Institut für Parallele und Verteilte Systeme

Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3716

# Entwurf und Realisierung eines Systems zur Fehlerdiagnose und Fehlerprävention auf Grundlage einer Case-Based-Reasoning-Fallbasis und eines Fehlermodells

Ernst, Stamp

| | |
|---|---|
| **Studiengang:** | Informatik |
| **Prüfer:** | Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel |
| **Betreuer:** | Sebastian, Abele |
| **begonnen am:** | 16.03.2015 |
| **beendet am:** | 15.09.2015 |
| **CR-Klassifikation:** | G.2.2,H.2.1,H.3.4,I.2.4 |

## Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 15.09.2015

## Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, 15.09.2015

# Inhaltsverzeichnis

**Abbildungsverzeichnis** ..... **iv**

**Tabellenverzeichnis** ..... **v**

**Abkürzungsverzeichnis** ..... **vi**

**Begriffsverzeichnis** ..... **vii**

**Zusammenfassung** ..... **ix**

**Abstract** ..... **x**

**1 Einleitung** ..... **1**

**1.1 Problemstellung** ..... **1**

**1.2 Motivation** ..... **2**

**1.3 Ausgangssituation** ..... **3**

**1.4 Zielsetzung und Anforderungen** ..... **4**

**2 Grundlagen** ..... **5**

**2.1 Fehlerdiagnose** ..... **5**

2.1.1 Symptome, Fehlerfälle und Fehlerursachen ..... 6

2.1.2 Fehlermodell ..... 6

2.1.3 Fehlerdiagnoseverfahren ..... 9

**2.2 JavaScript Object Notation (JSON)** ..... **11**

**2.3 Representational State Transfer (REST)** ..... **13**

2.3.1 Eindeutige Identifikation von Ressourcen ..... 14

2.3.2 Hypermedia ..... 14

2.3.3 Standartmethoden ..... 15

2.3.4 Ressourcen und Repräsentationen ..... 15

2.3.5 Statuslose Kommunikation ..... 15

**2.4 Datenbanken** ..... **16**

2.4.1 Relationale Datenbanken ..... 17

2.4.2 Dokumentenorientierte Datenbank ..... 18

2.4.3 Graph-Datenbanken ..... 18

2.4.4 OrientDB ..... 19

**3 Konzeption des Diagnosesystems ..... 21**
**3.1 Systemaufbau ..... 21**
**3.2 Schnittstellen ..... 22**
3.2.1 Diagnoseanfrage ..... 22
3.2.2 Diagnoseergebnis ..... 23
3.2.3 Feedback ..... 25
**3.3 Diagnosemanager ..... 26**
**3.4 Diagnosemodul ..... 26**
**3.5 Fallbasis ..... 27**
**3.6 Fehlermodell ..... 27**
**3.7 Fehlerdiagnose ..... 30**
3.7.1 Diagnosealgorithmus ..... 31
3.7.2 Risiken und mögliche Probleme ..... 37
**3.8 Fehlerprävention ..... 38**
3.8.1 Fehlermodell für die Fehlerprävention ..... 38
3.8.2 Fehlerpräventionsalgorithmus ..... 38
3.8.3 Risiken und mögliche Probleme ..... 41
**3.9 Feedbackverarbeitung ..... 41**
3.9.1 Feedbackalgorithmus ..... 41
3.9.2 Risiken und mögliche Probleme ..... 42
**3.10 Systemablauf ..... 43**
3.10.1 Fehlerdiagnose ..... 44
3.10.2 Feedback ..... 44

**4 Test und Evaluation ..... 45**
**4.1 Erzeugung von Testdaten ..... 45**
4.1.1 Testdaten für die Fehlerdiagnose ..... 45
4.1.2 Testdaten für Feedbackverarbeitung ..... 47
4.1.3 Testdaten für die Fehlerprävention ..... 47
**4.2 Testablauf ..... 47**
4.2.1 Diagnosefunktion ..... 48

4.2.2 Feedbackfunktion ... 48

**4.3 Bewertungsverfahren ... 48**
**4.4 Testszenarien und Ergebnisse ... 49**
**4.5 Evaluierung ... 54**

**5 Umsetzung und Bedienung des Prototyps ... 56**
**5.1 Entwicklungsumgebung ... 56**
**5.2 Datenbank ... 56**
5.2.1 Installation und Start ... 56
5.2.2 Erstellung der Datenbank für den Prptotyp Betrieb ... 56
**5.3 Klassendiagramm ... 57**
**5.4 JSON Verarbeitung ... 58**
**5.5 Testdatengenerierung ... 58**
**5.6 Start und Bedienung des Prototypen ... 59**
**5.7 Erweiterung des Prototpyen ... 60**
5.7.1 Erstellung neuer Diagnosemodule ... 60
5.7.2 Erweiterung der Schnittstelle ... 60
5.7.3 Veränderung oder Austausch des Fehlermodells ... 61

**6 Zusammenfassung ... 62**

**7 Ausblick ... 65**

**Literaturverzeichnis ... 66**

# Abbildungsverzeichnis

Abbildung 1.1: Ablauf Fahrzeugprüfung und manuelle Nachkontrolle ..... 2
Abbildung 2.1: Ablauf einer Diagnose (nach [LEAY97] ) ..... 5
Abbildung 2.2: Beispiel eines Fehlermodells ..... 7
Abbildung 2.3: Serielle Verbindung ..... 8
Abbildung 2.4: divergierende Verbindung ..... 8
Abbildung 2.5: konvergierende Verbindung ..... 9
Abbildung 2.6: Aufbau JSON-Objekt (nach [ECMA13]) ..... 11
Abbildung 2.7: Aufbau JSON-Wert (nach [ECMA13]) ..... 12
Abbildung 2.8: Aufbau Array (nach [ECMA13]) ..... 12
Abbildung 2.9: Aufbau Zahl (nach [ECMA13]) ..... 12
Abbildung 2.10: Aufbau Zeichenkette (nach [ECMA13]) ..... 13
Abbildung 2.11: Beispiel Personaltabelle in einer Relationalen Datenbank ..... 17
Abbildung 2.12: Webschnittstelle OrientDB ..... 20
Abbildung 3.1: Systemschaubild ..... 21
Abbildung 3.2: JSON-File für Diagnoseanfrage ..... 22
Abbildung 3.3: Beispiel JSON-File Fehlerdiagnoseergebnis ..... 24
Abbildung 3.4: Beispiel JSON-File Fehlerpräventionsergebnis ..... 25
Abbildung 3.5: Beispiel JSON-File Feedback ..... 26
Abbildung 3.6: bedingte Wahrscheinlichkeit bei Symptomen ..... 28
Abbildung 3.7: bedingte Wahrscheinlichkeit bei Symptomen (Beispiel) ..... 29
Abbildung 3.8: Fehlermodell ..... 30
Abbildung 3.9: Diagnosealgorithmus Score-Wert Berechnung ..... 31
Abbildung 3.10: Verbindungskonstellationen bei Score-Wert Berechnung ..... 32
Abbildung 3.11: Beispiel Widerspruch Score-Wert Berechnung ..... 35
Abbildung 3.12: Beispiel Score-Wert Berechnung ..... 36
Abbildung 3.13: Beispiel Symptompfadsuche ..... 37
Abbildung 3.14: Extraktion der Präventionsdaten aus der Wissensbasis ..... 39
Abbildung 3.15: Eintrittshäufigkeit der Symptome im Präventionsbereich ..... 40
Abbildung 3.16: Systemablaufdiagramm ..... 43
Abbildung 4.1: Erzeugung von Testdaten für die Fehlerdiagnose und Fehlerprävention ..... 45
Abbildung 4.2: Symptompfad Generierung ..... 46
Abbildung 4.3: Testablauf Diagnosefunktion ..... 48
Abbildung 4.4: Ergebnisverlauf Szenarien 1 bis 8 ..... 55
Abbildung 5.1: Klassendiagramm ..... 57
Abbildung 5.2: Postman URL Eingabe ..... 60
Abbildung 5.3: Postman http Methode ..... 60

# Tabellenverzeichnis

Tabelle 1.1: Anforderungsübersicht ..... 4
Tabelle 2.1: Symptombedeutung ..... 7
Tabelle 2.2: REST-Methoden (nach [DOGL15] und [BAYE02]) ..... 15
Tabelle 2.3: Vergleich REST und SOAP (nach [DOGL15]) ..... 16
Tabelle 3.1: Schlüssel in Diagnoseergebnis ..... 24
Tabelle 3.2: Parameter für Knoten im Fehlermodell ..... 27
Tabelle 3.3: Score-Werte ..... 31
Tabelle 3.4: größte bedingte Wahrscheinlichkeit bei ausgehender Verbindung ..... 32
Tabelle 3.5: Score-Wert Berechnung ..... 34
Tabelle 3.6: Feedbackverarbeitung ..... 42
Tabelle 4.1: Kriterien Fehlermodell 1 ..... 49
Tabelle 4.2: Kriterien Fehlermodell 2 ..... 49
Tabelle 4.3: Eigenschaften Testszenario ..... 50
Tabelle 6.1: Anforderungsübersicht ..... 63

# Abkürzungsverzeichnis

| | |
|---|---|
| GUI | **G**raphical **U**ser **I**nterface |
| HTTP | **H**yper**t**ext **T**ransfer **P**rotocol |
| ID | **Id**entificatior |
| IP | **I**nternet **P**rotocol |
| NaN | **N**ot **a** **N**umber |
| ODX | **O**pen **D**iagnostic Data E**x**change |
| OS | **O**peration **S**ystem |
| REST | **Re**presentational **S**tate **T**ransfer |
| SOAP | **S**imple **O**bject **A**ccess **P**rotocol |
| SQL | **S**tructured **Q**uery **L**anguage |
| URI | **U**niform **R**esource **I**dentifier |
| XML | E**x**tensible **M**arkup **L**anguage |

# Begriffsverzeichnis

| | |
|---|---|
| **Bedingte Wahrscheinlichkeit** | Wahrscheinlichkeit, dass ein Ereignis eintritt unter der Bedingung, dass bereits ein anderes Ereignis eingetreten ist. |
| **Bibliothek** | Sammlung von verschiedenen Methoden für die Softwareentwicklung |
| **Breitensuche** | Algorithmus für die Traversierung eines Graphen. [PEPP05] |
| **Diagnoseergebnis** | Resultat einer Diagnose, bestehend aus einer Liste von Symptompfaden |
| **Fehlerfall** | Komplette Liste aller Symptome inklusive Symptomstatus des zu diagnostizierenden Systems. |
| **Fehlerpfad** | Symptomliste, die den Weg von einem Symptom bis hin zu einer Fehlerursache, im Fehlermodell beschreibt |
| **Fehlerursache** | Fehlt noch Symptom, das andere Symptome beeinflusst aber nicht selbst beeinflusst wird |
| **Hauptsymptom** | Symptom von dem aus die Score-Werte der Nachbarsymptome berechnet werden. |
| **HTTP-Body** | Beinhaltet Informationen die mit einer http POST Anfrage von dem Client an den Server übertragen werden. |
| **Nachbarsymptom** | Symptome die im Fehlerbaum über eine Kante miteinander verbunden sind |
| **ODX-Datei** | Beinhaltet Informationen wie mit Steuergeräten im Fahrzeug kommuniziert wird und wie erhaltene Informationen interpretiert werden. |
| **Score-Wert** | Kennzahl für die Wahrscheinlichkeit dass der angegebene Symptomstatus dem tatsächlichen entspricht. |
| **SQL-Batch Datei** | Datei, die mehrere SQL-Befehle beinhaltet. Datei wird an Datenbankserver geschickt der alle Befehle der Reihe nach ausführt |
| **Steuerzeichen** | Wird unter anderem für die Textformatierung verwendet. Beispielsweise steht \n, hexadezimal 0x20, für eine neue Zeile [STEU15]. |
| **Symptom** | Im Vorfeld definiertes Merkmal, dessen Eintreten auf ein Fehlerverhalten des Systems hinweist. |
| **Symptomstatus** | Gibt an ob ein Symptom eingetreten ist oder nicht. |

| | |
|---|---|
| **Symptompfad** | Liste von zusammenhängenden Symptomen, ausgehend von einem Wurzelsymptom hin zu einer Fehlerursache |
| **Traversierung** | Jeden Knoten in einem Graphen einmal besuchen. |
| **Webservice** | Schnittstelle mittels derer verteilte Systeme untereinander kommunizieren. |
| **Wurzelsymptom** | Symptom das von anderen Symptomen beeinflusst wird, jedoch keine Symptome beeinflusst |

# Zusammenfassung

Eine gleichbleibend hohe Qualität der Produkte ermöglicht es Unternehmen sich langfristig im Markt zu positionieren. Dies gilt auch für Hersteller von Premiumfahrzeugen. Durch eine Prüfung der Fahrzeuge nach der Endmontage können Mängel aufgedeckt und behoben werden. Somit wird ein hohes Qualitätsmaß der Produkte garantiert. Für eine zuverlässige Fahrzeugprüfung muss sichergestellt sein, dass die Prüfsysteme fehlerfrei funktionieren. Aus diesem Grund werden die Prüfsysteme kontinuierlich überwacht. Wird dabei ein Fehlerverhalten erkannt, kann durch eine Diagnose die Fehlerursache am Prüfsystem bestimmt und die notwendigen Schritte für die Behebung durch einen Prüfer eingeleitet werden. Längere Standzeiten der Prüfanlage und somit höhere Produktionskosten können dadurch verhindert werden.

Das Institut für Automatisierungstechnik und Softwaresysteme (IAS) der Universität Stuttgart forscht hierfür zusammen mit seinen Kooperationspartnern an geeigneten Methoden für die Diagnose von Prüfsystemen.

Ziel dieser Arbeit ist die Entwicklung eines Diagnosesystems, das sich leicht in bestehende Prozesse integrieren lässt und in der Lage ist mehrere Diagnoseverfahren zeitgleich zu verwalten. Darüber hinaus wird ein Diagnosekonzept für die Diagnose von Prüfsystemen erstellt. Das Konzept gibt anhand des Systemzustandes der Prüfanlage und mithilfe eines Fehlermodells sowie einer Fallbasis die wahrscheinlichsten Fehlerursachen aus. Das Fehlermodell repräsentiert die Fehlerzusammenhänge in dem Prüfsystem. Die Fallbasis beinhaltet alle bereits erfolgten Diagnosen und stellt somit die Erfahrung des Diagnoseverfahrens dar. Zusätzlich zu einem Diagnosekonzept soll ein Fehlerpräventionskonzept entwickelt werden, welches mit Hilfe der Fallbasis und des Systemzustandes auf potentiell zukünftig auftretende Fehler hinweist.

Nach der Umsetzung der entwickelten Konzepte, in einem Prototyp, folgt abschließend die Evaluierung. Die dabei verwendeten Daten sowie die einzelnen Testszenarien werden ebenfalls in dieser Arbeit vorgestellt.

# Abstract

The ability of companies to manufacture goods of a constantly high quality enables them to gain lasting success on markets. This principle does also apply for manufacturers of premium cars. Inspecting and testing vehicles after their final assembly can detect and remove deficiencies to ensure a high quality standard of the products. To ensure reliable testing, accurately and error-free working testing systems are required. Therefore, these testing systems are to be monitored constantly. In case of detecting a malfunction, a diagnosis can determine the cause, and have a person monitoring the process initiate the necessary counter measures to remove the error. Extended periods of idle time of the testing site – and therefore higher production costs – can thus be eliminated.

To find suitable methods for the diagnosis of testing systems the Institute of Automation Engineering and Software Systems (IAS) of the University of Stuttgart engages in profound research, closely cooperating with their partners.

The purpose of this thesis is to develop a diagnostic system, which can be easily integrated into already existing processes and systems, and, moreover, which will also be able to conduct several diagnostic procedures at one time. Furthermore, a diagnostic concept for the diagnosis of testing systems will be generated. By processing the status of the system of the testing site in combination with a fault model and certain case-bases, this concept renders the most probable cause of error. The fault model represents the error coherences within the testing system. The case-basis contains all diagnoses previously done, therefore depicting the experiences of the diagnostic procedure. In addition to the diagnostic concept, a concept to prevent malfunctions has to be developed which will – supported by the case-basis and the status of the system – point out potentially upcoming malfunctions.

By actually executing the developed concepts while using a prototype, a final evaluation takes place in the end. The applied data – as well as the scenarios tested – will be also presented in this thesis.

# 1 Einleitung

Eine langfristige Positionierung am Markt ist für industriell produzierende Firmen nur mit qualitativ hochwertigen Produkten möglich [HANS03]. Dies gilt auch für die Hersteller von Premiumfahrzeugen. Die Sicherung einer gleichbleibend hohen Qualität der Fahrzeuge ist nur durch eine Überprüfungen der einzelnen Komponenten sowie des Fahrzeugs als Gesamtsystems möglich. Aus diesem Grund werden alle Fahrzeuge nach der Endmontage einer ausführlichen Fahrzeugprüfung mit Hilfe eines Prüfsystems unterzogen. Werden bei einer Prüfung Fehler erkannt, wird das Fahrzeug durch einen Service-Techniker nochmals manuell überprüft. Durch diesen nachträglichen Prüfprozess können alle entdeckten Fehler vor der Auslieferung an den Kunden behoben werden, um so eine gleichbleibend hohe Qualität zu garantieren.

## 1.1 Problemstellung

Bei einer Fahrzeugprüfung werden alle Funktionen eines Fahrzeugs auf eine korrekte Ausführung hin untersucht. Die Prüfung eines Fahrzeugs erfolgt hierbei größtenteils automatisch. Ein Prüfer überwacht lediglich den Prüfprozess und führt die noch manuell anfallenden Arbeitsschritte aus. Tritt im Verlauf des Prüfprozesses ein Fehler auf, wird der Prüfer benachrichtigt und das Fahrzeug als fehlerhaft im System gespeichert.

Ein Fehler, der während einer Überprüfung des Fahrzeugs auftritt, kann durch das Prüfsystem selbst verursacht werden. Aus diesem Grund darf das Prüfsystem selbst nicht fehlerhaft sein, damit der Fehler auf eine fehlerhafte Fahrzeugfunktion zurückgeführt werden kann. Jedoch kann nicht ausgeschlossen werden, dass Fehler durch das Prüfsystem verursacht werden.

Für einen Prüfer ist nicht ersichtlich, ob der Fehler durch eine Fehlfunktion im Fahrzeug oder durch das Prüfsystem selbst verursacht wurde. Das Fahrzeug wird automatisch als fehlerhaft deklariert und muss durch einen Service-Techniker manuell nachbereitet werden. Kommt es über einen längeren Zeitraum hinweg zu fehlerhaften Fahrzeugprüfungen muss die komplette Fahrzeugüberprüfung gestoppt werden, da nur begrenzte Ressourcen für die manuelle Nachbearbeitung pro Zeiteinheit zur Verfügung stehen. Zeitgleich kann dies zu einem Produktionsstopp führen und damit immense Kosten für das Unternehmen nach sich ziehen. Folgende. Abbildung 1.1 veranschaulicht den Ablauf einer Fahrzeugprüfung inklusive der Nachkontrolle von fehlerhaften Fahrzeugen.

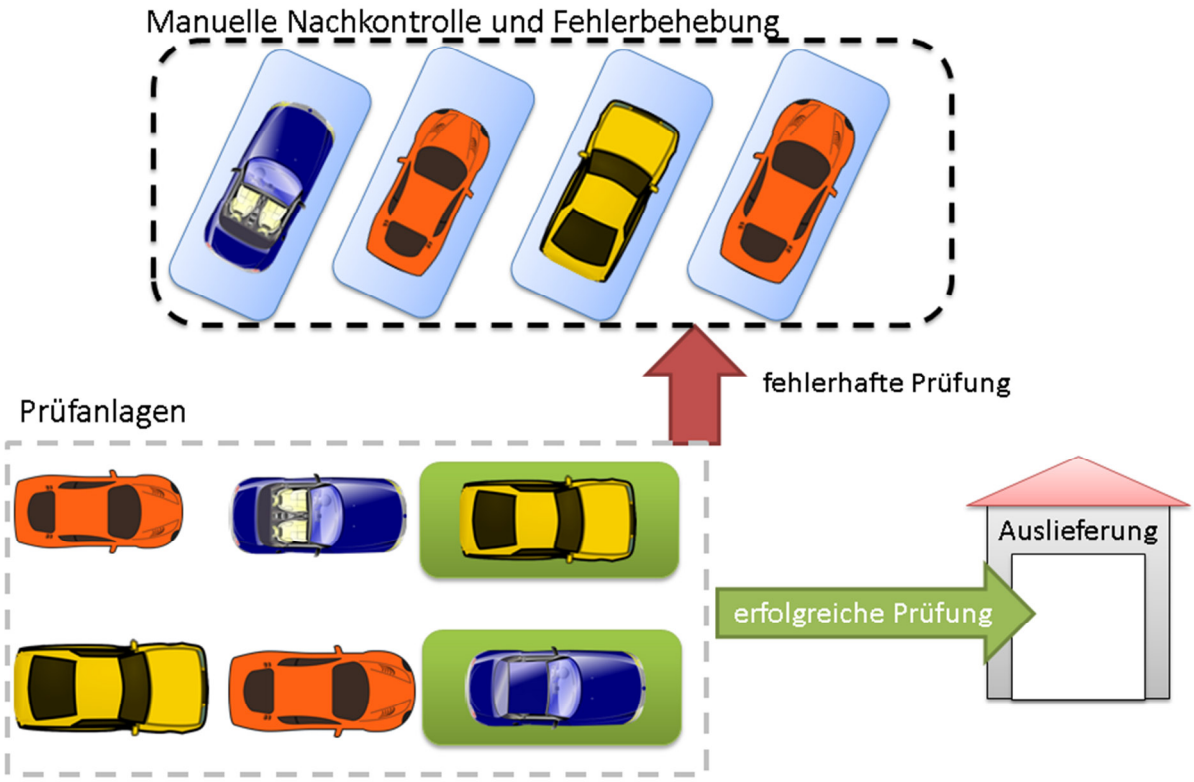



Abbildung 1.1: Ablauf Fahrzeugprüfung und manuelle Nachkontrolle

Wird ein Fehler im Prüfsystem erkannt, kann dieser meistens nicht durch den Prüfer selbst behoben werden. Dies ist nur speziell ausgebildeten Wartungstechnikern möglich, da nur diese das notwendige Hintergrundwissen über das Prüfsystem besitzen. Durch den Einsatz eines Diagnosesystems kann der Prüfer erkennen, ob eine Fehlermeldung durch die Fehlfunktion im Fahrzeug oder durch das Prüfsystem selbst verursacht wurde. Hierbei wird die Abwicklung des Prüfprozesses beschleunigt, da kein speziell ausgebildeter Wartungstechniker für den Rückschluss zur Fehlerursache benötigt wird. Das Hintergrundwissen der Wartungstechniker, um zu erkennen ob dieser Fehler durch das Prüfsystem selbst hervorgerufen wurde, kann durch das Diagnosesystem ersetzt werden. Hiermit können Personal-, Prüf- und Produktionsausfallkosten des Fahrzeugherstellers eingespart werden.

Das Problem der richtigen Fehlerursachenzuordnung und Fehlerursachenerkennung wird im Rahmen dieser Arbeit durch den Entwurf eines Diagnosekonzepts für ein Prüfsystem behandelt.

## 1.2 Motivation

Kommt es aufgrund eines Stopps der Fahrzeugprüfung zu einem Stopp der Produktion ist dies mit hohen Kosten für das Unternehmen verbunden. Daher gilt es die Fehler am Prüfsystem frühzeitig zu erkennen und zu beheben. Durch den Einsatz eines Diagnosesystems für Prüfsysteme kann der Prüfer erkennen, ob die fehlerhafte Fahrzeugüberprüfung an einer Fehlfunktion im Prüfsystem lag oder nicht. Die Zahl der Fahrzeuge, die fälschlicherweise als fehlerhaft markiert wurden, wird dadurch gesenkt und die Produktivität der

Fahrzeugprüfstationen gesteigert. Somit lässt sich die Zahl der Unterbrechungen an der Fahrzeugprüfstation und damit auch in der gesamten Produktionskette verringern.

Mithilfe eines Diagnosesystems kann der Prüfer nicht nur Aussagen darüber treffen ob ein Fehler im Prüfsystem vorliegt oder nicht, es lassen sich auch Aussagen über die Fehlerursache treffen. Damit werden hochqualifizierte Wartungstechniker seltener für die Fehlererkennung an Prüfsystemen benötigt und können in anderen Aufgabenbereichen eingesetzt werden. Bei der Erkennung eines Fehlers können vom Prüfer selbst, ohne Umweg über einen Wartungstechniker, die notwendigen Schritte zur Fehlerbehebung eingeleitet werden, was die Ausfallzeit der Prüfanlage verringert.

Für das Unternehmen selbst bietet ein Diagnosesystem für Prüfsysteme daher verschiedene Vorteile. Zum einen wird der Fahrzeugdurchsatz an der Fahrzeugprüfstation erhöht. Zum anderen werden weniger Wartungstechniker für den Betrieb der Prüfsysteme benötigt beziehungsweise vorhandene Wartungstechniker können in anderen Aufgabenbereichen eingesetzt werden. Alle Vorteile gehen zeitgleich mit einer Senkung der Produktionskosten einher.

## 1.3 Ausgangssituation

Zu Beginn dieser Arbeit stand das Konzept für eine Fallbasis aus einer vorangegangenen Arbeit zur Verfügung[WAIB14]. Dieses Konzept wurde komplett überarbeitet und an die neuen Anforderungen angepasst. Des Weiteren stand ein vereinfachtes Fehlermodell eines Prüfsystems zur Verfügung. Dieses Fehlermodell wurde im Verlauf der Arbeit erweitert.

### Fehlermodell

Ein Fehlermodell beschreibt die Zusammenhänge zwischen Symptomen und Ursachen in einem System (siehe 2.1.2). Das in der Arbeit verwendete Modell entstand in Zusammenarbeit des IAS mit Experten der Audi AG für Prüfsysteme. Das Fehlermodell beinhaltet nicht das gesamte Spektrum der möglichen Symptome und Ursachen (siehe 2.1.1) eines Prüfsystems. Zu Testzwecken wurden hierfür lediglich die Symptome und Ursachen betrachtet die in Zusammenhang mit fehlerhaften ODX-Dateien eintreten. Dadurch wurde die Komplexität des Fehlermodells verringert.

### Ursprüngliches Konzept Fallbasis

Die Fallbasis ist eine Datenbank in der alle bisher eingetretenen Fehlerfälle am Prüfsystem gespeichert werden. Jeder einzelne Eintrag in der Datenbank stellt einen Fehlerfall dar und beinhaltet folgende Informationen [WAIB14]:

- Eine komplette Symptomliste des Prüfsystems mit den dazugehörigen Informationen über den Status der einzelnen Symptome, der angibt ob ein Symptom eingetreten ist oder nicht.
- Die tatsächliche Ursache des Fehlerfalls, falls bekannt.

