Using Models for Dynamic System Diagnosis
A Case Study in Automotive Engineering

Oliver Niggemann, Institut Industrial IT (inIT), Hochschule Ostwestfalen-Lippe, Lemgo, Germany,
Benno Stein, Bauhaus University, Weimar, Germany,
Thomas Spanuth, Heinrich Balzer, University of Paderborn, Paderborn, Germany

Abstract: Though various sophisticated concepts for the diagnosis of technical systems have been developed, diagnosis technology in practical applications often boils down to the use of simple heuristics, associative case memories, or manually designed decision trees. These approaches are robust, but restricted with respect to the complexity of the diagnosed system and the faults to be detected.

An inviting direction, which is desired by practitioners and investigated by researchers, aims at the reuse of those models for diagnosis purposes, which were originally designed for system construction and simulation. A promising principle in this connection is model compilation: the model of the interesting system is simulated in various fault modes, and the resulting (huge) set of simulation data is analyzed with machine learning methods, yielding tailored diagnosis rules [Ste03]. To verify this approach, we present a case study from the field of automotive software development. The case study highlights strengths and weaknesses of model compilation. One key challenge is addressed in this paper: the identification of suited symptoms in the data.

Keywords automotive diagnosis, model compilation, simulation, machine learning

1 DIAGNOSIS AND MODEL COMPILATION

1.1 Learning of Diagnosis Algorithms

Diagnosing technical dynamic systems is part of an engineer's core expertise—a task for which engineers rely on their deep knowledge about the application field and about effects of faults in systems. But since systems become more complex it is exceedingly difficult to understand fault impacts and therefore to implement the diagnosis functionalities. This holds true especially for electronic systems in the automotive industry. So from a computer science perspective, a main questions is whether methods from the field of machine learning and knowledge-based systems can be used to support an engineer's work. The goal is not the replacement of human expertise but its formalization, to make it usable by computer algorithms.

Generally speaking, a to-be-diagnosed system can be abstracted as shown in Figure 1.

1. The system itself. The system can either be the real system – e.g. a vehicle – or a simulation of the system. In the later case, which applies to this paper, a system model must exist.

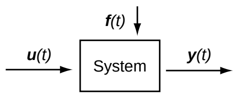


Figure 1: A system stimulated by a vector of input functions $\mathbf{u}(t)$ while failures $\mathbf{f}(t)$ occur. Diagnosis algorithms must resort to the set of measurable signals $\mathbf{y}(t)$.

2. A vector of input functions $\mathbf{u}(t) \in R^k$ defines the driving scenario, such as driving maneuvers and the street situation.
3. During this scenario particular failures (e.g. CAN bit errors), defined by $\mathbf{f}(t) \in \{0, 1\}^l$, may occur. A value of 1 at the ith position in $\mathbf{f}$ indicates that the ith failure occurred.
4. The diagnosis algorithm analyses a set of observable variables $\mathbf{y}(t) \in R^m$ to identify the failure causes $\mathbf{f}$.

On which information does a diagnosis algorithm rely, say, what distinguishes a faulty system from an error free system? Which information is required, to decide whether or not the system is faulty? Obviously a system fault can only be identified by comparing the system's behavior to the correct behavior. Such correct behavior might be given as a (simulatable) formalized computer model. This model is often called golden model.

To identify faulty behavior a simple comparison is not enough: Values might differ without constituting significant or critical faults; especially in complex feedback systems such as vehicles random value fluctuations might occur. So differences between golden model and system must be assessed according to their fault significance and criticality. The output of this assessment is a vector of symptoms.

This consideration leads to a model of a diagnosis algorithm shown in Figure 2. A model of an error free system, the golden model, is used to identify incorrect system behavior: The golden model computes the same measurable variables as the system to be diagnosed.

