technical University Munich, Germany
Sascha Kirstan
Altran Technologies, Germany
Helmut Krcmar
Technical University Munich, Germany
Bernhard Schätz
fortiss, Germany
Jens Zimmermann
Altran Technologies, Germany

ABSTRACT
Model-based development becomes more and more popular in the development of embedded software systems in the car industry. On the websites of tool vendors many success stories can be found, which report of efficiency gains from up to 50% in the development, high error reductions and a more rapid increase of the maturity level of developed functions (The Mathworks, 2010) (dSPACE 2010) just because of model-based development. Reliable and broadly spread research that analyze the status quo of model-based development and its effects on the economics are still missing. This article describes the results of a global study by Altran Technologies, the chair of software and systems engineering and the chair of Information Management of the University of Technology in Munich which examines the costs and benefits of model-based development of embedded systems in the car industry.

1. INTRODUCTION

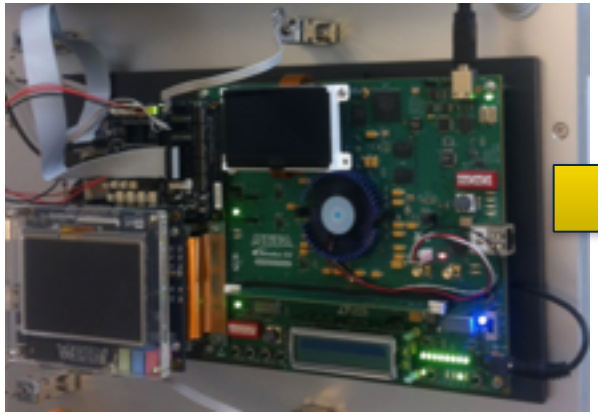
In the last 20 years the value chain in the car industry has changed drastically. All car producers and suppliers worldwide have worked on improvements in the area of mechanics, the improvement of quality requirements, and improvements in the logistic area. A lot of the potential in these areas is already exploited. A main differentiation factor turns out to be the electronics area, where a change from hardware to software development is carried out. The meaning electronics will have in the next years has been analyzed by a study of Mercer Management Consulting (Mercer, 2004). The study focuses mainly on the question how the cost factors in the development of a car will change until the year 2015 in comparison to the year 2002. In 2015 the costs for the development of electronics will have a value of 35% of the total car production costs. Whereas areas as power train and body have small increases, the costs for the development of electronic systems will be almost tripled. The predicted increases result from a variety of innovations which are being expected in this area. The majority of innovations are realized with embedded systems and especially with software. „90 percent of the future innovations in the car will be based on electronics and from that 80 percent will be realized by software" (Lederer, 2002). However, today's software development has big challenges to master like shortened development times for the cars in total versus longer development times for the software, high safety requirements and especially the growing complexity because of the rising number of functions and the increasing interaction between the functions. To master these challenges car producers and suppliers conduct a paradigm change in the software development from hand-coded to model-based development.



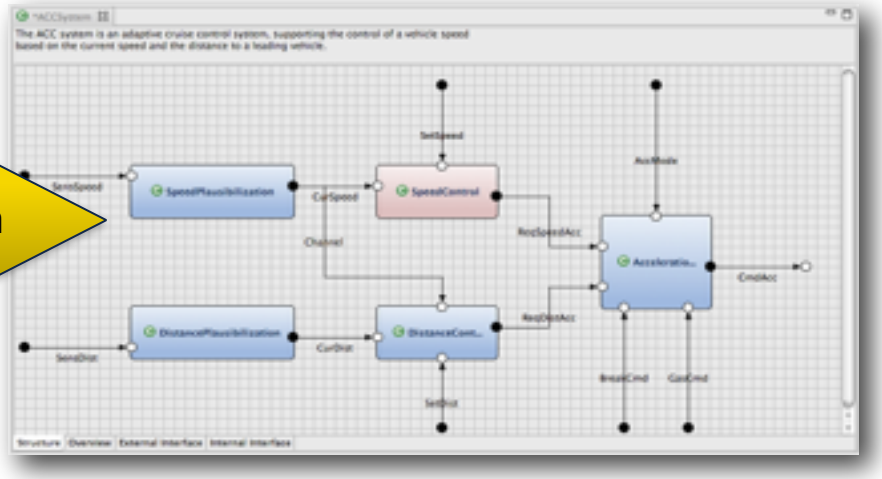
Increase efficiency of development by division of complexity

Basic Principles of Model-Based Engineering

1. Abstraction



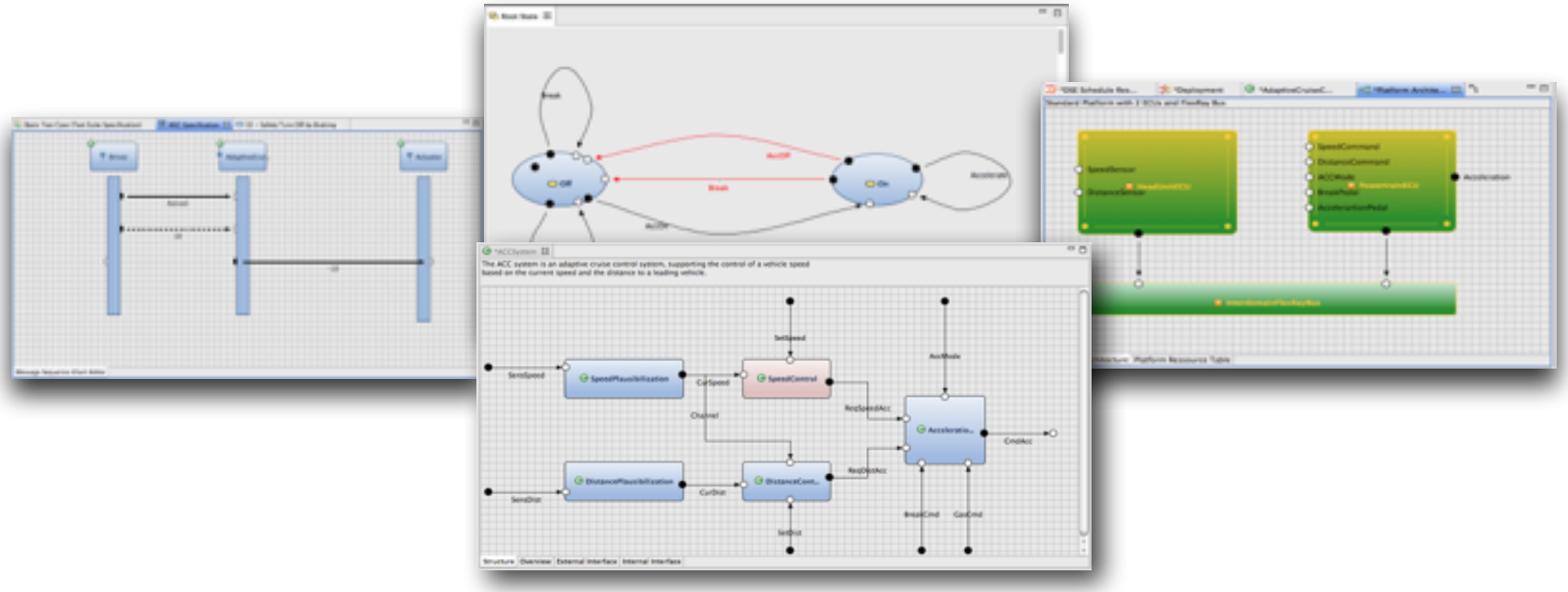
Abstraction



Idealized modeling assumptions eliminate accidental complexity of implementation level