- Datum und Uhrzeit
- Fahrzeugfabrikat, welches durch das Prüfsystem getestet wurde

## 1.4 Zielsetzung und Anforderungen

Ziel dieser Arbeit ist der Entwurf eines Diagnosekonzepts für ein Prüfsystem sowie die Konzeption und Umsetzung eines Diagnoseverfahrens auf Basis eines Fehlermodells und einer Fallbasis. Das Diagnosekonzept soll neben der Fehlerursachenbestimmung auf Basis von Fehlersymptomen eines Systems auch eine Fehlerprävention ermöglichen. Zur Evaluierung des entwickelten Diagnosekonzepts soll im Zuge dieser Arbeit ein Prototyp erstellt werden. Folgende funktionale (PAF) und nicht funktionale Anforderungen(PANF) wurden während der Anforderungsanalyse dieser Arbeit an das Konzept und den Prototyp gestellt.

Tabelle 1.1: Anforderungsübersicht

| Name | Bedeutung |
|---|---|
| /PAF10/ | Entwurf eines Systemkonzepts welches die Möglichkeit bietet verschiedene Diagnosemethoden in einem System zu vereinen. |
| /PAF20/ | Entwicklung eines Konzepts für eine Fehlerprävention anhand des aktuellen Systemzustandes und bereits vorhandenem Wissen. |
| /PAF30/ | Realisierung eines Prototyps zur Diagnose und Fehlerprävention anhand des übermittelten Systemzustands. |
| /PAF40/ | Einheitliche Verarbeitung des übermittelten Systemzustands. |
| /PAF50/ | Entwicklung von Fehlerszenarien für Prototypen Test. |
| /PAF60/ | Entwicklung einer geeigneten Methode für die Speicherung der Diagnoseanfragen an das Diagnosesystem. |
| /PANF10/ | Das Systemkonzept muss sich leicht erweitern bzw. auf andere Einsatzszenarien übertragen lassen. |
| /PANF20/ | Es soll eine Schnittstelle gewählt oder entwickelt werden, die es ermöglicht den Prototyp leicht in bestehende Systeme zu integrieren. |
| /PANF30/ | Die Diagnoseergebnisse sollen für Prüfer nachvollziehbar ausgegeben werden. |
| /PANF40/ | Das Vorgehen bei einer Systemübertragung bzw. Erweiterung des Prototyps soll in der Dokumentation verständlich erläutert werden. |

Während der Arbeit muss ebenfalls auf veränderte oder neue Anforderungen eingegangen werden.

# 2 Grundlagen

Im nachfolgenden Kapitel werden notwendige Grundlagen vorgestellt. Es werden Grundlagen der Informatik vorausgesetzt.

## 2.1 Fehlerdiagnose

Der Begriff der Diagnose stammt ursprünglich aus der Medizin. Gleichwohl lässt sich die medizinische Definition der Diagnose auch auf die Diagnose von mechatronischen Systemen übertragen. Reif definiert die Diagnose von Fahrzeugen wie folgt: „Aus konkreten und diffusen Symptomen, die der Fahrer schildert, wird im Service unter Zuhilfenahme der Diagnosesysteme ein exaktes Fehlerbild erstellt und es werden geeignete Reparaturmaßnahmen eingeleitet“ [REIF07]. Der Ablauf einer Fehlerdiagnose lässt sich durch folgende Abbildung veranschaulichen.

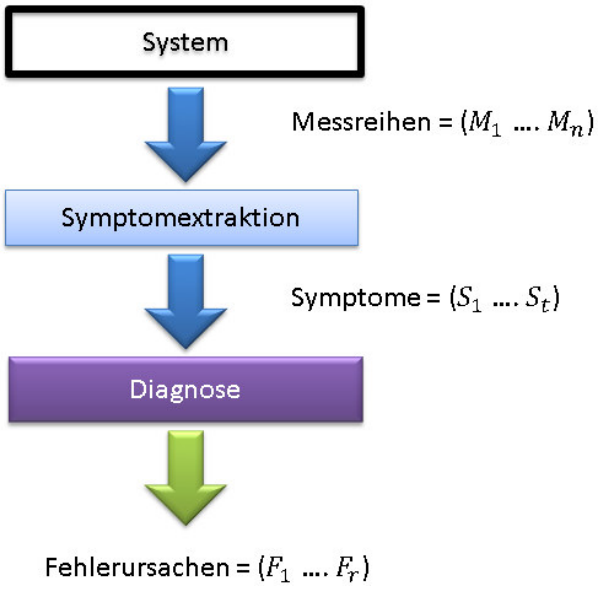



Abbildung 2.1: Ablauf einer Diagnose (nach [LEAY97] )

Das zu diagnostizierende System wird durch die Messung verschiedener systeminterner und externer Kenngrößen überwacht. Einige Beispiele für solche Kenngrößen sind Geschwindigkeit, Laufzeit oder die Auslastung eines Prozesses. Eine Reduktion des Datenvolumens erfolgt durch eine Symptomextraktion aus den einzelnen Messreihen. Hierbei wird nach Auffälligkeiten in den Messreihen gesucht, um daraus Symptome abzuleiten. Mithilfe der abgeleiteten Symptome werden im Diagnosesystem die möglichen Fehlerursachen für das vorliegende Fehlerverhalten des Systems diagnostiziert. Anschließend lässt sich auf Basis der diagnostizierten Fehlerursachen eine Handlungsempfehlung für die Behebung des Fehlers formulieren [LEAY97].

### 2.1.1 Symptome, Fehlerfälle und Fehlerursachen

Symptome sind, wie auch die Diagnose, Begriffe aus der Medizin. Kopfschmerzen, Halsschmerzen oder auch Bluthochdruck sind Beispiele für Symptome des menschlichen Körpers. Symptome deuten im Allgemeinen auf ein Fehlverhalten im System hin, was jedoch nicht immer zu einer Fehlfunktion führen muss. In der Fehlerdiagnose werden Symptome zur Bestimmung einer Fehlerursache verwendet.

In der Symptomextraktion ist festgelegt welche Anomalien in den Messreihen als eingetretene Symptome gewertet werden. Das bedeutet, für jedes System ist ein Normalverhalten definiert. Liegt kein Normalverhalten mehr vor wird dies bei der Symptomextraktion aus Messreihen des Systems erkannt und das dazugehörige Symptom als eingetreten markiert. Voraussetzung hierfür ist jedoch, dass das Symptom sich aus den vorhandenen Messreihen extrahieren lässt. Liegt beispielsweise der Arbeitsbereich eines Motors zwischen 1000 – 2000 Umdrehungen pro Minute wird bei einer Drehzahl größer als 2000 Umdrehungen pro Minute bei der Symptomextraktion das Symptom „überhöhte Drehzahl“ als eingetreten markiert. [ISER12]. Die Bestimmung des Symptomzustandes ist nicht immer durch eine Analyse der Messreihen möglich, da sich gewisse Kennzahlen eines Systems nicht über eine Messung bestimmen lassen. Ein Beispiel hierfür ist der Ölstand eines Fahrzeugs und die Qualität der Schmierung im Motor. Der Ölstand lässt sich über die Füllstandsmessung einfach bestimmen. Für die Bestimmung der Qualität der Schmierung im Motorraum existieren hingegen keine direkten Messreihen. Damit lässt auch während einer Symptomextraktion kein Symptom bezüglich der Schmierung extrahieren.

Eine Diagnose wird nicht auf Basis eines einzigen Symptoms ausgeführt, sondern schließt alle Symptome des Systems, unabhängig davon ob diese eingetreten sind oder nicht, mit ein. Der Zusammenschluss aller Symptome eines Systems wird Systemzustand genannt. Ein Systemzustand enthält neben allen Symptomen eines Systems noch den Status zum Zeitpunkt des Diagnosebeginns.

Zwischen manchen Symptomen eines Systems besteht ein Kausalzusammenhang. Das Eintreten eines Symptoms hat Einfluss auf das Eintreten eines anderes. Anhand dieser Zusammenhänge können komplexe Modelle entwickelt werden, die die Zusammenhänge der einzelnen Symptome beschreiben. Symptome deren eintreten zwar Einfluss auf andere Symptome hat, die aber selbst von anderen Symptomen nicht beeinflusst werden, werden für diese Arbeit als Fehlerursachen definiert. Fehlerursachen beeinflussen das Eintreten anderer Symptome. Symptome die nur durch andere Symptome beeinflusst werden, jedoch selbst keinen Einfluss auf andere Symptome nehmen, werden im Zuge dieser Arbeit Wurzelsymptome genannt. Von der Menge aller Wurzelsymptome lässt sich jedes Symptom im Fehlermodell über die Kausalzusammenhänge erreichen.

### 2.1.2 Fehlermodell

Die Zusammenhänge zwischen Symptomen lassen sich als Fehlermodell in einem gerichteten Graphen modellieren.

**Gerichtete Graphen**

Ein gerichteter Graph $G = (V, E, \sigma, \tau)$ besteht aus einer Knotenmenge $V$, einer Kantenmenge $E$ und zwei Abbildungen $\sigma, \tau: E \rightarrow V$. Die Abbildung $\sigma$ ordnet jeder Kante aus $E$ einen Startknoten aus $V$ zu, die Abbildung $\tau$ jeder Kante aus E einen Zielknoten aus $V$. Zwei Knoten heißen adjazent wenn sie durch eine Kante verbunden sind. Eine Kante $e \in E$ lässt sich durch einen Pfeil $\sigma(e) \rightarrow \tau(e)$ darstellen. [DKR13]

In einem Fehlermodell besteht die Kotenmenge aus allen Symptomen eines Systems. Die Kantenmenge repräsentieren die Kausalzusammenhänge des Systems. Die Richtung der Kannten beschreibt die Kausalreihenfolge der Symptome. Ein Beispiel eins Fehlermodells ist in folgendem Abbild zu sehen.

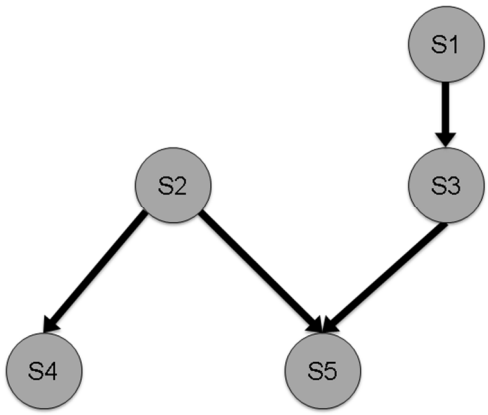



Abbildung 2.2: Beispiel eines Fehlermodells

Das Beispiel beschreibt ein stark vereinfachtes Fehlermodell eines Verbrennungsmotors. Die Bedeutungen der einzelnen Symptome sind in folgender Tabelle aufgelistet.

Tabelle 2.1: Symptombedeutung

| **Symptomname** | **Symptombedeutung** |
|---|---|
| S1 | geringer Ölstand |
| S2 | erhöhte Drehzahl |
| S3 | unzureichende Schmierung |
| S4 | erhöhter Verbrauch |
| S5 | Motorschaden |

Damit lässt sich das Fehlermodell wie folgt lesen: *Ein geringer Ölstand (S1) führt zu einer unzureichenden Schmierung (S3*). Dies kann wiederum *einen Motorschaden (S5)* hervorrufen. Ein Motorschaden wird neben einer unzureichenden Schmierung durch eine überhöhte Drehzahl hervorgerufen. Die Drehzahl wirkt sich neben dem Motorzustand auch auf den Verbrauch aus. In dem Beispiel sind die Symptome S1 und S2 Ursachen und S4 und S5 Wurzelsymptome. Die

Art des Zusammenhangs kann von „Ursache löst Symptom aus“ hin zu „Symptom wird durch Ursache ausgelöst“ entwickelt werden. Die Interpretation des Fehlermodells ist wie folgt: *Ein Motorschaden* wird *entweder durch eine erhöhte Drehzahl oder eine unzureichende Schmierung hervorgerufen.*

Im Allgemeinen lässt sich zwischen drei Arten von Zusammenhängen in einem Fehlermodell unterscheiden [JENS01]. Eine serielle Verbindung wie in Abbildung 2.3 zu sehen ist, beschreibt einen linearen Zusammenhang zwischen Symptomen. Das Eintreten des Symptoms S1 hat über S3 auch Einfluss auf S5. Ist hingegen bekannt, dass S3 eingetreten ist, hat S1 keinen Einfluss mehr auf S5. Anhand des Beispielmodells aus Abbildung 2.2 bedeutet das, dass bei Eintritt des Symptoms „unzureichende Schmierung“ die Information über den Ölstand für den Status des Motors uninteressant ist. Daraus folgt, die Symptome S1 und S5 sind durch S3 D-Separiert [JENS01]. Die Definition der D-Separierung erfolgt nach der Definition der verschiedenen Verbindungstypen.

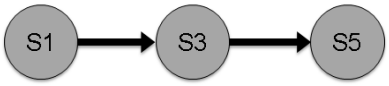



Abbildung 2.3: Serielle Verbindung

Die Zweite Art einer Verbindung zwischen Symptomen wird divergierende Verbindung genannt (siehe Abbildung 2.4).

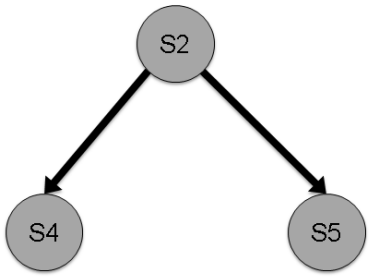



Abbildung 2.4: divergierende Verbindung

Bei einem divergierenden Zusammenhang zwischen Symptomen, hat ein Symptom Einfluss auf mehrere Symptome. Das Symptom „erhöhten Drehzahl“ aus dem Beispielmodell hat zeitgleich Einfluss auf den Verbrauch sowie auf den Motorzustand. Ein Fehlermodell ermöglicht neben Folgerungen der Form S2→ S4 auch Rückschlüsse der Form S4 → S2. Das Eintreten des Symptoms S4 hat auch indirekt Einfluss auf den Status von S2, falls dieser noch nicht bekannt ist. Damit hat S4 auch indirekt Einfluss auf S5. Ist hingegen der Status von S2 bekannt hat S4 keinen indirekten Einfluss auf S5. Hier ist S4 durch S2 von S5 d-Separiert [JENS01].

Bei der dritten Verbindungsart handelt es sich um eine konvergierende Verbindung (siehe Abbildung 2.5)

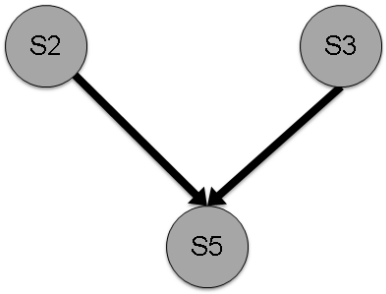



Abbildung 2.5: konvergierende Verbindung

Bei dieser Art des Symptomzusammenhangs wird ein Symptom durch mehrere andere Symptome beeinflusst. Das Symptom des Motorschadens aus dem Beispielmodell (siehe Abbildung 2.2) wird zum einen durch eine unzureichende Schmierung und zum anderen durch eine erhöhte Drehzahl beeinflusst. Hierbei können die Symptome S2 und S3 zeitgleich eintreten. Ist das Symptom S2 eingetreten, hat dies nur Einfluss auf S5 und nicht auf S3. Ist hingegen S2 und S5 eingetreten spielt der Status von S3 keine Rolle mehr [JENS01].

Eine D-Separation, die bei seriellen und divergierenden Zusammenhängen eintreten kann, wird für das Symptomfehlermodell wie folgt definiert.

**D-Separation**

Sei $G = (V, E, \sigma, \tau)$ ein gerichteter Graph und X, Y und Z drei voneinander verschiedene Symptome aus V. X und Y werden in G durch Z genau dann D-separiert, wenn der Zustand von Z bekannt ist und für alle Pfade zwischen X und Y eine der drei Bedingungen erfüllt ist

1. Die Verbindung zwischen X und Y ist seriell und Z ist Teil der Verbindung
2. Die Verbindung zwischen X und Y ist divergent und Z beeinflusst X und Y
3. Die Verbindung zwischen X und Y ist konvergent und Z wird von X und Y beeinflusst (nach [KBK+11] und [JENS01])

Anhand des Fehlermodells und einer Teilmenge an Symptomen deren Status bekannt ist lassen sich Aussagen über mögliche Fehlerursachen treffen. Hierfür werden mithilfe der d-Separation relevante Symptome erkannt und ein möglicher Pfad durch das Fehlermodell erzeugt.

### 2.1.3 Fehlerdiagnoseverfahren

Ein Fehlerdiagnoseverfahren welches durch Symptome Fehlerursachen diagnostiziert, kann als eine mathematische Abbildung von Fehlersymptomen S auf Fehlerursachen F gesehen werden.

$$\{S_1, \dots, S_t\} \rightarrow \{F_1, \dots, F_r\}$$

Wie in [LEAY97] beschrieben gibt es zahlreiche Wege durch die Verarbeitung von Symptomen Fehlerursachen zu erhalten. In dieser Arbeit wird ein Diagnoseverfahren mithilfe eines Kausalnetzes(Abschnitt 2.1.2) verwendet.

Eine Diagnose ausschließlich auf Basis eines Kausalnetzes und den Symptomzuständen liefert Diagnoseergebnisse ohne das diese Rückschlüsse über die Wahrscheinlichkeit ihres Auftretens zulassen. Wird das Kausalnetz um die Eintrittswahrscheinlichkeit der einzelnen Symptome erweitert, können die Diagnoseergebnisse anhand der Eintrittswahrscheinlichkeit sortiert werden. Eine Sortierung hat eine schnellere Fehlerbehebung zur Folge, da die Diagnoseergebnisse anhand ihrer Bewertung abgearbeitet werden können.

Informationen bezüglich der Eintrittswahrscheinlichkeit eines Symptoms in Abhängigkeit von anderen Symptomen können in der Bewertung von Diagnoseergebnissen Anwendung finden. Die Wahrscheinlichkeit für das Eintreten eines Ereignisses in Abhängigkeit eines anderen bereits eingetretenen Ergebnisses wird bedingte Wahrscheinlichkeit genannt. Die bedingte Wahrscheinlichkeit für das Eintreten von S5 unter der Voraussetzung das S2 bereits eingetreten ist, wird wie folgt berechnet:

$$P(S5|S2) = \frac{P(S5 \cap S2)}{P(S2)}$$

Für die Berechnung der bedingten Wahrscheinlichkeit $P(S5|S2)$ werden alle Fehlerfälle, die in der Vergangenheit bereits diagnostiziert wurden, auf das gemeinsame Eintreten beziehungsweise die Schnittmenge der Symptome S5 und S2 untersucht. Die Anzahl der gefundenen Fehlerfälle wird durch die Gesamtanzahl der Fehlerfälle für S2 geteilt und ergibt dadurch $P(S5 \cap S2)$. Für die Bestimmung der Wahrscheinlichkeit $P(S2)$ werden ebenfalls alle Fehlerfälle untersucht und durch die Gesamtanzahl der Fehlerfälle geteilt. Jedoch werden hierbei alle Fehlerfälle gezählt in denen S2 eingetreten ist. Der Status von S5 spielt hierbei keine Rolle. Ist S2 noch nie eingetreten gilt $P(S5|S2) = 0$.

Eine Fehlerdiagnose wird mit der Hilfe eines Kausalnetzes, einer Teilmenge von Symptomzuständen, der D-Separierung und der bedingten Wahrscheinlichkeit durchgeführt. Die Fehlerdiagnose lässt sich in folgende Schritte unterteilen.

1. Anhand der übermittelten Symptome, die den Status eingetreten besitzen und der D-Separierung, werden alle Symptome markiert die keinen Einfluss auf die Ursache haben beziehungsweise deren Eintreten durch die bereits eingetretenen Symptome wahrscheinlicher wird
2. Die wahrscheinlichsten Fehlerpfade zu möglichen Fehlerursachen suchen
3. Die gefundenen Fehlerpfade zu den möglichen Ursachen bewerten und nach Eintrittswahrscheinlichkeit sortiert

Das entwickelte Diagnoseverfahren wird in Abschnitt 3.7 genauer beschrieben. In den folgenden Kapiteln werden die Technologien vorgestellt die in dem Entwurf des Diagnosesystems Anwendung finden.

## 2.2 JavaScript Object Notation (JSON)

Die Kommunikation zwischen Informationssystemen erfordert ein einheitliches Datenaustauschformat. Auf diese Weise ist gewährleistet, dass gesendete und empfangene Informationen von beiden Kommunikationsparteien verstanden und richtig interpretiert werden können. Ein Diagnosesystem stellt hier keine Ausnahme dar. Für die Gewährleistung einer einheitlichen Übertragung der Diagnoseinformationen vom Prüfsystem zum Diagnosesystem wird ein geeignetes Datenaustauschformat benötigt.

Das Datenaustauschformat **J**ava**S**cript **O**bject **N**otation (JSON) wird durch den ECMAScript Programming Language Standard 404 definiert [EMAC13]. JSON ist ein Textbasiertes Datenaustauschformat, welches leicht von Computern verarbeitet wird und von Menschen ohne vorherige Verarbeitung lesbar ist. [NPRI09].

JSON bietet durch einen schlanken Aufbau eine performantere Verarbeitung im Vergleich zu anderen Datenaustauschformaten wie beispielsweise XML [NPRI09]. In einem JSON-Objekt werden Zahlen unabhängig ihres tatsächlichen Zahlen-Typs (*INTEGER*, *FLOAT*, …) einheitlich als eine Reihe von Ziffern repräsentiert und als Text verschickt. Auf diese Weise kann ein JSON-Objekt von verschiedenen Programmiersprachen unter der Vorrausetzung, dass eine einheitliche Textkodierung verwendet wird, richtig interpretiert werden. Dies wäre mit verschiedenen Zahlen-Typen in einem JSON-Objekt nicht ohne weitere Informationen über die Zahl möglich. Programmiersprachen verwenden intern Zahlen-Typen wie *INTEGER*, *FLOAT* oder *DOUBLE,* um die Zahlen zu verwalten. Aufgrund der Übermittlung der Zahlen als Reihe von Ziffern ist jedes Programm in der Lage die Zahl auch ohne zusätzliche Informationen richtig zu interpretieren [EMAC13].

### Aufbau von JSON-Objekten

In diesem Abschnitt wird näher auf den genauen Aufbau von JSON-Objekten eingegangen.

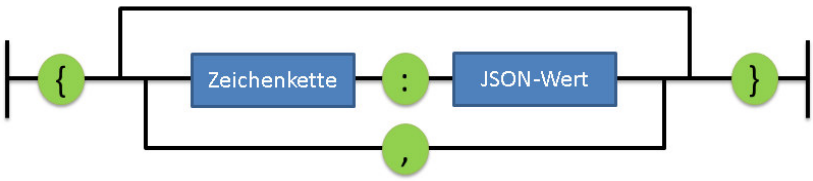



Abbildung 2.6: Aufbau JSON-Objekt (nach [ECMA13])

In Abbildung 2.6 werden die Aufbauregeln für ein JSON-Objekt dargestellt. Jedes korrekte JSON-Objekt beginnt mit einer geschweiften Klammer und wird durch eine geschlossen. Ein leeres Objekt wird durch ein leeres Klammerpaar erzeugt.

In einem Klammerblock stehen beliebig viele Paare, bestehend aus einer Zeichenkette gefolgt von einem JSON-Wert. Zwischen Zeichenkette und JSON-Wert steht ein Doppelpunkt. Nach

jedem Schlüssel-Wert-Paar muss ein Komma folgen, um die einzelnen Paare voneinander zu trennen. Dies gilt jedoch nicht für das letzte Paar in dem JSON-Objekt.

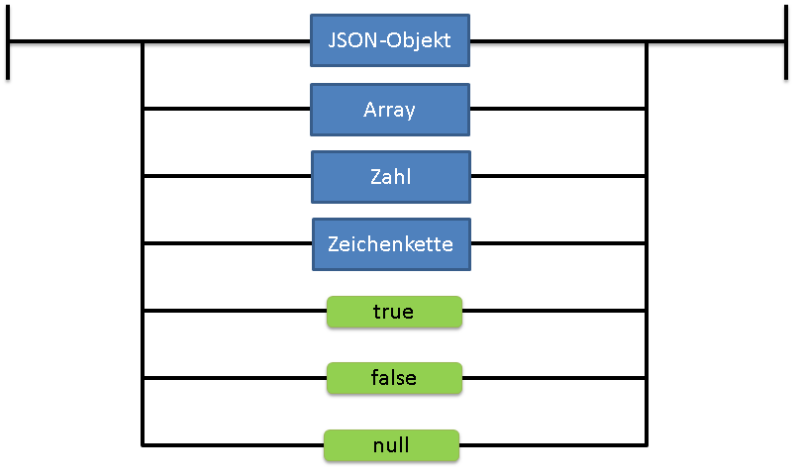



Abbildung 2.7: Aufbau JSON-Wert (nach [ECMA13])

Die Aufbauregeln für JSON-Werte sind in Abbildung 2.7 zu sehen. Ein JSON-Wert kann wiederum selbst ein JSON-Objekt sein. Dadurch lassen sich beliebig tief verschachtelte Strukturen in einem JSON-Objekt generieren. Des Weiteren kann ein JSON-Wert durch eine Zeichenkette eine Zahl oder die Werte *true*, *false* und *null* repräsentiert werden.

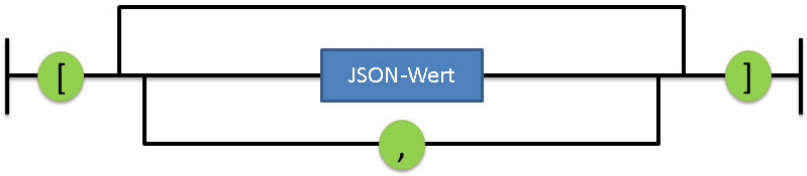



Abbildung 2.8: Aufbau Array (nach [ECMA13])

Ein Array beginnt und endet ähnlich wie auch das JSON-Objekt mit einer Klammer, nur handelt es sich bei dem Array um eckige Klammern. Dies ist in Abbildung 2.8 zu sehen. Die einzelnen Array-Werte sind JSON-Werte aus Abbildung 2.7 und werden durch ein Komma getrennt.

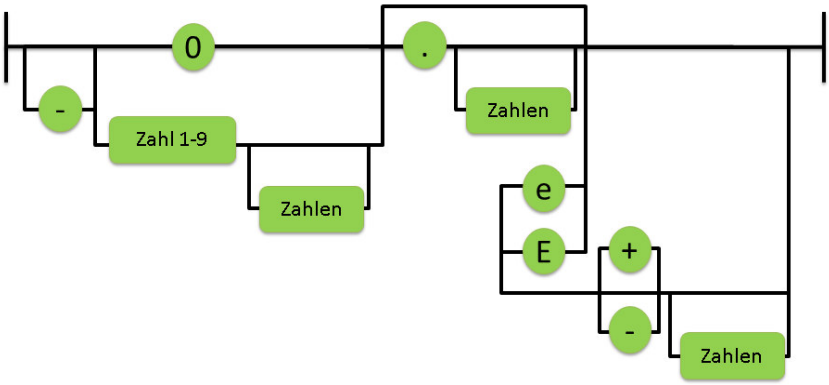



Abbildung 2.9: Aufbau Zahl (nach [ECMA13])

Die Aufbauregeln für Zahlen in Abbildung 2.9 stellen sicher, dass damit alle endlich natürlichen Zahlen gebildet werden können. Für die vereinfachte Darstellung von großen Zahlen ist auch eine wissenschaftliche Notation möglich. Komplexe Zahlen oder Zahlen die sich nicht mit Ziffern darstellen lassen, wie beispielsweise NaN werden von JSON nicht als Zahl unterstützt [EMAC13]. Sie können bei Bedarf als Zeichenkette übertragen werden.

