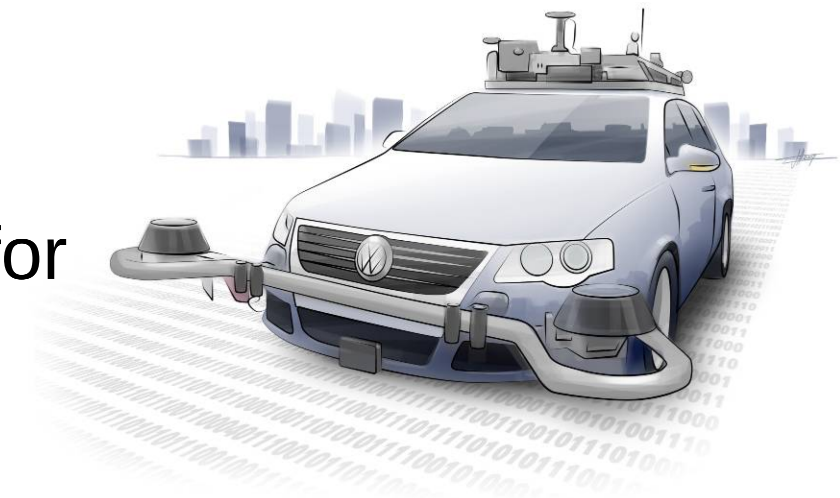


Intelligent Simulations for Quality Assurance

Christian Berger

Contents

- **Software & System Development Process**
  How will we achieve the goal of developing an autonomous vehicle?

- **Design Decisions**
  What do we have to consider?

- **Quality Assurance Activities**
  Are we still on the right way?

Contents

- **Software & System Development Process**
  How will we achieve the goal of developing an autonomous vehicle?

- **Design Decisions**
  What do we have to consider?

- **Quality Assurance Activities**
  Are we still on the right way?

- June 2006. *Hey, there‘s an interesting upcoming competition: Let‘s build an autonomous vehicle.* ☺

Traffic Scenarios

Agile Software Engineering

Requirements

Extreme Programming

Waterfall-Model

V-Model

Configuration-Management

Release-Management

But, how?

Iterative Development Cycles

Quality Assurance

Unit Tests

Real Vehicle Tests

Bug Tracking

Record Raw Data

Tight Schedule

Visualization

Fixed Deadlines

Software & System Development Process ▪ Design Decisions ▪ Quality Assurance Activities

<u>Important basic conditions:</u>

- fixed and tight schedule by DARPA → **NO** re-negotiation
- collaboration of five institutes of the faculties for mechanical and electrical engineering as well as computer science
- vehicle is the bottleneck

<u>Important implications:</u>

- planning backwards
- early integration of hardware and software modules
- early escalation of arising problems

➔ *„Ensure the software is running in the lab before changing it on the vehicle!“*

based on:

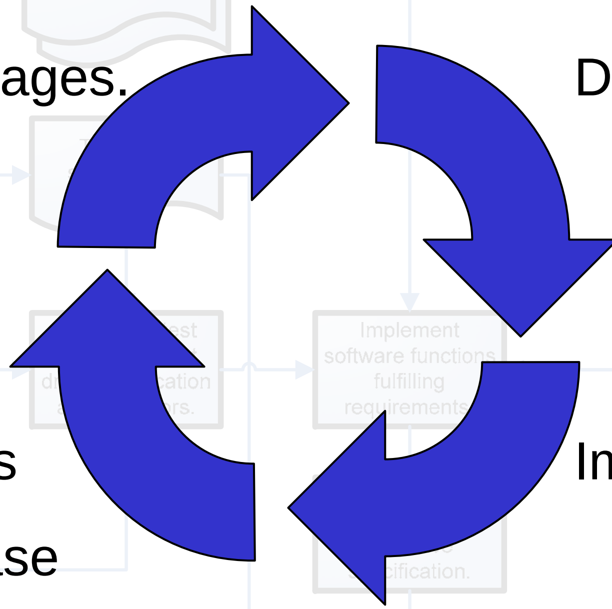
- Beck, K. et al., *Manifesto for Agile Software Development*, http://www.agilemanifesto.org, 2008-04-03.

Software & System Development Process ▪ Design Decisions ▪ Quality Assurance Activities

Main principles behind our iterative development cycle:

Analyse requirements and plan work packages.

Define „Fit Criteria“.



Check requirements fulfillment and release the hard- and software.

Implement functionality.

Fig. 1: Iterative development cycle.

Contents

Engineering principles – used in hardware AND software:

- **DO**s:
  - *prefer simple and robust solutions*
  - *use standards where possible*
  - *modularize tasks and reuse parts*
  - *integrate continuously and release often*
  - *document your changes for traceability*

- **DON'T**s:
  - *reinvent the wheel*
  - *„Premature optimization is the root of all evil.“*

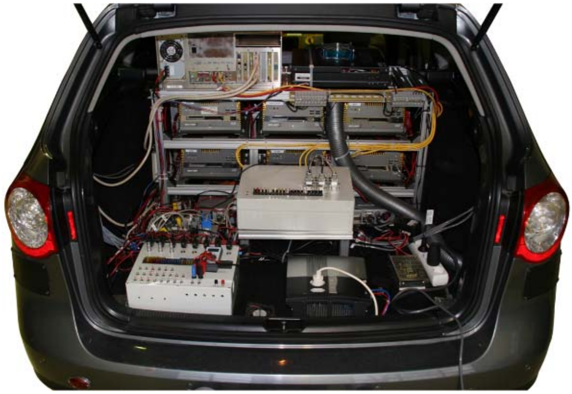


Fig. 2: Caroline's trunk.

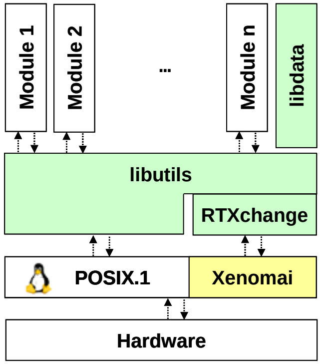


Fig. 3: Caroline's layered software architecture.

based on:

- Broy, M., *Automotive Software and Systems Engineering*, in: Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design, 2005.
- Gamma, E. et al., *Design Patterns. Elements of Reusable Object-Oriented Software.*, Addison-Wesley, 1995.
- Raymond, E., *The Art of UNIX Programming*, Addison-Wesley, 2004.
- Raymond, E., *The Cathedral and the Bazaar,* O'Reilly, 2001.

Main software architecture traits:

- *lean middleware (threading, template module patterns, transparent communication using different media,…)*
- *concurrent thread-safe pipeline concept (parallelized producer/consumer)*
- *separation of concerns (layered modules for abstraction & encapsulation)*
- *composable code (stand-alone applications & linkable libraries)*

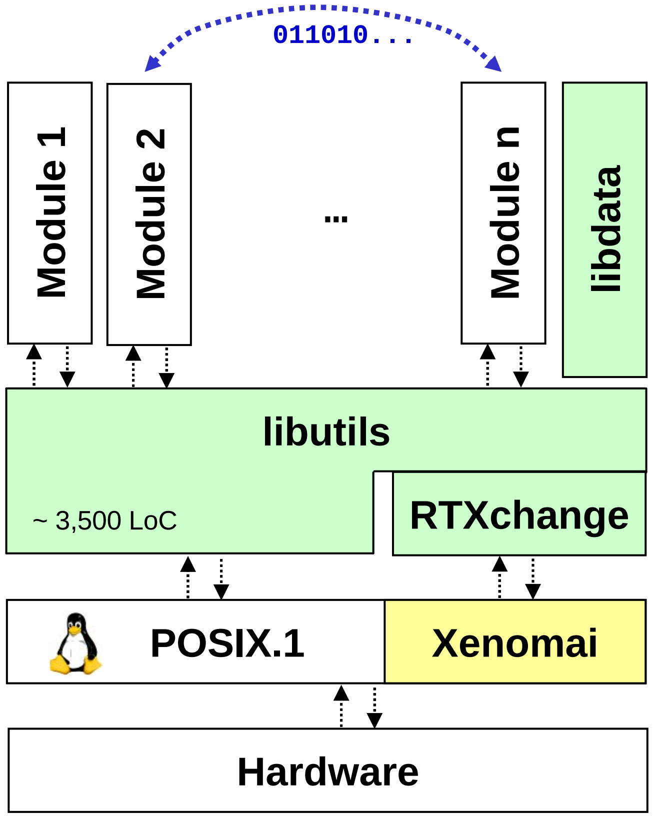


Fig. 4: Caroline's layered software architecture in detail.