Basic Principles of Model-Based Engineering

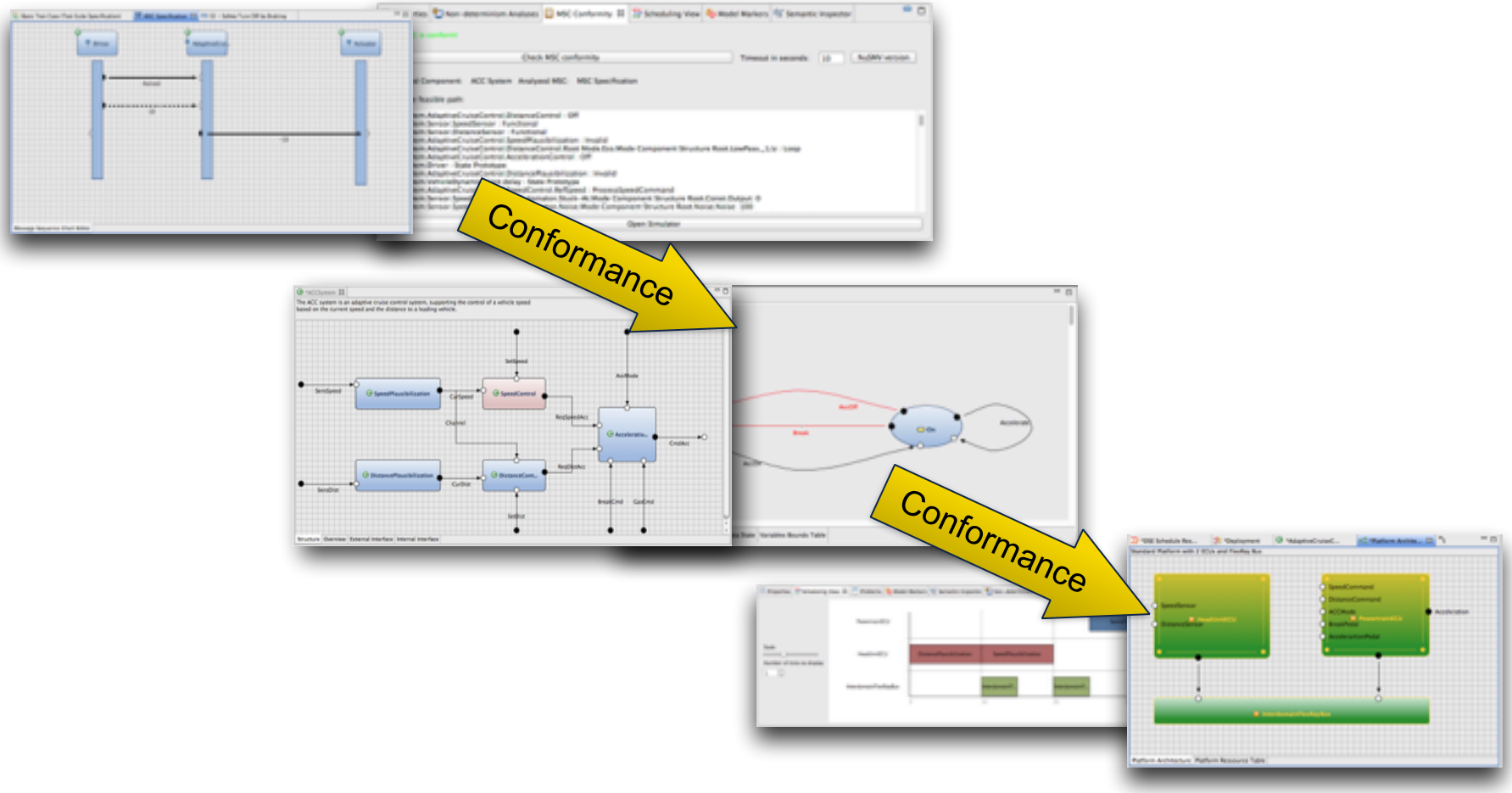
2. Views



Modular projections of system reduce essential complexity of problem

Basic Principles of Model-Based Engineering

3. Analysis/Synthesis



Automated steps ensure conformance across abstractions and views

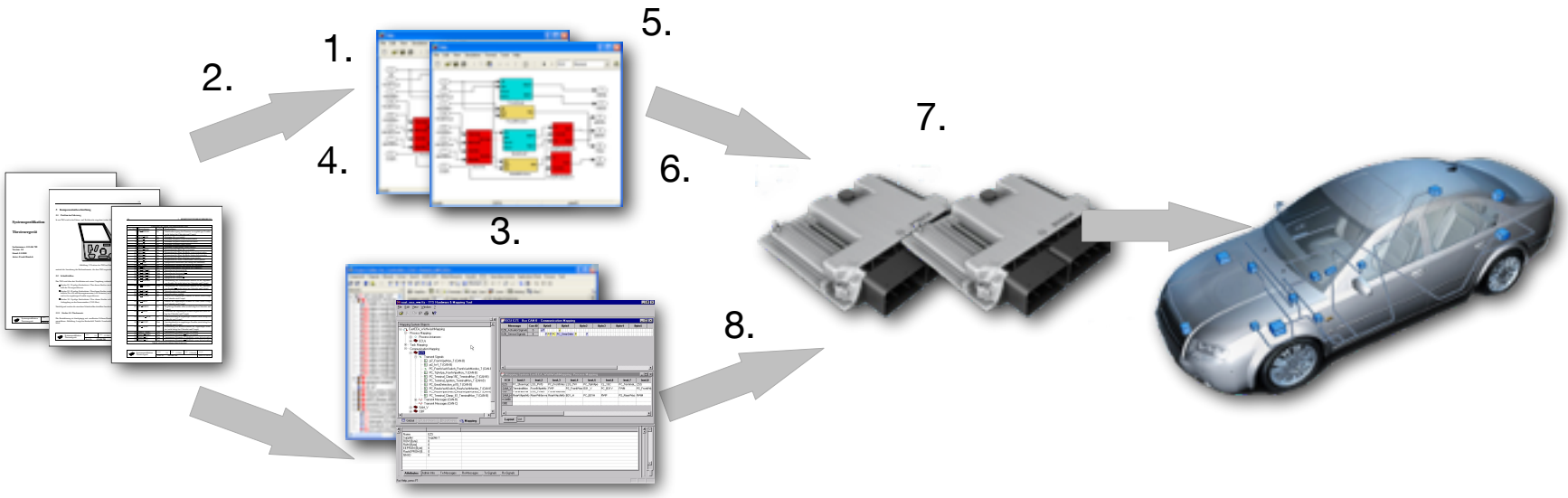
fortiss

Example: MBSE with AF3

AF3-Demo — see af3.fortiss.org

fortiss

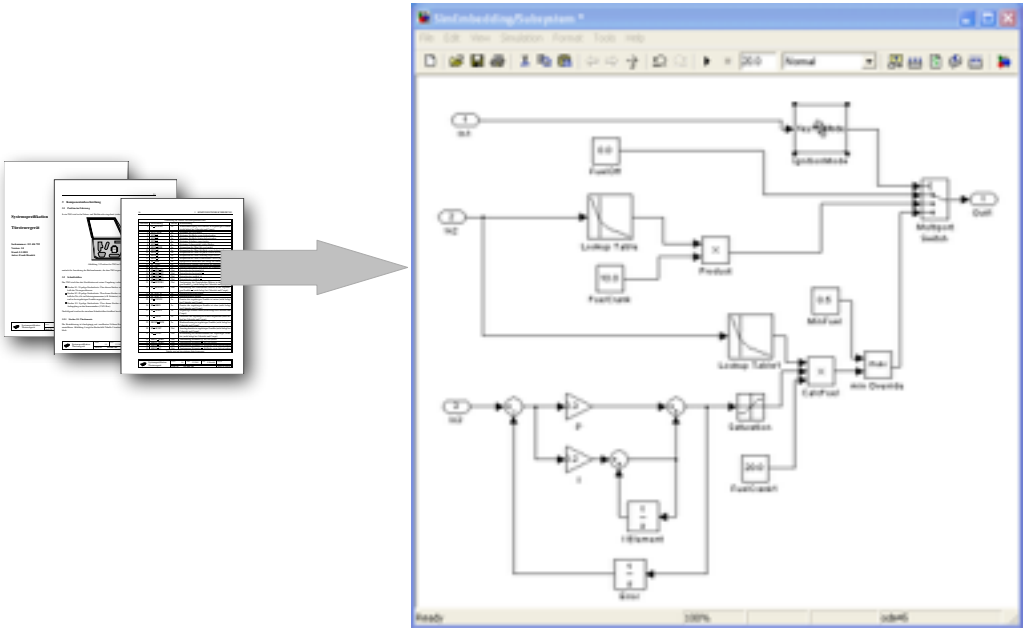
Automotive Software Development: Process



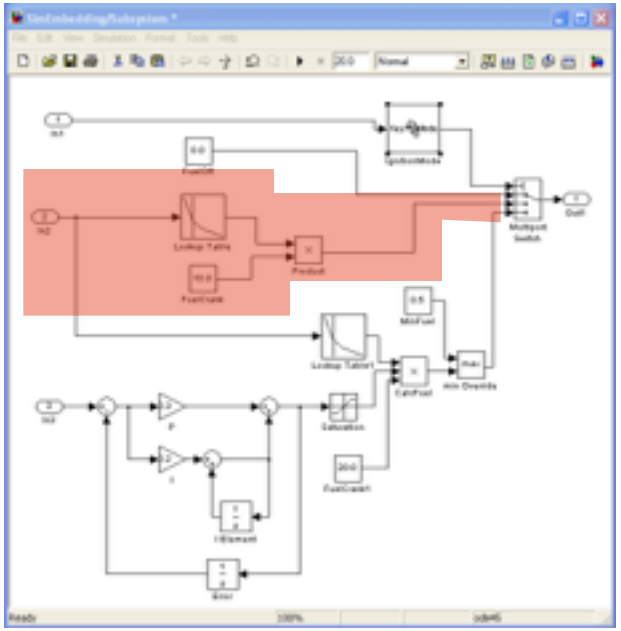
State-of-Art Development: Deficit by lacking models or inadequate use

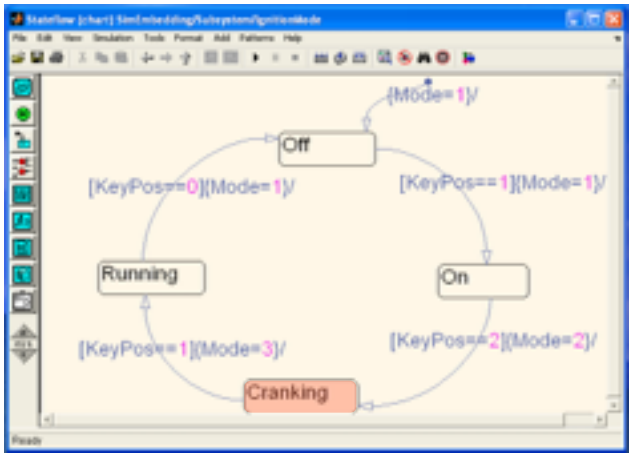
1. Example Component Specification: Lacking domain concepts
2. Example Requirements Specification: Lacking descriptive techniques
3. Example Implementation: Lacking view integration
4. Example Component Design: Lacking consistency analysis
5. Example Component Evolution: Lacking maintenance support
6. Example Component Verification: Lacking property verification
7. Example Verification Specification: Lacking test case generation
8. Example system integration: Lacking design space exploration

Example 1: Component specification

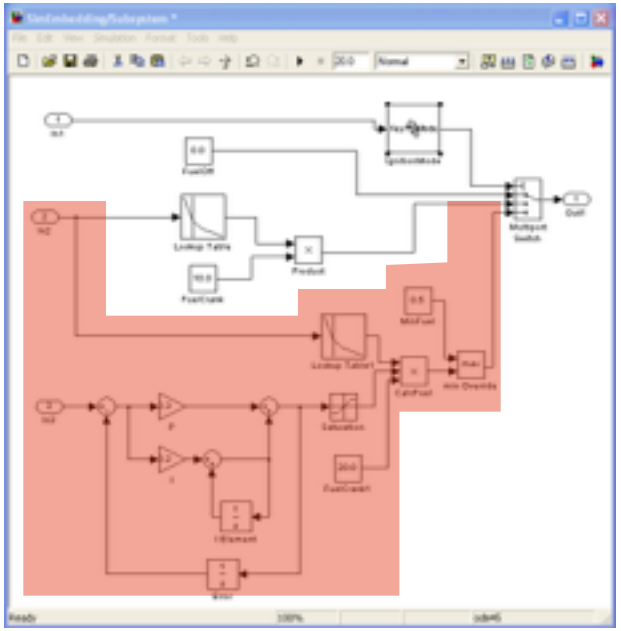


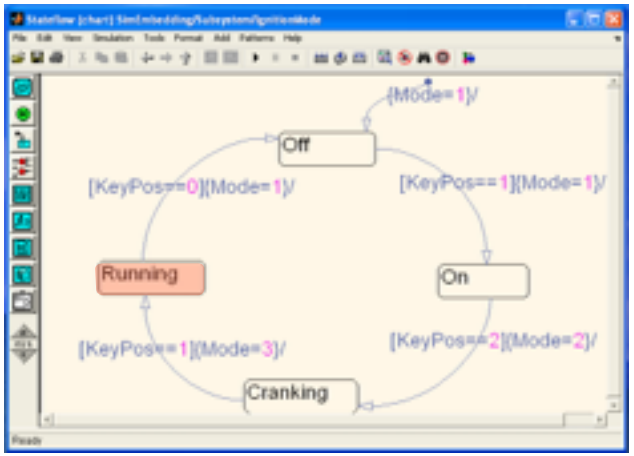
Example 1: Component specification



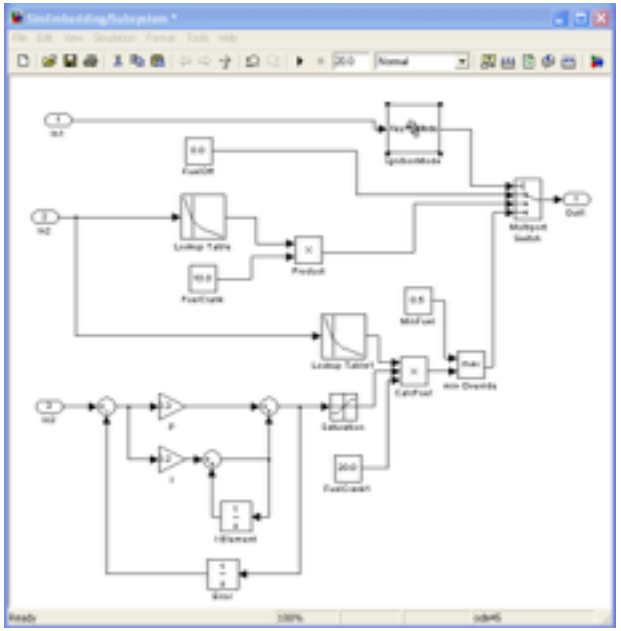


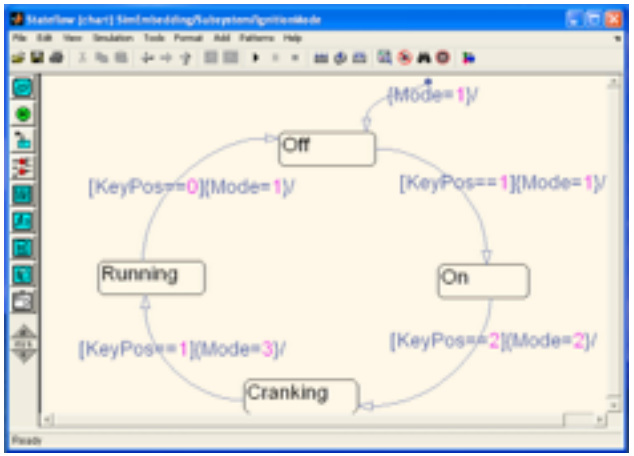
Example 1: Component specification





Example 1: Component specification



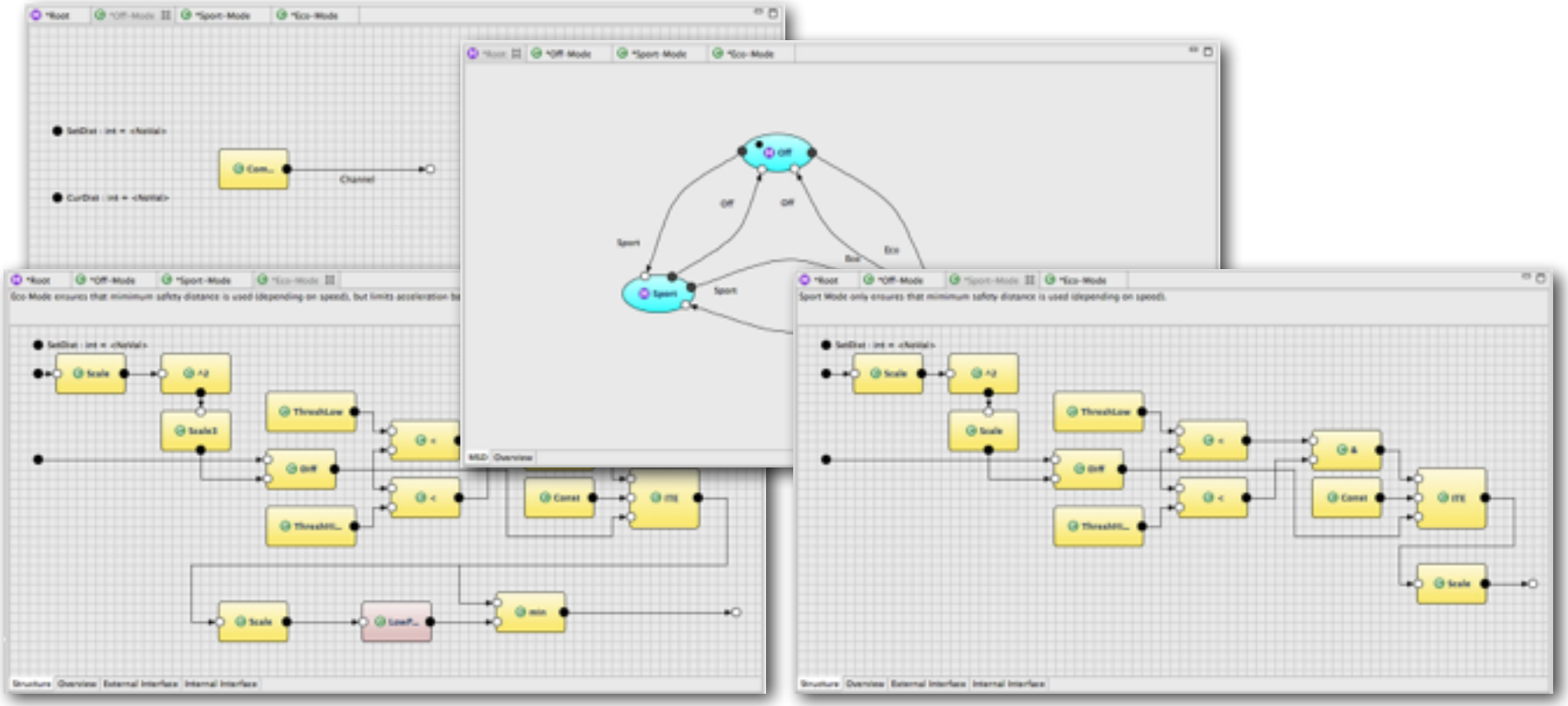


State-of-art: Inadequate models - Models capture technical solution

Problem: Lacking concepts of problem domain

Consequence: Inadequate models of functionality of application

Domain-specific models: Modes of operation



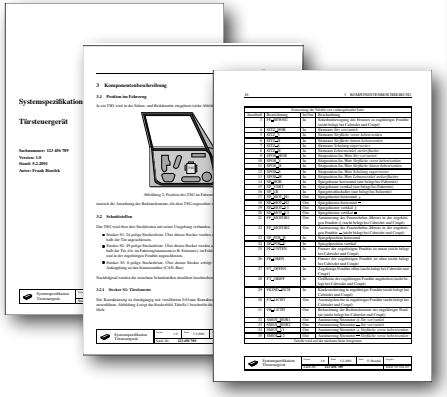
Improvement: Adequate models - Application domain vs technology domain