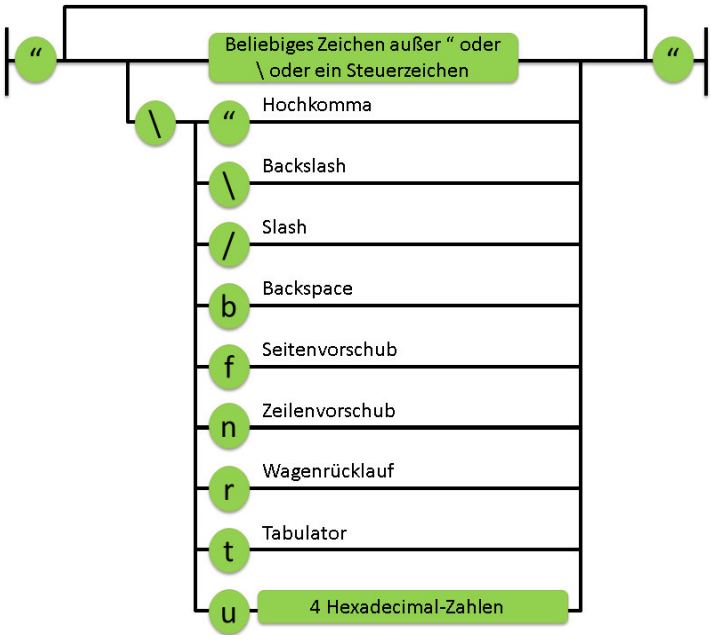



Abbildung 2.10: Aufbau Zeichenkette (nach [ECMA13])

Der Inhalt für Zeichenketten in JSON-Objekten ist nur hinsichtlich der Verwendung von Steuerzeichen und den Zeichen Hochkomma und Backslash reglementiert. Die Verwendung aller andern Zeichen ist möglich. Ein Hochkomma markiert den Beginn und das Ende einer jeden Zeichenkette in einem JSON-Objekt. Das Einfügen eines Hochkommas in eine Zeichenkette ist nur unter der Verwendung eines Backslash möglich. In einer Zeichenkette wird das Backslash-Zeichen als Markierer verwendet und ermöglicht dadurch die Verwendung von Hochkomma, Steuerungszeichen und anderen Sonderzeichen.

## 2.3 Representational State Transfer (REST)

Für eine Kommunikation zwischen verteilten Systemen wie beispielsweise einem Prüfstand und einem Diagnosesystem bedarf es einer geeigneten Schnittstelle mittels derer die Diagnoseinformationen im JSON-Format übertragen werden können. Im Folgenden wird die Schnittstellentechnologie vorgestellt die in dem Konzept des Diagnosesystems zum Einsatz kommt. Am Schluss des Kapitels wird noch ein Vergleich zu einer anderen Schnittstellentechnologie gezogen, um die Wahl von REST als Schnittstellentechnologie zu verdeutlichen.

Der Begriff Representational State Transfer (REST) steht für einen Architekturstil für verteilte vernetzte Systeme und wurde im Jahr 2000 von Roy Fielding während seiner Dissertation

entwickelt und vorgestellt [DOGL15]. Fielding abstrahierte REST aus dem schon bestehenden von ihm mitentwickelten Hypertext Transfer Protocol (HTTP) [TILK11].

Für die Kommunikation zwischen einzelnen Komponenten eines verteilten Systems bedarf es Schnittstellen, über die kommuniziert werden kann. Eine Schnittstelle muss folgende fünf Kernprinzipien erfüllen um als REST konform oder RESTful zu gelten [TILK11]:

- Eindeutige Identifikation von Ressourcen
- Hypermedia
- Standartmethoden
- Ressourcen und Repräsentationen
- Statuslose Kommunikation

Im Folgenden werden die Prinzipien des REST Architekturstils näher vorgestellt.

### 2.3.1 Eindeutige Identifikation von Ressourcen

Der REST Architekturstil sieht vor, dass alle Ressourcen eines verteilten Systems zu jedem Zeitpunkt eindeutig über einen Uniform Ressource Identifier (URI) identifizierbar sind. Hierbei kann es sich bei einer Ressource um Dokumente wie Textdateien oder Bilder, sowie Dienste oder auch um Verzeichnisse handeln [FIEL00].

Eine Identifikation muss nach dem REST Architekturstil den kompletten Pfad zu einer Ressource angeben [DOGL15]. Die Identifikation einer Tabelle in einer Datenbank, ausschließlich über ihren Namen ist nicht zulässig. Eine zweite Tabelle in einer anderen Datenbank kann denselben Namen besitzen. Aus diesem Grund ist eine eindeutige Identifikation nicht möglich.

Eine einheitliche Vergabe von URIs existiert bereits und wurde über mehrere Jahre hinweg erfolgreich getestet. Im Internet werden Ressourcen über URIs so identifiziert, dass die URI weltweit eindeutig ist. [TILK11].

### 2.3.2 Hypermedia

Das Prinzip von Hypermedia ermöglicht es einem Client weiterführende Informationen zu erhalten. In der Antwort des Servers sind dazu Verweise enthalten denen gefolgt werden kann. Die neu erhaltenen Links zu anderen Ressourcen können dem Client vor Erhalt der Serverantwort noch unbekannt gewesen sein [DOGL15]. Das Hypermedia-Prinzip kommt beispielsweise bei einer Anfrage durch einen Client über den Inhalt seines Warenkorbs zum Einsatz. Bei dem Aufruf eines Warenkorbes mit den darin enthaltenen Waren müssen nicht die kompletten Informationen zu allen Waren mitgeliefert werden. Eine Verlinkung auf die jeweiligen Waren genügt um dem Client die Möglichkeit zu bieten die gesuchten Informationen zu erhalten [TILK11].

### 2.3.3 Standartmethoden

Die Verwendung von einheitlichen Schnittstellen die den Zugriff auf Ressourcen regeln, ist verglichen mit anderen Technologien eines der Hauptpunkte die für REST sprechen [DOGL15]. Einheitliche Schnittstellen ermöglichen eine Kommunikation zwischen Client und Server ohne vorherige Vereinbarung eines Protokolls [BAYE02]. Darüber hinaus vereinfachen einheitliche Schnittstellen die Systemarchitektur und erhöhen die Nachvollziehbarkeit von Interaktionen zwischen den einzelnen Komponenten in einem verteilten System[FIEL00]. Eine REST-konforme Schnittstelle unterstützt folgende Methoden:

Tabelle 2.2: REST-Methoden (nach [DOGL15] und [BAYE02])

| **Methoden Name** | **Bedeutung** |
|---|---|
| GET | Lesezugriff auf eine Ressource |
| POST | Erzeugung neuer Ressourcen oder Ausführung beliebiger Aktionen die nicht durch andere Methoden abgedeckt werden |
| PUT | Aktualisierung oder Erzeugung von Ressourcen |
| DELETE | Löschen von Ressourcen |
| HEAD | Abfrage ob Ressource vorhanden |
| OPTIONS | Anfrage an Ressource welche Methoden unterstützt werden |

Hierbei ist zu erwähnen, dass Ressourcen zwar alle Methoden unterstützen, also die Möglichkeit eines Aufrufs bieten. Jedoch wird mit Aufruf nicht die Ausführung einer Methode, die die gewünschte Operation umsetzt, garantiert.

### 2.3.4 Ressourcen und Repräsentationen

Die Repräsentation einer Ressource besteht aus den Informationen welche die Ressource beinhaltet. Zusätzlich existieren Metainformationen, die angeben wie die Daten zu lesen sind beziehungsweise um was für ein Datenformat es sich handelt (JSON, XML, …). Ressourcen können in verschiedenen Datenformaten vorliegen und dieselben Informationen beinhalten [DOGL15].

Die einheitlichen Schnittstellen jeder Ressource bieten dem Client die Möglichkeit mit jeder Ressource zu interagieren, welche der Client auch verarbeiten kann. Durch eine Bereitstellung von Ressourcen in verschiedenen Datenformaten wird die Kompatibilität eines REST Services erhöht, da dadurch mehrere Clients in der Lage sind den REST-Service zu verwenden [TILK11].

### 2.3.5 Statuslose Kommunikation

Verteilte vernetzte Systeme, die nicht nach dem REST-Architekturstil aufgebaut sind, speichern bei einer Kommunikation die Verbindungsinformationen zwischen Client und Server teilweise

auf dem Server. Dies sind verbindungsrelevante Informationen wie beispielsweise die IP eines Clients, der Inhalt seines aktuellen Warenkorbs oder den Beginn der Verbindung. Eine andere Einheit im verteilten System kann beispielsweise bei einem Serverausfall die Verarbeitung der Daten eines Clients nicht einfach übernehmen.

Der REST Architekturstil sieht eine Kommunikation zwischen Client und Server, ohne temporär angelegte Daten, auf dem Server vor. Es ist entweder die Aufgabe des Clients die notwendigen Informationen der Verbindung zu speichern oder die Aufgabe des Servers aus den Informationen der Verbindung eine Ressource zu generieren. Die auf dem Server neu erzeugte Ressource wäre dann über eine URI und die Standartmethoden für jeden Client erreichbar der das Datenformat der Ressource verarbeiten kann. Der Vorteil dieser Vorgehensweise ist, dass die Kopplungen zwischen Server und Client reduziert wird und das System dadurch skalierbar wird. Der Client muss seine Anfrage nicht immer an denselben Server senden, da er alle notwendigen Verbindungsinformationen bei jeder Anfrage mitsendet. Die Verarbeitung der Anfrage durch ein Server-Cluster, welcher die Auslastung auf mehrere Server verteilt, ist dadurch ermöglicht.

REST ist neben SOAP heutzutage bei der Entwicklung von Webservices sehr verbreitet. Viele große Unternehmen wie Twitter, Ebay oder Amazon implementieren ihre Webservices auch oder ausschließlich nach dem REST Architekturstil [CBA11]. Bei SOAP handelt es sich im Gegensatz zu REST nicht um einen Architekturstil sondern um ein konkretes Protokoll für die Entwicklung von Webservices. Die Tabelle 2.3 bietet eine Übersicht über den Vergleich zwischen REST und SOAP.

Tabelle 2.3: Vergleich REST und SOAP (nach [DOGL15])

| **REST** | **SOAP** |
|---|---|
| Ein Webservice kann von verschiedenen Clients aufgerufen werden, unabhängig der verwendeten Programmiersprache. | Für jede Programmiersprache muss ein eigener Soap-Client entwickelt werden um den Webservice aufrufen zu können |
| Für die Ausführung eines Webservice genügt es dem Client die Haupt URI eines Webservices zu kennen. Über das Hypermedia Prinzip holt er sich alle zusätzlichen Informationen. | Ein Client muss zuerst alles über einen Webservice wissen um ihn ausführen zu können |
| Durch die Verwendung von Standartmethoden für den Zugriff auf Webservices besteht eine lose Kopplung zwischen Client und Server. Serverseitig kann Quellcode geändert werden ohne Änderungen am Client vorzunehmen | Der Webservice wird direkt aus dem Code auf dem Client aufgerufen. Wenn serverseitig der Webservice und damit auch der Webserviceaufruf angepasst wird muss der Webserviceaufruf auch clientseitig angepasst werden. |

## 2.4 Datenbanken

Für die Verwaltung der Daten, die im Laufe einer Diagnose anfallen, wird eine geeignete Datenbanktechnologie benötigt. In diesem Kapitel wird die verwendetet Datenbanktechnologie

erläutert. Hierzu werden im Vorfeld die grundlegenden Datenbanktechnologien vorgestellt um dann im Anschluss die Vorteile der verwendetet Datenbanktechnologie zu verdeutlichen.

Datenbanken finden Einsatz in der Datenverwaltung sowie der Datenverarbeitung. Steiner definiert Datenbanken wie folgt: „Eine Datenbank ist eine selbständige und auf Dauer ausgelegte Datenorganisation, welche einen Datenbestand sicher und flexibel verwalten kann“ [STEI94]. Eine Datenbank bietet dem Anwender die Möglichkeit, Daten in der Datenbank zu lesen, anzulegen, zu ändern und zu löschen. Hierbei werden die Berechtigungen jedes einzelnen Anwenders beachtet. Ein Anwender hat nur Zugriff auf die Daten für die er die notwendigen Berechtigungen besitzt [STEI94].

### 2.4.1 Relationale Datenbanken

Im Jahre 1970 schuf Codd [CODD70] die Grundlage für das heute weit verbreitete relationale Datenbank-Modell. Relationale Datenbanken organisieren die Daten als Menge von Datensätzen mit gleicher Dimensionsanzahl und identischer Domänenreihenfolge. Datensätzen mit gleicher Dimensionsanzahl und gleicher Datendomäne werden zu Tabellen zusammengefasst. Jeder Datensatz wird durch eine ID, eine eindeutige Zeichenfolge, eindeutig identifiziert. Ein Verweis zwischen Datensatz A und Datensatz B wird durch die Verwendung der ID von Datensatz A in Datensatz B generiert [CODD70]. Die Selektion bestimmter Datensätze aus der Datenbank wird durch die Verwendung von Abfragen realisiert. SQL als Abfragesprache wird von vielen Datenbanken unterstützt [STEI94].

| ID | Vorname | Nachname | Geburtstag | Abteilung | Vorgesetzter |
|---|---|---|---|---|---|
| 1 | Richard | Müller | 02.05.1964 | Verwaltung | NULL |
| 2 | Beate | Müller | 24.06.1980 | Verwaltung | 1 |
| 3 | Nadin | Schmidt | 28.06.1995 | Verwaltung | 2 |
| 4 | Petra | Klöss | 04.06.1975 | Verwaltung | 2 |
| 5 | Herbert | Fielker | 17.11.1970 | Fertigung | 1 |
| 6 | Walter | Probst | 21.05.1968 | Fertigung | 5 |
| 7 | Susanne | Hütten | 08.04.1998 | Fertigung | 5 |

Abbildung 2.11: Beispiel Personaltabelle in einer Relationalen Datenbank

Abbildung 2.11 zeigt einen Auszug aus einer Personaltabelle in einer relationalen Datenbank. Eine Filterung nach Mitarbeiter die in der Abteilung Fertigung arbeiten würde in SQL wie folgt aussehen

```
SELECT * FROM Personal WHERE Abteilung = "Fertigung"
```

Auf diese Weise lassen sich mittels SQL Daten in der Tabelle beliebig nach Abteilung, Geburtsdatum, Nachname oder Vorname filtern. Die Tabelle beinhaltet neben den Personalinformationen auch noch Informationen über die Hierarchie der Vorgesetzten in der Firma. Die Spalte Vorgesetzter beinhaltet die ID des Vorgesetzten. Eine mögliche Anforderung könnte beispielsweise darin bestehen die Daten nach allen Vorgesetzten oder allen Untergebenen eines Arbeitnehmers zu filtern. Aufgrund der Unwissenheit bezüglich der

Hierarchietiefe in dem Unternehmen ist diese Anforderung in SQL nur durch rechenintensive Funktionen zu bewerkstelligen. Dieses Szenario einer Filterung von Datensätzen anhand ihrer Beziehungen zueinander veranschaulicht die Schwierigkeit der Verarbeitung von vernetzten Daten in relationalen Datenbanken. Laut Iordanov werden nur mengenbasierte Abfragen auf vernetzten Daten durch das relationale Datenbankmodell abgedeckt. Die beiden anderen Abfragetypen, Traversierung (Beispiel: Ausgabe aller Untergebenen eines Arbeitnehmers) und Mustererkennung von Beziehungen (Beispiel Erkennung von Zyklen in Beziehungen) werden durch SQL und relationalen Datenbanken nur schlecht abgedeckt [IORD10].

### 2.4.2 Dokumentenorientierte Datenbank

In einer dokumentenorientierten Datenbank werden einzelne Datensätze durch Dokumente repräsentiert. Die Dokumente selbst werden im JSON-Format gespeichert. Deren Inhalt kann sich grundlegend voneinander unterscheiden und unterliegt keiner festen Tabellenstruktur wie es bei relationalen Datenbanken der Fall ist [BROW12]. Damit eigenen sich dokumentenorientierte Datenbanken für die Speicherung von sich kontinuierlich verändernde Datenstrukturen.

### 2.4.3 Graph-Datenbanken

Anstelle von Tabellen verwenden Graph-Datenbanken Graphen zur Organisation ihrer Daten. Graphen eigenen sich für die Veranschaulichung von Relationen zwischen einzelnen Objekten. Objekte werden in einem Graph durch Knoten repräsentiert. Beziehungen werden durch Kanten zwischen zwei Knoten dargestellt. Knoten sowie auch Kanten können mit Attributen versehen werden. In einem gerichteten Graphen wird die Beziehung zwischen zwei Knoten A und B durch eine Kante von A nach B beschrieben. Hierbei ist A der Start und B der Zielknoten der Beziehung. Mit anderen Worten bedeutet das, dass A der Elternknoten von B und B der Kindknoten von A ist.

Ein Beispiel für die Verwendung von Graph-Datenbanken ist die Speicherung eines Kausalnetzes (siehe Abbildung 2.2). Hierbei repräsentieren die Knoten die Symptome und die Kanten die Beziehungen zwischen den Symptomen. Graph-Datenbanken sind für die Verarbeitung von Graphen ausgelegt und bieten neben der Speicherung Interaktionsmöglichkeiten mit einem Graphen an. Dies äußert sich in einer speziell auf Graphen abgestimmte Abfragemethodik.

Das Kausalnetzt lässt sich auch in Tabellen in einer relationalen Datenbank speichern. Hierfür wird eine Tabelle Symptome und eine Tabelle Zusammenhänge angelegt. Damit werden Symptome und Verbindungen durch Einträge in einer Tabelle gespeichert Im Vergleich dazu spiegelt die Speicherung des Kausalnetzes, in einer Graph-Datenbank als Graph, den Anwendungsfall verständlicher wider und ist damit für einen Anwender auch intuitiver zu verarbeiten.

Die Verwendung von Graph-Datenbanken bringt neben dem bereits genannten Vorteil auch die Möglichkeit bekannte Problemlösungsalgorithmen aus der Graphentheorie für die Suche in einer

Graph-Datenbank zu verwenden [DKR13]. Im Folgenden wird die Graph-Datenbank OrientDB vorgestellt, die bei der Entwicklung des Prototypen verwendet wurde.

### 2.4.4 OrientDB

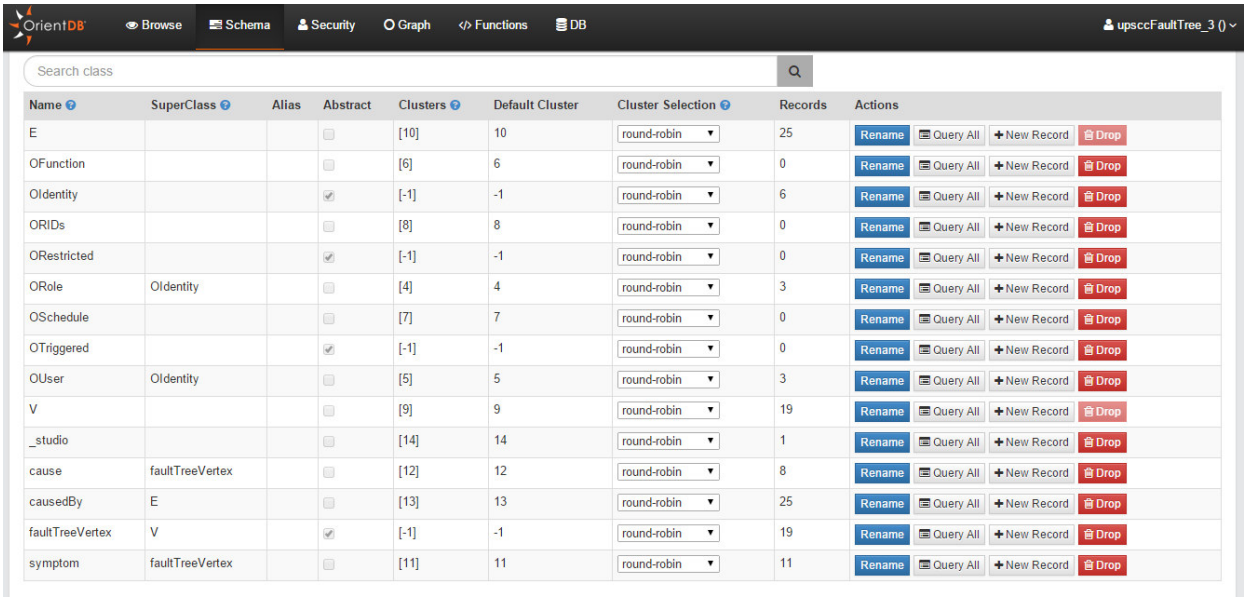
Die Graph-Datenbank OrientDB wird von dem Unternehmen Orient Technologies entwickelt und gewartet. OrientDB wird unter der Apach2Lizenz [APA15] veröffentlicht. Die Datenbank steht damit auch für den kommerziellen Einsatz zur Verfügung. Das Unternehmen Orient Technologies bietet auch eine Enterprise Version an. Diese ermöglicht unter anderem eine Analyse und Verbesserung bestehender Abfragen, zentrale Online-Backups und die Inanspruchnahme eines 24 Stunden Support. OrientDB kommt bereits in großen Unternehmen und Institutionen, wie beispielsweise der UN und Lufthansa zum Einsatz [ODB15].

In der aktuellen Datenbankrangliste liegt die Graph-Datenbank Neo4j [NEO15] vor OrientDB [DBE15]. Ungeachtet dessen hat OrientDB im Vergleich zu Noe4j einige Vorteile. Einer der größten Vorteile von OrientDB ist der multi-modell Ansatz der Datenbank. OrientDB ist keine reine Graph-Datenbank. Es besteht auch die Möglichkeit Datensätze als Dokumente in der Datenbank zu hinterlegen, wie in einer dokumentenorientierten Datenbank.

Ein weiterer Vorteil von OrientDB ist die Unterstützung der Abfragesprache SQL. Durch die Unterstützung von SQL für die Entwicklung von Abfragen wird der Umstieg von einer relationalen Datenbank zu OrientDB erleichtert, da keine von Grund auf neue Abfragesprache erlernt werden muss [ODB15].

OrientDB bietet eine HTTP-REST (siehe Kapitel 2.3) Schnittstelle an, über die mit der Datenbank kommuniziert werden kann. Die Kommunikation findet über das HTTP-Protokoll und JSON-Files statt. Die Schnittstelle ermöglicht es unter anderem Abfragen an den Datenbankserver zu senden, Informationen über den Server einzuholen oder die Struktur bestehender Datenbanken anzupassen. Über diese Schnittstelle kann die Datenbank mit jeder Programmiersprache angesprochen werden, welche in der Lage ist HTTP-Anfragen zu versenden und zu empfangen. [ODB15].

Eine Web-GUI ist ebenfalls Teil des Funktionsumfangs von OrientDB. Über die GUI lassen sich neue Datenbanken anlegen oder bestehende anpassen. Das Ausführen von SQL Abfragen sowie das visualisieren von Graphen ist ebenfalls über die GUI möglich. [ODB15]. In Abbildung 2.12 ist ein Teil der Webschnittstelle von OrientDB zu sehen.




| Name | SuperClass | Alias | Abstract | Clusters | Default Cluster | Cluster Selection | Records | Actions |
|---|---|---|---|---|---|---|---|---|
| E | | | ☐ | [10] | 10 | round-robin | 25 | Rename Query All New Record Drop |
| OFunction | | | ☐ | [6] | 6 | round-robin | 0 | Rename Query All New Record Drop |
| OIdentity | | | ☑ | [-1] | -1 | round-robin | 6 | Rename Query All New Record Drop |
| ORIDs | | | ☐ | [8] | 8 | round-robin | 0 | Rename Query All New Record Drop |
| ORestricted | | | ☑ | [-1] | -1 | round-robin | 0 | Rename Query All New Record Drop |
| ORole | OIdentity | | ☐ | [4] | 4 | round-robin | 3 | Rename Query All New Record Drop |
| OSchedule | | | ☐ | [7] | 7 | round-robin | 0 | Rename Query All New Record Drop |
| OTriggered | | | ☑ | [-1] | -1 | round-robin | 0 | Rename Query All New Record Drop |
| OUser | OIdentity | | ☐ | [5] | 5 | round-robin | 3 | Rename Query All New Record Drop |
| V | | | ☐ | [9] | 9 | round-robin | 19 | Rename Query All New Record Drop |
| _studio | | | ☐ | [14] | 14 | round-robin | 1 | Rename Query All New Record Drop |
| cause | faultTreeVertex | | ☐ | [12] | 12 | round-robin | 8 | Rename Query All New Record Drop |
| causedBy | E | | ☐ | [13] | 13 | round-robin | 25 | Rename Query All New Record Drop |
| faultTreeVertex | V | | ☑ | [-1] | -1 | round-robin | 19 | Rename Query All New Record Drop |
| symptom | faultTreeVertex | | ☐ | [11] | 11 | round-robin | 11 | Rename Query All New Record Drop |

Abbildung 2.12: Webschnittstelle OrientDB

Bei der Erstellung der Datenbanken kann zwischen zwei Datenbanktypen gewählt werden. Einer dokumentenorientierten oder einer Graph-Datenbank. Mit OriendDB lassen sich auch relationale Datenbanken modellieren, da OrientDB von schemafrei, also ohne feste Tabellen, bis hin zu einem festen Datenbankschema jedes Datenbankmodell unterstützt. Des Weiteren können beim Erstellen eines festen Datenbankschemas die einzelnen Tabellen voneinander erben oder sogar als abstrakt deklariert werden. [ODB15].

# 3 Konzeption des Diagnosesystems

In diesem Kapitel wird das entwickelte Diagnosekonzept näher vorgestellt. Hierbei wird auf den Aufbau des Diagnosesystems sowie auf den Diagnoseablauf im Detail eingegangen. Darüber hinaus wird auf die Funktionsweise der einzelnen Systemkomponenten, wie der Fehlerdiagnose oder der Fehlerpräventionsansatz eingegangen.