A symptom vector $\mathbf{s}(t)$ is computed from $\mathbf{y_0}$ and $\mathbf{y}$ in a preprocessing step. The symptom vector might contain elements of $\mathbf{y_0}, \mathbf{y}$, or is computed by complex operations on the output vectors. Generally speaking, symptoms correspond to deviations from the normal behavior, or they describe the context in which these deviations occur. For complex systems, identification of symptoms is a major problem and is treated in Section 2. Based on the computed symptom vector a diagnosis algorithm starts (*i*) to explore if a failure occurred in the system and (*ii*) to classify the failure.

Where do the models – golden model and system model – come from? During the development of automotive control software, control and software engineering models are used for specification purposes. These models fall into two classes: (*i*) models for the electronic control units, the ECUs, (e.g. AUTOSAR [AUT] or Simulink®models) that model the behavior and the structure of the ECUs and their respective software, and (*ii*) environment models (e.g. Modelica or Simulink models) that model the behavior of the vehicle (sensors, actuators, engine, gear) and the environment (driver behavior, street conditions). Models comprising both ECU models and environment models are called closed-loop models.

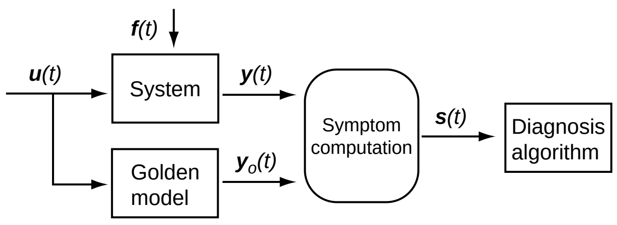


Figure 2: Most diagnosis algorithms rely on knowledge about the system behavior in the non-error case. A golden model is used to compute a set of signals $\mathbf{y_0}(t)$ that are concordant with the measured signals in the no-error case. From a symptom $\mathbf{y}(t)$ and $\mathbf{y_0}(t)$ a vector $\mathbf{s}(t)$ is computed that contains sufficient information to identify the faults.

In this paper those closed-loop models are used as golden models.

The next section will outline the general diagnosis strategy applied in this paper, while Section 2 will describe the key contribution of this paper: A symptom identification approach. This approach is applied in the case study in Section 3. Results are given in section 4.

1.2 The Model-Compilation Approach

In the automotive industry, model-based diagnosis approaches are used quite frequently [FBSI07, SP04, Nyb02, CPD03]. These approaches use symptoms to generate reasonable fault hypotheses, for this the system model is analyzed (see e.g. [Rei87, dKW87]). This generation of fault hypotheses and the ranking of these hypothesis often leads to a non-trivial model analysis problem. Hence, often a specialized *diagnosis model* is constructed by domain experts, where fault deduction is treated in a forward reasoning manner.

The approach applied in this paper is based on a different idea, the so-called model compilation paradigm that was introduced in [Ste01, Hus01]. A salient property of this approach is that neither a too complex analyses of the system model nor a tailored diagnosis model is needed. Model compilation aims at the creation of a fast and precise diagnosis algorithm which is learned as a classification and diagnosis function mapping symptoms onto faults. The learning procedure is based on data recorded during a run of the real system[1] with injected failures and the parallel simulation of the golden model. This empirical data, comprising failures, measurements, and symptoms, is abstracted and generalized in form of the mentioned classification function that *inverts* the causality between failures and symptoms. Such an identification of classification functions is subject of the fields of machine learning and statistics (see [TSK06, Har99] for an overview).

Several model-based diagnosis algorithms use a qualitative model of the system (e.g. in [SS97]) – either generated form the original system model or created manually. Since in this paper model-compilation is used, the system model itself remains unchanged. Instead, the abstraction step is moved to the data mining algorithm. While this approach puts much trust into the machine learning algorithm, it avoids the problem of creating a qualitative

[1]Later on, we will replace recorded data by simulated data.