based on:

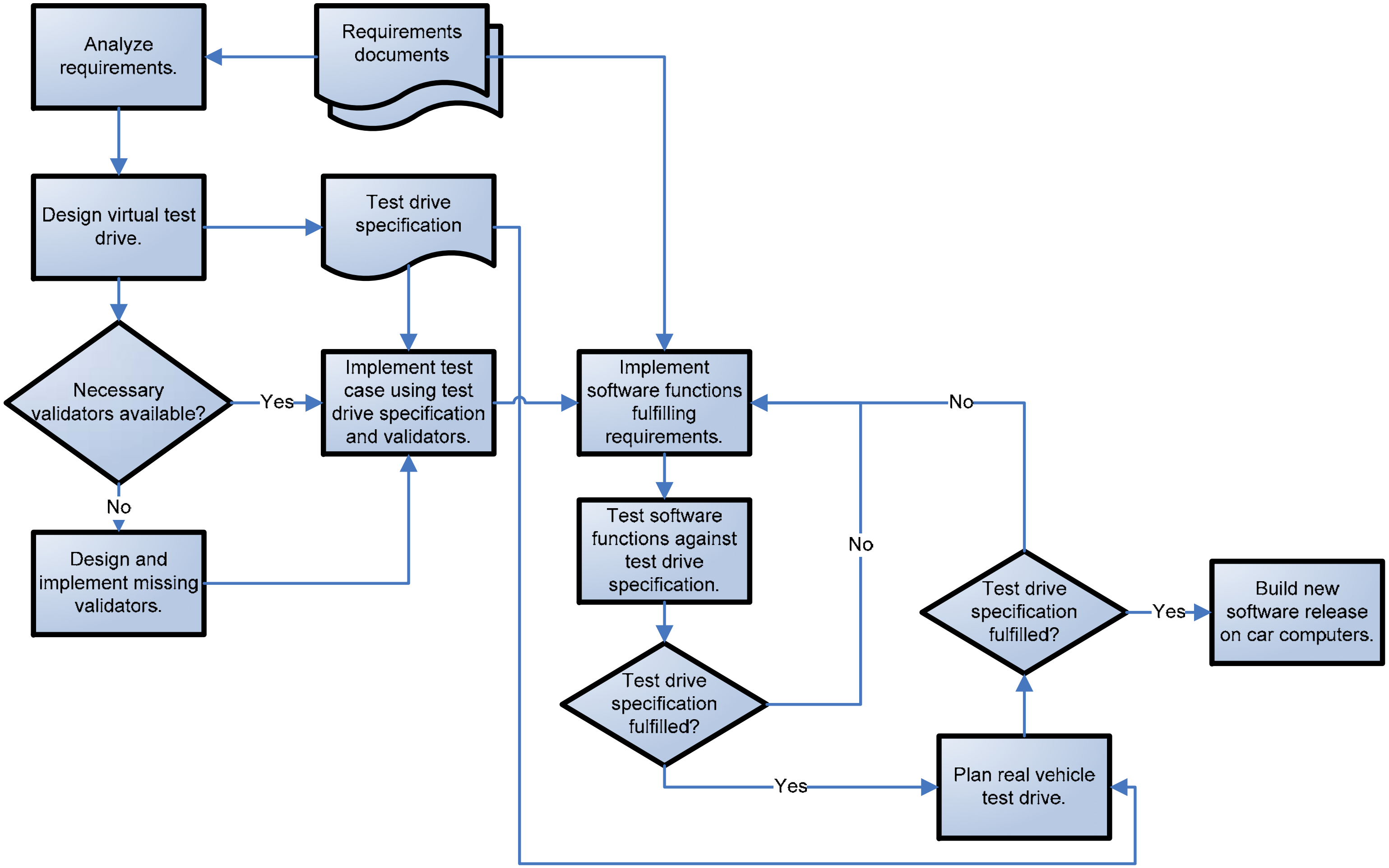
- ISO/IEC 14882:1998.
- ISO/IEC 9945-1:1990.
- Buschmann, F. et al., *Pattern-oriented Software Architecture*, John Wiley & Sons, 2001.
- Dijkstra, E., *Selected writings on computing: A personal perspective*, Springer, 1982.
- Fowler, M., *Writing Software Patterns*, http://www.martinfowler.com/articles/writingPatterns.html, 2007-04-02.
- Hughes, C. and Hughes, T., *Parallel and Distributed Programming Using C++*, Addison-Wesley, 2003.
- Langer, A. and Kreft, K., *Standard C++ IOStreams and Locales.*, Addison-Wesley, 2005.

Contents

- **Software & System Development Process**
  How will we achieve the goal of developing an autonomous vehicle?
- **Design Decisions**
  What do we have to consider?
- **Quality Assurance Activities**
  Are we still on the right way?

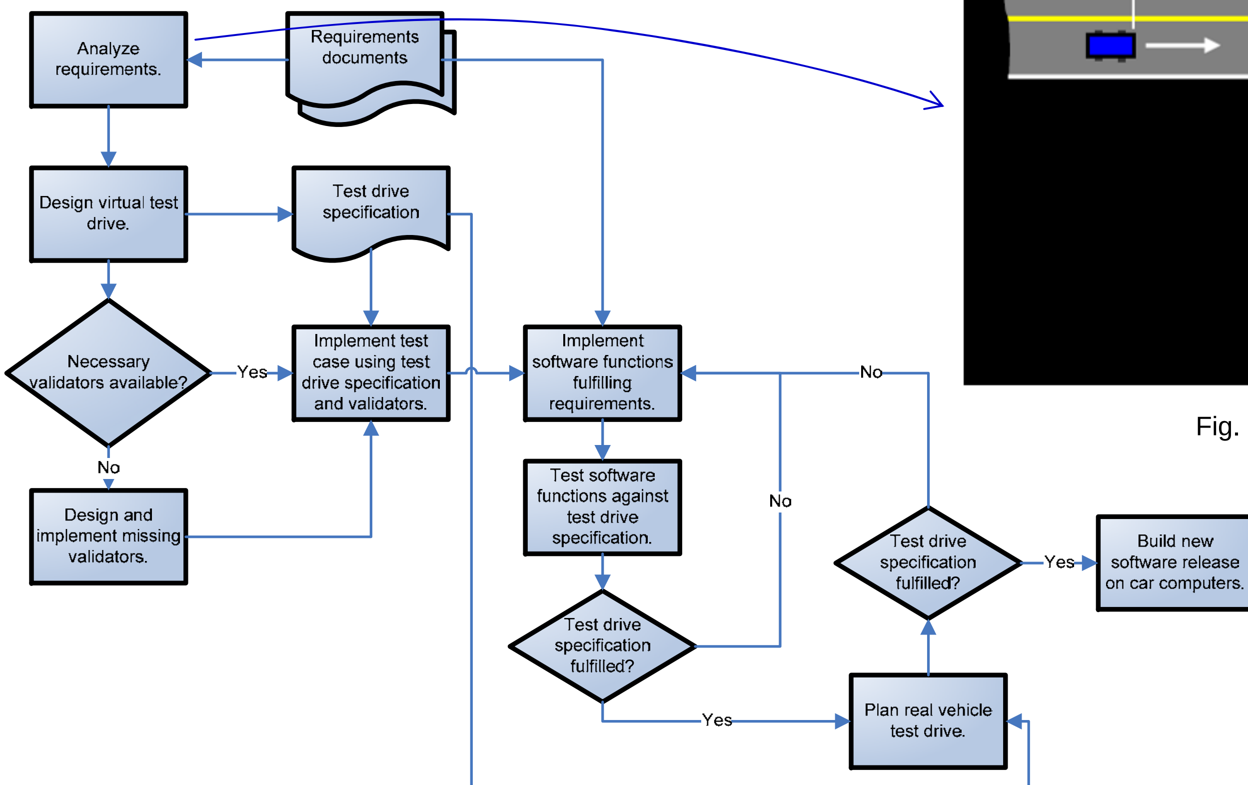
Software & System Development Process ▪ Design Decisions ▪ Quality Assurance Activities

Overview of an iterative development cycle:



Setting up system simulation:

Analyze requirements.