## 3.1 Systemaufbau

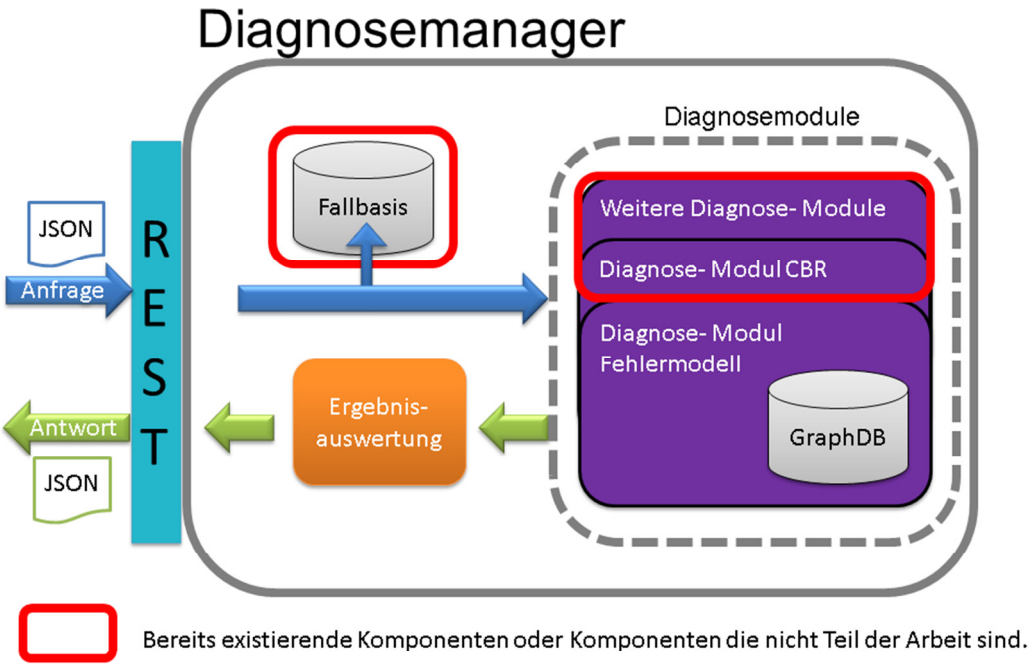



Abbildung 3.1: Systemschaubild

Abbildung 3.1 zeigt die einzelnen Komponenten des Diagnosesystems sowie den Datenfluss im System. Das Diagnosesystem beinhaltet folgende Komponenten:

- einen Diagnosemanager, welcher den Systemablauf steuert
- eine Fallbasis
- verschiedene Diagnosemodule
- eine Diagnoseergebnisauswertung
- eine REST-Schnittstelle

Über die Schnittstellen können andere Anwendungen Diagnoseanfragen senden oder neue Fehlerfälle in die Fallbasis einfügen. Die Fallbasis enthält sämtliche Fehlerfälle aus vergangenen Diagnosen und stellt somit die Erfahrung des Diagnosesystems dar. Für Fehlerdiagnose sowie

Fehlerprävention können mehrere verschiedene Diagnosemodule parallel verwaltet werden. Eine Kommunikation zwischen den einzelnen Diagnosemodulen findet nicht statt. Der komplette Diagnoseablauf wird durch den übergeordneten Diagnosemanager gesteuert. Nach einer Diagnose durch die einzelnen Diagnosemodule werden die Diagnoseergebnisse bewertet und in einem Diagnoseprotokoll zusammengefasst. Im Anschluss wird das Diagnoseprotokoll in eine JSON-Datei konvertiert. Diese beinhaltet alle notwendigen Diagnoseinformationen für die Anwendung, die die Diagnoseanfrage gestellt hat.

## 3.2 Schnittstellen

Eine Kommunikation zwischen dem Diagnosesystem und anderen Anwendungen erfolgt ausschließlich über die definierten REST-Schnittstellen. Die Schnittstelle ermöglicht es anderen Anwendungen eine Diagnose zu starten. Hierfür muss mit der Anfrage ein JSON-File mitgeliefert werden, welches die notwendigen Informationen für die Diagnose enthält. Ein weiterer Anwendungsfall der durch die Schnittstelle abgedeckt wird, ist Feedbackverarbeitung bezüglich der gelieferten Diagnoseergebnisse. Im Detail bedeutet das, dass ein Anwender Rückmeldung über die tatsächliche Fehlerursache geben kann. Diese Information wird vom System verwendet um die Qualität der Diagnoseergebnisse stetig zu verbessern. Im Folgenden werden die Schnittstellen im Detail beschrieben

### 3.2.1 Diagnoseanfrage

Die zum Start der Diagnosefunktion erforderlichen Informationen sind Abbildung 3.2 aufgelistet

```
{
    "metaInformation":
    {
                        "type":"diagnosis"
                        "carId": "1234",
                        "testerId": "1",
                        "startTime": "2015-08-22 15:40:32",
                        "endTime": "2015-08-22 15:41:05",
                        "location": "21",
                        "assemblyLine": "MLA",
                        "assemblyLineSection": "unknown",
                        "testProgramVersion": "1.1xy",
                        "testProgramName": "nameXY",
                        "systemVersion": "1.3",
                        "test_result": "1.3"
    },
    "symptoms":
    [
                        {
                            "symptomId": "S.ODX.2",
                            "name": "Spezifizierte ODX Daten können nicht initialisiert werden",
                            "zustand": "OCCURED"
                        },
                        {
                            "symptomId": "S.ODX.3",
                            "name": "Keine ODX Daten vorhanden bzw. keine neuen mehr empfangen",
                            "zustand": "NOTOCCURED"
                        }
    ]
}
```

Abbildung 3.2: JSON-File für Diagnoseanfrage

Das JSON-File besteht aus zwei Hauptkomponenten. Zum einen das JSON-Objekt „metaInformationen“, das alle Metainformationen zu der Diagnose beinhaltet. Zum anderen das JSON-Array „symptoms“. Im Array ist der Systemstatus des Prüfsystems gespeichert. Der Status besteht aus allen Symptomen aus dem Fehlermodell sowie dem aktuellen Status jedes einzelnen Symptoms. Das in Abbildung 3.2 gezeigt JSON-File ist mit Beispieldaten befüllt. Unter realen Bedingungen können sich die in dem JSON-Objekt „metaInformationen“ enthaltenen Informationen ändern. Für eine Diagnose werden momentan nur die Informationen aus dem JSON-Array „symptoms“ verwendet. Dies kann sich mit neu erstellten Diagnosemodulen ändern. Die Einträge aus „metaInformationen“ werden unbearbeitet zusammen mit der Symptomliste in der Fallbasis gespeichert.

### 3.2.2 Diagnoseergebnis

Nach einer abgeschlossenen Diagnose erstellt die Diagnosesoftware ein Diagnoseprotokoll im JSON Format mit folgenden Informationen:

- Diagnose Metainformationen
- Liste aller Symptome
- Fehlerdiagnoseergebnisse
- Fehlerpräventionsergebnisse

Das Integrieren der Liste aller Symptome sowie der Metainformationen über die Diagnose (Abbildung 3.2) in das Diagnoseprotokoll erleichtern die Verarbeitung des Diagnoseprotokolls. Notwendige Informationen zu einer Diagnose sind in einem Dokument gespeichert und müssen nicht aus verschiedenen Informationsquellen gelesen werden.

```
            {
                ...
            }
        ],
        "results":
        [
            {
                "diagModul":"FalutTree",
                "probability":72.8,
                "reliability":0.789,
                "errorCause":
                                {
                                    "symptomId": "S.ODX.42",
                                    "name": "WLAN Verbindung unterbrochen"
                                },
                "symptomPath":
                                [
                                    {
                                    "symptomId": "S.ODX.1",
                                    "name": "Spezifizierte ODX Daten können nicht initialisiert werden",
                                    "zustand": "OCCURED"
                                    },
                                    {
                                    "symptomId": "S.ODX.2",
                                    "name": "Spezifizierte ODX Daten können nicht initialisiert werden",
                                    "zustand": "OCCURED"
                                    },
                                    {
                                    "symptomId": "S.ODX.3",
                                    "name": "Keine ODX Daten vorhanden bzw. keine neuen mehr empfangen",
                                    "zustand": "OCCURED"
                                    }
                                ]
            }
        ],
        "prediction":
        [
            {
```


Abbildung 3.3: Beispiel JSON-File Fehlerdiagnoseergebnis

In Abbildung 3.3 ist der Fehlerdiagnoseergebnisteil eines Diagnoseprotokolls zu sehen. Die Diagnoseergebnisse sind in dem JSON-Array „results“ gespeichert und beinhalten Informationen über die diagnostizierte Fehlerursache. In Tabelle 3.1 sind die einzelnen Schlüssel eines Diagnoseergebnisses mit dazugehöriger Bedeutung aufgelistet.

Tabelle 3.1: Schlüssel in Diagnoseergebnis

| **Schlüsselname** | **Bedeutung** |
|---|---|
| diagModul | Name des Diagnosemoduls welches dieses Diagnoseergebnis erzeugt hat. Wird zur Nachverfolgung des Diagnoseergebnisses genutzt. |
| probability | Gibt die durch das Diagnosemodul berechnete Wahrscheinlichkeit (0-100%) an mit der die vorgeschlagene Fehlerursache der tatsächlichen Fehlerursache entspricht. |
| relaibilitiy | Kennzahl (0-1) die angibt wie zuverlässig die Fehlervorschläge des Diagnosemoduls in der Vergangenheit waren. Wobei die 1 als Kennzahl maximale Zuverlässigkeit bedeutet. |
| errorCause | Diagnostizierte Fehlerursache mit ID und Fehlername. |
| SymptomPath | Symptomliste ausgehend vom Wurzelsymptom hin zur Fehlerursache. Zu jedem Symptom auf dem Symptompfad werden ID, Name und Zustand gespeichert. |

Eine Speicherung mehrerer verschiedener Diagnoseergebnisse in dem JSON-Array „result“ ist durch die verwendeten Schlüsse möglich, da durch die gemeinsame Betrachtung aller Schlüssel jede diagnostizierte Fehlerursache eindeutig identifizierbar ist.

Die Ergebnisse der Fehlerprävention, die auch Teil einer Diagnose ist, werden in dem JSON-Array „prediction“ gespeichert. Jedes Fehlerpräventionsergebnis beinhaltet eine Liste der potentiellen zukünftig eintretenden Fehlersymptome. Ein Beispiel eines Fehlerpräventionsergebnisses ist in Abbildung 3.4 zu sehen


```
"prediction":
[
        {
            "diagModul":"FalutTree",
            "possibleErrorSymptoms":
                        [
                            {
                            "symptomId": "S.ODX.1",
                            "name": "Spezifizierte ODX Daten können nicht initialisiert werden",
                            "probability":72.8
                            },
                            {
                            "symptomId": "S.ODX.2",
                            "name": "Keine ODX Daten vorhanden bzw. keine neuen mehr empfangen",
                            "probability":65.2
                            }
                        ]
        }
]
```


Abbildung 3.4: Beispiel JSON-File Fehlerpräventionsergebnis

Der Name des Diagnosemoduls wird wie beim Fehlerdiagnoseergebnis zum Zwecke der Nachverfolgbarkeit gespeichert. Ein Fehlerpräventionsergebnis besitzt weder eine Gesamtwahrscheinlichkeit noch eine Kennzahl, die Rückschlüsse über die Zuverlässigkeit des Diagnosemoduls im Hinblick auf eine Fehlerprävention zulässt. Die Wahrscheinlichkeit für das Eintreten eines Fehlersymptoms wird für jedes Fehlersymptom einzeln angegeben.

### 3.2.3 Feedback

Die Diagnosesoftware ermöglicht es einem Anwender, die erhaltenen Diagnoseergebnisse zu bewerten. Durch eine Rückmeldung des Diagnoseergebnisses, dass der tatsächlichen Fehlerursache am ähnlichsten ist, kann der Anwender zur stetigen Qualitätssteigerung der Diagnoseergebnisse beitragen. Der Prozess der Rückmeldung wird jedoch nicht überwacht. Das erhaltene Feedback wird ohne Plausibilitätsprüfung verarbeitet. Die Verantwortung einer gewissenhaften Verwendung liegt hier auf Seiten des Anwenders. Es ist für das Diagnosesystem nicht möglich zu verifizieren, ob es sich bei der Feedbackmeldung um den tatsächlichen Fehler handelt. Die Folgen einer unsachgemäßen Bedienung der Feedbackfunktion werden in Kapitel 3.9 vorgestellt.

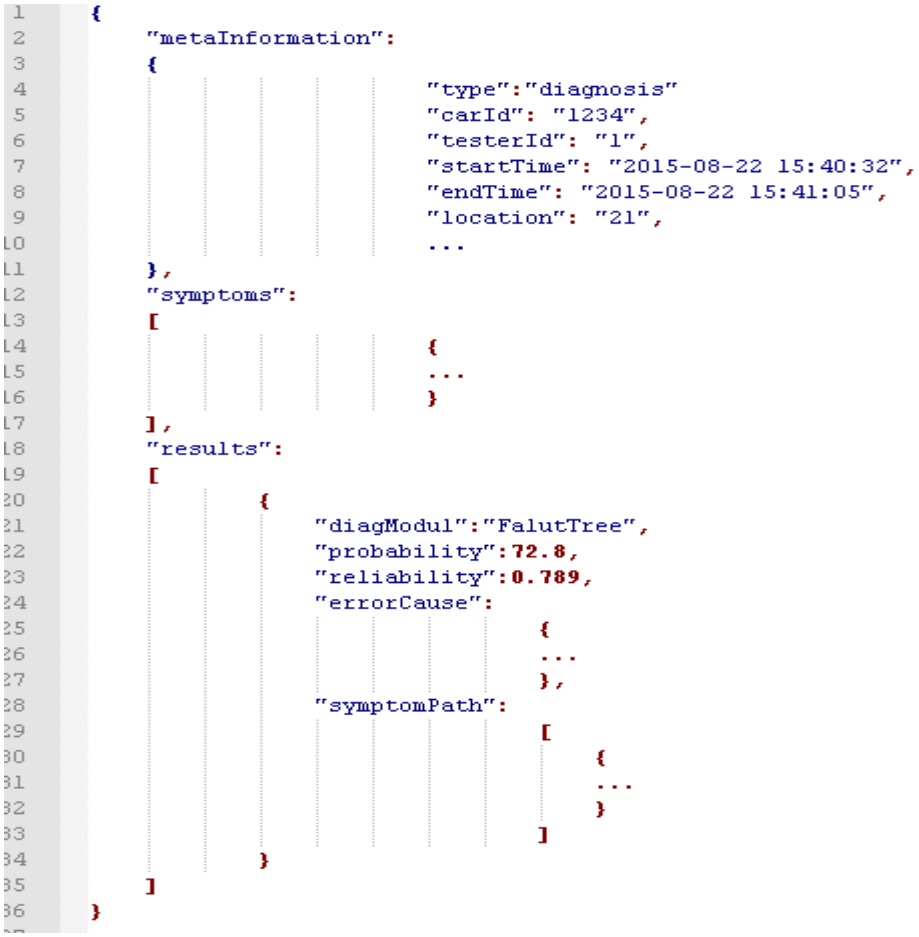



Abbildung 3.5: Beispiel JSON-File Feedback

Für eine Feedbackmeldung an das Diagnosesystem wird mit der Anfrage ein JSON-File mitgesendet, wie es in Abbildung 3.5 zu sehen ist. Das JSON File besteht aus den Informationen einer Diagnoseanfrage und dem Diagnoseergebnis welches an das System als richtig gemeldet werden soll.

## 3.3 Diagnosemanager

Der Diagnosemanager ist die Verwaltungseinheit des Diagnosesystems. Er stellt die REST-Schnittstellen bereit und steuert die Abläufe einer Anfrage. Darüber hinaus verwaltet er die Fallbasis und steuert die einzelnen Diagnosemodule. Durch verschiedene Diagnosemodule kann das Diagnosesystem mehrere Diagnoseverfahren zeitgleich starten. Eine abschließende Bewertung der einzelnen Diagnoseergebnisse wird durch den Diagnosemanager übernommen.

## 3.4 Diagnosemodul

In einem Diagnosemodul ist ein Diagnoseverfahren umgesetzt. Diagnosemodule sind untereinander unabhängig und kommunizieren nur mit dem übergeordneten Diagnosemanager. Alle notwendigen Ressourcen für eine Diagnose wie Datenbanken oder andere Informationsquellen sind in einem Diagnosemodul gekapselt. Aus diesem Grund können Diagnosemodule ohne Seiteneffekte im System eingebunden beziehungsweise entfernt werden.

Jedes Diagnosemodul stellt dem Diagnosemanager die gleichen Schnittstellen zur Verfügung. Über diese Schnittstellen kann der Diagnosemanager das Diagnosemodul steuern.

## 3.5 Fallbasis

In der Fallbasis werden alle Anfragen an das Diagnosesystem gespeichert. Damit beinhaltet die Fallbasis die Erfahrung des Diagnosesystems, da auch alle eingetretenen Fehlerfälle darin gespeichert sind. Darüber hinaus kann die Fallbasis dazu verwendet werden einen Verlauf der bisher erfolgten Anfragen an das Diagnosesystem zu erstellen. Aussagen über das Nutzerverhalten des Diagnosesystems lassen sich damit treffen.

In [WAIB14] ist die Fallbasis als relationale Datenbank modelliert und beinhaltet die eingetretenen nicht redundanten Fehlerfälle. In dieser Arbeit wird die Fallbasis jedoch als dokumentenorientierte Datenbank modelliert. Eine dokumentenorientierte Datenbank ist im Vergleich zu einer relationalen Datenbank flexibler, da kein festes Datenschema eingehalten werden muss (siehe Kapitel 2.4). Diagnoseanfragen sowie Feedbackanfragen werden als komplettes JSON-File in der Datenbank hinterlegt. Ändert sich beispielsweise die Struktur einer Diagnoseabfrage durch hinzukommen eines neuen Symptoms, sind keine Änderungen an der Datenbank notwendig. Jedes JSON-File ist ein eigenständiger Eintrag in der Datenbank und unabhängig von anderen Einträgen. Die Master Slave Replikation [WAIB14] für die zentrale Verwaltung von Fehlerfällen kann durch Verwendung bestehender Replikationsmechanismen der Datenbank erreicht werden [ODB15]. Eine Versionsverwaltung der Datenbank für Fehlerfälle über die manuelle Vergabe von Nummern ist nicht mehr notwendig. Die eingesetzte Datenbank besitzt bereits eine integrierte Versionsverwaltung für jeden Eintrag in der Datenbank [ODB15].

## 3.6 Fehlermodell

Das Fehlermodell wird als Graph in einer Graph-Datenbank (OrientDB) gespeichert und repräsentiert die Symptomzusammenhänge in dem Prüfsystem. Es entstand in einer Kooperation zwischen dem Institut für Automatisierung- und Softwaretechnik der Universität Stuttgart und der Audi AG. Bei der Erstellung wurden die Kausalzusammenhänge zwischen den einzelnen Symptomen modelliert. Das Fehlermodell besteht aus Knoten und Kanten. Die Knoten repräsentieren Fehlersymptome beziehungsweise Fehlerursachen, die Kanten stehen für Zusammenhänge zwischen den einzelnen Knoten. Ausgehende Kanten eines Knoten beschreiben die möglichen Folgenfehlersymptome. Eingehende Kanten beschreiben die Eintrittsbedingungen des Knoten. Im Zuge dieser Arbeit wird das Fehlermodell um die bedingte Eintrittswahrscheinlichkeit zwischen den Knoten erweitert. Dadurch ist eine wahrscheinlichkeitsbasierte Diagnose möglich. Jeder Knoten des Fehlermodells besitzt folgende Parameter

Tabelle 3.2: Parameter für Knoten im Fehlermodell

| **Parametername** | **Bedeutung** |
|---|---|

| | |
|---|---|
| id | Eindeutiger Identifikator in der Datenbank |
| name | Beschreibung des Knoten |
| probability (nur Fehlersymptome) | Wahrscheinlichkeit für das Eintreten des Symptoms |

Katen beinhalten die bedingten Wahrscheinlichkeiten zwischen Start und Zielkonten. Ausgehend von einem Knoten beschreibt die bedingte Wahrscheinlichkeit das Eintrittsverhalten der Folgesymptome in Anhängigkeit des aktuellen Symptoms. Folgendes Abbild soll dies Beispielhaft veranschaulichen.

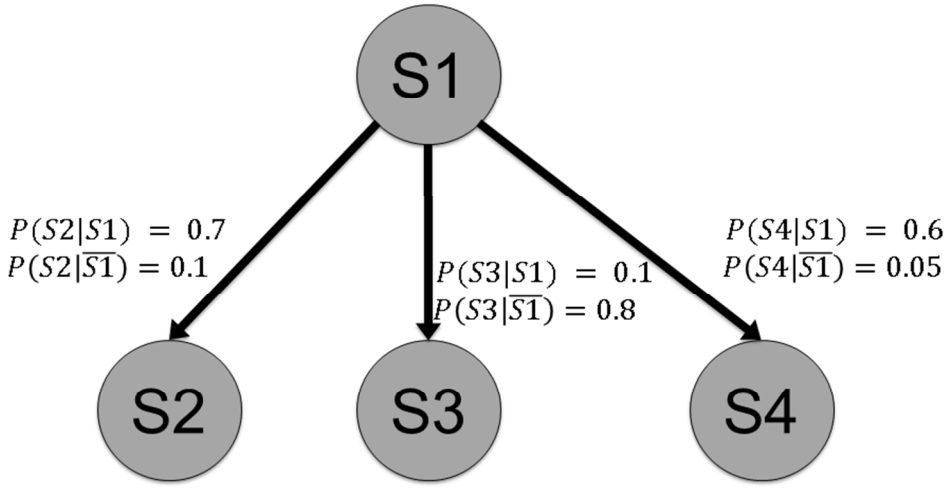



Abbildung 3.6: bedingte Wahrscheinlichkeit bei Symptomen

Ist Symptom S1 eingetreten liegt die Wahrscheinlichkeit für das Eintreten vonS2 bei 70%. Im Umkehrschluss bedeutet dies auch, dass in 30% der Fälle in denen S1 Eingetreten ist, S2 nicht auftrat. Ist S1 hingegen nicht eingetreten liegt die Wahrscheinlichkeit für das Eintreten von S3 bei 80%. Eine Eintrittsableitung von Symptomen auf Basis nicht eingetretener Symptome ist aufgrund der Eintrittsbedingungen mancher Symptome notwendig. Hierbei gilt für gewisse Symptome folgendes.

$$P(U1|S1, S3) = x$$

$$P\left(U1\middle|S_1, \overline{S3}\right) = y$$

$$y \geq x$$

Die Wahrscheinlichkeit für das Eintreten der Ursache $U1$ anhand des Symptompfades $S1 \rightarrow S3 \rightarrow U1$ ist größer unter der Voraussetzung dass, das Symptom $S3$ nicht eingetreten ist. Mithilfe des modifizierten Fehlermodells aus Abbildung 2.2 lässt sich dies an einem Beispiel veranschaulichen.

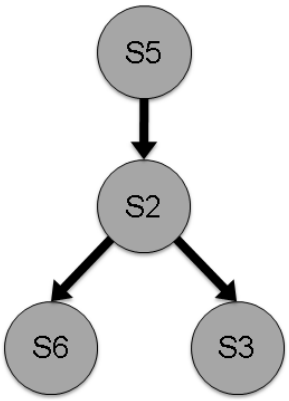



Abbildung 3.7: bedingte Wahrscheinlichkeit bei Symptomen (Beispiel)

Ist ein Motorschaden eingetreten kann durch eine Überprüfung der Drehzahl die Ursache ermittelt werden. Ist das Symptom einer erhöhten Drehzahl eingetreten, liegt die Ursache des Motorschadens an einer Motorüberhitzung (S6). Bei einer normalen Drehzahl kann von einer mangelnden Schmierung des Motors ausgegangen werden. Damit ergeben sich folgende bedingte Wahrscheinlichkeiten.

$$P(S3|S5, \overline{S2}) \gg P(S3|S5, S2)$$

$$P(S6|S5, \overline{S2}) \ll P(S6|S5, S2)$$

Für diese Arbeit wird zu Testzwecken ein abgeschlossener Teilbereich des Fehlermodelles verwendet. Abbildung 3.8 zeigt den verwendeten Teilbereich. Das Wurzelsymptom des Fehlermodells ist S.ODX.1. Alle Symptompfade beginnen mit diesem Symptom.

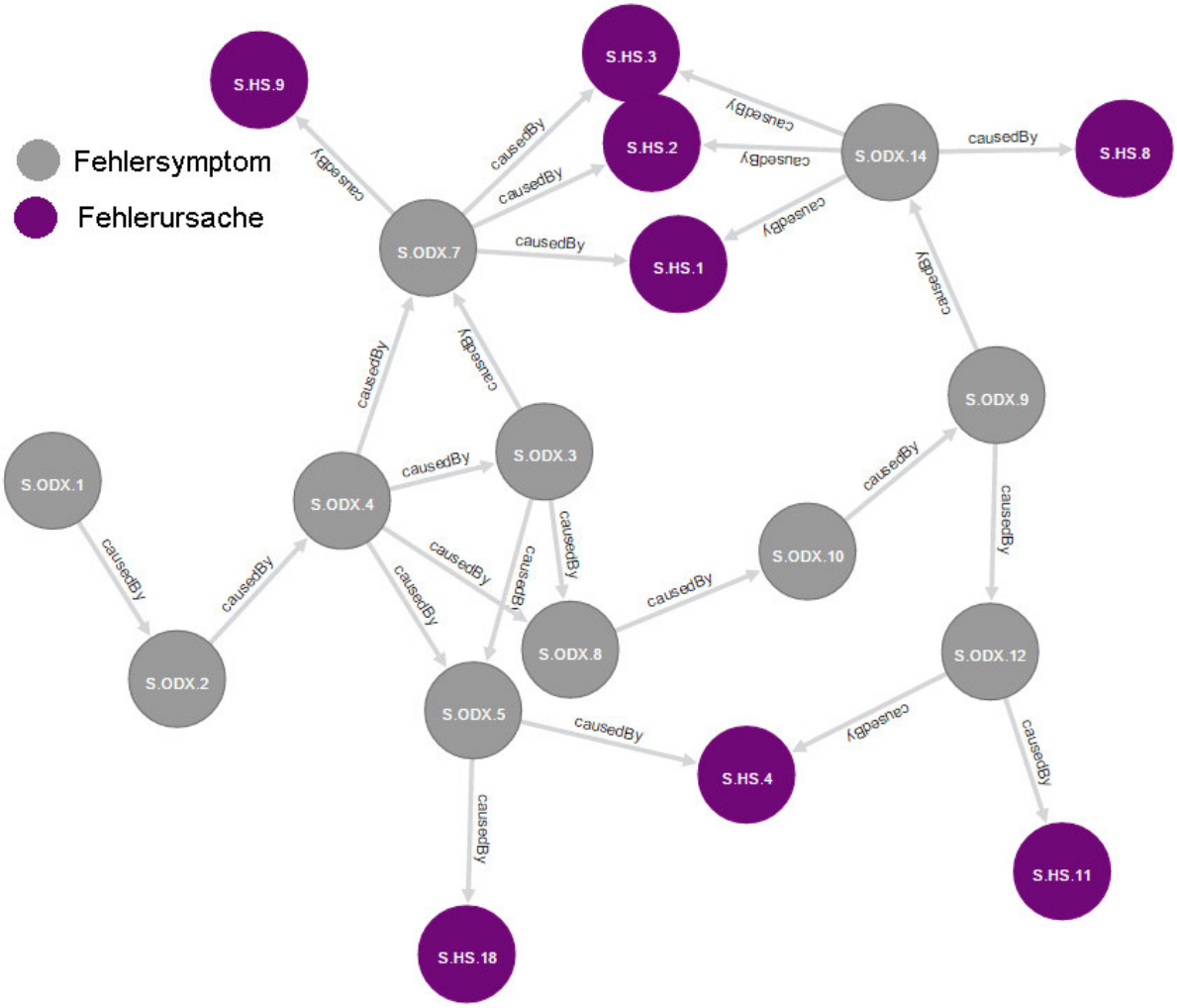



Abbildung 3.8: Fehlermodell

## 3.7 Fehlerdiagnose

Bei der Diagnose wird versucht auf Basis des erhaltenen Fehlerfalls die wahrscheinlichsten Symptompfade und Ursachen zu finden. Im Fehlerfall selbst haben nicht alle Symptome den Zustand „eingetreten“. Dies liegt mitunter daran, dass die Symptome tatsächlich nicht eingetreten sind oder dass sie falsch aus den Messreihen extrahiert wurden. Durch diese Unsicherheit ist eine Diagnose nur unter Verwendung eines einfachen Fehlermodells ohne zusätzliche Informationen wie beispielsweise bedingten Eintrittswahrscheinlichkeiten unzureichend möglich. Über bedingte Wahrscheinlichkeiten lassen sich Rückschlüsse über den Status der nicht eingetretenen Symptome treffen. Eine konkretere Aussage über den Status von Symptomen lässt eine Generierung zusammenhängender Symptompfade zu. Im Folgenden werden die einzelnen Schritte des Diagnosealgorithmus vorgestellt. Danach wird auf die möglichen Risiken und mögliche Probleme eingegangen.