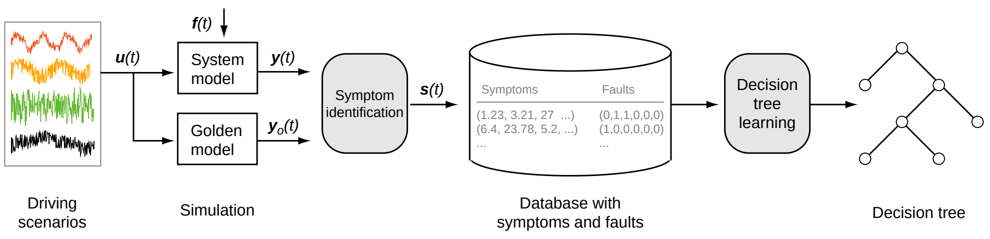


Figure 3: The model compilation process. A large set of driving scenarios are used to simulate the system including faults $\mathbf{f}$ and the golden model. The output vectors of the simulation $\mathbf{y}, \mathbf{y_0}$ are used to generate symptoms $\mathbf{s}$. Domain knowledge is used to generate complex symptoms and form the input for machine learning algorithms to generate a classification tree.

model.

So far, we have not discussed where the data and measurements of the system may come from; especially since for the model-compilation approach a large set of such driving scenarios are necessary. In practice, data about driving scenarios and corresponding faults are hard to obtain; often the faults are unknown or not enough scenarios have been recorded. Beside this it is very expensive to do enough test runs of the real system. A possible way out is the simulation of the real system—in this paper on a PC with an offline simulation. Using this approach, an arbitrary number of scenarios can be simulated and recorded, i.e. huge sets of empirical data can be produced. Based on this empirical data, the classification and diagnosis function is then learned. When such a simulation is used, it is important to verify that the simulation approximates sufficiently the real system.

Since we use simulation to obtain data from the (simulated) “real” system and since in these simulations failures are injected deliberately, the faults $\mathbf{f}(t)$ are known for each point in time. This is a prerequisite for the application of most classification learning algorithms: Since for the learning data the correct classification is known, relations between input data and classification results can be learned. [TSK06] gives an overview of these supervised learning methods. The entire learning process can be seen in Figure 3..

So now besides the golden models for the faultless case, further system models are needed which are able to simulate fault effects, i.e. these additional models are used to simulate the real system. For this, the already mentioned closed-loop models from Section 1.1 can be used also. For automotive diagnosis, the same models are used as golden models for the simulation of the faultless case and as system models for the simulation of the misbehavior. The reader may note that in other domains, golden models and system models might very well be different.

Automotive system models are often able to predict the system behavior in the fault case. For the control part, this follows from the fact that nowadays the software in the ECUs is directly generated from such models, i.e. no significant behavior variations between model and real system are to be expected (see [BOJ04]). For the environment part of the closed-loop model, a general statement is more difficult. But in most cases the models are used during the development to verify diagnosis algorithms, using offline simulation or later in the process using Hardware-in-the-Loop simulation (see [ONS$^+$07]). Note that these environment models therefore have to cover the prediction of fault effects since they are used for the testing of diagnosis algorithms. Hence, the environment models can be used to simulate fault effects in most cases.

Finally, in this paper decision trees [BFOS84] are used as classification function since they are applied on a frequent basis by engineers for automotive diagnosis. Automotive diagnosis happens at different stages of the vehicle life-cycle: within the vehicle, during the development, and in the garage. For the later case – which is the focus of this paper – decision trees are used in most tool chains, i.e. other approaches could not be integrated into existing working processes.

2 THE PREPROCESSING STEP: GENERATION OF SUITED SYMPTOMS

The model compilation approach from section 1.2 has already been applied successfully to various problems in the past—e.g. in [Ste01, Hus01]. But its key advantage—using machine learning to inverse the *fault→symptom* causality which allows for the usage of existing models—also causes its main weakness: Much trust is put in the power of machine learning algorithms. In this section that weakness is addressed by means of preprocessing steps that compute a set of high-level symptoms which eases the machine learning task.