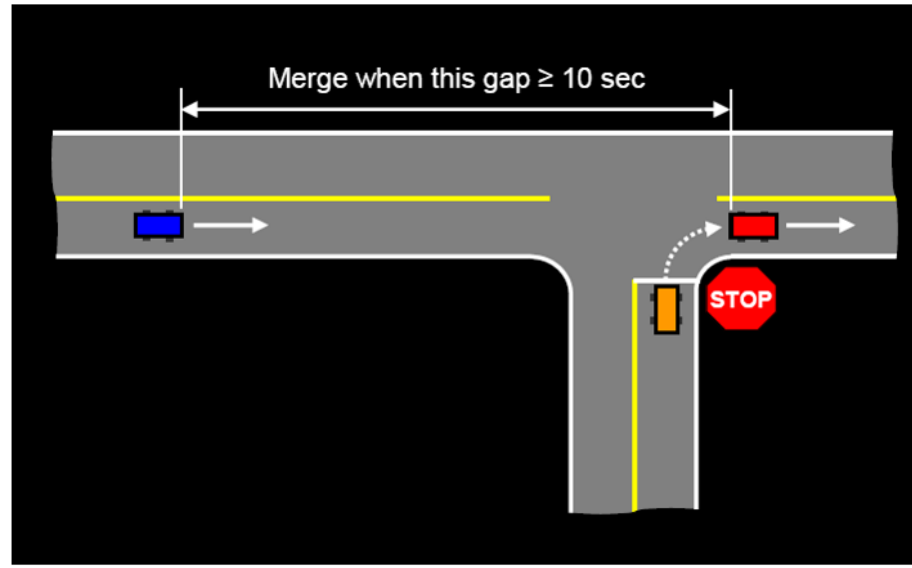
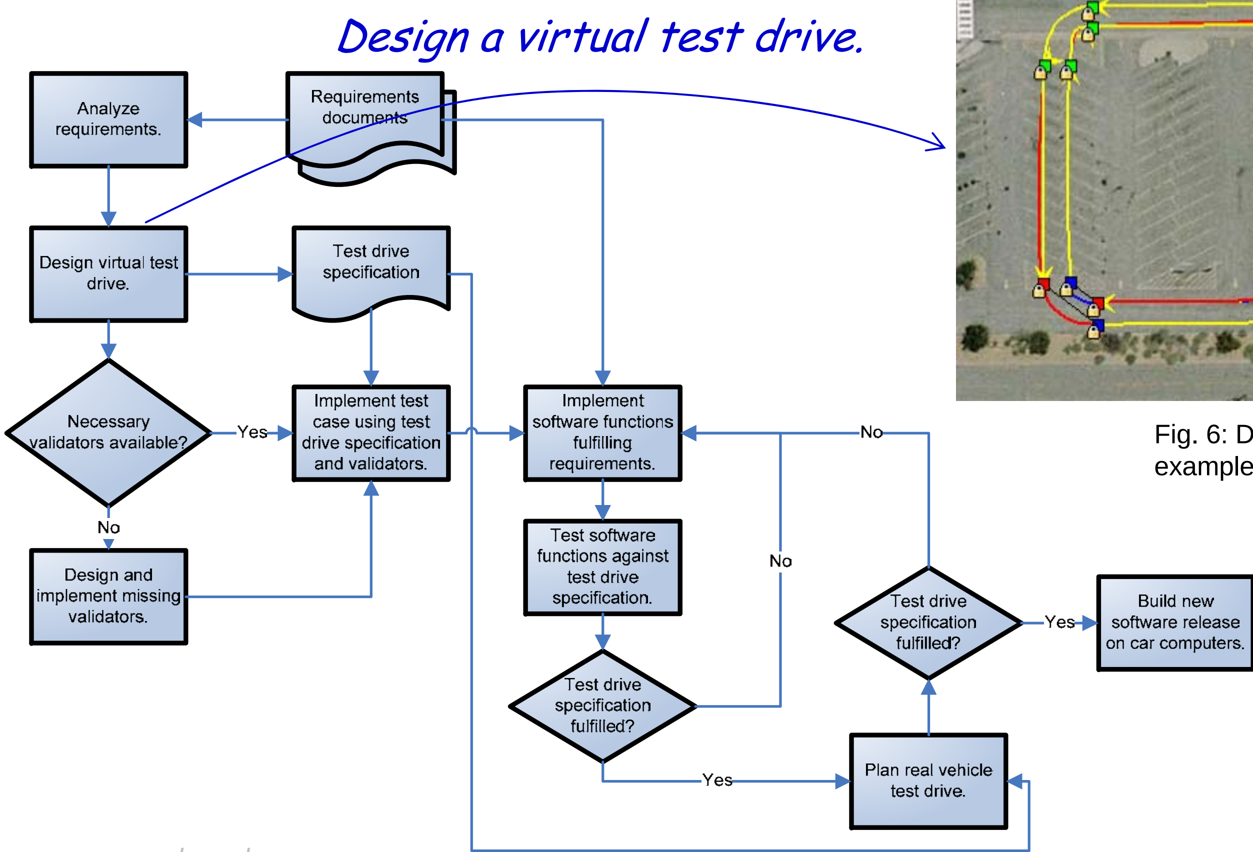


Fig. 5: Requirement set by DARPA.

based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:



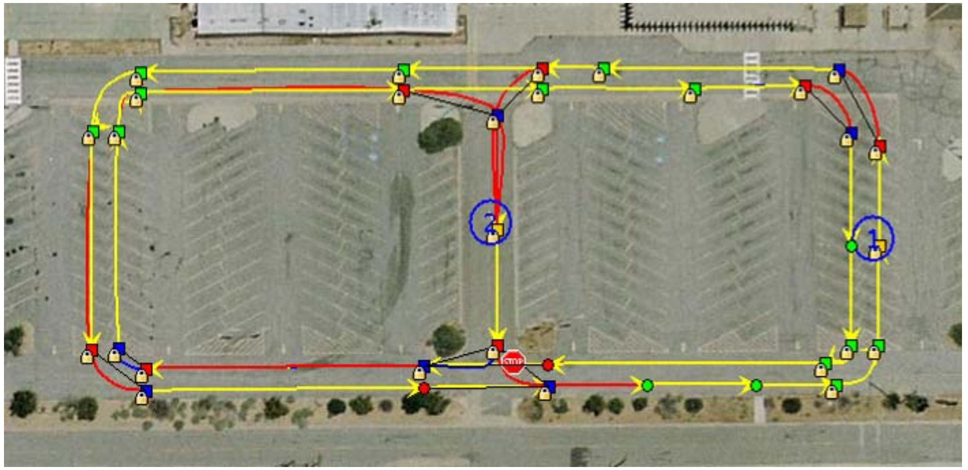


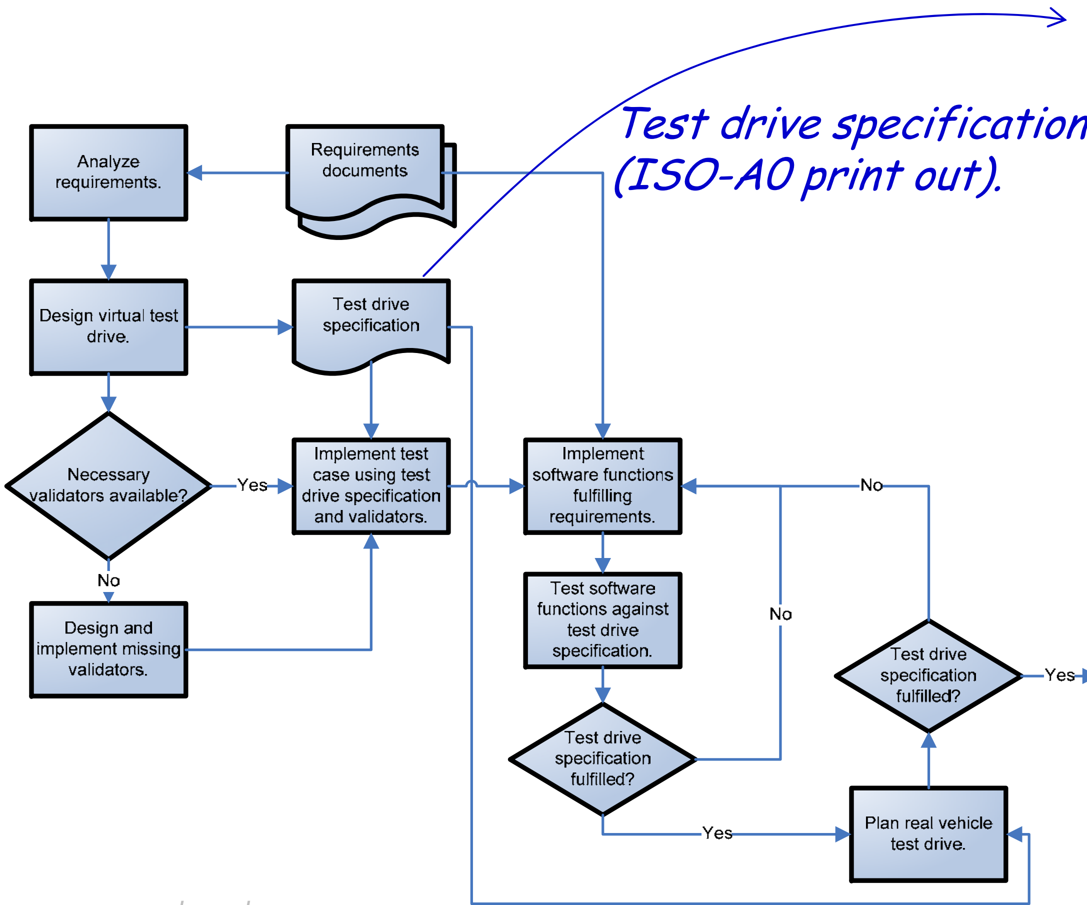
Fig. 6: Design virtual test drive (RNDF+MDFs). In this example: Test area A.