### 3.7.1 Diagnosealgorithmus

Der erste Schritt einer Diagnose besteht in der Speicherung aller erhaltenen Symptome in einer Liste. Bei der Speicherung wird jedem Symptom, welches als eingetreten markiert übermittelt wurde, ein Score-Wert von 50 zugewiesen. Symptome, die als nicht eingetreten markiert wurden, erhalten den Score-Wert von 0. Der Score-Wert repräsentiert das aktuelle Wissen des Diagnosemoduls über den Status eines Symptoms. Im weiteren Verlauf der Diagnose wird der Score-Wert auch für die Bewertung der gefundenen Symptompfade verwendet. Der Wertebereich des Score-Wertes liegt im Intervall von -50 bis 50. Folgende Tabelle listet die Bedeutung der einzelnen Score-Werte auf.

Tabelle 3.3: Score-Werte

| **Score-Wert** | **Bedeutung** |
|---|---|
| 50 | Symptom ist sicher eingetreten. |
| 0 | Keine Aussagen über den Status des Symptoms möglich |
| -50 | Symptom ist sicher nicht eingetreten. |

Im zweiten Schritt der Diagnose erfolgt eine Berechnung sämtlicher Score-Werte die ungleich 50 sind. Das Verfahren bricht ab, sobald für keine Symptome neue Score Werte mehr berechnet werden können. Der Ablauf dieses Diagnoseschritts ist in folgendem Ablaufdiagramm dargestellt.

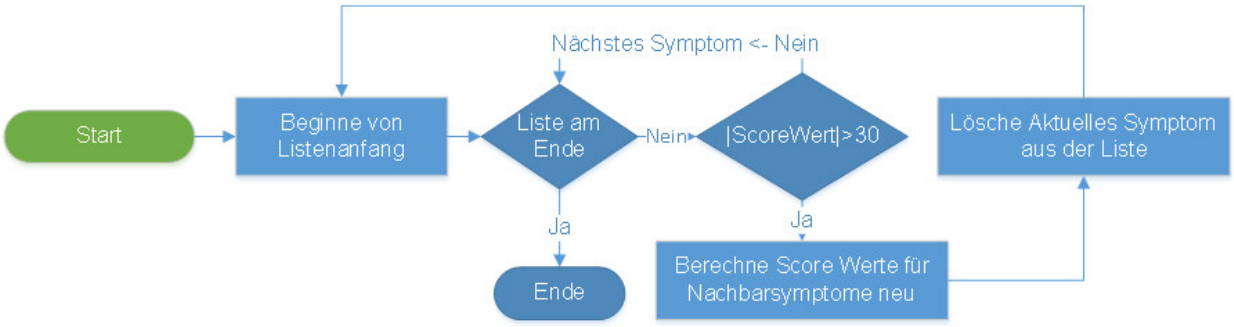



Abbildung 3.9: Diagnosealgorithmus Score-Wert Berechnung

Im ersten Iterationsschritt erhalten nur die Symptome neue Score Werte, die in direkter Nachbarschaft zu einem Symptom mit einem Score-Wert von 50 stehen. Neue Score-Werte werden nur mithilfe von Symptomen berechnet, deren eigener Status ein bestimmtes Maß an Gewissheit aufweist. Im Folgenden werden solche Symptome Hauptsymptome genannt.

Jedes Symptom wird nur einmal für die Berechnung der Score-Werte seiner Nachbarsymptome verwendet. Dadurch wird eine wiederholte Einflussnahme eines Symptoms auf die Score-Werte seiner Nachbarsymptome verhindert. Nach dem Entfernen eines Symptoms aus der Liste wird die Liste wieder von Beginn abgearbeitet. Symptome die beim letzten Durchlauf einen zu geringen Score-Wert hatten, können so für eine Berechnung nochmals überprüft werden.

Ist ein Symptom für die Berechnung von Score-Werten geeignet, wird für jedes seiner Nachbarsymptome der Einfluss auf den Score-Wert berechnet. Als Nachbarsymptome gelten Symptome die über eine Kante direkt mit dem Symptom verbunden sind. Abbildung 3.10 stellt die möglichen Verbindungskonstellationen die bei der Berechnung eintreten können dar. Die Hauptsymptome sind grau hinterlegt.

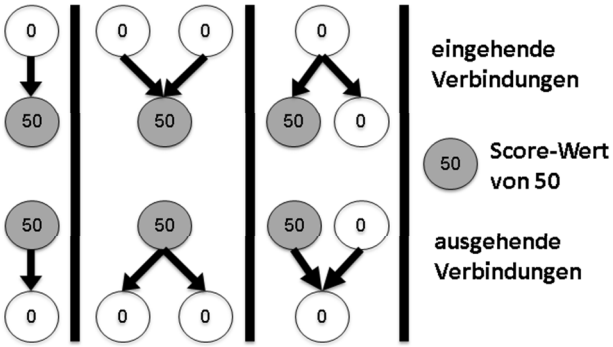



Abbildung 3.10: Verbindungskonstellationen bei Score-Wert Berechnung

Eine Score-Wert Berechnung für Symptome die mit dem Hauptsymptom durch eine Kante verbunden sind erfolgt in 3 Schritten.

**Schritt 1: Status Hauptsymptom**

Der Status des Hauptsymptoms hat direkten Einfluss auf die umliegenden Symptome und somit auch auf die Score-Werte. Im ersten Schritt der Score-Wert Berechnung wird festgestellt, ob das Hauptsymptom vom Diagnosemodul als eingetreten oder als nicht eingetreten angesehen wird.

**Schritt 2: Größte bedingte Wahrscheinlichkeit**

Eine ausgehende Verbindung zwischen Hauptsymptom und Symptom beinhaltet zwei bedingte Wahrscheinlichkeiten.

$$P(S|H)$$

$$P(S\,|\,\overline{H})$$

Für die Bestimmung der größten bedingten Wahrscheinlichkeit spielt der Status des Hauptsymptoms eine Rolle. Folgende Tabelle repräsentiert die Lösungsmatrix für die Ermittlung der größten bedingten Wahrscheinlichkeit im Falle einer ausgehenden Verbindung zwischen Hauptsymptom und Symptom.

Tabelle 3.4: größte bedingte Wahrscheinlichkeit bei ausgehender Verbindung

| Status Hauptsymptom | Auswertung | Auswertung Wahr | Auswertung Falsch |
|---|---|---|---|
| eingetreten | $P(S\|H) > P(\overline{S}\|H)$ | $P(S\|H)$ | $P(\overline{S}\|H)$ |
| nicht eingetreten | $P(S\|\overline{H}) > P(\overline{S}\|\overline{H})$ | $P(S\|\overline{H})$ | $P(\overline{S}\|\overline{H})$ |

Wird in Schritt 1 der Status des Hauptsymptoms als eingetreten festgelegt, werden die bedingten Wahrscheinlichkeiten $P(S|H)$ und $P(\overline{S}|H)$ verglichen. Wobei $P(\overline{S}|H) = 1 - P(S|H)$ gilt. Daraus folgt, dass einer Überprüfung von $P(S|H) > 0.5$ ausreicht, um die größte bedingte Wahrscheinlichkeit zu ermitteln.

Eine Bestimmung der größten bedingten Wahrscheinlichkeit bei einer eingehenden Verbindung erfolgt auf identischem Weg. Jedoch müssen hierbei die bedingten Wahrscheinlichkeiten mithilfe des *Satz von Bayes* [BAYE15] umgewandelt werden, da eine eingehende Verbindung zwischen Symptom und Hauptsymptom folgende bedingte Wahrscheinlichkeiten beinhaltet.

$$P(H|S)$$

$$P(H\,|\,\overline{S})$$

Für die Berechnung der Score-Werte werden jedoch bedingte Wahrscheinlichkeiten der Form $P(S|H)$ benötigt. Eine Umformung erfolgt wie folgt:

$$P(H|S) = \frac{P(S \cap H)}{P(S)}$$

$$P(H|S) = \frac{\frac{P(S \cap H)}{P(H)} P(H)}{P(S)} \; (Z\ddot{a}hler * \frac{P(H)}{P(H)})$$

hierbei gilt

$$P(S|H) = \frac{P(S \cap H)}{P(H)}$$

$$P(H|S) = \frac{P(S|H)}{P(S)} * P(H)(einsetzen)$$

$$\frac{P(H|S)}{P(H)} * P(S) = P(S|H)(umformen)$$

Für die Umformung von $P(H\,|\,\overline{S})$ erfolgt auf identischem Weg. $(H|\overline{S})$ zu $P(S|\overline{H})$ bedarf es einiger zusätzlicher Schritte.

$$P(\overline{H}|\overline{S}) = 1 - P(H\,|\,\overline{S})$$

$$P(\overline{H}|\overline{S}) \xrightarrow{Satz\ von\ Bayes} P(\overline{S}|\overline{H})$$

$$P(S|\overline{H}) = 1 - P(\overline{S}|\overline{H})$$

Die notwendigen Wahrscheinlichkeiten $P(H)$ und $P(S)$ und damit auch die Umkehrwahrscheinlichkeiten $P(\overline{H})$ und $P(\overline{S})$ sind in den dazugehörigen Knoten des Fehlermodells gespeichert (siehe Tabelle 3.2). Damit sind alle notwendigen Informationen für eine Umformung vorhanden. Die größte bedingte Wahrscheinlichkeit für eingehende

Verbindungen zwischen Symptom und Hauptkonten können nach Tabelle 3.4 bestimmen werden.

**Schritt 3: Berechnung Score-Wert**

Im Schritt 1 und 2 erfolgte die Bestimmung der notwendigen Parameter für die Berechnung des Score-Wertes. In Schritt 3 wird die Berechnung anhand folgender Tabelle veranschaulicht. In der Tabelle wird der Score-Wert des Hauptsymptoms als ScH und der Score-Wert des Symptoms als ScS dargestellt.

Tabelle 3.5: Score-Wert Berechnung

| **Status Hauptsymptom** | **größte Wahrscheinlichkeit** | **Berechnung** |
|---|---|---|
| eingetreten | $P(S\|H)$ | $ScS = ScS + P(S\|H) * ScH$ |
| eingetreten | $P(\overline{S}\|H)$ | $ScS = ScS - P(\overline{S}\|H) * ScH$ |
| nicht eingetreten | $P(S\|\overline{H})$ | $ScS = ScS + P(S\|\overline{H}) * ScH$ |
| nicht eingetreten | $P(\overline{S}\|\overline{H})$ | $ScS = ScS - P(\overline{S}\|\overline{H}) * ScH$ |

Für die Berechnung der Score-Werte werden folgende zusätzliche Regelungen definiert um gewisse Sonderfällt zu behandeln beziehungsweise das Verhalten unter bestimmten Bedingungen zu regeln.

**Begrenzung**

Score-Werte dürfen durch die Berechnung den Wert |50| nicht übersteigen. Die Score-Werte lassen sich dadurch auch nach der Berechnung vergleichen.

**Mindestmaß an Änderung**

Die Differenz zwischen neuem und altem Score-Wert muss einen bestimmten Betrag übersteigen. Eine Aufsummierung vieler kleiner Änderungen zu einer Großen und damit zu einer Fehlinterpretation des Score-Wertes ist somit nicht möglich.

**Wechselwirkung vermeiden**

Symptome die den Score-Wert ihrer Nachbarsymptome beeinflusst haben dürfen nicht mehr durch selbige beeinflusst werden. Dadurch wird eine künstliche Verstärkung der Score-Werte verhindert.

**Vorzeichen Änderung**

Hat der Score-Wert eines Symptoms aufgrund des Einflusses seiner Nachbarsymptome einen bestimmten Wert über bzw. unterschritten darf die Differenz zwischen |Score-Wert| und 50 nur

kleiner werden. Damit ist sichergestellt, dass ein bereits als sicher geltender Symptomstatus nicht erneut unsicher wird.

**Widerspruch erkennen**

Würde ein Score-Wert der größer als |38| ist, durch eine Berechnung wieder annähernd auf 0 gebracht, deutet dies auf einen Widerspruch hin. Folgende Abbildung stellt dies anhand eines Beispiels dar.

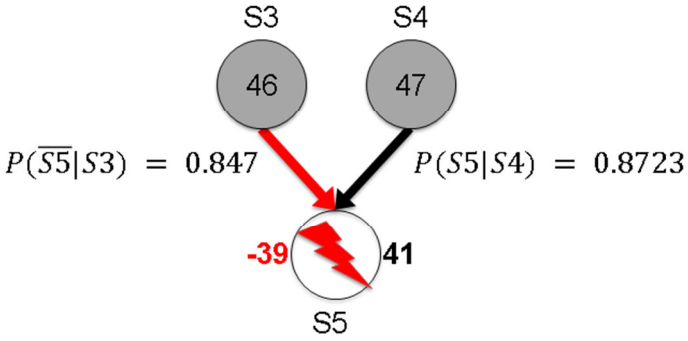



Abbildung 3.11: Beispiel Widerspruch Score-Wert Berechnung

Eine Erkennung eines Widerspruchs kann somit dem Anwender der Diagnosesoftware gemeldet werden. Worauf hin die betroffenen Symptome manuell durch den Anwender überprüft werden können um dann erneut die Diagnose mit einem angepassten Fehlerfall zu starten

Folgendes Schaubild veranschaulicht die Berechnung der einzelnen Score-Werte in einem Fehlermodell. Als Fehlermodell wurde das Beispiel aus Abbildung 2.2 gewählt. In der Ausgangssituation ist das Symptom S3 (unzureichende Schmierung) als eingetreten markiert und hat somit einen Score-Wert von 50. Alle anderen Symptome sind nicht eingetreten. Auf die Darstellung der bedingten Wahrscheinlichkeiten wurde aus Platzgründen verzichtet.

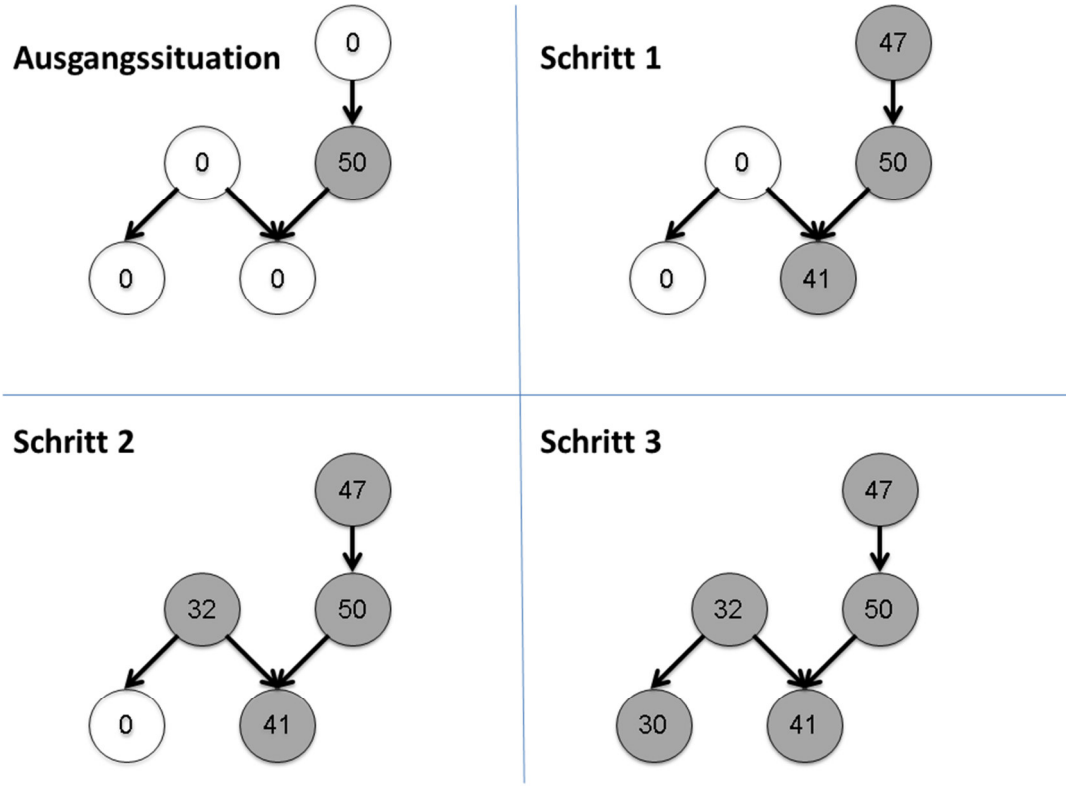



Abbildung 3.12: Beispiel Score-Wert Berechnung

Der letzte Schritt einer Diagnose besteht in der Bestimmung der wahrscheinlichsten Symptompfade inklusive Fehlerursachen. Hierfür wird jeder Knoten des Fehlermodells abgearbeitet. Für die Traversierung des Fehlermodells wird eine Breitensuche [PEPP05] verwendet. Bei einer Breitensuche werden zuerst alle Kindknoten des Einstiegsknoten abgearbeitet. Danach werden die Folgeknoten abgearbeitet. Als Einstiegsknoten werden die Wurzelknoten gewählt.

Bei der Bearbeitung eines Knoten im Fehlermodell wird auf Score-Wert > |minSicherheit| geprüft. Die |minSicherheit| gibt für einen Knoten das Mindestmaß an Gewissheit über seinen Status an. Erfüllt ein Knoten dieses Mindestmaß nicht, werden dieser Knoten und alle seine Folgeknoten von der Suche nicht mehr abgearbeitet. Auf diese Weiße erzeugt die Suche nur Symptompfade, die ausschließlich aus Knoten bestehen deren Status durch die Berechnung bekannt ist. Erfüllt ein Knoten dieses Kriterium, wird der Betrag seines Score-Werts zu dem Score-Wert der bisher durchlaufenen Knoten addiert. Mithilfe der in Abbildung 3.12 berechneten Score-Werte wird im folgenden Abbild das Vorgehen bei einer Symptompfadsuche aufgezeigt. Für eine Suche von Symptom zu Ursache wurden die Verbindungspfeile umgekehrt.

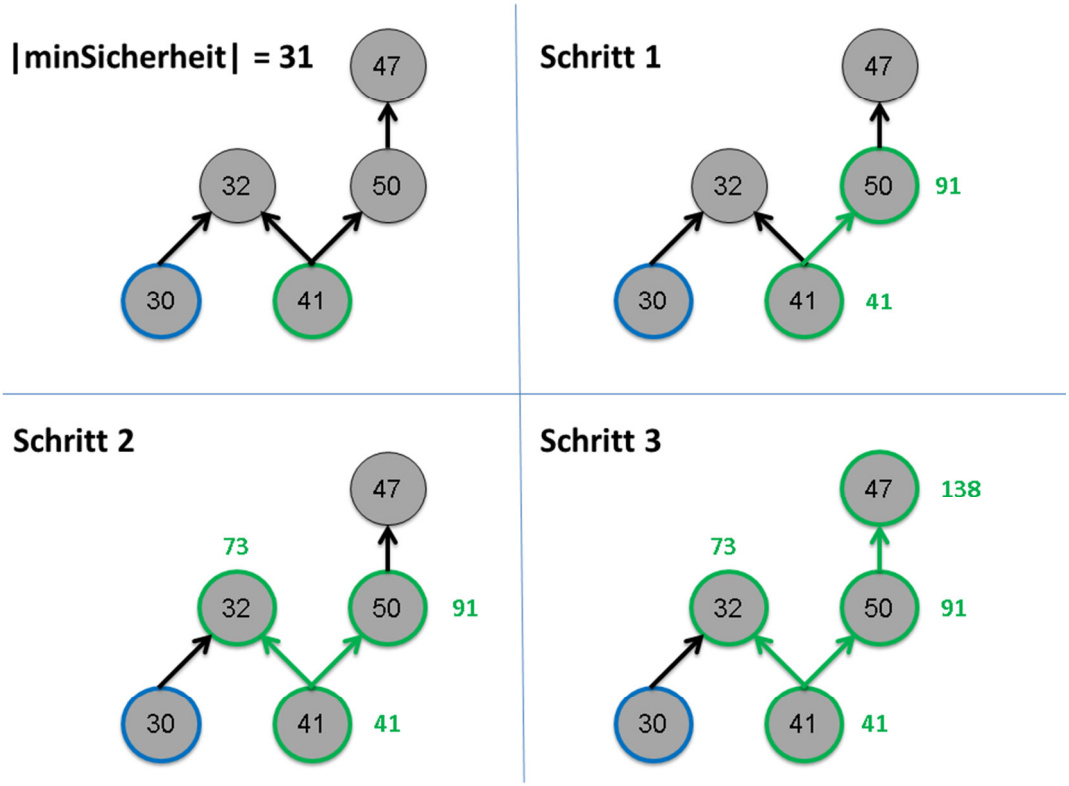



Abbildung 3.13: Beispiel Symptompfadsuche

Die durch die Suche erzeugte Symptompfadliste enthält die gefundenen Symptompfade mit dazugehörigen Score-Werten. Für die Bewertung der einzelnen Pfade wird der durchschnittliche Score-Wert der einzelnen Symptome im Pfad berechnet. Dadurch hat die Länge des Pfades keinen Einfluss auf die spätere Positionierung in der sortierten Ergebnisliste. Die Berechnung des durchschnittlichen Score-Werts der gefundenen Pfade aus Abbildung 3.13 ergibt folgende Ergebnisse. Der Pfad 41 → 50 → 47 hat einen durchschnittlichen Score-Wert von 46, der Pfad 41 → 32 hat den Durchschnittswert 36,5. Eine bessere Interpretation der Zahlen wir durch die Umrechnung in Prozent erreicht. Der maximale durchschnittliche Score-Wert liegt bei 50. Damit berechnet sich die Wahrscheinlichkeit für das Eintreten des Symptompfades mithilfe folgender Formel:

$$P(Symptompfad) = 1 - \frac{50 - Symptompfad_{\emptyset\ Score-Wert}}{50}$$

### 3.7.2 Risiken und mögliche Probleme

Nicht eingetretene Symptome können für ein korrektes Ergebnis einer Diagnose von Bedeutung sein. Dies wird durch die Abbildung 3.7 verdeutlicht. Ist das Symptom S2 eingetreten, ist die Wahrscheinlichkeit für das Eintreten von S6 deutlich höher als von S3. Ist S2 hingegen nicht eingetreten tritt S3 im Vergleich zu S6 mit größerer Wahrscheinlichkeit aus. Der Status von S2 ist damit ausschlaggebend für die Bestimmung der Fehlerursache. Erhält ein solches Symptom bei der Symptomextraktion fälschlicherweise den Status eingetreten anstelle von nicht eingetreten, kann der Diagnosealgorithmus die tatsächliche Ursache nicht bestimmen. Existiert

jedoch ein weiterer Pfad im Fehlerbaum zu der tatsächlichen Ursache, besteht die Möglichkeit, den falschen Status des Symptoms über einen Widerspruch bei der Berechnung der Score-Werte zu bemerken.

## 3.8 Fehlerprävention

Im Prüfsystem können bei einer Fahrzeugprüfung auch Fehlersymptome eintreten ohne dass dies ein sofortiges Fehlschlagen der Fahrzeugprüfung zur Folge hat. Beispielsweise führt der Ausfall der Netzwerkverbindung nicht in allen Fällen sofort zu einer fehlerhaften Fahrzeugüberprüfung. Dies gilt nur unter der Voraussetzung, dass noch nicht alle notwendigen Daten für die Fahrzeugüberprüfung geladen worden sind. Ändert sich bei der nächsten Überprüfung der Fahrzeug-Typ, müssen neue Fahrzeuginformationen geladen werden. Bei einer fehlenden Netzwerkverbindung ist dies nicht möglich und die Fahrzeugdiagnose schlägt fehl. Eine Fehlerpräventionsanalyse kann in bestimmten Fällen auf zukünftigen Fehler hinweisen. Dadurch hat der Anwender die Möglichkeit die Fehlerursache frühzeitig zu erkennen und zu beheben.

### 3.8.1 Fehlermodell für die Fehlerprävention

Bei der Fehlerdiagnose wird versucht anhand von Symptomen auf Fehlerursachen zu Schließen. Bei der Fehlerprävention wird versucht auf Basis von Symptomen auf zukünftig eintretende Symptome und damit auch auf zukünftige Fehlerursachen zu schließen. Der in Kapitel 3.7.1 beschriebenen Diagnosealgorithmus verwendet für die Diagnose ein Fehlermodell welches den Zusammenhang zwischen Symptomen wiederspiegelt. Die Berechnung der Eintrittswahrscheinlichkeiten basiert auf einzelnen Systemzuständen. Eintrittsverhalten über mehrere Systemzustände hinweg wurden für die Berechnung nicht beachtet. Das bedeutet, ist der Status eins Symptoms bekannt, kann mithilfe des Fehlermodells lediglich die Wahrscheinlichkeit berechnete werden ob ein Nachbarsymptom im Laufe einer Fahrzeugprüfung auch eingetreten ist. Aussagen über das Eintrittsverhalten der Nachbarsymptome im Verlauf folgender Fahrzeugüberprüfungen sind mit dem Fehlermodell nicht sicher zu treffen. Damit ist das Fehlermodell für eine Fehlerprävention nicht einsetzbar.

### 3.8.2 Fehlerpräventionsalgorithmus

Für die Fehlerprävention werden nur die Symptome mit dem Status eingetreten verwendet. Eine Vorhersage von zukünftig eintretenden Symptomen auf Basis von momentan nicht eingetretenen Symptomen wird durch den Präventionsalgorithmus nicht abgedeckt. Die Prävention wird für jedes eingetretene Symptom separat durchgeführt. Aus diesem Grund besteht der erste Schritt einer Prävention darin die als eingetreten markierten Symptome aus dem übermittelten Systemzustand zu extrahieren und in einer Liste zu speichern. Für jedes Symptom werden im Anschluss folgende Schritte durchgeführt.