The reader may ask why such a preprocessing step is necessary. Looking at Figure 3, it becomes clear that theoretically all information needed by the learning algorithms is already part of $\mathbf{y}$ and $\mathbf{y}_0$. For two main reasons a preprocessing step may significantly improve the model compilation approach: (*i*) Engineers often apply domain-depended data preprocessing steps which can not be identified automatically. This domain knowhow should be provided to the machine learning algorithms. (*ii*) Most machine learning algorithms have problems with learning complex, e.g. nonlinear or time-dependent, functions: Either they are only able to handle a limited set of rather simple function patterns or they support complex functions but rely for that on non-deterministic and time-consuming optimization methods—details can be found in [TSK06]

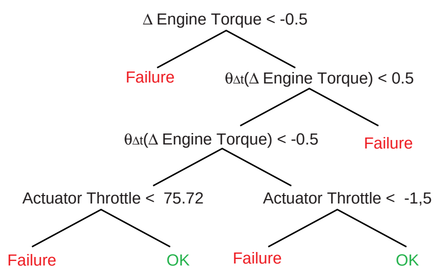
In detail, machine learning algorithms face three main problems when applied to such diagnosis tasks:

1. The diagnose function need not make decisions based on discrete values because otherwise only remembered old situations could be treated correctly. Instead the diagnosis function must abstract from discrete values, i.e. it must use value ranges and other complex value features. That way, new so-far unknown situations can be classified correctly.

2. The measured and simulated values are functions over time. These values must be transferred into stationary, i.e. time independent, features.

3. Often value combinations must be used to classify faults successfully. Such value combinations can be identified in a preprocessing step.

This section will therefore first present some symptom abstraction methods which either



Δ Engine Torque < -0.5
Failure
θΔt(Δ Engine Torque) < 0.5
θΔt(Δ Engine Torque) < -0.5
Failure
Actuator Throttle < 75.72
Actuator Throttle < -1,5
Failure
OK
Failure
OK

Figure 4: Decision tree for signal interruption of the speed sensor.

are computed automatically or are computed manually by engineers—this takes care of point 1 from above. Next, algorithms for generating stationary symptoms are presented.

2.1 Value Abstraction

The most important value abstraction step in this case study is done automatically by the decision tree learning algorithm. The algorithm used (REPTree and J48 from the Weka-Library, [WF05]) splits values into value ranges. Figure 4 shows an example: Each decision splits the value range of one variable, i.e. the decision is based on value intervals instead of discrete values. Because this step is done inherently by the machine learning algorithm, value ranges need not be computed explicitly in a preprocessing step.

We found that in practice the following preprocessing operations are often applied by engineers to diagnose failures in automotive systems; each of these steps computes abstracted value features:

Generation of Deviations from the Golden Model. The simplest operation in this category is the computation of the difference between the output of the (simulated) real system and the golden model: $\mathbf{y} - \mathbf{y}_0$. This makes the diagnosis function independent of the absolute values.

Generation of Statistic Characteristics. Important symptoms can be based on statistical properties of $\mathbf{y}(t)$:

- The standard deviation of $\mathbf{y}(t)$ carries valuable information about the amount of noise in the data. The standard variation σ is normally computed for the last Δt time steps: $\sigma_{\Delta t}(\mathbf{y}(t)) = \sqrt{E(X^2) - (E(X))^2}$ with $X = \mathbf{y}(t-\Delta t), \ldots, \mathbf{y}(t-1), \mathbf{y}(t)$. The same operations can of course be applied to $\mathbf{y}_0$.
- A very helpful symptom is the information whether a value $\mathbf{y}(t)$ is still "normal". The term "normal" is defined by the probability that the measured value $\mathbf{y}(t)$ is still within the normal variance of $\mathbf{y_0}(t)$: $P_{\Delta t}(\mathbf{y}(t)) = \int_{\mathbf{y}(t)}^{\text{inf}} n(\mathbf{e}_{\Delta t}(\mathbf{y_0}), \sigma_{\Delta t}(\mathbf{y_0}))$ where $\mathbf{e}_{\Delta t}$ denotes the expected value and $\sigma_{\Delta t}$ the standard variance for the data vector during the last Δt time steps. n is the Gaussian probability distribution.