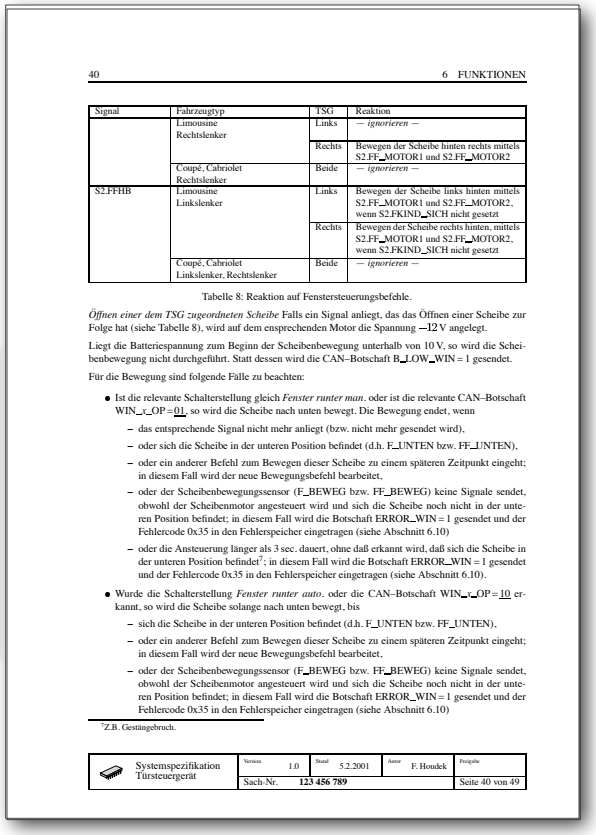
Method: Extension of modeling techniques by domain concepts

Further examples: Signal classes (state/event signals), timing constraints

Improvement: More effective modeling, more targeted use

Example 2: Requirement Specification





40 6 FUNKTIONEN

Signal	Fahrzeugtyp	TSG	Reaktion
	Limousine Rechtslenker	Links	— *ignorieren* —
		Rechts	Bewegen der Scheibe hinten rechts mittels S2.FF_MOTOR1 und S2.FF_MOTOR2
	Coupé, Cabriolet Rechtslenker	Beide	— *ignorieren* —
S2.FFHB	Limousine Linkslenker	Links	Bewegen der Scheibe links hinten mittels S2.FF_MOTOR1 und S2.FF_MOTOR2, wenn S2.FKIND_SICH nicht gesetzt
		Rechts	Bewegen der Scheibe rechts hinten, mittels S2.FF_MOTOR1 und S2.FF_MOTOR2, wenn S2.FKIND_SICH nicht gesetzt
	Coupé, Cabriolet Linkslenker, Rechtslenker	Beide	— *ignorieren* —

Tabelle 8: Reaktion auf Fenstersteuerungsbefehle.

Öffnen einer dem TSG zugeordneten Scheibe Falls ein Signal anliegt, das das Öffnen einer Scheibe zur Folge hat (siehe Tabelle 8), wird auf dem ensprechenden Motor die Spannung −12 V angelegt.

Liegt die Batteriespannung zum Beginn der Scheibenbewegung unterhalb von 10 V, so wird die Scheibenbewegung nicht durchgeführt. Statt dessen wird die CAN–Botschaft B_LOW_WIN = 1 gesendet.

Für die Bewegung sind folgende Fälle zu beachten:

- Ist die relevante Schalterstellung gleich *Fenster runter man.* oder ist die relevante CAN–Botschaft WIN_r_OP = 01, so wird die Scheibe nach unten bewegt. Die Bewegung endet, wenn
  - das entsprechende Signal nicht mehr anliegt (bzw. nicht mehr gesendet wird),
  - oder sich die Scheibe in der unteren Position befindet (d.h. F_UNTEN bzw. FF_UNTEN),
  - oder ein anderer Befehl zum Bewegen dieser Scheibe zu einem späteren Zeitpunkt eingeht; in diesem Fall wird der neue Bewegungsbefehl bearbeitet,
  - oder der Scheibenbewegungssensor (F_BEWEG bzw. FF_BEWEG) keine Signale sendet, obwohl der Scheibenmotor angesteuert wird und sich die Scheibe noch nicht in der unteren Position befindet; in diesem Fall wird die Botschaft ERROR_WIN = 1 gesendet und der Fehlercode 0x35 in den Fehlerspeicher eingetragen (siehe Abschnitt 6.10)
  - oder die Ansteuerung länger als 3 sec. dauert, ohne daß erkannt wird, daß sich die Scheibe in der unteren Position befindet[7]; in diesem Fall wird die Botschaft ERROR_WIN = 1 gesendet und der Fehlercode 0x35 in den Fehlerspeicher eingetragen (siehe Abschnitt 6.10).
- Wurde die Schalterstellung *Fenster runter auto.* oder die CAN–Botschaft WIN_r_OP = 10 erkannt, so wird die Scheibe solange nach unten bewegt, bis
  - sich die Scheibe in der unteren Position befindet (d.h. F_UNTEN bzw. FF_UNTEN),
  - oder ein anderer Befehl zum Bewegen dieser Scheibe zu einem späteren Zeitpunkt eingeht; in diesem Fall wird der neue Bewegungsbefehl bearbeitet,
  - oder der Scheibenbewegungssensor (F_BEWEG bzw. FF_BEWEG) keine Signale sendet, obwohl der Scheibenmotor angesteuert wird und sich die Scheibe noch nicht in der unteren Position befindet; in diesem Fall wird die Botschaft ERROR_WIN = 1 gesendet und der Fehlercode 0x35 in den Fehlerspeicher eingetragen (siehe Abschnitt 6.10)

[7] Z.B. Gestängebruch.

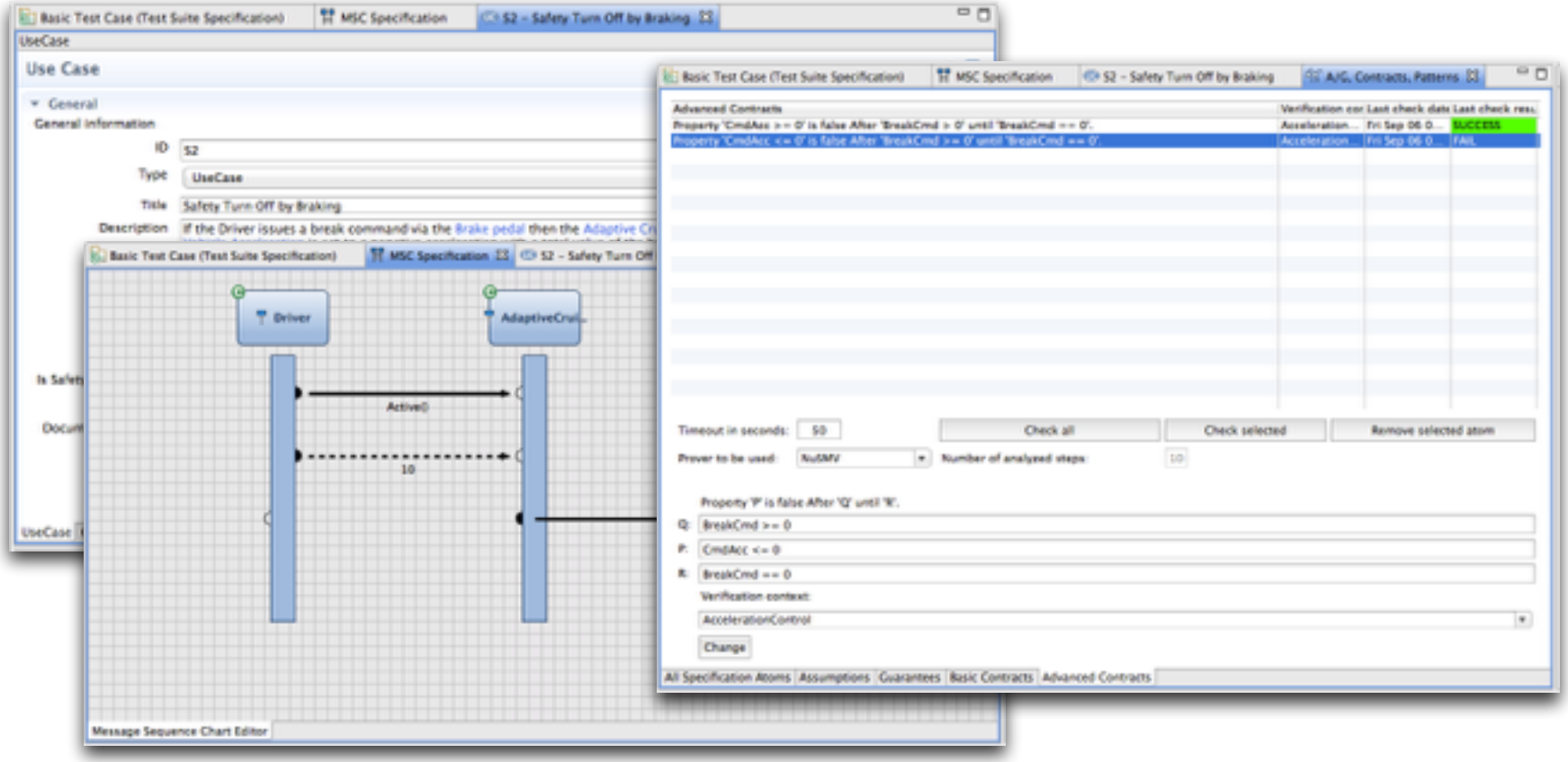
Systemspezifikation Türsteuergerät | Version 1.0 | Stand 5.2.2001 | Autor F. Houdek | Freigabe | Sach-Nr. 123 456 789 | Seite 40 von 49

State-of-art: Lacking models - No modeling of required (forbidden) behavior

Problem: Lacking models for non-constructive specifications

Consequence: No explicit reference specifications for (early) verification

Descriptive Techniques: Interaction Sequences



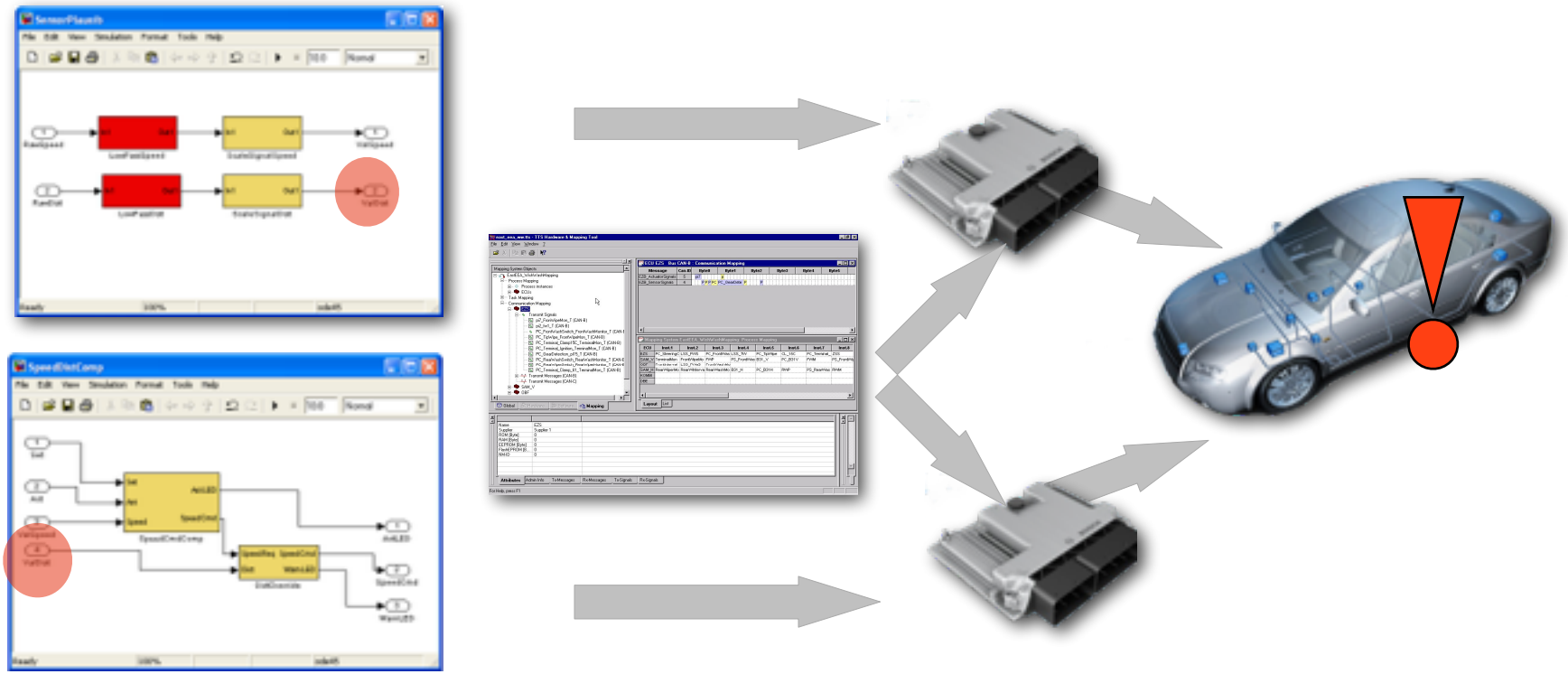
Improvement: Early phases - Specifications of requirements