**Schritt 1: Extraktion der Präventionsdaten aus der Wissensbasis**

In Abhängigkeit des Symptoms wird in der Wissensdatenbank nach Diagnoseanfragen gesucht, in denen das Symptom ebenfalls als eingetreten markiert war. Jede Diagnoseanfrage ist mit einem Datum versehen. Auf Basis dieses Datums werden mithilfe einer Abfrage alle Diagnoseanfragen aus der Datenbank selektiert die ausgehend von dem Datum in einem abgegrenzten Bereich in der Zukunft liegen. Dieser Bereich steht für die Reichweite der Prävention. Folgende Abbildung veranschaulicht das Verfahren.

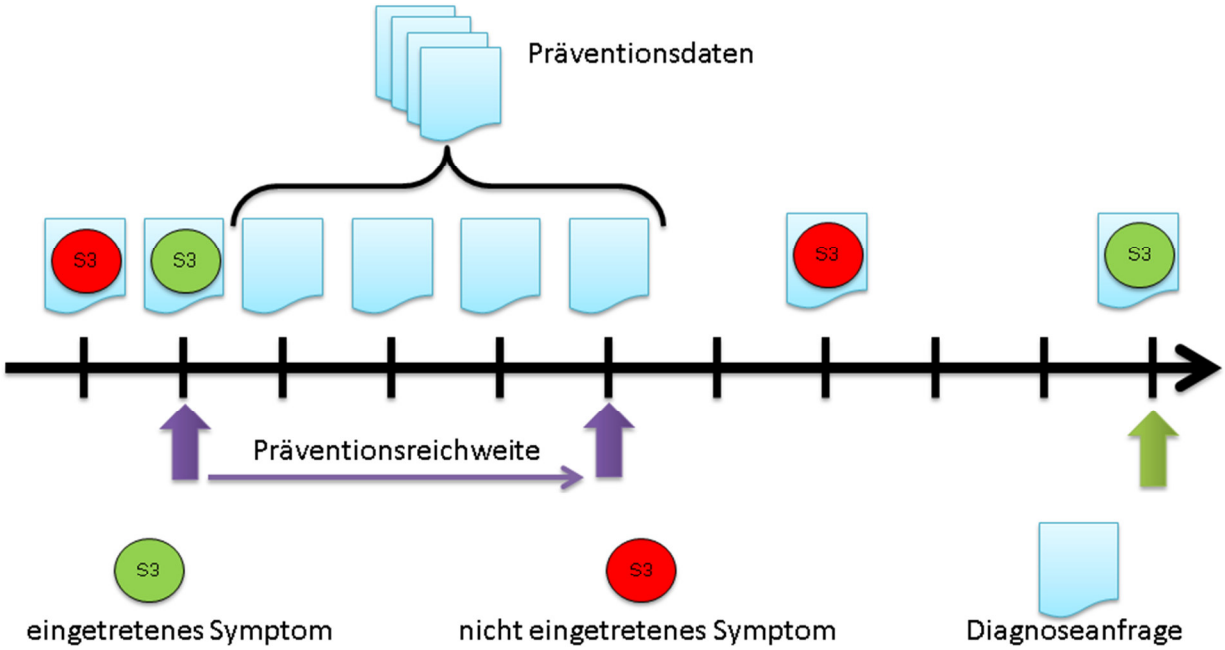



Abbildung 3.14: Extraktion der Präventionsdaten aus der Wissensbasis

Der schwarze Pfeil stellt eine Zeitlinie dar, die die chronologische Reihenfolge der Diagnoseanfragen, die sich in der Wissensbasis befinden, wiedergibt. Der grüne Pfeil markiert die aktuelle Diagnoseanfrage von der aus in der Wissensbasis nach passenden Präventionsdaten gesucht wird. Die lila Pfeile markieren den zeitlichen Start und das Ende der Präventionsreichweite.

Für jede Diagnoseanfrage in der das aktuelle Symptom als eingetreten markiert war, werden alle Diagnoseanfragen in der Präventionsreichweite aus der Datenbank selektiert.

**Schritt 2: Verarbeitung Präventionsdaten**

Für die Prävention wird jede Diagnoseanfrage aus Schritt 1 analysiert. Hierbei werden die Diagnoseanfragen nach eingetretenen Symptomen durchsucht. Wird ein eingetretenes Symptom gefunden, wird es in eine Liste geschrieben. Steht das Symptom bereits in der Liste wird der Zählt für das Symptom um 1 erhöht. Auf diese Weiße entsteht im Verlauf der Verarbeitung eine Liste mit Symptomen und der Häufigkeit ihres Auftretens im Präventionsbereich.

**Schritt 3: Interpretation der Daten**

Ob und inwieweit das Eintreten eines Symptoms Einfluss auf das zukünftige Eintreten eines anderen hat, kann durch eine Analyse der in Schritt 2 erstellten Liste aus Symptomen und ihrer

Eintrittshäufigkeit bestimmt werden. Zur Veranschaulichung sind in Abbildung 3.15 zwei verschiedene Listen aus Schritt 2 in einem Diagramm dargestellt. Entlang der X-Achse sind die im Präventionsbereich gefundenen Symptome aufgelistet. Das Symptom S3 ist nicht aufgelistet, da die Prävention für dieses Symptom ausgeführt wird. An der Y-Achse kann die Häufigkeit des Eintretens der Symptome abgelesen werden.

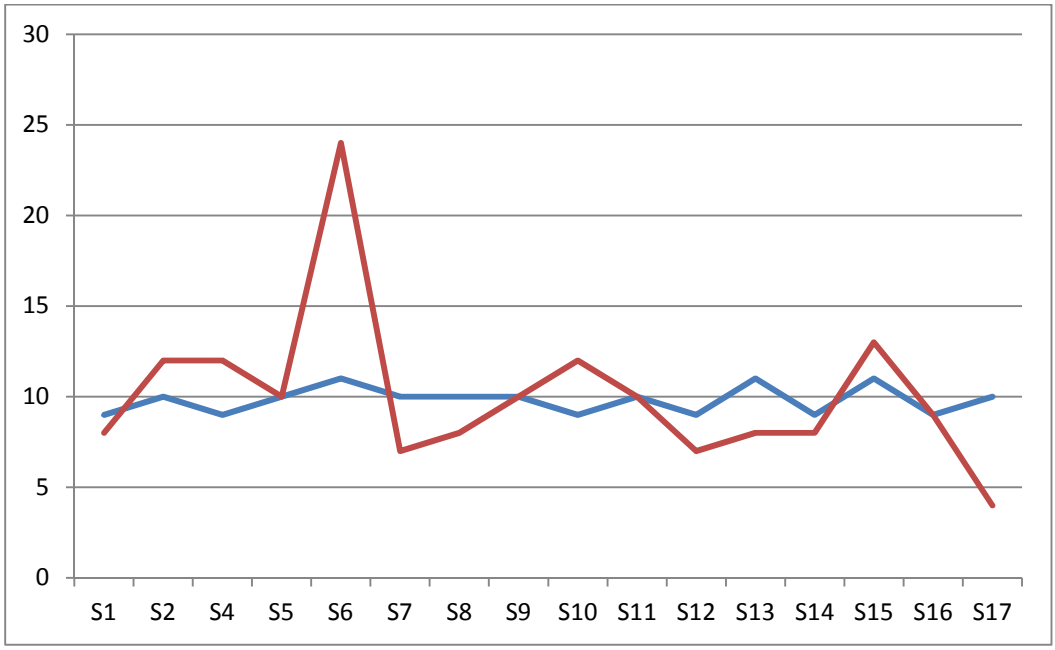



Abbildung 3.15: Eintrittshäufigkeit der Symptome im Präventionsbereich

Erzeugt Schritt 2 eine Liste wie sie durch die blaue Linie repräsentiert wird lässt sich keine Aussage über das Eintreten zukünftiger Symptome Treffen. Jedes gefundene Symptom tritt annähern mit derselben Häufigkeit ein. Dies kann zwei Ursachen haben. Zum einen besteht die Möglichkeit, dass das Eintreten des Symptoms S3 Einfluss auf alle gefundenen Symptome hat. Zum anderen ist es auch möglich, dass überhaupt kein Zusammenhang zwischen S3 und den gefundenen Symptomen besteht. Der Fall das S3 Einfluss auf alle anderen nimmt ist sehr unwahrscheinlich und wird daher in der Fehlerprävention als unmöglich eingestuft.

Wird in Schritt 2 hingegen eine Liste, wie sie durch die rote Linie dargestellt wird, erzeugt lässt sich ein Zusammenhang zwischen S3 und S6 erkennen. Das häufige Auftreten von S6 in der Präventionsreichweite im Vergleich zu den anderen Symptomen deutet auf einen Zusammenhang hin. Für eine automatisierte Erkennung solcher Zusammenhänge wird zuerst der Durchschnitt der Eintrittshäufigkeit aller gefundenen Symptome berechnet. Im Anschluss wird für jedes Symptom überprüft wie groß der Unterschied zwischen der eigenen und der durchschnittlichen Eintrittshäufigkeit ist. Übersteigt der Unterschied einen gewissen Schwellwert kann von einem Zusammenhang ausgegangen werden. Der Schwellwert gibt die notwendige Differenz in Prozent an.

Die durchschnittliche Eintrittshäufigkeit der roten Linie liegt bei 10,125. Damit liegt die Differenz zwischen eigener und durchschnittlicher Eintrittshäufigkeit für S6 bei 13,875. Liegt

der Schwellwert beispielsweise bei 100%, würde hier ein Zusammenhang zwischen S3 und S6 erkannt werden.

Die Festlegung auf einen Schwellwert sowie die Bestimmung der Präventionsreichweite kann nur im Rahmen einer praktischen Erprobung und unter realen Bedingungen stattfinden. Ein zu hoch oder zu niedrig angesetzter Schwellwert kann dazu führen, dass bestehende Zusammenhänge nicht oder zu häufig erkannt werden. Die Größe der Präventionsreichweite hat Einfluss auf die Anzahl der gefundenen Präventionsdaten. Wobei die Reichweite nicht zu groß gewählt werden darf. Mit größer werdendem Abstand zwischen gefundener Diagnoseanfrage und dem Start der Reichweite wird das Finden von Zusammenhängen immer unrealistischer, da zwischenzeitlich zahlreiche andere Einflüsse auf das Prüfsystem wirken konnten und somit das Eintreten der Symptome beeinflussten.

### 3.8.3 Risiken und mögliche Probleme

Eine Fehlerprävention kann zu einer Produktivitätssteigerung des Prüfers sowie des Prüfstandes beitragen. Vor potentiell auftretenden Fehler wird frühzeitig gewarnt und der Prüfer kann die notwendigen Schritte einleiten um den Fehler erst gar nicht entstehen zu lassen. Jedoch haben falsche Präventionsaussagen genau den entgegen gesetztem Effekt. Wird zu häufig vor potentiellen Fehlern gewarnt ist der Prüfer durch die Ausübung der Maßnahmen zur Fehler Vermeidung in seinen üblichen Arbeitsabläufen gestört. Hier muss untersucht werden inwieweit und wann vor potentiellen Fehlern gewarnt werden soll. Nur auf diese Weise ist effizienter Einsatz einer Fehlerprävention gewährleistet.

## 3.9 Feedbackverarbeitung

Die bedingten Eintrittswahrscheinlichkeiten werden mithilfe der in der Fallbasis gespeicherten Fehlerfälle berechnet. Damit repräsentieren sie das Eintrittsverhalten der Symptome zu einem bestimmten Zeitpunkt. Dieses Eintrittsverhalten der Symptome ist jedoch nicht konstant und kann sich über die Dauer verändern. Eine kontinuierliche Anpassung der Eintrittswahrscheinlichkeit ermöglicht es dem Diagnosemodul, auf ein verändertes Eintrittsverhalten der Symptome zu reagieren.

Diese Anpassung erfolgt auf zwei Wegen. Einerseits werden die Eintrittswahrscheinlichkeiten alle $N$ Diagnoseaufrufe auf Basis der aktuellsten $N$ Fehlerfälle, aus der Fallbasis, berechnet. Andererseits bietet das Diagnosesystem eine Feedback-Schnittstelle mittels derer sich richtige Diagnoseergebnisse an das System zurück melden lassen. Als richtig werden Diagnoseergebnisse erachtet, wenn sie die tatsächliche Fehlerursache beschreiben. Die Verarbeitung eins Feedbackaufrufs hat sofortigen Einfluss auf das Fehlermodell und damit auf zukünftige Fehlerdiagnosen.

### 3.9.1 Feedbackalgorithmus

Als Eingabeparameter erhält der Feedbackalgorithmus ein Diagnoseergebnis, welches dem System als richtig gemeldet werden soll. Das Diagnoseergebnis beinhaltet einen Symptompfad

sowie die Fehlerursache. Bei der Verarbeitung der Feedbackanfrage werden die Eintrittswahrscheinlichkeiten entlang des gegebenen Symptompfades angepasst. Die Anpassung ist davon abhängig, ob die einzelnen Symptome im Symptompfad eingetreten sind oder nicht. Aus diesem Grund wird nur die Eintrittswahrscheinlichkeit angepasst, die das Eintrittsverhalten der verbundenen Symptome im Symptompfad beschreibt. Von der Festlegung auf eine bestimmte Höhe der Änderung bei einer Feedbackverarbeitung wurde abgesehen. Der Einfluss den eine Feedbackverarbeitung auf zukünftige Diagnosen haben soll, muss unter realen Bedingungen getestet werden.

Folgt im Symptompfad auf das Symptom S1, S2 und sind beide eingetreten, dann wird $P(S2|S1)$ erhöht. Folgende Tabelle listet die verschiedenen Wege auf, wie zwei Symptome in Abhängigkeit ihrer Staus miteinander verbunden sein können und welche Eintrittswahrscheinlichkeit wie angepasst werden muss.

Tabelle 3.6: Feedbackverarbeitung

| **Verbindungsart S1 → S2** | **bearbeitete Eintrittswahrscheinlichkeit** |
|---|---|
| eingetreten, eingetreten | erhöhe $P(S2\|S1)$ |
| eingetreten, nicht eingetreten | verringere $P(S2\|S1)$ |
| nicht eingetreten, eingetreten | erhöhe $P(S2\|\overline{S1})$ |
| nicht eingetreten, nicht eingetreten | verringere $P(S2\|\overline{S1})$ |

Ist S1 eingetreten und S2 nicht, muss die bedingte Eintrittswahrscheinlichkeiten $P(\overline{S2}|S1)$ erhöht werden. Eine Erhöhung von $P(\overline{S2}|S1)$ wird durch eine Senkung von $P(S2|S1)$ erreicht, da folgendes gilt:

$$P(\overline{S2}|S1) = 1 - P(S2|S1)$$

Durch die Anpassung der Eintrittswahrscheinlichkeiten entlang des Symptompfades wird die Wahrscheinlichkeit für das erneute Eintreten dieses Symptompfades im Vergleich zu den anderen erhöht. Eine Anpassung der übrigen Wahrscheinlichkeiten im Fehlermodell wird nicht vorgenommen.

### 3.9.2 Risiken und mögliche Probleme

Wie bereits in Kapitel 3.2.3 erwähnt, erfolgt die Verarbeitung der Feedbackinformationen ohne Prüfung auf Plausibilität. Das Diagnosesystem ist nicht in der Lage zu beurteilen, ob und wie gut die gemeldete Fehlerursache mit der tatsächlichen übereinstimmt. Darüber hinaus hat der Prüfer die Möglichkeit zu einem Diagnoseergebnis mehr als einmal die Feedbackfunktion aufzurufen.

Wird die Feedbackfunktion durch den Prüfer unsachgemäß verwendet beziehungsweise mit fehlerhaften Informationen bedient spiegelt sich dies im Fehlermodell wider. Ein falscher wiederholter Aufruf der Feedbackfunktion mit denselben Daten führt im Fehlermodell zu einer

erhöhten Eintrittswahrscheinlichkeit eines Symptompfades. Diese Eintrittswahrscheinlichkeit entspricht somit nicht mehr der Realen und dadurch ergeben sich zukünftig fehlerhafte Diagnoseergebnisse.

## 3.10 Systemablauf

Das Diagnosesystem bietet zwei Grundfunktionalitäten. Zum einen die Anfrage einer Diagnose anhand eines mitgelieferten Systemzustands bestehend aus einer Liste von Fehlersymptomen. Zum anderen die Rückmeldung von Diagnoseergebnissen um die Qualität der Diagnoseergebnisse zu steigern. Die Fehlerprävention wird immer zeitgleich mit einer Diagnose ausgeführt und ist nicht als eigenständige Funktionalität im System vorgesehen. Dies liegt daran, dass die Diagnose und die Fehlerprävention auf denselben Daten basieren. In Abbildung 3.16 ist der komplett Systemablauf durch ein Ablaufdiagramm dargestellt.

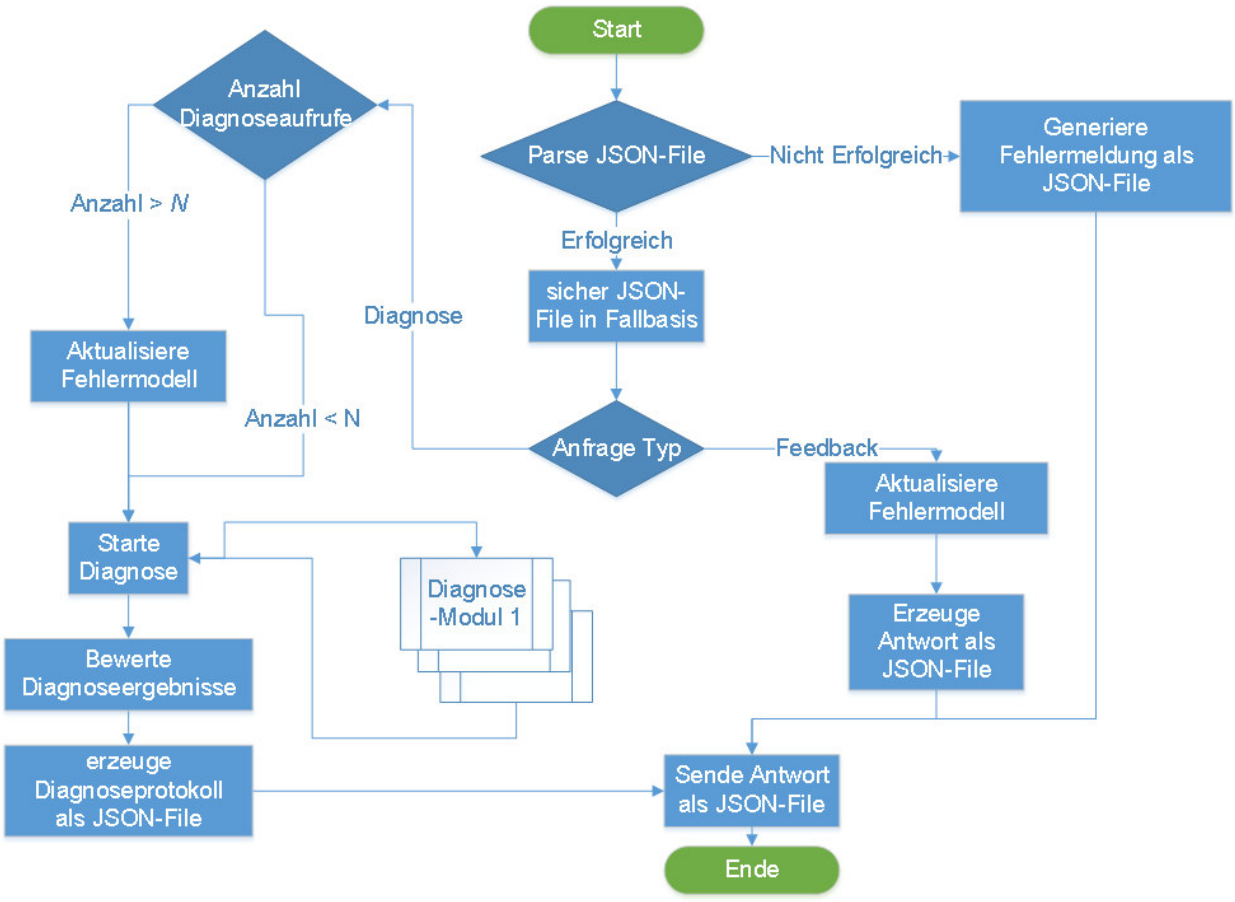



Abbildung 3.16: Systemablaufdiagramm

Die Informationen zu jeder Anfrage an das Diagnosesystem werden durch ein JSON-File übertragen. Daher besteht der erste Schritt in einer Syntaxüberprüfung des JSON-Files. Schlägt die Überprüfung fehl, können die Informationen nicht aus dem JSON-File gelesen werden. Eine Bearbeitung der Anfrage ist damit nicht möglich. Die Diagnosesoftware generiert daraufhin eine Fehlermeldung und sendet diese als Antwortet auf die Anfrage. Bei einer erfolgreichen Syntaxüberprüfung der Anfrage werden die Anfrageinformationen in der Fallbasis hinterlegt. Dies dient in erster Linie der Erweiterung der Fallbasis, da auf diese Weise die Menge der

bekannten Diagnosefälle kontinuierlich erweitert wird. Die Speicherung aller Anfrageinformationen ermöglicht auch die Erstellung einer Verlaufshistorie und lässt somit auch Rückschlüsse über die Verwendung der Diagnosesoftware zu.

Nach der Speicherung der Anfrageinformationen wird der Typ der Anfrage bestimmt. Wie bereits erwähnt bietet die Diagnosesoftware zwei Funktionen. Eine Fehlerdiagnose sowie eine Feedbackfunktion.

### 3.10.1 Fehlerdiagnose

Durch äußere Umwelteinflüsse kann sich das Fehlerverhalten des Prüfsystems ändern. Das bedeutet, dass beispielsweise eine Fehlerursache oder eine Klasse von Fehlerursachen, aufgrund äußerer Einflüsse auf das Prüfsystem, häufiger Eintreten kann. Eine kontinuierliche Anpassung der Systemparameter ermöglicht es, gewissen Diagnosemodulen auf ein verändertes Fehlerverhalten des Prüfsystems zu reagieren. Aus diesem Grund werden die Diagnosemodule alle *N* Diagnoseaufrufe auf Basis der aktuellsten Diagnoseinformationen aktualisiert.

Für den eigentlichen Start der Diagnosefunktion erhält jedes Diagnosemodul aus dem Diagnosesystem den übermittelten Systemstatus des Prüfsystems. Im Anschluss werden die verschiedenen Diagnoseergebnisse hinsichtlich ihrer Gesamtwahrscheinlichkeit und ihrer Zuverlässigkeit bewertet und sortiert. Die Diagnoseergebnisse werden zusammen mit den durch die Anfrage erhaltenen Informationen als Diagnoseprotokoll im JSON Format als Antwort versendet.

### 3.10.2 Feedback

Bei einer Feedbackverarbeitung wird die Anfrage durch den Diagnosemanager an jedes einzelne Diagnosemodul weitergeleitet. Über die Feedbackschnittstellen übergibt der Diagnosemanager die Feedbackinformationen an die Diagnosemodule. Die eigentliche Feedbackverarbeitung findet in jedem Diagnosemodul unabhängig statt. Nachdem die Feedbackverarbeitung von jedem Modul erfolgreich beendet wurde, antwortet der Diagnosemanager mit einer positiven Bestätigung auf die Anfrage.

# 4 Test und Evaluation

Ein Test des entwickelten Diagnoseverfahrens zeigt auf wie gut sich das Verfahren für den produktiven Einsatz eignet. Hierfür müssen aber in erster Linie geeignete Testszenarien sowie die notwendigen Testdaten bereitgestellt werden. In diesem Kapitel wird die Erzeugung der Testdaten, der Testablauf sowie die einzelnen Testszenarien mit dazugehörigem Testergebnis näher beschrieben.

## 4.1 Erzeugung von Testdaten

Die Testdaten bestehen aus einer Liste aller Symptome in der einige Symptome als eingetreten markiert sind, sowie einer List aus möglichen Fehlerursachen. Die Testdaten werden nur für das entwickelte Diagnosemodul erzeugt. Ein Test des Gesamtsystems ist nicht vorgesehen. Bei der Erzeugung der Testdaten muss auf Plausibilität geachtet werden. Die Testdaten müssen Fehlerszenarien entsprechen die so oder so ähnlich an Prüfständen eintreten können. Bei einem kleinen Fehlermodell ist eine Testdatengenerierung von Hand möglich. Jedoch wird für ein repräsentatives Testergebnis eine größere Menge an Testdaten benötigt. Folglich müssen die Testdaten automatisch generiert werden. Ein Test aller Funktionen des entwickelten Diagnosemoduls bedarf unterschiedlicher Testdatenformate. Für den Test der Fehlerdiagnose sowie der Fehlerprävention wird pro Testdurchlauf ein Datensatz benötigt, der fiktive Fehlerursachen sowie alle Symptome aus dem Fehlerbaum mit dazugehörigem Status beinhaltet. Ein Test der Feedbackverarbeitung benötigt hingegen lediglich eine Fehlerursache mit dazugehörigem Symptompfad ausgehend vom Wurzelsymptom bis hin zur Fehlerursache. Im Folgenden wird die Erzeugung beider Testdaten näher beschrieben.