  This can also be seen in Figure 5: The probability that a measurement y is still "normal" is defined by the integral under the Gaussian distribution of the golden

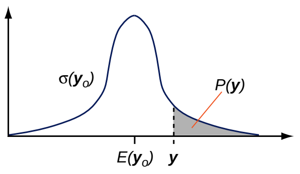


Figure 5: The probability of a measurement from the real system y to be “not normal” depends on the distribution of that measurement in the golden model, i.e. of y_0.

model y_0. $e(y_0)$ denotes the expected value of y_0 while $\sigma(y_0)$ denotes the standard variance.

- The quotient $\frac{\sigma_{\Delta t}(\mathbf{y}(t))}{\sigma_{\Delta t}(\mathbf{y_0}(t))}$ denotes the increase of noise in the data and is therefore a valuable symptom for noise related faults.

2.2 Stationary Symptoms

Only few machine learning algorithm can directly use temporal data (e.g. [CPD03])—none of which have established themselves. The easiest way to handle temporal data is to make the data stationary, i.e. time independent. E.g. the time-dependent function *vehicleVelocity*(t) is replaced for one diagnosis scenario by an approximation v, v_dev (modeling the current speed and the first current derivation). Here, a two step approach is used:

Step I: Mode Identification. Diagnosis functions assume that the causal relation between failure causes and failure effects does not chance between the generation of the diagnosis rules and the usage of those rules during a vehicle’s lifetime. These relations remain stable only under specific conditions — e.g. similar driving situations. Such a situation, that can be addressed by one set of diagnosis functions, is called a mode. In this example, modes correspond to one gear of the vehicle, i.e. the mode can be identified in a rather straight forward way.

Algorithms for the automatic identification of modes can be found in [Ste01, Hus01].

Step II: Generation of Temporal Stationary States. Within each mode, stationary features are generated from time dependent data. For this, diagnosis algorithms often use the history of the vectors $\mathbf{y}(t), \mathbf{y}_0(t)$: An increasing position of the gas pedal should result in an increasing vehicle speed. If the vector values are only used at one specific point in time t, such correlations can not be exploited. Several methods exist to capture the history:

- The history $\mathbf{y}(t_1 \dots t_n)$ can be captured by the following function $\theta_{\Delta t}(\mathbf{y}(t)) = \mathbf{y}(t) - \mathbf{y}(t - \Delta t)$. The identification of a suited Δt is a challenge by itself. Here an initial Δt has been guessed based on system properties, during the machine learning process an optimization method has been used to find an optimal value.

- While the previous function is a linear approximation of $\mathbf{y}$’s history during the last Δt time steps, more complex approximations—e.g. using polynomial approximation functions—can be applied.

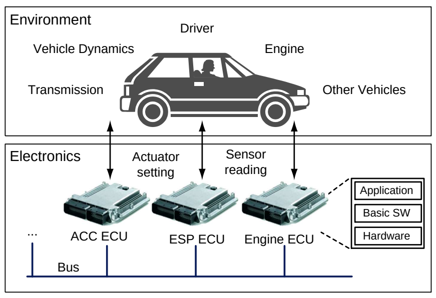


Figure 6: A coarse ACC architecture with the differentiation between ECU models (bottom) and environment models (top).

The same operations are of course applied to $\mathbf{y}_0$.

3 AN EXAMPLE: ADAPTIVE CRUISE CONTROL

3.1 The Adaptive Cruise Control Application

An Adaptive Cruise Control (ACC) is used in modern vehicles to control automatically the distance to cars driving ahead in the same lane. If the car in front of the driver slows down, the ACC will also decrease the vehicle's speed by reducing the throttle setting or even by activating the brakes. If the traffic situation allows for an increase of the vehicle's speed, the ACC will resume again the predefined cruising speed. In order to measure the distance to the up-front traffic, radar, laser, and optical sensors are used.