based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Software & System Development Process ▪ Design Decisions ▪ Quality Assurance Activities

Setting up system simulation:



CarOLO – Übersicht StoryCards (MS13-SC01, MS13-SC02 und MS13-SC03)

Beschreibung:

Meilenstein 13

Termin 13. März 2007

Titel **Kameragestützte Fahrt auf Rundkurs, sensorgestütztes Ausweichen und E-Stop**

Ort Campus Süd

Beschreibung Vorraussetzung ist das Bestehen des vorigen Szenarios!
- Caroline fährt autonom mit einer Geschwindigkeit von 8 – 15 km/h kameragestützt in der äußeren Fahrspur den kompletten Rundkurs, ohne die Fahrspur zu verlassen.
- Caroline fährt autonom mit einer Geschwindigkeit von 08 – 15 km/h ohne Fahrspurerkennung nach RNDF auf ein Hindernis zu und weicht diesem aus.
- E-Stop funktioniert gemäß Anforderungen

Modul	#	MS	Beschreibung	Verantwortlich	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So
					9.3	10.3	11.3	12.3	13.3	14.3	15.3	16.3	17.3	18.3
Aktorik	AK-1	8	Regler muss stabil min. 8km/h fahren können.	Wille										
	AK-2	8	Regler muss Kurven stabil abfahren können.	Wille										
	AK-3	8	Trajektorienplanung funktionsfähig	Wille										
	AK-4	8	Falls keine Trajektorie vorhanden ist → Notbremsung	Wille										
API	AP-1	8	alive()-Funktion in den Applications werden durch Scooby abgefragt. Die Messungen müssen geloggt werden und Abweichungen errechnet werden.	morgenroth										
CarOLOVisA	CV-1	8	Darstellung EgoPosition.	lipski										
	CV-2	8	Darstellung RNDF/Roadgraph.	lipski										
	CV-3	8	Darstellung LaneNetworkObject.	lipski										
	CV-4	8	Darstellung Trajektorien.	lipski										
Digitalmap	DM-1	8	Erkennen, dass ein USB-Stick eingesteckt worden ist und Abfragen sowie Verarbeiten der Daten darauf.	nothdurft										
	DM-2	8	Eintragen der Hindernisse in VisA.	nothdurft										
	DM-3	13	Konstruktion der iMap: Interpretation der Belegungskarte und Weitergabe von Fahranweisungen (in Form von Votes?) an die KI	nothdurft										
E-Stop	E-1	9	Durchführung und Erläuterung E-Stop inkl. Video-Aufnahmen mit HBK am 09.03.	cornelsen										
	E-2	9	Wireless E-Stop	cornelsen										
Infrastruktur	I-1	8	Erstellen einer Liste, welcher Service auf welcher IP, Rechner (Seriennummer und Bezeichnung) läuft.	doering										
	I-2	8	Erstellen einer effizienten Backup- und Restore-Lösung.	doering										
	I-3	9	Errichtung Headquarters am Campus Süd (Tische, Stühle, WLAN, Steckdosen, etc.).	doering										
IMU	IMU-1	8	Anbindung Odometer.	nothdurft										
	IMU-2	8	GPS-Zeit steht über NTP an allen Rechnern zuverlässig zur Verfügung.	nothdurft										
KI	KI-1	8	RNDF Campus Nord erstellen, wobei keine Waypoints zwischen den Fahrspurmarkierungen liegen.	homeier										
	KI-2	8	MDF für RNDF erstellen, um eine Runde im Norden zu beginnen, anschließend gegen den Uhrzeigersinn um die Halle herumzufahren.	homeier										
	KI-3	8	LaneBehaviour erstellen, um auf LaneNetworkObject-Daten gestützt Trajektorien herauszugeben.	homeier										
	KI-4	8	MinSpeedBehaviour erstellen, um nicht unter die Mindestgeschwindigkeit zu fallen.	homeier										
	KI-5	8	Beim Beenden des Kurses soll Caroline regulär Anhalten.	homeier										
	KI-6	8	FollowWaypointBehaviour erstellen.	homeier										
	KI-7	13	AvoidBehaviour erstellen zum Ausweichen statischer Hindernisse.	homeier										
Q	Q-1	8	Durchführung Fahrzeugtest am 08.03.07.	guelke										
	Q-2	13	Durchführung Fahrzeugtest am 13.03.07	guelke										
Simulator	S-1	8	Die Broadcaster senden aktuell sequenziell, müssen aber parallel senden.	basarke										
	S-3	6	TimeOutValidator.	basarke										
	S-4	6	ArrivalValidator.	basarke										
	S-5	6	StayInLaneValidator.	basarke										
	S-6	6	ChangeLaneValidator.	basarke										
	S-7	6	FollowWaypointValidator.	basarke										
	S-8	6	PassCheckpointValidator.	basarke										
	S-9	6	SpeedValidator.	basarke										
	S-10	13	CorrectLaneValidator (rechts halten).	basarke										
	S-11	13	MinimumDistanceValidator	basarke										
Vision	VI-1	8	Kalibrierung der zu verwendenden Kamera.	lipski										
	VI-2	8	Inbetriebnahme des CarPCs im Fahrzeug.	lipski										
	VI-3	8	Parametrierung des LaneProcessors an die Kamera und örtlichen Gegebenheiten (Kopfsteinpflaster).	lipski										
	VI-4	8	Befüllung des LaneNetworkObjects mit den Daten des Algorithmus der Fahrspurerkennung.	lipski										
	VI-5	8	Kontinuierlicher Versand von LaneNetworkObjects an die KI und CarOLOVisA.	lipski										

0 = Nicht angefangen, 1 = Konzipiert, 2 = Implementiert, 3 = Getestet im Testbett, 4 = Im Fahrzeug in Betrieb genommen, 5 = 100% erfolgreich im Fahrzeug getestet

Fig. 7: ISO-A0 print out of test drive specification.

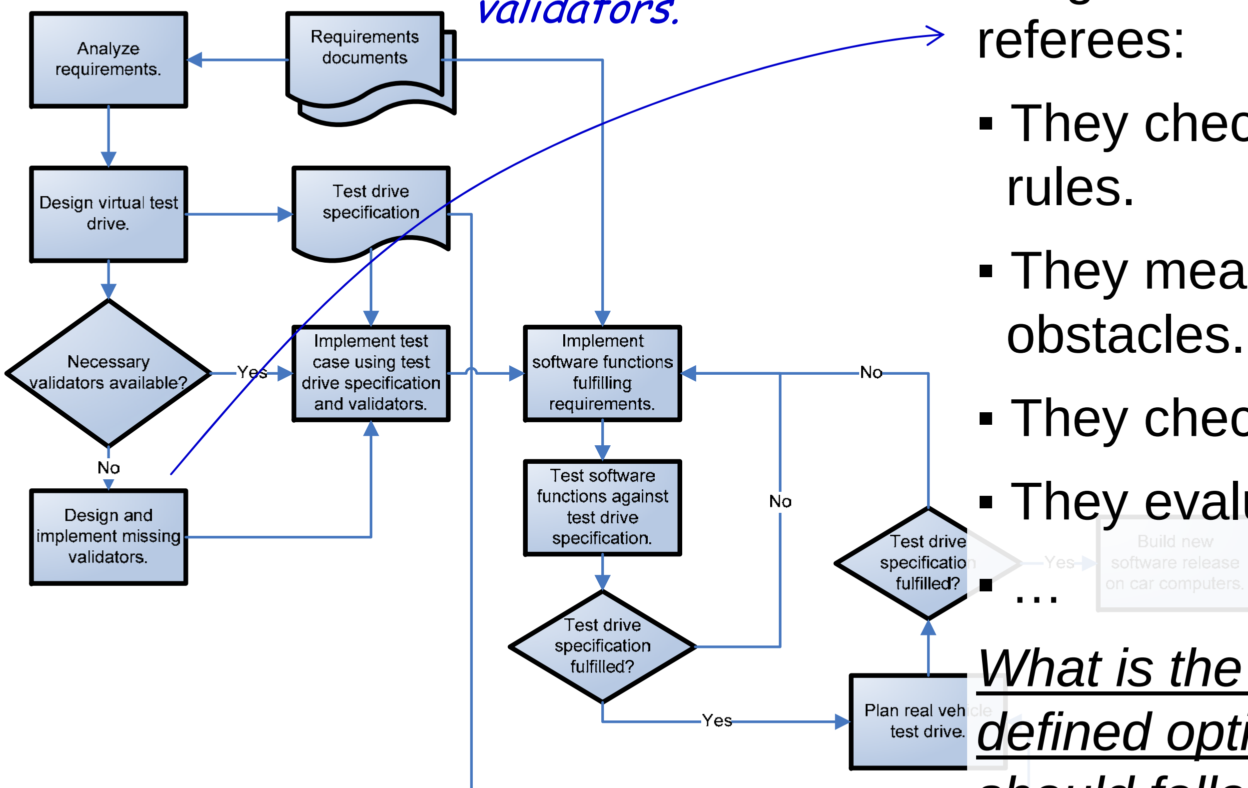
based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Software & System Development Process ▪ Design Decisions ▪ **Quality Assurance Activities**