### 4.1.1 Testdaten für die Fehlerdiagnose

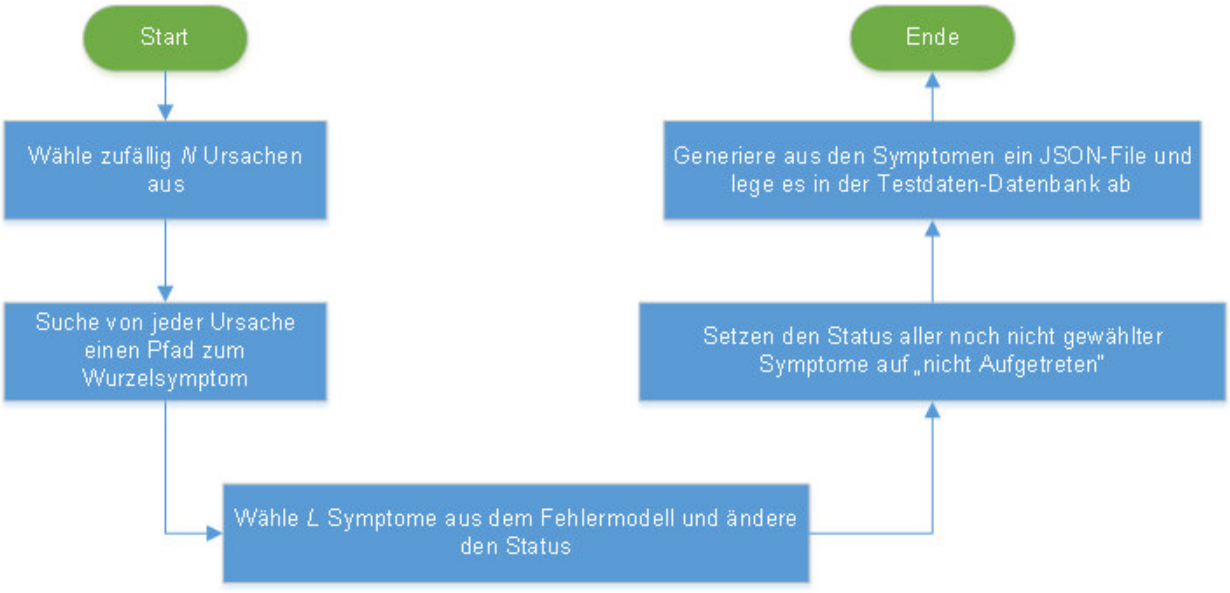



Abbildung 4.1: Erzeugung von Testdaten für die Fehlerdiagnose und Fehlerprävention

Die Punkte, die während der Erzeugung der Testdaten durchlaufen werden, sind in Abbildung 4.1 dargestellt. Der erste Schritt besteht in der Auswahl der fiktiven Ursachen, die den Fehler auslösen. Hierbei kann die Anzahl der Ursachen variieren. Eine Diagnose von mehr als einer Ursache erhöht den Schwierigkeitsgrad für das Diagnosemodul. Nach der Auswahl von *N* zufälligen Fehlerursachen wird für jede Fehlerursache ein Symptompfad generiert. Der Symptompfad beschreibt die Kausalkette der Symptome ausgehende von dem Wurzelsymptom bis hin zur eigentlichen Fehlerursache. Bei der Generierung des Symptompfades können zwei Strategien zum Einsatz kommen. Zum einen kann ausgehend vom Wurzel-Symptom immer nur das wahrscheinlichste Folgesymptom gewählt werden, welches zu der ausgewählten Fehlerursache führt. Zum andern besteht die Möglichkeit das Folgesymptom zufällig zu wählen. Eine zufällige Auswahl des Folgesymptoms erzeugt einen Symptompfad der schwerer zu diagnostizieren ist, da er eine geringere Wahrscheinlichkeit besitzt. Abbildung 4.2 veranschaulicht die beiden Strategien zur Symptompfadgenerierung.

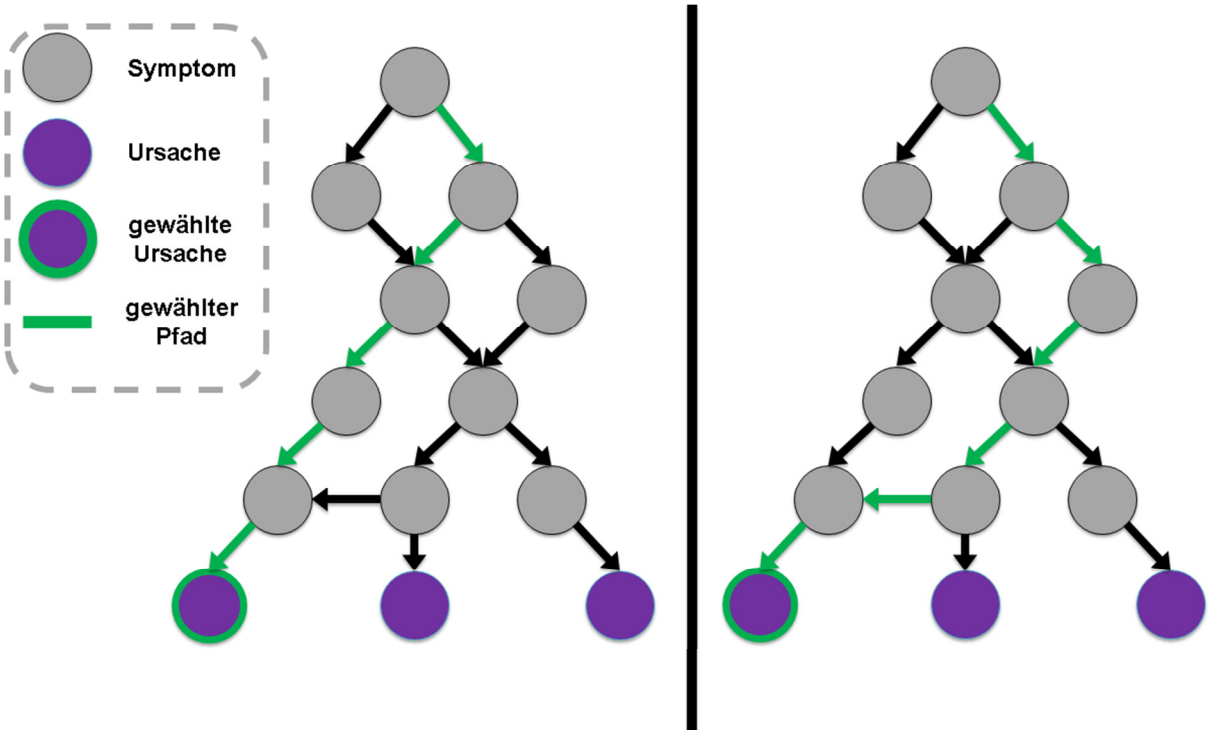



Abbildung 4.2: Symptompfad Generierung

Beim linken Pfad wurde immer das Folgesymptom gewählt, welches die höchste Eintrittswahrscheinlichkeit besitzt. Ausgehend vom Wurzel-Symptom bedeutet das, dass die Eintrittswahrscheinlichkeit des rechten Folgesymptoms, im Vergleich zu dem linken Folgesymptom, höher ist. Bei dem rechten Pfad hingegen wurde das Folgesymptom zufällig gewählt. Dadurch ist der Symptompfad länger. Hier muss jedoch angemerkt werden, dass die Länge des Symptompfades im Normalfall keine Rückschlüsse auf die Gesamtauftrittswahrscheinlichkeit des Symptompfades zulässt. Nach der Wahl des Symptompfades für jede Fehlerursache folgt die Bestimmung der Symptomzustände der einzelnen Symptome. Die Berechnung der Symptomzustände unter Realbedingungen findet zeitlich diskret statt. Daher kann nicht zu jedem Zeitpunkt garantiert werden, dass immer alle

Symptomzustände aktuell sind. Eine Simulation von fehlerhaften oder noch nicht aktualisierten Symptomzuständen findet über die Änderung des Symptomstatus statt. Das bedeutet, bei $L$ Symptome aus dem Fehlermodell wird der aktuelle Zustand invertiert. Abschließend wird der Symptompfad ohne gewählte Fehlerursache in einem JSON-File gespeichert. Die in Abbildung 3.2 gezeigten „metaInformation“ sind zum Test des Diagnosemoduls nicht notwendig und werden daher auch nicht erzeugt. Eine Überprüfung der Diagnoseergebnisse erfordert die zu Anfang gewählten Fehlerursachen. Sie werden unter dem JSON-Array „causes“ im JSON-File gespeichert.

### 4.1.2 Testdaten für Feedbackverarbeitung

Eine separate Generierung von Testdaten für die Feedbackverarbeitung ist nicht notwendig. Die erforderlichen Testdaten entstehen beim Test der Diagnosefunktion des Diagnosemoduls. Damit sind alle erforderlichen Daten für die Erzeugung eines JSON-Files für die Feedbackverarbeitung vorhanden. Wie in 4.1.1 wird auch zum Test der Feedbackverarbeitung die „metainformation“ nicht verwendet.

### 4.1.3 Testdaten für die Fehlerprävention

Ein Test der Fehlerprävention besitzt mit simulierten Daten keine Aussagekraft über die Qualität der Vorhersage. Die Reihenfolge der Testdaten ist zufällig gewählt und folgt somit keinen Regeln. Unter realen Bedingungen ist hingegen mit gewissen Gesetzmäßigkeiten bezüglich dem Eintreten von zukünftigen Fehlern zu rechnen. Aus diesem Grund kann im Zuge dieser Arbeit die Fehlerprävention nicht getestet werden.

## 4.2 Testablauf

Der Test des entwickelten Diagnosemoduls beschränkt sich auf die Systemkomponenten die dafür notwendig sind. Komponenten die keinen Einfluss auf das Diagnoseergebnis haben oder für die Diagnose nicht benötigt werden, finden keine Anwendung im Verlauf der Tests.

Der Testverlauf gliedert sich in zwei Abschnitte. Im ersten Abschnitt wird die Diagnosefunktion des Diagnosemoduls getestet. Im zweiten Abschnitt die Feedbackfunktion. Hierbei ist anzumerken, dass sämtliche Testdaten zuerst den ersten Abschnitt durchlaufen müssen bevor die dabei entstehenden Ergebnisse im zweiten Abschnitt getestet werden. Der Grund liegt in der Änderung der Wahrscheinlichkeiten des Fehlermodells durch den Test der Feedbackfunktion. Damit ist das Fehlermodell welches zur Generierung der Testdaten verwendet wurde nicht mehr identisch mit dem Fehlermodell nach dem Test der Feedbackverarbeitung.

### 4.2.1 Diagnosefunktion

Beim Test der Diagnosefunktion werden mithilfe der Testdaten aus 4.1.1 Diagnoseaufrufe erzeugt und im Anschluss die diagnostizierten Fehlerursachen mit den gewählten Fehlerursachen verglichen. Abbildung 4.3 zeigt den Ablauf beim Test der Diagnosefunktion.

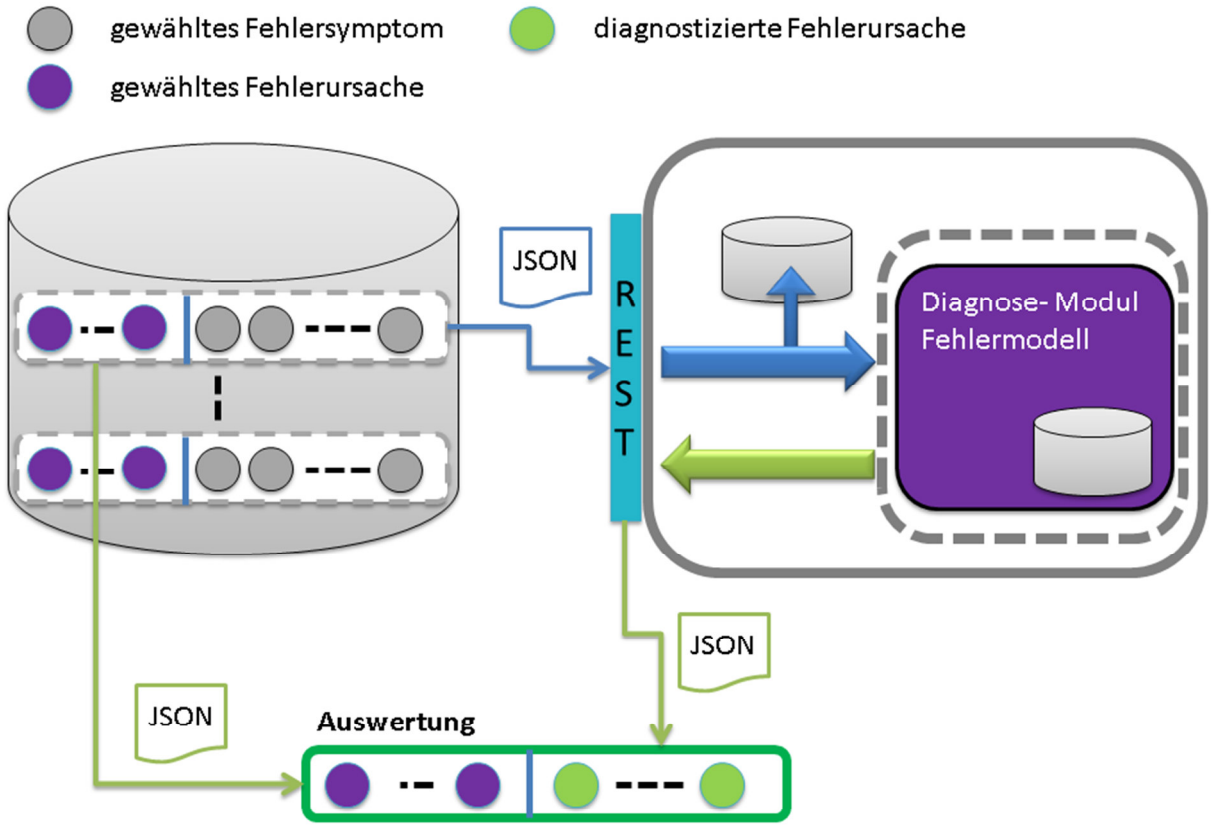



Abbildung 4.3: Testablauf Diagnosefunktion

### 4.2.2 Feedbackfunktion

Beim Testablauf der Feedbackfunktion wird eine Feedbackmeldung durch das Diagnosesystem verarbeitet und im Anschluss das veränderte Fehlermodell mit dem alten verglichen. Bei einer Erfolgreichen Verarbeitung besitzt der gemeldete Symptompfad eine höhere Eintrittswahrscheinlichkeit. Eine Überprüfung erfolgt durch die symptomweise Abarbeitung des gemeldeten Fehlerpfades und dem Vergleich der alten und neuen Wahrscheinlichkeiten.

## 4.3 Bewertungsverfahren

Das Ergebnis der Feedbackverarbeitung kann nicht bewertet werden, da erst erprobt werden muss inwieweit eine Feedbackmeldung das bestehende Fehlermodell verändern muss um den gewünschten Effekt auf zukünftige Diagnoseergebnisse zu haben.

Das Bewertungsverfahren für die Fehlerdiagnose setzt sich aus zwei Berechnungen zusammen. Die erste Berechnung wird für jede Antwort auf eine Diagnoseanfrage ausgeführt. Hierbei werden folgende Kennzahlen berechnet.

- Anzahl der enthalten Diagnoseantworten.
- Anzahl gefundenen Fehler aus dem Test-Diagnoseaufruf.
- Position des gefundenen Fehlers in der Diagnoseantwort

Mit der zweiten Berechnung werden anhand der Kennzahlen aus der vorrangegangenen Berechnung, Werte bestimmt die eine Aussage über mehrere Diagnoseantworten hinweg zulassen.

- Trefferquote. Verhältnis zwischen den Diagnoseaufrufen bei denen ein Fehler gefunden wurde, zu der Gesamtanzahl an Diagnoseaufrufen.
- Durchschnittliche Platzierung der gefundenen Fehler.
- Durchschnittliche Anzahl der gefundenen Fehler aus dem Test-Diagnoseaufruf

## 4.4 Testszenarien und Ergebnisse

Der Testablauf der Diagnosefunktion ist immer identische. Die einzelnen Szenarien unterscheidenden durch die verwendeten Testdaten. Um die Kompatibilität mit anderen Fehlermodellen zu testen, wurde der Algorithmus auf zwei verschiedenen Fehlermodellen getestet. Damit wird auch gleichzeitig die universelle Einsatzmöglichkeit des Diagnosesystems getestet. Das erste Fehlermodell ist in Abbildung 3.8 dargestellt. Das zweite Fehlermodell wurde durch einen, im Zuge dieser Arbeite entstandenen, Algorithmus automatisch generiert. Die internen Wahrscheinlichkeiten beider Fehlermodelle wurde auf Basis von jeweils 15000 Testdatensätzen berechnet. Die Eckdaten der beiden Fehlermodelle sind in den Tabellen aufgelistet.

Tabelle 4.1: Kriterien Fehlermodell 1

| **Kriterium** | **Wert** |
|---|---|
| Anzahl Wurzelsymptome | 1 |
| Anzahl Symptome | 11 |
| Anzahl Fehlerursachen | 8 |
| Anzahl Kanten | 25 |

Tabelle 4.2: Kriterien Fehlermodell 2

| **Kriterium** | **Wert** |
|---|---|
| Anzahl Wurzelsymptome | 1 |

| | |
|---|---|
| Anzahl Symptome | 24 |
| Anzahl Fehlerursachen | 26 |
| Anzahl Kanten | 59 |

Damit lässt sich ein Testszenario anhand folgender Eigenschaften beschreiben.

Tabelle 4.3: Eigenschaften Testszenario

| **Eigenschaftsname** | **Wertebereich** |
|---|---|
| Anzahl Testdaten | 1-1000 |
| Verwendetes Fehlermodell | Fehlermodell 1 \| Fehlermodell 1 |
| Anzahl der Fehler in einem Diagnoseaufruf | 1 - 2 |
| Verfahren zur Erzeugung von Testdaten | Maximale Wahrscheinlichkeit \| Zufällig |
| Anzahl der verfälschten Symptomstatus | 1 - 3 |

In der Testphase wurden 16 verschiedene Testszenarien durchgeführt. Die hierbei entstandenen Ergebnisse wurden dokumentiert und gespeichert. Im Folgenden werden die verschiedenen Testszenarien sowie die dazugehörigen Ergebnisse vorgestellt

**Szenario 1**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 0

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 83,6% | durchsch. Platzierung: 1,602 | durchsch. Fehler gefunden: 1 |

**Szenario 2**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 1

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 93,5% | durchsch. Platzierung: 14,502 | durchsch. Fehler gefunden: 1 |

**Szenario 3**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 2

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 88,3% | durchsch. Platzierung: 12,275 | durchsch. Fehler gefunden: 1 |

**Szenario 4**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 3

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 87,5% | durchsch. Platzierung: 13,36 | durchsch. Fehler gefunden: 1 |

**Szenario 5**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 1
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 0

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 100% | durchsch. Platzierung: 4,899 | durchsch. Fehler gefunden: 1 |

**Szenario 6**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 1
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 1

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 100% | durchsch. Platzierung: 6,425 | durchsch. Fehler gefunden: 1 |

**Szenario 7**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 1
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 2

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 100% | durchsch. Platzierung: 6,322 | durchsch. Fehler gefunden: 1 |

**Szenario 8**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 1
- Anzahl der Fehler: 1
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 3

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 100% | durchsch. Platzierung: 6,383 | durchsch. Fehler gefunden: 1 |

**Szenario 9**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: zufällig
- Anzahl verfälschte Symptome : 0

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 51% | durchsch. Platzierung: 2,988 | durchsch. Fehler gefunden: 1 |

**Szenario 10**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: zufällig
- Anzahl verfälschte Symptome : 1

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 87,2% | durchsch. Platzierung: 14,86 | durchsch. Fehler gefunden: 1 |

**Szenario 11**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: zufällig
- Anzahl verfälschte Symptome : 2

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 86,1% | durchsch. Platzierung: 15,44 | durchsch. Fehler gefunden: 1 |

**Szenario 12**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 1
- Testdatenerzeugung: zufällig
- Anzahl verfälschte Symptome : 3

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 87,9% | durchsch. Platzierung: 15,73 | durchsch. Fehler gefunden: 1 |

**Szenario 13**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 2
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 0

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 98,5% | durchsch. Platzierung: 7,19 | durchsch. Fehler gefunden: 1,62 |

**Szenario 14**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 2
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 1

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 99,6% | durchsch. Platzierung: 14,36 | durchsch. Fehler gefunden: 1,68 |

**Szenario 15**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 2
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 2

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 99,3% | durchsch. Platzierung: 13,8 | durchsch. Fehler gefunden: 1,674 |

**Szenario 16**

- Anzahl Testdaten : 1000
- Verwendetes Fehlermodell : Fehlermodell 2
- Anzahl der Fehler: 2
- Testdatenerzeugung: maximale Wahrscheinlichkeit
- Anzahl verfälschte Symptome : 3

Ergebnisse:

| | | |
|---|---|---|
| Trefferquote: 99,7% | durchsch. Platzierung: 13,96 | durchsch. Fehler gefunden: 1,64 |

## 4.5 Evaluierung

Mithilfe der Testszenarien 1 bis 8 lässt sich der Diagnosealgorithmus in Abhängigkeit des verwendeten Fehlermodells bewerten. In Szenario 1 bis 4 wurde das größere Fehlermodell 2 verwendet. Wohingegen in Szenario 5 bis 8 das Fehlermodell aus Abbildung 3.8 zum Einsatz kam. Werden Trefferquote und durchschnittliche Platzierung der Szenarien mithilfe eines Diagramms visualisiert entsteht folgende Abbildung.

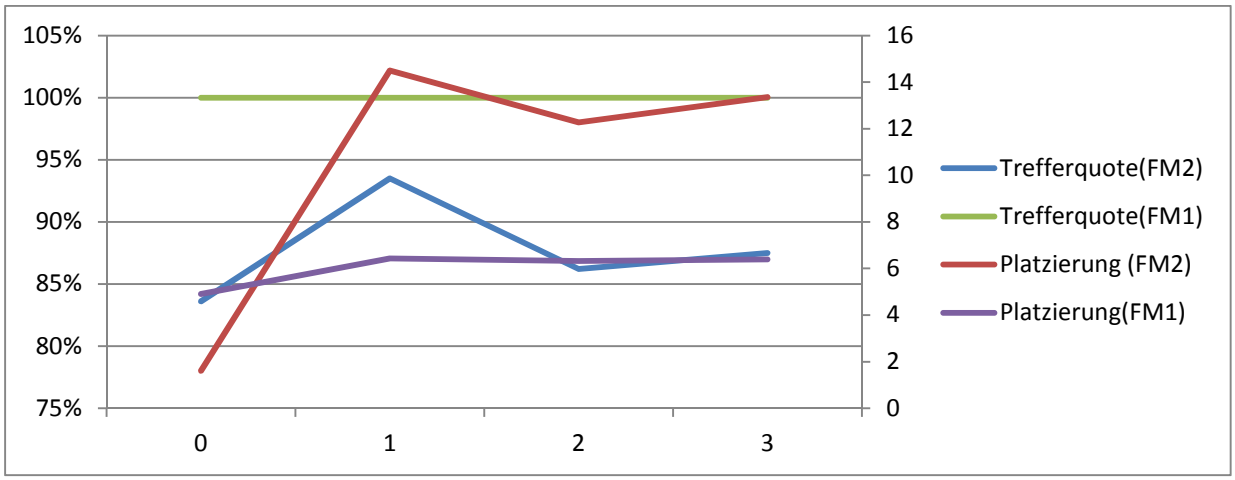


Abbildung 4.4: Ergebnisverlauf Szenarien 1 bis 8

Eine Diagnose unter Verwendung des größeren Fehlermodells liefert die beste durchschnittliche Platzierung der gefundenen Fehler, solange in den Testdaten keine Symptome verfälscht wurden. Bezüglich der Trefferquote ist Fehlermodell 1, Fehlermodell 2 überlegen. Dies kann jedoch auf die geringe Größe des Fehlermodell 1 zurückgeführt werden. Bei der Score-Wert Berechnung übersteigen nahezu alle Symptome im Fehlermodel die |minSicherheit| Grenze. Damit können nahezu alle Symptome auch teil eines Symptompfades werden. Somit sind annähernd alle Pfade im Fehlermodell abgedeckt und die Fehlerursache aus den Testdaten kann Teil der Diagnoseantwort werden.

Wird ein Symptom in den Testdaten verfälscht hat dies negative Auswirkungen auf die durchschnittliche Platzierung der gefundenen Fehler. Es besteht jedoch kein linearer Zusammenhang zwischen verfälschten Symptomen und der durchschnittlichen Platzierung. Eine signifikante Verschlechterung der Platzierung erfolgt nur bei dem Sprung von keinem verfälschten Symptom zu einem verfälschten Symptom. Eine Änderung von einem auf zwei verfälschte Symptome hat nahezu keine Auswirkungen auf die Platzierung. Dies Verhalten gilt sowohl für Fehlermodell 1 und 2 und ist auch im Verlauf der anderen Testszenarien aufgetretenen.

# 5 Umsetzung und Bedienung des Prototyps

Die Funktionalität des entwickelten Konzeptes wurde in einem Prototyp umgesetzt, um somit Tests und eine Evaluierung des System-Konzeptes zu ermöglichen. Im Folgenden wird auf die einzelnen Aspekte der Entwicklung eingegangen sowie die Verwendung des Prototyps erläutert.

## 5.1 Entwicklungsumgebung

Entwickelt wurde der Prototyp auf einem Rechner mit folgenden Eckdaten:

- i7 2670QM 4x2.2 GHz
- 8GB RAM
- Samsung 850 pro 250GB SSD
- Win 8.1 Pro

Der Prototyp wurde mithilfe der Entwicklungsumgebung Eclipse Luna Service Releas 2 (4.4.2) in Java geschrieben. Für den Prototyp wurden keine zusätzlichen Plugins installiert. Als Programmiersprache wurde Java gewählt. Kompiliert wurde unter Java 1.8.

## 5.2 Datenbank

Für die Verwaltung der anfallenden Daten wurde eine OrientDB Datenbank verwendet. Wie in 2.4.3 bereits erwähnt, ist OriendDB eine Graph- und Dokumentenorientierte Datenbank. Während der Entwicklung des Prototyps wurde OrientDB Version 2.0.7 verwendet.

### 5.2.1 Installation und Start

Die Installation der Datenbanksoftware wurde auf einem Rechner mit einem Windows-Betriebssystem durchgeführt. Als erster Schritt der Installation muss die zum Betriebssystem passende Installationsdatei von http://orientdb.com/download/ heruntergeladen und entpackt werden. Anschließend wird der Datenbankserver über *%orientDBVerzeichniss%/bin/server.bat* gestartet. Beim ersten Start des Datenbankservers muss ein Administrator-Konto angelegt werden. Sind alle Schritte erfolgreich ausgeführt, ist der Datenbankserver über http://127.0.0.1:2480/ zu erreichen. Über das Webinterface des Datenbankservers lassen sich neue Datenbanken anlegen sowie grundlegende Einstellungen des Datenbankservers anpassen.

### 5.2.2 Erstellung der Datenbank für den Prptotyp Betrieb

Das entwickelte Diagnosemodul benötigt für eine Diagnose ein Fehlermodell. Dieses Fehlermodell ist in einer Graph-Datenbank gespeichert. Die Erstellung dieser Graph-Datenbank erfolgt über das Ausführen zweier SQL-Batch Dateien mithilfe eines OrientDB Tools. Die Dateien entstand im Zuge dieser Arbeit. Das Tool console.bat ist im Ordner

*%orientDBVerzeichniss%/bin/*. Für die Ausführung der Batch-Datei wird das Tool console.bat mit dem Dateiname der Batch-Datei als Übergabeparameter aufgerufen.

```
%orientDBVerzeichniss%/bin/console.bat %PfadZuDatei%/fillDBWithData.sql
```

Die Batch Dateien müssen in folgender Reihenfolge ausgeführt werden.

1. *createDB.sql*
2. *fillDBWithData.sql*

In den Batch Dateien sind die Anmeldedaten für den Server enthalten. Diese müssen gegebenenfalls vor der Ausführung angepasst werden.