To implement the ACC functionality, several electronic control units (ECUs) play together: (*i*) Usually the sensor interpretation and sensor fusion happen in a special ACC ECU. (*ii*) Braking activities are often controlled by an Electronic Stability Program (ESP®) ECU. (*iii*) Engine control including throttle position control reside in the engine ECU. (*iv*) User interactions such as turning on the ACC or setting the cruising speed are handled by the central ECU or the display ECU.

All these ECUs are connected via one or several communication buses such as CAN or FlexRay. This architecture is illustrated in Figure 6. Details about the ACC functionality can be found in [Gmb03].

Figure 7 shows the model used in this example: Here Simulink has been used to model both ECU software (left-hand side) and simple models for the engine and ESP ECU (right-hand side). The environment side also comprises models for sensors and actuators.

3.2 ACC-relevant Faults

Figure 7 also pinpoints the main fault sources within ACC systems. The control algorithms in the ECUs (first column in Figure 7) rely on environment information delivered by the sensor layer (rightmost column). Furthermore, the control algorithms use the actuators to control the environment. Hence, four main fault locations can be identified:

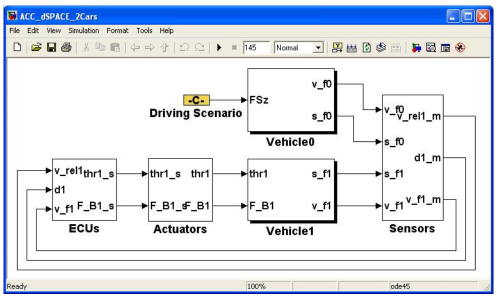


Figure 7: A Simulink Model of an ACC. The ECU models are bottom left. The output of these models forms the input for the actuator models (2nd column) which are directly connected to the environment model of the vehicle ("Vehicle1" in the 3rd column). The subsystem "Vehicle0" models the behavior of the car ahead. On the right, the sensor models can be found that measure data in the vehicle and transport them to the ECU models.

1. Signals coming into the sensor layer could be faulty. This may correspond to faults in the vehicle's wiring.
2. Signals leaving the sensor layer could be faulty. This could either model faulty sensors or a faulty interpretation and preprocessing of the sensor data by the ECU software.
3. Faulty signals coming into the actuator layer correspond to faulty control algorithms or to faults in basic software modules such as I/O drivers.
4. Faulty actuators (including the corresponding wiring) correspond to faulty signals leaving the actuator layer.

In our case study we concentrate on signal faults at these four locations. Faults can be diagnosed at different point in times: While the car is driven, in the garage, or during the vehicle's development. Here we mainly target the second scenario.

4 DIAGNOSIS RESULTS

In cooperation with a leading automotive tool provider and modeling company, the diagnosis method presented in this paper has been used to diagnose the following faults in the ACC system from section 3: (*i*) A faulty throttle actuator, (*ii*) a faulty brake actuator, (*iii*) a faulty computation of the velocity of the vehicle ahead, and (*iv*) a faulty engine crankshaft speed sensor.

For each faulty component four fault subtypes have been addressed: **1**. Additional noise on the signal, **2**. 10% positive offset onto the signal, **3**. 10% negative offset onto the signal and **4**. total loss or interruption of the signal.

Fault i with fault subtypes j is in this paper denoted as $f_{i,j}$. The term f_i denotes the fault i without further differentiation between fault subtypes. All error rates are measured on test sets that have not been used to learn the decision tree. In this paper a separate classification

Fault	f_3	f_4	$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$
Error Rate	14%	12%	16%	13%	9%	1%

Table 1: Typical Error Rates

tree t_i is learned for each f_i, i.e. $t_i : \mathbf{s} \rightarrow \{0, f_{i,1}, f_{i,2}, f_{i,3}, f_{i,4}\}$, where 0 denotes the diagnosis "no fault". Classification functions are learned separately for each gear setting since gear settings form different modes of the underlying mathematical models. So in a later diagnosis setting, the overall diagnosis function will choose the correct classification tree depending on the actual gear.