Setting up system simulation:

Design and implement validators.



What are validators?

Imagine they are *virtual* DARPA referees:

- They check compliance with traffic rules.
- They measure distances to obstacles.
- They check timings.
- They evaluate using penality points.
- …

What is the difference to an a priori defined optimal route the robot should follow?

based on:

- Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:

Imagine the following situation:

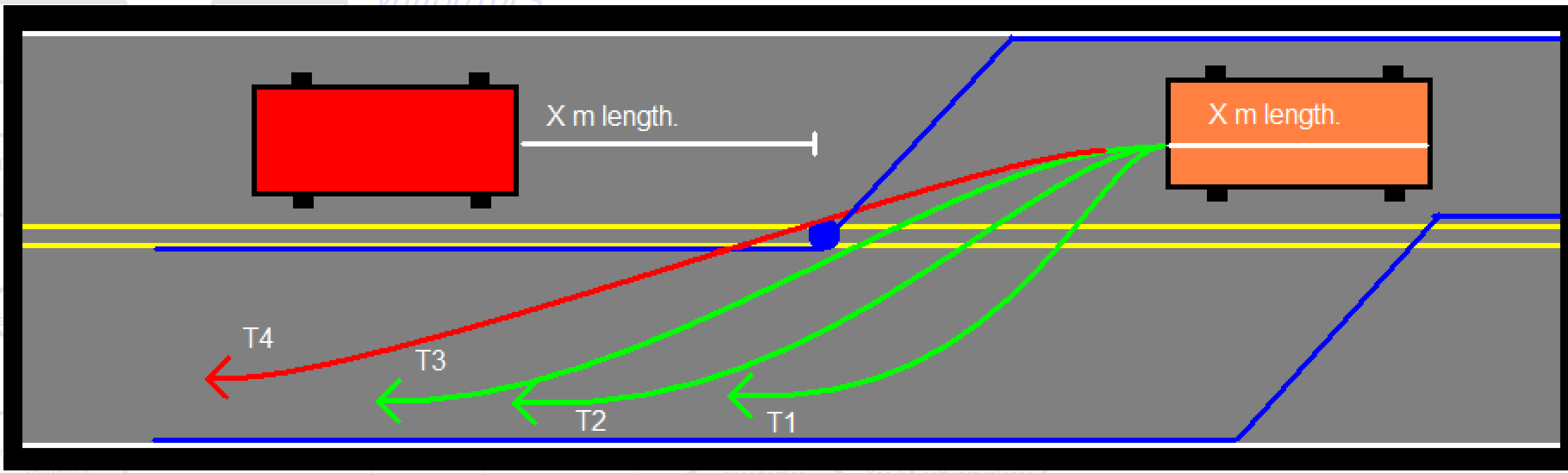


Fig. 8: Passing a stationary obstacle.

Even if T3 is the optimal route due to the smallest lateral forces, T1 and T2 are allowed as well. This example could alternatively checked using thresholds (indicated by blue lines).

based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:

But, imagine another situation:

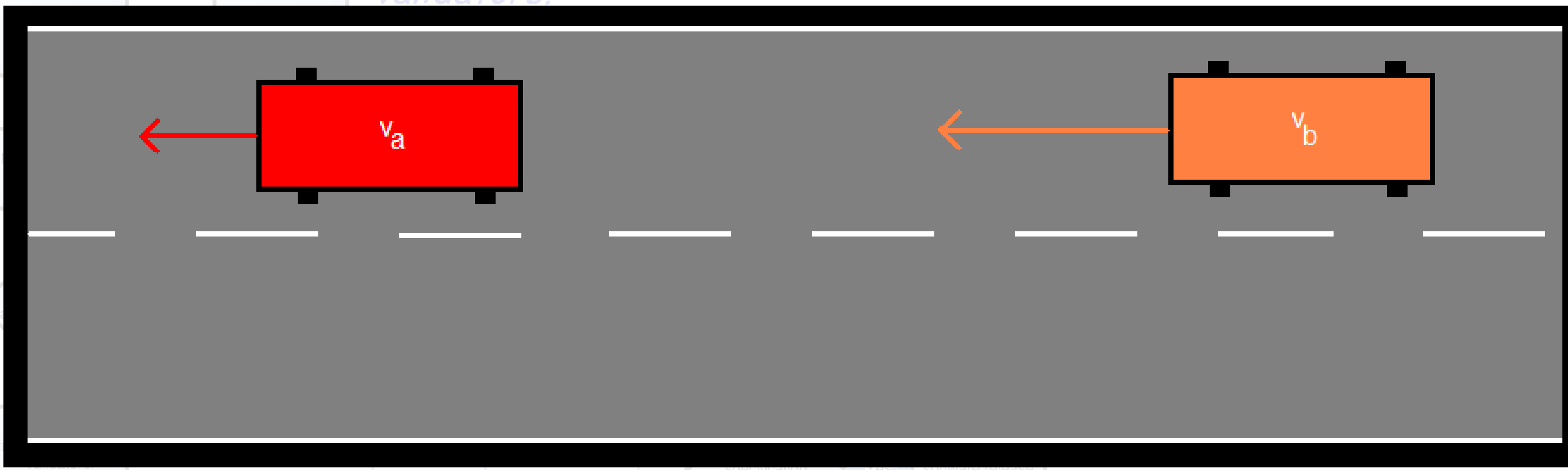


Fig. 9: Following or passing another vehicle.

In this example, $v_a < v_b$. Thus, the robot can either overtake the other vehicle, simply follow the car or choose a completely different route by dynamic re-planning. In this case, many solutions are suitable.

based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:

Validators…

- can be used to check compliance with a given set of constraints Γ (timing, geometrical relations, logical constraints, …).
- can be applied interactively or unattendedly.
- hold: $\forall \upsilon \in \Psi_S$: `!failed`($\upsilon$) after a system simulation S in case of no error, where Ψ is the set of validators applied to S and `failed`: $\Psi \rightarrow$ `bool` is a method checking for any penalties for evaluation.
- are from the technical point of view a reporting interface that can be used online during a running system simulation or offline by post-processing dumped simulation data.

based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:
„Intelligent Simulations"…

Here is the intelligent part of the naïve simulation component.

