# Eine Testfallspezifikationssprache für das funktionsorientierte Testen von reaktiven eingebetteten Systemen im Automobilen Bereich

Dissertation im Fach Informatik

Vom Fachbereich Informatik der Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation

von

**Dipl.-Inform. Mesut Ipek**

Datum der wissenschaftlichen Aussprache: 02. Mai 2011

| | |
|---|---|
| **Dekan:** | Prof. Dr. Arnd Poetzsch-Heffter |
| **Vorsitzender des Prüfungsausschusses:** | Prof. Dr. Müller |
| **1. Berichterstatter:** | Prof. Dr. Peter Liggesmeyer |
| **2. Berichterstatter:** | Prof. Dr. Müller-Glaser |

D 386

# Kurzfassung

Beim funktionsorientierten Testen von Steuergeräten im automobilen Bereich ist das Expertenwissen aufgrund der hohen Komplexität der Testfälle unersetzlich.

Bei Basistesttechniken wie der Grenzwertanalyse ist die Absicht eines Testfalls implizit durch die Technik gegeben. Beim Expertenwissen wird jedoch zur Zeit zu jedem erstellten Testfall zusätzlich ein Prosatext verfasst um die Testabsicht anzugeben. Diese Prosabeschreibung ist anfällig für Mehrdeutigkeiten, fällt bei jedem Testentwickler unterschiedlich aus und der inhaltliche Bezug zum Testfall ist lose.

Ziel der Arbeit ist eine Spezifikationssprache für die Testfallbeschreibung zu entwerfen um die Nachteile der natürlichen Sprache zu minimieren und testablaufspezifische Sprachelemente zu definieren, so dass sie als ein Grundgerüst für einen Testfall verwendet werden kann.

Dazu wird aus der Einsatzumgebung (Systemspezifikation, Testimplementierung und Testprozessthemen) Sprachelemente für die Beschreibung abgeleitet und Ansätze für die Überführung der Beschreibung in die Testimplementierung betrachtet.

Das Ergebnis ist eine Testfall-Spezifikationssprache, die auf formaler Grundlage basiert und u.a. in eine graphische Sicht überführt werden kann. Ähnlich der UML wird der Mehrwert erst durch eine werkzeugunterstützte Eingabe deutlich: So sind die Testentwickler in der Lage, einheitliche, formale, wieder verwendbare, verständliche Testfälle zu definieren.

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

## 1.1 Testen der Elektrik/Elektronik im Automobil

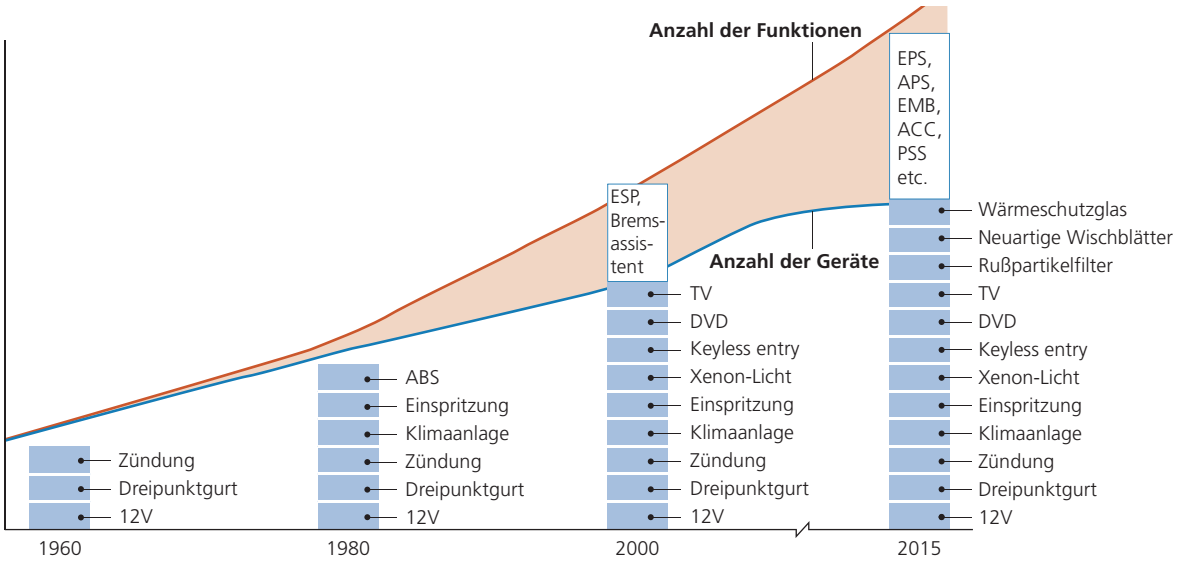
Die Elektrik/Elektronik (E/E) ist gegenwärtig ein fester Bestandteil im Automobil [Vol09]. Sie erfüllt im Fahrzeug mittlerweile Funktionen, die mit herkömmlichen mechatronischen Systemen nicht möglich wären (vgl. Entwicklung der Innovationen in Abbildung 1.1).

Wo früher einzelne eingebettete Systeme für Funktionen lokalen Verhaltens verantwortlich waren wie z.B. das Türsteuergerät mit der Fensterheber Funktion, werden durch Vernetzung vieler dieser Systeme im Fahrzeug komplexe Funktionalitäten gemeinsam erfüllt. Der Trend geht so von einzelnen Innovationen hin zu systemweiten Innovationen [Wym07].

Dieser Trend wird u.a. durch die Gesetzgebung, die Wirtschaft und durch die Wünsche des Kunden geprägt.

Mittlerweile können die von der Gesetzgebung erlassenen Schutzmaßnahmen für die Minimierung von Unfällen (z.B. wie die Pflicht für den Bremsassistenten [Kut09]) ohne die E/E nicht verwirklicht werden. Die Forderung der Gesetzgebung nach dem serienmäßigen Einbau des elektronischen Stabilitätsprogramms (ESP) in Pkws ab 2014 [Dan09] ist ein weiterer Beleg dafür. Weitere Systeme werden in Zukunft durch Gesetzgebungen gefordert [KEGZ08].

Der Einfluss der Wirtschaft auf die Entwicklung eines Fahrzeugs verhält sich ebenso: Reduktion der $CO_2$ Emmision, ein „Systemwechsel“ von klassischen Antrieben zu Elektroautos [MST10] und der Druck durch die kürzliche Krise; letztendlich werden




ESP = Electronic Stability Program, EPS = Electronic Power Steering, APS = Adaptive Power Steering, EMB = Electro-mechanical Braking, ACC = Adaptive Cruise Control, PSS = Predictive Safety Systems

**Abbildung 1.1:** Wandel der Innovationen im Automobil [Wym07]

die auftretenden Anforderungen an das Automobil weitestgehend durch die Elektrik/Elektronik des Fahrzeugs aufgegriffen [Gra10b].

Der Trend, Fahrzeuge zu bauen, die sich an den Kunden orientieren, bringt durch die stark kundenerlebbaren Funktionalitäten wie die aktive und passive Sicherheit, aber auch Komfortfunktionen wie die Multimedia Konnektivität (vgl. [Gra10a], [Gra09]) zahlreiche neue Herausforderungen in der Einbettung bzw. der reibungslosen Vernetzung zum Gesamtsystem.

Der zunehmend steigende Umfang der Funktionalitäten erschweren somit die Integration im Fahrzeug [Kno08]. So ist es nicht verwunderlich, dass die Elektronik im Automobil mit 40% bei der Pannenstatistik den ersten Platz belegt [ADA09].

Hinzu kommt, dass durch den Druck zur Kostensenkung die Produktionszyklen kürzer gestaltet werden müssen ohne dabei die Qualität der Produkte zu verringern [RS09]. Wenn man bedenkt, dass in einem Fahrzeug bis zu 80 eingebettete Systeme vorhanden sind [Hut06], wird es deutlich, dass die Integration eine Herausforderung ist, die sich in allen Teilbereichen der Entwicklung niederschlägt.

## Verflochtene Entwicklungsprozesse

Dabei ist die Entwicklung eines Automobils kein einfacher Prozess. Viele Gruppen sind beteiligt (vgl. Abschnitt 2.3.2 über die beteiligten Partner bei der Entwicklung, S. 23). Abbildung 1.2 zeigt die Verschachtelung der Vorgehensmodelle bei der Entwicklung[1]. Der Automobilhersteller entwickelt zwar das Auto (großes V in der Abbildung; vgl. Abschnitt 2.2 über Vorgehensmodelle in der Entwicklung, S. 18), die Entwicklung der einzelnen Komponenten oder Teile des Gesamtsystems werden aber durch Zulieferer übernommen [Wil06], so dass eine Vielzahl an Entwicklungsprozessen ineinander greifen. Des Weiteren basieren die Entwicklungsprozesse auf einer prototypischen Entwicklung, d.h. sie besteht aus mehreren einzelnen Etappen, in denen anfänglich Entwicklungsmuster bereitgestellt werden, bis nach mehreren Iterationen eine Version entsteht, womit die Serienfertigung gestartet werden kann (in der linken Abbildung mehrere kleine Vs nebeneinander). Und das aufgebrochen auf jedes einzelne eingebettete System[2]. Hier ist eine enge Zusammenarbeit der beteiligten Partner und Prozesse zwingend notwendig (vgl. [Gra10b]). Diese Zusammenarbeit besteht u.a. in der Synchronisation der Prozesse, der Kommunikation und Austausch von Informationen um ein reibungsloses Ganzes zu entwickeln.

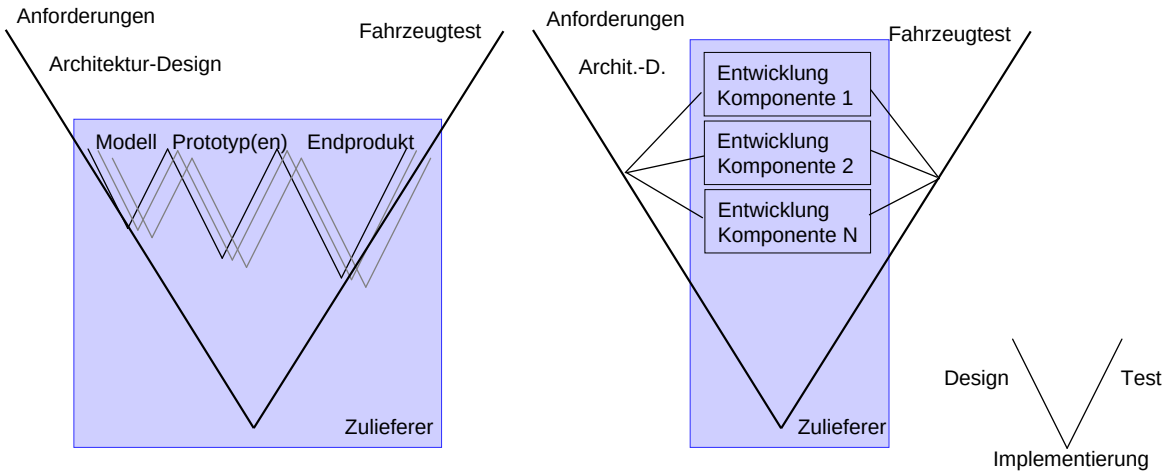



**Abbildung 1.2:** Verschachtelte V-Modelle bei der Entwicklung (vgl. [BN03])

[1] Das V-Modell (vgl. Abschnitt 2.2, S. 18) wird Aufgrund der grundlegenden Schritte „Design, Implementierung und Testen“ sowie dem Bezug vom Testen zum Design oftmals als visuelle und inhaltliche Darstellung der Fahrzeugentwicklung verwendet.

[2] Bei dieser Betrachtung werden mechanische Teile des Automobils nicht berücksichtigt. Mehr zum Systembegriff in Abschnitt 2.1, S. 13

### Entwicklungsbegleitendes Testen

Ein Teilbereich der Entwicklung ist die Qualitätssicherung. Das Testen von den eingebetteten Systemen ist wiederum ein Teil dieser Qualitätssicherung und der letzte Teilbereich im V-Modell. Wie aus der Abbildung 1.2 deutlich wird, ist das Testen der eingebetteten Systeme entwicklungsbegleitend, d.h. in jeder Iteration werden Tests durchgeführt und die Ergebnisse fließen in den Entwicklungsprozess mit ein.

Dabei ist das Ziel, so früh wie möglich Fehler zu finden und sie in den Anfangsphasen der Entwicklung zu beseitigen um die Nachfolgekosten eines Fehlers zu reduzieren (vgl. [Boe06]). Hierfür gibt es unterschiedliche Ansätze des Testens um Systeme prüfen zu können, wie im Abschnitt 2.6, S. 35 erläutert wird. So werden beispielsweise (noch) nicht vorhandene Komponenten oder Systeme simuliert um die Module, Codeteile oder das ganze eingebettete System (Hardware und Software) testen zu können. Das eingebettete System wird zum Testen sozusagen in eine Schleife virtuell vorhandener Systeme geschaltet.

### Qualität der Testfälle

Das Testen selbst ist wieder ein Prozess unterstellt, der Testprozess (siehe Abschnitt 2.5, S. 29). Der Zweck ist, die Testaktivitäten zu optimieren. Eine Phase im Testprozess ist die Testfallspezifikation, d.h. die Erstellung der Testfälle aus den Anforderungen an das zu testende System (dem Testobjekt).

Da die Arbeit diese Thematik betrachtet, soll eine genauer Betrachtung in diesem Bereich die Thematik veranschaulichen.

## 1.2 Erstellung von Testfällen

### Aus der Sicht des Unternehmens

In den frühen Entwicklungsphasen stehen zum Teil Informationen nicht vollständig zur Verfügung. Anforderungen können sich ändern oder als veraltet erweisen. In den späteren Phasen der Entwicklung erfolgen die Testaktivitäten zur Vermeidung von Regressionen (beseitigte Fehler treten wieder auf) und selten auftretender Fehler. Hier gilt: Das Wissen und die Erfahrung eines Testfallerstellers (Test-Designer) ist unverzichtbar für eine hohe Qualität der Testfälle, genau wie die Kommunikation zwischen Entwickler und Tester (siehe Abbildung 1.3).

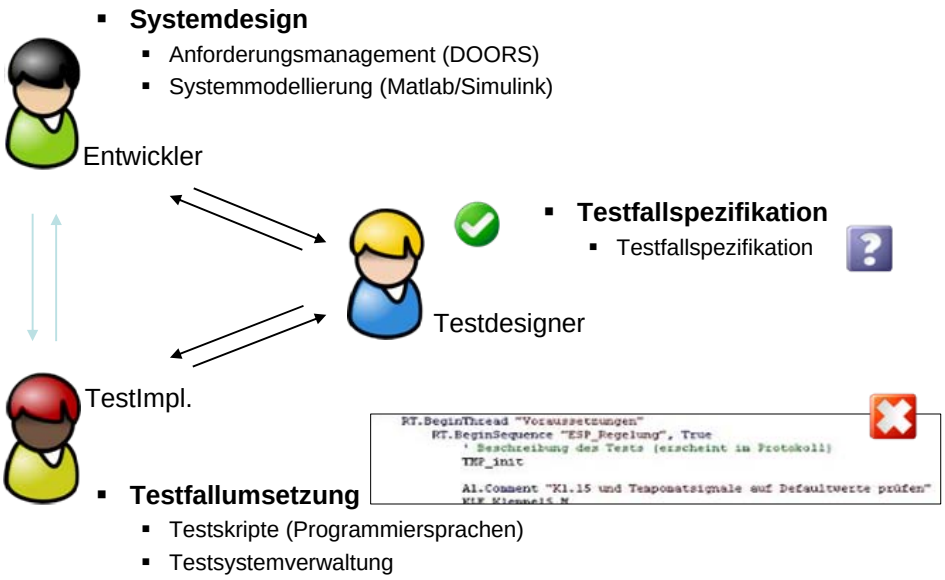



**Abbildung 1.3:** Aufgaben der Rollen

Eine direkte Erstellung der Testfallimplementierungen aus den Anforderungen ist unvorteilhaft, denn die Implementierungen bestehen aus Programmiercodes. Der Programmiercode enthält einen starken Bezug zu den Systemen, auf denen die Testfälle durchgeführt werden (Testsysteme). Die ursprüngliche Idee des Testfalls, d.h. welche Anforderungen in welcher Form getestet werden, ist nicht mehr ohne Weiteres zu erkennen.

Der Testimplementierer und die Testfallimplementierung ist somit keine Basis für eine Abstimmung über die Testaktivitäten.

Gegen diese direkte Erstellung der Testfallimplementierungen ist die Rolle des Test-Designers gedacht. Er soll die Testfälle spezifizieren. Zum einen ist mit der Spezifikation des Testfalls eine Basis vorhanden, um die Testaktivitäten mit den Entwicklern abzustimmen. Des Weiteren ist die Testfallspezifikation abstrakt gehalten, um ähnliche Anforderungen gleichwertig testen zu können. Zum Anderen kann anschliessend der Testimplementierer aus der Spezifikation des Testfalls (Testfallspezifikation) die Implementierung vornehmen. Der Schwerpunkt der Beschreibung ist somit eine andere als die der Implementierung (vgl. Kapitel 2, S. 13).

Wegen der oben genannten Anforderungen braucht der Test-Designer eine Sprache für sich. Genau so wie die Entwickler ihre Spezifikations- und Modellierungswerkzeuge und -sprachen besitzen und Testimplementierer ihre Programmiersprachen für die Umsetzung der Testfälle.

#### Beschreibung der Testfälle

Die natürliche Sprache ist zur Beschreibung nahe liegend. Doch ist sie mehrdeutig bei der Interpretation. Der Einsatz vorhandener Werkzeuge wie einer tabellen-basierten Beschreibung, endet in der Unübersichtlichkeit bei komplexen Testfällen. Eine Wiederverwendung für gleichartig zu testende Anforderungen wird erschwert und die Beschreibung erfolgt auf unterschiedlichen Detaillierungsgraden.

Ein Pendant zur System-Spezifikation im Bereich Testen ist notwendig, das die Absicht von Testfällen beschreibt und nicht die Umsetzung (Testfallimplementierung), d.h. eine implementierungsunabhängige, abstrakte Beschreibung von Testfällen ermöglicht. Trotzdem soll die Beschreibung konkret genug sein, um funktionalen Testfällen im Automobilen Bereich beschreiben zu können [Sax08].

### Aus der wissenschaftlichen Sicht

Die Kernfrage, womit der Testentwickler die Testfälle beschreibt soll in dieser Arbeit betrachtet werden.

Bei den Testverfahren wie der Äquivalenzklassentests oder Grenzwertanalyse wird die Absicht der Testfälle durch das Verfahren deutlich, d.h. die erstellten Testfälle (Testdaten) sind aus der Idee entstanden, auf denen die Testverfahren basieren[3]. Das Expertenwissen (vgl. [Lig02]) umfasst den Ablauf, dass der Testentwickler sich ein Bild aus der Spezifikation erstellt und aus dieser inneren Vorstellung die Testfälle ableitet (s. Abschnitt 3.4.1, S. 70). So ist die Absicht, welches mit dem Testfall verfolgt wird, nicht durch das Testverfahren erkenntlich.

Hier muss zusätzlich zu den Testdaten (Stimuli und Sollergebnisse), die Absicht in die Testfallbeschreibung eingebracht werden. Vor allem dann, wenn durch die Rollenaufteilung (vgl. Abschnitt 2.5.3, S. 32) unterschiedliche Personen für die Erstellung eines Testfalls, ihrer Implementierung und der Durchführung verantwortlich sind.

## 1.3 Ziel der Arbeit

Ziel dieser Arbeit ist es eine Sprache, ein Tool oder - noch allgemeiner formuliert - ein Mittel für den Test-Designer zur Verfügung zu stellen, mit dem er Testfälle für funktionale Steuergerätetests beschreiben kann. Das Mittel ist eine Beschreibungsmöglich-

[3] Anhang C, S. 267 zeigt die Testidee bzw. Testabsicht einiger klassischen Testverfahren auf.

keit für den Test-Designer; ähnlich der Programmiersprachen für die Testfallimplementierungen oder die Spezifikations- bzw. Modellierungssprachen für die Entwicklung eines Systems.

Das Mittel ...

- ... ist für den Testdesigner, der kein Programmierer ist.
- ... ist für Testfälle, die für das funktionale Testen von Steuergeräten entwickelt werden.
- ... ist für Testfälle, die eindeutig und wiederverwendbar sind.
- ... **vermittelt die Absicht eines Testfalls**.

Die klassische Definition eines Testfalls (vgl. Abschnitt 2.5, S. 29) mit der Definition der Eingabe- und erwartete Resultate, sowie die Vor- und Nachbedingungen, reicht nicht aus, um eine Sprache für die Beschreibung definieren zu können[4].

So entstehen Teilfragen bei der Betrachtung des Ziels:

- Was sind Testfälle für funktionale Steuergerätetests? Welche Eigenschaften haben sie?
- Welche Anforderungen werden an eine Beschreibungssprache für Testfälle gestellt?
- Welche vorhandene Mittel ausser der natürlichen Sprache und tabellen-basierte Beschreibung gibt es? Welche Anforderungen können abgedeckt werden und welche nicht?
- Welche Bestandteile hat eine Testfallspezifikation, damit der Test-Designer die Testfälle spezifizieren kann?[5]
- Welche Elemente haben Einfluss auf die Idee eines Testfalls? Wie kann dieser in der Notation festgehalten werden?

## 1.4 Gliederung der Arbeit

Das Thema wird wie folgt behandelt (vgl. Abbildung 1.4):

[4] siehe Kapitel 2, bei dem die Definition eines Testfalls in Bezug zu den zu testenden Systemen gesetzt wird

[5] Hiermit sind Bestandteile einer Testfallspezifikation gemeint, die nicht mit der ersten Frage beantwortet werden kann.

- Umfeld darstellen in Kapitel 2

  Im ersten Schritt wird das Umfeld rund um den Testprozess und den zu testenden Systemen aufgezeigt. Hier werden die Entwicklungsprozesse bei der Fahrzeugentwicklung, der zugehörige Testprozess, die Testumgebung und das Testobjekt dargestellt. Beim zu testenden System werden Eigenschaften betrachtet, die das System ausmachen. Dieses Kapitel ist Grundlage zur Ableitung der Anforderungen an eine Testfallbeschreibung.

- Inhalt und Umfang an die Sprache bestimmen in Kapitel 3

  Es wird der Inhalt und Umfang von Testfällen für funktionale Steuergerätetests bestimmt. Dabei werden die Besonderheiten des zu testenden Systems, die Dokumente der System-Spezifikation und der Testfallumsetzung analysiert und davon die Auswirkungen auf Testfälle dargestellt. Das Ergebnis ist eine Menge von Sprachelementen für die Beschreibung der Testfälle. Weiterhin werden die Eigenschaften der Testfälle herausgestellt.

- Anforderungen an die Testfallbeschreibung definieren in Kapitel 4

  Die Elemente zur Beschreibung von Testfällen sind eine Anforderung an die Testfallbeschreibung. In diesem Kapitel werden zusätzliche Anforderungen abgeleitet, die sich aus Anwendungsfällen der Testfallbeschreibung ergeben. So werden Anwendungsfälle vorgestellt, klassifiziert und Anforderungen abgeleitet. Die Anforderungen werden in funktionale und nicht-funktionale unterteilt. Aus den Anforderungen wiederum werden Bewertungskriterien für das nächste Kapitel und für die Bewertung des Konzepts abgeleitet.

- Stand der Technik analysieren in Kapitel 5

  Die Analyse und Bewertung aktueller Testfallbeschreibungsmittel erfolgt auf Basis der Bewertungskriterien. Daraus folgt die Aufdeckung der Lücken in den vorhandenen Mitteln.

- Sprache erstellen und umsetzen in Kapitel 6 und Kapitel 7

  Der Inhalt und Umfang der Testfälle beschreibt die notwendigen Elemente der Sprache. Auf dieser Grundlage wird die eigene Testfallbeschreibungssprache definiert. Diese Theorie ergänzt ein praktischer Kapitel, in denen ein Editor für die Erstellung der Testfallspezifikationen vorgestellt wird.

- Sprache anwenden (Validieren) und bewerten in Kapitel 8

  Die Validierung erfolgt durch die beispielhafte Anwendung der Testfall-Spezifikationsspr
  für funktionale Steuergerätetests. Hier wird die Beschreibung im Hinblick auf unterschiedliche Aspekte bei den Steuergerätetests betrachtet und die Sprache bewertet.

- Zusammenfassen und Ausblick darstellen in Kapitel 9

  Die Arbeit schließt mit einer Zusammenfassung und dem Ausblick ab.

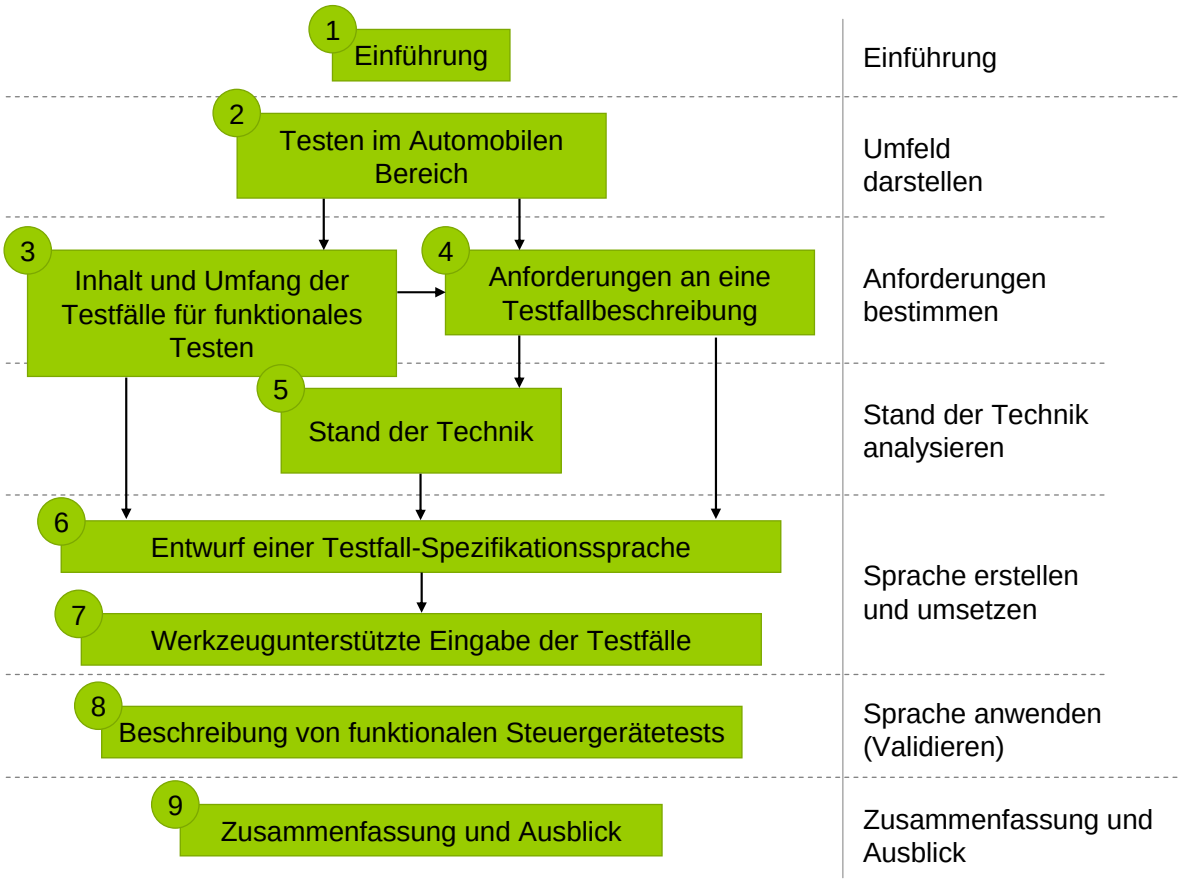



**Abbildung 1.4:** Gliederung der Arbeit

# Kapitel 2

# Entwicklungsbegeleitendes Testen von eingebetteter Software in der Automobilindustrie

Zweck dieses Kapitels ist das Umfeld der Arbeit darzustellen, um hieraus Einflüsse auf die Testfallbeschreibung abzuleiten. Hierfür wird im ersten Schritt der Systembegriff aus zwei unterschiedlichen Aspekten betrachtet. Anschliessend erfolgt eine Betrachtung der Vorgehensmodelle bei der Entwicklung der Systeme und damit Darstellung der Entwicklungsprozesse. Der Testprozess wird hier als ein Teil des Entwicklungsprozesses vorgestellt und die für das Testen relevanten Beziehungen zu der Testumgebung und die Eigenschaften des zu testenden Systems werden dargestellt. Anschliessend werden die Auswirkungen auf die Testfälle aufgezeigt.

## 2.1 Von Systemen und Komponenten

Den Mittelpunkt stellt das zu testende System dar. Dabei wird der Systembegriff in der Literatur unterschiedlich definiert. Zwei Arten werden für die unterschiedlichen Betrachtungen vorgestellt, die für diese Arbeit gelten soll.

### 2.1.1 Systembegriff nach der technischen Umgebung

Eine Betrachtung des Systembegriffs, die auch in den Arbeiten im Automobilumfeld ( [Hut06] und [Sch03]) zu finden ist, erfolgt aus der Sicht der technischen Umge-

bung:

**Definition 2.1 (System (Umgebung))** *Ein System besteht aus einer Menge von Komponenten, welche Eigenschaften besitzen und welche durch Relationen miteinander verknüpft sind [Pat82][a].*

[a] [Pat82] sagt weiterhin aus, dass eine Definition des Systems im Kontext betrachtet werden muss und von dem jeweiligen Ziel bzw. Zweck abhängt.

Komponenten, deren Eigenschaften und deren Relationen zeichnen also ein System aus. Dabei ist:

- Komponente

  Elemente, Teile, Bausteine oder Glieder eines Systems.
- Eigenschaften

  Die Eigenschaften einer Komponente können als diskrete Attribute, Funktionen oder Merkmale angegeben werden.
- Relationen

  Relationen definieren die Beziehungen zwischen Komponenten.

Je nach Betrachtung können unterschiedliche Systemgrenzen gezogen werden (nach Aufbau, nach Struktur oder nach Funktion). Abbildung 2.1 zeigt den Aufbau eines Systems nach dieser Begriffsdefinition (vgl. Abschnitt 2.3.2, S.23 über die Beteiligten bei der Entwicklung).

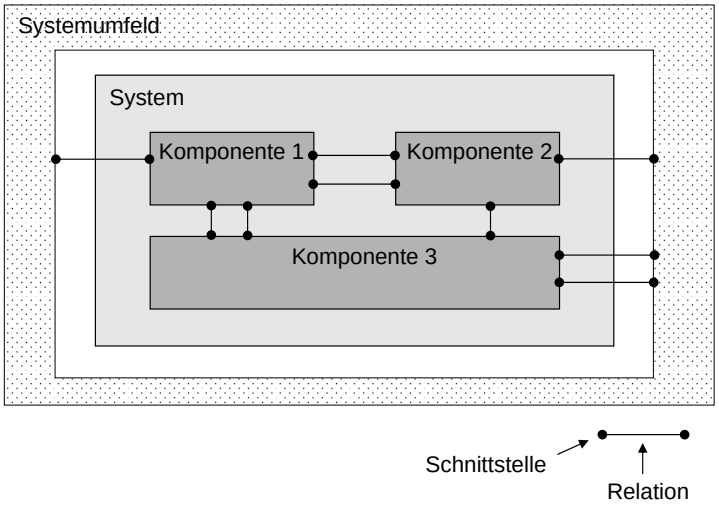



**Abbildung 2.1:** Aufbau eines Systems nach dem Systembegriff

Das System ist in sich abgeschlossen und wird über die Relationen und Schnittstellen zu seinem Umfeld, dem Systemumfeld, definiert.

Je nach Betrachtung kann eine Komponente selbst ein Teilsystem bilden, d.h. sie besteht wiederum aus Komponenten. Dann wäre das Systemumfeld die Umgebung der Komponente im System.

### Von Systemen und Komponenten im Fahrzeug

Abbildung 2.2 zeigt die Aufteilung der Ausstattungen im Fahrzeug auf die Systemebene.

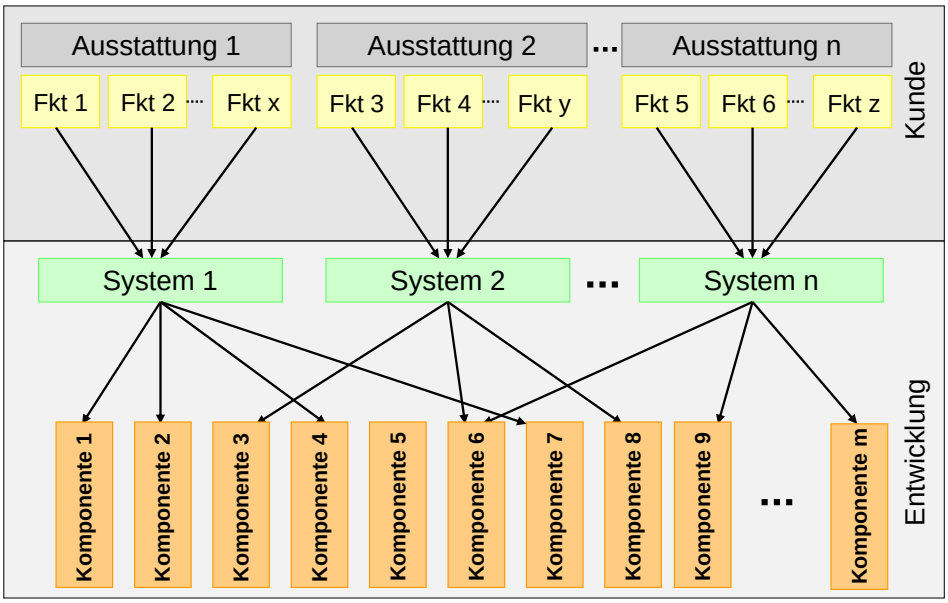



**Abbildung 2.2:** Aufteilung der Funktionen im Fahrzeug auf Systeme und Komponenten

Die Ausstattungsmöglichkeit ist die Grundlage für die Erstellung von Systemen für ein Fahrzeug. Die Wischerfunktion ist beispielsweise eine Ausstattung für ein Fahrzeug. Diese Ausstattung beinhaltet unterschiedliche Funktionen. So wäre eine Grundfunktion das Wischen und eine weitere das Intervallwischen.

Diese Ausstattung wird auf Systeme abgebildet. Ein System besteht auf dieser Ebene unter anderem aus elektrischen Komponenten, aber auch mechanischen. Bei dem Beispiel wäre die elektrische Komponente das eingebettete System (Definition siehe unten) und die mechanische Komponente das Wischerblatt bzw. die Mechanik dahinter.

Dabei kann eine Komponente in mehreren Systemen eingesetzt werden. Eine Komponente ist somit bei dieser Betrachtung (Sichtweise) die kleinste Einheit.

In dieser Arbeit wird das eingebettete System betrachtet, d.h. die mechanische Komponenten ausser Betracht gelassen. Diese eingebetteten Systeme erfüllen gemeinsam eine oder mehrere Funktionen. Dabei wird das Fahrzeug in Bereiche unterteilt, um die Komplexität zu beherrschen und für die unterschiedlichen Anforderungen passende Lösungen einzusetzen. Bevor diese Aufteilung in Abschnitt 2.7.4 betrachtet wird, ist eine Betrachtung der einzelnen Komponente notwendig. Dadurch können die unterschiedlichen Anforderungen später herausgestellt werden.

Für die Definition des eingebetteten Systems wird folgende hergenommen:

**Definition 2.2 (Eingebettetes System)** *Ein eingebettetes System (abgekürzt: ES) ist eine Software-/Hardware-Einheit, die über Sensoren und Aktoren mit einem Gesamtsystem verbunden ist und darin Überwachungs-, Steuerungs- beziehungsweise Regelungsaufgaben übernimmt. In der Regel handelt es sich bei eingebetteten Systemen um reaktive, häufig auch um hybride verteilte Systeme mit Echtzeitanforderungen. Typischerweise sind solche Systeme dem menschlichen Benutzer nicht direkt sichtbar, er interagiert unbewusst mit dem eingebetteten System. [BvdBK98]*

## Die Komponente bzw. das System Steuergerät

**Definition 2.3 (Steuergerät)** *Ein Steuergerät (engl.: electronic control unit, ECU) ist (insbesondere in der Automobiltechnik) die physikalische Umsetzung (Implementierung) eines eingebetteten Systems. In mechatronischen Systemen bilden Steuergerät und Sensorik-/Aktuatorik oft eine Einheit. [BvdBK98]*

Steuergeräte sind also eingebettete Systeme. Mechatronische Systeme sind solche, bei denen die „Elektronik zur Steuerung und Regelung mechanischer Vorgänge räumlich eng mit den mechanischen Systembestandteilen verbunden“ [BvdBK98] sind.

Aus der obigen Herleitung werden die Steuergeräte als Komponenten betrachtet, d.h. relativ zu dem Gesamtsystem ist ein Steuergerät eine Komponente. Bei der Entwicklung eines einzelnen Steuergerätes wird sie jedoch per Definition zu einem System. Dann sind die Komponenten die einzelnen Teile im Steuergerät.

Eine andere Sichtweise auf das System entsteht aus der Betrachtung, wie das System die eingehenden Informationen verarbeitet. Das führt dazu, dass der Systembegriff aus Sicht der Verarbeitung ebenfalls definiert werden kann.

### 2.1.2 Systembegriff nach der Verarbeitung

**Definition 2.4 (System (Verarbeitung))** *Unter einem System versteht man ein mathematisches Modell S, das einem Eingangssignal der Größe x ein Ausgangssignal y der Größe $y = S(x)$ zuordnet. [Sch07]*

Das bedeutet, das ein System die Eingangssignale in Ausgangssignale umwandelt bzw. verarbeitet. Diese formale Definition wird durch die Klassifikation der Systeme deutlicher.

#### Klassifikation der Systeme

Aus der Sicht der Verarbeitung werden Systeme grundsätzlich in drei Arten aufgeteilt ( [Sch07]):

- Transformationelle Systeme
- Interaktive Systeme
- Reaktive Systeme

Transformationelle Systeme besitzen alle benötigten Informationen (Eingaben) vor der Verarbeitung dieser in die Ausgabe. Als Beispiel sind hier die Kommandozeilenprogramme zu nennen, wo der Programmname zusätzlich Eingabewerte erhalten kann. Durch die Verarbeitung ('transformation') erfolgt eine Ausgabe und das Programm ist damit auch terminiert.

Interaktive Systeme agieren von sich aus: Benötigen sie bestimmte Eingaben, so verlangen sie diese von der Umgebung z.B. durch eine Eingabeaufforderung. Das System reagiert nicht auf die Umwelt, sondern interagiert und synchronisiert sich selbst mit der Umwelt. Als Beispiel hierfür können alle Systeme mit einer Benutzerschnittstelle (Mensch-Maschine-System / Human Machine Interface) aufgezeigt werden, die mit dem Benutzer interagieren und nicht terminieren.

Bei den reaktiven Systemen hat die Umwelt den entscheidenden Einfluss auf das System, d.h. die Abstimmung der internen Daten mit der Umwelt findet durch die Umwelt selbst statt. Solche Systeme sind meist nebenläufig, müssen zuverlässig arbeiten und besitzen Zeitschranken, in denen sie die Verarbeitung (mit Ausgabe) abgeschlossen haben müssen.

Wie aus der Definition eines eingebetteten Systems zu erkennen ist, sind diese zum größten Teil reaktiv. Im Abschnitt 2.7, S. 40 wird auf das Steuergerät als System aus der Sicht der Verarbeitung genauer eingegangen.

### 2.1.3 Begriffsverwendung im Automobilen Umfeld

Die Definition mit Bezug auf die technische Umgebung ist bei der Betrachtung der Aufteilung der eingebetteten Systeme notwendig. So ist bei dem Entwicklungsprozess von Systemen und Komponenten die Rede, wenn die hierarchische Zusammensetzung von Bedeutung ist. Ebenso wird bei der Definition der Funktionsumfangs diese Definition verwendet.

Der Systembegriff ist aus der Sicht auf die Verarbeitung wird dann benutzt, wenn das einzelne oder die Menge von zusammengeschalteten Systemen Informationen verarbeitet bzw. umwandelt.

Im Folgenden wird zuerst der Entwicklungsprozess betrachtet, bei dem das System in Bezug zu seiner Umgebung betrachtet wird. Anschliessend erfolgt die Betrachtung des Systems nach dem informationsverarbeitenden Aspekt.

## 2.2 Vorgehensmodelle bei der Entwicklung von Software für eingebettete Systeme

Wie im Vorfeld bereits dargestellt, greifen mehrere Entwicklungsprozesse ineinander, um ein Fahrzeug zu entwickeln. Dabei wird das Gesamtsystem Fahrzeug in Teilsysteme unterteilt, die aus Komponenten bestehen. Jedes dieser Teile wird entwickelt und sukzessive zuerst in Teilsysteme und daraufhin zum Gesamtsystem integriert.

In diesem Abschnitt werden die Vorgehensmodelle bei der Entwicklung der eingebetteten Systeme genauer betrachtet, sowie die beteiligten Partner vorgestellt und typische Adaptionen des Fahrzeugentwicklungsprozesses visualisiert. Darauf folgend wird der Testprozess als ein Teil der Qualitätssicherung und des Enwicklungsprozesses dargestellt.

Beim Entwicklungsprozess steht der Prozess im Vordergrund.

**Definition 2.5 (Prozess)** *„Eine teilweise geordnete Menge von Aktivitäten, die ausgeführt werden können, um ein erwünschtes Endergebnis im Sinne eines erstrebten Ziels zu erreichen.“ [ISO19439]*

Ein besonderer Prozess ist das Vorgehensmodell bei der Entwicklung. Vorgehensmodelle werden durch Aktivitäten und Ergebnisse definiert um strukturiert und organisiert Software zu entwickeln (vgl. auch Definition zum Vorgehensmodell).

**Definition 2.6 (Vorgehensmodell)** *[Ein Software-entwicklungsmodell] beschreibt einen festgelegten organisatorischen Rahmen der Softwareentwicklung. Festgelegt wird, welche Aktivitäten in welcher Reihenfolge von welchen Rollen zu erledigen sind und welche Ergebnisse dabei entstehen und wie diese in der Qualitätssicherung überprüft werden. [IST07]*

Es existieren in der Literatur unterschiedliche Arten von Vorgehensmodellen [Bal98]. Im Folgenden werden einige Vorgehensmodelle mit unterschiedlichen Schwerpunkten betrachtet.

Das Wasserfallmodell [Roy87] teilt die Entwicklung in Stufen ein. Es besteht im Groben aus der Definitions-, Entwurfs- und Implementierungsphase. Bei diesem Modell *fliesst* das Ergebnis jeder Phase in die nächste Phase. Zusätzlich sind die Phasen mit jeweils der vorherigen gekoppelt, um bei nicht ausreichenden Ergebnissen der Vorphase diese nochmal durchlaufen zu können. Der Nachteil ist, dass eine Rückkopplung zu davor liegenden Phasen nur für die erste Ebene definiert ist und weitere Rückkopplungen somit nicht möglich sind.

Das Wasserfallmodell bildet für das V-Modell 97 (nur Subsystem Softwareentwicklung) die Grundlage, wo zusätzlich zu den Phasen Aspekte der Qualitätssicherung betrachtet werden. So werden die Ergebnisse der Implementierunggegen die Spezifikationen der einzelnen Phasen des Entwurfs **verifiziert** und zum Schluss folgt eine **Validierung** gegen die ursprüngliche Vorstellung, was man sich von der Entwicklung erwartet hat. Mehr zum V-Modell im nächsten Abschnitt.

**Definition 2.7 (Verifikation)** *Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind. [IST07] (siehe auch [Pat05b])*

**Definition 2.8 (Validierung)** *Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind. [IST07] (siehe auch [Pat05b])*

Neben diesen Vorgehensmodellen, die von Beginn eine vollständige Spezifikation des zu entwickelnden Systems verlangen, existieren auch Vorgehensmodelle zur evolutionären bzw. inkrementellen Entwicklung [Bal98].

Das Spiralmodell [Boe86] wird als Metamodell bezeichnet, da innerhalb seiner Phasen weitere Vorgehensmodelle verwendet werden können. Das Modell wird in vier Phasen gegliedert, wo in der ersten Phase die Ziele bestimmt werden und danach eine Risikoanalyse durchgeführt wird. Anschliessend wird die Entwicklung auf einem definierten Vorgehensmodell erstellt. Hierfür können existierende Modelle wie das Wasserfallmodell oder das V-Modell verwendet werden. In der letzten Phase wird die Planung für den nächsten Zyklus erstellt.

## 2.3 Entwicklung von Software für eingebettete Systeme nach dem V-Modell

Die Entwicklung von Steuergeräten wird an das V-Modell angelehnt [BSS05].

Obwohl eine neuere Fassung des V-Modells 97 existiert (V-Modell XT [HH08]), wird in diesem Abschnitt dennoch das V-Modell 97 betrachtet.

Der Hauptgrund für die Darstellung des alten Modells ist, dass die Aktivität im V-Modell 97 im Vordergrund steht, wohingegen im neuen Modell (XT) Produkte in den Vordergrund gestellt werden [HH08]. Für die Hinführung zum Testen ist das alte Modell somit geeigneter.

Weiterhin basieren die visuellen Darstellungen der Fahrzeugentwicklung auf der älteren Version (vgl. folgende Abschnitte), genau sowie die inhaltlichen Erklärungen, der an die Entwicklung angepassten Prozesse (vgl. Abschnitt 2.3.3, S. 24).

Die Grundidee wird aufgegriffen, wie sie durch Boehm [Boe79] gelegt und durch das V Zeichen geprägt wurde. Somit liegt der Fokus auf dem Subsystem Softwareentwicklung (siehe unten), sodass in den Folgenden Abschnitten mit dem Begriff V-Modell (bzw. V-Modell 97) das Subsystem Softwareentwicklung gemeint ist.

### 2.3.1 V-Modell 97

Das V-Modell 97 besteht aus vier Submodellen. Das Gesamtmodell besteht aus dem Submodell Systemerstellung (SE), um den herum sind die Submodelle zur Qualitäts-

sicherung (QS), zum Konfigurationsmanagement (KM) und zum Projektmanagement (PM) definiert [IAB97].

Das Gesamtmodell muss mit seinen Submodellen an das jeweilige Projekt bzw. Einsatzumgebung angepasst werden (tailoring).

Bekannt ist das V-Modell vor allem durch das V Zeichen, die sich aus der Darstellung der Systemerstellung in der linken Seite die Entwicklung, unten die Implementierung und in der rechten Seite das Testen ergibt. Abbildung 2.3 zeigt ein sehr nah am V-Modell orientiertes Modell, das das Vorgehen im Automobilumfeld widerspiegelt (nach [Hut06]).

Die Phasen besitzen fassbare Artefakte, d.h. nach jeder Phase werden Ergebnisse in Form von Dokumenten erstellt, die aus den Aktivitäten der Phasen entstehen.

Wie im vorherigen Abschnitt dargestellt, erfolgt die Verwendung des Systembegriffs aus der Sicht der jeweiligen Betrachtung. Bei der Entwicklung eines Fahrzeugs wird von einem Gesamtsystem gesprochen, das in seine Teilsysteme (Subsysteme) gegliedert wird und die wiederum in die einzelnen Komponenten, den Steuergeräten. So wird eine System-Spezifikation wie folgt definiert[1]:

**Definition 2.9 (System-Spezifikation)**
*Die Systemspezifikation beschreibt alle funktionalen und nicht-funktionalen Anforderungen an ein Systemelement (System, [...] oder Segment) [IAB09]*

Wenn im Zusammenhang mit der System-Spezifikation von einer Komponenten-Spezifikation gesprochen wird, so ist diese eine Spezifikation einer Komponente des zu testenden Systems (meist ein Teildokument der übergeordneten System-Spezifikation).

### Designphasen: Spezifikation, Design und Implementierung

In der Abbildung 2.3 ist zu erkennen, dass die Aufteilung des Gesamtsystems in Teilsysteme bis hin zu Komponenten hierarchisch erfolgt. Zu jedem Teilabschnitt existieren eine Anforderungs- und Designphase.

So muss nach der Spezifikationsphase (Anforderungsphase) ein Lastenheft vorliegen. Nach dem Grobentwurf liegt das Pflichtenheft vor und bis hin zur Implementierung

[1] Eine inhaltliche Begriffsdefinition der Spezifikation erfolgt im nächsten Kapitel in Abschnitt 3.2.2, S. 62

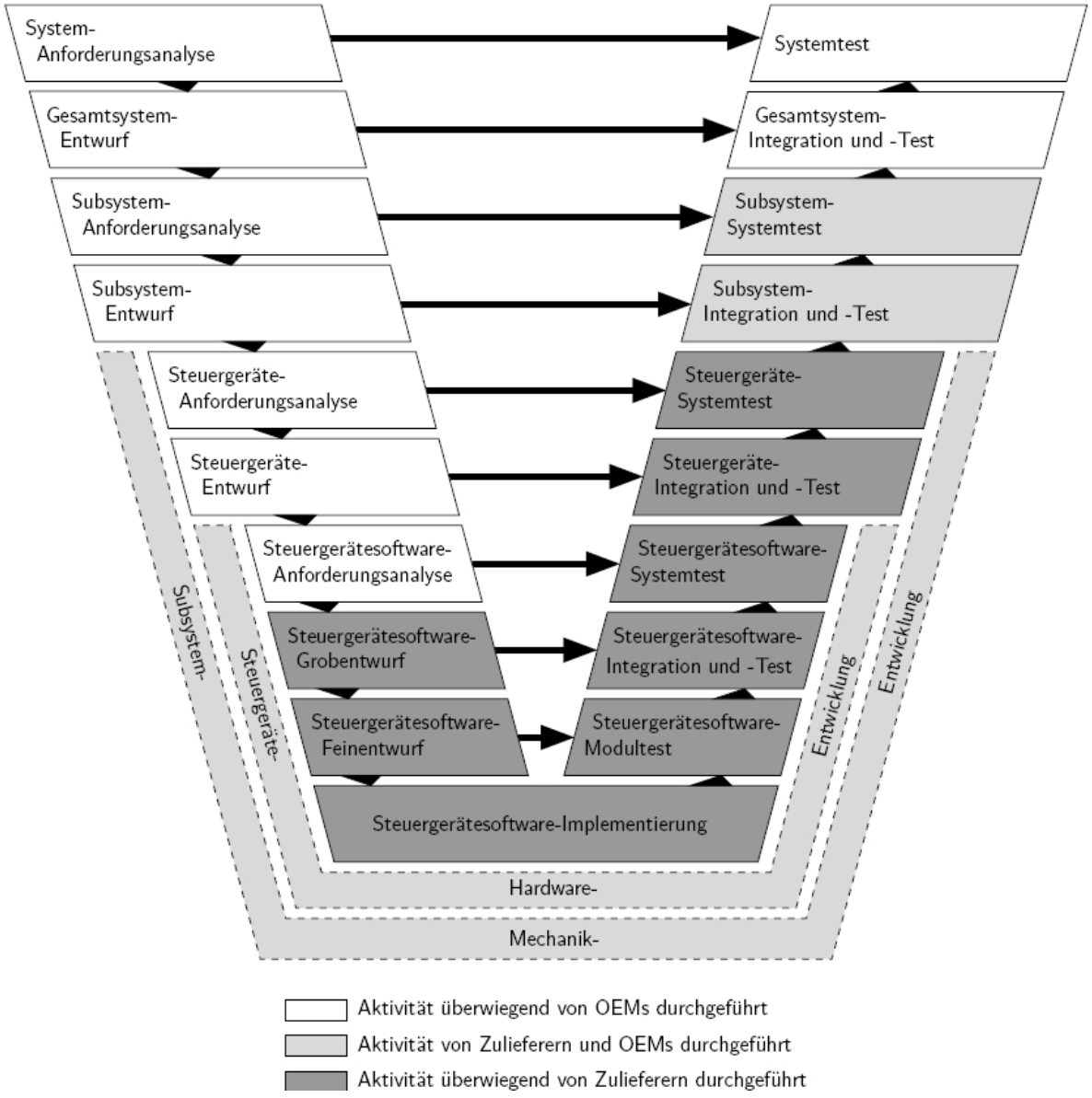



**Abbildung 2.3:** Erweitertes V-Modell für die Steuergeräteentwicklung in der Automobilindustrie [Hut06]

werden zahlreiche Dokumente erstellt, die im rechten Ast des Vs geprüft werden sollen. Diese bezeichnet man als **Testbasen**.