So all error rates given in this paper denote the percentage of incorrectly diagnosed situations whereat the diagnosis rules have been learned using scenarios different from those later used to measure the error rate. The driving scenarios include different acceleration and deceleration situations; in these situations all gear settings are used.

For the learning itself the REPTREE and the J48 algorithms from the WEKA library (see [WF05]) have been used. The error rates given for such a decision tree always take into consideration that faults $f_{i,j}$ have to be distinguished from other faults $f_{k,l}$ $(k \neq i \vee l \neq j)$—i.e. the learned classification functions have to separate between the different faults.

Typical results can be seen in Table 1.

The learned decision trees are also meaningful from an engineer's perspective: Figure 4 shows an example of a learned decision tree for a signal interruption at the engine speed sensor. Taking knowhow from the ACC domain into consideration, this tree makes sense because the actuator for the throttle position receives its values against the engine speed. Moreover there is a direct relation between engine torque and engine speed. So a faulty engine speed sensor would be noticeable in the relation between engine torque and engine speed which is exactly what has been exploited by the learned decision tree.

5 SUMMARY

In this paper a case study for the diagnosis approach of model compilation is given. The case study is used to outline open challenges and to give first solutions. One focus of this paper is the challenge of symptom identification. Methods for symptom identification have been tested using the diagnosis of an Adaptive Cruise Control (ACC) as an example. It has been shown that sound classification trees can be learned using this approach. The application of model compilation would enable engineers to leverage on their existing system models to implement diagnosis functions for increasingly complex systems and might render specific diagnosis models unnecessary.

References

[AUT] AUTOSAR. Internet Homepage www.autosar.org.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[BOJ04] M. Beine, R. Otterbach, and M. Jungmann. Development of Safety-Critical Software Using Automatic Code Generation. *Society of Automotive Engineers (SAE)*, 2004.

[CPD03] L. Console, C. Picardi, and D.T. Dupre. Temporal Decision Trees: Model-based Diagnosis of Dynamic Systems On-Board. *Journal of Artificial Intelligence Research*, 19:469–512, 2003.

[dKW87] Johan de Kleer and Brian C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[FBSI07] D. Fischer, M. BÃűrner, J. Schmitt, and R. Isermann. Fault detection for lateral and vertical vehicle dynamics. *Control Engineering Practice*, 15(3):315–324, 2007.

[Gmb03] Robert Bosch GmbH. ACC Adaptive Cruise Control. The Bosch Yellow Jackets, 2003.

[Har99] Jens Hartung. *Statistik*. Oldenbourg, 1999.

[Hus01] Uwe Husemeyer. *Heuristische Diagnose mit Assoziationsregeln*. Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 2001.

[Nyb02] Mattias Nyberg. Model-based diagnosis of an automotive engine using several types of fault models. *IEEE Transaction on Control Systems Technology*, 10(5):679–689, September 2002.

[ONS+07] Rainer Otterbach, Oliver Niggemann, Joachim Stroop, Axel ThÃijmmler, and Ulrich Kiffmeier. System Verification throughout the Development Cycle. *ATZ Automobiltechnische Zeitschrift*, 2007.

[Rei87] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[SP04] Peter Struss and Chris Price. Model-based systems in the automotive industry. *AI Magazine*, 24(4):17–34, 2004.

[SS97] Martin Sachenbacher and Peter Struss. Fault Isolation in the Hydraulic Circuit of an ABS: A Real-World Reference Problem for Diagnosis. In *Working Papers of the Eighth International Workshop on Principles of Diagnosis (DX-97), Mont Saint Michel, France*, 1997.

[Ste01] Benno Stein. *Model Construction in Analysis and Synthesis Tasks*. Habilitation, Department of Computer Science, University of Paderborn, Germany, June 2001.

[Ste03] Benno Stein. Model Compilation and Diagnosability of Technical Systems. In M. H. Hanza, editor, *Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications (AIA 03), BenalmÃądena, Spain*, pages 191–197. ACTA Press, September 2003.

[TSK06] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[WF05] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.