Methods: Description of intended blackbox behavior

Further examples: Property languages

Improvement: Specification without details on construction

Example 3: Implementation

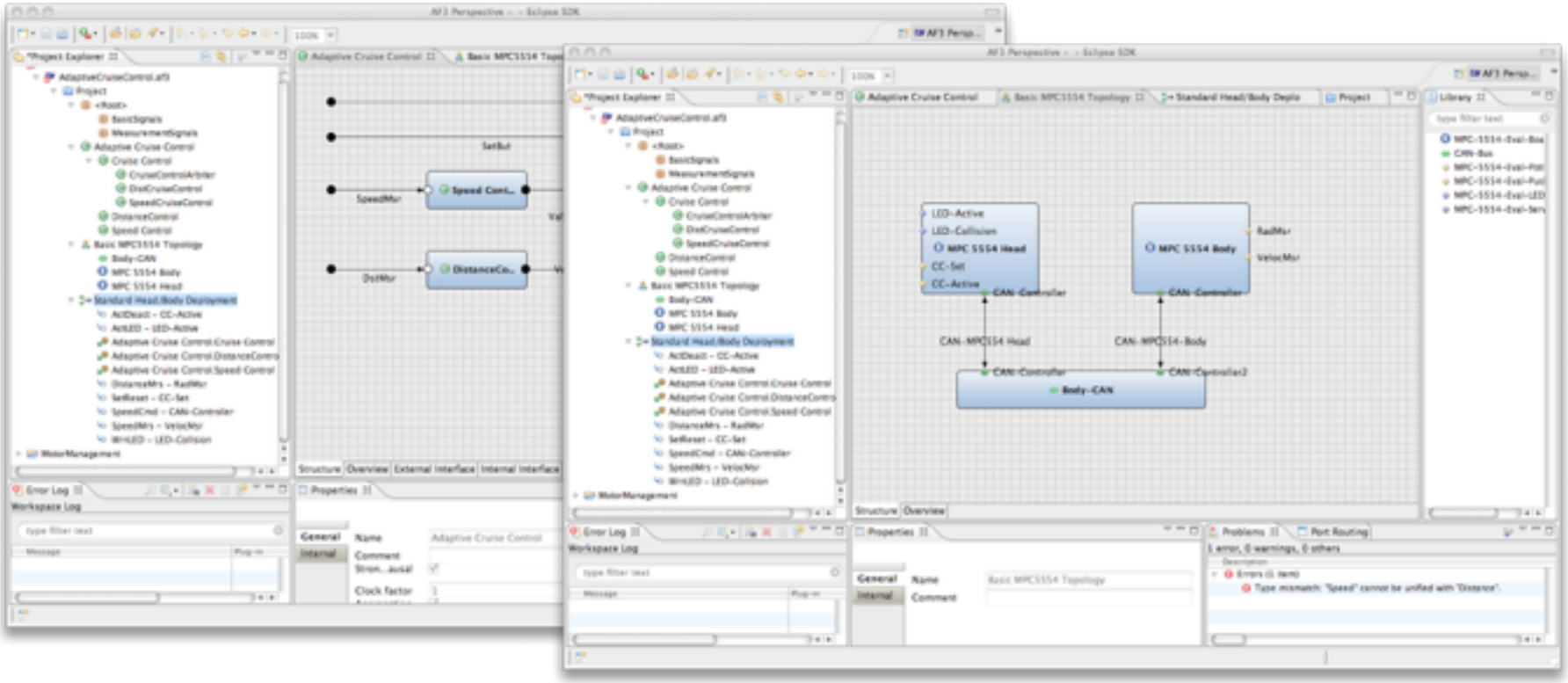


State-of-art: Isolated models - Models for individual steps of development

Problem: Gaps in development process

Consequence: Defects by contradicting decisions in later phases

Integrated models: Component-topology-mapping



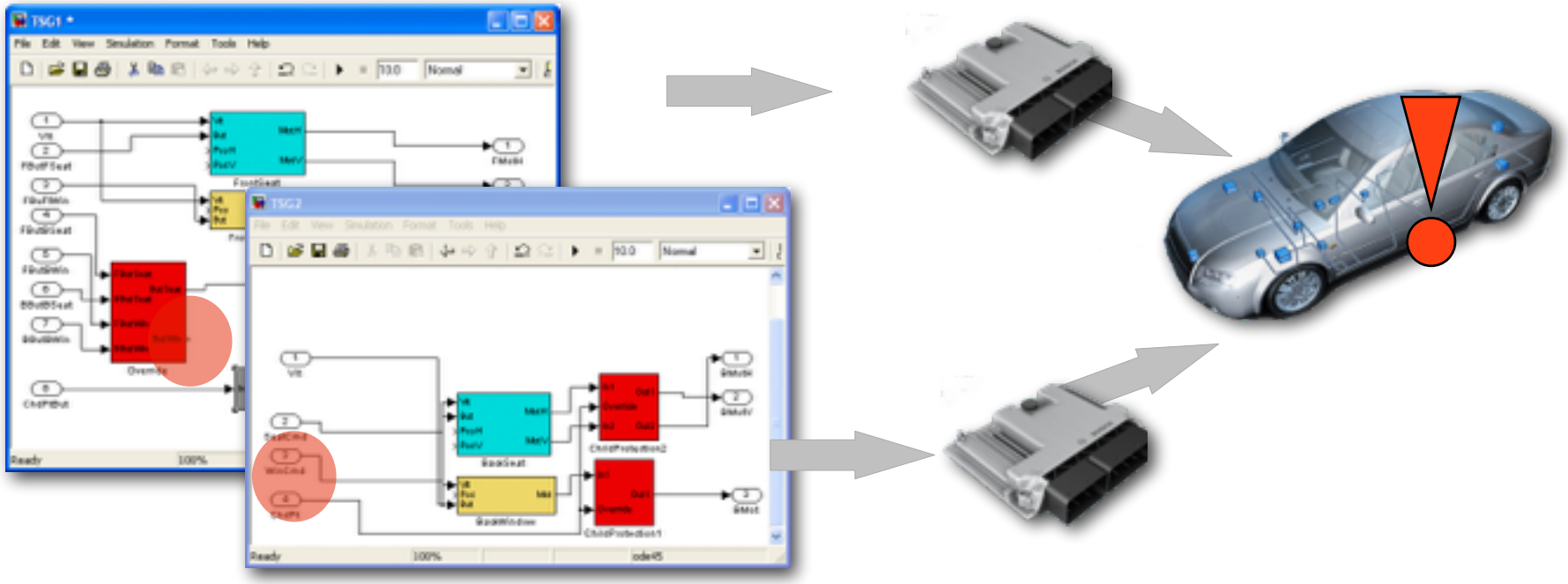
Improvement: Integrated models - Coherent description of system

Methods: Integrated formalization of all system views

Further Examples: Test case refinement (design-level/implementation level)

Improvement: Avoidance of redundancy/inconsistency

Example 4: Component Desgin



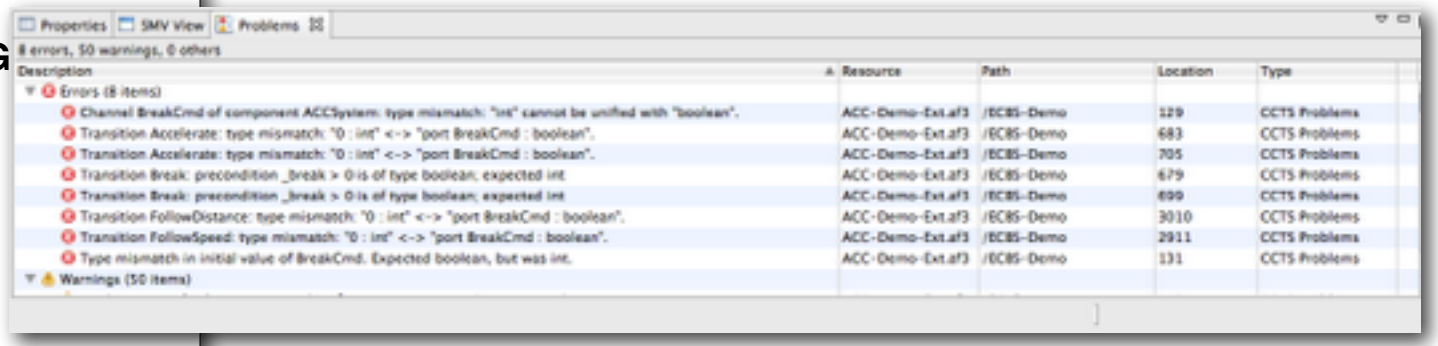
State-of-art: Late quality assurance - Late detection of defects

Problem: Lacking analysis support for early development steps

Consequence: Conservation of early defects throughout the development process

Analysis Method: Conformance Checking of Models

Modeling G



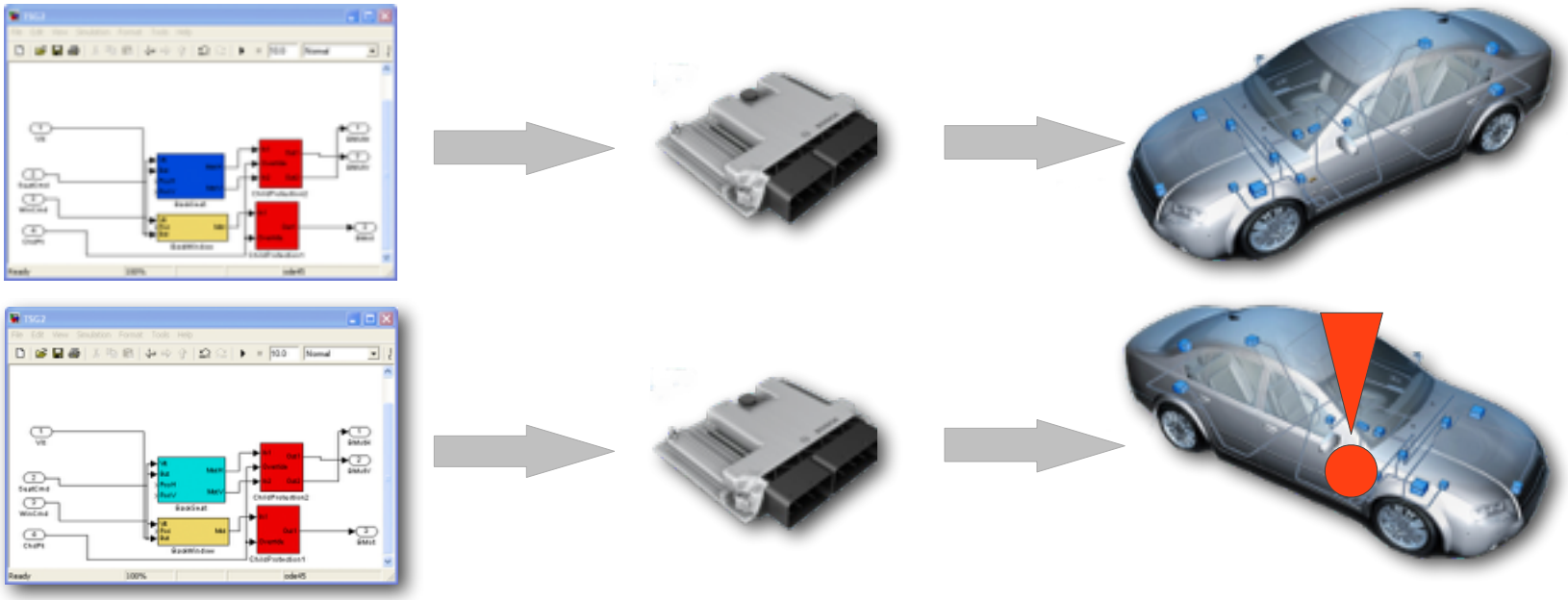
Improvement: Early Quality assurance - Early sound models via front-loading QS