Die Spezifikation wird in der Automobilbranche in erster Linie in natürlicher Sprache verfasst, die in der Designphase bis hin zur Implementierung durch ein Modell umgesetzt wird [GS05], [CFGK05], [Con09]. Hier werden häufig computergestützte graphische Modellierungswerkzeuge verwendet.

**Definition 2.10 (Testbasis)** *[...] Dokumente, aus denen die Anforderungen ersichtlich werden, die an eine Komponente oder ein System gestellt werden, bzw. die Dokumentation, auf der die Herleitung oder Auswahl der Testfälle beruht. [IST07]*

#### Testphasen: Komponententest, Integrationstest, Systemtest

In den einzelnen Testphasen werden Tests zu den Dokumenten aus der Entwicklungsphase erstellt. Umgekehrt zu den Phasen im Design, wird das Gesamtsystem von unten, d.h. von den Komponenten anfangend überprüft. Nach den Komponenten werden die Teilsysteme und schliesslich das Gesamtsystem geprüft. So wird z.B. in dem Integrationstest die Implementierung gegen die Spezifikationen in dem Groblastenheft verifiziert.

Zum Schluss erfolgt eine Validierung. Hier wird validiert, ob das Endprodukt tatsächlich nach den Wünschen des Kunden erstellt wurde. Dazu dient in der Theorie kein Dokument mehr als Grundlage, sondern der Kunde nimmt die Validierung anhand seiner Wünsche vor.

### 2.3.2 Die Beteiligten Partner bei der Entwicklung

Die Entwicklung von einem Fahrzeug wird nicht nur durch ein Unternehmen vollzogen, sondern durch eine Reihe von Unternehmen. Diese können hauptsächlich drei Kategorien zugeordnet werden:

- Der OEM

  OEM ist eine Abkürzung für „Original Equipment Manufacturer“. Mit dieser Abkürzung werden die Automobilhersteller benannt.

- Die Zulieferer

Zulieferer entwickeln u.a. Steuergeräte. Für diese ist das Steuergerät das System. Bei jeder Entwicklung eines Steuergerätes werden Vorgehensmodelle ähnlich dem V Modell verwendet. Auch hier wird angefangen von einem Kundenwunsch (hier OEM), zur Spezifikation, dem Grob- und Feinentwurf bis hin zur Implementierung entwickelt. Das Testen wird ebenfalls auf Komponententest, Integrationstests, Systemtests und Abnahmetest ausgedeht.

Zulieferer werden auch als Tier bezeichnet. Dabei bezeichnet ein Tier 1 einen direkten Kontakt eines Zulieferers zu einem OEM, ein Tier 2 einen Kontakt eines Zulieferers zu einem Tier 1 Zulieferer. Diese Kette wird durch die Nummerierung fortgeführt.

- Die Dienstleister

  Die Entwicklung beschränkt sich nicht nur auf OEMs und Zulieferer, sondern auch auf Dritte, die Aufgaben und Tätigkeiten übernehmen: den Dienstleistern. Ihre Aufgaben umfassen weitestgehend den Aufgaben der OEMs und der Zulieferer, soweit diese die Arbeiten auslagern. Die Dienstleister passen ihre Vorgehensmodelle an die Modelle der Unternehmen an oder verwenden, falls möglich, eigene Vorgehensmodelle.

### 2.3.3 Modifikationen des V-Modells

Besonders durch die vielen Beteiligten reicht die Darstellung der Entwicklung durch ein einfaches V-Modell nicht aus [BN03]. Das entwickeln eines Fahrzeugs wird in mehrere Teile untergliedert und das V-Modell in seinen Grundzügen (Design, Implementierung, Testen) in einem Metamodell verwendet. So sind Modelle definiert, die mehrere V Zeichen[2] nebeneinander bzw. hintereinander definieren, aber auch untereinander oder ineinander.

Die Darstellung hängt im Grunde von der Sicht ab mit der man auf die Entwicklung blickt.

Ein Metamodell, das beispielsweise in der Automobilindustrie eingesetzt wird, ist nach den Musterphasen ( [Gud03], [Mül07], [Har01b]) gegliedert und entspricht der evolutionären bzw. inkrementellen Entwicklung (siehe oben). Mit dem Modell der Musterphasen werden Abschnitte bezeichnet, bei denen eine vordefinierte Form des zu entwickelnden System vorliegen muss, um Qualitätssichernde Maßnahmen vorzeitig einzuleiten. Wobei Beginn Muster reine Funktionsdemonstratoren darstellen, werden in späteren Phasen vollwertige Funktionsstände als Ergebnisse definiert. Hierbei spricht man vom A-Muster, das die Grundfunktionen abbildet, bis hin zum D-Muster das zur Serienfertigung benutzt wird. Nach dem Musterphasen Modell können

[2] Mit dem V Zeichen ist der Gedanke der Verifikation und Validierung gemeint, wie sie in [Boe79] geprägt wurde.

so mehrere V-Modelle nebeneinander bzw. hintereinander auftauchen, genannt *multiple V-Modelle* [BN03] (vgl. auch [NSL08]).

Bei einem OEM-Zulieferer Verhältnis werden die V-Modelle der jeweiligen Beteiligten untereinander geordnet betrachtet, so dass mit der Implementierungssicht seitens des OEMs, das V-Modell des Zulieferers beginnt. Sind (üblicherweise) mehrere Zuflieferer beteiligt, so können mehrere V-Modelle unterhalb nebeneinander nochmal auftauchen. Abbildung 1.2, S. 5 verdeutlicht diesen Zustand.

Die ineinander geschachtelten V-Modelle sind bei den Software- und Hardwareentwickelnden Zulieferern zu finden (vgl. auch Abbildung 2.3, wo die Aktivitäten der Beteiligten symbolisch durch ein V im V Zeichen eingebettet sind).

### 2.3.4 Zusammenfassung

Der Entwicklungsprozess entsteht aus inkrementellen Vorgehensmodellen mit mehreren aufeinander folgenden Teilprozessen, die als Grundlage den Entwurf, die Implementierung und das Testen besitzen, wie sie im V-Modell zu finden ist. Die Teilprozesse werden durch unterschiedliche Beteiligte ausgeführt.

Das Testen ist somit ein fester Bestandteil des inkrementellen Vorgehensmodells bei der Fahrzeugentwicklung, bei dem die Ergebnisse in jeder Teilphase in die Entwicklung zurückfliessen.

## 2.4 Prüfverfahren aus der Qualitätssicherung

Das Testen ist ein Teil der Qualitätssicherung.

**Definition 2.11 (Qualitätssicherung)** *Teil des Qualitätsmanagements, das darauf gerichtet ist, Vertrauen in die Erfüllung der Qualitätsanforderungen zu erzeugen [IST07]*[a]

[a] nach ISO 9000.

Dabei ist das Qualitätsmanagement, „[a]ufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität. [...]“ [IST07]. Und die Qualität, der „[...] Grad, in dem ein System, eine Komponente oder ein Prozess die Kundenerwartungen und -bedürfnisse erfüllt“ [IST07].

Im Folgenden wird ausgehend von möglichen Fehlerarten eine Klassifikation von Prüfverfahren vorgestellt. In diese Klassifikation wird das Testen, genauer das dynamische Testen eingeordnet. Ein weiterer Zweig des dynamischen Testverfahrens ist das sogenannte funktionale Testen, wie sie als Aktivität bei dem V-Modell auftaucht und den Schwerpunkt der Untersuchung dieser Arbeit darstellt.

### 2.4.1 Klassifikation von Fehlern

Die Begriffe Fehlverhalten, Fehler und Irrtum werden nach der Definition von [Lig02] Definition verwendet.

- Irrtum

  Die Ursache eines Fehlers ist ein Irrtum seitens des Programmierers.

- Fehler

  Ein Fehler ist im Programmcode die Ursache für ein Fehlverhalten bei der Ausführung.

- Fehlverhalten

  Ein Fehlverhalten oder Ausfall taucht bei der Durchführung auf. Die Ursache hierfür ist ein Fehler bei der Software.

Wie aus der Definition leicht zu erkennen ist, sind die Fehlerarten miteinander verknüpft, d.h. bei einem Fehlverhalten muss es ein Fehler im Programmcode gegeben haben, das auf einem Irrtum des Programmierers zurückzuführen ist. Allerdings können Fehler nicht immer durch eine Durchführung aufgedeckt werden, obwohl sie im Programmcode existieren.

Um diese Fehlerarten aufdecken zu können, wurden unterschiedliche Ansätze konzipiert. Im Groben werden zwei sich vervollständigende Techniken, die statischen und die dynamischen Prüftechniken, verwendet.

So kann zwar ein Fehler z.B. bei einer Durchsicht (Überprüfung des Dokuments bzw. des Quellcodes der Implementierung) aufgedeckt werden (statisch), jedoch nicht ein Fehlverhalten. Hierfür ist die Durchführung des zu testenden Systems notwendig (dynamisch).

### 2.4.2 Arten der Qualitätssicherung

Die statischen und die dynamischen Prüftechniken werden weiter feiner unterteilen. In [Lig02] werden die Prüftechniken klassifiziert. Abbildung 2.4 zeigt ein Ausschnitt aus dieser Klassifikation.

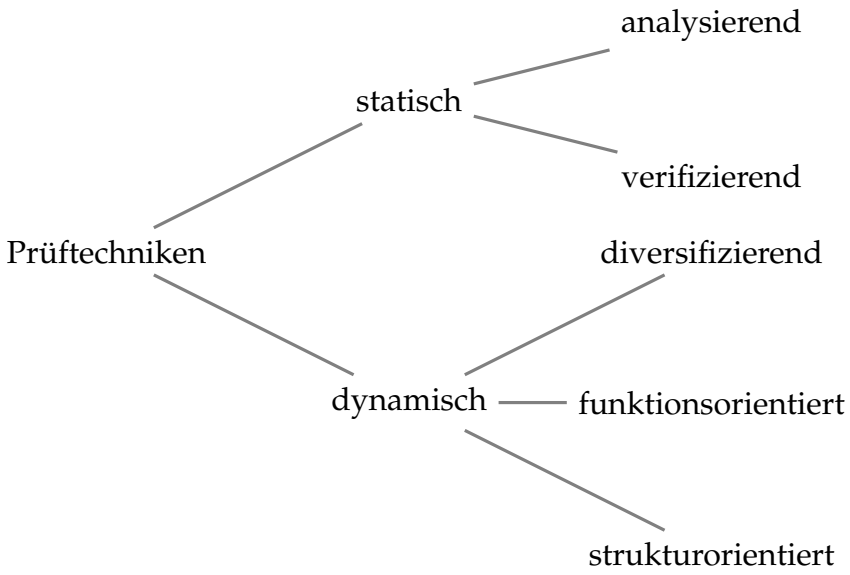



**Abbildung 2.4:** Ausschnitt aus der Klassifikation der Prüftechniken nach [Lig02]

Die statischen Prüftechniken führen keine Implementierung aus, sondern die Analyse beschränkt sich auf die Untersuchung der erstellten Dokumente, z.B. dem Programmcode.

Im Gegensatz dazu wird bei dynamischen Tests die Software ausgeführt: Die statische Prüfung kann fehlerhaft sein (bei der statischen Verifizierung) und berücksichtigt nicht die realen Bedingungen bei der Durchführung [Hal90]. Durch die Ausführung wird gewährleistet, dass sich Fehler aus dem Zusammenspiel zwischen Software und der Einsatzumgebung bzw. der Hardware aufgedeckt werden können[3].

In der Automobilindustrie werden sowohl statische als auch dynamische Prüftechniken eingesetzt. Im Folgenden wird aus dem dynamischen Bereich die funktionalen Tests genauer betrachtet.

[3] wo u.a. die Programmiersprache, das Betriebssystem und die Hardware einen Einfluss auf die Durchführung haben.

### Dynamisches Testen

In „The Art of Software Testing“ [MSBT04] wird das Testen mit dem Gedanken der Ausführung des Programms definiert: Die Absicht (der Gedanke, der Wille) Fehler zu finden, bei der Ausführung steht im Vordergrund: „Testing is the process of executing a program with the intent of finding errors.“. Der Unterschied zur normalen Ausführung um - mit der Absicht - diese zu benutzen, ist rein von der Aktorik nicht vorhanden. Dabei hat Myers bei seiner Definition nicht den **Prozessbegriff** als solches aufgegriffen[4].

Jedoch wird das Testen aufgrund seiner Komplexität einem Prozess unterworfen.

So wird in der neueren Literatur das Testen aus einer prozessorientierten Sicht dargestellt: „[Testen ist] Der Prozess, bestehend aus allen Aktivitäten des Lebenszyklus, der sich, sowohl statisch als auch dynamisch, mit der Planung, Vorbereitung und Bewertung eines Software-Produkts und damit verbundener Arbeitsergebnisse befasst, um sicherzustellen, dass sie die festgelegten Anforderungen erfüllen, um zu zeigen, dass sie ihren Zweck erfüllen, und um Fehler zu finden.“ [IST07] (vgl. auch [PvV98]).

In der Automobilindustrie werden hierfür Testprozesse aufgesetzt (vgl. [Bär08], [KRRS05], [BSV06]), die in den Grundzügen ähnlich aussehen, aber an die Tätigkeitsbereiche der Unternehmen angepasst sind. In Abschnitt 2.5, S. 29 wird auf den Testprozess näher eingegangen.

### Funktionales Testen

Wie im V-Modell dargestellt wird gegen die Spezifikation verifiziert. Diese Art von Prüfverfahren wird funktionales Testen bzw. funktionsorientiertes Testen genannt.

**Definition 2.12 (Funktionsorientierter Test)** *Dynamischer Test, der die Testfälle aus der Spezifikation ableitet und die Testvollständigkeit anhand der Abdeckung der Spezifikation mit Testfällen bewertet. [...] [Lig02]*

Die Überprüfung erfolgt auf den Spezifikationen, die aus den Designphasen (im V-Modell) entstanden sind.

[4] Das Wort ’process’ beschränkt er in der Definition sogar nur auf die Ausführung des Programms.

### 2.4.3 Zusammenfassung

Im Automobilen Bereich werden neben den statischen Prüftechniken auch dynamische Prüftechniken angewandt. Hier sind vor allem die funktionalen Tests für das Testen von Steuergeräten im automobilen Umfeld von Bedeutung, da die Wechselwirkung zwischen dem zu entwickelnden Code, dem Betriebssystem, der (entwickelten) Hardware, aber auch nur der Code an sich (in einer simulierten Umgebung) getestet werden kann.

In der Definition des dynamischen Tests sind drei Objekte von Bedeutung: Das System, die Betriebsumgebung und die Testfälle.

Im Folgenden wird zuerst auf den Testprozess eingegangen. Anschliessend werden Testumgebung (Betriebsumgebung) und Testobjekte (System) aufgezeigt.

## 2.5 Der Testprozess

**Definition 2.13 (Testprozess)** *Umfasst alle Aktivitäten, die zur Planung und Steuerung, Analyse und Design, Realisierung und Durchführung, Auswertung und Bericht sowie wie zum Abschluss der Testaktivitäten in einem Projekt benötigt werden. [...] [IST07]*

Das Testen unterliegt wie beschrieben einem Prozess. Wie jeder Prozess ist der Testprozess in Phasen aufgeteilt (siehe Definition Testprozess). Abbildung 2.5 zeigt eine mögliche Ausprägung des Testprozess: *PROVEtech:TP5* [BHS07]. Im Folgenden werden die einzelnen Phasen kurz erklärt. Die detaillierte Ausführung des Testprozesses wird in [Bär08] beschrieben.

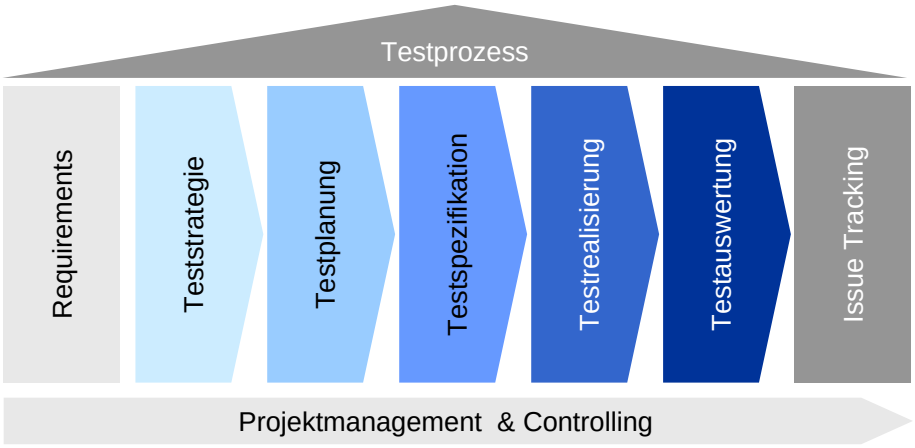



**Abbildung 2.5:** PROVEtech:TP5 [BHS07] als eine mögliche Instanz des Testprozesses

Die Idee bzw. der Wunsch eines Kunden werden durch die Anforderungen festgehalten. Im Requirements Prozess wird in einem systematischen Vorgehen das externe Verhalten eines Systems aus der Anwendersicht dokumentiert. Diese Anforderungen werden in der (System-, Komponenten-)Spezifikation niedergeschrieben.

Diese Spezifikation des eingebetteten Systems, die Testbasis, wird im Testprozess aufgegriffen, sodass diese Phase eine notwendige Voraussetzung für den Testprozess ist.

### 2.5.1 Phasen im Testprozess

Der Testprozess ist in fünf Phasen gegliedert: Teststrategie, -planung, -spezifikation, -implementierung und -durchführung [Sax08].

Zuerst wird eine Teststrategie definiert, d.h. Personen werden allokiert, welche Komponenten getestet werden sollen, welche Testbasis hergenommen wird und welchen Umfang der Test haben soll wird bestimmt. Letzteres beinhaltet die durchzuführenden Tests und die explizite Aufzeichnung welche Testarten nicht durchgeführt werden sollen. Hier wird auch festgelegt, wie die Inhalte, Umfänge und Ziele in bestimmten Abständen überprüft und angepasst werden.
Testintensität, Testart, Testherleitung, Priorisierung, Risikomanagement und Aufwandsabschätzung sind weitere Aufgaben in dieser Phase.

In der Testplannung werden die Arbeitspakete aus den Ergebnissen der vorherigen Phase abgeleitet und nach Priorisierung angeordnet. Welche Ressourcen (Anlage,

Ausrüstung, Kompetenzen) für die Arbeitspakete benötigt werden und das fortlaufende Verfolgen (Tracking) der Arbeitspakete werden geklärt. Zum Schluss steht ein Plan über den Ablauf der verfolgt wird.

In der Testspezifikation wird die Testspezifikation erstellt. Die Phase beinhaltet die Gliederung und Erstellung der Testfallspezifikationen aus der Testbasis. Auch hier gehört das fortlaufende Verfolgen von den erstellten Dokumenten und deren Anpassung zur Aufgabe. Werden Informationslücken und Fehler in der Spezifikation entdeckt, so werden in einer Schleife zusammen mit dem Entwickler diese beseitigt und die neu gewonnen Informationen fliessen in die Spezifikation und in das Testspezifikation zurück.

In der Testrealisierung werden die spezifizierten Testfälle implementiert und ausgeführt. Auch hier können Informationslücken und Fehler gefunden werden, so dass das Testlastenheft bearbeitet werden muss. Diese Informationen fliessen ebenfalls in die Spezifikation mit ein.

In der Testauswertung werden die Ergebnisse der Testrealisierung überprüft: Entspricht ein Testergebnis nicht dem erwarteten Ergebnis so wird diese in der Phase Issue Tracking dem Entwickler zurückgemeldet. Das Issue Tracking ist dabei kein Teil des Testprozesses, sondern wird dem Entwicklungsprozess zugeordnet.

Die Phasen sind nicht als zeitbeschränkte, in sich abgeschlossene Schritte zu sehen, sondern legen jeweils die Grundlagen und den Überbau für die folgenden Phasen fest, sowie die Handlungen für die fortlaufende Überwachung und die eventuelle Anpassung an die Gegebenheiten (Ressourcen, ...).

### 2.5.2 Einordnung der Testphasen zu den Entwicklungsphasen

Weil eben das Testen einem Prozess unterliegt, müssen die ersten Phasen vor dem erreichen der rechten Seite des V-Modells durchgeführt werden. Abbildung 2.6 zeigt die Zuordnung der einzelnen Testphasen in das V-Modell nach [Mül07].

Die Teststrategie und Testplanung werden zu Beginn mit den (System-, Komponenten-)Spezifikations- und Designphasen eingeleitet, um die parallel stattfindende Testspezifikation zu den Artefakten der Entwicklungsphasen zu erstellen. Die Testspezifikation setzt sich bei der Implementierungsphase der Entwicklung fort. Gleichzeitig findet die Testrealisierung statt, d.h. die erstellten Testfallspezifikationen werden umgesetzt und ausgeführt. Die Überlappung der Testspezifikation mit der Testrealisierung ist notwendig, da hier Informationen (Machbarkeit) aus der Testrealisierung in die Testspezifikation eingehen. Anschliessend wird die Testrealisierung mit der Testauswertung über die Testphasen der Entwicklung hin fortgeführt.

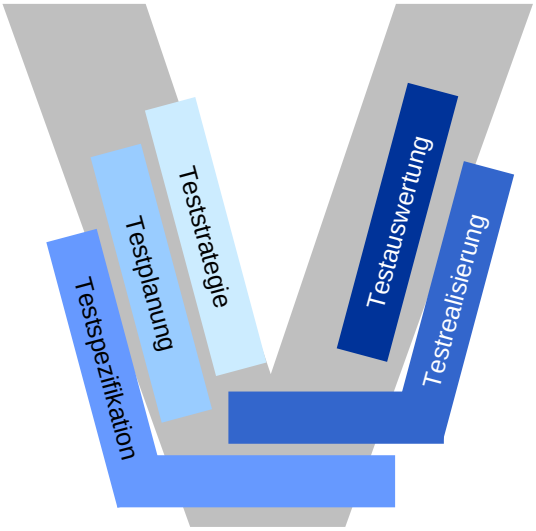



**Abbildung 2.6:** Testphasen im V-Modell eingeordnet (nach [Mül07])

Bei diesem Vorgehen spricht man von einem „entwicklungsbegleitendem Testprozess“.

### 2.5.3 Beteiligte Rollen

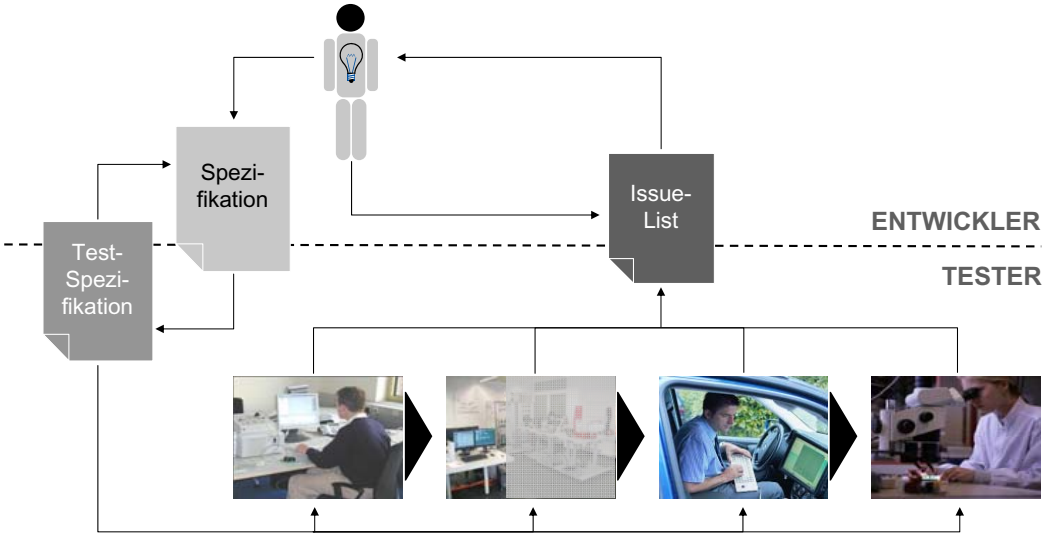



**Abbildung 2.7:** Rollenkommunikation aufgeteilt in Gruppen [Sax08]

Die Testspezifikation wird von mehreren Rollen in einem Testprozess verwendet. Abbildung 2.7 zeigt welche Rollen hauptsächlich mit der Testspezifikation involviert

sind (vgl. [Sax08]). Im Groben gibt es drei Arten von Rollengruppen: Die Tester, die Entwickler und die Reviewer.

Die Gruppe der Tester wird aufgegliedert in folgende Rollen:

- Testmanager / Projektleiter

  Der Testmanager ist besonders in den Phasen Teststrategie und Testplanung aktiv und übernimmt die Tätigkeiten aus diesen Bereichen. Er bildet gewissermaßen eine Schnittstelle zwischen den Projektleitern der Entwicklung und dem Testen. Zusätzlich ist er dafür verantwortlich, dass die Folgephasen nach Plan ausgeführt werden, die zuvor definiert wurden.

  Die Rollenbezeichnung 'Projektleiter' bezieht sich hierbei nur auf den Testprozess.

  Verwendete Werkzeuge: keine speziellen für das Testen, ausser für Planung und Tracking.

- Testingenieur / Test-Designer

  Der Testingenieur kommt in der Phase Testspezifikation zum Einsatz. Er erstellt die Testfallspezifikationen in enger Zusammenarbeit mit den Entwicklern.

  **Testingeniuere sind keine Programmierer**, d.h. sie besitzen keine Programmierkenntnisse.

  Verwendete Werkzeuge: natürliche Sprache, tabellen-basierte Werkzeuge für die Beschreibung von Testfällen.

- Testentwickler / Test-Programmierer

  Der Testentwickler setzt die spezifizierten Testfälle in ausführbare Testskripte um. Hier erfolgt die Software- und Hardwareabhängigkeit des Testfalls.

  Verwendete Werkzeuge: Testsystem (siehe Abschnitt 2.6, S. 35), Programmiersprachen (Skriptsprachen)

- Tester (Rolle) / Test-Operator

  Die Durchführung der Testkripte werden vom Tester übernommen. Eventuelle Auffälligkeiten gelangen durch diese Rolle in die Issue List.

  Verwendete Werkzeuge: Testsystem (siehe Abschnitt 2.6, S. 35)

Neben den Rollen im Testprozess gibt es noch die Rollen in der Entwicklung eines Steuergerätes, die sich durch ihre Aufgaben bedingt ebenfalls mit der Testfallspezifikation auseinandersetzen müssen.

- Entwickler

Diese Rolle steht für eine Gruppe von Rollen auf der Entwicklerseite. So kann damit ein Entwickler für eine Komponente gemeint sein, aber auch ein Entwickler für eine Reihe von Komponenten. Die Aufgabe aus der Sicht der Testfallspezifikation ist die gleiche: Die Rolle besitzt zusätzliche Informationen über das Testobjekt, die nicht festgehalten sind und ist gleichzeitig daran interessiert, welche Anforderungen mit der Testfallspezifikation überprüft werden sollen.

- Reviewer

  Für die Qualitätssicherung sind neben dem dynamischen Prüfverfahren auch die statischen Verfahren ausschlaggebend (siehe [Lig02]). Hier werden die vorhandenen Dokumente auf ihre formale und inhaltliche Korrektheit hin überprüft. Dabei fällt die Detailtiefe der Prüfung unterschiedlich aus.

  Die Rolle Reviewer bezeichnet somit alle Rollen, die an statischen Prüfverfahren teilnehmen und Einsicht und vor allem Verständnis in die Testfallbeschreibung erlangen müssen. [5]

#### Informations-Bedürfnisse der Rollen: Die Testfallspezifikation

Abbildung 2.7 zeigt eine schematische Darstellung der beteiligten Rollengruppen und ihrer Kommunikation bei dem entwicklungsbegleitenden Testprozess. Das zentrale Dokument bildet die Testspezifikation. Die beteiligten Rollen in diesem Szenario sind der Entwickler des Testobjekts, der Test-Designer und der Testentwickler. Nach dem Testprozess erfolgt im ersten Schritt die Kommunikation des Test-Designers mit dem Entwickler des Steuergerätes. Den Entwickler interessiert die Idee eines Testfalls: Welche Funktionalität der Anforderung wird durch diesen Testfall wie überprüft? Dabei sind die Detailinformationen bezüglich der Umsetzung des Testfalls weniger wichtig.

In den späteren Phasen des Testprozesses wird die Testspezifikation in die Implementierung überführt. Hier benötigt der Testentwickler, im Gegensatz zu dem Entwickler, Detailinformationen wie der Test ablaufen soll. Für das Grundverständnis ist es ebenfalls sinnvoll, dass der Testentwickler einen Überblick über den Testfall bekommt, d.h die Idee hinter dem Test kennt.

Das heisst, dass die Testfallspezifikation sowohl einen Überblick über den Testfall (die Idee) als auch Detailinformationen für die Testfallimplementierung zur Verfügung stellen muss.

[5] [CNP08] zeigt, dass u.a. nicht technisch orientierte Akademiker (fach-fremde) mehr Fehler finden als fachkundige Akademiker.

### 2.5.4 Erstellte Dokumente

Wie bei dem Vorgehensmodell in der Entwicklung entstehen auch bei den Phasen des Testprozesses unterschiedliche Dokumente.

Anhang A verweist auf die Dokumentationslandschaft des Testprozesses, wie sie vom IEEE 829 vorgeschlagen wurde. In der Praxis werden vielmals durch die Aufgaben der Rollen bedingt und durch die Prozessadaption die Dokumente zusammengefasst und umbenannt (oder gleich benannt mit anderen Schwerpunkten bei den Inhalten).

Im vorherigen Abschnitt wurde ein wesentliches Dokument bei der Verifikation und Validierung eingeführt: Die Testspezifikation. Im nächsten Schritt wird auf dieses Dokument näher eingegangen bis hin zum Testfall, dem elementaren Bestandteil bei der dynamischen Qualitätssicherung.

Weitere Dokumente werden mit der Definition des Testfalls in Kapitel 3, S. 59 eingeführt.

### 2.5.5 Der Testfall als elementarer Bestandteil des Testprozesses

In der Phase der Testspezifikation werden Testfälle entwickelt (siehe oben), die aus der Testbasis abgeleitet werden.

Im Automobilen Bereich besitzen diese Testfälle besondere Eigenschaften. Die Eigenschaften leiten sich aus dem Umfeld ab, d.h. der (System-, Komponenten-)Spezifikation und der Testumgebung und natürlich aus den Merkmalen der zu testenden Systeme.

Um den Inhalt und Umfang eines Testfalls definieren zu können, ist es notwendig, dieses Umfeld genauer zu betrachten. Anschliessend kann die Definition erfolgen. Die Definition wird im Kapitel 3, S. 59 ausgearbeitet.

## 2.6 Die Testumgebung

Die Tests werden in einem dafür konzipierten Umfeld durchgeführt. Dieses Umfeld wird Testsystem genannt.

**Definition 2.14 (Testsystem)** *Ein Testsystem ist ein rechnergestütztes Werkzeug, welches den automatisierten Test von elektronischen Komponenten und Systemen eines Kraftfahrzeugs gestattet. [Har01b]*

**Definition 2.15 (Testobjekt)** *Die Komponente oder das System, welches getestet wird. [...] [IST07]*

An das Testsystem wird das Testobjekt angeschlossen. Je nach Ausführung wird hier von einer Koppelebene gesprochen, die die Hardware oder Softwareschicht mit der internen Verarbeitungsschicht des Testsystems verbindet. Die Verarbeitungsschicht besteht aus einer Simulationseinheit, die die Umwelt des Testobjekts simuliert und einer Steuerungseinheit für die Durchführung der Testfälle. In der Regel befindet sich diese Verarbeitungseinheit in einem Echtzeitrechner. Die Steuerungseinheit besteht wiederum aus einer Kontroll- und Konfigurationseinheit. Abbildung 2.8 zeigt den Aufbau eines solchen Testsystems.

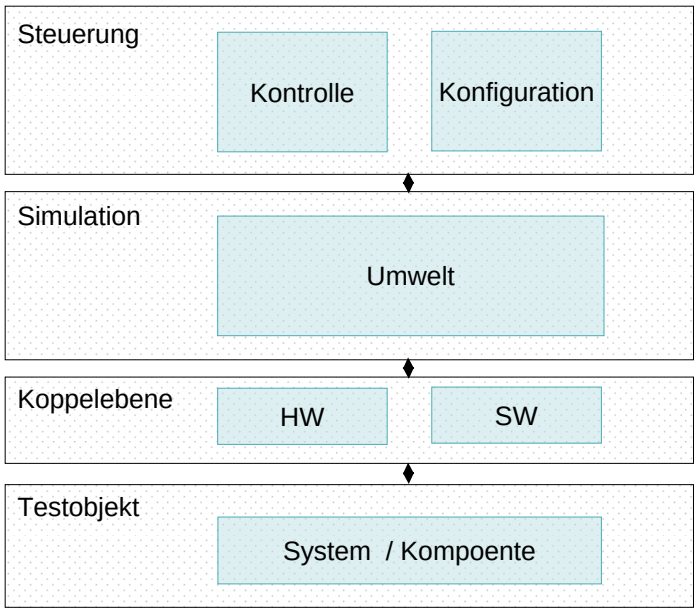



**Abbildung 2.8:** Allgemeiner Aufbau eines Testsystems (nach [Har01b])

Zwei Unterscheidungsmerkmale werden bei der Klassifizierung der Testsysteme betrachtet: Das Steuergerät auf der einen und die Umwelt des Steuergerätes auf der anderen Seite. Das Steuergerät kann dabei real oder simuliert sein; die Umwelt ebenfalls. Somit ergeben sich vier Arten von Testsystemen.

### 2.6.1 Model-in-the-Loop und Software-in-the-Loop

Im Model-in-the-Loop (MIL)) wird das Steuergerät simuliert, genauer definiert existiert ein Modell aus der Spezifikation oder das Modell bildet die Spezifikation selbst. Die Umwelt wird so realistisch wie nötig modelliert. Dabei werden im MIL Testsystem die notwendigen Sensoren und Aktoren im Fahrzeug, bis hin zu den äusseren Umweltbedingungen wie die Strassenlage und Witterungsbedingungen modelliert. Alle für das Steuergerät relevanten Einflüsse liegen im Modell vor und können beeinflusst werden. Beispielsweise nimmt die 'Schaltung' der Kupplung, das 'treten' des Gaspedals, die Steigung oder Krümmung der Strasse durch Beschreibung der Signalverläufe einen Einfluss auf das Modell und so reagiert dieser auf seine Umgebung. Die Reaktion wird aufgezeichnet und mit den Sollwerten verglichen.

Bei der Software-in-the-Loop Version wird statt des Modells ein kompilierter Code als Testobjekt hergenommen. Die Modellierungstiefe der Umwelt wird angepasst: Die Kopplung des Steuergerätes mit seiner Umwelt wird nicht in der Modellierung realisiert, sondern der Code wird über eine Softwareebene an das Umgebungsmodell verbunden. In diesem Testsystem werden die Testfälle immer noch über die Beschreibung der Signale durchgeführt.

### 2.6.2 Rapid-Prototyping

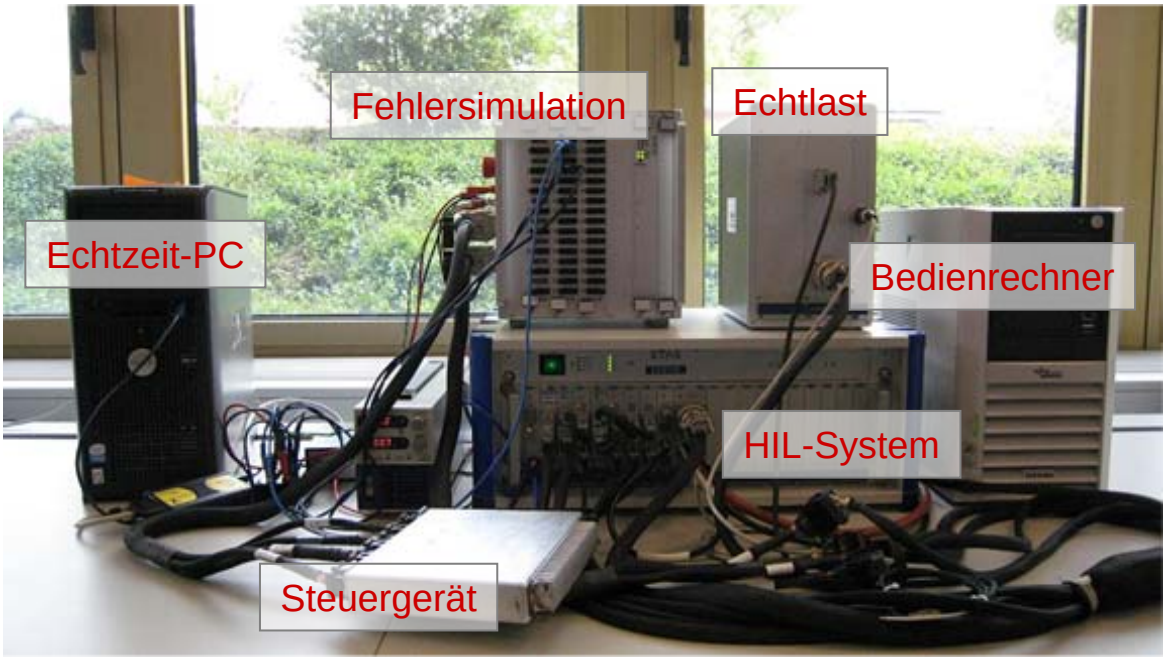
Rapidprototyping ist ein Testsystem welches eine reale Umgebung aber eine simulierte Steuergerät verwendet. Dabei wird ein Fahrzeug als Umgebung verwendet. Auf einem Evaluations-Board wird das Modell oder ein kompilierter Code vom Testobjekt aufgespielt und das Board an jedem Platz angeschlossen, wo das Testobjekt letztendlich fungieren soll. Rapid-Prototyping findet dort Einsatz, wo reale Umweltbedingungen relevant sind. In der Automobilindustrie wird dieses Testverfahren weniger eingesetzt als die anderen Testsysteme.

### 2.6.3 Hardware-in-the-Loop

Die Umwelt wird weiterhin simuliert, jedoch liegt das Testobjekt als reales Teil bestehend aus Hard- und Software zur Verfügung. Damit das Testobjekt an die Umgebung angekoppelt werden kann, müssen die Ein-und Ausgänge zwischen Testobjekt und dem Modell angepasst werden: Betrachtet man die Batteriespannung im Fahrzeug als Beispiel wird die Kopplung und die damit notwendigen Bausteine verständlich. Die Batteriespannung arbeitet auf einer Skala von 0 bis 12 V Bordspannung. Bei einem Testsystem sind interne Spannungen von 0 bis 5 V üblich. So muss eine Signalkondi-

tionierung zwischen diesen Werten erfolgen. Damit man nun im Modell die 0 bis 5 V verändern oder lesen kann wird eine zweite Umsetzung benötigt. So wird beispielsweise die Werteskala von 0 bis 255 auf 0 bis 5 V abgebildet. Im Modell kann nun der Modellierer auf seine Ganzzahlenwerte zugreifen und das Steuergerät bekommt diese 'für ihn gewohnten' Spannungsbereich zurück.

Die reale Umgebung des Testobjekts wird durch das HIL System simuliert oder - soweit sinnvoll - durch Echtlasten repräsentiert, besonders falls diese komplexes Verhalten aufweisen. Nicht vorhandene aber benötige Kommunikationspartner werden nachmodelliert. Die Modellierung erfolgt in einer Tiefe, wie sie vom Testobjekt benötigt wird. Die Umgebung kann durch die tatsächlichen Lasten, wie z.B. einem Blinker, durch Ersatzlasten (z.B. veränderbarer Widerstand) oder wiederum durch das HIL System (Modell) dem Testobjekt zur Verfügung gestellt werden. Somit 'denkt' das Testobjekt es sei in einem realen Umfeld tätig und kann Tests unterzogen werden.




Quelle: MBtech

**Abbildung 2.9:** Ein HIL Testsystem für den Test eines Steuergerätes

In Abbildung 2.9 ist ein HIL-Testsystem, die Betriebsumgebung, für den Test eines Steuergerätes zu sehen.

### 2.6.4 Fahrzeugversuch

Beim Fahrzeugtest wird das reale Steuergerät in der realen Umwelt getestet. Hier werden keine Modelle und keine Koppelebenen mehr benötigt, die nicht im Fahrzeug eingesetzt werden. Die Schnittstellen zum Stimulieren des Testobjekts sind beim Fahrzeugtest die gleichen wie sie ein Fahrer besitzt. Zur Erfassung der Reaktion des Steuergerätes können spezielle Werkzeuge eingesetzt werden. Der Fahrzeugversuch wird beispielsweise in [Mül07] genauer betrachtet.

### 2.6.5 Zusammenfassung der Testsysteme

In MIL, SIL und HIL Testsystem abstrahieren die Modelle die physikalischen Werte (Signale) zu dem Testobjekt. Neben den Signalen sind Ereignisse (Events) eine bedeutende Prozessgröße (vgl. [Har01b], [Hut06], [Mül07]).

Da unterschiedliche Detailierungstiefen bei den drei Testsystemen gebraucht werden, besitzen die Modelle unterschiedliche Verfeinerungsgrade.

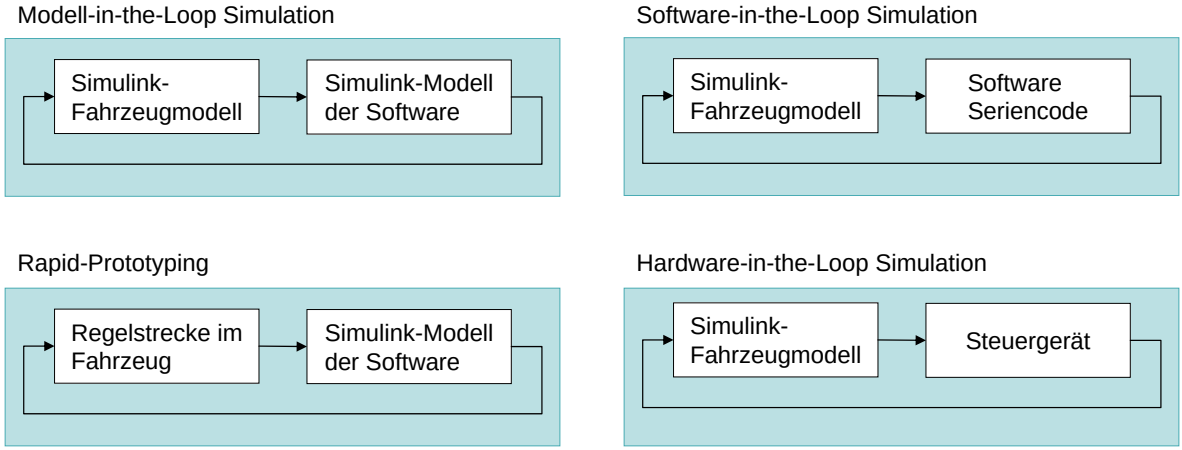



**Abbildung 2.10:** MIL, SIL, Rapid Prototyping und HIL im Überblick

### 2.6.6 Auswirkungen auf den Testfall

Die unterschiedlichen Verfeinerungsgrade bei den Modellen sind ausschlaggebend für die Übertragbarkeit der Testfälle. Je mehr sie voneinander Abweichen (z.B. durch unterschiedliche Blickwinkel und somit unterschiedliche Aspekte bei der Modellierung; vgl. Abschnitt 3.4.1, S. 70 über Sichten und Abstraktion bei der Modellierung), desto geringer ist die Wahrscheinlichkeit, vorhandene Testfälle wiederzuverwenden.

Die Signale der Testobjekte spielen hierbei nur eine Untergeordnete Rolle: Alle Signale werden durch das Testsystem einheitlich durch die Steuerungseinheit dargestellt. Für die relevanten Größen werden in den Modellen Mechanismen bereitgestellt, so dass man bei der Testfallimplementierung auf diese zugreifen kann. Ereignisse sind die zweiten wichtigen Prozessgrößen.

## 2.7 Das Testobjekt

Das Testobjekt, d.h. das Steuergerät wird wie folgt dargestellt: Zuerst erfolgt eine Betrachtung der ein- und ausgehenden Signale des Testobjekts sowie die Betrachtung der Verarbeitung seiner Signale. Daraufhin wird der Zusammenschluss mehrerer Steuergeräte dargestellt. Dabei wird auf die Funktionalität bei diesem Verbund und die Verbindungsmöglichkeiten eingegangen.

Anschliessend erfolgt eine genaue Betrachtung der unterschiedlichen Bereiche im Fahrzeug. Schwerpunkt dabei ist, wie die einzelnen Steuergeräte sich in diesen Bereichen unterscheiden und welche Besonderheiten beim Testen gelten. Zum Schluss erfolgt eine Zusammenfassung der Merkmale der Steuergeräte in den Fahrzeugbereichen.

Aus diesen Merkmalen werden die Auswirkungen auf den Testfall abgeleitet.

### Reaktive Systeme mit Echtzeiteigenschaften im Automobil

**Definition 2.16 (Reaktives System)** *Ein reaktives System setzt Eingabeereignisse (deren zeitliches Auftreten meist nicht vorhergesagt werden kann) - oft unter Einhaltung von Zeitvorgaben - in Ausgabeereignisse um. [BvdBK98]*

Die eingebetteten Systeme im Fahrzeug sind in der Regel reaktive Systeme (siehe Definition eingebettetes System 2.2, S. 16) mit Echtzeiteigenschaften. Solche Systeme werden auch Echtzeitsysteme genannt.

Viele eingebettete Systeme müssen innerhalb einer gegebenen zeitlichen Schranke auf die Umwelt reagieren. So muss beispielsweise das Airbag-Steuergerät innerhalb von 20 ms bis 40 ms nach einer Aufprallerkennung die Zündung für das Luftkissen betätigen.

Wie in [WB05] beschrieben kommt es bei Echtzeitsystemen neben der logischen Korrektheit auch auf die zeitliche Korrektheit der Ergebnisse an. Dabei sind drei Anforderungen genannt: Rechtzeitigkeit, Gleichzeitigkeit und die spontane Reaktion auf Ereignisse. In Abschnitt 3.3.1, S. 67 sind die zeitlichen Aspekte und der Bezug zur Echtzeitfähigkeit detaillierter dargestellt.

So sind die Echtzeitsysteme eine echte Untermenge der reaktiven Systeme und diese wiederum eine echte Untermenge der eingebetteten Systeme wie in Abbildung 2.11 dargestellt.

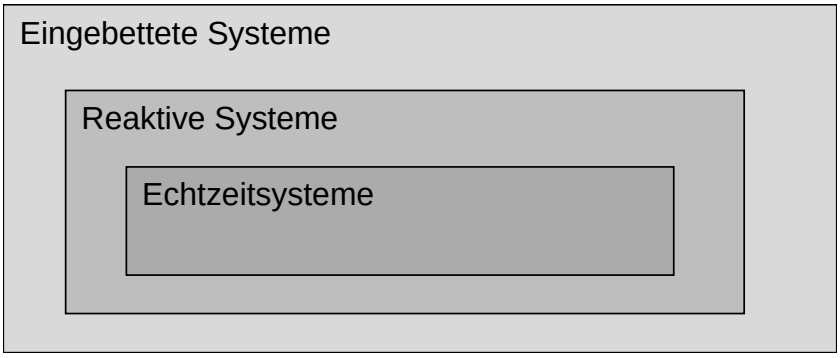



**Abbildung 2.11:** Zuordnung der reaktiven Systeme in die eingebetteten Systeme

#### EVA-Prinzip

Steuergeräte arbeiten nach dem klassischen EVA-Prinzip: Eingabe, Verarbeitung, Ausgabe [SZ04].

### 2.7.1 Verarbeitung

#### Steuerung und Regelung

Es gibt zwei Arten, wie die Eingabe und Ausgabedaten miteinander verwoben sind und Einfluss auf die Verarbeitung bzw. das Verhalten des Steuergerätes nehmen. Hierbei spricht man von einem Prozess.

Definition Prozess nach DIN 6620:

**Definition 2.17 (Prozess)** *Ein Prozess ist die Umformung und / oder Transport von Materie, Energie und / oder Information. Dabei wird zwischen stetigen und diskreten Prozessen unterschieden. (DIN 6620)*

**Steuerung** Bei der Steuerung haben die Ausgabedaten keinen Einfluss auf die Eingabedaten. Die Steuerung arbeitet je nach den Eingabedaten ein definiertes Vorgehen ab. Der Prozess bei der Steuerung läuft diskret ab.

Eine ausführliche Definition liefert die DIN Norm 19 226 [iDuVD94]

**Definition 2.18 (Steuerung)** *Das Steuern, die Steuerung ist ein Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der offene Wirkungsweg oder ein geschlossener Wirkungsweg, bei dem die durch die Eingangsgrößen beeinflussten Ausgangsgrößen nicht fortlaufend und nicht wieder über dieselben Eingangsgrößen auf sich selbst wirken. [iDuVD94]*

**Regelung** Die Regelung führt die Daten der Ausgabe oder die Auswirkungen auf diese zurück in die Eingabedaten. Dabei wird mit einer sogenannten Führungsgröße verglichen: Entsprechen die Eingabedaten und der Wirkung dieser nicht den vorgegebenen bzw. ermittelten Werten, so werden durch die Verarbeitung die Ausgabewerte ermittelt, die einen Einfluss auf die Eingabedaten haben. Beispiel: Die automatische Klimaanlage soll im Auto konstant 20 Grad halten. Der Temperaturfühler dient als Eingabewert. Das Steuergerät für die Klimaanlage setzt je nach Eingabe die Klimaanlagefunktion ein. Das Ergebnis, d.h. die Veränderung der Raumtemperatur, wandert durch den Temperaturfühler wieder zurück zu dem Steuergerät. Der Kreis ist geschlossen, die Regelung wirkt. Die Regelung ist somit ein stetiger Prozess.

Die DIN 19 226 [iDuVD94] definiert die Regelung wie folgt.

**Definition 2.19 (Regelung)** *Das Regeln, die Regelung ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (die zu regelnde Größe), erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst. [iDuVD94]*

### 2.7.2 Eingabe und Ausgabe

Die Ein- und Ausgaben an den Steuergeräten können unterteilt werden in: Sensor- und Aktorverbindungen, die Verbindungen zu den Bussystemen (relevant bei verteilten Funktionen; Definition siehe nächster Abschnitt) und die Stromversorgungsverbindung. Auf die Sensoren und Aktoren sowie die verteilten Funktionen wird im

Folgenden näher eingegangen und die Auswirkungen auf die Testbeschreibung aufgezeigt.

### Sensoren und Aktoren

Die Ein- und Ausgaben können nach Werten unterschieden werden, die sich über die Zeit ändern. Die Betrachtung des Wertes und der Zeit kann diskret oder kontinuierlich erfolgen. Dadurch sind vier mögliche Kombinationen möglich [CS03].

Abbildung 2.12 zeigt graphisch die Kombinationsmöglichkeiten.