- need a kind of *intelligent* components. Imagine following scenario:

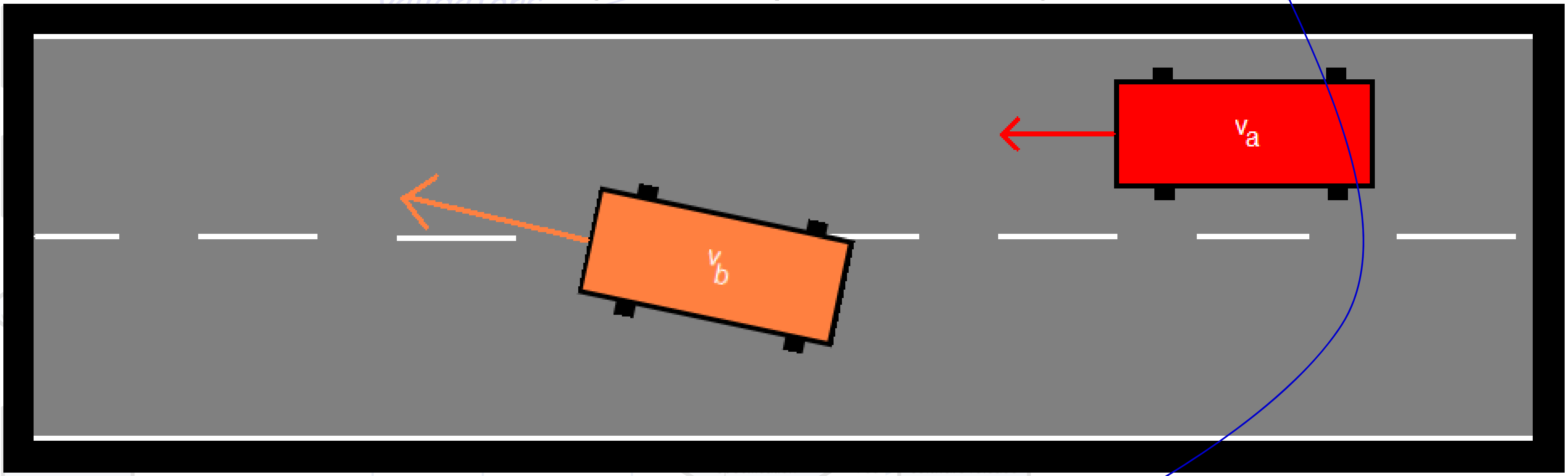


Fig. 10: Robot has passed dynamic vehicle A.

- Robot B has passed vehicle A. A should follow its own behavior rule set but must consider the behavior of robot B: It must not collide and must switch to follow-mode *if* necessary.

based on:

▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:

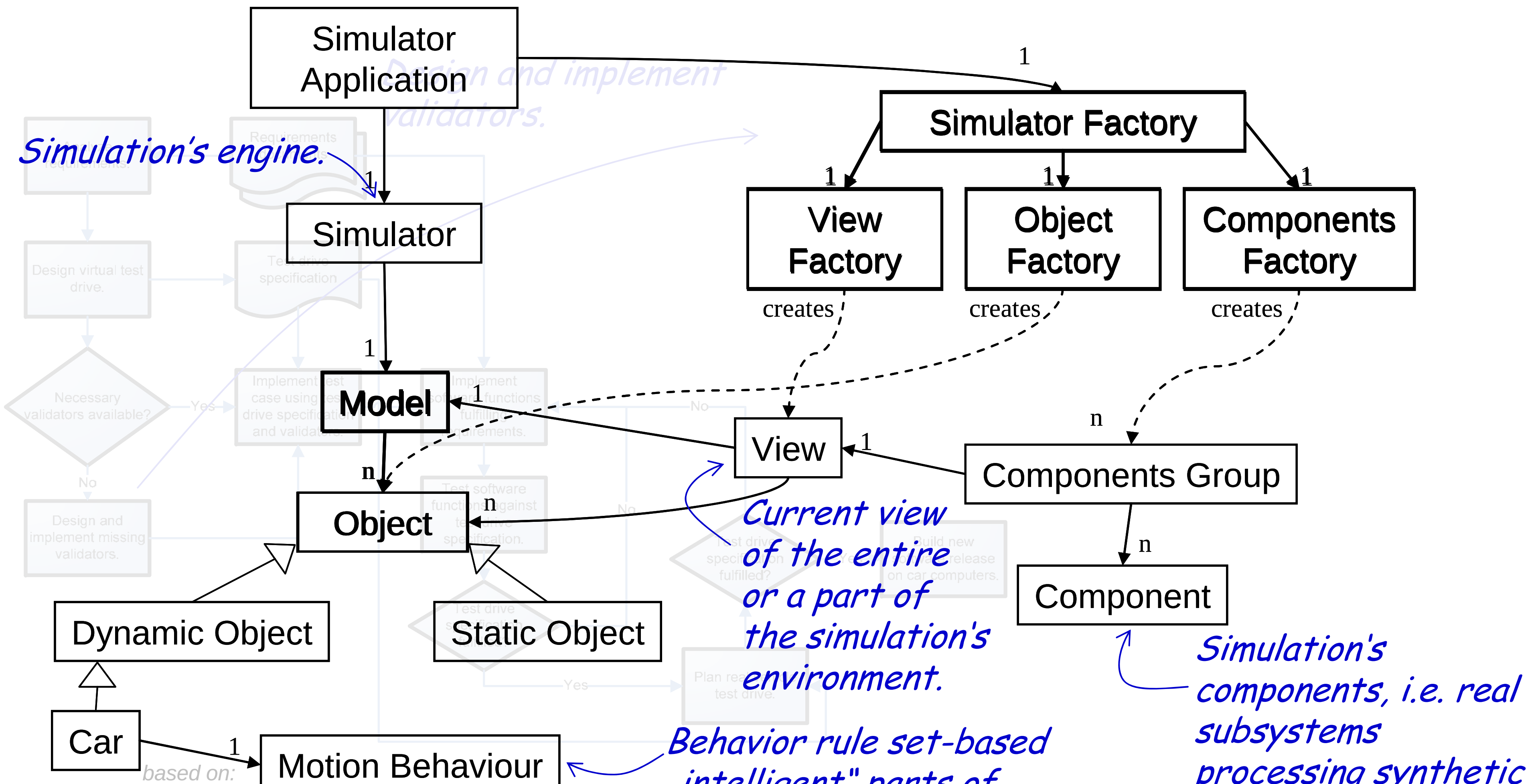
Intelligent components…

- follow primarily their default behavior rule set (e.g. arrive at intersection when the robot stopped for at least five seconds at its own stop line).
- must consider the behavior of other elements if they are part of a long-term system simulation.
- should be used very wisely for setting up system simulations with reproducible results → decompose traffic scenario into smaller parts.

based on:

- Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

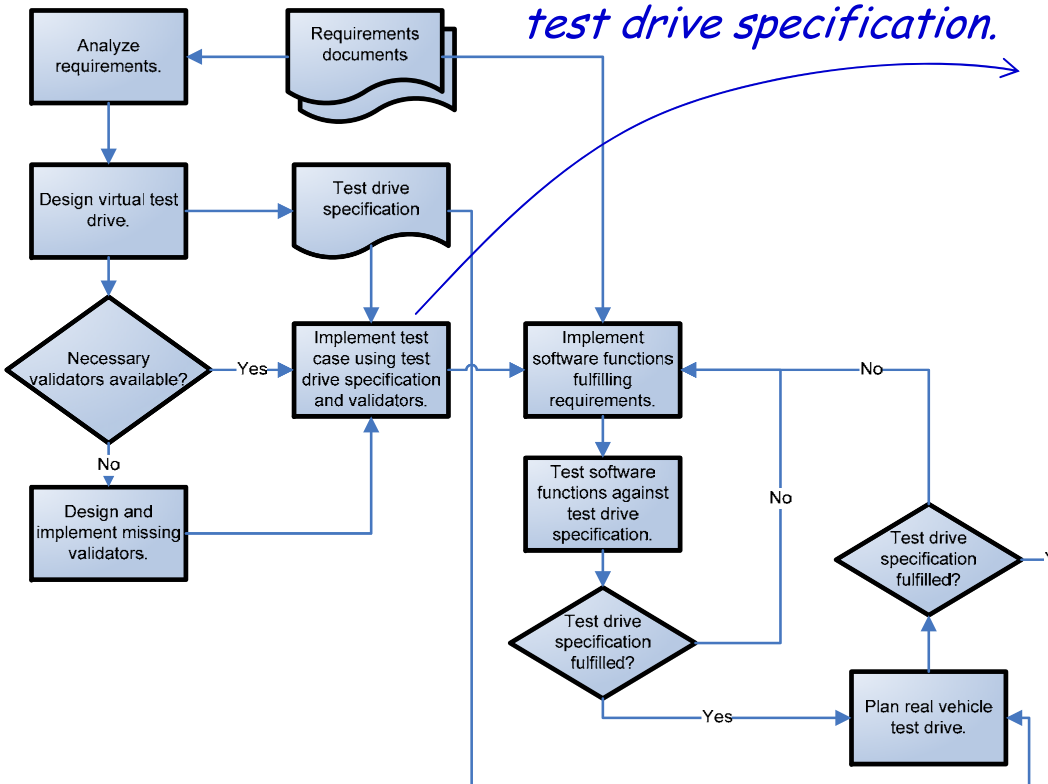
Components of a system simulation:



▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the "virtual vehicle"*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

Setting up system simulation:

Putting everything together: Executable test drive specification.




```
// Prepare "USB stick" for getting RNDF and MDF.
Acceptance_USBStickService *usbstick = new
       Acceptance_USBStickService(sRNDF.str(), sMDF.str(), 1);
TS_ASSERT(!usbstick->isRunning());

// Create AI service.
Acceptance_AIService *aiservice = new
       Acceptance_AIService(m_argc, m_argv);
TS_ASSERT(!aiservice->isRunning());

// Create Simulator service.
Acceptance_SimulatorService *simservice = new
       Acceptance_SimulatorService(m_argc, m_argv);
TS_ASSERT(!simservice->isRunning());
////////////////////////////////////////////////////////////
// Timeout validator.
Validator_TimeOut *valTimeOut = new Validator_TimeOut(2350);
valTimeOut->startService();
simservice->getSimulatorObjectValidatorObserver()
       ->addValidator("SimulatorView-0", valTimeOut);

// Checkpoint validator.
Validator_PassCheckpoint *valPassCheckpoint = new
       Validator_PassCheckpoint(sMDF.str(), sRNDF.str(), 2);
simservice->getSimulatorObjectValidatorObserver()
       ->addValidator("SimulatorView-0", valPassCheckpoint);
// Start simulation.
simservice->startService();
...
```

Sort of requirements- and constraints-DSL.

based on:

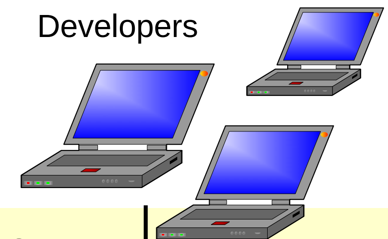
- Basarke, C. et al., *Design and quality assurance of intelligent vehi* Automotive Technology Conference, 2007.

Software & System Development Process ▪ Design Decisions ▪ **Quality Assurance Activities**

Setting up system simulation:

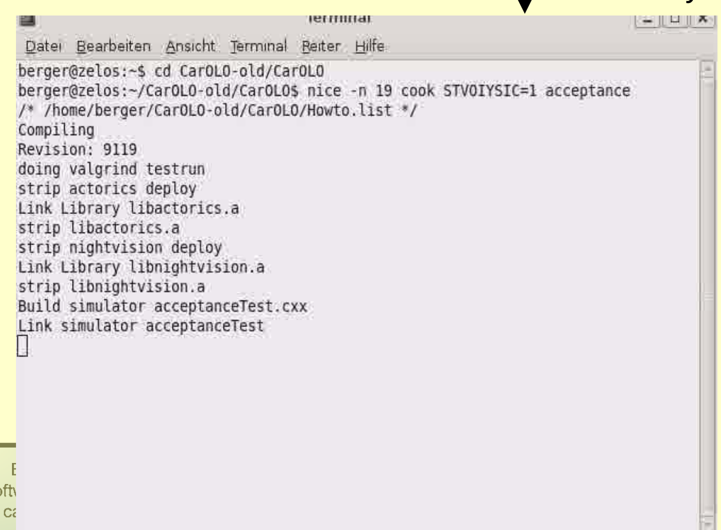
Now, every developer can work independently and safely implementing the software function for the required traffic scenario.

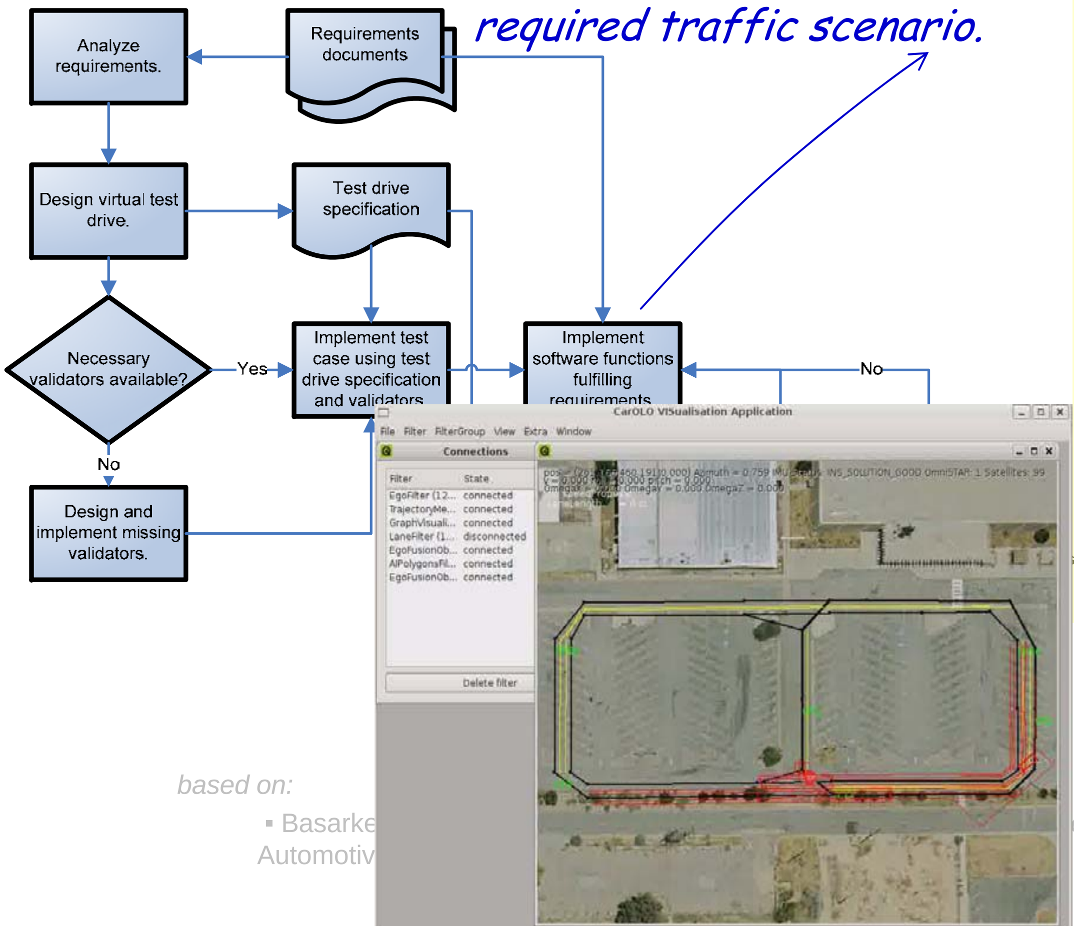
Developers



System simulation

run interactively





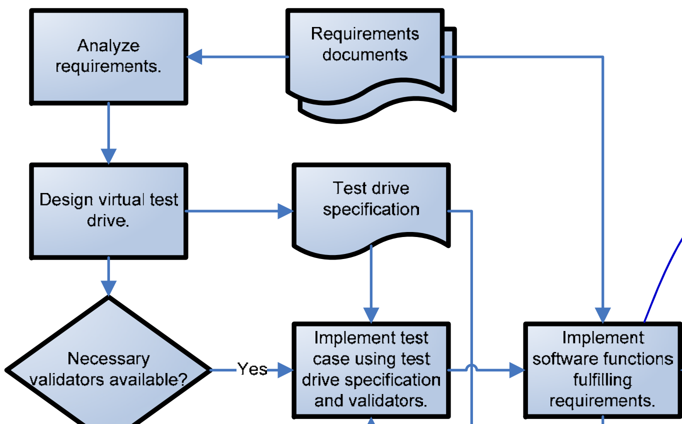
Visualization data (can be viewed by independent GUI application).

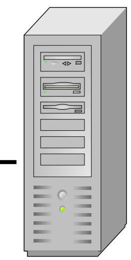
based on:

▪ Basarke … e functions in the “virtual vehicle”, in: Proceedings of the 11th Automotiv…

Setting up system simulation:

But furthermore, these system simulations can be executed automatically, e.g. for every commit, right before important milestones, …





Acceptance test bugbuster server.