Method: Automatic checks for model structure/model content

Further examples: Type conformance, definedness of signals

Improvement: Early defect elimination

Example 5: Model Maintenance

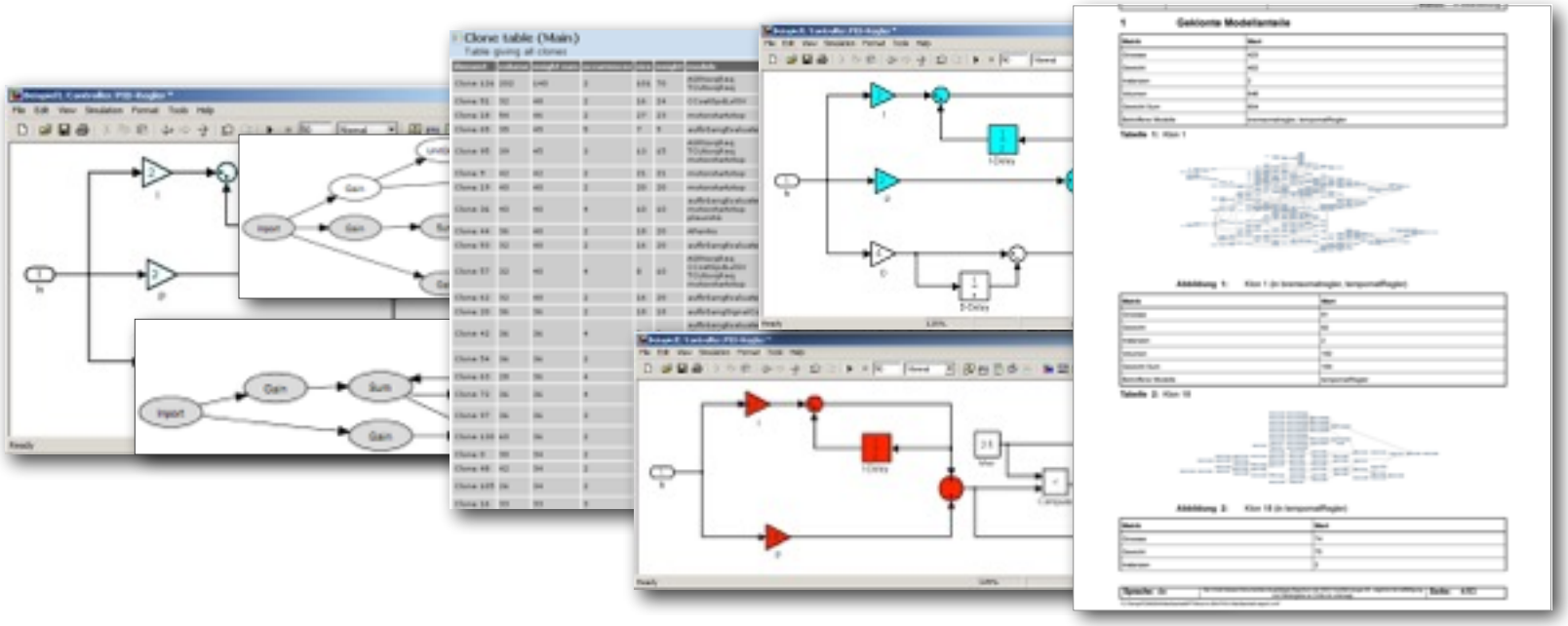


State-of-art: Unsystematic reuse - Introduction of change anomalies

Problem: Lacking Analysis methods supporting maintenance

Consequence: Erschwerung und Verteuerung von Wartungsaufgaben

Analysis Method: Clone Detection



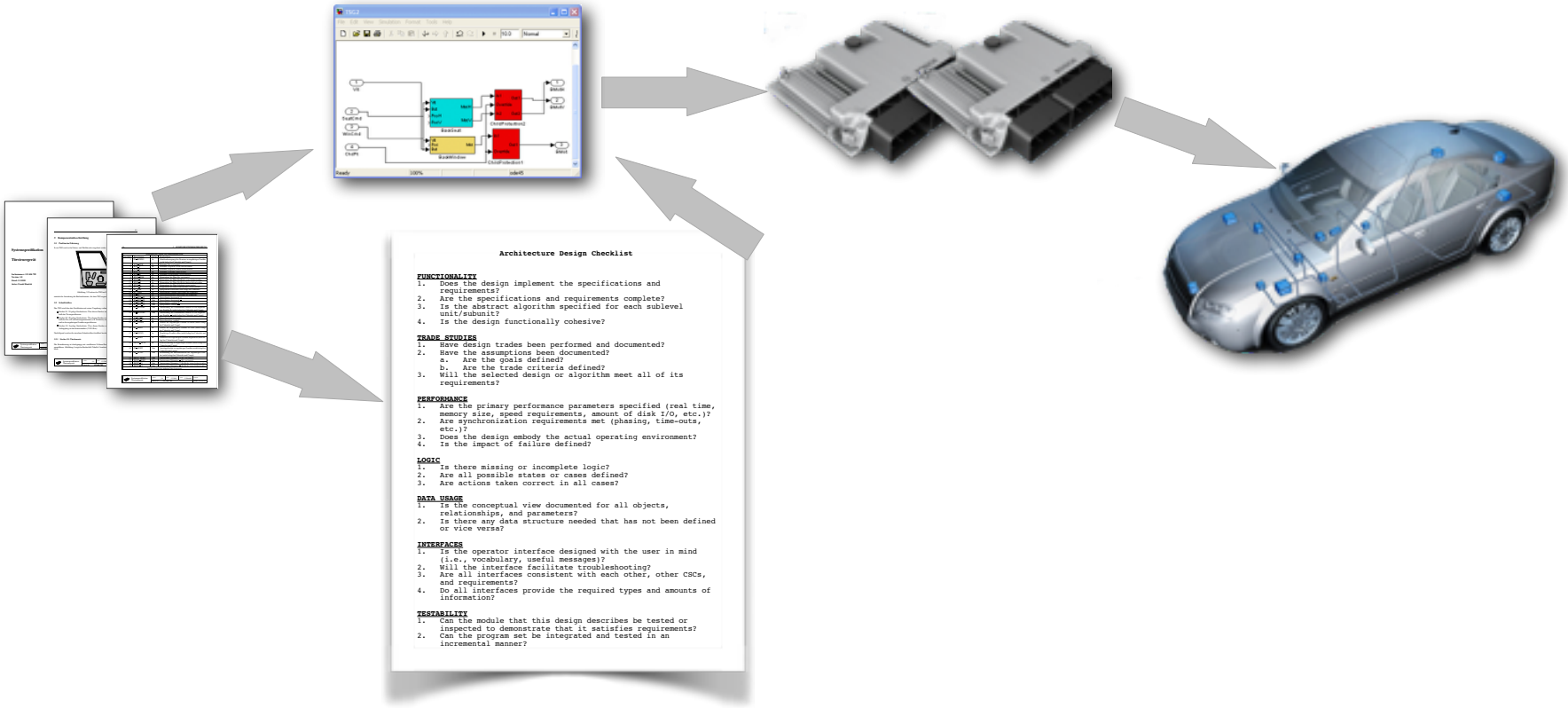
Improvement: Identification of Identical Parts - Avoidance of change anomalies

Methods: Automatic analysis of (structure of) component behavior

Further examples: Requirements clones, IP-Libraries, product lines

Improvement: Simplification of model maintenance

Example 6: Component Verification

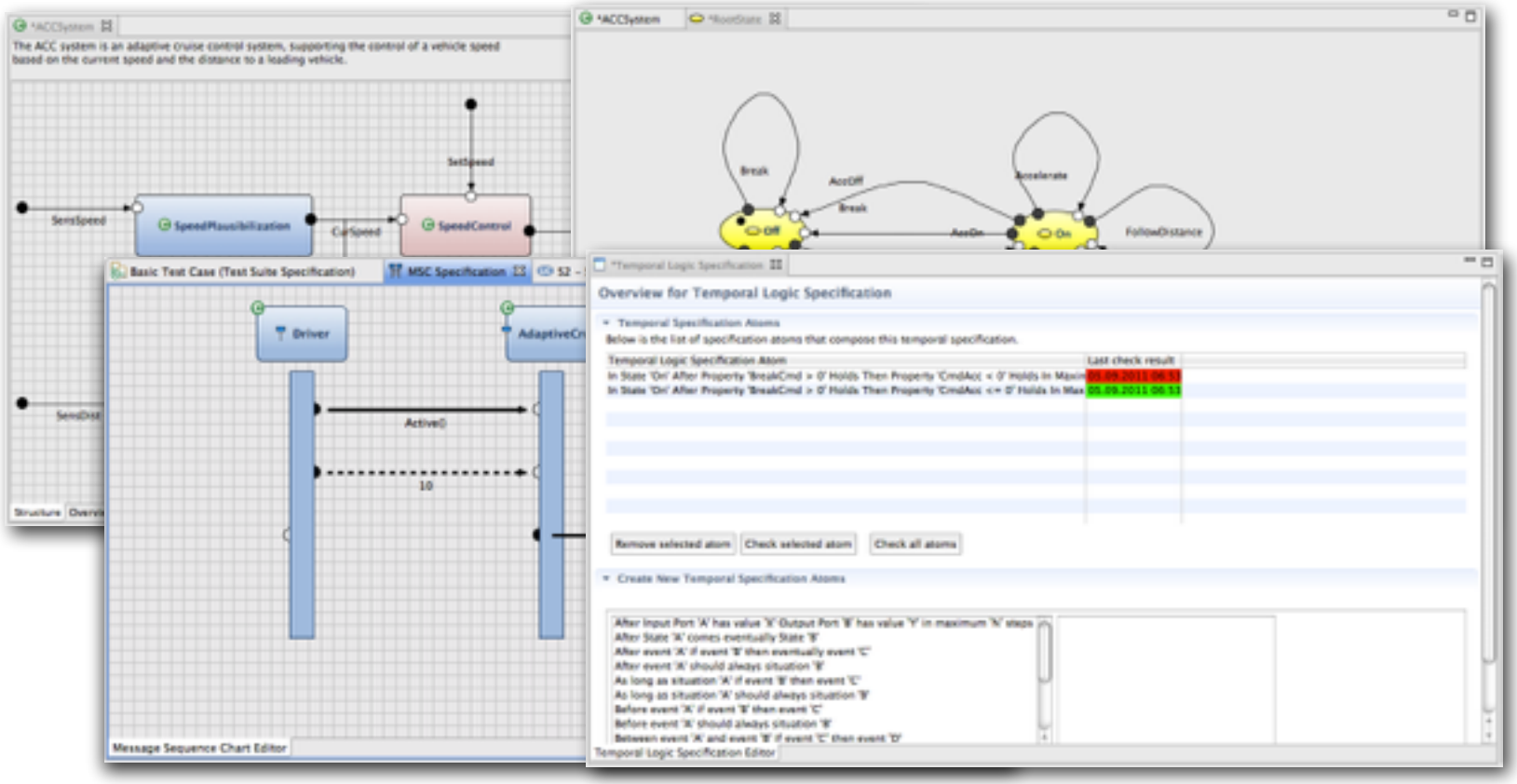


State-of-art: Manual Review Prozess - Expensive early verification

Problem: Lacking automatization of early validation

Consequence: Expensive and error-prone verification of core properties

Analysis method: Property Verification



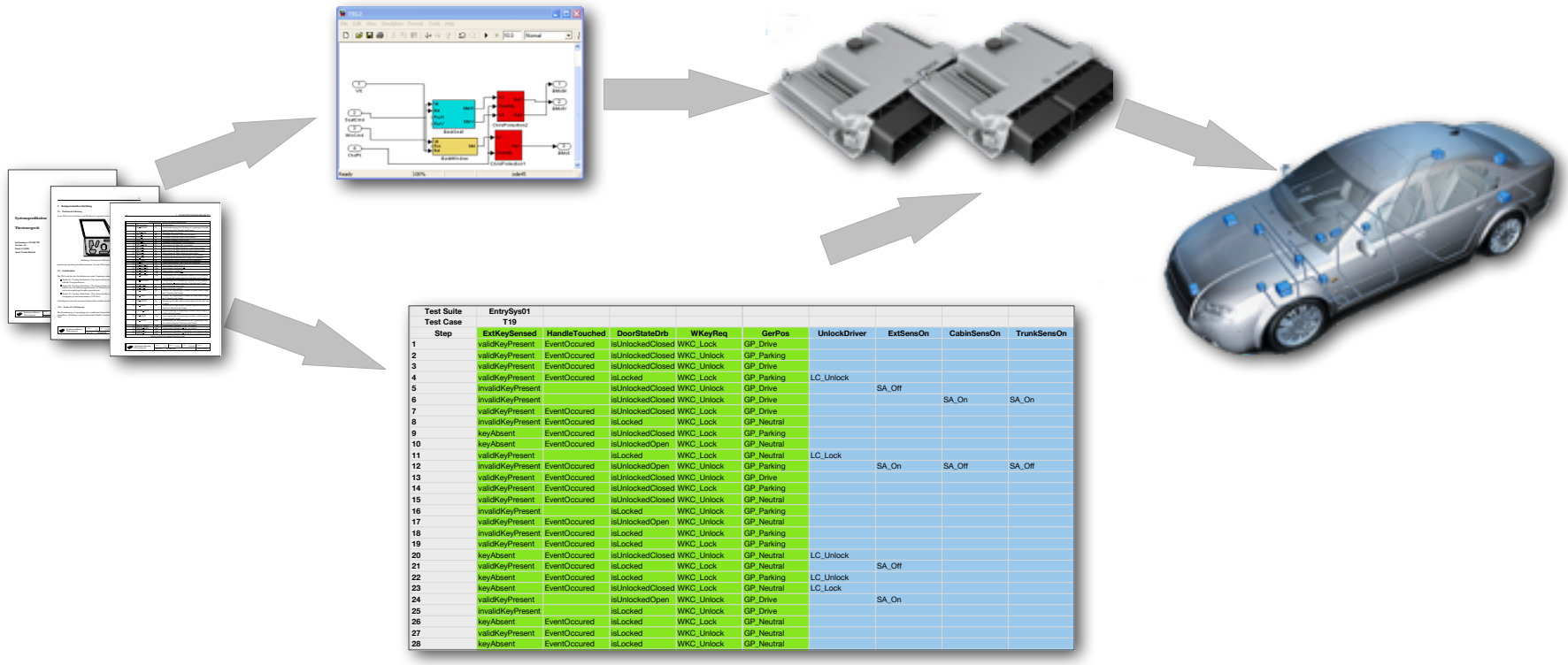
Improvement: Automated property verification - Systematic early verification