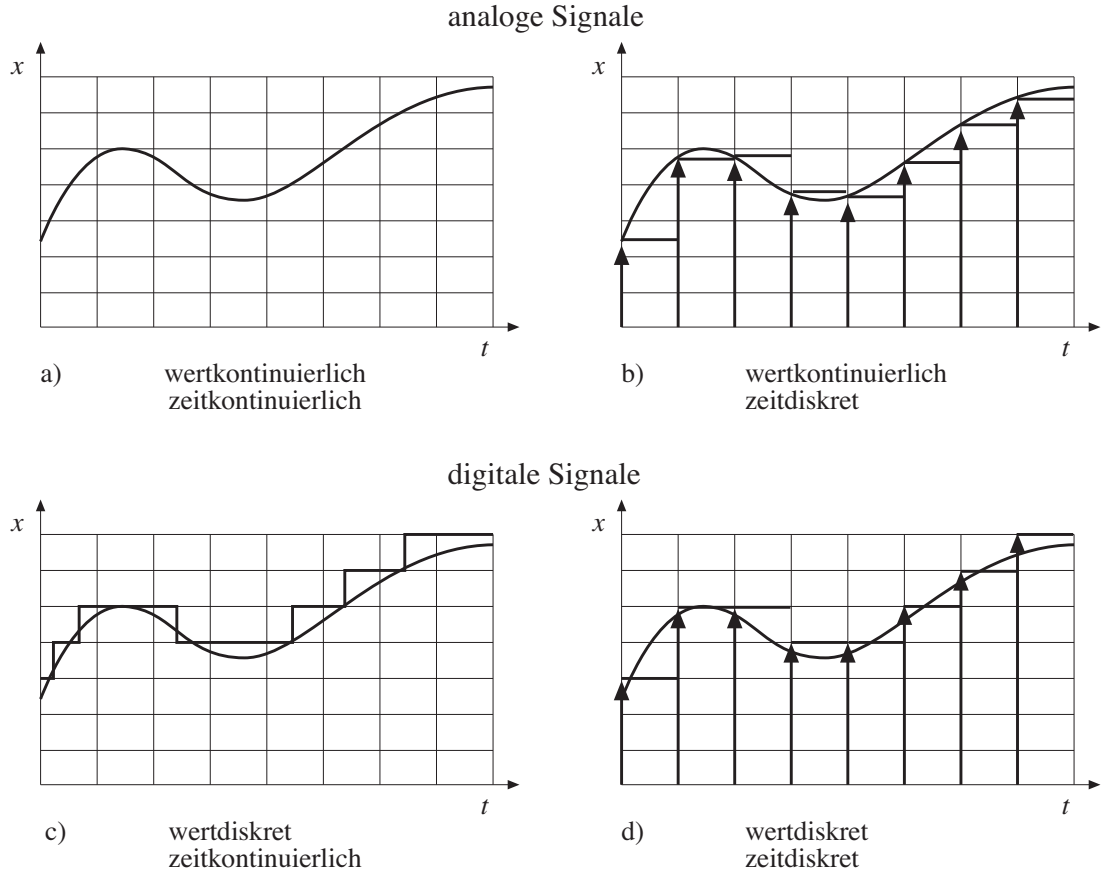



**Abbildung 2.12:** Arten von Signalen bei der Ein- und Ausgabe

Im Automobilbereich werden die Sensor- und Aktorschnittstellen in analoge und digitale Signalübertragungen unterteilt (vgl. [Spi01], siehe auch Abbildungen 2.13 und 2.14, S. 44).

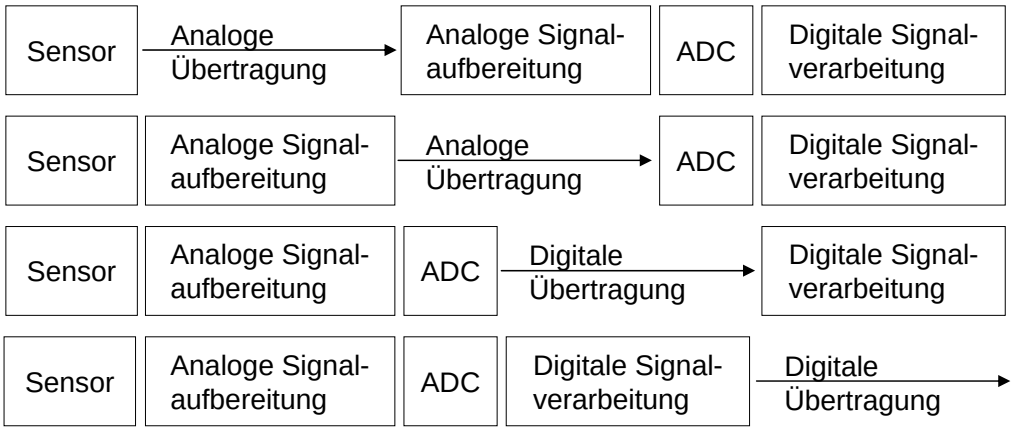



**Abbildung 2.13:** Arten der Signalübertragung bei Sensoren

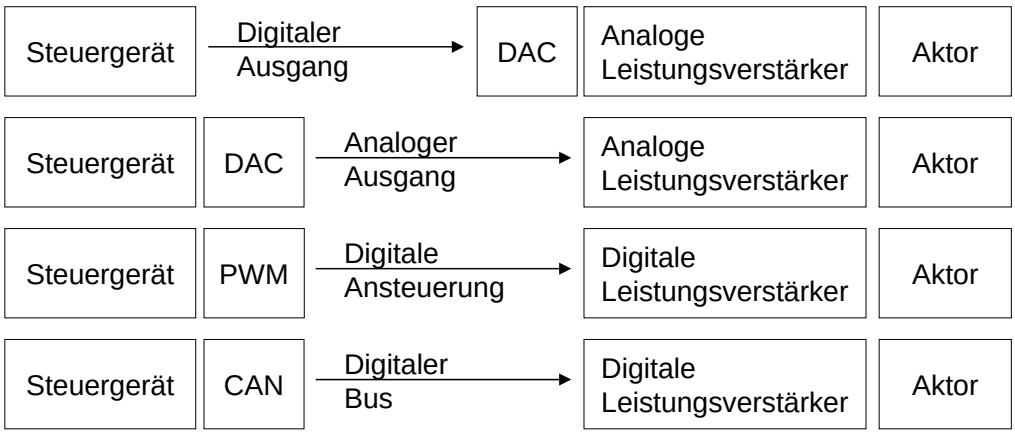



**Abbildung 2.14:** Arten der Signalübertragung bei Aktoren

### Hybride Systeme

Da meistens ein Steuergerät sowohl diskrete als auch kontinuierliche Signale verarbeitet, spricht man von einem hybriden System.

**Definition 2.20 (Hybrides System)** *Systeme, die sowohl kontinuierliche (analoge), als auch diskrete Datenanteile (wert-kontinuierlich) verarbeiten und/oder sowohl über kontinuierliche Zeiträume (zeit-kontinuierlich), als auch zu diskreten Zeitpunkten mit ihrer Umgebung interagieren, heißen hybride Systeme. [Sch07] (vgl. auch [BvdBK98])*

Der analoge (kontinuierliche) Teil solcher Systeme werden basierend auf Differenzialgleichungen erstellt ( [Lam01] , [Sch07]). Die Spezifikation kann durch eine Reihe von graphischen Spezifikationssprachen erfolgen. Beispiele hierfür sind hybride

Petri-Netze, hybride Statecharts und hybride Automaten. In [BvdBK98] werden weitere Sprachen für die Spezifikation solcher Systeme im Automobilen Bereich aufgelistet.

### 2.7.3 Verteilte Funktionen

Die Vernetzung von Steuergeräten ermöglicht Funktionalitäten zu realisieren, die durch das Agieren einzelner Steuergeräte zustande kommen.

Ein Steuergerät beinhaltet dabei den zentralen Algorithmus. Die anderen - an einer Funktion beteiligten - Steuergeräte dienen zur Informationsweiterleitung bzw. -verarbeitung (Beispiel: Information über Blinkerbetätigung wird vom Mantelrohrschaltermodul (MRSM) auf das Bus gelegt,...). Durch die verwendeten Bussysteme im Fahrzeug wie dem CAN und dem Flexray Bussystem bekommt jedes an das Bus angeschlossene Steuergerät die gesendete Nachricht, so dass jedes Steuergerät diese Information verwerten kann. Im Beispiel erfolgt die Verwertung allerdings durch das Steuergerät, welches den zentralen Algorithmus beinhaltet (SAM) um bei Bedarf weitere Steuergeräte über den Bus anzusprechen (Türsteuergerät, Steuergerät für Anhängerkopplung usw.), um angeschlossene Aktoren zu stimulieren (Seitenblinker an der Tür; Anhängerblinker) oder Informationen von weiteren Steuergeräten für die Funktionserfüllung zu fordern.

Somit sind Steuergeräteverbünde spezielle verteilte Systeme, die sich im Algorithmus unterscheiden.

**Definition 2.21 (Verteiltes System)**
*Ein verteiltes System besteht aus Komponenten, die räumlich oder konzeptuell verteilt sind und koordiniert (gekoppelt oder vernetzt) zum Erreichen der Funktionalität des Gesamtsystems beitragen. [BvdBK98]*

Bei einem verteilen Funktion beinhaltet das zentrale Algorithmus ein Steuergerät und die notwendigen Information werden durch die Bussysteme verteilt.

#### Bussysteme

Bussysteme werden im Automobil für die Vernetzung eingesetzt [Lem05].

Abbildung 2.15 zeigt die vergleichende Anordnung der Bussysteme LIN (Local Interconnect Network), CAN (Controller Area Network), Flexray und MOST (Media

Oriented Systems Transport) in der automobilen Technik. Daneben ist der WLAN und - für kurze Strecken - der Bluetooth Standard eingezeichnet, die immer mehr im Fahrzeug eingesetzt werden.

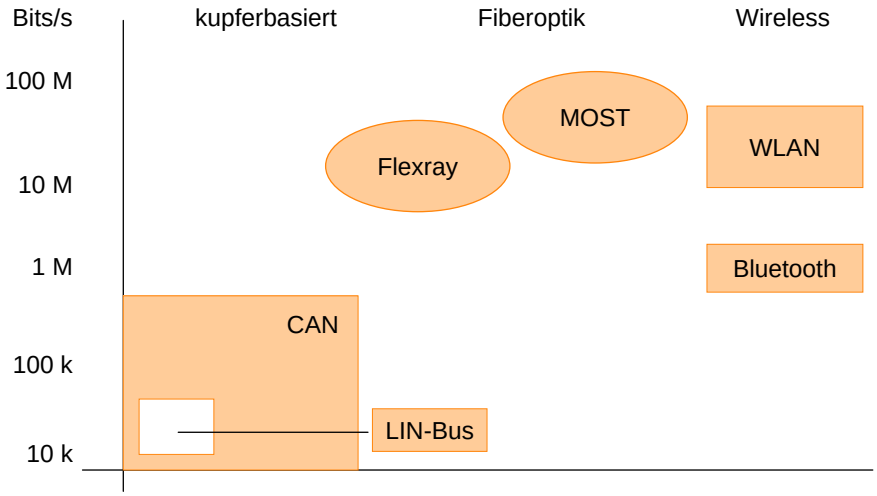



**Abbildung 2.15:** Bussysteme in der Automobil-Technik

Für eine ausführliche Erklärung der Bussysteme wird auf [Lem05] und [WR06] verwiesen.

Der CAN Bus ist das am häufigsten eingesetzte Bussystem in Fahrzeugen. Nichtzuletzt, weil eben dieser Bus für die ersten Schritte in der Vernetzung der Steuergeräte im Fahrzeug konzipiert wurde [Rei06]. Neuere Bus Technologien sind Aufgrund der geringen Bandbreite von bis zu 1MBit/s (Flexray und MOST bieten eindeutig mehr) und der Kosten (kostengünstige Alternative: LIN) des CAN Busses entstanden.

Der LIN-Bus wird dort eingesetzt, wo die Wörter 'kostengünstig' und 'Anbindung' die Gewichtung haben. Der Bus ist hauptsächlich für Anwendungen gedacht, die kein großes Datenaufkommen erzeugen und die in der Regel nur Steuersignale übertragen sollen. Beispiele hierfür sind: Ansteuerung der Fenstermotoren, der Lampen und die Anbindung der Bedieneinheiten an den Türen an das jeweilige Türsteuergerät. Der LIN-Bus kann mit Datenvolumen bis zu 20 kBit/s zurechtkommen.

Der Flexray Bus wurde mit dem zeitgesteuerten Aspekt und dem Sicherheitsaspekt entwickelt. So ist der bevorzugte Einsatzort die X-by-wire Technologie und wird überall dort eingesetzt, wo hohe Datenmengen übertragen werden müssen. Bei großen Datenmengen wird im Infotainment Bereich ein anderer Bus in Betracht gezogen: Der MOST Bus.

Der MOST Bus ist speziell für die Bedürfnisse im Multimedia und Telematik Bereich

entwickelt worden. Der Bus ist auf Übertragungsgeschwindigkeiten bis zu 24 Mbit/s ausgelegt.

### 2.7.4 Domänen im Fahrzeug

In der klassischen Aufteilung wird das Fahrzeug in vier Domänen unterteilt[6]. Für die weitere Betrachtung wird die Aufteilung nach Antriebsstrang (Powertrain), Karosserie (Body), Fahrwerk (Chassis) und Multimedia & Infotainment verwendet ( [Sax08], [SZ04], [Lem05]).

Abbildung 2.16 zeigt einen Verbund von Steuergeräten mit ihren zugehörigen Domänen. Sogenannte Gateway Steuergeräte dienen, neben der Aufgabe Funktionen für die Ausstattungen zu realisieren auch als Kommunikationsglied zwischen den verschiedenen Bereichen. Der Hauptgrund für diese Aufteilung in Bereiche liegt bei den unterschiedlichen Anforderungen der Steuergeräte(-kommunikation) an die Bussysteme. Die Bus Technologien LIN, CAN, MOST und Flexray werden nach ihren Eigenschaften hin in den jeweiligen Bereichen eingesetzt.

Im Folgenden wird ein Überblick über die vier Fahrzeugdomänen im Hinblick auf die Steuergeräte gezeigt. Dabei werden zuerst die Funktionen der Bereiche betrachtet um hieraus die Eigenschaften der Bereiche abzuleiten. Detaillierte Informationen sind in [Rei06], [WR06], [Lem05], [HE07] und [Gev06a] zu finden.

#### Karosserie / Body

Die Karosserie wird in Komfort und passive Sicherheit unterschieden. Unter Komfort werden Fahrzeugzugangssysteme (z.B. Zentralverriegelung, Diebstahlsicherung), Wischer und Regensensor, Sitzverstellung und -heizung, Beleuchtung des Innenraums u.ä. aus dem inneren Bereich bedienbare Systeme eingeordnet.

Systeme der passiven Sicherheit sind z.B. Rückhaltesysteme, Airbagsteuerung und aktive Sicherheitsüberrollbügel [SZ04].

**Verarbeitung - Softwarefunktionen** Die meisten kundenerlebbaren Funktionen befinden sich in dieser Domäne. Im Komfort sind steuernde Funktionen zu finden,

[6] je nach Lektüre gibt es Abweichungen in der Anzahl der Domänen im Fahrzeug, bedingt durch aufkommende Aspekte wie z.B. der Sicherheit oder der Diagnose, aber auch durch die Größe des Fahrzeugs. Hier wird wegen einer allgemeinen Einführung auf die beispielhafte Darstellung geachet, die eine vollständige Abdeckung der im Automobil befindlichen Steuergeräte gewährleiset.

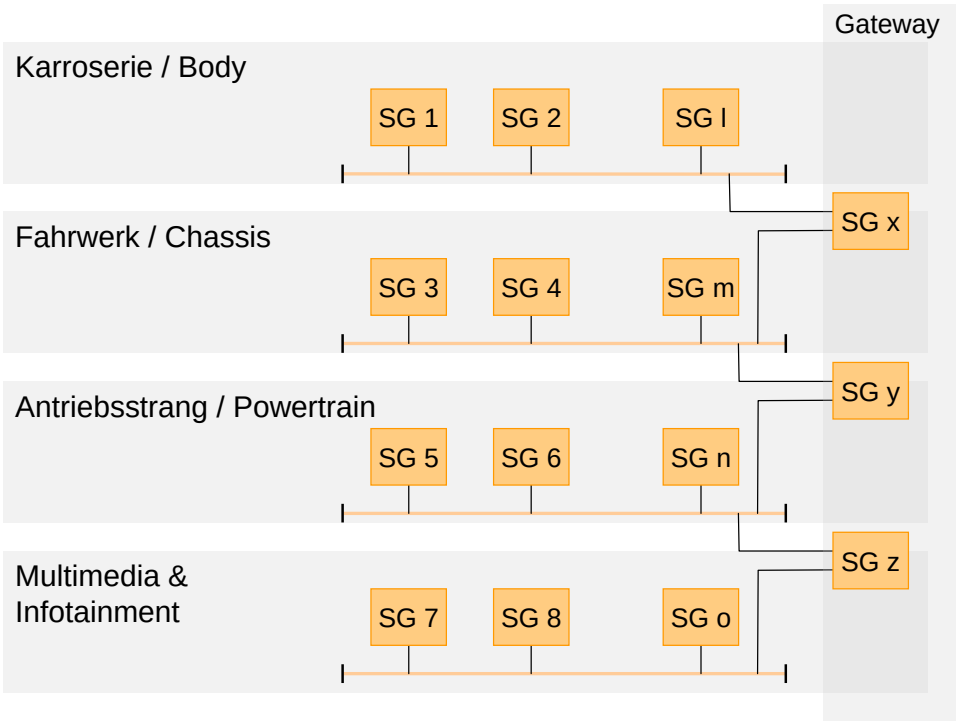



**Abbildung 2.16:** Beispielhafte Verteilung von Steuergeräten in die Subsysteme eines Fahrzeugs

während bei der passiven Sicherheit die Funktionen eine schnelle Verarbeitung und Reaktion garantieren müssen [Lem05].

Bei der passiven Sicherheit nehmen Steuergeräte Aufgaben für den Schutz der Insassen und für die Sicherheit anderer Verkehrsteilnehmer wahr.

**Eigenschaften der Ein- und Ausgabesignale** Die Eingabesignale werden bereitgestellt im Komfort Bereich durch Bedieneinheiten, um Aktoren wie elektronische Motoren (Sitzverstellung, Fenstersteuerung) zu steuern. Steuergeräte übergreifende Funktionen sind geringer als in den anderen Fahrzeugdomänen. Der Hauptgrund liegt darin, dass die Komfortfunktionen über das Fahrzeug hinweg verteilt sind, d.h. die einzelnen Steuergeräte vor Ort ihre Aufgaben erfüllen (direkter Anschluss an Sensoren und Aktoren).

Erforderlich sind geringe bis sehr kurze Reaktionszeiten bei niedrigen bis mittleren Datenraten.

**Testen von Steuergeräten** Die Bedieneinheiten zur Eingabe werden bevorzugt simuliert. Die Aktoren können entweder durch Echtlasten (den elektronischen Motoren) oder - aufwendiger - durch Modellierungen abgebildet werden.

### Fahrwerk / Chassis

Das Fahrwerk umfasst alle Elemente, die das Fahrzeug mit der Fahrbahn verbindet [HE07]. Bei der Zuordnung der Steuergeräte werden hier zusätzlich auch jene eingeschlossen, die für das Führen eines Fahrzeugs verantwortlich sind, d.h. die Servolenkung wird als ein Teil des Fahrwerks betrachtet. Mit Fahrdynamik und Fahrkomfort ist das Fahrwerk in zwei Gruppen geteilt. Fahrerassistenzsysteme wie ESP (Elektronisches Stabilitatsprogramm) oder ABS (Antiblockiersystem) fallen unter die Fahrdynamik.

**Verarbeitung - Softwarefunktionen** Lenkrad, Gaspedal und Bremspedal sind die Schnittstellen zum Fahrer. Durch die Sicherheitsanforderungen sind diese Schnittstellen zu anderen Systemen gering gehalten.

Die Zusammenfassung der einzelnen Bedienelemente Lenkrad, Gaspedal und Bremspedal und ihre Wirkung auf die Lenkung, den Antrieb bzw. die Bremse wird durch Systeme wie dem Global Chassis Control (GCC) erreicht [Lem05].

Die Funktionen betrachten die Soll-Reaktion (gewünschte Lenkung, Bremsung oder Gaszufuhr), um die Ist-Reaktion regelnd anzupassen.

**Eigenschaften der Ein- und Ausgabesignale** Die Signale erfassen die auftretenden Kräfte während der Fahrt. Als Aktoren werden die Systeme Antrieb, Bremse und Lenkung benutzt. Eine kontinuierliche Signalerfassung und Auswertung ist die Folge.

**Testen von Steuergeräten** In der Regel werden kontinuierliche Signale für die Stimulation verwendet. Das Ist-Ergebnis wird entweder punktuell oder kontinuierlich erfasst. Der Vergleich mit dem Soll-Ergebnis erfolgt demnach punktuell (diskrete Zeit- und Werteinheit) oder in einem Toleranzbereich vom Soll-Ergebnis.

| **Motorsteuerung** | **Fahrverhalten und Komfort** |
|---|---|
| • Einspritzmenge<br>• Zündung (Otto)<br>• Spritzbeginn (Diesel)<br>• Füllung (Otto)<br>• Nockenwellenregelung (Otto)<br>• Leerlaufregelung<br>• Abregeldrehzahl<br>• Lambda-Regelung (Otto)<br>• Abgasrückführung<br>• Ladedruckregelung<br>• Abgasnachbehandlung | • Fahrerwunsch<br>• Lastschlagdämpfung<br>• Geschwindigkeitsregelung<br>• Geschwindigkeitsbegrenzung<br>• Klimaanlage<br>• Zusatzheizung<br>• Getriebebeeinflussung<br>• Momentenreduktion bei Getriebeschaltvorgängen |
| **Diagnose/Sicherheit** | **Schnittstellen** |
| • Diagnose elektrischer Signale<br>• Erkennung abgasrelevanter Fehler<br>• Plausibilitätsprüfungen<br>• Fehlercode-Management | • Kommunikation mit anderen Fahrzeugsteuergeräten<br>• Kommunikation mit Werkstatt-Tester<br>• Bus-System |

**Abbildung 2.17:** Liste von Funktionen bei der Motorsteuerung [Gev06b]

### Antriebsstrang / Powertrain

Motor und Getriebe werden zum Antriebsstrang (oder Triebstrang genannt) zusammengefasst [Gev06b], [Lem05].

Das Motorsteuergerät dient der Einhaltung der gesetzlichen Vorgaben wie Abgaswerte und die Regelung des Motors aufgrund unterschiedlicher Vorgaben (Fahrverhalten, Kraftstoffverbrauch, Leistungs- und Drehmomentcharakteristik). Neben den eigentlichen Aufgaben werden zusätzliche Funktionen durch das Motorsteuergerät erfüllt. Abbildung 2.17 zeigt mögliche Funktionen eines Motorsteuergerätes neben den Hauptaufgaben.

Die typischen Funktionen für eine Getriebesteuerung sind in Abbildung 2.18 aufgezeigt.

**Verarbeitung - Softwarefunktionen** Die Funktionen basieren sehr stark auf Kennwerten, Kennlinien und Kennfeldern, die für die Varianten von Motor, Getriebe und Fahrzeug kennzeichnend sind [SZ04]. Die vorhandene hohe Anzahl der Funktionen

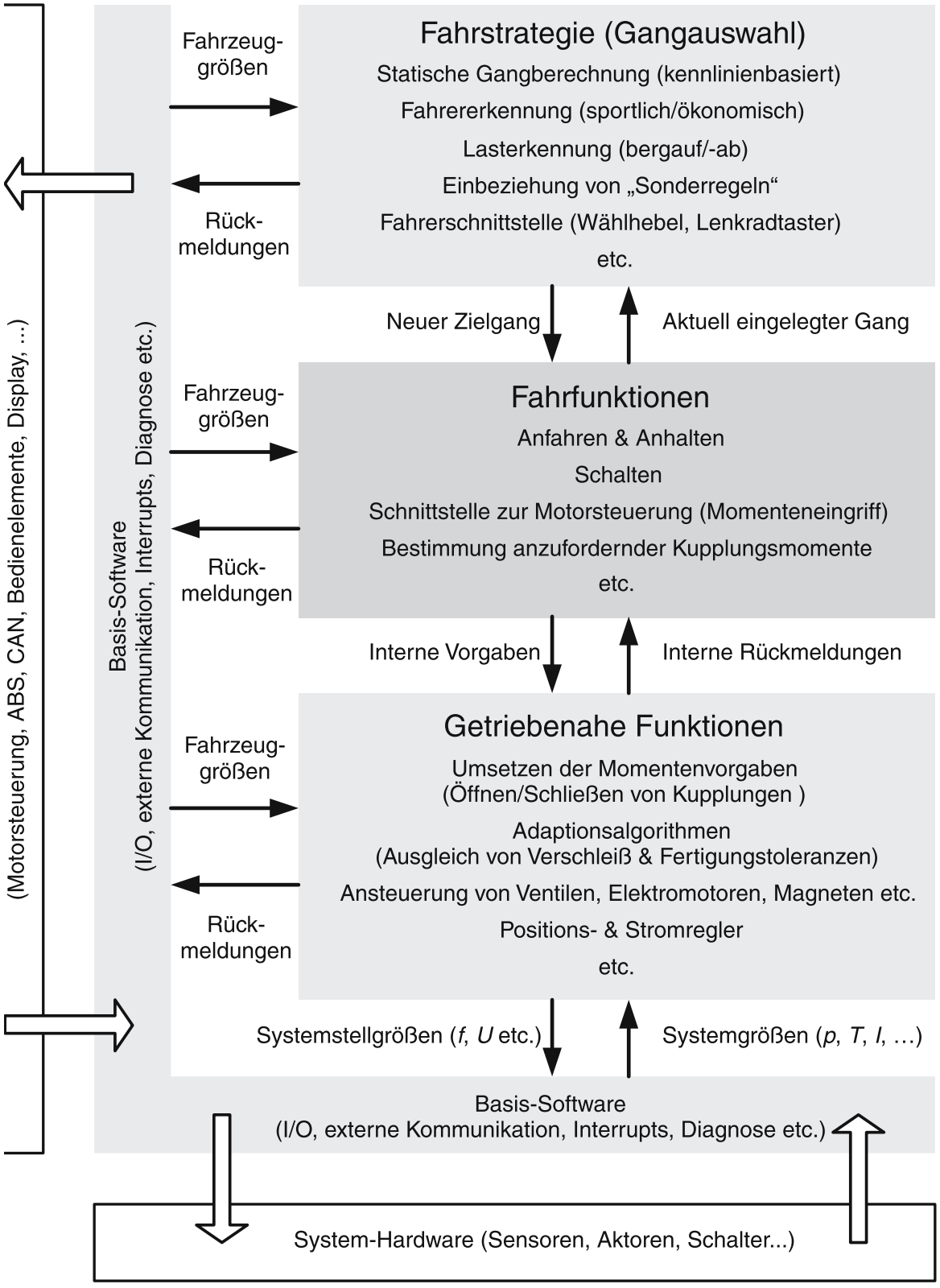



**Abbildung 2.18:** Typische Aufgaben einer Getriebefunktion [Lem05]

interagiert zusammen und stark reaktiv, mit harten Echtzeitanforderungen.

**Eigenschaften der Ein-und Ausgabesignale** Abbildung 2.19 zeigt mögliche Schnittstellen zum Motorsteuergerät.

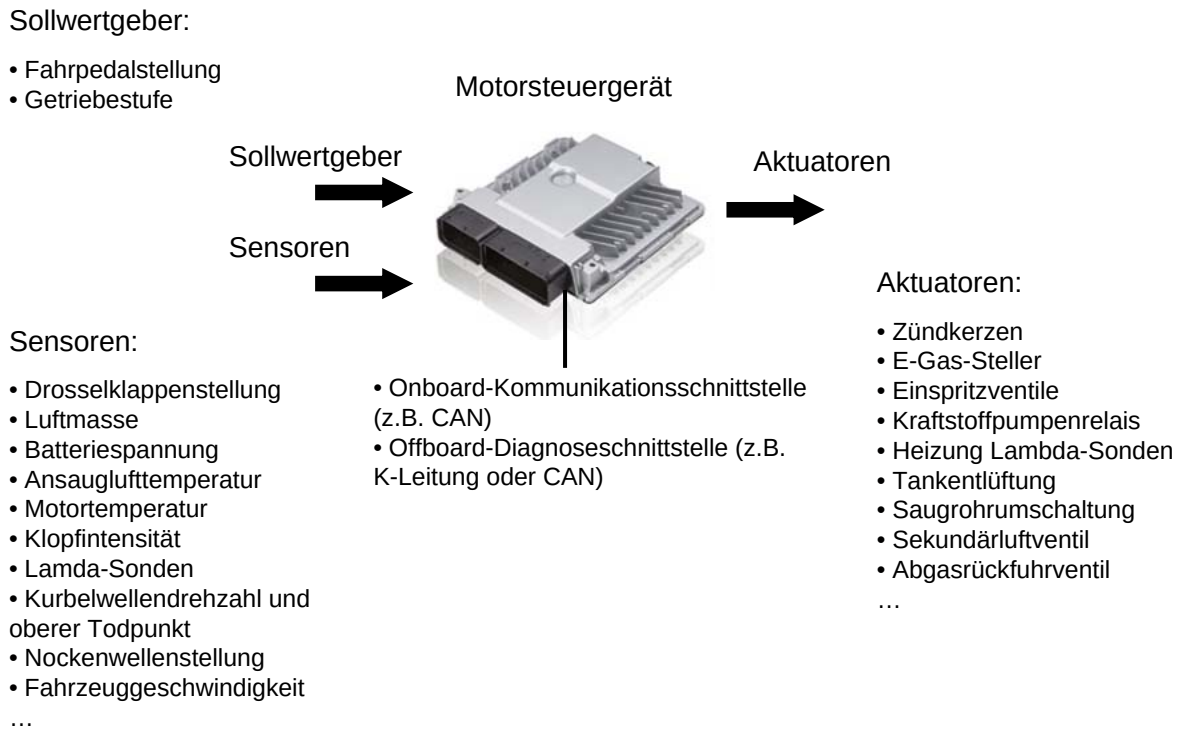



**Abbildung 2.19:** Schnittstellen eines Motorsteuergeräts für Ottomotoren [SZ04]

Signale wie für die Nockenwellenstellung oder die Kurbenwellendrehzahl sind kontinuierlich. Daneben existieren auch digitale Signale wie für die Temperatur oder Batteriespannung.

**Testen von Steuergeräten** Zeitliche Aspekte und kontinuierliche Signale stehen beim Testen von Steuergeräten im Bereich des Anstriebsstrang im Vordergrund. Diskrete Signale treten aufgrund der zu erfüllenden zusätzlichen Funktionen ebenfalls auf. Die Stimulation und der Vergleich des Ist-Ergebnisses mit dem Soll-Ergebnis entspricht dem des Fahrwerkbereiches.

#### Multimedia & Infotainment

Die Domäne Multimedia & Infotainment umfasst eine Reihe von Funktionen, die kundenerlebbar sind. Abbildung 2.20 zeigt die Aspekte im Infotainment Bereich.

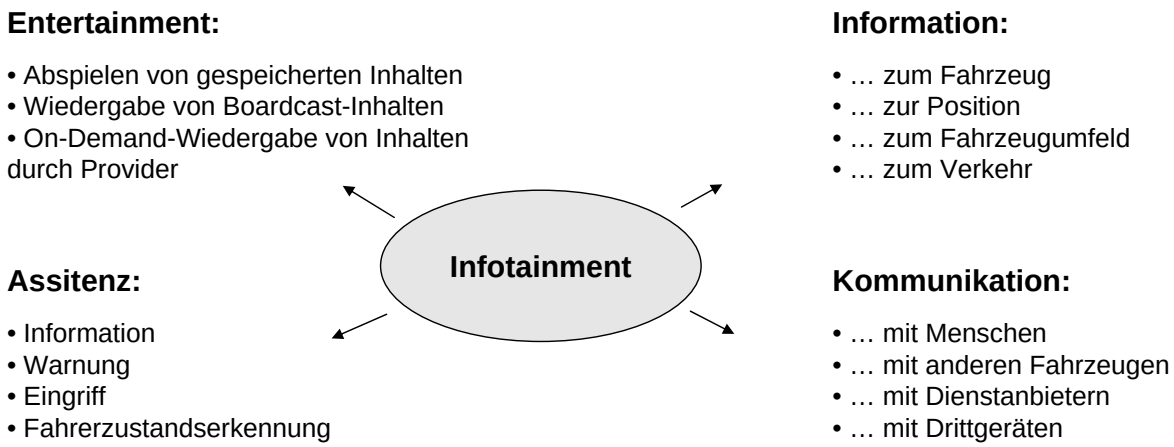



**Abbildung 2.20:** Aspekte des Infotainment im Fahrzeug [WR06]

Die Fahrzeugassistenzsysteme werden auch unter dieser Domäne aufgeführt, da ihr Ziel neben der Sicherheit auch die direkte Benachrichtigung des Fahrers ist.

**Verarbeitung - Softwarefunktionen** Ausgehend von den vier Aspekten sind folgende Funktionen u.a. im Infotainment Bereich vorhanden.

- Entertainment

  Neben dem Radio sind Video und internetbasierte Anwendungen als Entertainment zu finden. Das Radio wird zunehmend durch neue Technologien ergänzt bzw. erweitert (Beispiel: MP3).

- Information

  Routenführung, Traffic Message Channel (TMC) mit aktuellen Verkehrsinfomationen und Location based Services (z.B. nächste Tankstelle) werden für den Fahrer zur Verfügung gestellt. Aus der Fahrzeugumfelderkennung werden Informationen zum Fahrzeugzustand bereitgestellt.

- Assistenz

  Unter Assistenz sind alle Funktionen, die den Fahrer aktiv unterstützen, zu finden. Das Spurhaltesystem, Unfallvermeidung und Unfallfolgenvermeidung, Anfahr-, Kurvenfahr- und Bremsunterstützung, Fahrzeugzustandserkennung sind Teil der Unterstützung des Fahrers. Als Assistenz gelten in diesem Bereich auch das ACC (Adaptive Cruise Control), ABS (Antiblockiersystem) und ESP

(Elektronisches Stabilitätsprogramm), die in der vorherigen Domäne (Fahrwerk / Chassis) ebenfalls platziert wurden.

- Kommunikation

  Als Kommunikation wird hauptsächlich die Möglichkeit zur Kommunikation der Insassen mit dem Umfeld genannt. Daneben sind aber auch aufkommende Techniken wie die Kommunikation zwischen Fahrzeugen enthalten oder die Kommunikation des Fahrzeugs bei der Wartung. Hier sind Kommunikationstechniken von Bluetooth bis zu LAN- und WAN-Verbindungen möglich.

Die Herausforderungen in dieser Fahrzeugdomäne sind vor allem durch die stark kundenerlebbaren Funktionen, die visuell stattfinden, begründet. Gleichzeitig müssen Zuverlässigkeits- und Sicherheitsaspekte im Fahrzeug beachtet werden.

**Eigenschaften der Ein- und Ausgabesignale** Der Einsatz von Kamerasystemen und Radar als Sensoren erfolgt auf Signalverläufen, die eine hohe Informationsdichte besitzen. Ebenfalls sind die Aktoren visuelle Darstellungen für die Insassen (z.B. Monitor). Die Fahrerassistenzssysteme haben ein Einfluss auf Teile aus anderen Domänen (z.B. Antrieb, Bremse, Lenkung) oder benötigen Informationen zur Auswertung des Fahrzeugzustandes aus anderen Domänen. Insgesamt werden die vorhandenen Informationen aus dem Fahrzeug verwertet, d.h. eine enge Kopplung an die Informationen von vorhandenen Steuergeräten existiert.

**Testen von Steuergeräten** Das Testen in diesem Bereich ähnelt nicht den anderen Bereichen. Vor allem die Sensoren und Aktoren sind komplexer (Kamerasysteme, Radar, Bildschirm) als die Sensoren und Aktoren der anderen Bereiche.

Die Testsysteme müssen entsprechend angepasst werden um die neuartigen Sensoren und Aktoren ankoppeln zu können. Die Reduzierung der Betrachtung auf ein- und ausgehende Signale bei der Testfallbeschreibung vereinfacht jedoch die Zuordnung beim Testen: So sind die Signale kontinuierlich und aus dieser Betrachtung heraus kann das Testen dieser Steuergeräte den beiden vorangehenden Bereichen zugeordnet werden.

### 2.7.5 Merkmale von Steuergeräten bei den Fahrzeugdomänen

In der Tabelle 2.1 ist ein Vergleich der Steuergeräteeigenschaften aus den Fahrzeugdomänen nach ihrer Verarbeitung, Eingabesignalen (Sensoren), Ausgabesignalen (Aktoren), dem Grad der Vernetzung und der Echtzeitanforderungen dargestellt.

| Eigenschaften | Antriebsstrang | Karosserie | Fahrwerk | Mult. & Infotain. |
|---|---|---|---|---|
| Verarbeitung | Regelung | Steuerung, Regelung | Regelung | trans., interaktiv, reaktiv |
| Sensoren | kont., diskret | diskret | kont. | kont., digital und stark komplex |
| Aktoren | kont. | diskret | kont. | kont., digital und stark komplex |
| Vernetzung | gering | gering bis mittel | mittel bis stark | stark vernetzt |
| Echtzeitanf. | harte Echtzeit | gerine Zeitanf. | harte Echtzeit | geringe bis harte |

**Tabelle 2.1:** Vergleich der Fahrzeugdomänen

## Legende

Die Verarbeitung der Eingangssignale wird in transformationell (trans.), interaktiv und reaktiv unterteilt. Die reaktiven Systeme werden sofern sinnvoll in Steuerung und Regelung unterteilt.

Die Sensoren und Aktoren können diskreten und kontinuierlichen Charakter haben. Bei Sensoren und Aktoren wo Sensoren wie eine Kamera oder Radar und Aktoren wie ein Display benutzt werden, wird zusätzlich als 'stark komplex' angegeben.

Die Vernetzung ist aus der Sicht der Steuergeräte und beantwortet die Frage, ob das typische Steuergerät in dieser Fahrzeugdomäne für die Erfüllung seiner Funktion andere Steuergeräte oder Informationen aus anderen Steuergeräten braucht.

Bei den Echtzeitanforderungen wird die zeitliche Anforderung bei der Verarbeitung betrachtet und beantwortet die Frage, ob die Reaktion harten oder weichen Echtzeitanforderungen genügen muss oder ob die Zeiten geringe bis gar keine Rolle spielen.

## Schlussfolgerung

Die Ergebnisse im Vergleich sind relativ zu sehen, d.h. sie stellen nur ein Schwerpunkt der Steuergeräte in der jeweiligen Fahrzeugdomäne dar. So kann auf die unterschiedlichen Domänen keine konkrete Aufteilung vorgenommen werden, da die

zu erfüllenden Funktionen, sowie die Eingabe- und Ausgabesignale die breite Palette der möglichen auftretenden Arten der Eigenschaften abdecken. Beispiel: Das Motorsteuergerät übernimmt neben seiner eigentlichen Aufgabe, die Koordination des Motors, auch zusätzliche Aufgaben wie in der Abbildung 2.17, S. 50 zu sehen (z.B. im Bereich Komfort), so dass auch diskrete Signale von Sensoren steuernd verarbeitet werden.

Ebenso befinden sich im Bereich Karroserie Funktionen, die einen stark regelnden Charakter haben, wie z.B. die automatische Wischerfunktion, die auf dem Regen/-Licht Sensor beruht und je nach Stärke des Regens in unterschiedlichen Geschwindigkeiten die Wischer steuernd regelt.

Insgesamt ist bei neuartigen Funktionen (wie z.B. Fahrerassistenzssysteme und Infotainment) zu sehen, dass sie häufig Domänenübergreifend agieren und dadurch eine Zuordnung zu einem der vier Fahrzeugbereiche erschwert.

## 2.8 Zusammenfassung

### Systeme und Komponenten

Der Systembegriff wird nach der technischen Umgebung und der Verarbeitung unterschieden.

Aus Sicht des Fahrzeugs kann beispielsweise von einem System gesprochen werden, wenn dieses eine Ausstattung abbildet und durch elektrische und mechanische Komponenten realisiert wird.

Eine elektrische Komponente, ein eingebettetes System, wird im Fahrzeug Steuergerät genannt. Hier sind unterschiedliche Verarbeitungen der eingehenden Informationen vorhanden. Steuergeräte im Automobil sind zum größten Teil reaktive Systeme mit Echtzeiteigenschaften.

Bei der Betrachtung aus der Sicht eines eingebetteten Systems wird das Steuergerät ein System und die Bestandteile des Steuergeräts sind die Komponenten.

## Prozesse

Das Gesamtsystem wird bei der Entwicklung in Teilsysteme und diese wiederum in Komponenten aufgebrochen. Jeder der Teile wird durch unterschiedliche Gruppen entwickelt und entwicklungsbegleitend getestet. Die Entwicklungsprozesse der beteiligten Gruppen sind ineinander verflochten.

Der entwicklungsbegleitende Testprozess besteht aus Aktivitäten, die das Ziel haben, funktionale Testfälle durchzuführen, d.h. Testfälle zu erstellen, die die Anforderungen aus der System-Spezifikation überprüfen. Die Testfälle sind typischerweise von der System-Spezifikation, der Testumgebung und dem Testobjekt abhängig.

## Testumgebung

Die Testumgebung ist auf die Steuereinheit reduzierbar, in denen die Testfälle ausgeführt werden. Auf dieser Ebene wird den Testfällen alle notwendigen Signale einheitlich zur Verfügung gestellt. Ereignisse bilden die zweite wichtige Prozessgröße.

Die unterschiedlichen Testsysteme unterscheiden sich durch die Verfeinerungsgrade der Umgebungsmodelle.

## Steuergerät

Steuergeräte sind reaktive Systeme mit Echtzeitverhalten. Sie besitzen diskrete (Steuerung) und stetige (Regelung) Verarbeitung und analoge und digitale Ein- und Ausgänge. Durch die verteilten Funktionen werden Funktionalitäten durch mehrere Steuergeräte verwirklicht. Zusammengefasst haben folgende Aspekte einen Einfluss auf das Testen:

- Analoge und digitale Signalbeschreibung

  Die eingehenden und ausgehenden Signale bei Steuergeräten sind entweder analog oder digital. So muss bei einem Testfall die Stimulation oder die Auswertung beide Arten erzeugt werden, d.h. die Beschreibung muss beide Arten unterstützen.

- Diskrete und stetige Verarbeitung

  Steuergeräte besitzen sowohl eine diskrete als auch eine stetige Verarbeitung. Dabei kann ein Steuergerät sowohl diskret als auch stetig fungieren. Mit der

Verarbeitung sowohl analoger und digitaler Signalarten werden Steuergeräte zu den hybriden Systemen gezählt.

(Im nächsten Kapitel (Kapitel 3, S. 59) wird dieser Aspekt genauer betrachtet.)

- Verteilte Funktionen

  Eine Besonderheit bei Steuergeräten ist die Funktionsrealisierung mittels mehrerer Steuergeräte. Dabei werden Nachrichten durch Bussysteme zwischen den Steuergeräten transportiert, um diese verteilten Funktionen zu realisieren. Der zentrale Algorithmus ist bei einem Steuergerät vorhanden.

  Die Nachrichten werden durch Testsysteme einheitlich dargestellt, so dass die Umsetzung der Testfälle auf diese zugreifen kann.

  Auf der Ebene der Beschreibung muss ein Testfall den Austausch von Nachrichten definieren können.

- Echtzeiteigenschaft

  Die zeitlichen Aspekte reaktiver Systeme müssen beschrieben werden können. Dieser Aspekt wird ebenfalls im nächsten Kapitel genauer betrachtet.

Weitere Betrachtungen sind aus dem Umfeld des Testfalls notwendig: Wie wirkt sich die System-Spezifikation und die Testumgebung (indirekt über die Testfallimplementierung) auf den Testfall aus?

Diese und obige Fragen werden im folgenden Kapitel beantwortet.

# Kapitel 3

# Inhalt und Umfang der Testfälle für den funktionalen Steuergerätetest

## 3.1 Ein- und Abgrenzung des Kapitelinhalts

Ziel dieses Kapitels ist es den Inhalt und Umfang eines Testfalls zu definieren[1]. Die Definition erfolgt durch die Ein- bzw. Abgrenzung eines Testfalls auf das funktionale Testen von Steuergeräten im Automobilen Umfeld.

Die Definition des Inhalts sollen über die Eigenschaften eines Testfalls und seine Beziehung zur Spezifikation des Testobjekts, der Testfallimplementierung und des Testprozesses (vgl. vorheriges Kapitel, S. 13) erfolgen. Die Eigenschaften werden dabei durch die benötigten Sprachelemente festgelegt.

Bei der Definition des Umfangs eines Testfalls wird der Ablauf der Testfällebetrachtet, d.h. die Frage wird beantwortet, welche Ablaufstrukturen notwendig sind, um funktionale Tests von Steuergeräten durchführen zu können.

[1] Das Vorgehen ist angelehnt an die Begriffsdefinition aus dem Bereich der Terminologie (vgl. [FB89], [BB79]):

Die Begriffsdefinition wird durch zwei Aspekte festgelegt: Dem Begriffsinhalt und dem Begriffsumfang.

Der Begriffsinhalt „ist die Gesamtheit der Merkmale dieses Begriffs“ [FB89]. Merkmale dienen der Beschreibung, der Abgrenzung oder der (Ein-)Ordnung des Begriffs. Zwei Merkmalgruppen werden betrachtet: Beschaffenheitsmerkmale und Beziehungsmerkmale. Erstere definieren die Eigenschaften, letztere die Beziehungen zu umliegenden Gegenständen. Dabei kann ein Gegenstand ein materielles Ding oder Objekt oder Erkenntnis sein.

Der Begriffsumfang ist „die Gesamtheit aller Unterbegriffe, die auf der selben Abstraktionsstufe stehen“ [FB89].

### Vorgehen

Der erste Abschnitt gibt eine Übersicht über die Beziehungen von Testfällen zur Spezifikation des Testobjekts, der Testfallimplementierung und dem Testprozess.

Die Betrachtung dieser Beziehungen erfolgt auf einer Bottom-Up und einer Top-Down Analyse:
Bei der Bottom-Up Analyse wird die Testfallimplementierung untersucht. Anschliessend werden die hieraus ergebenden Auswirkungen auf einen Testfall aufgezeigt.
Bei der Top-Down Analyse wird die System-Spezifikation untersucht und ebenfalls sich ergebende Auswirkungen auf einen Testfall aufgezeigt. Zu dieser Analyse gehört auch die Untersuchung des Testprozesses.

Das bedeutet: Im ersten Schritt werden zuerst Inhalt, Umfang und Zweck einer Testfallimplementierung betrachtet.

Anschließend werden Inhalt und Umfang der Spezifikation des Testobjekts dargestellt. In diesem Abschnitt wird auch die Testfallherleitung im Testprozess betrachtet, die die Spezifikation des Testobjekts als Testbasis her nimmt und somit die Verknüpfung zwischen System-Spezifikation und Testfall darstellt.

Nach dieser Betrachtung wird der Inhalt eines Testfalls z.B. die benötigten Sprachelemente definiert. Im Anschluss daran wird der Umfang festgelegt. So werden die Ablaufstrukturen identifiziert, die die Testfälle besitzen.

## 3.2 Die Spezifikation, die Testfallspezifikation und die Testfallimplementierung

Der Testfall ist definiert als:

**Definition 3.1 (Testfall)** *Umfasst folgende Angaben: die für die Ausführung notwendigen Vorbedingungen, die Menge der Eingabewerte (ein Eingabewert je Parameter des Testobjekts) und die Menge der erwarteten Sollwerte, die Prüfanweisung (wie Eingaben an das Testobjekt übergeben und Sollwerte abzulesen sind) sowie die erwarteten Nachbedingungen. [Iee91], [IST07]*

Bei dieser Definition fehlen die Einzelheiten, wie z.B. welche Eigenschaft die Prüfanweisung hat oder wie die Eingabewerte und die erwarteten Sollwerte in Beziehung stehen. Am Ende dieses Kapitels werden diese Details beantwortet.

Im Folgenden wird eine Übersicht über die Beziehung bei der Testfallerstellung der drei Dokumente zueinander gezeigt. Danach werden die Begriffe im Testumfeld definiert und u.a. die Testfallspezifikation hier eingeordnet.

### 3.2.1 Übersicht über die Beziehung der Dokumente zueinander

Es gibt drei Hauptdokumente. Sie bilden drei Ebenen ab, wie in Abbildung 3.1 zu sehen.

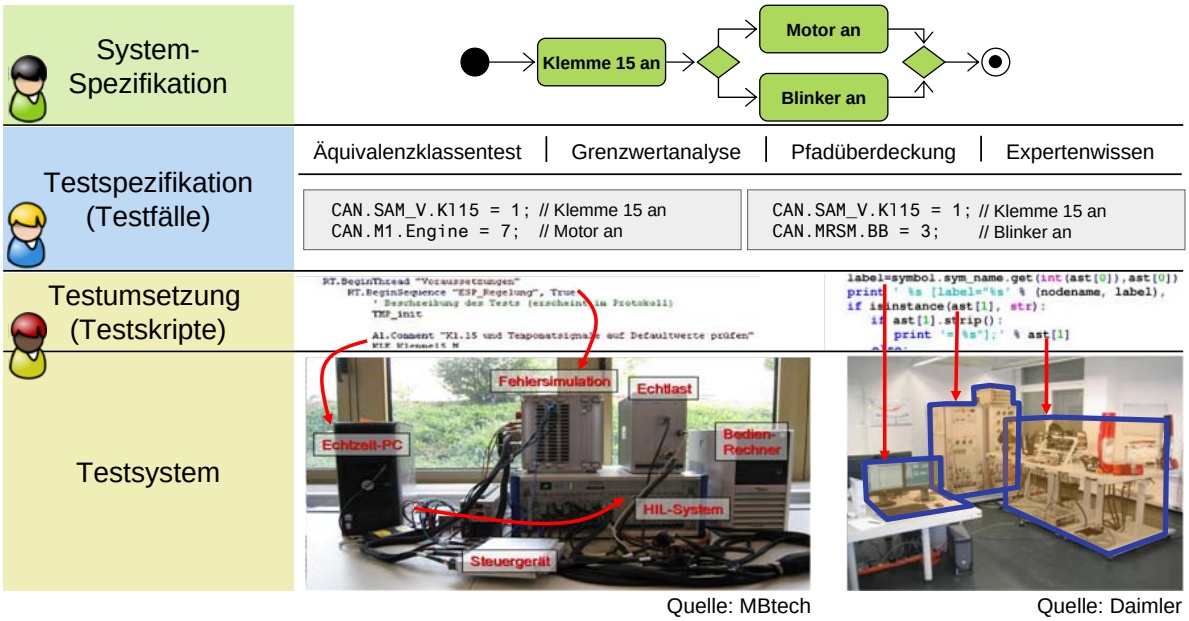



**Abbildung 3.1:** Von der Spezifikation zur Testfallimplementierung

In der System-Spezifikation werden Anforderungen - das geforderte Verhalten - an das Testobjekt festgehalten. Die Anforderungen sind typischerweise in Prosa verfasst oder durch geeignete - meist graphische - Spezifikationssprachen modelliert.

Ausgehend von der System-Spezifikation wird durch geeignete Testverfahren die Testfallspezifikation erstellt. Diese kann wiederum in Prosa verfasst oder - wie die Spezifikation - durch geeignete Spezifikationssprachen modelliert sein[2]. Die Testfallspezifikation beinhaltet keine implementierungstechnischen Details und ist somit Testsystem ungebunden.

[2]In den folgenden Abschnitten wird zu sehen sein, dass die Spezifikationsmittel für die Testfallspezifikation auf den Spezifikationsmitteln der Anforderungsdefinition basieren und durch eine Testsemantik adaptiert werden.

Aus der Testfallspezifikation werden die Testfallimplementierungen für die einzelnen Software- (SW) und Hardware (HW) Plattformen erstellt. Die Implementierungen sind an die Schnittstellen der SW/HW Ebene gebunden, so dass sie nicht austauschbar sind.

Offen ist die Frage, wie eine Testfallspezifikation im Steuergerätetest aussieht, d.h. welchen Inhalt und Umfang diese haben soll. Hierfür ist eine Abgrenzung zu den Inhalten der Spezifikation des Testobjekts und zu der Testfallimplementierung nötig.