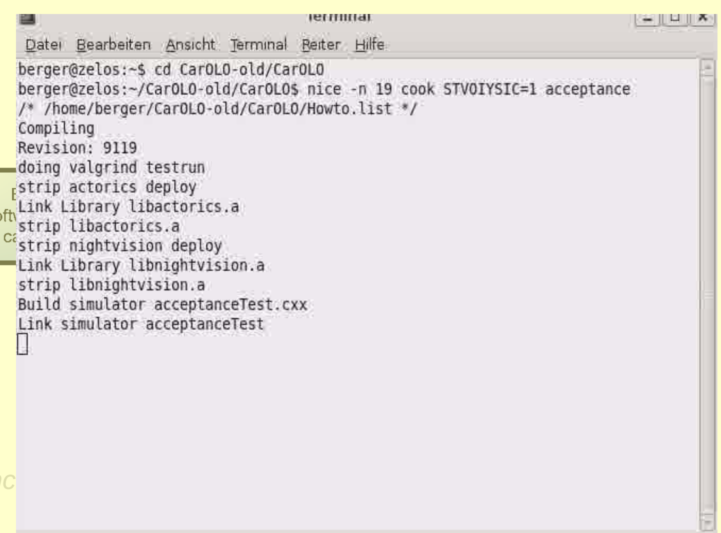
Report and history accessible for everyone using web portal Trac.

runs per commit and evaluates

trac Integrated SCM & Project Management

Filename	src/simulator/testsuites/acceptance/ai/CampusSued_StaticObstacles_Passing_01.h
11	Akzeptanz Test
Resultat:	fehlgeschlagen
Name	CampusSued Dynamic Replanning 01
Description	entsprechend der DARPA-Anforderungen C.4. (Technical Evaluation Criteria) wird ein Szenario mit blockierten Fahrbahnen vorgegeben und somit eine Neuplanung der Route sowie U-Turns ausserhalb von Stub-Roads erforderlich.
InputData	AI: CampusSued_DynamicReplanning (RNDF-Maker-Projekt)
Validator(TimeOut)	Zeitlimit ueberschritten.
Validator(StayInLane)	Die Fahrspur wurde eingehalten.
Validator(PassCheckpoint)	Checkpoint: 4 CC(-78.611, -188.939, 0) wurde nicht erreicht.

System simulation

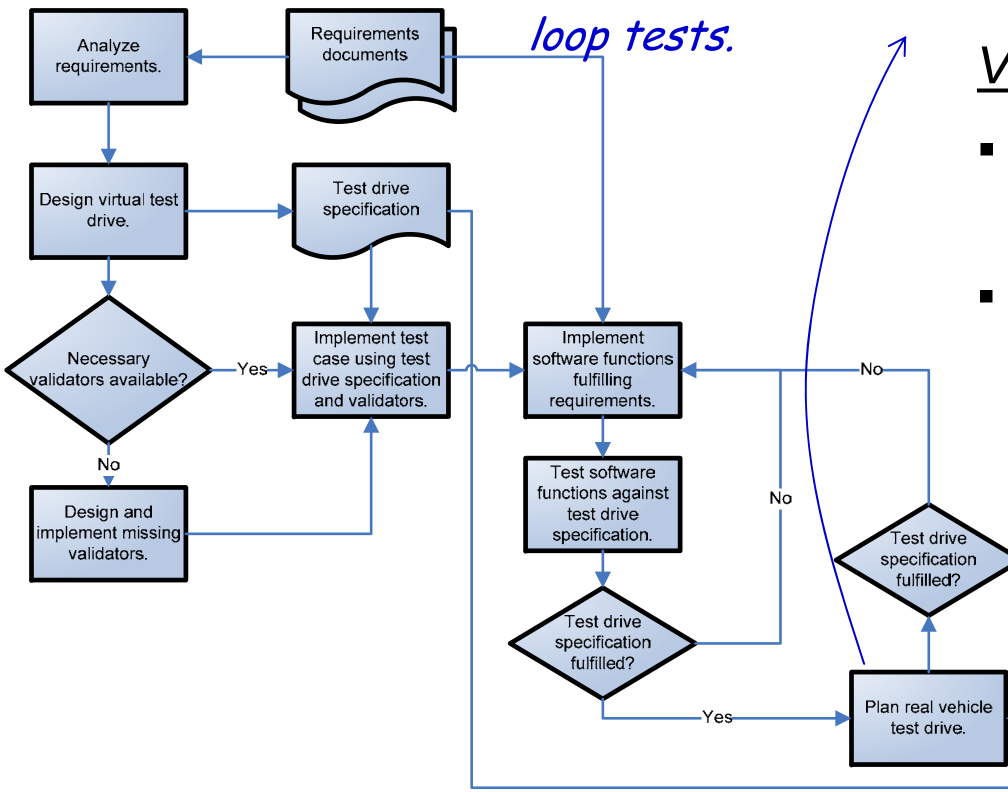


▪ Basarke, C. et al., *Design and quality assurance of intelligent vehicle func*
Automotive Technology Conference, 2007.

Software & System Development Process ▪ Design Decisions ▪ **Quality Assurance Activities**

Setting up system simulation:

Finally: Use system simulation with real-time data in vehicle-in-the-loop tests.



Would be nice, but unfortunately not used in the DARPA Urban Challenge…

Vehicle-in-the-loop:

- Reuse validators for measuring distances to obstacles, …
- Enrich real environment with virtual obstacles or vehicles for generating more complex traffic situations.

Fig. 11: Caroline in test area A → real vehicle test drive.

based on:

- Basarke, C. et al., *Design and quality assurance of intelligent v* … Automotive Technology Conference, 2007.

Conclusion and outlook:

- Simple and robust solutions should be preferred – for processes, tools, hard- and software.

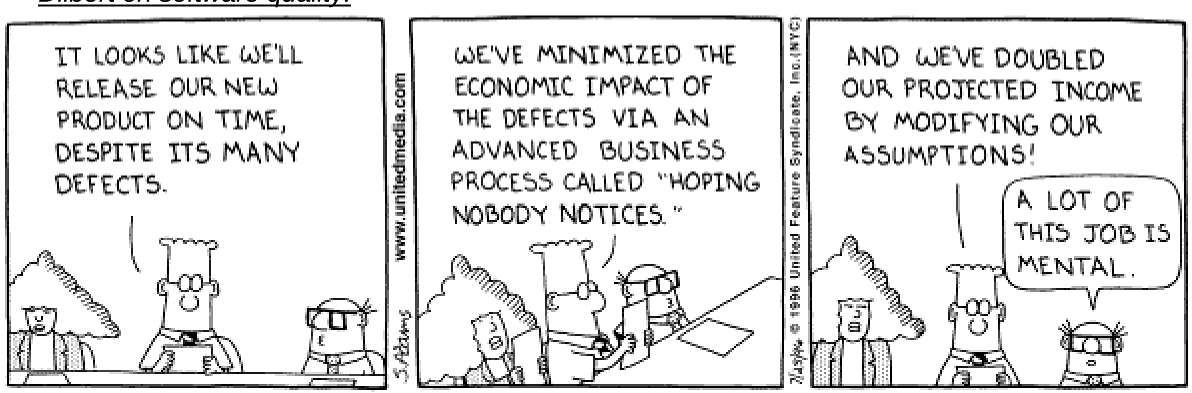
- For testing the quality of software, an extensible software architecture is mandatory for reusing software modules as stand-alone applications or as part of an integrated system simulation.

- „Intelligent Simulations“ are only one part of an integrated quality assurance process → real vehicle tests can not be substituted but complemented and improved.

- Further work should elaborate the requirements- and constraints-DSL besides design patterns for embedded system simulations.

Questions?

Dilbert on software quality:



Literature

[Bas07] Basarke, C. et al., *Design and quality assurance of intelligent vehicle functions in the “virtual vehicle”*, in: Proceedings of the 11th Automotive Technology Conference, 2007.

[Bro05] Broy, M., *Automotive Software and Systems Engineering*, in: Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design, 2005.

[Bus01] Buschmann, F. et al., *Pattern-oriented Software Architecture*, John Wiley & Sons, 2001.

[DARPA-01] DARPA, *Rules*, 2007-10-27, http://www.darpa.mil/grandchallenge/docs/Urban_Challenge_Rules_102707.pdf.

[DARPA-02] DARPA, *Technical Evaluation Criteria*, 2007-03-16, http://www.darpa.mil/grandchallenge/Technical_Evaluation_Criteria_031607.pdf.

[Dij82] Dijkstra, E., *Selected writings on computing: A personal perspective*, Springer, 1982.

[Fow07] Fowler, M., *Writing Software Patterns*, http://www.martinfowler.com/articles/writingPatterns.html, 2007-04-02.

[Gam05] Gamma, E. et al., *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[HH03] Hughes, C. and Hughes, T., *Parallel and Distributed Programming Using C++*, Addison-Wesley, 2003.

[ISO98] ISO/IEC 14882:1998.

[LK05] Langer, A. and Kreft, K., *Standard C++ IOStreams and Locales*, Addison-Wesley 2005.

[Ray01] Raymond, E., *The Cathedral and the Bazaar*, O‘Reilly, 2001.

[Ray04] Raymond, E., *The Art of Unix Programming*, Addison-Wesley, 2004.