Method: Automated analysis by formal verification

Further examples: Checking of determinism

Improvement: Early defect identification

Example 7: Verification



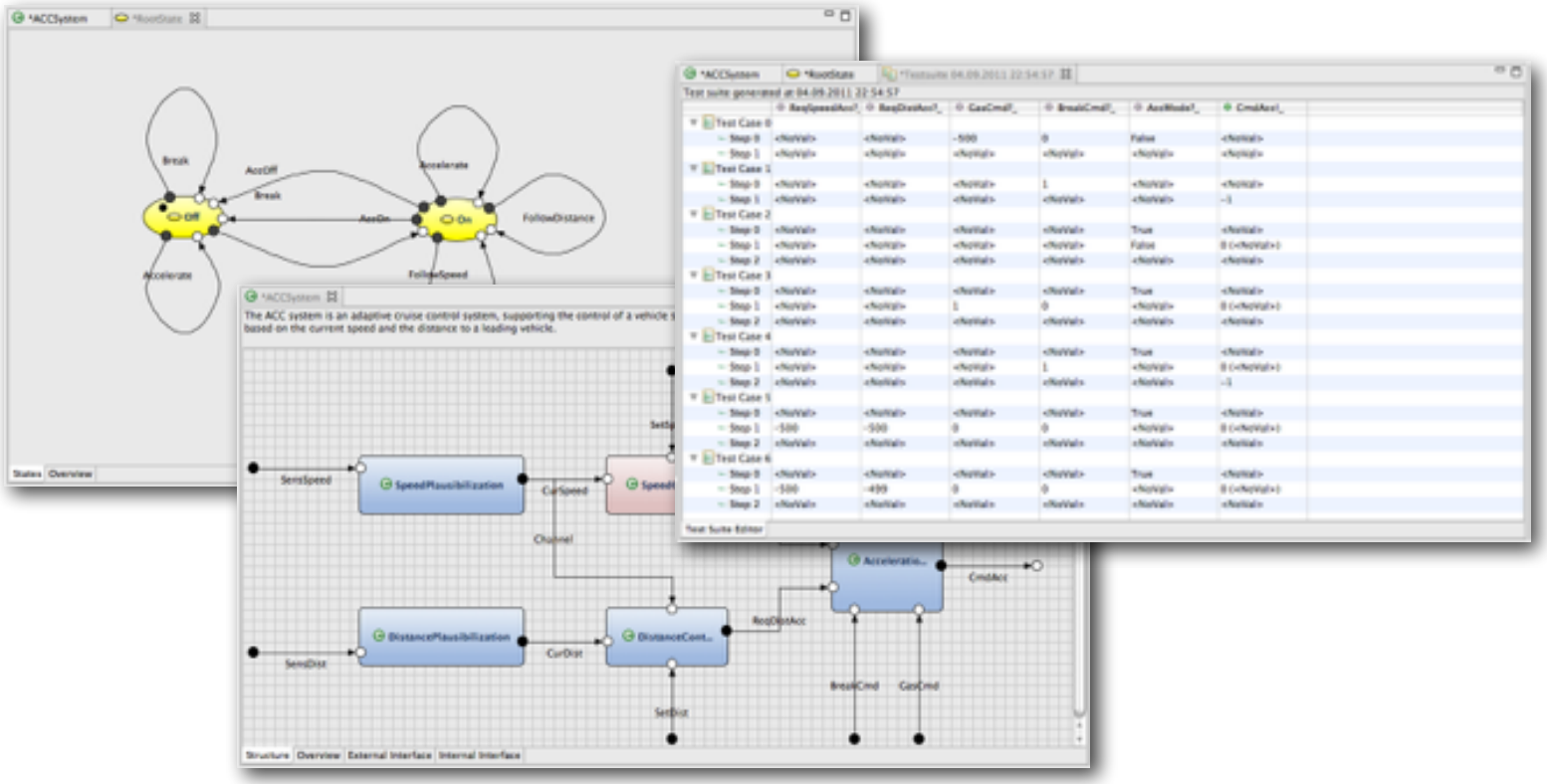
Test Suite	EntrySys01								
Test Case	T19								
Step	ExtKeySensed	HandleTouched	DoorStateDrb	WKeyReq	GerPos	UnlockDriver	ExtSensOn	CabinSensOn	TrunkSensOn
1	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Lock	GP_Drive				
2	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Unlock	GP_Parking				
3	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Unlock	GP_Drive				
4	validKeyPresent	EventOccured	isLocked	WKC_Lock	GP_Parking	LC_Unlock			
5	invalidKeyPresent		isUnlockedClosed	WKC_Unlock	GP_Drive		SA_Off		
6	invalidKeyPresent		isUnlockedClosed	WKC_Unlock	GP_Drive			SA_On	SA_On
7	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Lock	GP_Drive				
8	invalidKeyPresent	EventOccured	isLocked	WKC_Lock	GP_Neutral				
9	keyAbsent	EventOccured	isUnlockedClosed	WKC_Lock	GP_Parking				
10	keyAbsent	EventOccured	isUnlockedOpen	WKC_Lock	GP_Neutral				
11	validKeyPresent		isLocked	WKC_Lock	GP_Neutral	LC_Lock			
12	invalidKeyPresent	EventOccured	isUnlockedOpen	WKC_Unlock	GP_Parking		SA_On	SA_Off	SA_Off
13	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Unlock	GP_Drive				
14	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Lock	GP_Parking				
15	validKeyPresent	EventOccured	isUnlockedClosed	WKC_Unlock	GP_Neutral				
16	invalidKeyPresent		isLocked	WKC_Unlock	GP_Parking				
17	validKeyPresent	EventOccured	isUnlockedOpen	WKC_Unlock	GP_Neutral				
18	invalidKeyPresent	EventOccured	isLocked	WKC_Unlock	GP_Parking				
19	validKeyPresent	EventOccured	isLocked	WKC_Lock	GP_Parking				
20	keyAbsent	EventOccured	isUnlockedClosed	WKC_Unlock	GP_Neutral	LC_Unlock			
21	validKeyPresent	EventOccured	isLocked	WKC_Lock	GP_Neutral		SA_Off		
22	keyAbsent	EventOccured	isLocked	WKC_Lock	GP_Parking	LC_Unlock			
23	keyAbsent	EventOccured	isUnlockedClosed	WKC_Lock	GP_Neutral	LC_Lock			
24	validKeyPresent		isUnlockedOpen	WKC_Unlock	GP_Drive		SA_On		
25	invalidKeyPresent		isLocked	WKC_Unlock	GP_Drive				
26	keyAbsent	EventOccured	isLocked	WKC_Lock	GP_Neutral				
27	validKeyPresent	EventOccured	isLocked	WKC_Unlock	GP_Neutral				
28	keyAbsent	EventOccured	isLocked	WKC_Unlock	GP_Neutral				

State-of-art: Manual test case definition - Expensive test suite construction

Problem: Lacking automatization of test case definition

Consequence: Expensive and un-systematic verification of functionality

Synthesis method: Test case generation



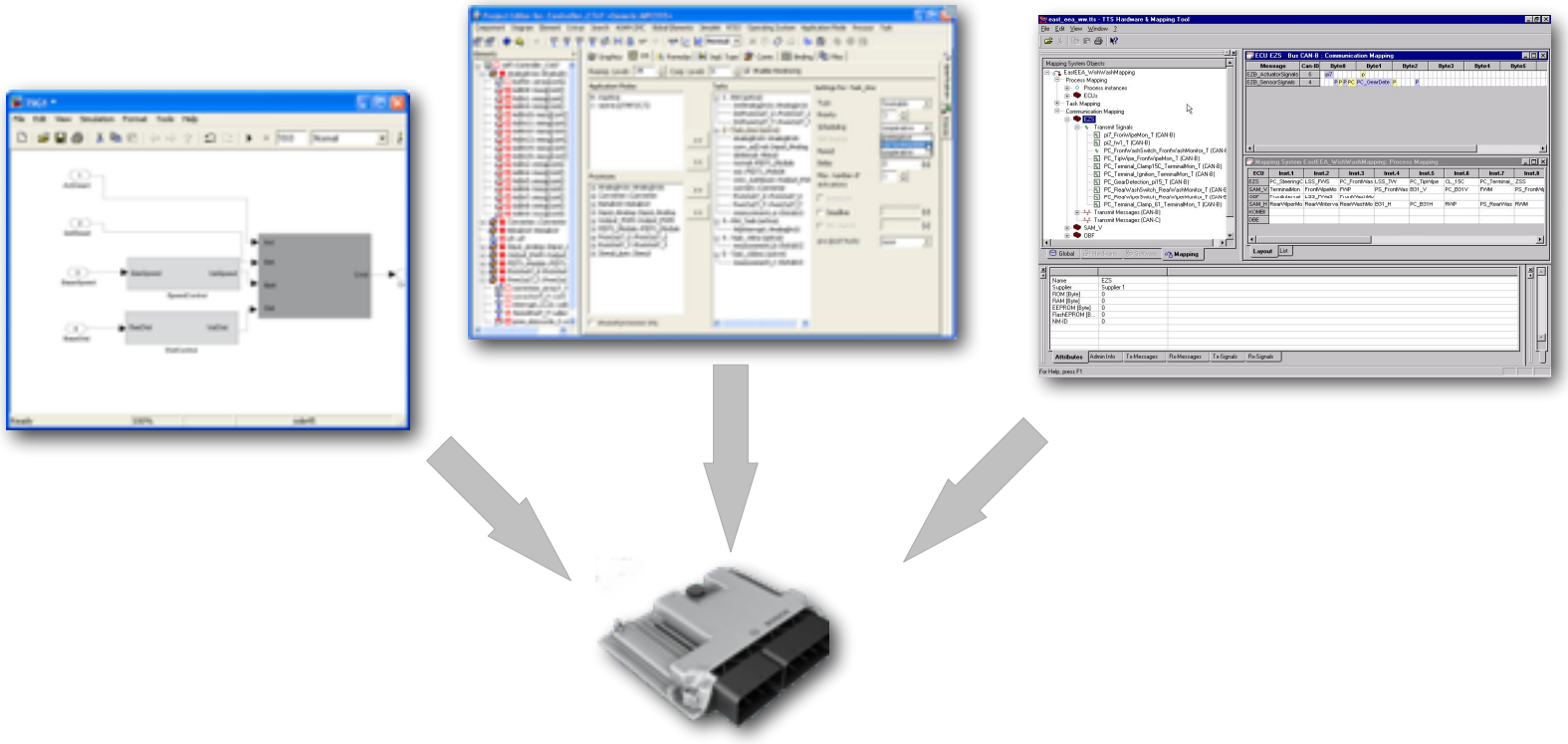
Improvement: Synthesis of models - Optimization of test suit construction