### 3.2.2 Definition der Begrifflichkeiten im Testumfeld

Um eine gemeinsame Basis für die Begriffe zu schaffen und die Arbeit einordnen zu können ist eine Begriffsdefinition der Dokumente sinnvoll. Hierbei wird als Grundlage der IEEE Standard 829 (Standard for Software Test Documentation) verwendet. Dieser wird durch die Begriffsdefinition von [Sax08] und [IST07] ergänzt bzw. erweitert.

#### Die Definiton der Spezifikation

In dem IEEE Standard 829 wird der Begriff „Spezifikation“ nicht definiert, da sich dieser außerhalb der Testdokumentation befindet[3]. Hierfür wird - wie im Kapitel 2 eingeführt - der Begriff inhaltlich wie folgt festgelegt:

**Definition 3.2 (Spezifikation)** *(1) Die Spezifikation beschreibt das geforderte Verhalten [Lig02].*
*(2) Ein Dokument, das die Anforderungen, den Aufbau, das Verhalten oder andere Charakteristika des Systems bzw. der Komponente beschreibt, idealerweise genau, vollständig, konkret und nachprüfbar. Häufig enthält die Spezifikation auch Vorgaben zur Prüfung der Anforderungen. [IST07].*

Die Definition vervollständigt die Definition der System-Spezifikation in Abschnitt 2.3, S. 20 aus der inhaltlichen Sicht. So ist im folgenden mit einer Spezifikation eine System-Spezifikation aus dem vorherigen Kapitel gemeint.
In der Automobilindustrie basieren diese Spezifikationen in der Regel auf der natürlichen Sprache [WRDZ07], die mit modellbasierten Spezifikationsmitteln kombiniert werden [CFGK05], [GS05].

[3] Die Definition des Begriffes ist im IEEE Standard 610 (Standard glossary of software engineering terminology) festgelegt, das auch [IST07] als Grundlage verwendet und hier benutzt wird.

### Zuordnung der Testfallbeschreibung in die Dokumentenlandschaft des IEEE 829

Die Testdokumentationen ist nach IEEE 829 in zwei Abschnitte unterteilt (siehe Abbildung 3.2): Dokumente, die in der Planungsphase erstellt werden und Dokumente, die in der Testdurchführung erstellt bzw. generiert werden. Somit trennt IEEE 829 die Inhalte beim Softwaretest in unterschiedliche - logisch abgegrenzte - Dokumente. Die einzelnen Phasen mit ihren Artefakten sind im Kapitel 2 ausführlich behandelt worden. Hier werden die Ergebnisse der Phasen, die Ergebnisdokumente oder Artefakte, rund um die Testspezifikation genauer betrachtet.

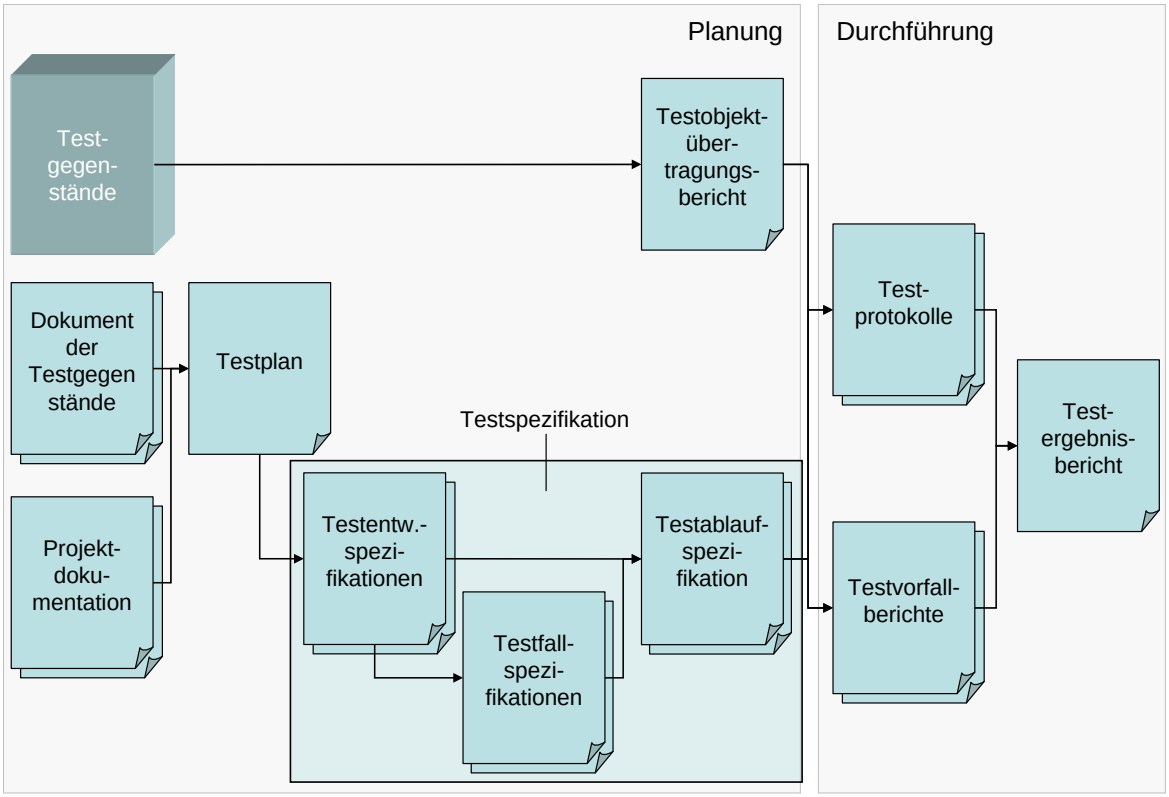



**Abbildung 3.2:** Testdokumente und ihre Beziehung zueinander nach IEEE 829

**Definition 3.3 (Testspezifikation)** *Ein Dokument, das aus der Testentwurfspezifikation, der Testfallspezifikation und/oder der Testablaufspezifikation besteht [IST07].*

Mit der Testspezifikation werden die drei Dokumente Testdesignspezifikation (Testentwurfspezifikation), Testfallspezifikation und Testablaufspezifikation zusammengefasst. Diese drei Dokumente besitzen folgende Definitionen:

**Definition 3.4 (Testentwurfspezifikation)** *Ein Ergebnisdokument, das die Testbedingungen für ein Testobjekt, die detaillierte Testvorgehensweise und die zugeordneten logischen Testfälle identifiziert [IST07].*

Wobei eine Testbedingung „eine Einheit oder ein Ereignis [ist], z.B. eine Funktion, eine Transaktion, ein Feature, ein Qualitätsmerkmal oder ein strukturelles Element einer Komponente oder eines Systems, welche bzw. welches durch einen oder mehrere Testfälle verifiziert werden kann." [IST07].

Die Testvorgehensweise ist die Umsetzung einer Teststrategie [4].

Ein logischer Testfall bzw. abstrakter Testfall ist „ein Testfall ohne konkrete Ein- und Ausgabewerte für Eingabedaten und vorausgesagte Ergebnisse. Er verwendet logische Operatoren, weil die konkreten noch nicht definiert oder verfügbar sind." [IST07]. Im Gegensatz dazu steht der konkrete Testfall, dessen Ein- und Ausgabewerte konkrete Werte sind.

**Definition 3.5 (Testfallspezifikation)** *Ein Dokument, das eine Menge von Testfällen für ein Testobjekt spezifiziert (inkl. Testdaten und Vor-/Nachbedingung), bei dem die Testfälle jeweils Ziele, Eingaben, Testaktionen, vorausgesagte Ergebnisse und Vorbedingungen für die Ausführung enthalten. [IST07]*

Die Testdaten sind „Daten die (z.B. in einer Datenbank) vor der Ausführung eines Tests existieren, und die die Ausführung der Komponente bzw. des Systems im Test beeinflussen bzw. dadurch beeinflusst werden." [IST07]

**Definition 3.6 (Testablaufspezifikation / Testszenario)** *Ein Dokument, das eine Folge von Schritten zur Testausführung festlegt. Auch bekannt als Testskript oder Testdrehbuch [IST07].*

#### Begrifflichkeiten bei der Testautomatisierung

Die oben vorgestellte Aufteilung von Inhalten der Testspezifikation in drei Dokumente wird - wie Vorgehensmodelle auch (siehe vorheriges Kapitel) - häufig angepasst. Die Anpassung erfolgt dabei auf die sinnvollste Dokumentenaufteilung im Projekt bzw. im konkreten Vorgehensmodell. Hierbei spricht man von dem sogenannten Tailoring.

[4] Die Teststrategie wurde im Abschnitt 2.5, S. 29 vorgestellt.

Beim automatisierten Testen mittels Testsystemen (vgl. Kapitel 2, S. 13) wird anstatt der Testablaufspezifikationen von Testskripten gesprochen. Diese sind programmierte Abläufe und damit selber Software.

**Definition 3.7 (Testskript)** *Bezeichnet üblicherweise eine Testablaufspezifikation, insbesondere eine automatisierte. [IST07]*

Um den Inhalt und Umfang eines Testfalls in diese Dokumentenstruktur einordnen zu können, ist eine Betrachtung aus der Sicht der Testautomatisierung (vgl. 2.5, S. 29) notwendig.

### Dokumente bei der Testautomatisierung

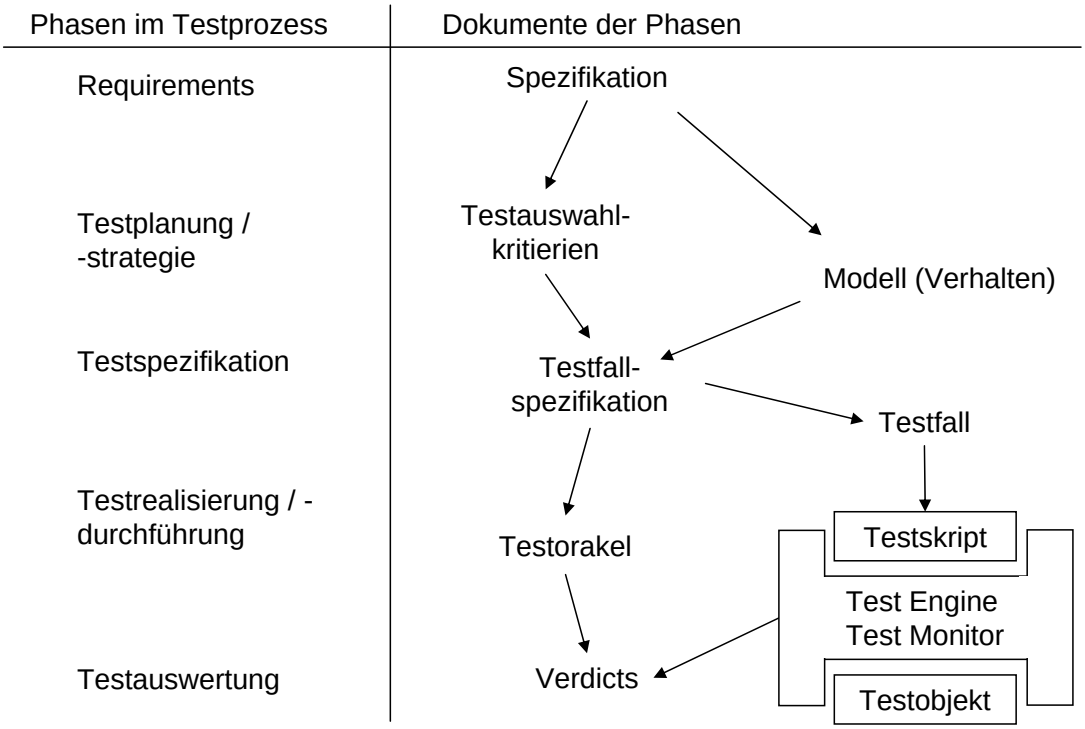



**Abbildung 3.3:** Dokumente bei der Testautomatisierung

In Abbildung 3.3 sind die Dokumente beim automatisierten Testen in Beziehung zu den Phasen im Testprozess (s. Abschnitt 2.5, S. 29) gesetzt. Zusätzlich ist das Testobjekt und das Testsystem, für eine bessere Verständlichkeit der Ergebnisdokumente aus den letzten Phasen (Zusammenspiel der Dokumente mit Testobjekt und Testsystem), eingezeichnet.

Aus der Spezifikation leitet der Testdesigner (siehe Abschnitt 2.5.3 über Beteiligte Rollen, S. 32 für die Aufgaben/Verantwortlichkeiten der Rollen) anhand definierter Testauswahlkriterien, die Testfallspezifikationen ab. Typisches, einfaches, Testauswahlkriterium wäre: „Führe alle Positiv-Testfälle durch“. Die Testfallspezifikation beinhaltet gleichzeitig Informationen für das Testorakel: hier wird die Frage beantwortet, welche Ergebnisse zu welchen Ergebnissen (Verdicts) führen. Besonders bei der automatisierten Testdurchführung ist das Testorakel in den Testskripten einprogrammiert und taucht explizit nicht auf.

**Definition 3.8 (Testorakel)** *Informationsquelle zur Ermittlung der jeweiligen vorausgesagten Ergebnisse, die mit den tatsächlichen Ergebnissen einer Software im Test zu vergleichen sind. [IST07]*

Die Modellerstellung (Verhaltensmodell) führt ebenfalls der Test-Designer durch. Diese Modellierung unterscheidet sich von der Modellierung in der Entwicklung (für die Spezifikation und Design). In den nächsten Abschnitten wird auf diese Thematik näher eingegangen.

Schließlich werden die erstellten Testskripte auf dem Testmittel ausgeführt um anschliessend die tatsächlichen Ergebnisse mit dem Testorakel zu vergleichen und damit das Testurteil (Verdict) zu fällen.

## 3.3 Bottom-Up Betrachtung: Testfallimplementierung

Die Testskripte bilden beim automatisierten Testen die Verknüpfung zwischen Testfallspezifikation und den Testsystemen. Dabei werden die Testskripte manuell von der Testfallspezifikation oder durch Testengines automatisiert abgeleitet (vgl. [Leh04]).

Die Programmierung der Testskripte kann wie die Programmierung der Implementierung bei der Entwicklung nicht ersetzt werden, da sie (die Programmierung) die feinste Auflösung der Anforderungen darstellt, d.h. sie kann nicht ignoriert oder abstrahiert werden (vgl. [Mar09]). **Demnach ist jede Anforderungsspezifikation, die ab einem bestimmten Detaillierungsgrad von einer Maschine ausgeführt werden kann eine Programmierung** [Mar09].

Im Folgenden ist das Ziel, die Sprachelemente auf der untersten Ebene darzustellen. Hieraus werden Erkenntnisse abgeleitet, in welchem Maße die Testfallimplementierung von den Plattformen abstrahiert und welche Voraussetzungen für die Testfallsspezifikation allgemein angenommen werden kann.

Wie in den vorherigen Abschnitten dargestellt sind die manuell oder automatisiert erstellten Testskripte die Testfallimplementierungen. Sie beinhalten zusätzlich zu den Abläufen eines Testfalls Informationen (Abläufe, Kontrollstrukturen, Daten, ...), die für die Ausführung auf der jeweiligen HW/SW Plattform notwendig sind und werden in Programmiersprachen implementiert.

### 3.3.1 Ablauf-, Testablauf-, Testumgebungsspezifische Sprachelemente

In [Hut06] (vgl. auch [Har01b]) werden folgende Elemente aus der Analyse (die notwendigen Bestandteile für den Steuergerätetest im Automobilen Bereich sind) erkannt und aufgezeigt:

- Zugriffsmöglichkeiten eines Tests auf Größen des Testsystems
- Zugriffsmöglichkeiten auf die Bussysteme
- Möglichkeit zur Beschreibung der Nachrichten
- Variablen und Konstanten
- Programmkonstrukte wie Schleifen und Verzweigungen
- Operationen zur Datenverarbeitung
- Elemente zur automatisierten Ergebnisbeurteilung
- Sprachmittel zur Integration der Zeit

Die Liste deckt sich mit den Experteninterviews und Analysen vorhandener Testskripte, die im Rahmen dieser Arbeit durchgeführt wurden. Die ausführliche Liste der Sprachelemente aus der Analyse ist am Ende des Kapitels vorhanden. Sie kann nie vollständig sein, beinhaltet jedoch die wichtigsten Elemente der Testfallimplementierung.

Abbildung 3.4 zeigt ein Beispiel-Testskript indem die unterschiedlichen Sprachelemente verwendet werden.

Erkennbar ist, dass die benötigten Elemente in drei Gruppen gegliedert werden können:

- Ablaufspezifische Sprachelemente

  Hierunter fallen Kontrollflussstrukturen wie Schleifen und Verzweigungen, Operationen zur Datenverarbeitung sowie Variablen und Konstanten. Sie sind in jeder Programmiersprache zu finden und bilden die Grundlage für eine Ablaufbeschreibung. Diese Sprachelemente werden sowohl für die Umsetzung auf die SW/HW Plattformen, als auch für die Umsetzung des Testablaufs benutzt.

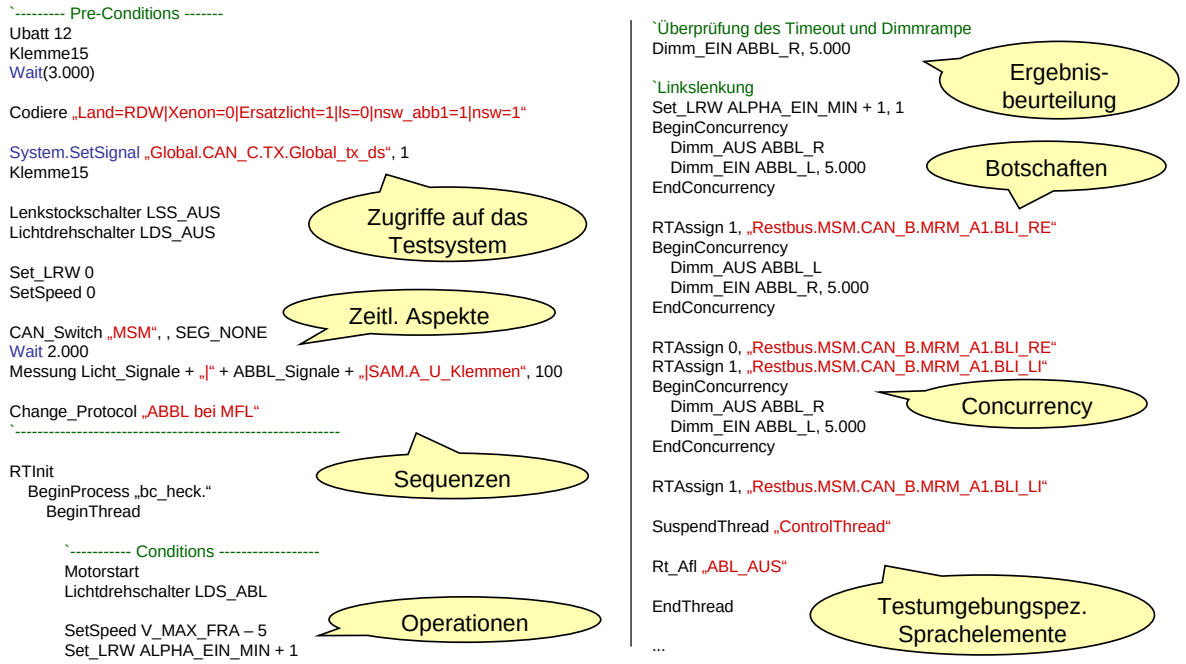



**Abbildung 3.4:** Verwendete Sprachelemente in einem Testskript

- Testablaufspezifische Sprachelemente

  Möglichkeiten zur Beschreibung von Nachrichten (Botschaften), Elemente zur automatisierten Ergebnisbeurteilung sind testablaufspezifisch.

  Die Sprachmittel zur Integration der Zeit (zeitliche Aspekte) werden ebenfalls hier eingeordnet.

- Testumgebungsspezifische Sprachelemente

  Hier sind die Zugriffsmöglichkeiten eines Tests auf die Größen eines Testsystems und auf die Bussysteme einzuordnen.

Ein Defizit bei der Testfallimplementierung ist die fehlende Transparenz der einzelnen Schritte eines Testfalls. So ist für das Verständnis einer Testfallimplementierung notwendig zu erkennen, ob es ein Schritt z.B. eine Aktion oder ein erwartetes Ergebnis ist. Durch eine explizite Darstellung in der Beschreibung - der Testfallspezifikation - wird die Lesbarkeit und das Verständnis erhöht. Etwaige Unstimmigkeiten, ob ein Schritt eine Aktion oder ein erwartetes Ergebnis ist, werden vermieden.

Auf der Ebene der Testfallimplementierung ist dies jedoch nur möglich, wenn der Kontext des ausgeführten Programmabschnitts bekannt ist, d.h. die Testfallspezifikation zu dem Testskript vorliegt.
Eine andere Möglichkeit ist das hinzufügen von redundanten Sprachelementen um den Inhalt der Testfallspezifikation abzubilden. Beispiel: Zusätzlich zu generischen

Aktionsbefehlen zum Setzen und Lesen von Signalen werden Befehle verwendet, die den Zweck kennzeichnen. Alleine durch diese Erweiterung wird das Verständnis über den Testfall nicht erhöht, denn dieser wird aufgrund der zusätzlichen Informationen erschwert, die für den Ablauf auf der SW/HW-Plattformen notwendig sind.

### 3.3.2 Auswirkungen auf den Testfall

Die Testfallimplementierung (Testskripte) besteht aus ablauf-, testablauf- und testumgebungspezifischen Sprachelementen.

Ablaufspezifische Sprachelemente sind in jeder allgemeinen Programmiersprache zu finden. Testablauf- und testumgebungsspezifische Sprachelemente sind Erweiterungen bzw. Spezialisierungen der allgemeinen Programmiersprache auf die Bedürfnisse der Testfallimplementierung.

So sind neben Kontrollstrukturen und Operationen auch Sprachelemente für den Nachrichtenaustausch, Sprachelemente für zeitliche Aspekte und die Ergebnisbeurteilung vorhanden. Zugriffe auf die Größen der Testsysteme sind testumgebungsspezifisch.

Offen bleibt, welche von diesen Sprachelementen für die Testskripte und welche für eine Testfallspezifikation gebraucht werden. Dazu wird nun der Weg von der Spezifikation bis hin zur Testfallimplementierung betrachtet. Dadurch wird auch dargestellt, welche Informationen die System-Spezifikation der Testfallspezifikation zur Verfügung stellt.
Aus der Betrachtung resultiert, dass eine Ablaufbeschreibung angereichert mit testablauf- und testumgebungspezifischen Sprachelementen die Beschreibung eines Testfalls abdeckt.

## 3.4 Top-Down Betrachtung: Spezifikation

### Ziel und Vorgehen

Ziel dieses Abschnitts ist es zu zeigen, dass grundsätzlich immer ein Modell vom Testobjekt zur Beschreibung der Testfälle erstellt wird. Um dies zu zeigen wird die Spezifikation aus der Perspektive der Testfallherleitung betrachtet. Anschliessend

werden Modelle klassifiziert, die Abstraktionsgrade und die Notationsarten betrachtet und die Auswirkungen auf den Testfall beschrieben.

### 3.4.1 Mentale Modelle bei der Testfallherleitung

**Definition 3.9 (Testverfahren)** *(1) Planmäßiges, auf einem Regelwerk aufbauendes Vorgehen zur Herleitung und/oder Auswahl von Testfällen. [IST07]*

Das Testverfahren bestimmt das Ergebnis: Einen Testfall. Äquivalenzklassen, Grenzwertanalysen, zustandsbasiertes Testen, Ursachen-Wirkungs-Graphen und Expertenwissen werden u.a. als Testverfahren für das funktionale Testen erwähnt. Im Anhang C erfolgt eine Vorstellung dieser Basis-Testverfahren.

**Das Expertenwissen ist eine entscheidende Basis bei der Herleitung der Testfälle in der Automobilbranche. Bei der Testfallherleitung werden insbesondere die Kenntnisse (Wissen, Erfahrung, ...) des Test-Designers herangezogen** ( [Sax08], [Kno07], [WRDZ07]).

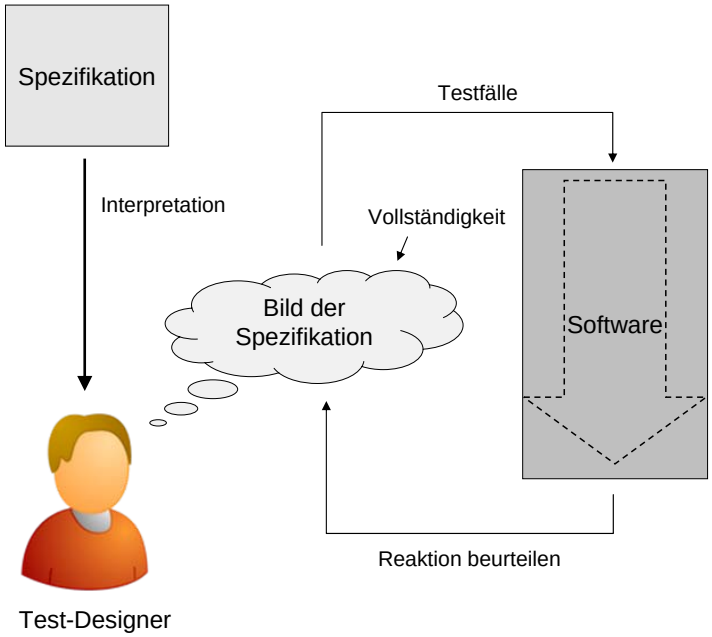



**Abbildung 3.5:** Der Test-Designer interpretiert die Spezifikation zu einem mentalen Modell [Lig02]

**Definition 3.10 (Mentales Modell)** *In interacting with the environment, with others, and with the artifacts of technology, people form internal, mental models of themselves and on the things with which they are interacting. [GS83]*

Der Test-Designer erstellt auf Basis seines Wissens eine Vorstellung des Systems [Lig02], das mentales Modell genannt wird (vgl. [HKO06], [Bei90], [BJK05], [Bin99], [BK04] sowie [Ley05]). Zitate aus diesen Quellen sind im Anhang B Mentales Modell, S. 263 vorhanden.

Beim modellbasierten Testen werden die Modelle explizit gestaltet (vgl. [BK04], [BJK05]).

**Aus diesen beiden Erkenntnissen folgt, dass Modelle immer erstellt werden.** Abbildung 3.5 stellt den Sachverhalt graphisch dar.

Je nachdem, welche Ziele beim Testen verfolgt werden (siehe Testplanung im Abschnitt 2.5, S. 29) sind unterschiedliche Betrachtungen auf das System notwendig.

Es gibt viele Definitionen des Modellbegriffs. Deshalb ist es sinnvoll hier auf die Begrifflichkeiten rund um die Modellierung einzugehen. Dazu wird folgende Definition verwendet:

**Definition 3.11 (Modell)** *Unter einem Modell ist ein System zu verstehen, das als Repräsentant eines komplizierten Originals auf Grund mit diesem gemeinsamer, für eine bestimmte Aufgabe wesentlicher Eigenschaften von einem dritten System benutzt, ausgewählt oder geschaffen wird, um letzterem die Erfassung oder Beherrschung des Originals zu ermöglichen oder zu erleichtern bzw. es zu ersetzen. (aus [Dre06]) nach [Wüs63])*

**Das heißt, dass ein Modell durch die Abstraktion („Repräsentant eines komplizierten Originals“) und durch die Sichtweise („für eine bestimmte Aufgabe wesentlicher Eigenschaften von einem dritten System“) definiert wird.**

Diese beiden Kernaussagen der Modellierung werden genauer betrachtet.

### 3.4.2 Abstraktionsprinzipien bei der Modellierung

Die Reduzierung des Systems auf das Wesentliche bedeutet, das System zu abstrahieren. Ein Modell, als Ergebnis der Systemabstraktion, kann nach den Funktionen, den Daten, der Kommunikation, den zeitlichen Aspekten und deren Kombination spezifiziert werden [PP04], [HCDG+02].

- Funktionale Abstraktion

  Das Modell wird aus funktionalen Aspekten bzw. Aspekten des Verhaltens heraus spezifiziert. Hierbei werden Funktionsmodelle erstellt, die in der Implementierung nicht explizit vorhanden sind, sondern durch eine Reihe von Funktionen erreicht werden. Beispiel: Die Funktion „Blinken“, die aus eine Reihe von Unterfunktionen besteht, wird im Modell als eine Funktion abgebildet.

- Datenabstraktion

  Daten können abstrahiert werden in entweder logische (z.B. Boolean, Integer) oder allgemeine Datentypen (z.B. Liste, Arrays, Dictionary,...). Beispiel: Bei dem Testverfahren bei denen Äquivalenzklassen angewandt werden, werden Eingabewerte in Klassen eingeteilt, die die gleiche Reaktion hervorrufen.

- Kommunikationsabstraktion

  Bei dieser Abstraktionsform werden Nachrichten, die gemeinsam vorkommen um eine Funktionalität zu erreichen, gebündelt und dadurch abstrahiert. Beispiel: Statt mehrere CAN Botschaften um die Blinkerbetätigung zu modellieren, wird eine Kommunikationsnachricht 'Blinker betätigen' modelliert. Ein anderes Beispiel wäre, dass eine Nachricht vom 'Sender' zum 'Empfänger' verschickt wird, wobei Sender und Empfänger abstrakt bleiben.

- Zeitliche Abstraktion[5]

  Zeitlich wird dann abstrahiert, wenn die Reihenfolge der eintretenden Ereignisse von Bedeutung sind und die Information in welchen zeitlichen Abständen sie zueinander liegen als irrelevant betrachtet wird.

  Eine zeitliche Abstraktion würde bei reaktiven Systemen verhindern, dass Aspekte der Echtzeiteigenschaften überprüft werden können. So taucht diese Abstraktionsform bei der Vorstellung der Modellierungsformen für hybride Systeme nicht auf.

**Je nachdem, welche Abstraktionsform bzw. Kombinationen benutzt werden, werden unterschiedliche Aspekte des Testobjekts getestet.**

Dabei ist beim Testen eine geeignete Abstraktion des zu testenden Systems notwendig um ein kontrollierbares und damit handhabbares Modell zu erstellen (vgl. [BK04], [MB08], [PAD+98]).

Grundsätzlich gilt: Die vorgestellten vier Arten müssen durch die Testfälle unterstützt werden, d.h. funktionale, Daten-, Kommunikations- und zeitliche Aspekte müssen beschrieben werden können.

[5] Diese Art wird auch Abstraktion von der Quality-of-Service genannt.

Der Test-Designer entwirft somit seine Testfälle und seine Vorstellung von dem System zwangsläufig unter Berücksichtigung anderer Gesichtspunkte, da ein unterschiedliches Ziel (Testen) verfolgt wird.

So unterscheidet sich nach diesem Gesichtspunkt ebenfalls ein Modell aus der System-Spezifikation gegenüber einem Modell aus der Testfallherleitung, obwohl beide das gleiche System spezifizieren.

Auf diese unterschiedlichen Gesichtspunkte (Sichten) wird im Folgenden eingegangen.

### 3.4.3 Sichtweisen: Internes und externes Systemverhalten

Bei der Modellierung ist die Sichtweise die zweite relevante zu betrachtende Eigenschaft.

Es folgt eine Analogie zur Regelungstechnik und der Softwaretechnik, um zwei Begriffe für die Modellierung der Systeme einzuführen, die für die weiteren Kapitel gelten sollen.

#### Vergleich zur Synthese und Analyse in der Regelungstechnik

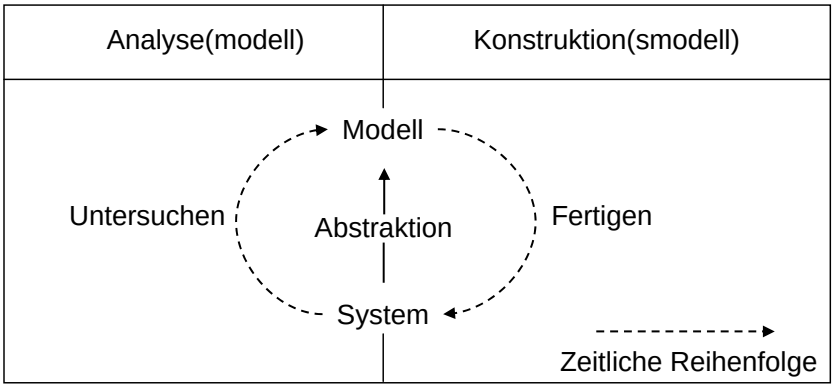



**Abbildung 3.6:** Analyse und Kontruktion in der Regelungstechnik [Tab07]

In der Regelungstechnik (und auch in anderen Gebieten) werden Systeme erstellt oder vorhandene (erstellte) Systeme analysiert. In der Automatisierungstechnik werden diese Vorgänge Synthese und Analyse genannt [Tab07]. Die Synthese entspricht der Entwurfsphase bei der Softwareentwicklung: Hier werden Modelle konstruiert

und dementsprechend das System durch sukzessive Weiterentwicklung bzw. Verfeinerung der Modelle definiert, bis eine (idealerweise) automatische Umsetzung mittels Werkzeugen möglich ist [CFGK05].

Je nachdem welche Eigenschaften beschrieben werden sollen, wird ein Spezifikationsmittel ausgewählt oder kombiniert. Beispiel aus den Diagrammen der UML verdeutlicht diesen Sachverhalt: Bei der Modellierung einer Softwarearchitektur wird am Anfang das externe Verhalten durch Anwendungsdiagramme, Interaktionsdiagramme und/oder auch Aktivitätsdiagramme festgehalten. Gegen Ende wird das interne Verhalten definiert. Hier kommen Zustandsdiagramme, Timingdiagramme und/oder auch Aktivitätsdiagramme zum Einsatz. So erfüllt jedes Diagramm den Einsatzzweck für den es konzipiert wurde.

Bei der Analyse in der Regelungstechnik wird ein vorhandenes System untersucht und ein Modell aufgebaut. Dieses Analysemodell kann für weitere Untersuchungen oder dem Verständnis des ursprünglichen Systems dienen. Häufig findet in der Analyse auch eine Abstraktion statt, mit deren Hilfe, die Komplexität realer Systeme reduziert werden kann.

Verständlicherweise werden bei der Synthese, d.h. der Konstruktion, andere Modelle verwendet als bei der Analyse.

**Die Analogie zum Testen**: Wie bei der Analyse wird beim Testen auf der Basis eines entworfenen Systems (der Spezifikation) ein Modell aufgebaut, um diese zu untersuchen bzw. zu überprüfen. Beim funktionalen Testen (vgl. Kapitel 2.4.2, S. 28) wird überprüft, ob sich das entworfene System wie in der Spezifikation definiert verhält. Hierfür ist die Umsetzung der Spezifikation, d.h. das interne Verhalten irrelevant. Wie sich das System - gemessen an seiner Spezifikation - ganzheitlich äußert wird bewertet, d.h. am externen Verhalten.

Diese Unterscheidung führt zu den folgenden Definitionen. In der Literatur wird der Begriff Verhaltensmodell verwendet (vgl. [NSL08]). Dabei wird ein Verhaltensmodell auf drei Arten klassifiziert:

- Abstrakte Definition des Verhaltens für frühe oder implementierungsunabhängige Simulationen und Analysen (abstraktes Verhaltensmodell).

  Bei der Testfallerstellung ist diese Art der Verhaltensmodelle zu finden. Ebenso werden in den ersten Phasen des Entwurfs (Vorphase zum Grobdesign und Grobdesign) und der Implementierung abstrakte Definitionen des externen Verhaltens erstellt.

- Konkrete Beschreibung des Verhaltens für implementierungsorientierte Simulation, Analyse und Codegenerierung (konkretes Verhaltensmodell).

- Die Implementierung an sich für Simulationen, Analysen und für die Umsetzung auf das Zielsystem (Implementierung).

Da der Begriff Verhaltensmodell in den Testsystemen für die Modellerstellung geprägt sind (vgl. [Hut06], [Har01b]), wird der Begriff Systemverhaltensmodell auf zwei Arten eingeführt:

**Definition 3.12 (Systemmodell / Internes Systemverhaltensmodell)** *Ein Systemmodell (...) ist eine Abstraktion zu einem System (...), welche nur eine Menge ausgewählter, gerade interessierender Sachverhalte des betrachteten Systems aufweist. [Tab07]*

Die Definition des Systemmodells entspricht dem des Modellbegriffs, nur eingeschränkt auf Systeme. Aus der obigen Klassifikation der Verhaltensmodelle werden mit dieser Definition die beiden letzten zusammengefasst.

Die Systemmodelle bedienen sich komplexer Modellierungen wie Differentialgleichungen [Rei06], [Unb08] [WR06] [Gev06a] aber auch graphische Modellierungssprachen wie UML, Satecharts und Werkzeuge wie Simulink [WB05], [WR06]. Eine Auflistung an möglichen Spezifikationssprachen ist in [Sch07] und [Mar08] zu finden. Alle haben ein Ziel: Die Modellierung. Zweck ist, das (interne) Verhalten zu spezifizieren.

Wird rein das Verhalten des Systems zu seinem Umfeld modelliert, so wird im Folgenden dieser Zustand begrenzt auf den Begriff des externen Verhaltensmodells bzw. Systemverhaltensmodells (vgl. [GS05]):

**Definition 3.13 (Systemverhaltensmodell / Externes Systemverhaltensmodell)** *Ein Systemverhaltensmodell wird primär der Spezifikation des Verhaltens der zu realisierenden Funktion verwendet. (vgl. [GS05])*

Aus der obigen Klassifikation ist die erste Aufteilung („Abstraktes Verhaltensmodell“) mit dem externen Systemverhalten gemeint.

Dabei ist das Verhalten definiert als: „Art und Weise, wie sich jemand oder etwas verhält“.

Die Verhaltensmodelle werden mit den gleichen Spezifikationsmitteln wie die Systemmodelle beschrieben (vgl. auch [GS05]) . Der einzige Unterschied besteht darin, dass das externe Verhalten des zu realisierenden Systems spezifiziert wird.

Auf Grundlage der Beschreibung des externen Systemverhaltens - und nicht ihrer Umsetzung (internes Verhalten) - werden die Testfallspezifikationen abgeleitet. Verhaltensmodelle für das Testen reduzieren die Systemmodelle auf bestimmte Aspekte der Spezifikation, die im Rahmen des Prüfung betrachtet werden sollen [PP04]. Beim funktionalen Testen ist dies das externe Verhalten (vgl. [Sch03], [Pre03], [Hor05], [HKO06], [Bei90], [BJK05] und insbesondere Zitate im Anhang B Mentales Modell, S. 263).

In [TD09] wird die unterschiedliche Betrachtung auf das gleiche System wie folgt aufgegriffen: „Classical models from computer science (e.g., finite-state automata) provide means for reasoning about discrete systems only. Classical models from engineering (e.g., differential equations) provide means for reasoning mostly about continuous systems.“[6]

Das heisst, dass auf der Ebene der Testfallerstellung die Systeme abstrakter betrachtet werden als auf der Entwurf- und Implementierungsebene der Systeme. So wird - wie in den folgenden Abschnitten zu sehen - bei der Testfallherleitung das kontinuierliche System diskret betrachtet.

### 3.4.4 Zusammenfassung

Grundsätzlich werden immer Modelle erstellt; sei es implizit (mentales Model) oder explizit. Die Modelle basieren auf zwei Kernaspekten: Abstraktion des Systems und die Sichtweise auf das System. Beide haben einen Einfluss auf den Testfall.

Die Abstraktion wird auf vier Arten (Funktion, Daten, Kommunikation, Zeit) aufgeteilt.

Die Sichtweise auf das System wird auf zwei Darstellungen fokussiert: Das interne Systemverhalten und das externe Systemverhalten.

Für den (Fein-)Entwurf (Synthese) eines System wird bevorzugt das interne Systemverhalten modelliert; in der Regelungstechnik als Konstruktionsmodell bekannt. Für die Ableitung von Testfällen für das funktionale Testen wird ausschließlich das externe Systemverhaltensmodell verwendet; in der Regelungstechnik vergleichbar mit dem Analysemodell, das das System aufs externe Auftreten hin beschreibt um - im Kontext - Verifikationen vornehmen zu können.

So unterscheiden sich Verhaltensmodelle, die zur Testfallherleitung benutzt werden

[6] Wie später bei der Analyse der Systemverhaltensmodelle zu sehen, werden hier Timed Automata bzw. die hybriden Automaten als Bindeglied zwischen den beiden Anschauungen dargestellt.

von den Verhaltensmodellen für die Spezifikation oder für eine automatisierte Codeableitung [PP04][7].

Vergleich zur Softwareentwicklung: Eine Abstraktionsschicht mit der die Struktur des Systems abgebildet werden kann (die Architektur), eignet sich nicht dafür, um hiervon eine Implementierung abzuleiten. Das dynamische Verhalten des Systems muss für die Implementierung abgebildet werden.

Wie oben an den Diagrammen der UML dargestellt, sind die Diagrammtypen für die unterschiedlichen Aspekte der Modellierung konzipiert.

Im nächsten Schritt werden die Notationen der Systemverhaltensmodelle näher betrachtet, denn ein Testfall muss die geeigneten Sprachelemente besitzen, um diese Verhaltensmodelle überprüfen zu können. Insbesondere Verhaltensmodelle für die Spezifikation des externen Verhaltens von hybriden Systemen mit Echtzeiteigenschaften.

## 3.5 Systemverhaltensmodelle als Testbasis

Das externe Verhalten wird über Ablaufstrukturen abgebildet (siehe Abbildung 3.7), die das dynamische Verhalten des Systems abbilden. So wird im Folgenden das Fokus auf das dynamische, externe Verhalten gelegt.

Zuerst folgt eine Klassifikation der Verhaltensmodelle und anschließend eine detaillierte Betrachtung der Klassen. Ziel dieses Abschnitts ist es, die Verhaltensmodelle zu beschreiben um hiervon Elemente für die Testfallbeschreibung abzuleiten.

Grundgedanken dieses Abschnitts, die aufgezeigt werden sind:

- Ein Testfall muss die geeigneten Sprachelemente besitzen, um das dynamische Systemverhalten überprüfen zu können.
- Dabei spielt die Notation der Modellierung dieses Systemverhaltens eine zentrale Rolle und hat auf die Testsemantik einen entschiedenen Einfluss.

[7] Bei der Abstraktion des Systems aus der Sicht der Testfallherleitung können kontinuierliche Aspekte in diskrete abgebildet werden.

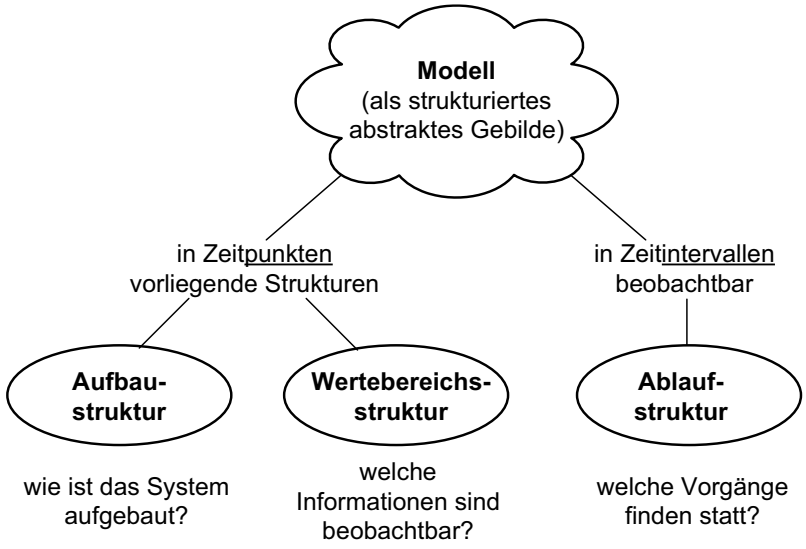



**Abbildung 3.7:** Strukturtypen eines Systemmodells [Tab07]

### 3.5.1 Klassifikation von Verhaltensmodellen

[Zei84] (nach [Ros01]) unterteilt die Systemverhaltensmodelle in drei Klassen: zeitdiskrete, zeitkontinuierliche und ereignis-diskrete Modelle (vgl. auch [BK04]).

In [Unb08] und [Tab07] werden Systeme in folgende Kategorien aufgeteilt: zeitinvariante und zeitvariante Systeme, kontinuierliche und zeit-diskrete Systeme, deterministische und stochastische Systeme, kausale und nicht-kausale Systeme und stabile und nicht-stabile Systeme.
[Sch07] setzt diese Aspekte der Systeme für die Modellierung in Beziehung. So wird hier die Modellierung diskreter und/oder kontinuierlicher Information, Zustands-, Aktivitäts- und Struktur-orientierte Verfahren und deren Mischformen, mit oder ohne Modellierung zeitabhängiger Informationen erwähnt.

Diese Klassifikationen sind sinnvoll, wenn die Verarbeitung betrachtet werden soll und hieraus die grundlegenden Elementgruppen abgeleitet werden sollen. Diese Gruppen wurden in Kapitel 2 identifiziert (siehe Abschnitt 2.8, S. 56).
Ebenfalls ermittelt wurde, dass sich die Untersuchung auf reaktive Systeme mit Echtzeitverhalten auf analoge und digitale Signalverarbeitung eingrenzen lässt.

Für die Ermittlung der benötigten Sprachelemente, d.h. eine Ebene tiefer als die Ermittlung der Elementgruppen, muss die Modellierung an sich und damit die Notation betrachtet werden.

### 3.5.2 Paradigmen der Modellierung

Eine andere Art der Klassifikation der Modelle ensteht, wenn man sie über die unterschiede in der Notation klassifiziert. Die Paradigmen bei der Notation können in folgende Klassen unterteilt werden ( [UPL06], [vL00]):

- Pre/Post Notation

  das Modell wird durch eine Reihe von Variablen und Operatoren definiert. Die Operation wird durch Vor- und Nachbedingungen definiert. Beispiele hierfür sind die Sprachen Z und VDM.

- Transistionsbasierte Notation

  Die Übergänge zwischen Zuständen, die Transitionen, sind Schwerpunkte bei dieser Notationsart. Beispiel: Endliche Automaten, UML Diagramme und andere auf Graphen (Knoten und Kanten) basierte Notationsmittel [8].

- History-basierte Notation

  Auf die Zeitachse wird das definierte Verhalten abgebildet. Beispiele: MSC (Message Sequence Charts) und auf diesen basierende Diagrammtypen wie die UML Sequenzdiagramme.

  Typischerweise werden hier Komponenten und deren Beziehung zueinander (Nachrichtenaustausch) modelliert.

- Funktionale Notation

  Mathematische Funktionen bilden die Grundlage bei dieser Notationsart.

- Operationelle Notation

  Bei dieser Notationsart wird das zu definierende System als eine Menge von gleichzeitig laufender Prozesse definiert. Typischerweise werden hier verteilte Systeme und Kommunikationsprotokolle abgebildet. Beispiele: CSP/CCS (Communicating Sequential Processes / Calculus of Communicating Systems) und Perti-Netze.

- Stochastische Notation

  Durch Wahrscheinlichkeitsmodelle der eintretenden Ereignisse und Eingabewerte wird mit dieser Notationsart modelliert. Beispiel: Markov-Ketten

- Datenfluss Notation

  Bei dieser Notationsart stehen die Daten im Vordergrund. Beispiel: Blockdiagramme und die synchrone deklarative Programmiersprache Lustre.

  Typischerweise werden hier Modellierungen des kontinuierlichen Systemverhaltens definiert.

[8] Anmerkung: Die Knoten/Kanten Beziehung kann auch in textueller Form erfolgen.

### 3.5.3 Die Testsemantik und das Analyseschema

Bei der Analyse der Sprachelemente auf der Ebene der Testfallimplementierung wird gezeigt, dass sich diese in ablauf-, testablauf- und testumgebungsspezifische Sprachelemente unterteilen lassen (vgl. Abschnitt 3.3, S. 66). So kann jede Ablaufbeschreibungssprache mit speziell für das Testen notwendige Sprachelemente angereichert werden, um hieraus eine Sprache für die Testfallimplementierung zu erstellen.

Diese Idee soll weiter fortgeführt werden. Hierfür wird eine Definition eingeführt:

**Definition 3.14 (Testsemantik)** *Die Testsemantik bezeichnet die Sprachelemente oder die Mittel, die gebraucht werden um Testen zu können.*

Beispiel für die Testsemantik: Gegeben ist eine generische Programmiersprache. Die Testsemantik wird erreicht durch z.B. hinzufügen einer Funktion, die die Überprüfung eines Wertes auf sein Sollwert vornimmt und gegebenenfalls Maßnahmen (Eintrag ins Testprotokoll) einleitet. Ohne diese Funktion würde die Programmiersprache keine Bedeutung für das Testen besitzen.

Konkret werden Sprachelemente für:

- die Signalbeschreibung,
- die Überprüfung der diskreten und stetigen Verarbeitung,
- der verteilten Funktionen und
- die Echtzeiteigenschaften

untersucht, die sich aus der Betrachtung in dem vorherigen Kapitel herausgestellt haben (vgl. Abschnitt 2.8, S. 56).

### 3.5.4 Einschränkung des Betrachtungsraumes

Nicht alle Notationen der allgemeinen Modellierung werden für die Beschreibung von Testfällen eingesetzt bzw. können eingesetzt werden, d.h. eine Untersuchung wird nicht vorgenommen, weil wissenschaftliche Arbeiten in diesen Bereichen nicht gefunden wurden.

Im Kapitel Stand der Technik Abschnitt 5.3.1, S. 130 werden auf den obigen Notationsarten basierende Spezifikationssprachen aufgezeigt.

Deutlich wird, dass Automaten einen breiten Einsatz bei der Beschreibung der Testfälle finden, d.h. bei der Beschreibung der Testfälle die transistionsbasierte Notation ihren Schwerpunkt hat. Häufig werden die Pre/Post und die stochastischen Notationsarten in Kombination mit der transitionsbasierten Notation zur Testfallerstellung verwendet.

Die history-basierte Notation wie z.B. MSC oder UML Sequenzdiagramme werden für die Modellierung des Nachrichtenaustausches über der Zeit verwendet. Die Testsemantik erfolgt hier durch Aufruf von Funktionen, so dass komplexe Testabläufe in Funktionen verlagert werden (s. Abschnitt 5.3.4, S. 152).

Die funktionale Notation wird aufgrund ihrer geringen Verwendungsgrades nicht betrachtet (vgl. [UPL06]).

Blockdiagramme (Datenfluss Notation) eignen sich für die Modellierung von System-Spezifikationen mit kontinuierlichen Aspekten, so dass auf diesem Konzept basierende Werkzeuge wie Matlab/Simulink oder Labview in der Automobilbranche häufig anzutreffen sind. Hier ist insbesondere die Arbeiten von [Leh04] und [GWWH00] zu erwähnen, die im Stand der Technik betrachtet werden. [PCG05] und [CFGK05] betrachten die Erstellung Testdaten, um diese Art der Diagramme zu testen, sodass diese auf Sequenzen von Testdaten zurückzuführen ist (vgl. folgende Untersuchung).

Im Folgenden wird die Analyse auf die transistionsbasierte (Automaten) und die operationelle (CSP) Notation eingeschränkt.

### 3.5.5 Analyse der Systemverhaltensmodelle

Die Analyse ist in drei Unterabschnitte aufgeteilt: Modellierung diskreter Zeit und diskretem Wert, Modellierung kontinuierlicher Zeit und diskretem Wert und Modellierung kontinuierlicher Zeit und kontinuierlicher Werte.

#### Diskrete Modellierung

Bei der Betrachtung von diskreter Zeit und diskreten Werten können Prozessalgebren zur Testfallspezifikation / Modellierung benutzt werden [BK04], [PAD+98]. Weiterhin sind Automaten bekannte Mittel für die diskrete Modellierung. Arbeiten, die die Partitionierung der Daten behandeln, werden im Stand der Technik untersucht. Im Folgenden wird eine auf Communicating Sequential Processes (CSP) basierte Spezifikationsmöglichkeit vorgestellt. Weitere Prozessalgebren wie Calculus of Communicating Systems (CSS) sind ähnlich aufgebaut. Anschliessend wird Timed Automata

als Automaten Repräsentation für die diskrete Modellierung betrachtet.