## 5.3 Klassendiagramm

Die Klassenstruktur des Prototyps wird in folgendem Abbild veranschaulicht. Aus Platzgründen wurde auf die Aufzählung aller Methoden und Attribute verzichtet. Des Weiteren ist der Webserver über den die REST-Schnittstellen bereitgestellt wird auch nicht Teil des Klassendiagramms

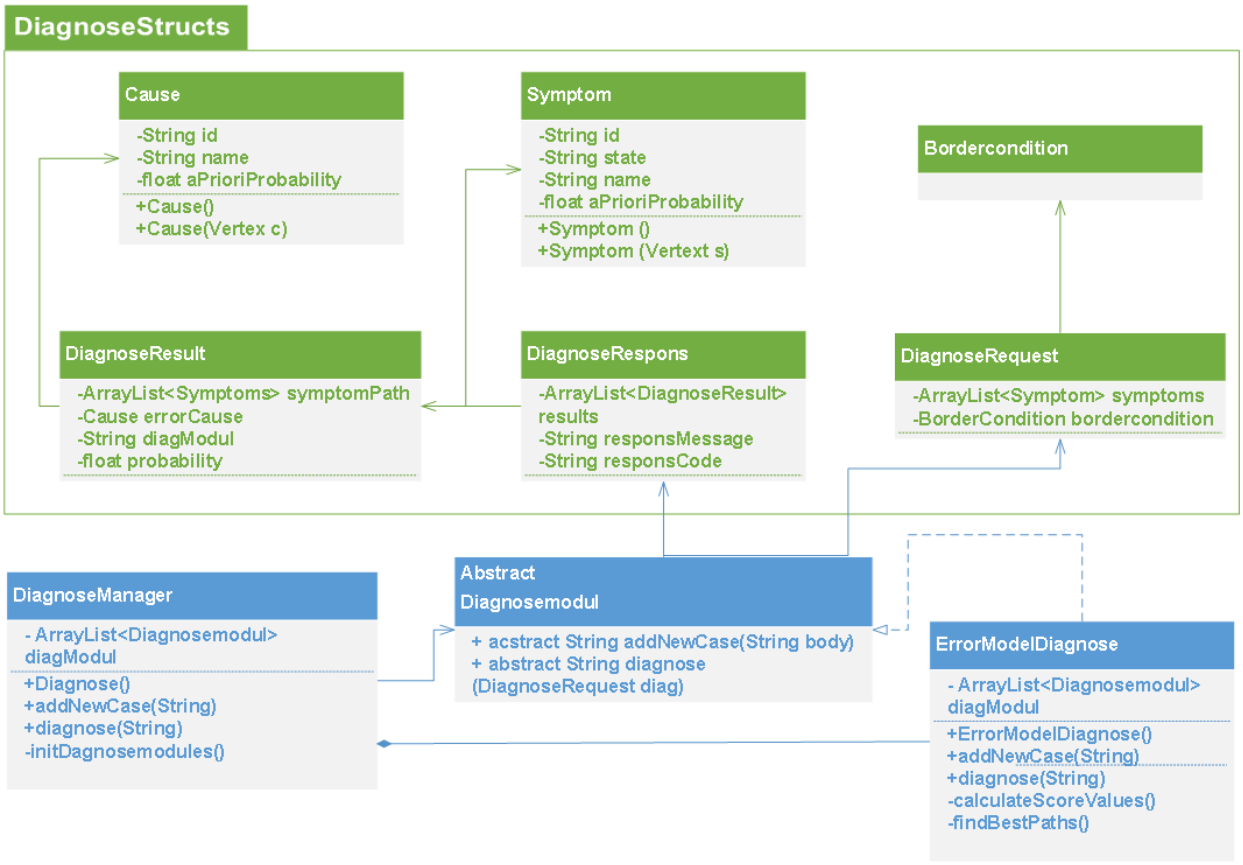



Abbildung 5.1: Klassendiagramm

Wie in 3.3 beschrieben, übernimmt der Diagnosemanager die Verwaltung des kompletten Systemablaufs. Bei der Instanziierung eins Diagnosemanagers werden auch alle Diagnosemodule mit Instanziiert. Erst hier entscheidet sich welche Diagnosemodule tatsächlich

Teil des Diagnosesystems sind. Die einzelnen Diagnosemodule implementieren die abstrakte Klasse Diagnosemodul. Dadurch sind einheitliche Schnittstellen gewährleistet.

Eine einheitliche Verwaltung der Diagnoseanfragen sowie Ergebnisse wird über die grün markierten Klassen erreicht. Sie beinhalten die Diagnoseinformationen wie beispielsweise Fehleruhrsachen und Symptompfade und werden für die Kommunikation zwischen dem Diagnosemanager und den einzelnen Diagnosemodulen eingesetzt.

## 5.4 JSON Verarbeitung

Das Diagnosesystem verwendet für die Kommunikation REST-Schnittstellen. Über die REST-Schnittstellen werden Informationen empfangen sowie versendet. Die Informationen sind ausschließlich im JSON-Format.

Für eine leichtere Verarbeitung werden die JSON konformen Informationen bei erhalt über die Schnittstelle in Java-Objekte umgewandelt. Für die Umwandlung zwischen JSON-File und Java-Objekt wurde die von Google entwickelte Bibliothek gson [GSON14] verwendet. Eine Umwandlung eines JSON-Files in ein Java-Objekt wird durch folgende Programmzeile erreicht:

```
DiagnoseRequest diagReq = gson.fromJson(body, DiagnoseRequest.class);
```

In der Variable *body* stehen die Informationen im JSON-Format. Die Klasse *DiagnoseRequest* spiegelt die Struktur der Informationen in der Variable *body* wieder (vgl. Abbildung 3.2 und Punkt 3.2). Strukturinformationen die nicht durch die Java-Klasse abgedeckt werden, werden bei der Umwandlung nicht beachtet und gehen verloren. Das bedeutet, enthält die Variable *body* beispielsweise den Eintrag *„probability“:50.0*, die Java-Klasse *DiagnoseRequest* aber kein Attribut *float probability,* geht die Information bei der Umwandlung verloren. Die Umwandlung eines Java-Objekts in ein JSON konformen String erfolgt durch Ausführung folgender Programmzeile:

```
gson.toJson(diagResponse)
```

Bei der Umwandlung von und zu Java-Objekten werden nur die Attribute der Klasse verwendet. Methoden bleiben dabei unbeachtet.

## 5.5 Testdatengenerierung

Ein Diagnoseaufruf erfordert ein JSON-File wie in Abbildung 3.2. Für die automatische Generierung solcher JSON-Files wurde die Java-Klasse *diagnoseCreateTestData.CreateTestdataForTesting.java* geschrieben. Sie erzeugt nach einem bestimmten Verfahren zufällig JSON-Files die für eine Diagnose verwendet werden können und und legt diese dann in der Datenbank *upsccTestData* ab. Über die Methode *createTest()* können die JSON-Files erzeugt werden. Bei dem Aufruf der Methode werden folgende Übergabeparameter erwartet:

**number**

Anzahl der zu erzeugenden Testfälle

**numOfCausesActivate**

Anzahl der Fehlerursachen die in einem Testfall vorkommen

**pathFindingStrategie**

Gibt an nach welcher Strategie die Testfälle erzeugt werden sollen. *1* erzeugt Testfälle anhan der der Maximalen Wahrscheinlichkeit. *2* erzeugt zufällige Testfälle. Das genaue Verfahren der Erstellung wir in 4.1 beschrieben

**numOfSwichStatus**

Angabe wie stark die Testfälle verfälscht werden sollen. Je höher der Wert desto mehr Symptomstatus werden verfälscht

## 5.6 Start und Bedienung des Prototypen

Bevor das Diagnosesystem gestartet werden kann muss sichergestellt sein, dass der Datenbankserver bereits läuft und die Graph-Datenbank installiert ist (siehe 5.2). Für den Start des Diagnosesystems aus Eclipse muss das Projekt *upscc_Prototyp2* geöffnet werden. Durch die Ausführung der Methode *main* in der Klasse *diagnose.main* wird das Diagnosesystem gestartet. Nach erfolgreichem Start sind die REST-Schnittstellen des Diagnosesystems unter der URL http://127.0.0.1:8080/upscc/ zu erreichen. URL Ist der Port 8080 bereist belegt muss ein neuer Port gewählt werden. Eine Portanpassung erfolgt in der Klasse *server.Server*. Die einzelnen Funktionen des Diagnosesystems werden wie folgt gestartet:

**Start Diagnose:**

URL: http://127.0.0.1:8080/upscc/diagnose , HTTP-Methode: POST, HTTP-Body: (siehe Abbildung 3.2)

**Start Feedback:**

URL: http://127.0.0.1:8080/upscc/addNewCase , HTTP-Methode: POST, HTTP-Body: (siehe Abbildung 3.5)

Während der Entwicklung wurden mithilfe des Chrome-Plugins Postman [POST15] mit dem Diagnosesystem kommuniziert. Bei der Verwendung des Tools Postman müssen folgenden Punkte beachtet werden.

Die URL der gewünschten REST-Schnittstelle muss korrekt angegeben werden. Abbildung 5.2 zeigt die Eingabe der URL für den Aufruf einer Diagnose

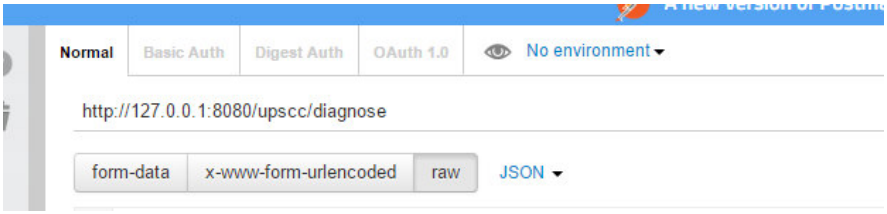



Abbildung 5.2: Postman URL Eingabe

Als HTTP-Methode muss POST ausgewählt sein, da sonst kein HTTP-Body mit der Anfrage mitgeschickt werden kann.

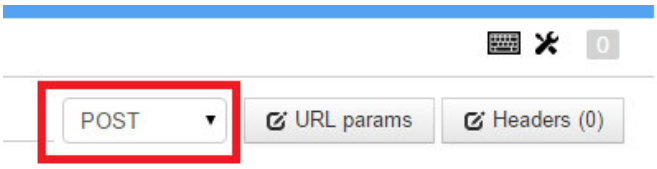



Abbildung 5.3: Postman http Methode

Mit der Anfrage muss immer auch das passende JSON-File im http-Body mitgeschickt werden.

Eine Kommunikation mit dem Diagnosesystem kann auch aus anderen Programmen erfolgen. Programme beziehungsweise Programmiersprachen müssen lediglich HTTP-Aufrufe unterstützen.

## 5.7 Erweiterung des Prototpyen

Eine Anforderung an das Konzept sowie auch an den Prototyp war ein leichte Erweiterbarkeit beziehungsweise Übertragbarkeit. Es werden verschiedene Möglichkeiten der Erweiterung und Übertragung auf andere Aufgabengebiete unterstützt.

### 5.7.1 Erstellung neuer Diagnosemodule

Die Erweiterung des Diagnosesystems um weitere Diagnosemodule erfolgt durch die Erstellung neuer Klassen die, die abstracte Klasse Diagnosemodul erweitern (siehe Abbildung 5.1). Die Methode *diagnose()* und *addNewCase()* fungieren als Schnittstelle zwischen DiagnoseManager und dem Diagnose Modul. Die Methode *diagnose()* wird vom Diagnosemanager bei einer Diagnoseanfrage aufgerufen und muss daher als Rückgabewert die Ergebnisse einer Diagnose liefern. Die Methode *addNewCase()* wird bei einer Feedbackanfrage aufgerufen. Wird eine Feedbackfunktion durch das neuentwickelte Diagnosemodul nicht unterstützt, kann diese Methode leer bleiben.

### 5.7.2 Erweiterung der Schnittstelle

Werden im Zuge neu entwickelter Diagnosemodule mehr Informationen für eine Diagnose oder eine Feedbackverarbeitung benötigt, kann die Schnittstelle einfach um diese Informationen erweitert werden. Aufgrund der Informationsübertragung zwischen Client und Server mittels JSON-File muss bei einer Erweiterung die REST-Schnittstelle nicht angepasst werden.

Lediglich für die Konvertierung von JSON-File zu Java-Objekt spielt eine veränderte Struktur des JSON-Files im Vergleich zu Kapitel 3.2 eine Rolle. Bei einer Erweiterung um Zusätzliche Informationen muss auch die dazugehörige Java-Klasse erweitert werden. Wird beispielsweise bei einer Diagnoseanfrage auch eine Liste von Fehlerursachen mitgeschickt die definitiv ausgeschlossen werden können, muss die Klasse *DiagnoseRequest* um das Attribut *ArrayList<Cause> impossibleCauses* erweitert werden. Nur auf diese Weise gehen die Informationen bei einer Konvertierung von JSON zu Java-Objekt nicht verloren. Die Erweiterung der Schnittstelle hat keinen Einfluss auf bestehende Diagnosemodule. Werden hingegen bestehende Attribute verändert oder gelöscht kess es zu einem Ausfall eines Diagnosemoduls beziehungsweise der gesamten Diagnose kommen.

### 5.7.3 Veränderung oder Austausch des Fehlermodells

Eine Anpassung oder ein Austausch des kompletten Fehlermodells ist unter Beachtung gewisser Punkte möglich.

**Anforderungen an das Fehlermodell:**

- Das Fehlermodell muss Zyklen frei sein
- Das Fehlermodell hat mindestens ein Wurzelsymptom
- Das Fehlermodell besitzt zwei Klassen von Knoten, Symptome und Ursachen

**Anforderung an Symptome und Ursachen**

- Symptome und Ursachen besitzen ein Attribut *id* über das sie eindeutig identifiziert werden können
- Symptome und Ursachen besitzen ein Attribut *aPrioriProbabilitiy* das ihre Wahrscheinlichkeit ihres Auftretens angibt

**Anforderung an die Verbindung**

- Der Name der Verbindung muss *causedBy* sein.
- Eine Verbindung von Knoten A zu Knoten B beinhaltet die Wahrscheinlichkeiten $P(B|A)$ und $P(A|\overline{B})$

# 6 Zusammenfassung

In dieser Arbeit wurde das Problem der fehlenden Zuordnungsmöglichkeit von eingetretenen Fehlern während einer Fahrzeugdiagnose behandelt. Ein Prüfer an einem Fahrzeugprüfstand kann bei einer fehlgeschlagenen Fahrzeugüberprüfung keine Aussage darüber treffen, ob die Prüfung aufgrund eines Fehlers am Fahrzeug oder im Prüfstand fehlschlug. Fahrzeuge, die irrtümlich als fehlerhaft klassifiziert werden, können hohen Folgekosten für das Unternehmen bedeuten. Werden Fahrzeuge häufig falsch klassifiziert, kann dies zu einem Stopp der gesamten Produktionsanlage führen. Darüber hinaus bedarf es speziell ausgebildete Wartungstechniker, die für die Instandhaltung der Prüfsysteme zuständig sind.

Durch das in dieser Arbeit entwickelte Diagnosesystem für Fahrzeugprüfstände erhält der Prüfer die Möglichkeit, Aussagen darüber zu treffen, ob und welche Fehler an einem Prüfstand vorliegen. Die Zahl der fälschlicherweise als fehlerhaft deklarierten Fahrzeuge wird somit verringert. Darüber hinaus ist ein Prüfer mithilfe des Diagnosesystems in der Lage, selbst gewisse Instandhaltungen an dem Prüfsystem durchzuführen beziehungsweise diese anzustoßen. Die Auslastung der Wartungstechniker wird damit verringert.

Neben einer Diagnose wurde ein Konzept für eine Fehlerprävention erstellt. Die Fehlerprävention weist den Prüfer auf mögliche Fehler hin, die in Zukunft an dem Prüfsystem auftreten können. Das Diagnosesystem bietet einem Prüfer noch die Möglichkeit die erhaltenen Diagnoseergebnisse zu bewerten. Die Rückmeldungen werden von dem Diagnosesystem dazu verwendet, um die Qualität der Diagnoseergebnisse kontinuierlich zu steigern.

Das entwickelte Diagnosesystem bestimmt anhand des Systemzustandes des Prüfsystems die wahrscheinlichsten Fehlerursachen. Während der Diagnose kommt ein Fehlermodell, in dem die Systemzusammenhänge des Prüfsystems modelliert sind sowie eine Wissensdatenbank zum Einsatz. Für die Fehlerprävention wurde anhand des Systemzustandes nach Anomalien in der Wissensdatenbank gesucht, um so kausale Auftrittszusammenhänge zwischen Fehlern in der Vergangenheit zu finden und dem Prüfer bereitzustellen.

Zu Beginn dieser Arbeit wurde das Problem der fehlenden Zuordnungsmöglichkeit von Fehlern im Detail veranschaulicht. Anschließend folgten die Motivation sowie die Zielsetzung der Arbeit. Im Grundlagenkapitel wurden das Thema Diagnose sowie die Komponenten und Technologien, die in dem Diagnosesystem Anwendung finden, vorgestellt.

Kapitel 3 beschreibt das Diagnosesystem in einem Top-Down Ansatz. Angefangen von der Gesamtstruktur des Systems über die Schnittstellen bis hin zu den einzelnen Komponenten wurden die Funktionalitäten im Detail erläutert.

Zum Zweck der Evaluierung des Diagnosesystems wurden im Verlauf der Arbeit Teile des entwickelten Konzepts in einem Prototyp umgesetzt. In Kapitel 4 wurde die Erstellung der Testdaten sowie die einzelnen Testszenarien und die Ergebnisse der Evaluation vorgestellt.

Abschließend erfolgt eine Beschreibung der Umsetzung des Prototypen, sowie eine Bedienungsanleitung und eine Verfahrensanleitung für die Erweiterung des Prototypen.

Tabelle 6.1 zeigt, inwieweit die funktionalen und nicht funktionalen Anforderungen, die zu Beginn definiert wurden, durch die Arbeit abgedeckt werden.

Tabelle 6.1: Anforderungsübersicht

| Name | Bedeutung | Erfüllt in % |
|---|---|---|
| /PAF10/ | Entwurf eines Systemkonzepts, welches die Möglichkeit bietet, verschiedene Diagnosemethoden in einem System zu vereinen. | 100% |
| /PAF20/ | Entwicklung eines Konzepts für eine Fehlerprävention anhand des aktuellen Systemzustandes und bereits vorhandenem Wissen. | 100% |
| /PAF30/ | Realisierung eines Prototyps zur Diagnose und Fehlerprävention anhand des übermittelten Systemzustands. | 50% |
| /PAF40/ | Einheitliche Verarbeitung des übermittelten Systemzustands. | 100% |
| /PAF50/ | Entwicklung von Fehlerszenarien für Prototypen Test. | 100% |
| /PAF60/ | Entwicklung einer geeigneten Methode für die Speicherung der Diagnoseanfragen an das Diagnosesystem. | 100% |
| /PANF10/ | Das Systemkonzept muss sich leicht erweitern bzw. auf andere Einsatzszenarien übertragen lassen. | 100% |
| /PANF20/ | Es soll eine Schnittstelle gewählt oder entwickelt werden, die es ermöglicht den Prototyp leicht in bestehende Systeme zu integrieren. | 100% |
| /PANF30/ | Die Diagnoseergebnisse sollen für Prüfer nachvollziehbar ausgegeben werden. | 100% |
| /PANF40/ | Das Vorgehen bei einer Systemübertragung bzw. Erweiterung des Prototyps soll in der Dokumentation verständlich erläutert werden. | 100% |

Bis auf die Realisierung der Fehlerpräventionsfunktion durch den Prototypen, wurden alle gestellten Anforderungen an die Arbeit erfüllt. Aufgrund der fehlenden Möglichkeit den Präventionsansatz zu überprüfen, wurde anstelle dessen der Prototyp hinsichtlich seiner Kompatibilität zu anderen Fehlermodellen untersucht. Hierfür wurde ein Algorithmus zur automatischen Generierung von Fehlermodellen entwickelt.

Zusammengefasst lasst sich sagen, dass das entwickelte Diagnosekonzept das Problem der fehlenden Zuordnungsmöglichkeit von aufgetretenen Fehlern im Verlauf einer Fahrzeugprüfung, im vollen Umfang löst. Der Prüfer erhält durch das Prüfsystem zum einen Informationen über den aufgetretenen Fehler, zum anderen wird durch die Verwendung eines Fehlermodells der Pfad der Kausalzusammenhänge zu diesem Fehler beschrieben. Eine Plausibilitätsprüfung der vorgeschlagenen Fehlerursache ist somit für den Prüfer möglich. Des Weiteren ist das Diagnosesystem durch die verwendetet Schnittstellen leicht in bereits bestehende Prozessabläufe zu integrieren und durch die Möglichkeit einer Erweiterung um

zusätzliche Module universell einsetzbar. Zudem ist das Diagnosesystem, durch die Verarbeitung von Rückmeldungen, in der Lage sich kontinuierlich an das verändernde Eintrittsverhalten von Fehler anzupassen. Jedoch muss auch erwähnt werden, dass bei einer Diagnose nicht sichergestellt werden kann, dass die Menge der errechneten Fehlerursachen auch die tatsächlichen Fehlerursachen enthält. Eine nachträgliche Überprüfung der errechneten Fehlerursachen ist aufgrund dessen, zwingend notwendig.

# 7 Ausblick

Diese Arbeit hat gezeigt, dass ein Diagnoseverfahren basierend auf verschiedenen Diagnosetechnologien möglich ist. Eine Hauptkomponente des Diagnoseverfahrens war ein Fehlermodell. Durch die Erweiterung des Fehlermodells um die bedingten Eintrittswahrscheinlichkeiten der einzelnen Symptome im Fehlermodell konnten die möglichen Fehlerursachen anhand ihrer Wahrscheinlichkeit sortiert und ausgegeben werden.

Durch die Rückmeldung bezüglich einer gefundenen Fehlerursache kann ein Prüfer diese Wahrscheinlichkeiten ändern. Aktuell verarbeitet das Diagnosesystem die Rückmeldungen ohne Validierung. Eine Überprüfung der Feedbackmeldung würde beispielsweise verhindern, dass eine Nachricht zu oft hintereinander gesendet wird. Dadurch wird auch verhindert, dass sich die Wahrscheinlichkeiten im Fehlermodell immer weiter von den tatsächlichen Eintrittswahrscheinlichkeiten entfernen. Ein falsches Fehlermodell, sowie falsche Wahrscheinlichkeiten im Fehlermodell führen zu fehlerhaften Diagnoseergebnissen.

Damit spielt das Fehlermodell eine zentrale Rolle für die Qualität der Diagnose. In dieser Arbeit wurde ein statisches Fehlermodell verwendet. Die Wahrscheinlichkeiten im Fehlermodell änderten sich zwar, jedoch nicht die Struktur. Die Kanten- und Knotenmeng im Modell blieb konstant. Unter Realbedingungen kann dies nicht garantiert werden. Das Hinzukommen neuer Knoten oder Kanten wird zwar von dem Diagnosesystem unterstützt jedoch nicht automatisch ausgeführt. Schlussfolgernd muss das Fehlermodell manuell angepasst werden. Eine automatisierte Strukturextraktion auf Basis der erhaltenen Prüfsystemzustände würde die Funktionalität des Diagnosesystems jedoch steigern.

Das Diagnosesystem ist in der Lage mehrere Diagnosealgorithmen gleichzeitig zu verwalten. Ein Vergleich mehrerer verschiedener Algorithmen kann damit realisiert werden. Ein Test des entwickelten Diagnosealgorithmus im Vergleich mit anderen würde weitere Rückschlüsse über die Qualität des Diagnosealgorithmus zulassen.

# Literaturverzeichnis


[WAIB14] Waible, Manuel: Konzeption und Realisierung eines Fehlerdiagnosesystems für ein KFZ-Prüfsystem mit fallbasiertem Schließen. IAS-Masterarbeit 2622, Stuttgart, 2014

[ECMA13] ECMA International Standard 404 http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (Abgerufen am 14.09.2015)

[NPRI09] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta: Comparison of JSON and XML Data Interchange Formats: A Case Study , 2009

[FIEL00] Roy Thomas Fielding: Architectural Styles and the Design of Network-based Software Architectures, 2000

[TILK11] Stefan Tilkov: REST und HTTP Einsatz und Architektur des Web für Integrationsszenarien, ISBN 978-3-89864-732-8, 2011

[BAYE02] Thomas Bayer: REST Web Services Eine Einführung, 2002 , http://www.oio.de/public/xml/rest-webservices.pdf (Abgerufen am 14.09.2015)

[DOGL15] Fernando Doglio: Pro REST API Development with Node.js, ISBN 978-1-4842-0917-2, 2015

[CBA11] P.A Castillo, J.L. Bernier, M.G. Arenas, J.J.Merelo, P. Garcia-Sánchez: SOAP vs REST: Comparing a master-slave GA implementation, 2011

[STEI94] René Steiner: Grundkurs Relationale Datenbanken, 1994 (8. Auflage 2014)

[CODD70] Edgar F. Codd: A Relational Model of Data for Large Shared Data Banks, 1970

[IORD10] Borislav Iordanov : HyperGraphDB A Generalized Graph Database, 2010

[DKR13] Volker Diekert, Manfred Kufleiter, Rosenberger Gerhard: Elemente der diskreten Mathematik, ISBN 978-3-11-027816-3, 2013

[VMZN10] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, Dawn Wilkins: A Comparison of a Graph Database and a Relational Database, 2010

[APA15] Apache Lizenz Version 2. : http://www.apache.org/licenses/LICENSE-2.0 (Abgerufen am 14.09.2015)

[NEO15] Neo4j Graph-Datenbank: http://neo4j.com/ (Abgerufen am 14.09.2015)

[DBE15] Datenbankvergleich: http://db-engines.com/de/ranking (Abgerufen am 14.09.2015)

[BROW12] MC Brown: Getting Started with CouchDB, ISBN 978-1-449-30755-4, 2012

[REIF07] Konrad Reif: Automobilelektronik: Eine Einführung für Ingenieure, ISBN 978-3834802972, 2007

[LEAY97] S. Leonhardt, M. Ayoubi: Methods of fault diagnosi, Control Engineering Practice Volume 5, Issue 5, May 1997, Pages 683–692

[HANS03] Hans, Norbert : Strategische Wettbewerbsvorteile, ISBN 978-3-658-02374-4, 2003

[ISER12] Rolf Isermann : Von der örtlichen Überwachung bis zur Telediagnose, erschienen in chemie&more 3/2012, http://www.chemieundmore.com/archive/211934/Von-der-oertlichen-Ueberwachung-bis-zur-Telediagnose.html (Abgerufen am 14.09.2015)

[JENS01] Finn Verner Jensen, Bayesian Networks and Decision Graps, ISBN 0-387-95259-4, 2001

[KBK+11] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Georg Ruß, Matthias Steinbrecher : Computational Intelligence, ISBN 978-3-8348-1275-9, 2011

[PEPP05] Peter Pepper, Programmieren mit Java, ISBN 3-540-20957-3, 2005

[GSON14] Bibliothek für die Umwandlung zwischen JSON und Java-Objekt, https://sites.google.com/site/gson/gson-user-guide (abgerufen am. 14.09.2015)

[POST15] Plugin für das Versenden von HTTP-Anfragen, https://github.com/postmanlabs/postman-chrome-extension-legacy (abgerufen am 14.09.2015)

[STEU15] Erklährung Steuerzeichen: http://www.ling.uni-potsdam.de/~kolb/c-kurs/steuerz.html (abgerufen am. 14.09.2015)

[BAYE15] Satz von Bayes : http://matheguru.com/stochastik/36-satz-von-bayes.html (abgerufen am 14.09.2015)