Method: Automatic model construction by generative search

Further examples: Load tests, robustness tests, model validation

Improvement: Improved efficiency of system verification

Example 8: System integration

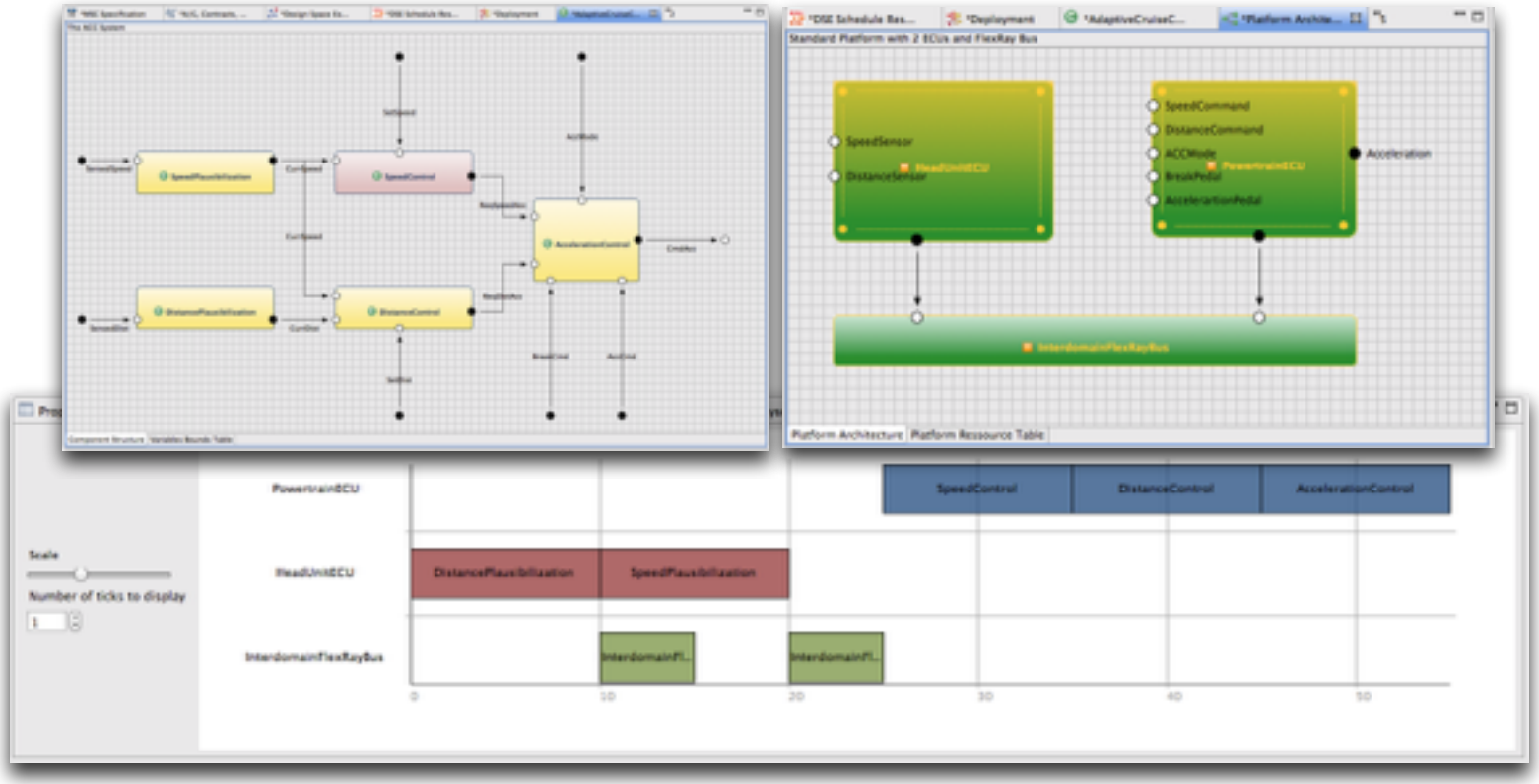


State-of-art: Manual integration steps - Constructed models

Problem: Lacking extensive automatization of design steps

Consequence: Limited support for design space exploration

Synthesis method: Schedule generation



Improvement: Automated development steps - Generated modes

Method: Automated model construction by generative search

Further examples: (Mixed-criticality-)deployment generation, code generation

Improvement: Accelerated exploration of design alternatives

fortiss

March 2015

Clone Analysis in Model-Based Development

McGill NECSIS Workshop 2015

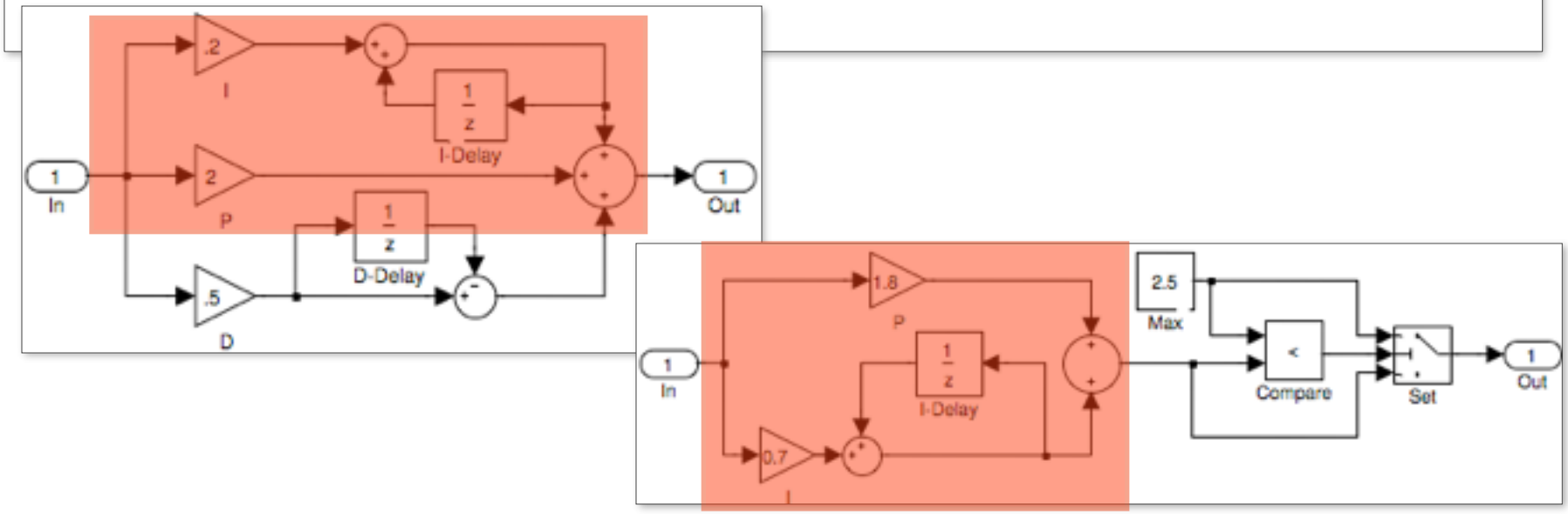
Bernhard Schätz

fortiss GmbH
An-Institut Technische Universität München

What‘s a Clone?

Software clones are segments of code that are similar according to some definition of similarity.

Ira Baxter, 2002

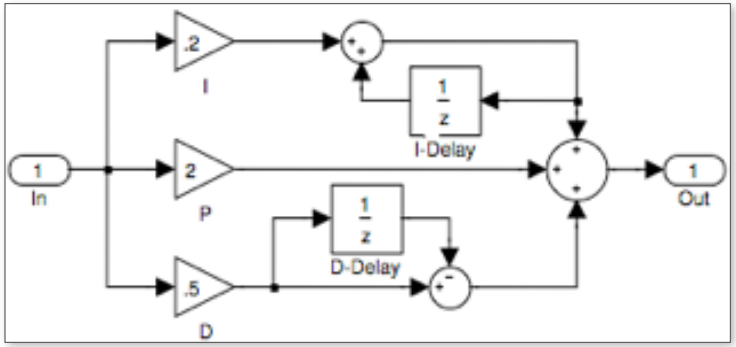


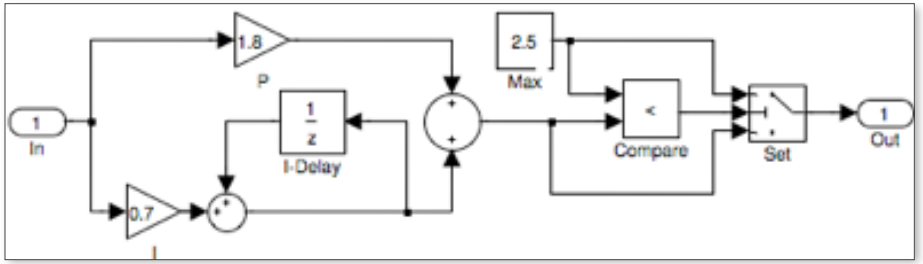
Reusing functionality is good engineering practice

Not being aware of reuse is a dangerous pitfall

Clone analysis detects unwanted forms of reuse

Dataflow Clones

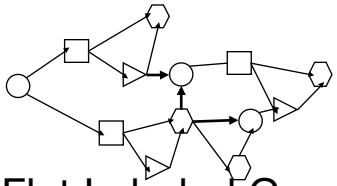




Basic Criteria:

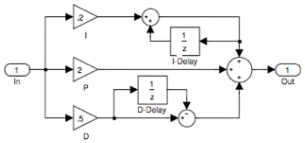
- Functionally independent (Abstracted elements)
- Reusable (Connected)
- Functionally complex (Size of functional elements)
- General (Number of instances)

Clone Detection Pipeline



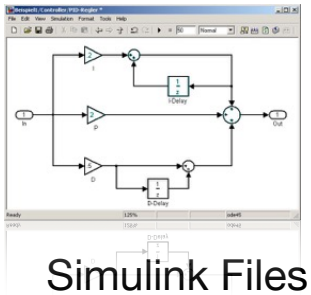
Flat Labeled Graph

Abstraction



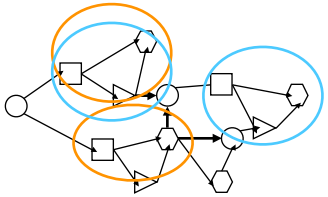
Simulink Models

Simulink Parser



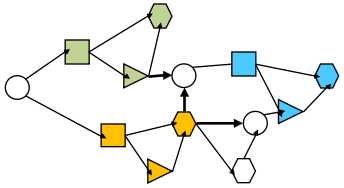
Simulink Files

Detection



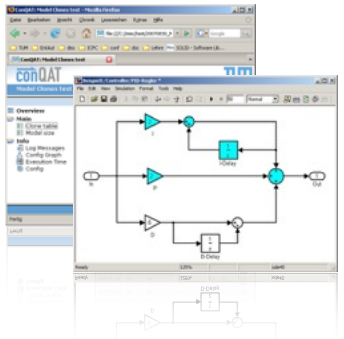
Clone Pairs

Clustering

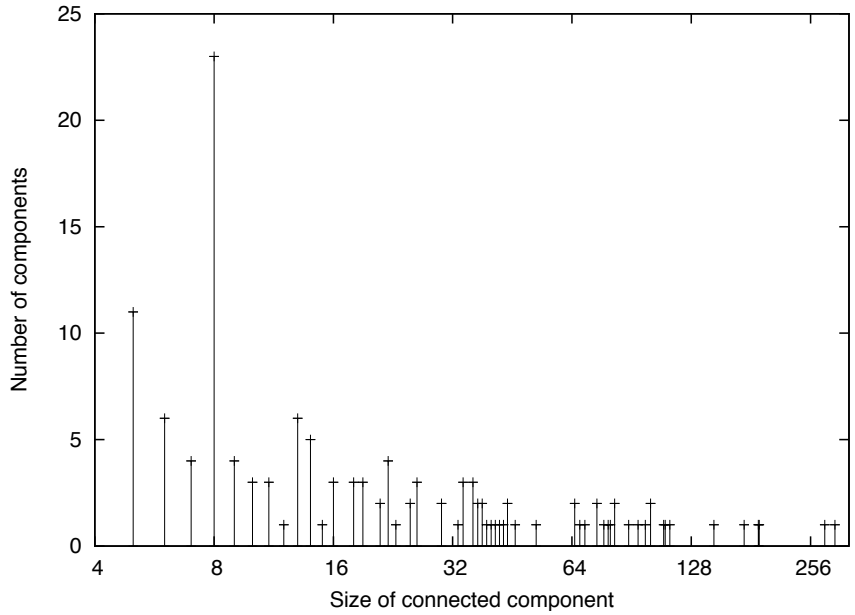


Clone Classes

Visualization



Practical Case Study Results



Simulink/TargetLink Model (ca. 20.000 blocks, 71 files):

- Identified: 139 clone classes after filtering
- Most clones are relatively small & singular/infrequent
- Most clones affect several files/transcend several hierarchies
- Includes clones of library blocks
- 37% of relevant blocks are part of at least one clone class
- Clone elimination substantially reduces models

Practical Case Study Results

Clone Size	Number of Clone Classes
5 - 10	76
11 - 15	35
15 -20	17
> 20	11

Simulink/TargetLink Model (ca. 20.000 blocks, 71 files):

- Identified: 139 clone classes after filtering
- Most clones are relatively small & singular/infrequent
- Most clones affect several files/transcend several hierarchies
- Includes clones of library blocks
- 37% of relevant blocks are part of at least one clone class
- Clone elimination substantially reduces models

Practical Case Study Results

Cardinality of Clone Class	Number of Clone Classes
2	108
3	20
4	10
5	1

Simulink/TargetLink Model (ca. 20.000 blocks, 71 files):

- Identified: 139 clone classes after filtering
- Most clones are relatively small & singular/infrequent
- Most clones affect several files/transcend several hierarchies
- Includes clones of library blocks
- 37% of relevant blocks are part of at least one clone class
- Clone elimination substantially reduces models

Practical Case Study Results

Number of Models	Number of Clone Classes
1	43
2	81
3	12
4	2

Simulink/TargetLink Model (ca. 20.000 blocks, 71 files):

- Identified: 139 clone classes after filtering
- Most clones are relatively small & singular/infrequent
- Most clones affect several files/transcend several hierarchies
- Includes clones of library blocks
- 37% of relevant blocks are part of at least one clone class
- Clone elimination substantially reduces models