**Timed CSP** Mit der Communicating Sequential Processes (CSP) Sprache werden verteilte Systeme (genauer die Kommunikation der Prozesse) modelliert. Diese wird bei der Timed CSP Variante durch zeitliche Eigenschaften erweitert, um Echtzeitsysteme zu analysieren und zu modellieren [RR86]. Hierbei betrachtet Timed CSP die Zeit als reel, d.h. dicht, so dass zwischen zwei aufeinanderfolgenden Ereignissen eine beliebig kleine Zeiteinheit vergehen kann.

Wichtige Sprachelemente sind die Ereignisse (Events genannt im weiteren Verlauf), die keine zeitliche Ausdehnung besitzen und die Warteanweisung Wait: das einzige Element, wodurch sich Timed CSP von CSP unterscheidet. Die Defintion der Prozesse erfolgt indem Events zusammengesetzt werden (im weiteren Verlauf: Traces). Zusätzlich werden Aspekte des Ressourcenverbrauchs mit in die Beschreibung eingebunden, die bei der Prozessmodellierung Sinn ergeben.

Beispiel: $P = bremse \rightarrow Stillstand$

Bedeutung: nach Event *bremse* verhält sich Prozess P wie Prozess Stillstand.

Weitere bemerkenswerte Sprachelemente sind ( [BHR84] für eine detaillierte Einführung): Alternative Auswahl, nichtdeterministische Auswahl, parallele Komposition und die Möglichkeit, rekursiv zu modellieren.

Die Testsemantik kann wie folgt erreicht werden [GRDNPG97]: Ein Prozess wird als Beobachter (Observer) eingeführt, der die Prozesse des Testobjekts anstößt. Durch die definierte Aktion ok wird die erfolgreiche Terminierung gekennzeichnet.

Events werden dazu verwendet, um die Eingaben und Ausgaben zu und von dem Testobjekt zu modellieren.

**Timed Transistion Systems** Die Timed Transistion Systems (TTS) basieren auf den Timed Automata. Die Theorie der Timed Automata [AD94] geht ebenfalls von einer dichten Zeiteinheit aus. Allerdings lassen sich hiermit auch diskretes Verhalten modellieren und Testen (vgl. [BK04]).

Wie jeder Automat besteht auch dieser aus einer Reihe von Zuständen, einem Initialzustand und einer beschrifteten Transition. Des Weiteren sind Variablen und eine Zeitinvariante vorhanden. Die TTS bestehen aus mehreren dieser Automaten. Die Beschriftung der Kanten kann Folgende sein: Guards, die angeben, wann die Transition geschaltet werden kann. Events, die angeben, bei welchem Event die Transition ge-

schaltet wird und Zuweisungen, die die Zeit und Variablen verändern können, wenn eine Transistion geschaltet wird.

Die Events dienen zur Synchronisation und somit zur Kommunikation zwischen den Automaten. Wie bei den Prozessalgebren vorgestellt, existieren die Events auch hier in zwei Formen: Eingehende und ausgehende Events. Dabei wird das ausgehende Event eines Automaten als Eingabe eines anderen Automaten benutzt um die Transitionen zu synchronisieren.

Die Testsemantik wird wie folgt erreicht: Einige Events werden als Schnittstelle zum Testobjekt definiert, d.h. eingehende Events zum Testobjekt und ausgehende Events vom Testobjekt. Die Zeitmessung erfolgt mittels einer diskreten Uhr, deren Auflösung von der verwendeten Hardware (Testobjekt und Testumgebung) abhängt.

Die Anbindung zur Testumgebung hat somit alle benötigten Informationen: Die Schnittstellen zu dem Testobjekt, die durch Events definiert werden. Gleichzeitig werden somit alle möglichen Events definiert, die vom Testobjekt ausgehen können oder mit denen man das Testobjekt stimulieren kann. Die Betrachtung der diskreten Zeitmessung erfolgt auf der Tatsache, dass die Testmittel selbst nur eine gewisse Auflösung erreichen können. Da die Werte (Ein- und Ausgaben) diskret betrachtet werden, ist die Modellierung kontinuierlicher Werte nicht möglich.

#### Zeitkontinuierliche und wertdiskrete Modellierung

Wie bei der diskreten Modellierung erwähnt, kann die Betrachtung der Zeit in beliebiger Auflösung erfolgen, d.h. die Anzahl der Zustände ist unendlich. Somit gilt für diese Modellierung die gleiche wie bei der diskreten Modellierung.

#### Kontinuierliche Modellierung

Für die Steuergerätetests ist die kontinuierliche Wertedarstellung von Bedeutung (vgl. Abschnitt über das Testobjekt 2.7, S. 40).

Die zeitkontinuierliche und wertkontinuierliche Betrachtung wird u.a. durch die Hybrid Automata [Hen96] aufgegriffen (s.a. [CAC+95]). Ebenfalls über eine Abstraktion des kontinuierlichen Systems durch Automaten wird in [MB08] diskutiert. Weitere Modellierungsmöglichkeiten der hybriden Systeme sind: Hybride CSP (Erweiterung der Timed CSP) und HybridUML. Da auf dem Stand der Technik bei dem erstgenannten einige Mittel aufsetzen, wird dieser hier näher vorgestellt.

[Tet09] zeigt eine umfangreiche Übersicht über die Spezifikationsmittel für hybride Systeme. Hierbei wird ebenfalls auf die Semantik bei der Beschreibung eingegangen.

**Hybrider Automat** Bei den hybriden Automaten ist die grundlegende Idee folgende: Die Zeit, die reel betrachtet wird, wird durch die Zustände in Äquivalenzklassen unterteilt. Die diskreten Abschnitte des hybriden Systems werden als Transitionen modelliert, wohingegen die kontinuierlichen Abschnitte innerhalb der Zustände durch z.B. Differentialgleichungen ausmodelliert werden. Die Zustandsübergänge nehmen keine Zeit in Anspruch. Auch hier werden Events dazu benutzt, bei einer schaltenden Transition, den Zustandübergang zu erzeugen. Eine Transition ist schaltbereit, wenn die Sprungbedingung von einem Zustand in den nächsten wechselt. Wie alle anderen Automaten besitzt auch diese Variante ein Initialzustand.

Bei der Testsemantik stellt die Überprüfung kontinuierlicher Werte eine Herausforderung dar: ein kontinuierlicher Verlauf über die Zeit und den Wert erfolgt in einem gewissen Toleranzrahmen zu dem erwarteten Verlauf. Die Modellierung mittels dieser Methode wird in Kapitel 5 Stand der Technik, S. 123 näher vorgestellt.

### Zusammenfasssung der Untersuchung

Die unterschiedlichen Modellierungsformen folgen den unterschiedlichen Verhaltens bzw. Betrachtungsformen. Echtzeitsysteme bestehen im Gegensatz zu klassischen Systemen aus der „Erweiterung“ der Zeit: der zeitliche Aspekt definiert das Echtzeitsystem.

So werden die klassischen Spezifikationssprachen wie Prozessalgebren und Automaten durch Sprachelemente erweitert, die ebend die Eigenschaft bezüglich der Zeit abdecken. Bei den Timed CSP besteht die Erweiterung gegenüber dem CSP aus einem Element: der *Wait* Anweisung.

Diese Erweiterung bzw. Ergänzung setzt sich bei den hybriden Systemen fort: hier wird zu den Aspekten der Echtzeitsysteme die kontinuierliche Betrachtung bei der Zeit und den Werten hinzugefügt. Die kontinuierliche Zeit wird entweder durch eine Abbildung der dichten Zeiteinheit auf die diskrete, oder durch hinzufügen weiterer Sprachelemente abgebildet.

Die Testsemantik wird erreicht, indem bei den unterschiedlichen Modellierungsarten die Interaktion des Testobjekts mit seiner Umgebung definiert wird und auf dem aufbauend der Test-Designer die Testfälle ableitet. Dabei bilden bei allen drei Arten der Zeit und Wertebetrachtung die Events die Grundlage.

Die Erstellung von Testfällen für kontinuierliche Systeme erfolgt durch die diskrete Betrachtung auf das kontinuierliche Systems, d.h. die diskrete Modellierung des kontinuierlichen Systems.

Für den kontinuierlichen Wertverlauf (Signalverlauf) muss bei den erwarteten Signalverläufen (Ausgabe des Testobjekts) eine Toleranz sowohl im Zeit- als auch im Wertbereich vorgenommen werden.

Die Ablaufstruktur der Testfälle besteht aus einer Reihe von Sequenzen oder einer Abbildung eines Automaten (bzw. Pfad im Automaten abgebildet durch Sequenzen).

Da der Automat als Kernelement den Zustand besitzt, werden die Eingaben durch Zustände formuliert, d.h. der Zustand als solches tritt als Stimulation eines kontinuierlichen Werteverlaufs auf und der Übergang von einem Zustand in den nächsten wird als Soll-Reaktion aufgefasst bzw. dem diskretem Verhalten zugeordnet. Der verteilte Aspekt bei mehreren Steuergeräten spielt eine untergeordnete Rolle, da bei Abstraktion das Gesamtverhalten auf ein Testobjekt reduziert werden kann (vgl. [Kho02]), mit den gleichen Ein- und Ausgabeverhalten wie oben definiert.

### 3.5.6 Zusammenfassung

Bei der Untersuchung der Systemverhaltensmodelle im Hinblick auf die Testsemantik wird deutlich, dass ein typischer Testfall für reaktive Systeme aus einer Reihe von Paaren mit ein Ein- und Ausgabewerten bestehen. Die Echtzeiteigenschaft wird durch Sprachelemente bezüglich der Zeit erweitert (z.B. wait) und der hybride Charakter durch eine Aufteilung: innerhalb der Zustände eines Automaten wird das kontinuierliche Verhalten modelliert und die diskrete Verarbeitung als Transition zwischen diesen Zuständen abgebildet.

Das nichtdeterministische Verhalten wird durch die Auswahl bei der Modellierung unterstützt. Für den Testfall bedeutet dies in einer Gabelung: die Reihe von Ein- und Ausgabepaaren werden zu Bäumen. Aus der Voruntersuchung (Einschränkung des Betrachtungsraumes) wird durch die history-basierte Notation deutlich, dass Sequenzen die Grundlage bei der Beschreibung von Testfällen bilden.

Das Ereignis (Event) ist ein weiteres Element bei der Modellierung, das ein nicht zeitlich fest definierbares Ereignis modelliert. Ein Event hat keine zeitliche Ausdehnung und kann sowohl durch die Eingabe (Umgebung zum Testobjekt) oder durch Ausgabe (Testobjekt an Umgebung) erfolgen.

Zusammengefasst bedeutet das, dass ein (bestmöglicher) Testfall eine Menge von

Ein- und Ausgabepaaren in einer Sequenz abbildet, die zeitlich in Beziehung stehen (z.B. Wait), wo Events die Schnittstellen zum Testobjekt definieren und zur Synchronisation dienen, bei dem die Zustände für die Abbildung des kontinuierlichen Verhaltens verwendet werden. Die Auswahl bildet das nichtdeterministische Verhalten ab, bringt jedoch für einen Testfall eine Aufteilung in mehrere Pfade (durch die Gabelung) ein.

In der Literatur wird statt der Eingabe von *actions* gesprochen. Die Ausgabe soll in dieser Arbeit demzufolge *reaction* genannt werden.

Das kontinuierliche Verhalten wird bei der System-Spezifikation in dem Automaten (Sprach-)element *Zustand* abgewickelt, kann jedoch beim Testen sowohl bei der Aktion als auch bei dem Soll-Ergebnis erfolgen, da hier keine Spezifikation des Verhaltens erfolgt.

## 3.6 Der Testfall

### Die Grundelemente

Wie erwähnt besteht ein Testfall aus Aktion und Reaktion, die zu einer Sequenz zusammengefasst werden. Dabei ist die Dauer der Aktion, sowie die Zeit bis zur Reaktion zu berücksichtigen. Ereignisse bilden besondere Aktionen und Reaktionen, da sie keine zeitliche Ausdehnung haben und das zeitliche auftreten unbekannt ist. Zustände sind für die Modellierung des kontinuierlichen Charakters bei hybriden Systemen vorhanden. Zeit und Wertangaben besitzen Toleranzen.Wegen des nichtdeterministischen Verhaltens der Systeme ist eine Auswahl notwendig, sowie Angaben, um die Toleranzen bezüglich Zeit und Wert zusätzlich zu definieren (Constraints).

### Die Ablaufstrukturen

Aus der Modellierung des Systemverhaltens werden die Testfälle abgeleitet. Hierfür gibt es eine Sequenz, die die Pfade in der Prozessalgebra oder den Automaten durchläuft.

Weiterhin werden eine Ebene über der Sequenz Automaten herangezogen, um das Systemverhalten zu testen. Wie im Stand der Technik zu sehen sein wird, werden Automaten, die hybride Systeme abbilden, ebenfalls zum Testen verwendet.

### 3.6.1 Der Testschritt und die Testablauf-Sequenz

In der vorgestellten Definition des Testfalls hieß es: „... die für die Ausführung notwendigen Vorbedingungen, die Menge der Eingabewerte (ein Eingabewert je Parameter des Testobjekts) und die Menge der erwarteten Sollwerte, die Prüfanweisung (wie Eingaben an das Testobjekt übergeben und Sollwerte abzulesen sind) sowie die erwarteten Nachbedingungen“ [IST07], [Iee91].

Vorbedingungen sind Zustände die das Testobjekt einnehmen muss, um den Testfall durchführen zu können. Die Menge der Eingabewerte und die Menge der erwarteten Sollwerte bei dem klassischen Softwaretest sind bei den reaktiven Systemen - wie gezeigt - gekoppelt, so dass ein Tupel aus Eingabe und Ausgabewerten ein Paar darstellt, die in einer Sequenz angeordnet ist.

Die Prüfanweisung für ein deterministisches System ist entweder eine Sequenz oder ein Automat.

Zwei Definitionen werden aus den Erkenntnissen vorheriger Abschnitte für die folgenden Abschnitte und Kapitel eingeführt:

**Definition 3.15 (Testschritt)** *Ein Testschritt besteht aus einem Paar von Eingabe- und Ausgabewerten. Bei Echtzeitverhalten stehen die Eingabe- und Ausgabewerte in zeitlicher Beziehung.*

Im Gegensatz über die Werte wird in [Con04] der Testschritt über die Aufteilung der Zeit eingeführt.

**Definition 3.16 (Testablauf-Sequenz)** *Eine Testablauf-Sequenz ist eine Reihe von Testschritten. Bei Echtzeitverhalten stehen die Testschritte in zeitlicher Beziehung.*

Durch den nichtdeterministischen Charakter sind zusätzlich Toleranzangaben beim zeitlichen Verhalten notwendig [PAD+98].

### 3.6.2 Die Aufteilung der Elemente des Testfalls

Für die Untersuchung des Stands der Technik und für das Konzept ist eine Aufteilung der Elemente des Testfalls sinnvoll. Die Aufteilung basiert auf folgender Grundsatzbetrachtung:

Auf einer deklarativen Ebene wird beschrieben was erreicht werden soll[9]. Im Gegensatz dazu steht die Umsetzung wie es gemacht wird. Ähnlich der objektorientierten Analyse (OOA) und dem objektorientierten Design (OOD) ist der Übergang zwischen der Beschreibung und der Umsetzung fließend.

Zwei Gruppen werden eingeführt, um identifizierte Elemente in Elemente der Beschreibung und Elemente der Implementierung einordnen zu können. Die Elemente werden mit folgenden Fragen zugeordnet:

- Wird das Element verwendet, um eine konkrete Umsetzung einer Idee zu verwirklichen?

  Falls die Frage mit einem Nein zu beantworten ist, ist das Element ein Kandidat für die Gruppe „Testfallbeschreibung“. Anderenfalls ist es ein Element der Gruppe „Testfallimplementierung“.

- Wird das Element verwendet, um ein Zugriff auf das Testsystem zu erlangen?

  Falls die Frage mit einem Nein zu beantworten ist, ist das Element ein Kandidat für die Gruppe „Testfallbeschreibung“. Anderenfalls ist es ein Element der Gruppe „Testfallimplementierung“.

- Dient dieses Element für die Beschreibung eines Testfalls?

  Wird die Frage mit einem 'Ja' beantwortet, so ist es ein Element der „Testfallbeschreibung“. Anderenfalls ist es ein Element der Gruppe „Testfallimplementierung“.

- Wird das Element für die Beschreibung des Ablaufs verwendet?

  Wird die Frage mit einem 'Ja' beantwortet, so wird überprüft, ob das Elemente für die Ablaufbeschreibung der Paradigmen der Testfälle eingesetzt werden kann (siehe vorheriger Abschnitt). In diesem Fall ist das Element ein Element der „Testfallbeschreibung“. Kann das Element zusätzlich für die Beschreibung einer Umsetzung eingesetzt werden, so ist das Element zusätzlich ein Element der Gruppe „Testfallimplementierung“.

Bei den Fragen muss zusätzlich untersucht werden, ob sich ein Element oder eine Gruppe von Elementen der „Testfallimplementierung“ soweit abstrahieren lassen kann bzw. können, dass ein oder mehrere Elemente für die Gruppe der „Testfallbeschreibung“ hervorgehen. Beispiel: *Sync*, *Fork*, *Split*, *Join* sind Sprachelemente für Nebenläufigkeiten und sind Elemente für eine konkrete Umsetzung einer Idee. Hieraus kann durch Abstraktion ein Element *Parallel* für die „Testfallbeschreibung“ abgeleitet werden, welches die eigentliche Idee hinter den vier Sprachelementen abbildet.

[9] Im folgenden Kapitel 4, S. 97 wird hierfür der Begriff „Testabsicht“ eingeführt.

Die Aufteilung ist aufgrund des fliessenden Übergangs nicht disjunkt (Abbildung 3.8), d.h. Elemente der Testfallbeschreibung können Elemente der Testfallimplementierung sein. Deswegen wird in der folgenden Aufteilung Elemente die zu beiden Gruppen gehören können in die Gruppe der Testfallbeschreibung zugeordnet. Etwaige andere Darstellungen bzw. Umsetzungen wird dabei nur dann betrachtet, wenn dies sinnvoll erscheint.

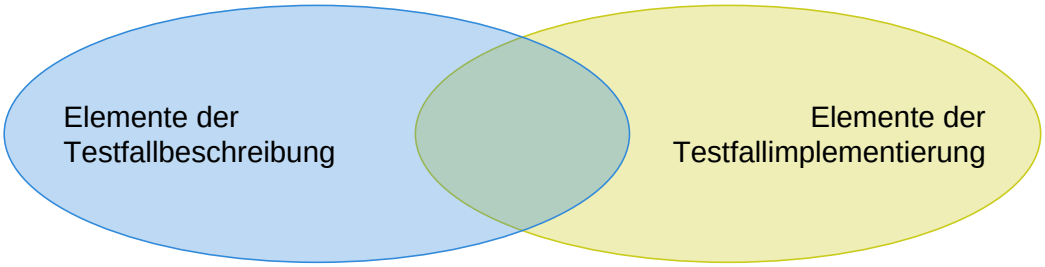



**Abbildung 3.8:** Die Gruppe der Testfallbeschreibung und die der Testfallimplementierung sind nicht disjunkt

### 3.6.3 Die Elemente des Testfalls

Die Systemspezifikation und Testfallimplementierung haben eine eindeutige Auswirkung auf die Testfallbeschreibung.

Wie in den vorherigen Abschnitten erläutert, werden folgende Elemente aus der Systemspezifikation extrahiert:

- Aktion
- Soll-Ergebnis
- Ereignis
- Zustand
- zeitliche Beziehung
- Toleranzangabe

Aus der Testfallimplementierung sind entnommen:

- Zugriffsmöglichkeiten auf die Größen des Testsystems
- Zugriffsmöglichkeiten auf die Bussysteme
- Möglichkeit zur Beschreibung der Nachrichten
- Variablen und Konstanten

- Programmkonstrukte wie Schleifen und Verzweigungen
- Operationen zur Datenverarbeitung
- Elemente zur automatisierten Ergebnisbeurteilung
- Sprachmittel zur Integration der Zeit

Weitere Auswirkungen auf die Testfallbeschreibung ergeben sich aus dem Testprozess.

- Bestimmung der Prioritäten für Ergebnisse (sowohl Soll-Ergebnis als auch unerwartetes Ergebnis).
- Elemente für die Ergebnisauswertung.

Bekannt ist, dass ein Testfall

- Vorbedingungen,
- Nachbedingungen und
- Testdaten

besitzt (vgl. Abschnitt 2.5, S. 29).

Abbildung 3.9 zeigt den Ursprung der Sprachelemente zu den umliegenden Strukturen der Testfallbeschreibung. Der Einfluss des Testprozesses wird bei den Anforderungen an eine Testfallbeschreibung in Kapitel 4, S. 97 dargestellt. Sie ist hier der Vollständigkeit halber eingezeichnet.

### Konkretisierung einiger Sprachelemente

Anforderungen an die Sprachelemente wie Kontrollstrukturen, zeitliche Angaben und Weitere von oben werden im Folgenden konkretisiert.

- Kontrollstrukturen

  Die Sequenz bildet die Grundstruktur (den Ablauf) eines Testfalls. Verzweigungen sind notwendig um nichtdeterministisches Systemverhalten abzudecken. Neben diesen werden für die Testfallbeschreibung Wiederholungen und Iteratoren definiert, die aus der Analysearbeit aktueller Testfallbeschreibungen identifiziert sind.

  Für die Testfallbeschreibung gilt somit: Sequenzen, Wiederholungen, Verzweigungen, Iterationen.

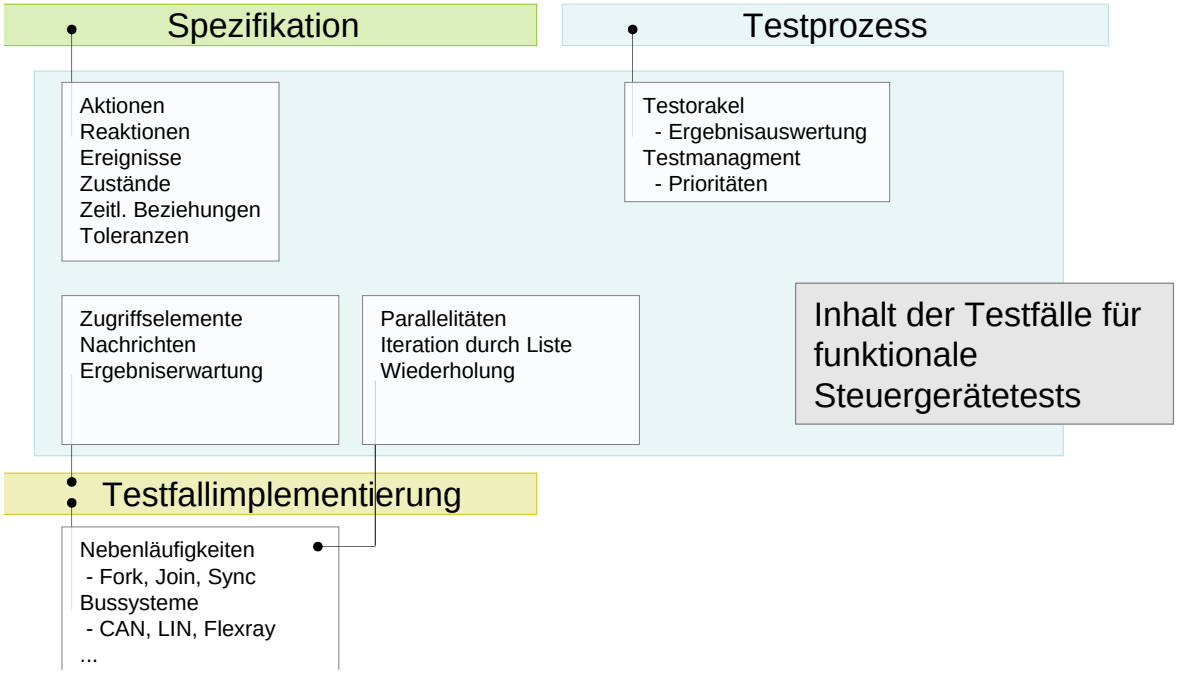



**Abbildung 3.9:** Sprachelemente aus der Umgebung der Testfallentwicklung

- Zeitbedingungen:

  In [LG98] werden vier Möglichkeiten für Zeitbedingungen gezeigt: Angabe eines genauen Zeitpunktes (a), eines spätesten Zeitpunktes (b), eines frühesten Zeitpunktes (c) und Angabe eines Zeitintervalls (d) (siehe Abbildung 3.10).

  Diese vier Eigenschaften decken die Taskeigenschaften bei Echtzeitsystemen wie in [SZ04] beschrieben ab (Aktivierungszeitpunkt, Deadline-Zeitpunkt, Response-Zeit, relative Deadline und Ausführungsrate). Ebenfalls von Bedeutung ist, ob man absolute und relative Zeitbedingungen angeben kann.

  Zusätzlich zu den Zeitbedingungen sind Warteanweisungen mit Zeitangabe (z.B. *wait* in Visual Basic oder *sleep* in Python) relevant. In Kombination mit den Kontrollflusstrukturen können somit periodische und aperiodische Ereignisse überprüft werden (siehe nächstes Kriterium).

- Echtzeitfähigkeit

  Wie in Kapitel 2 aufgezeigt müssen Echtzeitsysteme die Anforderungen Rechtzeitigkeit, Gleichzeitigkeit und spontane Reaktion auf Ereignisse erfüllen.

  Die Rechtzeitigkeit wird durch die Zeitbedingungen erfüllt. Mit den vier Angegebenen Zeitbedingungen kann jede echtzeitrelevante Eigenschaft eines Echtzeitsystems überprüft werden (vgl. auch [Koy91] über die Relation von Zeitpunkten als Kerngedanken für die Spezifikation von Echtzeiteigenschaften). Des Weiteren kommen Eigenschaften des Wiederholens hinzu: So können Zeitbedingungen neben absolut und realtiv auch periodisch und aperiodisch sein

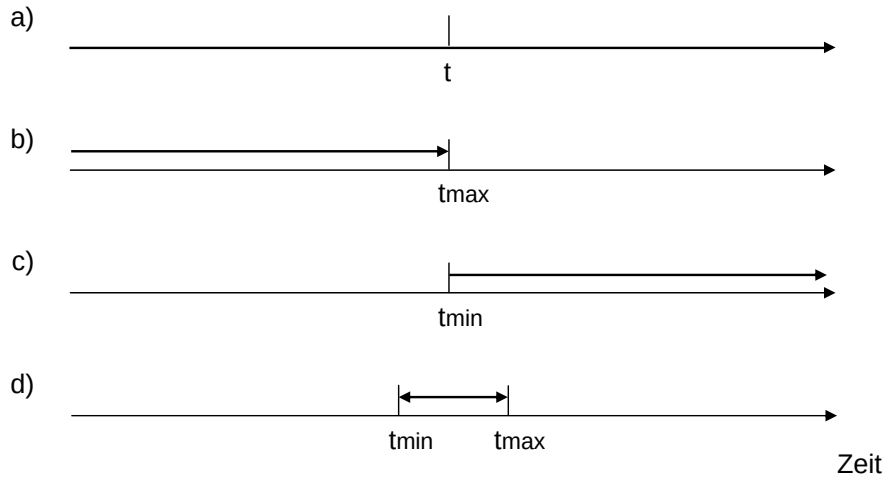



**Abbildung 3.10:** Möglichkeiten zur Angabe von Zeitbedingungen

[WB05].

Die Gleichzeitigkeit wird durch die Parallelität abgedeckt.

Die spontane Reaktion auf Ereignisse wird zwar bei einem Echtzeitsystem gefordert, ist aber beim Testen irrelevant, da die Ereignisse kontrolliert 'spontan' ausgelöst werden um die Reaktion des Systems überprüfen zu können.

- Testobjekt und Testumgebung (aus Kapitel 5, S. 123)

  Angaben zum Testobjekt (z.B. Schnittstellen) und zu der Testumgebung (z.B. beteiligte Komponenten) sind in einigen Ansätzen im Stand der Technik zu finden. Sie erleichtern die Verständlichkeit des Testfalls, obwohl sie nicht zwingend für die Beschreibung notwendig sind (wie in z.B. UML2TP). Für die Programmierung (wie in z.B. TTCN3) werden diese Angaben zur Deklaration bzw. Definition und damit zur weiteren Benutzung im Code verwendet. Um den Stand der Technik und den eigenen Entwurf aus der Sicht der Testfallbeschreibung bewerten zu können, werden beide Angaben zur Beschreibung berücksichtigt.

- Signalbeschreibung bzw. Wertbeschreibung

  Wie in Abschnitt 2.6 wurde, S. 35 herausgestellt, werden alle Signale (damit auch Werte) durch die Testsysteme einheitlich durch die Steuerungseinheit dargestellt. Die Signalbeschreibung für die Stimulation oder Definition der Soll-Reaktion muss auf der Ebene der Testfallbeschreibung auf zwei Arten erfolgen: Einmal durch einen Freitext, um das Signal zu benennen bzw. zu beschreiben (Wertbeschreibung) und zum anderen durch die genaue Definition des Signalverlaufs für die Spezifikation des Signals (Signalbeschreibung). Die Wertbeschreibung ist somit die abstrakte Darstellung der konkreten Signalbeschreibung bzw. -spezifikation.

### Sprachelemente der Testfallbeschreibung

Damit ergeben sich folgende Sprachelemente für die Testfallbeschreibung.

- Testobjekt
  - Angaben zum Testobjekt
- Testumgebung
  - beteiligte Komponenten
- Testschritt
  - Aktion
  - Ereignis
  - Reaktion
  - Zustand
- Soll-Reaktion
  - Angabe von Toleranzen
- Vor- und Nachbedingungen
  - Vorbedingung
  - Nachbedingung
- Testdaten
  - Wertbeschreibung
  - Signalbeschreibung
  - Liste
  - kontinuierliche Werte/Signale
- Nachrichtenaustausch
  - Senden von Nachrichten
  - Empfangen von Nachrichten
- zeitliche Aspekte
  - früheste Zeitangabe
  - späteste Zeitangabe
  - Zeitangabe über Dauer (Intervall)
  - Periodenangabe
  - absolute Zeitangabe
  - relative Zeitangabe
- höhere Kontrollstrukturen
  - Wiederholung

- Iteration durch Liste
- Warteanweisung

- Parallelitäten
  - Beschreibungselement für parallele Abläufe

#### Sprachelemente der Testfallimplementierung

Für die Testfallimplementierung werden folgende Sprachelemente betrachtet:

- Testobjekt
  - Schnittstellendefinition
- Kontrollstrukturen
  - Sequenzen
  - Schleifen
  - Verzweigungen
- Datentypen / Typisierung
  - einfache Datentypen
  - komplexe Datentypen
  - Typisierung
- Protokollierungs-Mechanismen
  - Aufzeichnungen
  - Protokollierung
- Operatoren
  - arithmetische Operatoren
  - relationale Operatoren
  - logische Operatoren
- Zeit-Operationen
  - Start / Stop von Timern
  - Ablauf von Timern
  - Zeitmessung
- Stream Handling
  - Operationen auf Streams
- Operatoren für Nebenläufigkeit
  - Definition von Nebenläufigkeit

- Fork, Join, Sync

- Testergebnis
  - Verdicts (Pass, fail, abort, error, undef, ...)

### 3.6.4 Die Ablaufstrukturen des Testfalls

Die Ablaufstrukturen sind auf die Paradigmen der Modellierung der Systemverhaltensmodelle zurückzuführen, so dass - wie in den vorherigen Abschnitten dargestellt - ein Testfall entweder aus einer Sequenz oder einem Automaten besteht. So soll im Folgenden - Analog zur Modellierung des Systemverhaltens - das Paradigma eines Testfalls deren Ablaufstruktur bedeuten.

- Testablauf-Sequenz

  Eine Sequenz ist die einfachste Art eines Testfalls. Hier werden Eingabedaten der Reihe nach ausgeführt und die Ausgaben bewertet. Bei Echtzeitverhalten stehen die Ein- und Ausgabedaten in zeitlicher Beziehung zueinander.

  Bei nichtdeterministischem Verhalten müssen Testfälle durch Verzweigungen definiert werden.

- Testablauf-Automat

  Eine andere Möglichkeit eines Testfalls besteht in der Abbildung der Ablaufstruktur in Automaten. Die Testsemantik wird dadurch erreicht, dass der Zustand für die Eingabe (Stimulation) benutzt wird und die Ausgabe in den Transitionen behandelt wird. Zeitliche Aspekte werden ebenfalls in den Transitionen spezifiziert.

#### Die Paradigmen des Testfalls

Neben den Sprachelementen für die Testfallbeschreibung und -implementierung wird der Aufbau eines Testfalls im Stand der Technik betrachtet.

- Aufbau eines Testfalls
  - Sequenzen
  - Automaten

# Kapitel 4

# Anforderungen an die Testfallbeschreibung

Offensichtlich benötigt eine Testfallbeschreibung die Elemente, die im vorherigen Kapitel erarbeitet wurden. Zusätzlich muss eine Testfallbeschreibung aus der Anwendung bedingt bestimmte Anforderungen erfüllen. In diesem Kapitel werden diese Anforderungen an eine Testfallbeschreibung hergeleitet. Weiterhin werden aus diesen Anforderungen Bewertungskriterien für den Stand der Technik abgeleitet und diese im gleichen Sinne für die Validierung des eigenen Ansatzes verwendet.

## 4.1 Methode und Vorgehen zur Anforderungsermittlung

**Definition 4.1 (Anforderung)** *1. Eine Bedingung oder Fähigkeit, die von einer Person zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.*

*2. Eine Bedingung oder Fähigkeit, die eine Software erfüllen oder besitzen muss, um einen Vertrag, eine Norm oder ein anderes, formell bestimmtes Dokument zu erfüllen.*

*3. Eine dokumentierte Repräsentation einer Bedingung oder Fähigkeit wie in (1) oder (2) genannt. [Iee91]* [a]

[a] Übersetzung von [Sch01a] entnommen.

Die Ermittlung der Anforderungen lehnt sich an die in der Literatur zu findende Technik an (vgl. [Rup07]), wie Interviews und Brain Storming.

Die Experten bestehen aus operativ tätigen Testern aus der Automobilindustrie, sowie aus Experten in Gremien, die für Standardisierungen im Umfeld der Testbeschreibung existieren (insbesondere [Tes09]).

Um die Anforderungen zu ermitteln, werden Interviews geführt. In den Interviews wird einzeln über Anwendungsfälle gesprochen. Meist vermischen sich jedoch Anwendungsfälle aus nahe angrenzenden Bereichen mit den Anwendungsfällen für die Testbeschreibung. So ist z.B. die Herleitung eines Testfalls[1] nicht Bestandteil dieser Arbeit, hat aber eine enge Bindung zur Beschreibung eines Testfalls.

Öfters treten in den Anwendungsfällen auch Anforderungen direkt auf. So waren die meisten Antworten bei den Interviews in folgender Form:

- Beschreibung soll klar und eindeutig sein.
- Beschreibung soll keinen Interpretationsspielraum zulassen.
- Beschreibung soll für Nicht-Test-Experten einfach zu verstehen sein.
- Beschreibung soll entweder direkt lauffähig sein oder eineindeutig in eine lauffähige Implementierung überführbar sein.

Die meisten dieser Anforderungen sind indirekte Anforderungen, d.h. sie werden genannt, obwohl sich die Anforderung bei genauer Betrachtung auf eine andere grundlegende Anforderung zurückzuführen ist[2]. Hier ist das Herausholen bzw. Herauslocken der Anforderungen (*elicitation*) eine wesentliche Aufgabe.

Grundsätzlich gilt, dass ein Testfallbeschreibungsmittel alle benötigten Beschreibungselemente für eine Testfallbeschreibung dem Test-Designer zur Verfügung stellen soll, wie sie im Unterabschnitt 3.6.3 dargestellt ist, so dass dieser seine Gedanken wohl formulieren kann. Wie diese Beschreibung eines Testfalls festgehalten wird, so dass sie bestimmten Anforderungen genügt, ist Ziel dieser Arbeit.

Eine Anspielung an „doing right things“ und „doing things right“, welches aus dem Software Engineering bekannt ist, soll dies verdeutlichen: **Bei dem Testfallbeschreibungsmittel geht es mehr um „describe the things right“ als um „describe the right things“**. Mit dieser Maxime werden im Folgenden die Anwendungsfälle betrachtet, bewertet und Anforderungen an eine Testfallbeschreibung bestimmt.

Weitere Anforderungen ergeben sich aus dem Umfeld der Testfallbeschreibung (siehe vorheriges Kapitel über das Umfeld der Testfallspezifikation in der Dokumentenlandschaft).

[1] In der Literatur bekannt als Test Design Techniken, Testverfahren oder Prüfverfahren.
[2] Mehr dazu in der Analyse der Anwendungsfälle.

## Funktionale und nichtfunktionale Anforderungen

Die Anforderungen an eine Testbeschreibung werden in Anlehnung an die Softwareentwicklung (vgl. [AMBD04]) in zwei Klassen unterteilt:

- Funktionale Anforderungen
- Nichtfunktionale Anforderungen

Bei den funktionalen Anforderungen werden die Anforderungen an die Testbeschreibung dargestellt, d.h. die Frage „was muss die Sprache darstellen können?“ wird beantwortet. Bei den nichtfunktionalen Anforderungen werden Aspekte betrachtet, die nicht für die Beschreibung gebraucht werden, aber für den Einsatzort und -zweck notwendig sind.

**Definition 4.2 (Funktionale Anforderungen)** *Anforderung, die ein funktionales Verhalten spezifiziert, die ein System oder eine Systemkomponente ausführen können muss [...] [IST07]*

**Definition 4.3 (Nichtfunktionale Anforderungen)** *Eine Anforderung welche sich nicht auf die Funktionalität des Systems bezieht sondern auf Merkmale wie Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit.Siehe auch Qualitätsziel. [IST07]*

## Überblick über das Vorgehen

Abbildung 4.1 zeigt die Kette für die Herleitung der Anforderungen und Bewertungskriterien. Aus den ermittelten Anwendungsfällen werden Anforderungen an die Testfallbeschreibung abgeleitet. Eine anschliessende Analyse und Bewertung der Anforderungen endet bei der Ableitung von Bewertungskriterien mit deren Hilfe der Erfüllungsgrad von Anforderungen bei Verifikation gemessen werden kann.

Im ersten Schritt wird jedoch auf die oben erwähnte Maxime „describe the things right“ eingegangen. Hierfür wird die Testfallbeschreibung als Spezifikation der Testfälle angesehen und mit der Spezifikation, die für die Definition Testobjekte verwendet wird, verglichen.

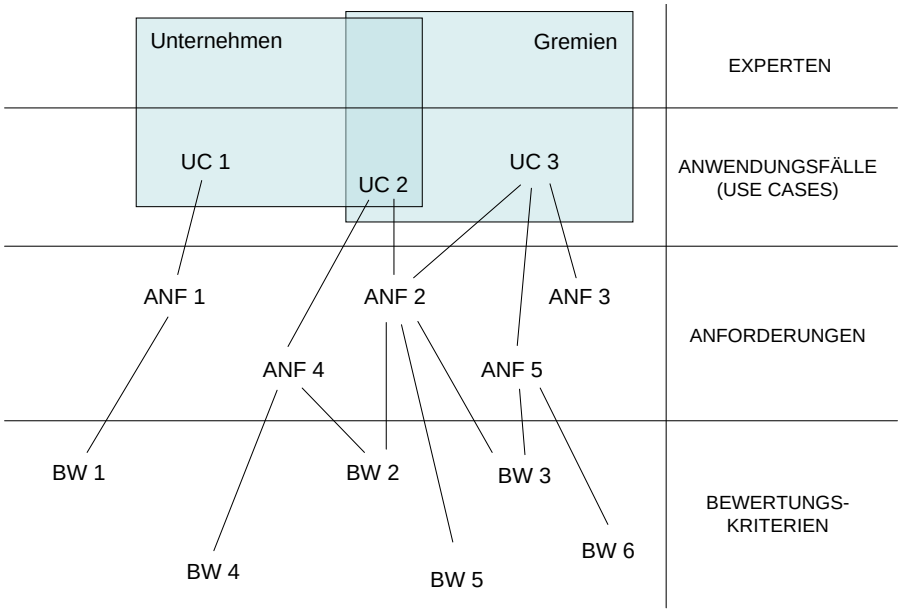



**Abbildung 4.1:** Schema zur Herleitung der Anforderungen und Bewertungskriterien

## 4.2 Die Testfallbeschreibung: Describe the things right

Für die Testfallspezifikation wurde die Definition eingeführt, dass sie ein Dokument ist, welches mehrere Testfälle für ein Testobjekt spezifiziert (vgl. Definition 3.5, S. 64). Unklar ist, wie die Spezifikation eines einzelnen Testfalls, die Beschreibung eines Testfalls (Testfallbeschreibung) aussieht und welche Definition sie besitzt.

Im Folgenden wird der Begriff Testfall-Spezifikation synonym zum Begriff Testfallbeschreibung verwendet, d.h. für die Spezifikation eines einzelnen Testfalls.

Bei der Testfall-Spezifikation steht die Testabsicht im Vordergrund. Um die Testabsicht vorstellen zu können, wird die Definition der Testfall-Spezifikation über den Weg der Spezifikation eingeführt.

### 4.2.1 Die Testabsicht: Ein Vergleich der Testfall-Spezifikation mit der System-Spezifikation

Der Hauptgrund jeder Spezifikation ist - auf einer deklarativen Ebene - das zu beschreiben, was erreicht werden soll, ohne dabei die Umsetzung (das wie) zu beschreiben [Sre02]. So ist auch bei der Testfall-Spezifikation die Absicht im Vordergrund, wobei die Umsetzung dieser Absicht das implementierte Testskript übernimmt. In

diesem Abschnitt soll eine Abgrenzung von Testfällen und Testskripten erfolgen und aufgezeigt werden welchen Inhalt und Umfang die Testfall-Spezifikation besitzen soll.

Die Rolle der Testabsicht zur Testfall-Spezifikation wird in [LBB+01] wie folgt dargestellt: „'The specification of test cases is a natural counter-part to the specification of programs. Specification provides a higher level of abstraction to the developer and it may be sensible to capture the essence of a test in a short and abstract description, before starting its design. Test purposes play such a role."

Die Testabsicht (Test purpose) wird hier definiert als: „description of a precise goal of the test case, in terms of exercising a particular execution path or verifying the compliance with a specific requirement.". In [Cal07] wird sie bezeichnet mit: „The test purpose is the specification of a test scenario." (Mit Testszenario ist hier der Testfall gemeint)

Die einfachste Art die Testabsicht zu spezifizieren ist eine Verbindung zur Spezifikation: Der Testfall wird an die Anforderungen der Spezifikation verknüpft und in Prosa die Testabsicht spezifiziert.

Eine Testfall-Spezifikation muss demzufolge auf einer abstrakten Ebene, das WAS beschreiben, d.h. die Frage beantworten, was man mit dem Testfall erreichen will. Sie muss verständlich sein und setzt weit über der Testfallimplementierung auf.

Bei Testfallherleitungen wie der Äquivalenzklasse ist das Testverfahren selbst die Testfall-Spezifikation.

Wird von einem mentalen Modell die Testfall-Spezifikation abgeleitet, so bildet das Modell die Grundlage für die Testfall-Spezifikation. Beispiel: Die Anforderungen sind als (hybrider) Automat vorhanden. Die Testabsicht ist hier ein Pfad der an einem Startknoten beginnt und zu einem Endknoten führt. Wie diese Testabsicht verwirklicht wird, wird durch den Testfall definiert. Die Umsetzung dieses Testfalls ist die Testfallimplementierung.

Folgender Vergleich in [UPL06] zeigt den Unterschied zur Testfallimplementierung: „The difference between a test case specification and a test suite is that the former is intensional ('fruit') while the latter is extensional ('apples, oranges, ...'): all tests are explicitly enumerated."

Die Zuordnung zu der Dokumentenlandschaft ist wie folgt: Die Testfall-Spezifikation beinhaltet die Ziele, Eingaben, Testaktionen, vorausgesagte Ergebnisse und Vorbedingungen aus der Testfallspezifikation und die Folge von Schritten aus der Testablaufspezifikation, die allerdings nicht Testmittelgebunden ist, sondern für die Durchführung des Testfalls notwendig ist.

Die Forderung nach der Testabsicht als Zwischenschritt zum Testfall wird in [Sax08] wie folgt dargestellt: „Es ist zumindest eine einfache High-Level Spezifikation des Testfalls in verständlicher Sprache erforderlich, mit deren Hilfe der Sinn eines Testfalls erfasst werden kann.“

### 4.2.2 Zusammenfassung

Die Testfall-Spezifikation ist zwischen dem Testfall und der Spezifikation einzuordnen, wo die Beschreibung der Testabsicht im Vordergrund steht.

Bei Basis-Testverfahren kann das Testverfahren selbst eine Testfall-Spezifikation darstellen. Bei der Testfallherleitung mittels Expertenwissen, muss die Testabsicht beschrieben werden.

Die im Folgenden aufgezeigten Anwendungsfälle und die daraus abgeleiteten Anforderungen basieren auf der Grundidee das richtige Beschreibungsmittel zu finden (describe the things right), bei dem die Testabsicht im Vordergrund steht.
Testverfahren, die ebenfalls einen offensichtlichen Einfluss auf die Beschreibung der Testfälle haben, werden nicht betrachtet.

## 4.3 Ableitung von Anforderungen aus Anwendungsfällen

Die Anwendungsfälle basieren auf der Nutzung einer Testfallbeschreibung in unterschiedlichen Situationen und basieren auf der Idee der Softwareentwicklung ein System durch seine beabsichtigte Anwendung zu analysieren. Dabei wird eine akkurate Abdeckung aller möglichen Situationen der Nutzung angestrebt. Jede einzelne Nutzung wird in unterschiedlichen Szenarien durchgespielt um mögliche Anforderungen abzuleiten. Dabei sind die Kommunikationspartner das System auf der einen Seite und alle extern auf das System zugreifenden Aktoren auf der anderen Seite.

Ähnlich diesem Vorgehen aus der Softwareentwicklung wird die Testfallbeschreibung als ein abgeschlossenes System angesehen. Sein Hauptnutzer ist der Test-Designer (siehe Abschnitt 2.5.3, S. 32 und Kapitel 3, S. 59).

Im Folgenden werden die Anwendungsfälle, die wie oben beschrieben ermittelt wurden, dargestellt.

**Definition 4.4 (Use Case / Anwendungsfall)** *Ein Anwendungsfall beschreibt eine (durch genau einen Akteur angestoßene) Reihe von erkennbaren Aktionen, die ein System ausführt und die zu einem konkretem Ergebnis mit Wert für den Handelnden (Akteur) führt. [IST07]*

## Schema für die Vorstellung

Die Anwendungsfälle werden kurz dargestellt. Beispiel-Szenarien dienen zur Verdeutlichung sofern notwendig. Aus den Anwendungsfällen werden Anforderungen abgeleitet, die eine Testbeschreibung erfüllen muss. Jede Anforderung wird soweit erforderlich mit einer kurzen Beschreibung erläutert und - falls die Ableitung nicht offensichtlich ist - begründet.

### 4.3.1 Testfallbeschreibung wiederverwenden

#### Darstellung

Bei diesem Anwendungsfall liegt der Wunsch, den Aufwand bei der Testfallerstellung zu reduzieren. Hier ist die Voraussetzung, dass wiederverwendbare Bestandteile bei den Testfällen vorhanden sind. Wie auch in anderen Anwendungsfällen ist hier eine enge Kopplung an die Herleitung der Testfälle vorhanden, denn die Wiederverwendbarkeit hängt stark von dem Engineering der Testfälle, dem Testsystem auf dem sie wiederverwendet werden sollen, und anderen Gesichtspunkten (z.B. Voraussetzung, dass die System-Anforderungen ähnlich überprüfbar sind) ab. Deswegen ist aus der Sicht der Beschreibung eine Fokussierung des Anwendungsfalls notwendig. Die mehrmalige Verwendung erfolgt konkret durch immer wiederkehrende Teile, die Verwendung von änderbaren Werten (z.B. Testfall durchlaufen mit 5, 10 und 14 Volt) oder durch Schablonen (Templates). Eine Schablone ist ein vorgefertigtes Gerüst und wird für sich wiederholende Strukturen definiert.

#### Beispiel-Szenarien

- Ein Testfall soll jeweils mit den Batteriespannungen 5, 10 und 14 Volt durchlaufen werden.
- Je nach Variante des Steuergerätes wird die Blinkerfrequenz unterschiedlich überprüft.

- Das Grundgerüst eines Testfalls ist durch den Prozess im Projekt festgelegt, an den sich die Test-Designer halten müssen.

#### Abgeleitete Anforderungen

Die Anforderungen ergeben sich aus der Einschränkung des Anwendungsfalls.

- Verweise auf Testfälle oder auf in sich abgeschlossene Teile von Testfällen sollen möglich sein. (funktionale Anforderung)

  Beinhaltet hierarchischen Aufbau von Testfällen.

- Platzhalter in den Testfällen sollen möglich sein. (funktionale Anforderung)

  Beinhaltet nicht-typisierte Variablen bzw. Parameterangaben.

  Parameter oder Platzhalter ermöglichen eine auf bestimmte Werte ungebundene Beschreibung, so dass die Beschreibung mit Parametrierung wiederverwendet werden kann.

- Schablonen (Templates) in der Testfallbeschreibung sollen möglich sein. (funktionale Anforderung)

  Ebenso sind Templates für die Wiederverwendung von sich wiederholenden Teilen sinnvoll.

  Beinhaltet zum Teil die Anforderung auf Platzhalter in den Testfällen.

### 4.3.2 Import und Export von bzw. in Dokumentmanagementsysteme

#### Darstellung

Die Testfallbeschreibung ist eine Information, die in Dokumentmanagementsystemen gelagert werden soll. Hier werden unterschiedliche Mechanismen verwendet von der elektronischen Verarbeitung, bis hin zu Ausdrucken der Beschreibungen. Ein Problem hierbei ist, dass z.B. Hyperlinks in einem Ausdruck nicht mehr ohne weiteres nachvollzogen werden können.
Eine z.B. binäre Darstellung als Ausdruck ist zweifellos nutzlos für die Weiterverarbeitung im Projekt.

#### Beispiel-Szenarien

- Eine graphisch erstellte Testbeschreibung wird textuell in DOORS[3] eingepflegt und kann dort editiert werden. Anschliessend können die so editierten Testbeschreibungen wieder in den graphischen Editor geladen und bearbeitet werden.

#### Abgeleitete Anforderungen

Die Testfallbeschreibung sollt unabhängig von einem Werkzeug veränderbar und darstellbar sein. So kann die Speicherung in einem proprietärem Format erfolgen, solange die Veränderung und Darstellung werkzeugunabhängig erfolgen kann.

- Die Testfallbeschreibung soll eine werkzeugunabhängige Darstellung besitzen. (nicht-funktionale Anforderung)

### 4.3.3 Testfallbeschreibung austauschen

#### Darstellung

Die Testfallbeschreibung soll ausgetauscht werden. Bei dem Austausch steht nicht der Aspekt des Dokumentmanagementsystems im Vordergrund, wie im obigen Anwendungsfall behandelt, sondern die unterschiedlich konfigurierten Testsysteme der beteiligten Partner.

#### Beispiel-Szenarien

- Eine Menge von Testfallbeschreibungen werden zwischen Automobilhersteller und -zulieferer ausgetauscht.
- Zwei Baureihen verwenden ähnliche Steuergeräte mit ähnlichen System-Spezifikationen, so dass die erstellten Testfallbeschreibungen ausgetauscht werden sollen.

[3] Werkzeug für die Verwaltung von Anforderungen und Testfällen.

#### Abgeleitete Anforderungen

Der Wunsch ist, den Testfall auf unterschiedlichen Systemen benutzen zu wollen. Das bedeutet: sind die grundlegenden Abläufe beider Testfälle gleich, so ist die Testfallbeschreibung bei beiden Plattformen lauffähig und die Testfallimplementierung bildet die Ablaufstrukturen auf die konkrete Umsetzung in den Testsystemen ab. Eine andere Möglichkeit ist, dass sich die Abläufe minimal bis maximal unterscheiden. Bei der maximalen Unterscheidung sind zwei unterschiedliche Testfälle notwendig, da keine Gemeinsamkeiten für einen Austausch des Testfalls gefunden werden kann. Bei der minimalen Unterscheidung können Schablonen, Platzhalter und sonstige Mechanismen eingesetzt werden, um den Testfall portabel zu gestalten. Portabilität ist allerdings ein Fall für die Maxime „describing the right things“ und nicht der Maxime „describing the things right“. So ist dieser Anwendungsfall ein Fall für die Herleitung des Testfalls und nicht der Beschreibung dieser.

- Keine Anforderung abgeleitet.

### 4.3.4 Testfallbeschreibung mit System-Anforderungen verknüpfen

#### Darstellung

Eine Verknüpfung zwischen der Testfallbeschreibung und den System-Anforderungen, die durch den Testfall überprüft werden, ist aus Prozesssicht notwendig. So können die Testfälle zu den System-Anforderungen verfolgt werden und umgekehrt.

#### Beispiel-Szenarien

- Der Einsatz eines einheitlichen Dokumentensystems erfordert die Verknüpfung zwischen den beschriebenen Testfällen und den dazugehörigen abgedeckten System-Anforderungen. So kann z.B. eine Statistik über die Testabdeckung der Anforderungen allein über die Verknüpfung erstellt werden.

#### Abgeleitete Anforderungen

- Metainformationen sollen eingebettet werden können. (funktionale Anforderung)

Eine Verknüpfung ist durch einen Verweis in einem übergeordneten Dokument der Testfallbeschreibung möglich. Ebenfalls soll es möglich sein innerhalb der Testfallbeschreibung Informationen über die Verknüpfung ablegen zu können. So ist hier die Anforderung abgeleitet, dass Metainformationen eingebettet werden können.

### 4.3.5 Testfallbeschreibung durchsehen

#### Darstellung

Erstellte Testfallbeschreibungen werden prozessbedingt mehrfach Qualitätskontrollen unterzogen. Dabei ist eine Qualitätssicherung die Durchsicht, ob der Testfall tatsächlich die System-Anforderung überprüft. Dieser Anwendungsfall liegt somit in der Maxime „describe the right things“. Auf die Maxime „describe the things right“ hat es jedoch sofern einen Einfluss, dass aus der Beschreibung der Sinn und Zweck (die Testabsicht, vgl. Kapitel 3, S. 59) erkenntlich ist und weniger die Umsetzung, d.h. die Testfallimplementierung.

Mit der Durchsicht ist ausserdem der Vorgang der Nachvollziehbarkeit gemeint. Die Nachvollziehbarkeit eines Testfalls ist geprägt durch deren Erstellung, kann jedoch durch Sprachelemente unterstützt werden. Deshalb ist die Nachvollziehbarkeit sowohl eine funktionale als auch eine nicht-funktionale Anforderung. Aus der Sicht der Notation muss die Testfallbeschreibung den Inhalt und Umfang eines Testfalls widerspiegeln können.

#### Beispiel-Szenarien

- Der Test-Designer beschreibt welche Funktionalität der Testfall überprüft, während der Implementierer genug Informationen für die Implementierung erhält.
- Der Testmanager bekommt einen Überblick über die Testabdeckung. Gemäß seiner Rollenaufgabe kann er nun Testlücken identifizieren, die Teststrategie anpassen, die Testaktivitäten erhöhen oder andere Maßnahmen ergreifen, die sich aus dem Testplan-Dokument ergeben und für die Beseitigung des Misstandes notwendig sind.

#### Abgeleitete Anforderungen

- Inhalt und Umfang eines Testfalls soll beschrieben werden. (funktionale Anforderung)
- Metainformationen sollen eingebettet werden können. (funktionale Anforderung)

  Ebenfalls ist die Anforderung nach den Metainformationen aus dem vorherigen Anwendungsfall hier notwendig, um bei der Durchsicht die relevanten System-Anforderungen identifizieren zu können.

### 4.3.6 Testfallbeschreibung erstellen

#### Darstellung

Ein offensichtlicher Anwendungsfall ist die Beschreibung von Testfällen. Die Beschreibung von Testfällen wird eingeschränkt auf das funktionale Testen von Steuergeräten im automobilen Bereich. Eine weitere Einschränkung ist die Beschreibung der Testabsicht und nicht der Umsetzung (Testfallimplementierung).

Eine genauere Darstellung dieser Aspekte ist in Kapitel 2, S. 13 bzw. detailierter in Kapitel 3, S. 59 erfolgt.

Wie im vorherigen Kapitel dargestellt haben Steuergeräte und Steuergeräteverbünde im automobilen Bereich spezielle Eigenschaften, die in anderen Domänen in dieser Konstellation nicht vorkommen.

Die Notwendigkeit verschiedenen Steuergeräte-Varianten, ergibt sich aus Gesetzen der Länder, Modellvarianten und Sonderausstattungen bei den Baureihen und der Tatsache, dass die Funktionen sich im Detail unterscheiden, grundsätzlich aber das Gleiche tun.

Testfallimplementierungen beinhalten, neben dem an Testmittel gebundene Informationen, auch solche, die unabhängig von den benutzen Plattformen sind und allgemeine Gültigkeit besitzen, d.h. in jeder Implementierung zu einem Test genau auf eine Weise codiert sind. Bei allen Implementierungen ist die Beziehung der zu setzenden Signale und Auslöseimpulse gleich und damit unabhängig vom Testmittel. Eine einheitliche Testfallbeschreibung muss sequentielle und parallele Beziehungen ermöglichen.

Ebenfalls gleich bei jeder Implementierung ist die Kopplung der Ausgaben des Testobjektes an die Eingaben desselben. Die Kopplung entsteht durch die Wirkungsweise der Steuergeräte: Durch Sensoren werden die Reaktionen der zu steuernden bzw. regelnden Einheit auf die Ausgabewerte ermittelt, die dann als Eingabewerte für die nächste Iteration verwendet werden. Diese Abhängigkeit wird bei den Implementierungen durch Verzweigungen und Schleifen festgehalten. Deshalb muss die Testfallbeschreibung auch diese Konstrukte unterstützen.

Neben den einfachen Formen von sequentiellen und parallelen Beziehungen sind auch zeitliche Aspekte wichtig. Das Auftreten eines Signals in einem Zeitraum, über einen Zeitraum oder zu bestimmten Zeitabständen sind Informationen die eine Implementierung beinhalten können und die unabhängig vom Testmittel sind. Vor allem bei Tests, wo Testmittel wie HIL-Systeme (vgl. Abschnitt 2.6, S. 35) zum Einsatz kommen, bei denen die Wirklichkeit unter Laborbedingungen simuliert wird und die echtzeitfähige Eigenschaften bieten sind zeitliche Aspekte essentiell für die Beschreibung der Testfälle.

Von den zeitlichen Aspekten sind auch die bei der Auswertung relevanten, zu erwartenden Ergebnisse betroffen, d.h. zuletzt sollte die Spezifikation abstrahiert darstellen, welche Ergebnisse aus der Durchführung zu erwarten sind. Diese Reaktionsauswertung - ein Szenario aus dem Testprozess - beinhaltet Soll-Ergebnisse, aber auch die Priorisierung bzw. Fällung eines Verdicts bei unerwarteten Ergebnissen.

#### Beispiel-Szenarien

- Überprüfung, ob die in Deutschland vorgeschriebene Blinkerfrequenz von 1,5 Hz mit einer Toleranz von +- 0,5 Hz eingehalten wird.
- Rechtsverkehr und Linksverkehr beeinflussen das Wirken der Steuergeräte.
- Die Geschwindigkeit des Fahrzeugs wird auf 70km/h gesetzt und anschliessend wird versucht den Aussenspiegel einzufahren.
- Während der Blinker zum Blinken gesetzt wird, werden parallel die Dauer der Blinkfrequenz und die Blinkdauer protokolliert.
- Der Aussenspiegel fährt in 30 Sekunden ein.
- Die Hallsensoren reagieren beim Anschlag des Fensters genau 10 Sekunden lang.
- Überprüfung, ob folgende Situation funktioniert: Die Blinkerfrequenz wird gesetzt und nach 2 Millisekunden wird der Blinker betätigt.
- Überprüfung, ob bei einem Aufprall mit dem Auto der Airbag in maximal 15 Millisekunden aktiviert wird.

- Die Blinkerfrequenz darf nicht kleiner als 700 ms und nicht größer als 800 ms sein.

  Falls das Soll-Ergebnis kleiner als 700 ms ist, soll das Ergebnis eine hohe Priorität für eine Überprüfung bekommen. Ist das Soll-Ergebnis größer als 800 ms wird ebenfalls die Ergebnisbeurteilung höher eingestuft.

#### Abgeleitete Anforderungen

In Kapitel 3, S. 59 wurde der Inhalt und Umfang eines Testfalls für den funktionalen Steuergerätest hergeleitet. Die Anforderung ist somit:

- Inhalt und Umfang eines Testfalls soll beschrieben werden können. (funktionale Anforderung)
- Die Testfallparadigmen für funktionale Steuergerätetests sollen unterstützt werden. (funktionale Anforderung)
- Reaktionsauswertung soll vorhanden sein. (funktionale Anforderung)

### 4.3.7 Testfallimplementierung aus der Testfallbeschreibung ableiten

#### Darstellung

Eine Analyse dieses Anwendungsfalls zeigt, dass der Grund für die Ableitung der Testfallimplementierung aus der Testfallbeschreibung darin liegt, die Kommunikation der beteiligten Partner bei der Erstellung der Testfälle so weit wie möglich zu reduzieren, so dass die Rückfragen seitens des Testers, der die Testfälle implementiert, auf ein Minimum reduziert wird.

Die Ableitung der Testfallimplementierung aus der Testfallbeschreibung kann manuell oder automatisiert ablaufen. Wichtig bei diesem Anwendungsfall ist, dass die Testfallbeschreibung die notwendigen Informationen für die Ableitung eines Testfall bereitstellen muss.

#### Beispiel-Szenarien

- Eine halbautomatisierte Testfallimplementierung wird auf Basis der beschriebenen Testfälle erstellt.

- Der Tester programmiert Testskripte auf Basis der Testfallbeschreibungen (manuelle Testfallimplementierung).

#### Abgeleitete Anforderungen

Um die gewünschte Reduzierung der Kommunikation durch die Notation („describing the things right“) aufzugreifen, bedarf es einer eindeutigen Schreibweise, welches keine Interpretationen erlaubt.
Dieses kann durch einen vorgegebenen Workflow erreicht werden, aber nur bedingt durch die Notation an sich. Beispiel: Die Überführung der Testfallbeschreibung in die Testfallimplementierung ist vorgeschrieben.

- Die Testfallbeschreibung soll formal genug sein. (funktionale Anforderung)

  Der Formalismus gibt einen Aufschluss über die Eindeutigkeit der Beschreibung. Jedoch bringt es Einschränkungen in der Ausdrucksmächtigkeit mit sich. So soll die Beschreibung eine formale Syntax besitzen ohne die Forderung nach einer formalen Semantik. Dadurch besitzt der Testfall eine eindeutige Schreibweise, ohne die Ausdrucksmächtigkeit zu verringern.

- Ein Workflow für die Ableitung der Testfallimplementierungen soll vorhanden sein. (nicht-funktionale Anforderung)

### 4.3.8 Testfallbeschreibung werkzeugunterstützt eingeben

#### Darstellung

Wie in den obigen Anwendungsfällen, ist auch dieser durch eine Reduzierung des Aufwands bei der Beschreibung der Testfälle begründet. Die werkzeugunterstützte Beschreibung erleichtert dem Test-Designer die Erstellung der Testfälle und die Konzentration auf das Wesentliche: dem Inhalt des Testfalls. Etwaige vorgegebene Strukturen bei der Beschreibung werden durch das Werkzeug unterstützend vorgegeben und Fehler bei der Syntax der Beschreibung vermieden. Weiterhin wird durch eine werkzeugunterstützte Eingabe die Handhabung der Testfälle erleichtert wie z.B. Verweise auf vorhandene Testfälle bei der Erstellung eines Testfalls.

#### Beispiel-Szenarien

- Der Test-Designer wird bei der Eingabe der Testfallbeschreibung durch das Werkzeug über mögliche Eingabe, fehlende Angaben und Querverweise informiert.
- Der Test-Designer extrahiert automatisiert Informationen aus den Testbasen mittels des Eingabewerkzeugs.

#### Abgeleitete Anforderungen

- Werkzeugunterstützung bei der Beschreibung soll vorhanden sein. (nicht-funktionale Anforderung)

### 4.3.9 Lesbare bzw. verständliche und eindeutige Testfälle beschreiben

#### Darstellung

Ein sehr oft diskutierter Anwendungsfall ist die Erstellung von lesbaren und verständlichen Testfällen. Eine Analyse der Gründe für die Nennung dieses Anwendungsfalls zeigt, dass die Lesbarkeit durch die fast kryptisch erscheinende Testfallimplementierung in einer Programmiersprache, die Verständlichkeit durch eine unübersichtliche Darstellung von Testfällen in tabellen-basierter Form, bedingt sind. Zu dieser Kategorie gehört auch die Forderung nach der Eindeutigkeit, bedingt durch die natürlich sprachliche Beschreibung von Testfällen.

Lesbarkeit und Verständlichkeit sind kognitive Aspekte der Testfallbeschreibung. Die Herleitung der Testfälle spielt eine entscheidende Rolle bei der Erstellung von verständlichen Testfällen.

Neben dem Testentwickler und Testimplementierer verwenden noch weitere Rollen die Testfallbeschreibung. Dazu gehören die Steuergeräte-Verantwortlichen und -Entwickler, die Software-Verantwortlichen und -Entwickler und weitere, die auf Grund des Entwicklungs- und Testprozesses sich einen Überblick über die Tests verschaffen müssen (vgl. Abschnitt 2.5, S. 29). Die benötigte Information aus einer Testfallbeschreibung dient diesen als Quelle für ihre eigenen Verantwortlichkeits- und Aufgabenbereiche.

Bei der Verständlichkeit müssen die Sprachelemente einen expliziten Charakter haben, d.h. eine Aktion muss als Aktion erkennbar sein, ein Nachrichtenaustausch ebenfalls als ein solcher. Obwohl beide die gleiche Abbildung auf der Testimplementierung beinhalten können (z.B. durch Aufruf einer Methode) wird durch die Redundanz die Verständlichkeit bei der Testbeschreibung erhöht.

Da die Maxime „describing the things right“ gilt, wird die Lesbarkeit und Verständlichkeit rein aus der Perspektive der Notation aufgegriffen. Somit gilt es auf diesem Anwendungsfall folgende Anforderung zu erfüllen: Ein Testfall ist lesbar bzw. verständlich, wenn der Inhalt und Umfang eines Testfalls bei der Beschreibung dargestellt ist. Dadurch sind die einzelnen Schritte eines Testfalls eindeutig zu identifizieren (Aktionen, Reaktionen, Ereignisse, Zustände, Kontrollstrukturen, ...).

Eine andere Anforderung ist die der Skalierbarkeit der Testfallbeschreibung. Je größer die Testfälle sind, desto unübersichtlicher werden sie. Hier sind Struktuierungsmechanismen notwendig. Hierarchisierung, Kapselung, Modularisierung sind hier Stichwörter aus der Programmierung. Bei der Testfallbeschreibung wird die Hierarchisierung gefordert.

#### Beispiel-Szenarien

- Die Testschritte der Testfallbeschreibung sind erkenntlich, so dass z.B. Aktionen, Reaktionen, Kontrollstrukturen eindeutig zuzuordnen sind.

#### Abgeleitete Anforderungen

- Inhalt und Umfang eines Testfalls soll beschrieben werden. (funktionale Anforderung)
- Strukturierungsmechanismen sollen vorhanden sein. (funktionale Anforderung)

### 4.3.10 Einheitliche Testfälle beschreiben

#### Darstellung

Die Forderung nach lesbaren, verständlichen und eindeutigen Testfällen deutet bei einer Analyse auf einen weiteren, wichtigten Aspekt: Es zeigt, dass jeder Test-Designer

eine unterschiedliche Art und Weise bei der Erstellung der Testfallbeschreibung besitzt, je nach Erfahrung, Wissen, Auffassungsgabe und anderen kognitiven Aspekten und folglich auch die Testfälle in der Darstellung, der Verfeinerung und dem Aufbau unterschiedlich ausfallen. Die Homogenität schwindet durch die Heterogenität der Test-Designer.

Ein weiterer Aspekt einheitlicher Testfälle ist die Zweideutigkeit in der natürlichen Sprache. Zweideutig ist eine Spezifikation dann, wenn aus dieser verschiedene - nicht äquivalente - Implementierungen herausgehen können.

Die Testfallbeschreibung wird von unterschiedlichen Rollen verwendet. Hauptanwender sind die Test-Designer und die Testimplementierer, die durch verschiedene Aufgaben, Verantwortungen und Eigenschaften definiert sind. Bei der Kommunikation der beiden Rollen bildet die Testfall-Spezifikation einen wichtigen Kern. Sie muss daher eindeutig sein, so dass in dem Testprozess die Implementierung der vorangehenden Spezifikation genügt.

Die Eindeutigkeit wird in der Literatur (vgl. [Har01b], [Con04]) durch den Grad des Formalismus definiert. Allerdings reicht eine formale Grundlage nicht aus damit der Testimplementierer ein eindeutiges Verständnis erhält, was mit der Testfallbeschreibung gemeint ist.

Die Testbeschreibung muss auf Signalebene überführt werden können und der Weg dorthin muss eindeutig sein. Einfaches Beispiel: Der Testschritt 'Zündung ein' wird durch die Testimplementierung 'setSignal( CAN.Klemme15 = 1' eindeutig umgesetzt. Schwieriger wird das Beispiel, wenn mehrere Zwischenschritte notwendig sind um den Testschritt 'Zündung ein' zu realisieren.

#### Beispiel-Szenarien

- Ein Testentwickler definiert für eine Steuergeräte-Funktionalität Testfälle. Diese Testfallbeschreibung wird von Testimplementierern der unterschiedlichen Testphasen als Grundlage für die Implementierung genommen. Die Ergebnisse bei der Durchführung der Implementierungen sind gleich.

#### Abgeleitete Anforderungen

Zwei Mechanismen des Erstellungsprozesses eines Testfalls haben einen Einfluss auf die Homogenität: Die Herleitung eines Testfalls und die Niederschrift der Beschreibung. Die Herleitung muss einem Workflow folgen, so dass die Test-Designer ein-

heitliche Verfeinerungsgrade und einen einheitlichen Aufbau besitzen. Bei der Niederschrift kann die Notation nur unterstützend die Homogenität wahren. Hier sind feste Vorgaben notwendig, die die Test-Designer soweit einschränken, dass sie ihre Gedanken wohl formulieren können, aber dennoch genug Freiraum bieten, dass sie Testfälle beliebigen Inhalts - im Rahmen der funktionalen Steuergerätes - formulieren können.

- Inhalt und Umfang eines Testfalls soll beschrieben werden können. (funktionale Anforderung)
- Die Testfallbeschreibung soll formal genug sein. (funktionale Anforderung)

### 4.3.11 Zusätzliche Informationen in der Testfallbeschreibung angeben

#### Darstellung

Besonders wegen [Tes09] sowie wegen der Verwendung von Tesfall-Spezifikationen im Testprozess sind zusätzliche Informationen für eine Testfallbeschreibung notwendig, sogenannte Metainformationen. Metainformationen wären beispielsweise: Identifikationsnummer des Testfalls, Autor des Testfalls, eingesetzte Testverfahren zur Herleitung des Testfalls, Priorität des Testfalls, geschätzte Dauer des Testfalls u.ä. relevante Informationen für den Testprozess.

#### Beispiel-Szenarien

- In der Testfallbeschreibung sind Informationen über den Autor, dem Erstellungsdatum, der Version, Priorität und der geschätzten Laufzeit enthalten.

#### Abgeleitete Anforderungen

- Die Testfallbeschreibung soll Metainformationen beinhalten können. (funktionale Anforderung)

## 4.4 Ausgrenzung von Anwendungsfällen

In diesem Abschnitt werden Anwendungsfälle genannt, die kein Anwendungsfall für eine Beschreibung eines Testfalls sind.

### 4.4.1 Testfallbeschreibung ausführen

#### Darstellung

Nicht notwendig ist eine ausführbare Beschreibung der Testfälle. Wie in dem Anwendungsfall 4.3.7 aufgezeigt, soll die Testfallimplementierung von der Beschreibung abgeleitet werden. Hierfür ist keine ausführbare Semantik der Testfallbeschreibung notwendig, wie bei Programmiersprachen es erforderlich ist. Die Testfallbeschreibung zielt primär auf die Beschreibung ab.

### 4.4.2 Testfälle „programmieren“

#### Darstellung

Ein Anwendungsfall der grundsätzlich ausgeschlossen ist, ist die programmierungsnahe Beschreibung der Testfälle. Typische Programmierkonstrukte wie Kontrollstrukturen sind für Ablaufstrukturen wie sie in Testfällen vorkommen notwendig (vgl. Kapitel 3, S. 59). Allerdings ist eine Typisierung (z.B. Integer, Boolean, Double, Float) für eine Beschreibung, bei der die Testabsicht im Vordergrund steht, hinderlich für die Beschreibung der Testabsicht[4].

### 4.4.3 Testfälle testsystemunabhängig und testphasenübergreifend beschreiben

Bei den Interviews wurden in den Anwendungsfall 4.3.1 (Testfallbeschreibung wiederverwenden) zwei Anwendungsfälle reininterpretiert:

- Die Testfall-Spezifikation soll unabhängig von der verwendeten Soft- und Hardware sein.

[4]Mehr dazu in Kapitel Inhalt und Umfang der Testfälle für den funktionalen Steuergerätetest, S. 59

- Weiterhin soll die Testfall-Spezifikation über Testphasen hinweg eingesetzt werden.

Die unterschiedliche Software (Bediensoftware im Bediensystem zur Ausführung der Testskripte) und Hardware (Testequipment im Testsystem) Plattformen, sowie der Einsatz unterschiedlicher Testfallspezifikationen bei OEMs und Zulieferern gepaart mit dem Bedarf der Kommunikation erfordern ein einheitliches Testfallbeschreibungsmittel. Die Folge einer Vereinheitlichung: Die Gebundenheit an Testmittel entfällt, die Prozess-Kommunikation zwischen den beteiligten Rollen wird unterstützt und der Austausch von Informationen zwischen den Herstellern wird vereinfacht.

Anwendungsfälle für testsystemunabhängige Verwendung sind damit eine oft genannte Untergruppe der Wiederverwendbarkeit der Testfallbeschreibung.

Die Verwendung eines einheitlichen Testfallbeschreibungsmittels in allen Testphasen stellt ausserdem sicher, dass die über die Phasen wachsenden Funktionen, die veränderten Testbedingungen und die damit verbundene Art des Testens sukzessive begleitet werden.

Bei den testsystemunabhängigen Testfällen gibt es jedoch keinen Einfluss auf die Testfallbeschreibung. Wie aus Kapitel 3, S. 59 ersichtlich ist zwischen den Testsystemen und der Testfallbeschreibung die Testfallimplementierung. Diese Zwischenschicht ist für die Testsystemunabhängigkeit verantwortlich.

Will man jedoch die Testfälle so gestalten, dass sie auf der Ebene der Testfallbeschreibung unabhängig sind, so ist das ein Fall der Maxime „describe the right things“ und ist im Verantwortungsbereich der Testfallherleitung.

Bei der testphasenübergreifenden Beschreibung gilt: Die Testfälle haben bei jeder Testphase einen unterschiedlichen Fokus (z.B. White-box- vs. Black-box-Tests) und sollten keine Gemeinsamkeiten aufweisen. Eine über die Testphasen hinweg sukzessive Erweiterung eines Testfalls z.B. für ein Modultest zu einem Systemtest würde an der Natur der Sache scheitern.

### 4.4.4 Zusammenfassung

Abbildung 4.2 zeigt eine Zusammenfassung der Anwendungsfälle und die abgeleiteten Anforderungen.

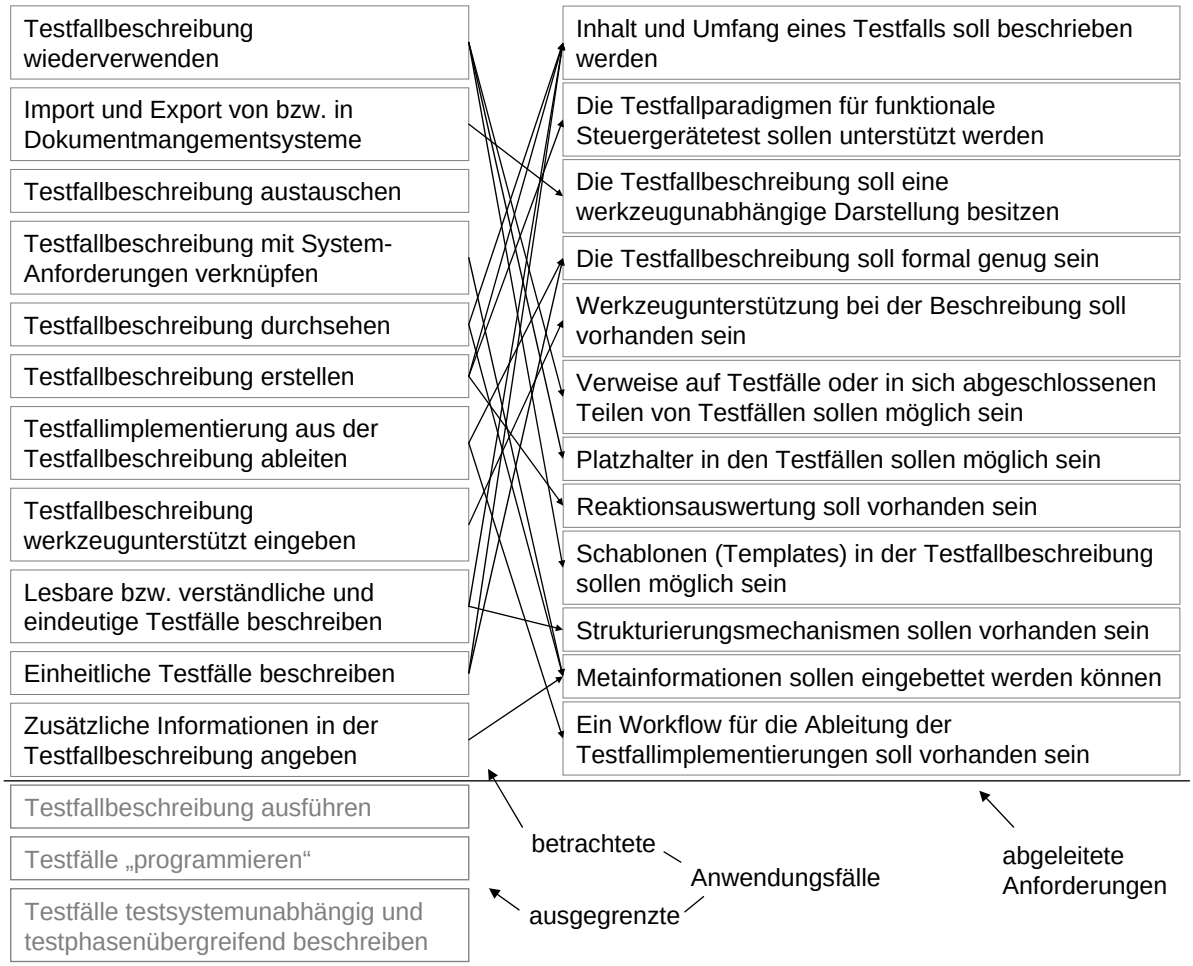



**Abbildung 4.2:** Ableitung der Anforderungen aus den Anwendungsfällen

## 4.5 Erstellung von Bewertungskritierien aus den Anforderungen

Die Anforderungen an eine Testfallbeschreibung werden in folgende Kategorien eingeteilt zusammengefasst.

Hinter jeder Anforderung steht eine Zahl, die angibt, wie oft die Anforderung hergeleitet wurde. Unter jeder Anforderung sind Bewertungskriterien aufgelistet. Aufgrund der offensichtlichen Ableitung der Bewertungskriterien aus den Anforderung wird hier auf eine detaillierte Begründung verzichtet. Jedem gefundenen Kriterium wird ein Punkt zugeordnet, falls nicht anders angegeben.

- Anforderungen aus dem Inhalt und Umfang eines Testfalls

- Inhalt und Umfang eines Testfalls soll beschrieben werden. 4x (funktional)

  Die Bewertungskriterien entsprechen den Sprachelementen der Gruppe „Testfallbeschreibung“, die im Kapitel 3, S. 47 herausgearbeitet wurden. Ein existierendes Sprachelement wird mit einem Punkt bewertet.
- Die Testfallparadigmen für funktionale Steuergerätetests sollen unterstützt werden. (funktional)
  - Sequenz, Automat. Jedem vorhandenen Ablauf wird ein Punkt zugeordnet.

- Anforderungen an die Notation einer Testfallbeschreibung
  - Die Testfallbeschreibung soll eine werkzeugunabhängige Darstellung besitzen. 1x (nicht-funktional)
    - Darstellungsform
      Die Darstellung kann textuell oder graphisch erfolgen. Jeder vorhandenen Darstellung wird ein Punkt zugeordnet.
  - Die Testfallbeschreibung soll formal genug sein. 2x (funktional)
    - Formalismus
      Kriterium kann informell, semi-formal oder formal ausfallen. Die formale Syntax wird bewertet, d.h. wenn die Beschreibung semi-formal (formale Syntax, aber keine formale Semantik) oder formal (formale Syntax und formale Semantik) ist, wird ein Punkt zugeordnet. Anderenfalls wird kein Punkt vergeben.

- Anforderungen an die Beschreibungstechnik
  - Werkzeugunterstützung bei der Beschreibung soll vorhanden sein. 1x (nicht-funktional)
    - Werkzeugunterstützte Eingabe. Eine werkzeugunterstützte Eingabe wird mit einem Punkt bewertet.

- Anforderungen aus dem Prozess und der Rollen
  - Verweise auf Testfälle oder auf in sich abgeschlossene Teile von Testfällen sollen möglich sein. 1x (funktional)
    - Sprachelemente für Verlinkung zu Testfällen bzw. Teilen von Testfällen.
    - Sprachelemente für Hierarchisierung. (siehe auch „Strukturierungsmechanismen sollen vorhanden sein“). Eine Möglichkeit zur Hierarchisierung wird mit einem Punkt bewertet.
  - Platzhalter in den Testfällen sollen möglich sein. 1x (funktional)
    - Sprachelemente für Platzhalter. Eine Möglichkeit für Platzhalter wird mit einem Punkt bewertet.
  - Schablonen (Templates) in der Testfallbeschreibung sollen möglich sein. 1x (funktional)

        - Sprachelemente für Templates. Eine Möglichkeit für Templates wird mit einem Punkt bewertet.
    - Reaktionsauswertung soll vorhanden sein. 1x (funktional)
        - Sprachelemente für Reaktionsauswertung. Eine Möglichkeit zur Reaktionsauswertung wird mit einem Punkt bewertet.
    - Strukturierungsmechanismen sollen vorhanden sein. 1x (funktional)
        - Sprachelemente für Hierarchisierung. Eine Möglichkeit zur Hierarchisierung wird mit einem Punkt bewertet.
    - Metainformationen sollen eingebettet werden können. 2x (funktional)
        - Darstellung von Metainformationen. Eine Möglichkeit zur Einbettung von Metainformationen wird mit einem Punkt bewertet.
    - Ein Workflow für die Ableitung der Testfallimplementierungen soll vorhanden sein. 1x (nicht-funktional)
        - Workflow zur Ableitung von Testfallimplementierungen. Falls ein Workflow zur Ableitung der Testfallimplementierungen vorhanden ist, wird dies mit einem Punkt bewertet.

### Gewichtung der Bewertungskriterien

Die Bewertungskriterien müssen eine Gewichtung erhalten, da die Anforderungen unterschiedliche Wichtigkeit besitzen. Die Zahl für die Häufigkeit der Ableitung der Anforderungen aus den Anwendungsfällen gibt hier einen guten Aufschluss darüber wie wichtig jene Anforderung ist, so dass die Bewertungskriterien bei der Untersuchung dementsprechend mehr gewichtet werden. Dadurch wird eine einfache aber dennoch effektive Gewichtung vorgenommen.

## 4.6 Vergleich mit existierenden Bewertungskriterien

Vorhandene Arbeiten in diesem Umfeld besitzen einen anderen Fokus gegenüber der Maxime „describe the things right“. Dadurch sind die meisten Anforderungen in den Arbeiten auf der Ebene der Testfallimplementierung anzusiedeln.

In [Con04] sind die *darstellbaren Signale* als Kriterium enthalten, die in die vier Bereiche (analog, digital, zeitquantisiert, wertquantisiert) aufgeteilt werden (vgl. mit Abschnitt 2.7.2, S. 42 über die Signalarten).
Weiterhin wird die *Ausdrucksmächtigkeit* als ein Bewertungskriterium herangezogen, die sich wieder der Beschreibung von Signalen widmet.

| Anforderungsgruppe | Anforderungen | enhalten in... |
|---|---|---|
| Inhalt & Umfang Testfall | Testabsicht | vgl. [Har01b] |
| | Testkonfiguration | vgl. [Har01b] |
| | Kontrollstrukturen | vgl. [Har01b], [Hut06] |
| | Zeitbedingungen | vgl. [Hut06], [FK03] |
| | Echtzeitüberprüfung | vgl. [Hut06], [FK03] |
| | Ergebniserwartung | vgl. [Hut06] |
| Notation | Darstellungsform | vgl. [Con04], [FK03] |
| | Formalismus | vgl. [Con04], [FK03] |
| Beschreibungstechnik | Werkzeugunterstützung | vgl. [Con04] |
| | Abstraktionsgrad | vgl. [Hut06] |
| Prozess & Rollen | Metainformationen | - |
| | Methodenunterstützung | vgl. [Con04] |

**Tabelle 4.1:** Anforderungen bei ähnlichen wissenschaftlichen Arbeiten

Nicht betrachtete Kriterien bei dieser Arbeit, die jedoch in [Con04] vorhanden sind, sind *Überdeckungskriterien* und *Verbreitung im Anwendungsgebiet*. Bei dem ersteren Kriterium wird die Testabdeckung durch das Testverfahren überprüft. Bei dem letzteren wird untersucht, inwieweit vorhandene Werkzeuge sich in dem Anwendungsgebiet durchgesetzt haben.

Die Arbeiten von [Har01b] und [Hut06] haben einen Schwerpunkt bei den Sprachelementen auf der Ebene der Testfallimplementierung.

[Har01b] definiert zusätzlich die *Testkonfiguration* als wichtiges Bestandteil bei der Beschreibung eines Testfalls.

In der Arbeit [FK03], die im Bereich der Automatisierungstechnik eingeordnet ist, werden ebenfalls Kriterien für Beschreibungsmittel gestellt. Hier werden Kriterien für die Strukturierung, für die Möglichkeit der Verhaltensbeschreibung, die explizite Zeitdarstellung und die Möglichkeit zur Synchronisation vorgestellt. Die Verhaltensbeschreibung ähnelt dem Bewertungskriterium des unterstützten Paradigma, wo ebenfalls Ablaufbeschreibungen definiert werden müssen.

Zusätzlich wird zum Inhalt und Umfang eines Testfalls die Sprachelemente für die Testfallimplementierung bei der Bewertung betrachtet (siehe Ergebnisse aus Kapitel 3, S. 59). So kann in der Bewertung die untersuchte Sprache eingeordnet werden, ob die Sprache mehr Elemente der Implementierung als der Beschreibung besitzt oder gleichviel, so dass man diese als eine hybride Sprache bezeichnen würde.

Mit den erstellen Bewertungskriterien wird nun der Stand der Technik untersucht. Das Ziel ist die Stärken und Lücken der Arbeiten im Hinblick auf die gestellten An-

forderungen zu identifizieren.

# Kapitel 5

# Stand der Technik

## 5.1 Klassifikation des Stands der Technik

### 5.1.1 Ordnungsprinzip „Klassifikation“

In [KSS04] wird die Klassifikation wie folgt beschrieben:

> Unter Klassifikation wird ganz allgemein eine Gruppierung oder Einteilung des gesamten Wissens, der Wissenschaft und ihrer Disziplinen nach einheitlichen methodischen Prinzipien verstanden... Bei der Verwendung des Begriffs 'Klassifikation' ist zu unterscheiden zwischen
>
> - Prozess der Klassifikationserarbeitung (d.h. der Klassenbildung)
> - Klassifikationssystem als Ergebnis des Klassenbildungsprozesses
> - Prozess des Klassierens bzw. des Klassifizierens, d.h. Zuordnung von Objekten/Gegenständen und Klassen.

Eine prägnantere Beschreibung der Klassifikation liefert [Gau05]: „Von allen Ordnungsprinzipien ist die Klassifikation das einfachste. Es beruht auf dem Grundsatz: ‚Jedes Ding (jeder Sachverhalt) an seinen Platz‘.“

Klassifikationssysteme sind Hilfsmittel zur Ordnung von Gegenständen oder Wissen über Gegenstände (vgl. DIN 32705).

Dabei wird das zu klassifizierende Gebiet vollständig in Teile, den Klassen, aufgeteilt. Ferner wird die Disjunktion zwischen den Klassen gefordert, jedoch wird über

eine starke Klassifikation bzw. schwache Klassifikation gesprochen, wenn die Klassen disjunkt sind bzw. eine Überlappung vorhanden ist. In dieser Klassifikation der Beschreibungsmittel ist mit Klassifikation eine schwache Klassifikation gemeint.

Eine Klasse bildet eine Abstraktion der zu ordnenden Objekte; hier spricht man von Klassifikationsmerkmalen (sog. Klassem; vgl. DIN 32705). In der Wissensorganisation (Bibliotheken) ist die Benennung (Deskriptor, Signatur, ...) einer Klasse ein weiterer Schritt im Klassifikationsprozess, die bei den Beschreibungsmitteln kein signifikantes Ausmaß wie z.B. bei der Bibliotheksklassifizierung hat.

Aus den Klassifikationsmerkmalen und der Benennung folgen die Schritte der Klassenbildung, d.h. zu ermitteln welche Eigenschaften eines Sachverhalts für die Betrachtung in der Arbeit eine Rolle spielen. Hieraus ergibt sich ein Klassifikationssystem, so dass im dritten Schritt der Prozess des Klassierens folgen kann (vgl. oben)[1]; dies entspricht dem Stand der Technik.

**Definition 5.1 (Klassifikationssystem)** *Ein Klassifikationssystem ist die strukturierte Darstellung von Klassen und der zwischen ihnen bestehenden Begriffsbeziehungen. [fN01]*

**Definition 5.2 (Klasse)** *Eine Klasse ist die Zusammenfassung derjenigen Begriffe, die mindestens ein identisches Merkmal (Klassem) haben. [fN01]*

**Definition 5.3 (Klassem)** *Ein Klassem (oder klassifikatorisches Merkmal) ist dasjenige Merkmal von Begriffen, das zur Bildung einer Klasse benutzt wird und diese von anderen Klassen unterscheidet. [fN01]*

### 5.1.2 Vorhandene Klassifikationschemata

Für die Klassen bzw. Klassifikationsmerkmale werden vorhandene Klassifikationen im Bereich Beschreibungsmittel betrachtet. Ferner werden aus anliegenden Bereichen (Automatisierungstechnik) Klassifikationsschemata aufgezeigt.

#### Klassifizierung nach Stakeholdern

Eine Möglichkeit der Klassifikation bildet die Betrachtung des Dokuments. Die Dokumente können Menschen oder Maschinen erstellen, die Zielgruppe können Mana-

[1] Als Ordnungsprinzip eingesetzte bekannte Klassifikationen sind, das „Periodensystem der Elemente von Mendelejew“ oder das „Linnésche System zur Klassifikation der Pflanzen“.

ger oder Experten sein und der Inhalt kann abstrakt oder detailliert beschrieben erfolgen. Das führt zu einem Dreier-Tupel (Ersteller, Inhalt, Zielgruppe) (vgl. [Pat05a], [BV03]).

### Klassifizierung nach Bewertungskriterien

Eine Klassifizierung nach den Bewertungskriterien ist eine naheliegende und sinnvolle Möglichkeit der Aufteilung.

Vorhandene Arbeiten, die sich mit den Beschreibungsmöglichkeiten für Testfälle auseinandergesetzt haben, nehmen die Bewertungskriterien zur Klassifizierung (vgl. [Con04]) oder betrachten diese nicht (vgl. [Kus06], [Har01b], [Leh04]). Klassifizierungen [FK03] aus naheliegenden Gebieten wie der Automatisierungstechnik, ordnen vorhandene Mittel ebenfalls nach den Bewertungskriterien (vgl. Tabelle 4.1 zeigt die Untersuchungskriterien bei den vorhandenen Arbeiten).

Da aber die Bewertungskriterien als Grundlage für die Analyse und Bewertung dienen und am Ende - bei der Zusammenfassung der Analyse des Stands der Technik - eine Übersicht erfolgt, wird in dieser Arbeit eine andere Klassifizierung für sinnvoll gehalten, die im Folgenden begründet wird.

### Klassifizierung nach verwendeten Grundkonzepten

Der Gedanke, dass jede Arbeit als Grundlage auf vorherige wissenschaftliche Arbeiten aufbaut, wird aufgegriffen. Die Anzahl der Arbeiten, die von Grund auf ein neues Konzept bieten sind übersichtlich. Zu diesen gehört zum Beispiel das Unified Modeling Language, kurz UML. Genauso dient die XML als Grundlage für verschiedene wissenschaftliche Arbeiten aufgrund ihrer universellen Einsatzmöglichkeiten. Die endlichen Automaten, als ein Teil der formalen Sprachen, und ihre Devirate werden in verschiedenen Arbeiten unterschiedlichster Wissenschaftsgebiete eingesetzt, aufgrund ihres universellen Charakters.

Dieser Gedanke ist bei der Betrachtung der Testablaufsprachen bzw. der Ablaufsprachen ebenfalls vorhanden: Sie bauen auf bereits bekannten Konzepten auf und erweitern oder spezifizieren diese für ein bestimmtes Anwendungsgebiet. Deshalb ist eine Klassifizierung vorhandener Beschreibungsmittel im automobilen Bereich aufbauend auf bereits vorhandenen Mitteln als sinnvoll: Ob man mit dem einen Derivat oder dem anderen Derivat ein und desselben Kozeptes 'describe the things right' erreicht, hängt von dem Konzept selbst ab.

Die Klassifikation in [Pat05a] ordnet die Sprachen der Informatik, wie sie in Abbildung 5.1 zu sehen ist. Die Aufteilung erfolgt in natürliche und künstliche Sprachen. Bei den künstlichen Sprachen wird die Definition betrachtet. Besteht die Definition aus einem Vokabular und einer Syntax, so ist die Sprache in der Klasse der formalen Sprachen. Hat die Definition zusätzlich eine Semantik, so wird diese zusätzlich nach ihrer Bedeutung aufgeteilt: Ist die Semantik deskriptiv, ist die Sprache in der Klasse der Modellierungssprachen. Ist die Semantik performativ, so ist die Sprache in der Klasse der Programmiersprachen.

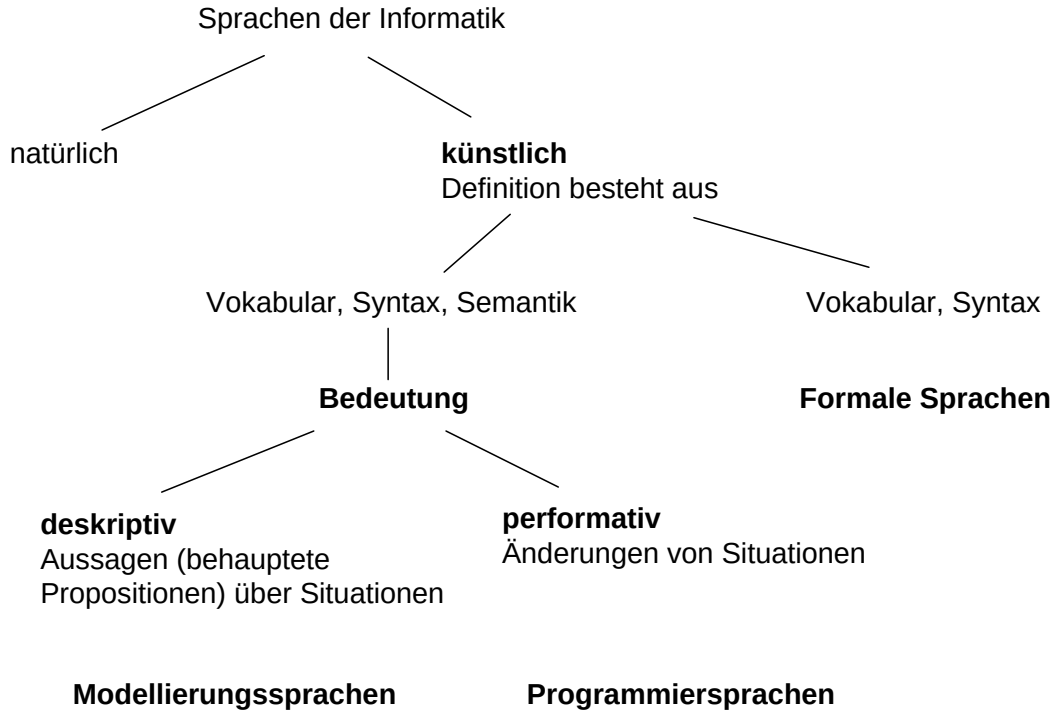



**Abbildung 5.1:** Klassifikation der Sprachen in der Informatik [Pat05a]

In [WWD+97] wird bei der Klassifizierung von Workflow Konzepten ebenfalls die grundlegenden Beschreibungsmöglichkeiten für die Klassifizierung verwendet. Die vorhandenen Konzepte werden in Skriptsprachen, netzorientierte Methoden, logikbasierte Methoden, algebraische Methoden und ECA-Regeln (Event-Condition-Action Regeln) unterteilt.

Ebenfalls findet sich ein ähnlicher Ansatz zur Klassifikation der Testfallspezifikationsmittel in [Pre03].

**Klassifikation nach verwendeten Grundkonzepten**

| Formale Syntax | Formale Semantik | Gruppen |
|---|---|---|
| + | + | Spezifikationssprachen - Endliche Automaten |
| + | + / - | Programmiersprachen - Domain Specific Languages |
| + | + | Auszeichnungssprachen - XML |
| + | - | Modellierungssprachen - UML |
| - | - | Werkzeuge – Tabellen-basierte Form |
| - | - | Natürliche Sprache |

**Abbildung 5.2:** Verwendete Klassifikation in dieser Arbeit

### 5.1.3 Darstellung der verwendeten Klassifikation

Die verwendete Klassifikation in Abbildung 5.2 basiert auf der Aufteilung in [Pat05a] und ist angepasst für die untersuchten Mittel in dieser Arbeit. Dabei wird grundsätzlich ebenfalls in natürliche und künstliche Sprache unterteilt. Bei der künstlichen wird der Formalisierungsgrad der Syntax und Semantik betrachtet[2].

Hier fällt die Betrachtung auf die Spezifikationssprachen (endliche Automaten), Programmiersprachen (DSL - Domain Specific Languages) und Auszeichnungssprachen (XML - Extensible Markup Language). Zusätzlich ist die UML (Unified Modeling Language) als eine Modellierungssprache vertreten, auf der viele wissenschaftliche Arbeiten basieren. Die natürlichsprachliche und werkzeugbasierte Beschreibung mit informeller Syntax und Semantik ist wegen des starken Industrieeinsatzes berücksichtigt.

Die aufgelisteten Grundkonzepte decken eine große Palette ab, können jedoch nie vollständig sein. So könnten bei den Spezifikationssprachen z.B. die Sprache Z betrachtet werden, die ebenfalls im wissenschaftlichen Umfeld für den Einsatz beim Testen als Grundlage untersucht wurde. Doch diese Arbeiten sind sehr übersichtlich, so dass sich eine Betrachtung erübrigt.

[2]Programmiersprachen besitzen theoretisch eine formale Semantik, d.h. bei wissenschaftlicher Betrachtung werden formale Semantiken als Grundlage für Untersuchungen verwendet. In der Praxis sind Programmiersprachen wie Java zum größten Teil in natürlicher Sprache spezifiziert und besitzen für eine Teilmenge eine formale Semantik, die aus dem wissenschaftlichen Umfeld definiert ist.

## 5.2 Untersuchungsschema

Jedes Testfallbeschreibungsmittel wird im ersten Schritt allgemein betrachtet. Hier werden eventuell vorhandene Erweiterungen aufgezeigt, die im Kontext der Betrachtung relevant sind. Danach werden die Kriterien wie sie im Abschnitt 4.5 aufgezählt sind herangezogen und die Möglichkeiten für jedes Kriterium dargestellt. Tabelle 5.1 zeigt die Grundstruktur jedes einzelnen Unterkapitels.

| Abschnitt | Beschreibung |
|---|---|
| Grundlagen | Dieser Abschnitt dient dazu einen Überblick über das Testbeschreibungsmittel zu geben. Dabei wird die Mächtigkeit der vorhandene Mittel dargestellt (, d.h. auch solche Eigenschaften, die im Gesamtkontext der Arbeit irrelevant sind.) |
| Erweiterungen | Dieser optionale Abschnitt zeigt vorhandene Erweiterungen auf. Dabei werden nur solche Erweiterungen erwähnt, die für den Kontext relevant sind (Erfüllung eines Kriteriums). |
| Kriterien | Die definierten Untersuchungsmerkmale werden hier explizit dargestellt und bewertet. |
| Beispiel | Hier wird die Frage beantwortet wie eine Testbeschreibung in dem jeweiligen Testbeschreibungsmittel aussehen würde. |
| Zusammenfasssung | In der Zusammenfassung werden die Stärken und Schwächen mit Bezug auf die Kriterien dargestellt und bewertet. |

**Tabelle 5.1:** Schema bei der Untersuchung der Testfallbeschreibungsmittel

Abbildung 5.3 zeigt das Schema zur Überprüfung der Kriterien. Die erste Spalte beinhaltet die Gruppierungen und die zweite Spalte die Bewertungskriterien, wie sie in Abschnitt 4.5, S. 118 aufgestellt wurde. Die dritte Spalte beinhaltet die Gewichtungen der einzelnen Kriterien. Die Gewichtung entspricht der Anzahl der Ableitungen einer Anforderung aus den Anwendungsfällen (vgl. Kapitel 4, S. 97).

Die letzte Spalte beinhaltet die ermittelte Anzahl der Punkte der betrachteten Sprache. In der Abbildung ist die Spalte mit Beispielwerten gefüllt. So erreicht die Sprache bei dem Kriterium „Inhalt und Umfang eines Testfalls (Testfallbeschr.)“ 25 Punkte und bei der Unterstützung der Testfallparadigmen 2. Diese Punkte werden jeweils mit den Gewichtungen multipliziert und anschliessend addiert (25 * 4 + 2 * 4), sodass die Punktzahl in dieser Gruppe 108 beträgt.

Die Bewertung der einzelnen Zeilen ist in Abschnitt 4.5, S. 118 dargestellt. Die Gesamtpunktzahl ist die Summe der Ergebnisse der einzelnen Gruppen (108 + 4 + 1 + 9 = 122).

| | | Gewichtung | |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 25 |
| | Unterstützung Testfallparadigmen | 4 | 2 |
| | | | 108 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | textuell, graphisch |
| | Formalismus | 2 | formal |
| | | | 4 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | ja |
| | | | 1 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | ja |
| | Sprachelemente für Platzhalter | 1 | ja |
| | Sprachelemente für Templates | 1 | ja |
| | Sprachelemente für Reaktionsauswertung | 1 | ja |
| | Sprachelemente für Hierarchisierung | 2 | ja |
| | Darstellung von Metainformationen | 2 | ja |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | ja |
| | | | 9 |
| | | | 122 |

**Abbildung 5.3:** Schema zur Kriterienbewertung

## Betrachtete Beschreibungsmittel für Testfälle

Grundsätzlich können die Beschreibungsmittel in Spezifikationssprachen aufgeteilt werden. Je nachdem wie formal bzw. informal das Mittel betrachtet wird: Formale Spezifikation, Programmiersprachen, Auszeichnungssprachen, Testdatenorientierte Ansätze und die natürliche Sprache. Bei den formalen Mitteln werden die Automaten bzw. Statecharts genauer betrachtet und ebenfalls wird die DSL (Domain Specific Language) genauer betrachtet.