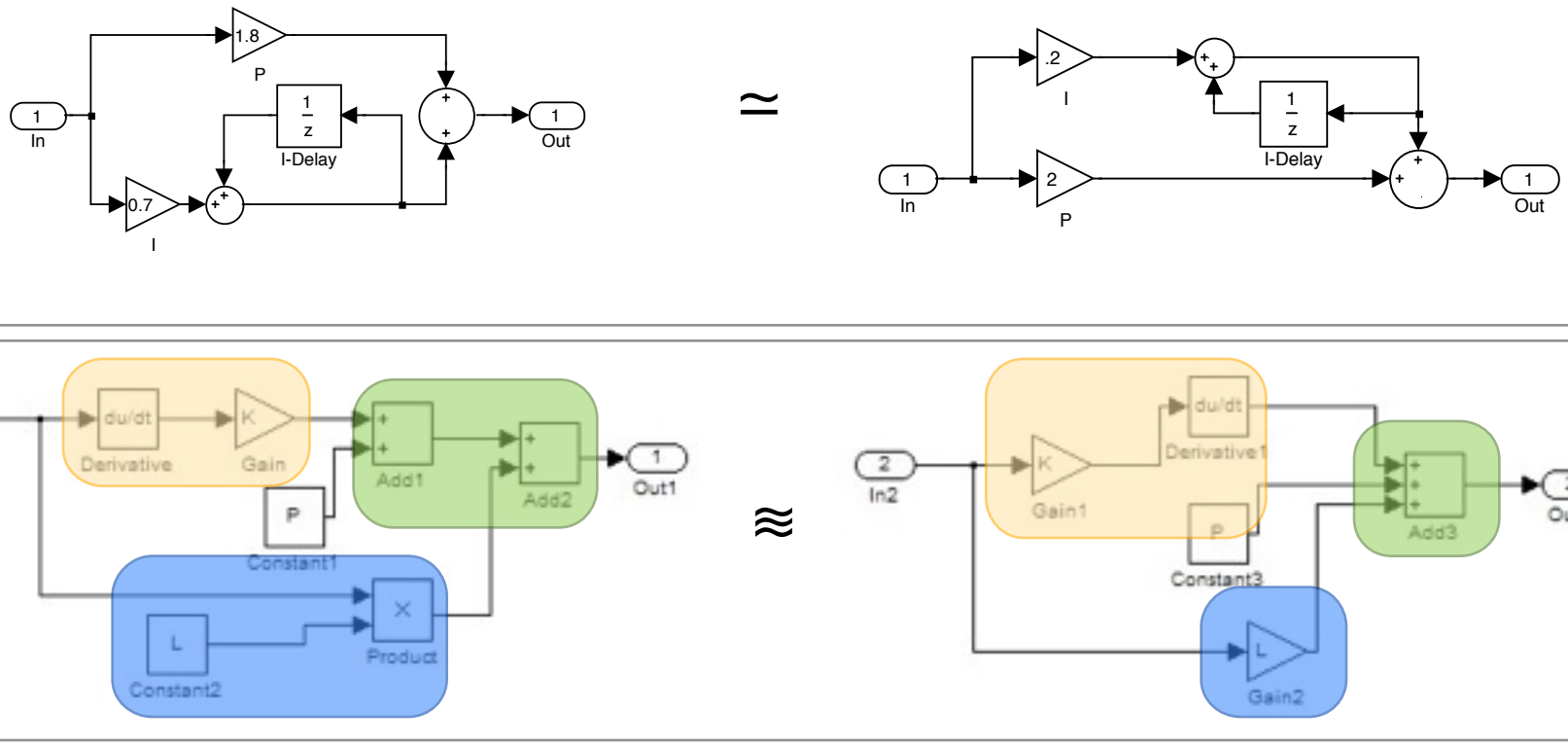
Practical Case Study Results

	Before Elimination		After Elimination	
	# Nodes	#Edges	# Nodes	# Edges
SIM	428	415	387	379
SEM	1741	2029	470	704
ECW	2312	2274	1315	984
AUT	98251	90056	66944	66026

Simulink/TargetLink Model (ca. 20.000 blocks, 71 files):

- Identified: 139 clone classes after filtering
- Most clones are relatively small & singular/infrequent
- Most clones affect several files/transcend several hierarchies
- Includes clones of library blocks
- 37% of relevant blocks are part of at least one clone class
- Clone elimination substantially reduces models

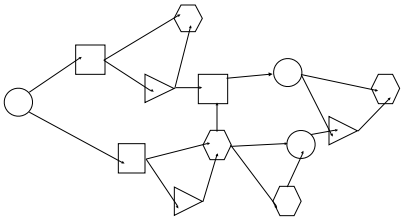
Syntactic vs Semantic Data Flow Clones



Notion of Similarity

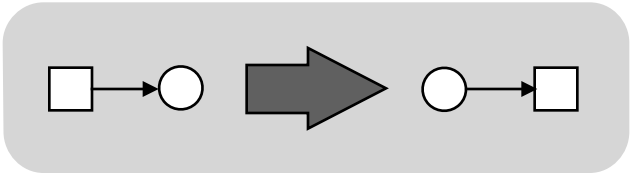
- Syntactic clones (Type 3): Topologically equivalent dataflow (≃)
- Semantic clones (Type 4): Computationally equivalent dataflow (≋)

Semantic Clone Detection Pipeline

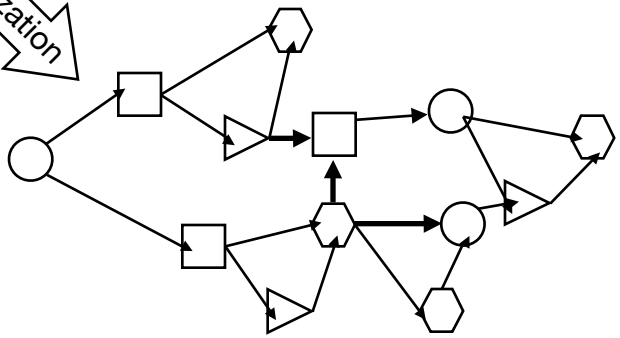


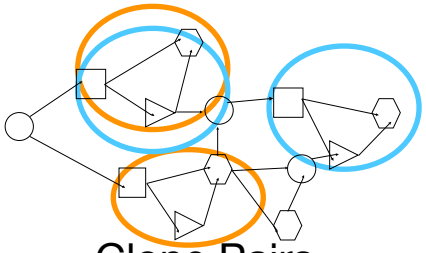
Flat Labeled Graph

Transformation Rule



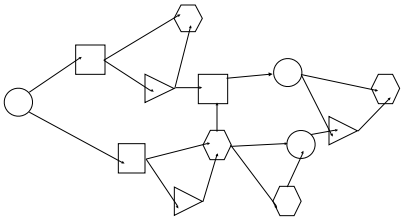
Normalization





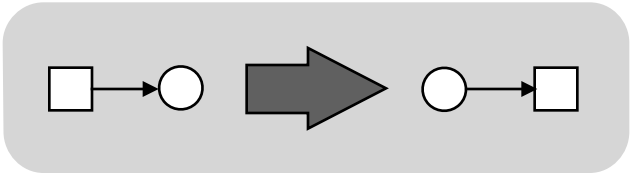
Clone Pairs

Semantic Clone Detection Pipeline

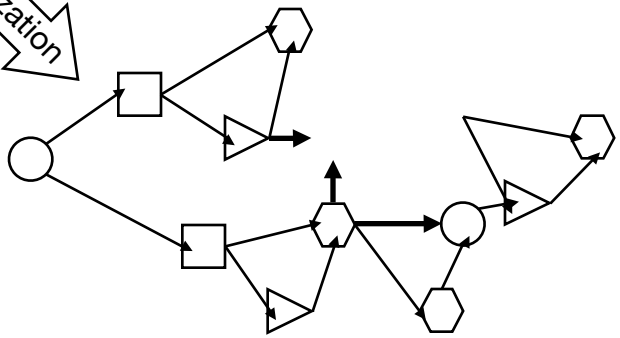


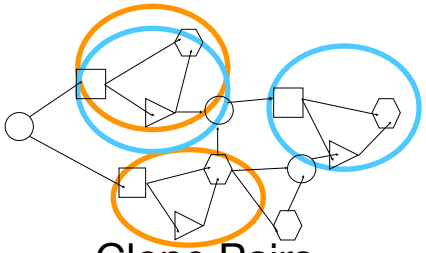
Flat Labeled Graph

Transformation Rule



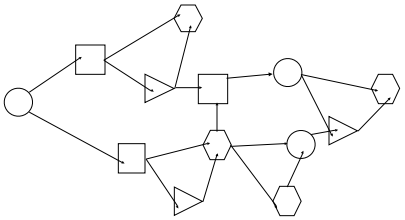
Normalization



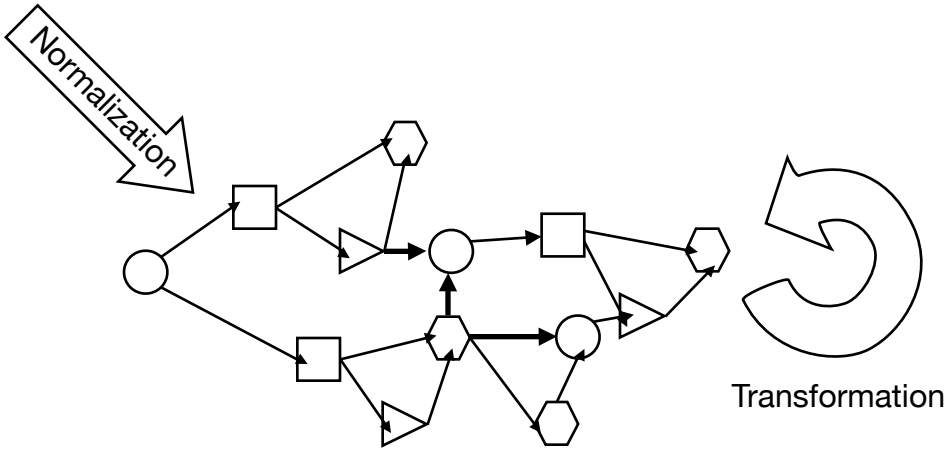


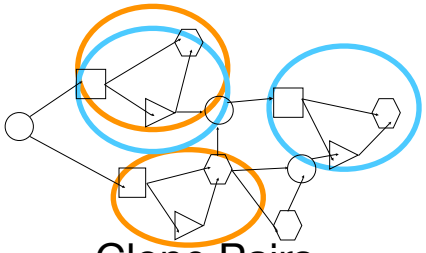
Clone Pairs

Semantic Clone Detection Pipeline



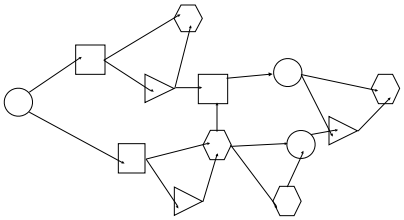
Flat Labeled Graph





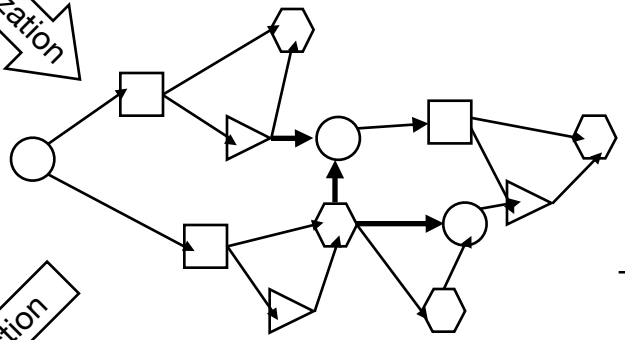
Clone Pairs

Semantic Clone Detection Pipeline



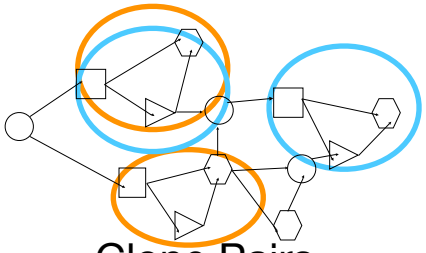
Flat Labeled Graph

Normalization



Transformation

Detection



Clone Pairs

Automotive Case Study Results

Rule	# Executions
Gain for Multiplying by Constant	104
Bias for Adding a Constant	58
Joining Consecutive Product Blocks	42
Joining Consecutive Sum Blocks	40
Placing Gain Block before Integrator Block	28
Joining Consecutive Gain Blocks	16
Sum Rule in Integration	6
Power Rule	5
Distribution of Multiplication over Addition	4
Replacing Comp. to Const. by Comp. to Zero	4
Placing Gain Block before Derivative Block	2
Joining Consecutive Mux Blocks	2
Joining Consecutive Bias Blocks	2
Trigonometric functions	2
Elimination of Rounding Blocks	2
Math functions	2
Replacing Unary Minus Block by Gain Block	2

Normalization of Simulink model (ca. 1400 blocks):
- Substantial transformation: 321 applications of 16 rules

Automotive Case Study Results

Clone Size	Number of Clones	Number of Clones
4-6	46	43
7-10	11	26
11-15	5	7
16-20	0	6
21-30	1	0
> 30	5	5
Total	**68**	**87**

Normalization of Simulink model (ca. 1400 blocks):
- Substantial transformation: 321 applications of 16 rules
- More clones: 87 clones vs. 68 clones

Automotive Case Study Results

Clone Size	Number of Clone Classes	Number of Clone Classes
4-6	31	17
7-10	8	13
11-15	4	4
16-20	0	4
21-30	1	0
> 30	4	4
Average clone	**12,7**	**14,5**

Normalization of Simulink model (ca. 1400 blocks):
- Substantial transformation: 321 applications of 16 rules
- More clones: 87 clones vs. 68 clones
- Larger clones: 14.5 blocks vs. 12.7 blocks

Automotive Case Study Results

Clone Class Cardinality	Number of Clone Classes (without	Number of Clone Classes (with normalization)
2	32	26
3	5	5
4	6	3
5	4	3
6	1	3
7	1	0
8	0	2
9	0	1
Total number of	**49**	**42**

Normalization of Simulink model (ca. 1400 blocks):
- Substantial transformation: 321 applications of 16 rules
- More clones: 87 clones vs. 68 clones
- Larger clones: 14.5 blocks vs. 12.7 blocks
- More frequent clones: 42 classes vs. 49 classes

Automotive Case Study Results

Clone Class Cardinality	Number of Clone Classes (without	Number of Clone Classes (with normalization)
2	32	26
3	5	5
4	6	3
5	4	3
6	1	3
7	1	0
8	0	2
9	0	1
Total number of	**49**	**42**

Normalization of Simulink model (ca. 1400 blocks):
- Substantial transformation: 321 applications of 16 rules
- More clones: 87 clones vs. 68 clones
- Larger clones: 14.5 blocks vs. 12.7 blocks
- More frequent clones: 42 classes vs. 49 classes
- New clones:2 additional classes