- XML, UML, DSL, Programmiersprachen und Automaten

Der Schwerpunkt der Untersuchung liegt im Beschreibungsmittel, welches universell einsetzbar ist (generische Beschreibungsmittel) und die im Umfeld der Testfallbeschreibung verwendet wird. Hier werden vor allem die XML als Auszeichnungssprache, die UML als Modellierungssprache, Programmiersprachen als generische Sprachen zur Beschreibung und Automaten als universelle Alleskönner betrachtet.

- Beschreibungssprachen in anderen Anwendungsgebieten

  Anwendungsgebiete bei denen Abläufe im Fokus der Beschreibung liegen, sind für die Testfallbeschreibung ebenfalls interessant zu betrachten. Im Kapitel über den Inhalt und Umfang der Testfälle wurde bei der Bottom-Up Analyse festgestellt, dass eine Ablaufbeschreibung angereichert mit testablauf- und testumgebungspezifischen Sprachelementen die Beschreibung eines Testfalls abdecken würde (vgl. Abschnitt 3.3, S. 66). Daraus resultiert, dass im Stand der Technik auch Mittel untersucht werden müssen, die rein für eine Ablaufbeschreibung konzipiert wurden. Meist können diese durch testablauf- und testumgebungsspezifische Sprachelemente erweitert werden, um eine vollständige Testfallbeschreibungssprache darzustellen.

  Für die Betrachtung anderer Anwendungsgebiete werden die Spezifikationsmittel für die Systemverhaltensmodelle betrachtet, da sie einen Einfluss auf die Testfallbeschreibung besitzen (vgl. 3, S. 59). So wird in Kapitel 3, S. 59 diese Betrachtung vorweggenommen um den Inhalt und Umfang der Testfälle ableiten zu können.

## 5.3 Beschreibungsmittel

### 5.3.1 Spezifikationssprachen

#### Grundlagen

Sprachen für die Spezifikation beinhalten grundlegende Eigenschaften für die Modellierung von Softwarearchitekturen. Besonderen Stellenwert haben formale Sprachen wie Automaten (Statecharts) und Z durch ihre leichte Verständlichkeit. So werden die Automaten auch durch ihre leichte Verständlichkeit zur Grundlage vieler der hier vorgestellten Lösungen gewählt (vgl. [Leh04], [GCF+06]).

Die Spezifikationssprachen werden in informelle, semi-formale und formale Spezifikationssprachen unterteilt. Die Unterteilung erfolgt nach der Formalität der Syntax

und der Semantik [3]. Informelle Sprachen wie die natürliche Sprache, besitzen weder formale Syntax und noch eine formale Semantik.

Die Sprachen der semi-formalen Klasse haben zwar eine formale Syntax, aber keine formale Semantik. Die Semantik wird informell, z.B. durch die natürliche Sprache festgelegt. UML gehört zu dieser Klasse.

Zuletzt besteht, bei den Sprachen der formalen Spezifikationssprachen, neben der formalen Syntax noch eine formale Semantik. Die formale Semantik wird dabei durch Regeln (Kalküle) definiert. Sie kann u.a. axiomatisch, operationel oder denotational erfolgen[4]. Wie erwähnt zählen zu dieser Klasse u.a. die endlichen Automaten (und Statecharts), Petri-Netze und die Spezifikationssprache Z.

Der Vorteil bei der Verwendung von formalen Spezifikationssprachen ist offensichtlich die Eindeutigkeit, die durch die natürliche Sprache nicht gegeben ist. Jedoch ist ein Einsatz von formalen Spezifikationssprachen beschränkt durch die Verständlichkeit [CF98]. Auf der anderen Seite fordert eine formale Spezifikationssprache eine genauere Überlegung von der Beschreibung, die ein Benutzer verfasst [Hal90].

Allerdings garantiert eine formale Grundlage nicht, das fehlerhafte oder inkonsistente Spezifikationen erstellt werden [Büs03]. Zusätzlich wird bei der Erstellung einer formalen Grundlage nicht die Verständlichkeit in den Vordergrund gestellt, sondern die mathematische Korrektheit.

#### Erweiterungen

Es existiert eine Fülle von Spezifikationssprachen für die Spezifikation eines Systems. In [Zha08] werden u.a. Statecharts, SDL (Specification and Description Language), Petri-Netze und Sequenzdiagramme für die Anwendbarkeit als Testfallbeschreibungssprache untersucht. Hierbei wird die Testsemantik (s. Abschnitt 3.5, S. 77) betrachtet. Als Fazit kann gezogen werden, dass alle vier Spezifikationssprachen mit der richtigen Erweiterung hinsichtlich der Testsemantik eingesetzt werden können, da sie ablaufbeschreibende Eigenschaften vorweisen. Ihre eigentlichen Verwendungszweck, z.B. Modellierung von Nebenläufigkeit und Dead-Lock Erkennung bei Petri-Netzen kommen bei der Beschreibung der Testfälle nicht zum Einsatz .

In [KS04], [SCS97] und [CMM+00] wird die Spezifikationssprache Z für das spezifikationsbasierte Testen verwendet. Hier wird aus der Spezifikation des Systems in Z

[3] In Kapitel 6, S. 173 wird die Syntax und Semantik detaillierter vorgestellt.
[4] In Kapitel 6, S. 173 werden diese drei Formen der Definition detaillierter vorgestellt.

ein Automat (Statecharts) abgeleitet[5]. Dabei wird der Automat auf zwei Arten verwendet: Einmal zur Zustandsüberprüfung und zum anderen zur Überprüfung der korrekten Ausgaben. Die Ein- und Ausgabewerte werden nur diskret betrachtet.

Eine weitere Verwendung der Automaten erfolgt in Kombination mit Markov-Ketten in [ZEKB09]. Hier werden ebenfalls Automaten (als Modell) für die Ableitung der Testfälle (Testdaten) verwendet. Die Markov-Ketten kommen als statistisches Werkzeug zum Einsatz.

Das auf MSC (Message Sequence Chart) basierende Sequenzdiagramm in der UML2 wird im Abschnitt 5.3.4, S. 152 betrachtet.

Speziell für das Testen von Steuergeräten, die kontinuierliches Verhalten aufzeigen ist TPT [Leh04] entwickelt worden, das auf den hybriden Automaten basiert. Sie wird im Folgenden untersucht.

**Auf endlichen Automaten basierende Ansätze:** Die endlichen Automaten zählen zu den formalen Spezifikationssprachen. Da ihre Präsenz bei der Spezifikation und beim Testen deutlich ist, werden sie gesondert betrachtet.

Die endlichen Automaten besitzen unterschiedliche Ausprägungen.

Statecharts sind eine Erweiterung der Automaten um Hierarchisierung und Nebenläufigkeit. Zustände, die über Eingaben in gleiche Zustände überführt werden, können bei den Statecharts zusammengefasst werden (Stichwort: Zustandsexplosion). Durch die Hierarchisierung und Kapselung sind Start- und Endknoten, sowie eine History definiert.

Eine weitere Ausprägung der endlichen Automaten sind die Timed Automata [AD94]. Hier sind die endlichen Automaten mit zeitlichen Bedingungen erweitert. Dadurch ist es möglich Echtzeitsysteme darzustellen[6].

### TPT

**Grundlagen** Bei TPT (Time Partition Testing) [Leh04] liegt der Fokus auf der Modellierung von Testfällen für das kontinuierliche Verhalten eingebetteter Systeme. Zusätzlich werden neben der Modellierung auch Mechanismen zur Durchführung

[5] In der Arbeit wird weiterhin auf ähnliche Ansätze zur Kombination der Statecharts mit Petri-Netzen verwiesen.

[6] Für eine mathematisch korrekte Darstellung der Timed Automata wird auf die Arbeit [AD94] verwiesen.

und Auswertung konzipiert. Dabei konzentriert sich die Arbeit in der Einsetzbarkeit im produktiven Umfeld, so dass ein Werkzeug zur Verfügung steht.

TPT basiert bei der Modellierung auf zwei Grundlagen: Den hierarchischen Automaten (Timed Automata) und den Klassifikationsbäumen. Die hierarchischen Automaten dienen zur Modellierung von Testfällen und die Klassifikiationsbäume zur Auswahl von Parametern eines (gleich ablaufenden) Testfalls.

Die Diagramme besitzen alle Elemente von Statecharts und diese können auch verwendet werden (z.B. Aktionen an Transitionen leiten Zustandsübergänge ein, parallele und hierarchische Kompositionen, ...). Nur die Semantik ist an die Bedürfnisse angepasst und basiert auf den stromverarbeitenden Funktionen[7].
Neben der graphischen Modellierung von Testfällen gibt es eine auf Gleichungssystemen aufbauende textuelle Möglichkeit.

Testdurchführung und Auswertung erfordert auch bei TPT eine Zwischenschicht. Die Testfälle sind abstrakt formuliert, d.h. sie besitzen keine Information um direkt auf einem Testmittel ausgeführt werden zu können. Damit sie aber die Testmittel spezifischen Schnittstellen benutzen können, wird eine Treiberschicht benötigt. Das Konzept eines sog. Testengines ist für die Ausführung der in XML hinterlegten Testfälle - mit den gewünschten Parametern - verantwortlich.

Bei der Auswertung der Testfälle kommen erwartete Werte (Assessments) zum Einsatz, die an die Zustände zugeordnet werden. Die Ergebnisse werden zur Bewertung dokumentiert.

Ein Werkzeug fasst die Konzepte des TPT zusammen und realisiert diese. Mit dem Werkzeug wird eine durchgängige Testbegleitung von der Modellierung bis hin zur Auswertung angeboten.

**Erweiterungen** Im kommerziellen Werkzeug TPT[8] wird Python zur Auswertung der Ergebnisse bzw. für die Bewertung der Ausgabesignale eingesetzt. Die Auswertung ist für Programmierer nachvollziehbar, bleibt jedoch für einen Reviewer ohne Erklärung undurchsichtig.

**Kriterien** Abbildung 5.4 zeigt die Abdeckung der Bewertungskriterien durch TPT. Eine detaillierte Betrachtung der einzelnen Sprachelemente ist in Abbildung 5.22, S. 171 im Vergleich zu den anderen untersuchten Mitteln zu finden.

[7] mehr zu den stromverarbeitenden Funktionen in [Bro97]
[8] http://www.piketec.com

| | | Gewichtung | TPT |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 8 |
| | Unterstützung Testfallparadigmen | 4 | 2 |
| | | | 40 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | graphisch, textuell |
| | Formalismus | 2 | formal |
| | | | 4 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | ja |
| | | | 1 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | ja |
| | Sprachelemente für Platzhalter | 1 | ja |
| | Sprachelemente für Templates | 1 | ja |
| | Sprachelemente für Reaktionsauswertung | 1 | ja |
| | Sprachelemente für Hierarchisierung | 2 | ja |
| | Darstellung von Metainformationen | 2 | nein |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | ja |
| | | | 7 |
| | | | 52 |

**Abbildung 5.4:** Abdeckung der Bewertungskriterien durch TPT

**Beispiel** Abbildung 5.5 zeigt eine Beispielimplementierung des Richtungsblinkertests in TPT.

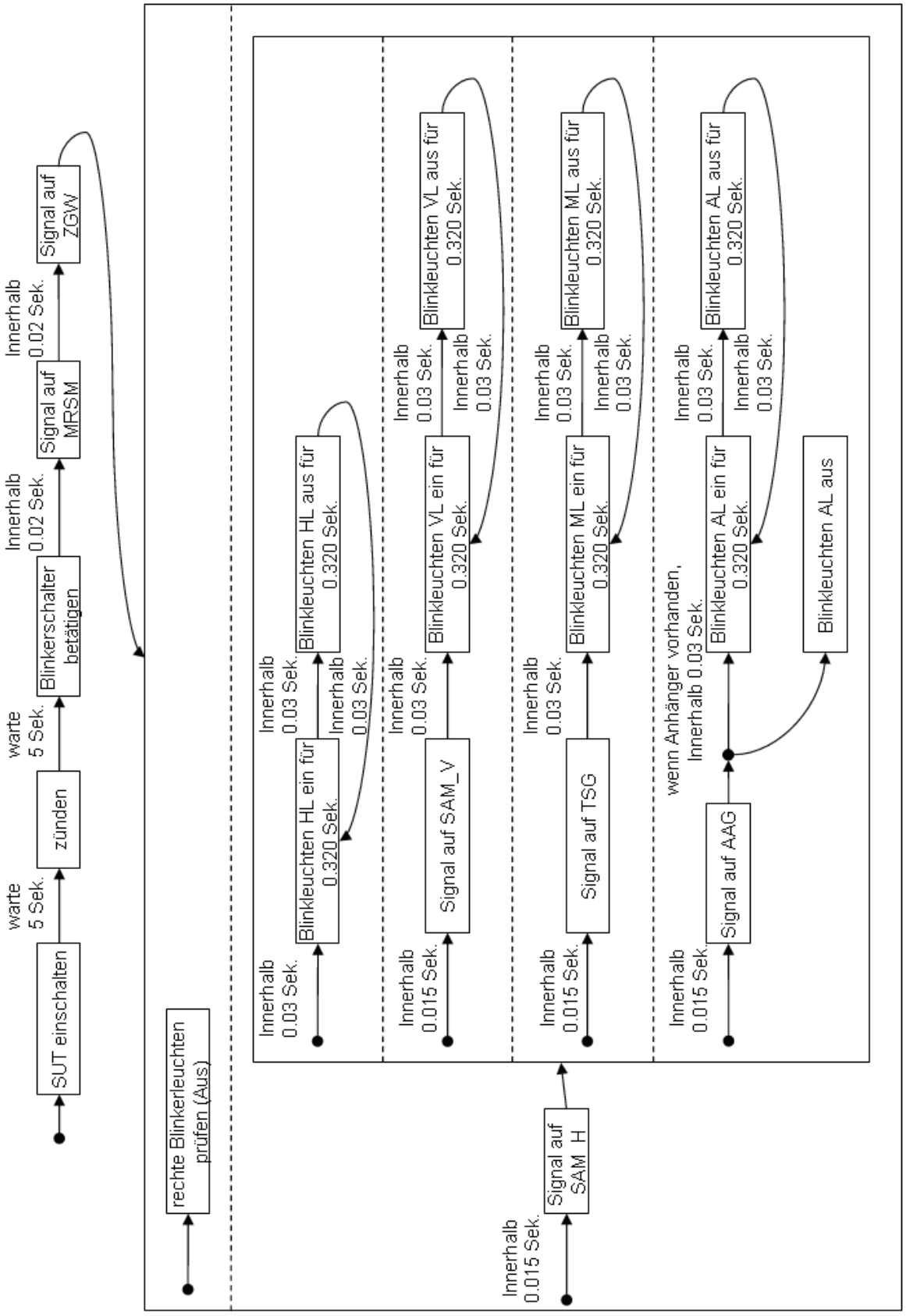



**Abbildung 5.5:** Beispielimplementierung des Richtungsblinkertests in TPT [Zha08]

**Zusammenfassung** Durch die Auszeichnung der Pfade für ein Szenario, sind unterschiedliche Variantenabdeckungen möglich. Die unterschiedliche Bedatung jedes Zustandes ermöglicht ebenfalls einen Testablauf mit unterschiedlichen Parametern.

Jedoch sind keine Aufrufe von Funktionen vorhanden bzw. Möglichkeiten gegeben womit man Testsystemabhängige Einstellungen vornehmen kann.

### 5.3.2 Programmiersprachen

#### Grundlagen

Programmiersprachen bilden Algorithmen ab um diese auf Rechnern ablaufen zu lassen. Die meisten bekannten Programmiersprachen wie Java, C, C++ und Skriptsprachen wie Python, Perl und Ruby werden als General Purpose Languages bezeichnet, da diese für keine bestimmte Anwendungsdomäne entwickelt wurden.

Mit General Purpose Languages können alle Probleme der Algorithmik abgebildet werden. So werden auch Testfälle bei der Implementierung bevorzugt durch Skriptsprachen umgesetzt.

Eine Aufteilung der Programmiersprachen kann unterschiedlich erfolgen.

Eine bekannte Aufteilung ist nach den Generationen. Die Maschinensprache wird als erste Generation bezeichnet. Die nächste Generation (2. Generation) ist die Abbildung der Maschinencodes durch Kürzel (Mnemonics): Assemblersprache.

Mit der dritten Generation sind „höhere Programmiersprachen“ gemeint, d.h. solche Sprachen, die Abstraktionen erlauben und nicht am Maschinencode aufbauen. Ein Compiler setzt dabei das Programm in Maschinencode um. Die Programmiersprache orientiert sich in der dritten Generation an dem Programmierer, der den Code im nach hinein verstehen soll.

Die vierte und fünfte Generation von Programmiersprachen zielen auf das Problem ab, nicht mehr auf die Lösung an sich. Durch die problemnahe Programmierung sind die Sprachen der vierten Generation auf die Anwendungsdomäne zugeschnitten. Beispiele hierfür ist die SQL für Datenbanken oder SVG für Computergrafik.

Eine andere Aufteilung der Programmiersprachen ist nach den Programmierparadigmen. Grundsätzlich werden hier nach imperativen und deklarativen Sprachen unterschieden. In manchen Literaturquellen ist eine dritte Aufteilung nach objektorientierten Paradigma zu finden, die bei den Methoden allerdings auf das imperative

Paradigma zurückzuführen sind.

Die imperative Programmierung zielt auf die Problemlösung hin und beschreibt das 'wie'.

Bei der deklarativen Programmierung werden die gesuchten Ergebnisse angegeben und die Lösung vom Rechner bestimmt.
Hier unterscheidet man in funktionale Programmierung (Lisp, ML, Erlang) und logische Programmierung (Prolog).

Beispiel: Bei Prolog werden Fakten und Regeln definiert. Bei den Fakten können Tatsachen festgehalten werden und die Regeln definieren Zusammenhänge aus denen neue Zusammenhänge mit den bekannten Fakten erstellt werden können. Durch Anfragen (Ziele) an das System werden Probleme gestellt, die anschliessend soweit möglich beantwortet werden.

#### Erweiterungen

Programmiersprachen wie Visual Basic und Python werden bei der Implementierung der Testfälle verwendet. Durch die Möglichkeit Bibliotheken zu definieren sind Abstraktionen zu den Zielplatformen möglich. So kann eine einheitliche API (Application Programming Interface)[9] definiert werden. Die konkrete Umsetzung der Schnittstellen kann durch Bibliotheken für die jeweilige Zielplatform realisiert werden.

Die grundlegende Idee für die Verwendung von Programmiersprachen für die Testfallspezifikation ist, dass eine Programmierschnittstelle definiert wird, die durch Bibliotheken realisiert werden kann. Auf diesem Abstrahierungsmechanismus aufbauend werden die Testfälle beschrieben ohne wirkliche Testdaten zu verwenden oder zielplatformabhängig zu sein.

**Echtzeitaspekte:** Bei der Implementierung werden Anforderungen an den Ablauf der Testfälle gestellt, d.h. um die Reaktion eines Steuergerätes Testen zu können muss eine Tastrate im Mikrosekundenbereich vorhanden sein. Diese ist allerdings bei der Durchführung eines Testfalls relevant und nicht in der Beschreibung (vgl. Abschnitt 3.6.3).

**Spezielle Programmiersprachen:** Verteilte Systeme und Mehrprozessorsysteme stellen unterschiedliche Anforderungen an die Programmierung. Hier können durch

[9] http://de.wikipedia.org/wiki/Programmierschnittstelle

neue Sprachelemente bei den General Purpose Languages (eventuell durch Erweiterung mittels Bibliotheken) die Anforderungen abgedeckt werden. Eine andere Vorgehensweise ist die Entwicklung einer neuen Sprache für diese spezielle Anwendungsdomäne.

So sind die synchronen Programmiersparchen wie Esterel[10] oder Lustre[11] für die Entwicklung von reaktiven Systemen konzipiert. Allerdings spielt dieser Aspekte für die Testfallspezifikation keine Rolle.

**Domain Specific Languages:** Eine spezielle Untergruppe der Programmiersprachen bilden die DSLs, die Domain Specific Languages. Sie sollen im Folgenden näher betrachtet und für die Bewertung herangezogen werden.

### Domain Specific Language / Modelling

Domain Specific Languages sind Sprachen, die spezielle für eine Domäne entwickelt wurden. Die Entwicklung kann auf einer vorhandenen Sprache aufbauen (interne DSL) oder von Grundauf neu aufsetzen (externe DSL) [Fow09], [Slo02].

Ihre Erstellung dient nur um ein Anwendungszweck abzudecken, um genau für ein Problem eine Lösung anzubieten (vgl. [CM09], [MHS05], [RGTL09]). Bei der Erstellung wird das Anwendungsgebiet analysiert, das Konzept der Sprache und somit die Elemente der Sprache werden an die Umgebung angepasst. Für die Implementierung werden unterschiedliche Herangehensweisen herangezogen.

Typische DSL Sprachen sind HTML für die Erstellung von WWW-Seiten, SQL für Datenbanken, VHDL für Hardwareentwurf oder auch BNF (Backus-Naur-Form) für die Darstellung kontextfreier Grammatiken (z.B. für die Syntaxdefinition von Sprachen).

**Definition 5.4 (Domänenspezifsiche Sprache)** *Eine Domänenspezifische Sprache (DSL) ist eine Programmiersprache oder eine ausführbare Spezifikationssprache, welche durch angemessene Notation und Abstraktion bereichert, eine Ausdrucksstärke bietet, die auf ein bestimmtes Domänenproblem fokussiert und eingeschränkt ist. (Übersetzung aus [vDKV00])*

Ebenfalls ist das Konzept des Domain Specific Modelling an die DSL angelehnt, mit

[10] http://de.wikipedia.org/wiki/Esterel_(Programmiersprache)
[11] http://de.wikipedia.org/wiki/Lustre_(Programmiersprache)

dem Unterschied, dass die Grundlage Modellierungssprachen bilden und nicht Programmiersprachen. Mit der Modellierung sind im weitesten Sinne alle graphischen Möglichkeiten gemeint, mit denen sich Strukturen und Verhalten abbilden lassen.

Interne DSLs haben den Nachteil, dass sie syntaktisch auf die Verwendung der zugrundeliegenden Programmiersprachensyntax eingeschränkt sind. Für die Entwicklung einer eigenen Syntax bzw. einer externen DSL spricht eine Adaption der Syntax an die Anwendungsdomäne.

Allerdings gibt es eine Möglichkeit für die Verwendung einer eigenen Syntax bei der internen DSL: die eigene definierte Syntax wird durch einen Zwischen-Kompiler in die Zielprogrammiersprache umformuliert (ähnlich [SA98]). Dadurch hätte man eine angepasste Syntax auf die Anwendungsdomäne, mit dem Vorteil grundlegende Mechanismen wie Kontrollstrukturen zu verwenden. Der Mehraufwand entsteht durch den zusätzlich eingeführten Zwischenschritt, das sich bei der Transformation auf die Zielsprache bemerkbar macht.

Im Folgenden wird TTCN3 als eine DSL für die Testfallspezifikation untersucht. Arbeiten wie [Liu00] basieren ebenfalls auf externen DSLs, haben jedoch für die Betrachtung in der Domäne funktionales Steuergerätetests wenig Relevanz.

## TTCN3

### Grundlagen

TTCN-3 (Testing and Test Control Notation) dient zur Erstellung von abstrakten Testfällen im Bereich des Black-Box Testens für reaktive und verteilte Systeme [GWWH00], [Gra00]. Während die Vorgänger-Versionen sehr stark auf den Telekommunikationsmarkt orientiert waren, ist die dritte Version allgemeiner gefasst, so dass der Einsatz dieser Version auch für den Automobilen Bereich vorstellbar ist.

Den Zentrum von TTCN-3 bildet eine textuelle Programmiersprache. Dieser Kern wird durch Zusätze zur Darstellung und Datenpräsentation erweitert. So definiert TTCN-3 zwei Präsentationsformen: ein Sequenzdiagramm- und eine tabellen-basierte Möglichkeit Testfälle zu beschreiben.

Da auch jeder Anwendungsbereich seine eigenen Formate für die Daten bringen kann, können auch auf der Datenseite eigene Formate eingebunden werden. Der Standard bietet hier ASN.1 (Abstract Syntax Notation One) an, die im Kommunikationsbereich beheimatet ist.

Die Spezifikation von TTCN-3 wird in sieben Teildokumenten definiert. Neben dem Kern sind Definitionen für die graphischen Präsentationsformate, die Einbettung in die Testumgebung und die Datenbeschreibung beschrieben.

Im Weiteren Verlauf wird nur die Kernsprache betrachtet, da die Präsentationsformate von dieser abhängig sind.

Ein Modul ist das zentrale Konzept der Programmiersprache. Im Wesentlichen besteht ein Modul aus einem statischen und einem optionalen dynamischen Teil. Der statische Teil ('description part') beinhaltet u.a. Datentypen, Funktionen, Templates und die Testfälle. Im dynamischen Teil ('control part') werden die Testfälle in Beziehung gesetzt. Die Testfälle können im statischen Teil beschrieben werden oder auch aus anderen Modulen importiert worden sein. Für die Beziehung der Testfälle werden bekannte Kontrollflussstrukturen wie if-then-else und while Schleifen zur Verfügung gestellt.

Vorlagen ermöglichen die Wiederverwendung sehr oft gebrauchter Abläufe mit unterschiedlichen Parametern.

Bei der Testdurchführung sind Bestandteile neben der eigentlichen SUT nötig die eine Driver oder Stub Rolle übernehmen. Die Umgebung der zu testenden Einheit wird durch die Sprache modelliert. Des Weiteren werden die Kommunikationsschnittstellen festgelegt. Eine Besonderheit in der Definition der Testfälle ist die Master-Slave Definition: Werden mehrere Komponenten definiert, so muss einer dieser als Master definiert werden um damit bei der Ausführung einen Einstiegspunkt zu ermöglichen (ähnlich dem obligatorischem main Teil eines C Programms).

Zur Ausführung wird die komplette Definition kompiliert. Dabei werden die abstrakt definierten Werte in die konkreten umgewandelt und man erhält eine ausführbare Datei (ETS - Executable Test Cases). Diese Datei liegt in der TTCN-3 Executable (TE). Für die Umwandlung der abstrakten Werte in die konkreten Werte des Testobjekts, für die eingesetzten Hardware- und Softwareteile, ist die TRI (TTCN-3 Runtime Interface) verantwortlich. Hier ist ein Systemadapter und ein Platformadapter definiert. Zusätzlich ist eine Steuerungsschnittstelle, die TCI (TTCN-3 Control Interface), im Standard, mit der es möglich ist TTCN-3 in die Testumgebung einzubetten. Diese beinhaltet einen Logger, einen Komponentenverwalter, einen De- und Endcodierer (CoDec) und eine Verwaltungskomponente. Abbildung 5.6 zeigt die TTCN-3 Architektur.

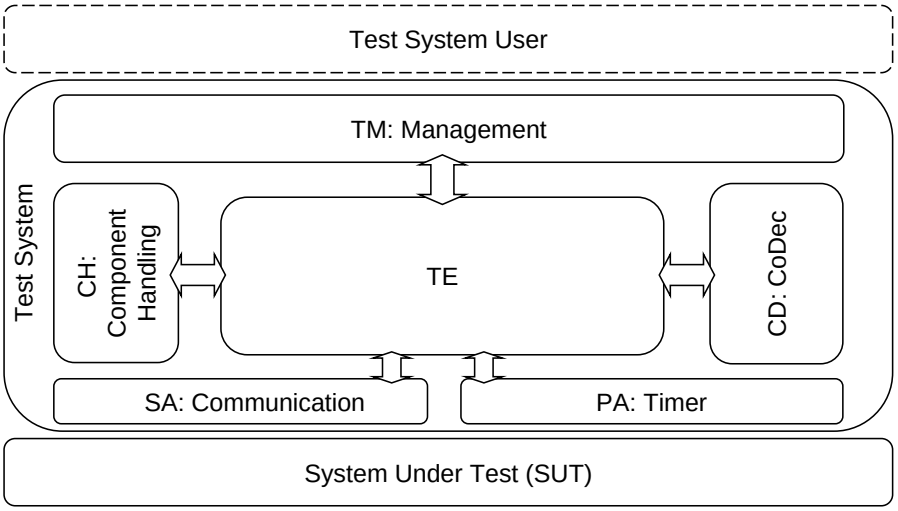



**Abbildung 5.6:** Die TTCN-3 Architektur [IES07]

### Erweiterungen

Für die zeitlichen Aspekte gibt es eine Erweiterung Namens Timed TTCN-3 [DGN02], das zum Testen von Echtzeitaspekten dient. Hier werden Zeitstempel eingeführt um laut der Arbeit harte Echtzeiteigenschaften wie Verzögerung, Antwortzeit, Latenzzeiten, Jitter und Durchsatz testen zu können.

Anmerkung: Die erwähnten harten Echtzeitanforderungen wie Latenzzeiten, Jitter und Durchsatz sind im Telekommunikationsbereich relevant.

Eine Erweiterung bezüglich der Darstellung (Präsentationsform) ist das UML2 Testing Profile (siehe Abschnitt 5.3.4, S. 152).

Eine weitere Erweiterung [SBG06] ermöglicht das Testen des kontiunierlichen Verhaltens. Als Spracherweiterung werden Variablen eingeführt, die eingehende Werte über die Zeit 'speichern' und bei denen Signalverläufe definiert werden können. So kann auf den i'ten Wert einer Variable oder der Wert zum j'ten Zeitpunkt zugegriffen werden. Listing 5.1 zeigt die Definition von kontinuierlichen Eingangssignalen in TTCN3.

```
1 var FloatStrm myStrm:=
        {1.0,2.0,45.0,66.0,77.0};
3
const FloatStrm mySecondStrm@t:= sin(t)+100.0;
5
const FloatStrm myThirdStrm@t:= mySecondStrm@(t-10.0)+4.0;
7
myFourthStrm:={
```

```
9         @(0..100) :=sin(t)*100.0,
          @(100..2000) :=4.0,
11        @(2000..infinity) :=4.0+cos(t/20.0)
}
13
myFifthStrm:={
15        [0..99] :=sin(t)*100.0,
          [100..1999] :=4.0,
17        [2000..infinity] :=4+sin(t/20.0)
}
```

**Listing 5.1:** Beispiel einer Signaldefinition im erweiterten TTCN3

**Kriterien** Abbildung 5.7 zeigt die Abdeckung der Bewertungskriterien durch TTCN 3. Eine detaillierte Betrachtung der einzelnen Sprachelemente ist in Abbildung 5.22, S. 171 im Vergleich zu den anderen untersuchten Mitteln zu finden.

| | | Gewichtung | TTCN3 |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 8 |
| | Unterstützung Testfallparadigmen | 4 | 1 |
| | | | 36 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | textuell, graphisch |
| | Formalismus | 2 | formal |
| | | | 4 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | ja |
| | | | 1 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | ja |
| | Sprachelemente für Platzhalter | 1 | ja |
| | Sprachelemente für Templates | 1 | ja |
| | Sprachelemente für Reaktionsauswertung | 1 | ja |
| | Sprachelemente für Hierarchisierung | 2 | ja |
| | Darstellung von Metainformationen | 2 | nein |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | nein |
| | | | 6 |
| | | | 47 |

**Abbildung 5.7:** Abdeckung der Bewertungskriterien durch TTCN 3

**Beispiel** Abbildung 5.8 zeigt eine Beispielimplementierung des Richtungsblinkertests in TTCN 3.

```
module LinksBlinkenTest{
    :
    type port MyMixedPortType mixed {
        in SGPortIn;
        inout float
    }
    type component BlinkenSystem {
        var integer Batterie, Zuendschluessel, Blinkerschalter_L;
        timer TimerMRSM, TimerZGW, TimerSAMH, TimerHL, TimerSAMV,
              TimerVL,TimerTSG, TimerAL;
        port SGPortIn MRSMIn, ZGWIn, SAMHIn, HLIn, SAMVIn, VLIn, TSGIn, ALIn;
    }
    testcase Linksblinken() runs on BlinkenSystem {
        :
        // Innerhalb 20ms muss CAN-Signal auf MRSM
        TimerMRSM.start(20E-3);
        alt {
            [] MRSMIn.receive(2) {setverdict(pass);}
            [] MRSMIn.receive() {setverdict(fail);}
            [] TimerMRSM.timeout {setverdict(fail);}
        }
        // Innerhalb 20ms muss CAN-Signal von CAN-C auf CAN-B
        TimerZGW.start(20E-3);
        alt {
            [] ZGWIn.receive(2) {setverdict(pass);}
            [] ZGWIn.receive() {setverdict(fail);}
            [] TimerZGW.timeout {setverdict(fail);}
        }
        // Prüfe, ob die rechten Blinker aus bleiben
        if (Blinker_VR.DC < 1) {setverdict(pass);} // vorne rechts
        else {setverdict(fail);}
        if (Blinker_MR.DC < 1) {setverdict(pass);} // mitte rechts
        else {setverdict(fail);}
        if (Blinker_HR.DC < 1) {setverdict(pass);} // hinten rechts
        else {setverdict(fail);}
        if (Blinker_AR.DC < 1) {setverdict(pass);} // Anhänger rechts
        else {setverdict(fail);}
        // Innerhalb 15ms muss CAN-Signal auf SAM_H
        TimerSAMH.start(15E-3);
        alt {
            [] SAMHIn.receive(2) {setverdict(pass);}
            [] SAMHIn.receive() {setverdict(fail);}
            [] TimerSAMH.timeout {setverdict(fail);}
        }
        // Prüfe die Ansteuerung der Blinkleuchten hinten links
        while (CAN_B.SAM_H.Schalter.Blinkerschalter_Pos == 2) {
            TimerHL.start(30E-3);
            alt {
                [Blinker_HL.DC > 50] HLIn.receive() {setverdict(pass);}
                [Blinker_HL.DC <= 50] HLIn.receive() {setverdict(fail);}
                [] TimerHL.timeout {setverdict(fail);}
            }
            TimerHL.start(30E-3);
            alt {
                [Blinker_HL.DC < 50] HLIn.receive() {setverdict(pass);}
                [Blinker_HL.DC >= 50] HLIn.receive() {setverdict(fail);}
                [] TimerHL.timeout {setverdict(fail);}
            }
        }
        // Prüfe die Ansteuerung der Blinkerleuchten vorne links
        // Innerhalb 15ms muss CAN-Signal auf SAM_V
        TimerSAMV.start(15E-3);
        alt {
            [] SAMVIn.receive(2) {setverdict(pass);}
            [] SAMVIn.receive() {setverdict(fail);}
            [] TimerSAMV.timeout {setverdict(fail);}
        }
```

**Abbildung 5.8:** Beispielimplementierung des Richtungsblinkertests in TTCN 3 [Zha08]

#### Zusammenfassung

TTCN-3 ist durch seine Modularität und den definierten externen Schnittstellen flexibel gehalten. Die Unabhängigkeit von verwendeter Hardware und Software ist dadurch gegeben. Genauso die Erweitbarkeit für zukünftige Anforderungen. Laut [Gra02] fehlt eine Methodik für eine Herangehensweise bei der Erstellung von Tests.

Die Wiederverwendbarkeit ist durch TRI und TCI gegeben. Die TRI und TCI sind voneinander entkoppelt konzipiert, so dass unterschiedliche Kombinationen von Software und Hardware eingesetzt werden können. Die Eindeutigkeit ist durch die formale Grundlage hinter TTCN 3 sichergestellt. Die textuelle Darstellung der Testbeschreibung ist unübersichtlich, damit unleserlich und schwierig zu verstehen. Spezielle Erweiterungsmöglichkeiten bietet TTCN 3 nicht an. Zur Ergebnisbeurteilung gibt es eine beschränkt definierte Menge von Möglichkeiten.

### 5.3.3 Auszeichnungssprachen

#### Grundlagen

Auszeichnungssprachen (Markup Languages) werden für die Beschreibung von Daten benutzt. Der bekannteste Vertreter ist die XML.

In XML wird im ersten Schritt für die Daten ein Schema definiert. Auf Grundlage dieses Schemas können die eigentlichen Daten beschrieben werden. Die so beschriebenen Daten können durch den Rechner weiter transformiert oder verarbeitet werden.

Als eine auf Datenhaltung konzipierte Sprache muss es für eine Ablaufbeschreibung um weitere Elemente wie Kontrollstrukturen erweitert werden, d.h. die Kontrollstrukturen müssen abgebildet (modelliert) werden [KST04].

#### Erweiterung

Die Arbeit [KH08] stellt eine Repräsentation der Testfälle in XML vor, die auf den Sprachelementen der Testfallbeschreibung nach TTCN3 aufbaut.

Auf der Basis von XML bestehen TestML[12] und ATML[13] für den Austausch bzw. die Beschreibung von Testfällen.

Zuerst folgt eine Betrachtung von TestML. Ziel der Arbeit ist es eine plattformunabhängige Beschreibung zu ermöglichen.

Anschliessend erfolgt eine Betrachtung von ATML. Die ebenfalls auf XML basierte Testfallbeschreibungssprache verfolgt jedoch das Ziel, abstrakte Testfälle beschreiben zu können.

### TestML

**Grundlagen** Die TestML (Test Markup Language) ist eine XML basierte Beschreibung, die entwickelt wurde, um eine einheitliche Schnittstelle im heterogenen Umfeld zu etablieren [GCF+06].

Für die Unterstützung der verschiedenen HW- und SW-Architekturen bedient man sich zweier Konzepte: Zum einen der Abbildung (mapping) der internen definierten Sprachelemente auf die vorhandenen Systeme. Zum anderen definiert man einen *common sense*, eine Menge an Konzepten, die alle Systeme haben müssen. Die Elemente der Sprache umfassen unter anderem Modellierungsmöglichkeiten für Daten (für Stimuli und Reaktion) und Testdurchführungen.

Das Abstrakte Testsystem dient als Grundlage für die Tests. Dieser muss bei allen Testfällen vorhanden sein, um überhaupt sinnvoll Tests durchführen zu können. So sind die Komponenten des Systems bei allen unterstützten Systemen ebenfalls - auf die eine oder andere Art - vorhanden. Das Abstrakte Testsystem besteht aus einer Simulationseinheit um den SUT die notwendigen Stimula für eine Testdurchführung abzusetzen. Die Komponente SUT ist für die Definition der Schnittstellen zum eigentlichen SUT vorhanden. Weitere Komponenten sind die Aufzeichnungs- und Auswertungseinheiten. Die Aufzeichnungseinheit speichert die Ausgaben des SUTs ab und die Auswertungseinheit dient dazu die gespeicherten Ergebnisse auszuwerten.

Beschreibung der Daten ist gegeben: Primitive Datentypen wie Boolean, Integer, Double u.a. werden unterstützt. Grundlegende Operatoren sowie Vergleiche und Verknüpfungen auf diesen sind ebenfalls gegeben.

Die Beschreibung von Wellenformen, wie z.B. einem Sinussignal ist möglich. Eine Reihe von grundlegenden Signalen wird für die Modellierung zur Verfügung gestellt.

[12] www.immos-project.de/site_immos/de/cont_immos/testml.php
[13] http://grouper.ieee.org/groups/scc20/tii/

Komplexe Signale werden durch Komposition von den einfachen Signalen verwirklicht [GM06].

Die Testdurchführung wird als Test Behavior bezeichnet. Hier legt man das Verhalten eines Testfalls fest. Sie geschieht analog zum Konzept von TPT unter Verwendung der hierarchischen Automaten [GCF+06].

Hierbei werden Kontrollflüsse durch die Automatendarstellung modelliert [GM06].

**Beispiel** Abbildung 5.9 zeigt einen Auszug einer Beispielimplementierung des Richtungsblinkertests in TestML.

```
<?xml version="1.0" encoding="UTF-8"?>
<testml>
<!-- Wurzelelement -->
    <testSequence ID="Test_RBL" name="Richtungsblinken: Links">
    <!-- Beschreibung des gesamten Tests, in diesem Fall in Nicht-Echtzeit, da kein
  Regelkreis vorliegt -->
        <declarations>
        <!-- Globale Deklaration von Schnittstellen und Variablen -->
            <testInterface>
            <!-- Abgriffpunkte an den Eingängen der einzelnen Steuergeräte -->
                <port ID="P_MRSM_In" type="double" direction="output" />
                <port ID="P_ZGW_In" type="double" direction="output" />
                <port ID="P_SAMH_In" type="double" direction="output" />
                <port ID="P_TSG_In" type="double" direction="output" />
                <port ID="P_SAMV_In" type="double" direction="output" />
                <port ID="P_AAG_In" type="double" direction="output"/>
                <port ID="P_VL_In" type="double" direction="output" />
                <port ID="P_ML_In" type="double" direction="output"/>
                <port ID="P_HL_In" type="double" direction="output" />
                <port ID="P_AL_In" type="double" direction="output" />
                <port ID="P_VR_In" type="double" direction="output"/>
                <port ID="P_MR_In" type="double" direction="output"/>
                <port ID="P_HR_In" type="double" direction="output"/>
                <port ID="P_AR_In" type="double" direction="output"/>
                <!-- Eingänge für die Testbedingungen -->
                <port ID="P_Batterie" type="integer" direction="input" />
                <port ID="P_Zuendschlüssel" type="integer" direction="input" />
                <port ID="P_Blinkerschalter_L" type="integer" direction="input"/>
            </testInterface>
            <!-- Variablen für die Testbedingungen -->
            <signal ID="Batterie"/>
            <signal ID="Zuendschlüssel"/>
            <signal ID="Blinkerschalter_L"/>
            <!-- Variablen für die Bus-Signale -->
            <signal ID="MRSM_In"/>
            <signal ID="ZGW_In"/>
            <signal ID="SAMH_In"/>
            <signal ID="SAMV_In"/>
            <signal ID="TSG_In"/>
            <signal ID="AAG_In"/>
            <signal ID="VL_In"/>
            <signal ID="ML_In"/>
            <signal ID="HL_In"/>
            <signal ID="AL_In"/>
            <signal ID="VR_In"/>
            <signal ID="MR_In"/>
            <signal ID="HR_In"/>
            <signal ID="AR_In"/>
            <!-- Definition der maximal erlaubten Verzögerung vor dem Blinken nach
  Steuergerät -->
            <timer ID="BLINK_delay">
                <time>
                    <unit>ms</unit>
                    <integer>
                        <value>30</value>
                    </integer>
                </time>
            </timer>
            <!-- Definition des halben Blink-Intervalls -->
            <timer ID="BLNK">
                <time>
                    <unit>ms</unit>
                    <integer>
                        <value>320</value>
                    </integer>
                </time>
            </timer>
        </declarations>

        <stimulus>
        <!-- Die Ansteuerung des zu testenden Systems wird in einem endlichen Automaten
  unter Eingabe und Verarbeitung der Signale dargestellt, in diesem Fall nur als Steuerung -->
            <interfaceMapping>
            <!-- Eingabesignale werden Variablen zugeordnet um sie ansprechen zu können -->
                <map>
                    <portRef IDREF="P_Batterie"/>
                    <signalRef IDREF="Batterie"/>
                </map>
                <map>
                    <portRef IDREF="P_Zuendschlüssel"/>
                    <signalRef IDREF="Zuendschlüssel"/>
                </map>
                <map>
                    <portRef IDREF="P_Blinkerschalter_L"/>
                    <signalRef IDREF="Blinkerschalter_L"/>
                </map>
            </interfaceMapping>

            <behavior>
            <!-- Endlicher Automat -->
                <start ID="STI_START">
                <!-- Anfangspunkt -->
                    <switch>
                        <succ>
                            <stepRef IDREF="STI_STEP_1"/>
                        </succ>
                    </switch>
                </start>

                <step ID="STI_STEP_1">
                <!-- Das SUT wird mit Strom versorgt und 5 Sekunden gewartet, bis alles
  initialisiert ist -->
                    <write>
                        <signalRef IDREF="Batterie"/>
                        <value>1</value>
                    </write>

                    <switch>
                        <cond>
                            <timer ID="bat_delay">
                                <time>
                                    <unit>s</unit>
                                    <integer>
                                        <value>5</value>
                                    </integer>
                                </time>
                            </timer>
                        </cond>
```

**Abbildung 5.9:** Beispielimplementierung des Richtungsblinkertests in TestML

**Zusammenfassung** TestML basiert auf XML, einer Auszeichnungssprache, die unter General Purpose Mittel anzusiedeln ist. Durch die Adaptierbarkeit der Auszeichnungssprache können alle Bewertungskriterien erfüllt werden. Speziell TestML bietet jedoch geringen Umfang an. Das ist besonders durch den *common sense* begründet, welches zum Ziel hat, die kleinste notwendige Gemeinsamkeit zu definieren, um die beschriebenen Testfälle portabel zu gestalten.

### ATML

**Grundlagen** ATML [Ins05] ist ein kommender Standard (Stand Anfang 2010) für die Beschreibung von Testfällen. Sie besteht aus mehreren Teilstandards für die Beschreibung der Ergebnisse, der Testkonfiguration, der Daten, der Testumgebung und der Schnittstellen. Weiterhin ist ein Teilstandard für die Testfallbeschreibung definiert.

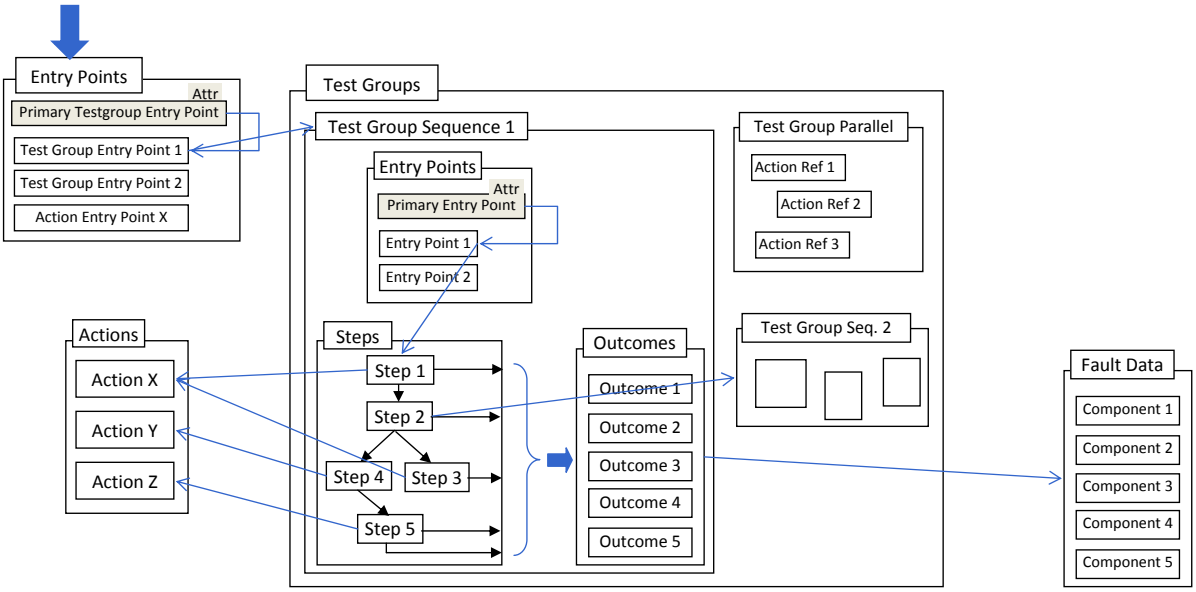



**Abbildung 5.10:** Schema der ATML Testfälle

Abbildung 5.10 zeigt das Schema der Testfälle in ATML auf.

Aktionen definieren die möglichen Testschritte (Aktionen). Eine Testgruppe definiert eine Sequenz oder einen parallelen Ablauf von Schritten, die wiederum aus Teilschritten (Aktionen) bestehen können oder auf andere Testgruppen verweisen. Für jeden Schritt wird eine Reihe von erwarteten Ergebnissen (Outcomes) definiert. Für die Ergebnisse können wiederum eine Reihe von Daten (Fault data) definiert werden.

Die einzelnen lose in der XML beschriebenen Gruppen, werden durch Einstiegspunkte (Entry Points) miteinander in Beziehung gesetzt, so dass sich am Schluss eine

durchgängige Testfallbeschreibung ergibt. Die Aktionen können so, einmal definiert, jederzeit wieder neu als eine Testgruppe in Beziehung gesetzt werden.

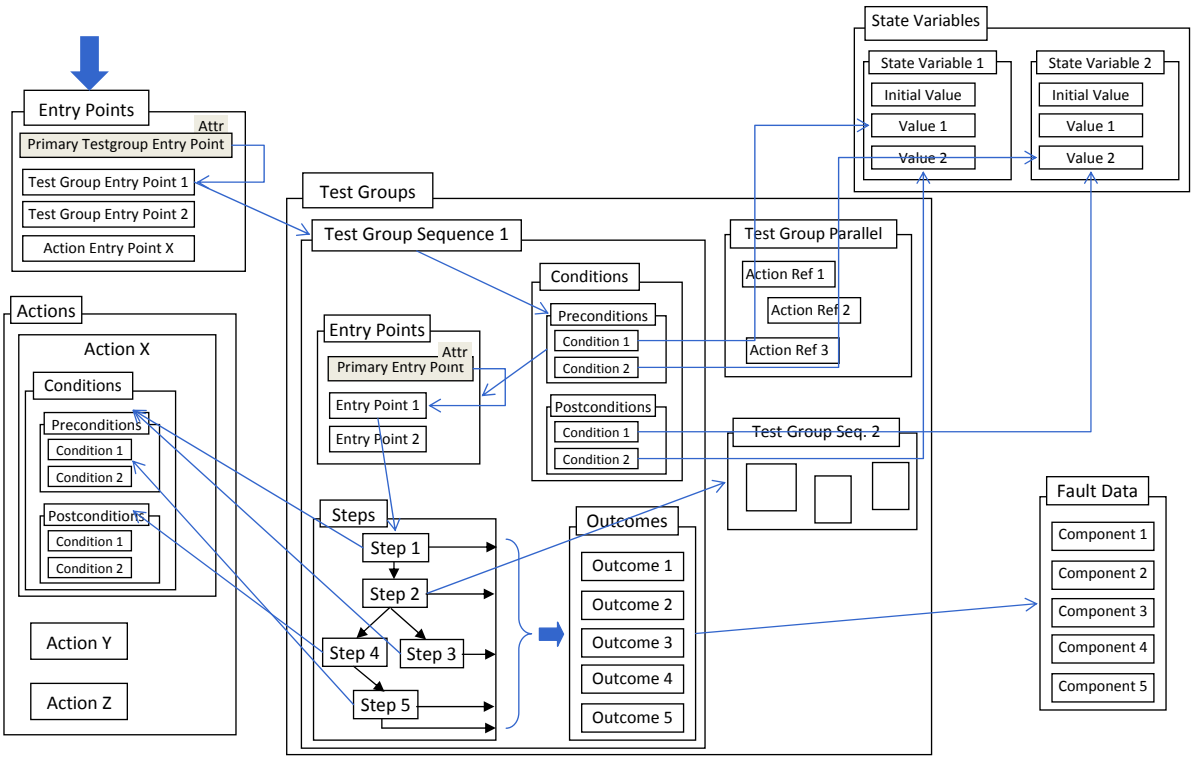



**Abbildung 5.11:** Schema der ATML Testfälle mit Angabe von Bedingungen

Zusätzlich können zu den Aktionen und Testgruppen Bedingungen angegeben werden. Abbildung 5.11 zeigt den Einsatz von Bedingungen. Eine Vorbedingung wird vor der Ausführung einer Aktion evaluiert und eine Nachbedingung nach diesem. Die Bedingungen greifen dabei auf Zustandsvariablen zu. In XML sind diese wieder lose im Dokument definiert und werden durch eindeutige Identifier verwendet.

Parameter sind in Aktionen möglich und können aus unterschiedlichen Quellen referenziert werden. So können innerhalb der Aktion Parameter definiert werden, auf Parameter der Testergebnisse oder auf Parameter anderer Aktionen oder Testgruppen zugegriffen werden.

Neben dieser abstrakten Beschreibung existieren Operationen, die eher der Testfallimplementierung zuzuordnen sind. Operationen für Signalmanipulation, dem Vergleich von Werten und der Definition von Kontrollstrukturen (Bedingung, Wiederholung und Variablensetzung), sowie Operationen für Nachrichtenaustausch sind definiert. Eine Operation *OperationOther* ist für den Einsatz beliebiger selbstdefinierter Operationen spezifiziert.

**Kriterien** Abbildung 5.12 zeigt die Abdeckung der Bewertungskriterien durch ATML. Eine detaillierte Betrachtung der einzelnen Sprachelemente ist in Abbildung 5.22, S. 171 im Vergleich zu den anderen untersuchten Mitteln zu finden.

| | | Gewichtung | ATML |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 17 |
| | Unterstützung Testfallparadigmen | 4 | 2 |
| | | | 76 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | textuell |
| | Formalismus | 2 | semi-formal |
| | | | 3 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | ja |
| | | | 1 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | ja |
| | Sprachelemente für Platzhalter | 1 | ja |
| | Sprachelemente für Templates | 1 | nein |
| | Sprachelemente für Reaktionsauswertung | 1 | ja |
| | Sprachelemente für Hierarchisierung | 2 | ja |
| | Darstellung von Metainformationen | 2 | ja |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | nein |
| | | | 7 |
| | | | 87 |

**Abbildung 5.12:** Abdeckung der Bewertungskriterien durch ATML

**Beispiel** Der Testfall wird ähnlich dem Beispiel aus TestML in XML spezifiziert. Da das grundlegende Schema in den Abbildungen 5.10 und 5.11 aufgezeigt ist, wird hier auf eine ähnlich aussehende Darstellung, wie in der Abbildung 5.9 zu sehen, verzichtet.

**Zusammenfassung** ATML verfolgt das Ziel abstrakte Testfälle zu beschreiben. Dabei werden hinreichend genug Elemente (Operationen) angeboten um auf abstraktem Niveau konkret genug zu bleiben. Von den Kriterien her deckt ATML viele Sprachelemente auf der Ebene der Testfallbeschreibung ab. Einzig und allein ist die Darstellung in XML undurchsichtig und kann ohne werkzeugunterstütze Hilfsmittel nicht bearbeitet werden. Für den Einsatz als Testfallbeschreibungssprache ist daher eine XML basierte Beschreibung nicht ohne weiteres einsetzbar.

### 5.3.4 UML basierte Ansätze

#### Grundlagen

Die UML[14] (Unified Modelling Language) ist entwickelt worden um Softwaresysteme zu modellieren, zu spezifizieren, zu dokumentieren und zu visualisieren. Sie bietet für die Modellierung und Spezifikation von Software in visueller Form dreizehn Diagrammtypen an. Es existieren Ansätze, die UML nicht nur für die Entwicklung sondern auch für das Testen zu verwenden ( [OMG05], [Har01b], [Ole08]).

In Abbildung 5.13 sind die dreizehn UML Diagrammtypen dargestellt. Grob werden die Diagramme in Strukturdiagramme und Verhaltensdiagramme aufgeteilt. Bei den Strukturdiagrammen werden die statischen Aspekte betrachtet. Die Verhaltensdiagramme spiegeln die Interaktion wieder, d.h. bei diesen Diagrammtypen können auch Abläufe modelliert werden.

Use Case Diagramm: Bei diesem Diagramm wird die Frage „Welche Dienstleistung bietet das System nach außen hin an?“ beantwortet. Dabei wird auf einem sehr hohen Abstraktionsniveau die externen Schnittstellen zum System beschrieben und die externen beteiligten Akteure benannt, die das System benutzen.

Aktivitätsdiagramm: Die Aktivitätsdiagramme dienen der Darstellung eines Ablaufs, sei es ein Prozessablauf oder ein Ablauf eines Algorithmus. Dabei können Sequenzen, Nebenläufigkeiten, Bedingungen, Schleifen und Verzweigungen modelliert werden. Eine Sycnhronisation der Nebenläufigkeiten ist ebenfalls möglich.

Zustandsdiagramm: Zustandsdiagramme stellen die möglichen Zustände eines Objekts dar. Hier können Sequenzen, Nebenläufigkeiten und Bedingungen modelliert werden und Ereignisse, die extern oder intern auftreten.

Sequenzdiagramm: Diese Diagramme stellen in einem zeitlichen Verlauf die Kom-

[14] http://www.uml.org

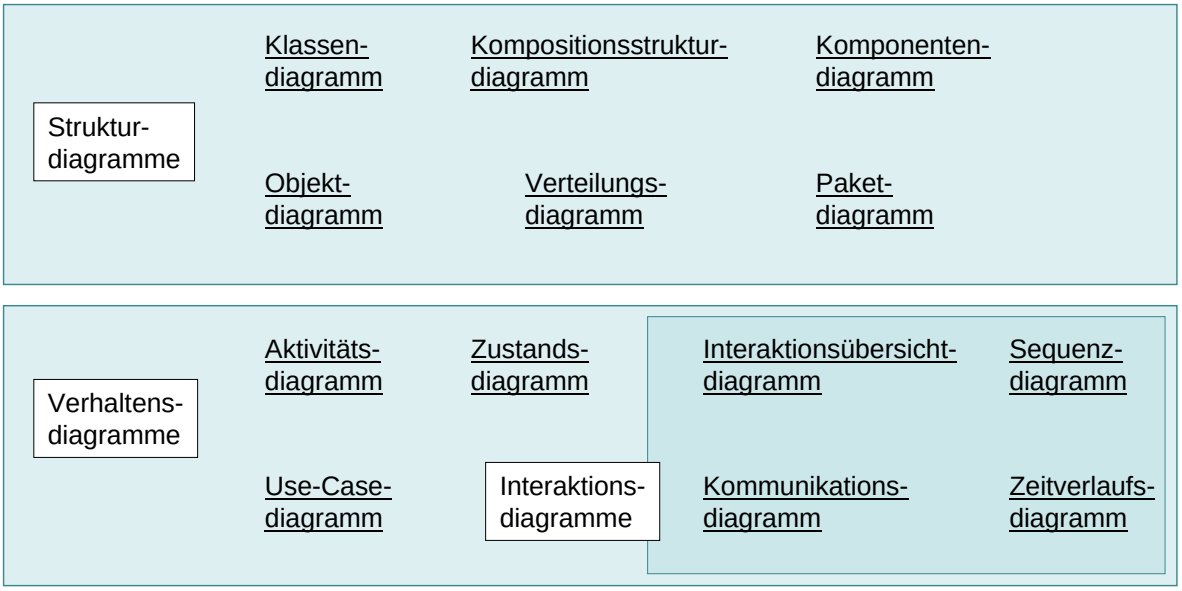



**Abbildung 5.13:** Die dreizehn Diagrammtypen bei UML 2.0

munikation zwischen Objekten dar. Seit der UML 2.0 sind die grundlegenden Modellierungsmöglichkeiten für Kontrollstrukturen ebenfalls vorhanden.

Kommunikationsdiagramm: Sind ähnlich den Interaktionsdiagrammen. Die Beziehungen zwischen den Objekten können betrachtet werden. Der zeitliche Ablauf ist zweitrangig.

Timingdiagramm: Die Zeit ist der Aufhängepunkt, wo die Zustände von Objekten zugeordnet werden. Damit wird deutlich zu welchem Zeitpunkt welches Objekt sich in welchem (internen) Zustand befindet.

Interaktionsübersichtsdiagramm: Die Interaktions-, Kommunikations- und Timingdiagramme können in diesem Diagramm in Beziehung gesetzt werden.

Besonders die Aktivitätsdiagramme sind mit UML 2 stark überarbeitet worden. Das Metamodell wurde auf Basis der Petri-Netze neu modelliert und baut nicht mehr auf den Elementen der Zustandsdiagramme auf.

### Erweiterungen

Die Diagramme der UML werden in unterschiedlichen Arbeiten als Grundlage hergenommen. [PJ04] zeigt die Verwendung der UML Sequenzdiagramme als Testfallbeschreibung auf. In [BG08] werden die Sequenzdiagramme in Verbindung mit den Zu-

standsdiagrammen verwendet, um Ausführungspfade zu ermitteln, die als Testfälle verwendet werden können. Dabei wird das externe Verhalten des Systems in Verbindung zu dem internen Verhalten gesetzt.

In diesem Abschnitt werden unterschiedliche Arbeiten zum Thema Testbeschreibung mit UML untersucht, die im Automobilen Bereich entwickelt wurden.

**Testbeschreibung nach HARTMANN** In der Arbeit von [Har01b] ist ein Konzept für eine Testbeschreibung im Automobilen Umfeld herausgearbeitet. Bei der Betrachtung werden HIL und SIL Systeme untersucht bzw. immer wieder Bezug dazu genommen. Die Grundidee dabei ist, dass jeder Test in einer bestimmten Umgebung läuft. Diesen Testkontext teilt der Verfasser in den Testaufbau und die Testvoraussetzungen auf. Der Testaufbau beinhaltet neben dem Testobjekt auch solche Objekte, die für die Beschreibung relevant sind. So wird der Testaufbau aus der Umgebung und dem Testobjekt definiert.

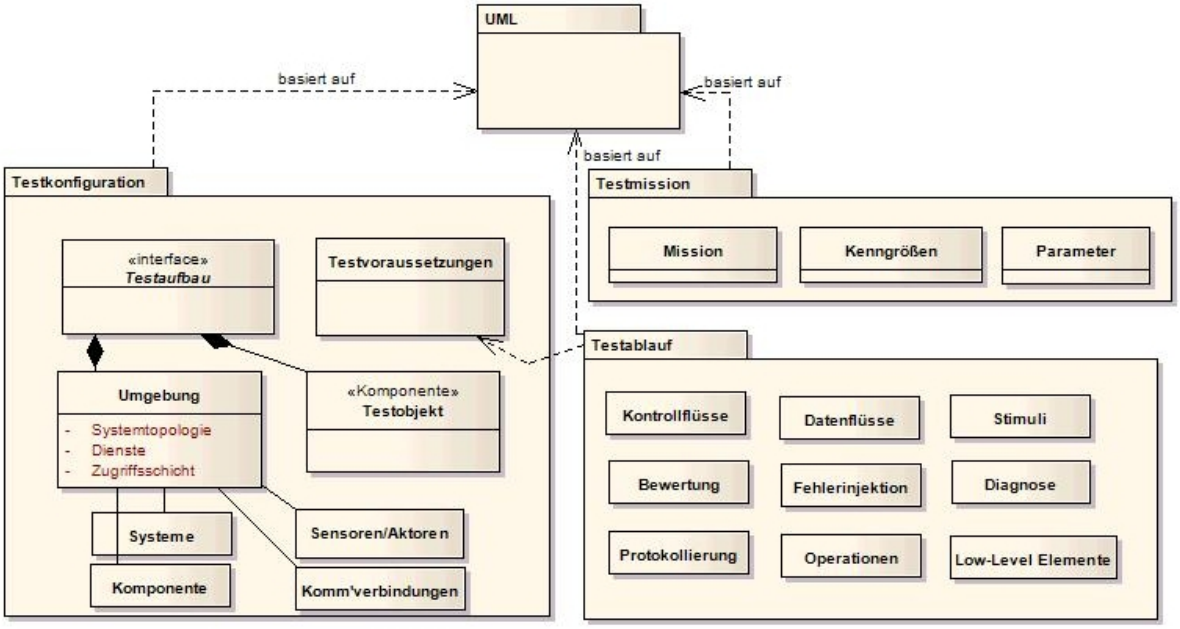



**Abbildung 5.14:** Aufbau der Testbeschreibung von [Har01b]

In Abbildung 5.14 ist der strukturelle Aufbau der Testbeschreibung modelliert. In der Arbeit werden drei Hauptbestandteile definiert: Die Testkonfiguration, die Testmission und der Testablauf. Die Testkonfiguration beinhaltet die Definition der Umgebung und die Testvoraussetzungen. Dabei wird modelliert wie die Systemtopologie aussieht, d.h. die beteiligten (benötigten) Komponenten werden modelliert und ihre Beziehung zueinander dargestellt. Zusätzlich wird definiert welche Sensoren und Aktoren die Komponenten besitzen, die für die Testfallbeschreibung relevant ist. Die

Kommunikationsverbindungen, wie der CAN Bus, werden bei Relevanz auch eingebettet.

Ein Schwerpunkt der Arbeit ist die Bindung der Testfälle an die Funktionen des Testobjekts bzw. an genau die Funktionalität, die durch den Test gedeckt werden soll. Hierfür ist die Testmission zuständig. In dieser wird in Prosatext, der Zweck definiert. Gleichzeitig werden für die Auswertung bzw. Bewertung benötigen Kenngrößen aufgezählt. Für die Unterstützung verschiedener Steuergerätevarianten können in der Testmission Parameter angegeben werden.

Der Testablauf nimmt Bezug auf die Testvoraussetzungen der Testkonfiguration, da durch die Testvoraussetzung immer der gleiche Zustand vor der Ausführung der Testablaufs erreicht wird. Für die Modellierung des Testablaufs werden Kontrollflüsse, Datenflüsse, Stimuli, Bewertung, Fehlerinjektion, Diagnose, Protokollierung und Operationen definiert. Des Weiteren werden low-level Sprachelemente definiert. So sind Kompositionen für Signalerstellung, arithmetische und logische Operationen, Filter, Trigger, Zuweisungen und Nebenläufigkeit bezogene Prozesskontrollelemente definiert (z.B. Join, Fork, Sync, Trap, Terminate, Exit,...).

In der Arbeit [Hut06] wird „eine Systematik zur Erstellung virtueller Steuergeräte für Hardware-in-the-Loop-Integrationstests“ vorgestellt. Die ziemlich umfassende Arbeit beschäftigt sich in einem Abschnitt auch mit Testfallbeschreibungen. Dabei beschreibt der Verfasser auf Grundlage von TTCN3 und dem [Har01b] Konzept wie eine Testfallbeschreibung auszusehen hat. Wie der Titel der Arbeit einschränkt, bilden HIL Systeme das Ziel-Testsystem.

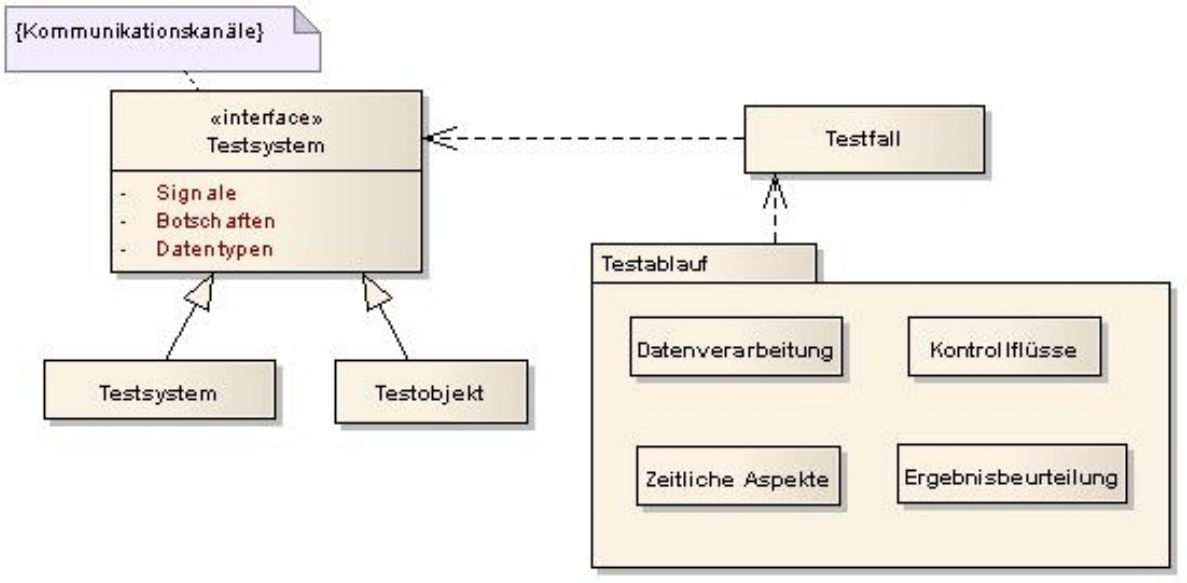



**Abbildung 5.15:** Aufbau der Testbeschreibung von [Hut06]

In der Abbildung 5.15 ist der Aufbau des Konzepts von [Hut06] wiedergegeben.

Zitat: „Das Testsysteminterface ist somit bei HIL-Simulatoren vollständig durch die Signaldatenbasis beschrieben.“. Das heisst, dass auf alle notwendigen Signale und Botschaften in einem HIL System durch Setter und Getter-Befehle verändert bzw. zugegriffen werden kann. Diese Abstraktion macht eine detaillierte Analyse des Testobjektes bzw. des Testsystems hinsichtlich der Kommunikationskanäle überflüssig. Datentypen werden angeboten und sind wieder an TTCN3 angelehnt.

Für den Testablauf sind Sprachelemente für Kontrollstrukturen, zeitliche Aspekte und Ergebnisbeurteilung konzipiert. Daneben sind datenverarbeitende Sprachelemente vorhanden. Beispiel hierfür sind die arithmetische und logischen Operationen und die mathematischen Standardfunktionen (sin, cos, ...) für die Erstellung von Signalen.

Der typischer Aufbau von Testfällen in dem Automobilen Umfeld wird ebenfalls erwähnt: Die (gleichzeitige) Stimulation der Eingänge des Testobjekts und die darauf folgende Reaktion des Testobjekts. Dabei werden mehrere Reaktionen gleichzeitig überprüft.

**UT2P** UML2.0 Testing Profile (UTP) ist eine Erweiterung der UML, damit man automatisch aus UML2.0 Beschreibungen Testfälle generieren kann. Profile (lightweight extensions) sind neben der Metamodellveränderung (heavyweight extension) eine Möglichkeit um UML zu erweitern. Bei den Profilen wird im Gegensatz zu der Metamodellveränderung kein Eingriff in die Modellierung von UML vorgenommen, so dass die Erweiterung von den vorhanden Werkzeugen ohne Aufwand unterstützt werden kann [PP05].

Bei den Profilen wird mit den zur Verfügung gestellten Mitteln (Stereotypen, Tagged Values und Constraints) UML um die Möglichkeit erweitert Testfälle zu modellieren. Die erstellten Modelle sind nicht ohne eine Abbildung ausführbar. Im Moment können die Testmodellierungen nach TTCN-3 und JUnit[15] umgewandelt werden. Als Aufsatz für die beiden Lösungen unterstützt U2TP auch nicht den Umfang, den TTCN-3 bzw. JUnit anbieten. Man kann z.B. TTCN-3 nicht nach UTP abbilden [SG03] oder alle Konzepte von UTP nach JUnit übertragen. Letzteres deswegen nicht, da UTP Integrations- und Systemtests unterstützt, JUnit aber nur für Unittests gedacht ist [BJK05].

Für die Modellierung können die Interaktionsdiagramme, die Zustandsdiagramme und die Aktivitätsdiagramme verwendet werden.

[15] http://www.junit.org

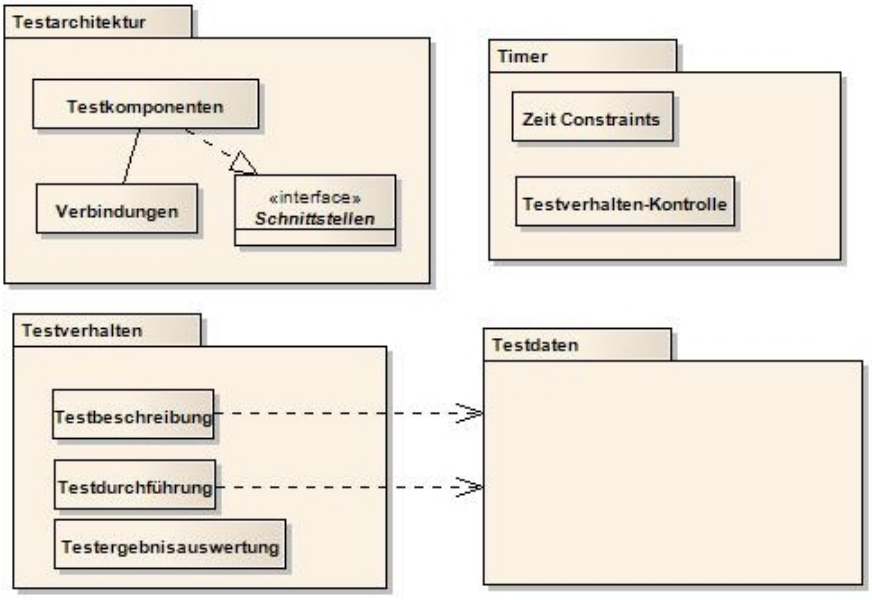



**Abbildung 5.16:** Aufbau von UTP

Es gibt vier definierte Konzepte: Die Testarchitektur, das Testverhalten, die Testdaten und die Zeitbetrachtung.

In der Testarchitektur werden die Testkomponenten und die Testumgebung festgelegt. Die Testumgebung beinhaltet die Testkonfiguration und Testkontrolle. In der Testkonfiguration werden die zu verwendenden Testkomponenten, deren Anbindung an die SUT (System Under Test), sowie ihre Anfangszustände definiert. Die Testkomponenten sind die mit dem SUT interagierenden Komponenten auf der Testseite um die Testfälle durchführen zu können. Mit der Testkontrolle beschreibt man den Einfluss der Testfälle auf die weitere Durchführung des Testens. Beispielsweise könnte eine erfolgslose Durchführung eines Testfalls die Durchführung der anderen Testfälle überflüssig machen und der Test wird beendet.
Ein sogenannter Arbiter kann für die Bewertung definiert (programmiert) werden. Hier können Zusammenhänge und Auswirkungen von Testfällen algorithmisch implementiert werden um die Auswertung der Testdurchführung zu automatisieren. Damit der Arbiter von den Testfallergebnissen mitbekommt, werden die Teilergebnisse diesem für die Gesamtauswertung mitgeteilt. Ein sogenannter Scheduler informiert den Arbiter über beendete Testkomponenten und kontrolliert die Ausführung der Testkomponenten.

Im Testverhalten werden die Testfälle definiert. Zusätzlich werden Standardverhalten für unerwartete Ereignisse definiert. Die Ergebnisse der Testfälle können pass, fail, inconclusive oder error sein. Durch einen Logger können die Testfälle bei ihrer Durchführung dokumentiert werden (trace).

Eine verständliche, allgemeine Testfall-Spezifikationssprache für das funktionale Steuergerätetesten

Für die Testdaten werden drei Aspekte betrachtet: Ein Aspekt betrachtet die Spezifikation der Daten, sowie deren De- und Enkodierung bei der Testdurchführung. Für die Datenbeschreibung können ASN.1, CORBA und XML eingesetzt werden.
Der zweite Aspekt kümmert sich um die Aufarbeitung und Bereitstellung der Testdaten. Die Daten können in Datenpools hinterlegt werden, wo sie mit Testinhalten und Testfällen verknüpft werden. Weiterhin ist eine Partitionierung der Daten möglich. Mit Datenselektoren, die Funktionen darstellen, können Testdaten aus dem Datenpool bzw. aus den Partitionen ermittelt werden.
Der dritte Aspekt behandelt vor allem die ankommenden Daten. Diese können mit Hilfe von sog. Wildcards (Platzhaltern) im breiten Rahmen erschlossen werden.

Neben der von der UML2.0 zur Verfügung gestellten (einfachen) Modellierungsmöglichkeiten für die Zeit (Erfassung der Zeit und der Dauer) gibt es von UTP ein Timer Konzept, das Klassen zugeordnet ist und von diesen gestartet, abgefragt und gestoppt werden kann. Bei einem Timeout wird automatisch eine Nachricht generiert.
Um verteiltes Testen zu unterstützen gibt es das Konzept der Zeitzonen, die obligatorisch sind. So werden die Testkomponenten mindestens einer Zeitzone zugeordnet.
Bei der von UML2 zur Verfügung gestellten Zeitmodellierungsmöglichkeiten gibt es ausser dem Zeitdiagramm keine fest vorgeschriebene Notation.

**Weitere Erweiterungen** Auf den Interaktionsdiagrammen (Sequenzdiagramm) basiert eine Erweiterung (Extended Automation Method, kurz EXAM genannt) für die Beschreibung von Testfällen [KP08]. Hierbei wird auf Bibliotheken in Programmiersprachen auf unterster Ebene mit einer einheitlichen Abstraktionssicht (Interface) aufgebaut. Diese Bibliotheken werden in den Interaktionsdiagrammen als Objekte verwendet, so dass ein „Nachrichtenaustausch“ zwischen zwei Objekten einen Aufruf von einer Funktionsbibliothek darstellt.
Durch die von den UML2 Interaktionsdiagrammen zur Verfügung gestellten Kontrollstrukturen wie Schleifen oder Abfragen können einfache Sequenzen durch diese erweitert werden. Die Absicht wird als Kommentar zu jeder Nachricht in Prosa hinzugefügt.
Durch diese Art der Testfallbeschreibung werden komplexe Abläufe eines Testfalls in die Funktionen übertragen, so dass die Beschreibung auf der Ebene der Interaktionsdiagramme einer Sequenz (von Funktionsaufrufen) entspricht.

### Kriterien

Abbildung 5.17 zeigt die Abdeckung der Bewertungskriterien durch UML2TP. Eine detaillierte Betrachtung der einzelnen Sprachelemente ist in Abbildung 5.22, S. 171 im Vergleich zu den anderen untersuchten Mittel zu finden.

| | | Gewichtung | UML2TP |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 10 |
| | Unterstützung Testfallparadigmen | 4 | 2 |
| | | | 48 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | graphisch |
| | Formalismus | 2 | semi-formal |
| | | | 3 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | ja |
| | | | 1 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | ja |
| | Sprachelemente für Platzhalter | 1 | ja |
| | Sprachelemente für Templates | 1 | ja |
| | Sprachelemente für Reaktionsauswertung | 1 | ja |
| | Sprachelemente für Hierarchisierung | 2 | ja |
| | Darstellung von Metainformationen | 2 | nein |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | nein |
| | | | 6 |
| | | | 58 |

**Abbildung 5.17:** Abdeckung der Bewertungskriterien durch UML2TP

### Beispiel

Abbildung 5.18 zeigt eine Beispielimplementierung des Richtungsblinkertests in UML2TP.

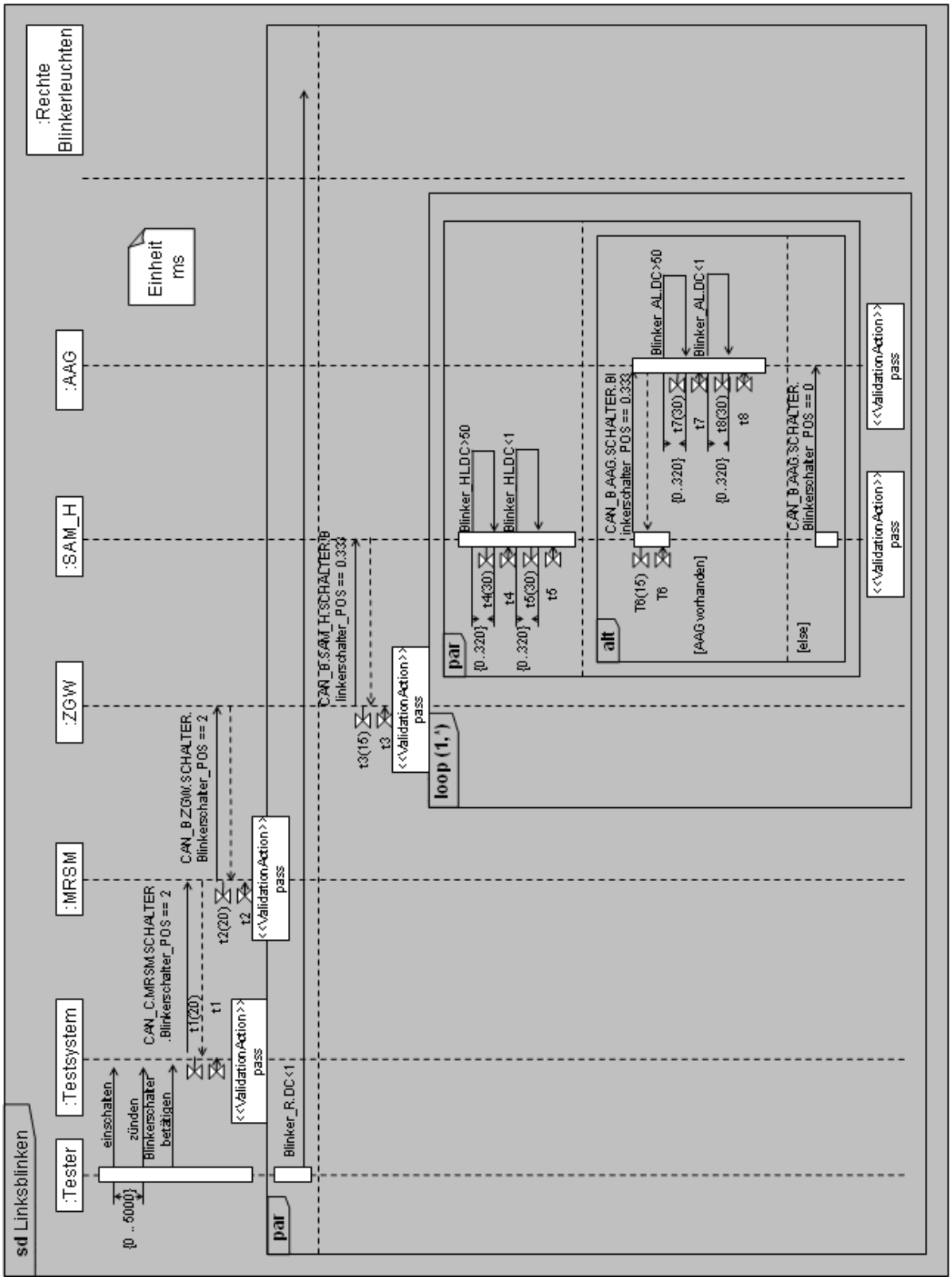



**Abbildung 5.18:** Beispielimplementierung des Richtungsblinkertests in UML2TP [Zha08]

**Zusammenfassung**

Die Modellierung, d.h. die graphische Erstellung der Testbeschreibung nach HARTMANN hat die Schwäche, dass sie testdurchführungsbezogene Sprachelemente mit konzipiert hat. Die Idee der Dienste, die durch Probes vor der Ausführung auf Vorhandensein überprüft werden, lässt sich auf Befehle reduzieren, wenn man die Testbeschreibung rein statisch betrachtet. Das durch Sprachkonstruktionen, die Überprüfung der Lauffähigkeit kontrolliert und garantiert werden soll erschwert die Handhabbarkeit für den Test-Designer. Durch die Annahme, dass die Testfallbeschreibung durch eine 'dritte Instanz' auf Lauffähigkeit überprüft wird, könnten die Dienste und Probes auf einfache Befehle reduziert werden, die sie letztendlich sind. Durch die Modellierung der low-level Sprachelemente und ihre Varianten wie z.B. unsigned integer und signed integer wird zusätzliche Komplexität erzeugt, die die Lesbarkeit und damit das Verständnis erschwert.

Die Arbeit [Hut06] ist ein Zusammenschluss der Arbeiten von [Har01b] und TTCN 3. So lehnt sich die Beschreibung stark an die Syntax von TTCN 3 an, d.h. eine Ähnlichkeit mit einer Programmiersprache ist beabsichtigt. So werden die low-level Sprachelemente nicht getrennt betrachtet. Dadurch erschwert sich die Erstellung einer testsystemunabhängigen Testbeschreibung.

Allerdings liegt auch der Schwerpunkt der Arbeit bei der Definition der benötigten Sprachelemente für eine Testbeschreibung. Weiterhin fehlt die Möglichkeit zur Parametrisierung.

UML2 Testing Profile zielt auf eine logische Beschreibung der Testfälle ab. So werden alle Elemente der logischen Ebene abgedeckt, bis auf die Ausführungsrate. Die Modellierung der Ausführungsrate kann mit einer Schleife erfolgen.

Die Darstellungsform ist graphisch und durch die UML hat UML2TP semi-formalen Charakter.

Wie alle graphischen Beschreibungssprachen wird auch die Beschreibung in UML Diagrammen bei umfangreichen Testfällen zu unübersichtlich.

### 5.3.5 Werkzeugbasierte Ansätze zur Beschreibung

#### Grundlagen

Beim Black-Box Testen, d.h. dem Stimulieren um die Ausgaben eines Testobjekts zu beobachten reichen meist die Erfassung der Eingaben und die Benennung der Sollergebnisse aus. Die Testdaten stehen also im Vordergrund (vgl. [PCG05]).

Hierfür werden tabellen-basierte Anwendungen wie Excel verwendet. Tools wie CTE werden für die Partitionierung der Testdaten verwendet, d.h. für die Herleitung bzw. Erzeugung des Testfalls. [CF05] zeigt einen Überblick über vorhandene Werkzeuge im Automobilen Umfeld.

Für die reine Beschreibung eines Testfalls wird im Folgenden eine tabellenbasierte Darstellung betrachtet. Der Inhalt kann ebenso in Form von XML gespeichert werden, so dass die hier genannten Vor- und Nachteile auch für XML gelten.

#### Erweiterungen

**Tabellenbasierte Testfall-Spezifikation** Bei der Verwendung von Werkzeugen wie Excel wird ein Template bzw. ein Schema für die Testfallbeschreibung erstellt. In Abbildung 5.20 ist so ein Schema zu sehen. Dabei werden Spalten für die Eingabewerte mit den eventuellen Auslöser (ebenfalls Eingabewerte) und den zu erwartenden Ausgabewerten der Signale definiert. Die Spalten stellen dabei die zu manipulierenden Signale dar.
Die Zeilen sind die einzelnen Testschritte. Dabei bezeichnet einen Testschritt eine Menge von Eingabewerten, die genau eine Ausgabe bzw. eine Menge von Ausgabesignalen erzeugt, die logisch in sich abgeschlossen und atomar sind.

Eventuell benötigte Erweiterungen wie Zeitangaben wandern als eine Spalte in ein Testschritt ein, die durch eine Definition entweder den Vor- oder Nachlauf definieren.

Die so beschriebenen Testfälle werden i.d.R. bei der Testfallimplementierung automatisiert verwendet. Da alle benötigten Informationen in einer Tabelle enthalten sind, kann ein Programm (Testskript), die einzelnen Zeilen einer Tabelle entnehmen, diese für die Stimulation des Testobjekts verwenden um anschliessend die Ausgaben von dem Testobjekt mit den Sollwerten, die in der Tabelle definiert wird zu vergleichen. Das Testskript übernimmt somit die Testausführung.

**Partionierung der Testdaten** Eine weitere Möglichkeit die Testdaten zu beschreiben (bzw. aufzuschreiben) ist die vorherige Aufteilung des Eingabebereiches jeder relevanten Signale in Bereiche, die bei der Stimulation die gleiche Reaktion hervorrufen. Dabei kommt die Testfallspezifikationstechnik *Äquivalenzklassen* zum Einsatz. Die so entstehenden Eingabebereiche können anschliessend in Verbindung gesetzt werden um sinnvolle Teststimulationen zu erstellen. Das Ergebnis ist eine Aufteilung der Eingabesignale in Äquivalenzklassen und die Verknüpfung der unterschiedlichen Signalklassen miteinander; die so erstellte Verknüpfung stellt jeweils einen Testschritt dar. Die Testidee wird durch die Testspezifikationstechniken, Äquivalenzklassen und durch die Grenzwertanalyse abgedeckt (vgl. mit der Testabsicht 4.2, S. 100).

Der Ablauf der einzelnen Testschritte zu einem Testfall wird meist sequentiell, eventuell mit Wartezeiten dazwischen, realisiert. Prozeduale Aspekte (z.B. parallele Abläufe oder Schleifen) sind nicht der Fokus.

**Kriterien**

Abbildung 5.19 zeigt die Abdeckung der Bewertungskriterien durch die tabellenbasierte Testfallbeschreibung. Eine detaillierte Betrachtung der einzelnen Sprachelemente ist in Abbildung 5.22, S. 171 im Vergleich zu den anderen untersuchten Mitteln zu finden.

| | | Gewichtung | Tabellenb. |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 8 |
| | Unterstützung Testfallparadigmen | 4 | 1 |
| | | | 36 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | textuell |
| | Formalismus | 2 | informell |
| | | | 1 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | nein |
| | | | 0 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | nein |
| | Sprachelemente für Platzhalter | 1 | nein |
| | Sprachelemente für Templates | 1 | nein |
| | Sprachelemente für Reaktionsauswertung | 1 | nein |
| | Sprachelemente für Hierarchisierung | 2 | nein |
| | Darstellung von Metainformationen | 2 | nein |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | nein |
| | | | 0 |
| | | | 37 |

**Abbildung 5.19:** Abdeckung der Bewertungskriterien durch die tabellen-basierte Testfallbeschreibung

### Beispiel

In Abbildung 5.20 ist eine tabellen-basierte Testfallbeschreibung zu sehen.

### Zusammenfassung

Wie bei jedem General Purpose Mittel (Sprache oder Werkzeug) ist auch bei der tabellenbasierter Form eine Adaption an die Bedürfnisse der Testfallbeschreibung

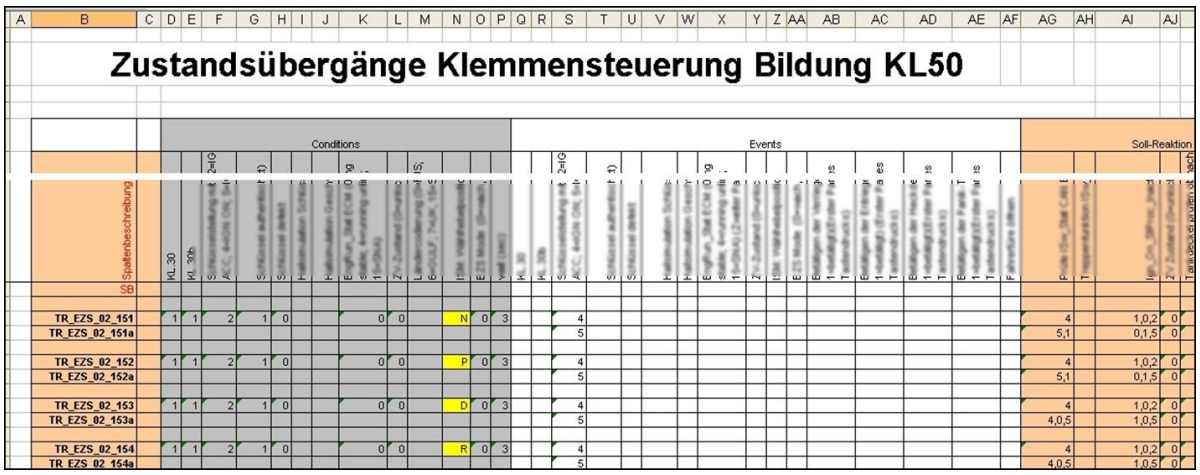



**Abbildung 5.20:** Testdaten basierte Testfallbeschreibung

möglich. Eine reine Betrachtung der tabellenbasierten Darstellung liefert keine der untersuchten Kriterien.

Allerdings wird gerade durch Werkzeuge wie Excel die tabellenbasierte Darstellung bei der Testfallspezifikation in der Industrie am häufigsten eingesetzt. Die Werkzeuge sind bei jedem Rechner mit installiert und die Skriptfähigkeit bei den Werkzeugen ermöglicht eine automatische Bearbeitung der Inhalte.

Die Verständlichkeit einer tabellenbasierten Form schwindet, so wie bei allen Mitteln, je größer die Tabellen werden. Eine reine Fokussierung auf den Inhalt ist nicht mehr möglich. Eventuelle farbliche Markierungen der Stimuli und Sollergebnisse werden zur Bewahrung der Übersicht eingesetzt.

### 5.3.6 Natürliche Sprache zur Testfallbeschreibung

#### Grundlagen

Die natürliche Sprache bietet Vorteile bei der Spezifikation von Testfällen. Die Beschreibung von Testfällen in natürlicher Sprache erlaubt dem Test-Designer die Umsetzung seines mentalen Modells ohne einen Zwischenschritt, z.B. durch 'Denken' in einer Programmiersprache. Dadurch beschreibt jeder Test-Designer die Testfälle für den ihm bekannten Ausdrucksumfang.

Die eigene Sprache bietet die Vorteile das es intuitiv und lesbar ist. Die Nachteile sind aber bekannt: So hat die natürliche Sprache keine formale Grundlage. Die Grammatik und die fülle der Wörter bieten keine Eindeutigkeit.

So kann der (fiktive) Satz „Die Drosselklappe wird zum Schutz vor Überbelüftung durch das Motorsteuergerät kontrolliert“ mehrere Bedeutungen haben. Zum einen wird die Drosselklappe vor dem Motorsteuergeräte geschützt, da diese die Drosselklappe überbelüften könnte. Zum anderen wird die Drosselklappe durch das Motorsteuergerät kontrolliert, damit keine Überbelüftung stattfindet. Schlussendlich sagt der Satz nichts über den Schutz vor Überbelüftung aus, das weder die Drosselklappe noch das Motorsteuergerät zu befürchten haben.

#### Erweiterungen

**Regelwerke:** Im Bereich Requirements Engineering gibt es Ansätze um die natürliche Sprache einsetzen zu können. Dafür werden Regeln definiert, wie Sätze aufgebaut werden sollen. [Rup07] definiert solche Regelwerke für Anforderungen.

**Kontrollierte Sprache:** Eine andere Möglichkeit um die Mehrdeutigkeit in der natürliche Sprache zu mindern, ist die Formalisierung der Sätze. Durch Einschränkungen wird eine sogenannte kontrollierte Sprache als Untermenge zur natürlichen Sprache definiert, die frei von Mehrdeutigkeiten ist. Zusätzlich werden Wörter in einem Wörterbuch definiert. Dabei wird definiert welche Bedeutung sie besitzen und in welcher Form sie eingesetzt werden dürfen. Synonyme, die ein Sachverhalt konkreter bezeichnen und eingesetzt werden müssen, geben dem Benutzer zusätzliche Informationen.

In [Sch98] werden kontrollierte Sprachen für Englisch für die Anforderungsdefinition untersucht. So bleibt der Vorteil der formalen Sprache gegenüber den Spezifikationssprachen wie Z, VDM, Petri-Netze u.a. die Lesbarkeit der natürlichen Sprache. Die Arbeit [Ley05] setzt sich mit kontrollierten Textstrukturen auseinander; ebenfalls ist Englisch die betrachtete Sprache. Keine der Arbeiten betrachtet jedoch ablaufbeschreibende Elemente.

#### Kriterien

Abbildung 5.21 zeigt die Abdeckung der Bewertungskriterien durch die an die natürliche Sprache angelehnte Testfallbeschreibung. Eine detaillierte Betrachtung der einzelnen Sprachelemente ist in Abbildung 5.22, S. 171 im Vergleich zu den anderen untersuchten Mittel zu finden.

| | | Gewichtung | Natürl. Spr. |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 0 |
| | Unterstützung Testfallparadigmen | 4 | 1 |
| | | | 4 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | textuell |
| | Formalismus | 2 | informell |
| | | | 1 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | nein |
| | | | 0 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | nein |
| | Sprachelemente für Platzhalter | 1 | nein |
| | Sprachelemente für Templates | 1 | nein |
| | Sprachelemente für Reaktionsauswertung | 1 | nein |
| | Sprachelemente für Hierarchisierung | 2 | nein |
| | Darstellung von Metainformationen | 2 | nein |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | nein |
| | | | 0 |
| | | | 5 |

**Abbildung 5.21:** Abdeckung der Bewertungskriterien durch eine Testfallbeschreibung in natürlicher Sprache

### Beispiel

Abbildung 5.2 zeigt eine Beispielimplementierung des Richtungsblinkertests in natürlicher Sprache.

```
2 Vorbedingungen
  Als Vorbedingung für den Test, muss das Testsystem in einen
4 vordefinierten Zustand gebracht werden. Dazu wird zunächst
  die Stromversorgung für das SUT (Steuergeräteverbund)
6 eingeschaltet. Danach wartet man 5 Sekunden auf die Initialisierung
  der Steuergeräte, bevor man die nächsten Schritte einleitet.
```

```
Am Schluss wird der Zündschlüssel noch in die entsprechende
Stellung gebracht, da das Richtungsblinken wie mit der Realität
konform nur bei eingeschalteter Zündung geprüft werden kann.

Aktionen
Nach den Initialisierungsschritten des Testsystems wird nun
als Aktion der (simulierte) Blinkschalter in die Stellung
Linksblinken gebracht.

Erwartete Ergebnisse
Zunächst wird in diesem Test Überprüft, ob die Betätigung des
Blinkschalters in Form eines CAN C Signals innerhalb von 20ms
vom MRSM erfasst und anschließend durch ZGW auf CAN B innerhalb
von 20ms geleitet wird. Danach wird betrachtet, ob das Signal
in weniger als 15ms vom SAM_H empfangen wird. Zudem wird zeitgleich
kontrolliert, dass die rechten Blinkleuchten während des Tests
inaktiv bleiben. Das SAM_H soll nach Erhalt des Signals innerhalb
von 30ms die hinten linken Blinkleuchten ansteuern. Gleichzeitig
soll das Signal noch innerhalb von 15ms an SAM_V, TSG_VL sowie AAG
weitergeleitet werden. Diese steuern daraufhin innerhalb von 30ms
die an ihnen angeschlossenen Blinkleuchten an. Zusätzlich soll vor
der Ansteuerung der Anhängerblinkleuchten noch das Vorhandensein
eines Anhängers überprüfen. In diesem Testfall ist ein Anhänger
in der simulierten Umgebung vorhanden.
```

**Listing 5.2:** Beispiel einer Testfallbeschreibung in der natürlichen Sprache

#### Zusammenfassung

Die natürliche Sprache ist neben der tabellenbasierten Darstellung die am meisten verwendete Testfallbeschreibung. Die Ausdrucksstärke zur Beschreibung der Testidee bietet kein anderes Testfallbeschreibungsmittel. Doch muss für eine Beschreibung wie bei der tabellenbasierten Darstellung die Sprache angepasst werden. Sprachelemente sind notwendig um funktionale Testfälle beschreiben zu können. Insgesamt eignet sich die natürliche Sprache ohne eine Adaption für die Testfallbeschreibung nicht.

## 5.4 Zusammenfassung der vorhandenen Beschreibungsmittel

Abbildung 5.22 (S. 171) zeigt eine Übersicht der Abdeckung der Sprachelemente für die Testfallbeschreibung und -implementierung sowie die Unterstützung der Paradigmen eines Testfalls von den untersuchten Mitteln.

Alle vorhandenen Beschreibungsmittel basieren auf sogenannten General Purpose (Modelling) Languages, d.h. als Grundlage wird ein Mittel verwendet, die für jeden Einsatzzweck geeignet ist und diese wird an die speziellen Bedürfnisse der Anwendungsdomäne angepasst.

Jeder der vorhandenen Mittel hat ihre Stärken und Schwächen. Die Stärke bei den UML Diagrammen ist die graphische Aufbereitung der Inhalte mit unterschiedlichen Schwerpunkten (z.B. Interaktion bei Sequenzdiagrammen, interne Zustände bei Zustandsdiagrammen und zeitliche Zustandsräume der Objekte bei Timingdiagrammen).

Bei den Spezifikationssprachen werden die ablaufbeschreibenden Eigenschaften für die Beschreibung der Testfälle verwendet. Werden spezielle Facetten wie zeitliche Aspekte durch die Spezifikationssprache zur Verfügung gestellt, so werden diese bei der Beschreibung der Testfälle berücksichtigt. Jedoch sind charakteristische Eigenschaften der speziellen Spezifikationssprachen für die Beschreibung der Testfälle irrelevant, wie z.B. die Modellierung von Nebenläufigkeiten und Deadlock Erkennung bei Petri-Netzen.
Bei den Spezifikationssprachen besitzen die endlichen Automaten durch ihren universellen Charakter auch eine universelle Einsatzmöglichkeit. Die Erweiterungen wie Statecharts oder Timed Automata werden bei der Modellierung und beim Test von reaktiven Systemen eingesetzt. Auch hier liegt der Schwerpunkt bei den Zuständen und Übergängen (Transitionen), d.h. der Abbildung der Testfälle auf Zustände und Ereignisse.

Bei der domänenspezifischen Sprache TTCN3 ist zu erkennen, dass der Schwerpunkt bei der Testfallimplementierung liegt, wohingegen die Sprachelemente für die Beschreibung gering ausfallen.

Der Schwerpunkt von XML ist die Datenhaltung für eine automatisierte Datenverarbeitung. Der durch zusätzliche Sprachelemente aufkommende Overhead ist bei der Automatisierung irrelevant, so dass sich XML basierte Ansätze für die maschinelle Verarbeitung eignen.
ATML, eine auf XML basierte Beschreibungssprache, ist mit dem Ziel erarbeitet worden, um Testfälle auf abstraktem Level zu beschreiben. Die Sprachelemente sind abstrakt gehalten und das Konzept bietet eine gute Grundlage für die Testfallbeschreibung. So ist zu erkennen, dass ATML die meisten Sprachelemente aus der Testfallbeschreibung unterstützt und eine gute Grundlage für die Testfallimplementierung bietet.
Zu bemängeln ist bei ATML, dass das Konzept auf XML aufbaut. Eine werkzeugunterstützte Eingabe ist zwingend notwendig und die Ausdrücke lassen sich schwer lesen. Weiterhin ist das Schema (Konzept für die Testfallbeschreibung) auf den Konzepten der XML aufgebaut, die ihren Ursprung in der Datenhaltung und automatisierten Datenverarbeitung haben.

Tabellen-basierte Beschreibungsmöglichkeiten bieten ebenfalls durch ihren universellen Charakter eine Möglichkeit Testfälle zu beschreiben. Hier ist der Schwerpunkt allerdings bei den Testdaten und weniger beim Ablauf. Zusätzliche Parser und die Möglichkeit jede Zelle einzeln definieren zu können ermöglicht eine individuelle Anpassung der tabellen-basierten Form für die Testfallbeschreibung.

Keins der vorhandenen Mittel bietet die Ausdrucksstärke, die bei der natürlichen Sprache vorhanden ist. Die Verständlichkeit wird bei den anderen Mitteln durch zusätzliche Kommentare in natürlicher Sprache ergänzt. Der Schwerpunkt der natürlichen Sprache liegt auch in der Übermittlung der Inhalte. Die fehlende Strukturierung ermöglicht keine maschinelle Verarbeitung.
Ansätze bei der Anforderungsdefinition (System-Spezifikation) sind die natürliche Sprache zu formalisieren (vgl. Abschnitt 5.3.6, S. 165). Der Unterschied der Testfallbeschreibung zu der Anforderungsdefinition ist der zu beschreibende Ablauf. Allerdings mündet eine Ablaufbeschreibung in natürlicher Sprache in Pseudocode (vgl. [Dij78] über die Verwendung natürlicher Sprachen für die Programmierung).

Hier muss ähnlich wie bei der Anforderungsdefinition (vgl. [Rup07], [Gud03]) neben der Sprache die Herleitung in Angriff genommen werden, d.h. die Frage danach wie man am Besten vollständige, verständliche Anforderungen definiert ist übertragbar auf die Erstellung einer Testfallbeschreibung. Unabhängig, ob diese in natürlicher Sprache stattfindet oder nicht.

Eine auf den Konzepten der ATML basierende, aber ohne die Unzulänglichkeit der Auszeichnungssprache XML, aber dafür mit dessen Stärken (Erweiterbarkeit, automatisierte Bearbeitung) und den Stärken der natürlichen Sprache (Verständlichkeit) für eine Testfallbeschreibung wäre somit ideal.

| | Bewertungsgruppe | Bew.kriterium | UML2TP | TPT | TTCN3 | ATML | hat. Sprache | Tabellen |
|---|---|---|---|---|---|---|---|---|
| Sprachelemente einer Testfallbeschreibung | | | | | | | | |
| | Testobjekt | | | | | | | |
| | | Angaben zum Testobjekt | + | - | + | + | - | - |
| | Testumgebung | | | | | | | |
| | | beteiligte Komponenten | + | - | + | + | - | - |
| | Testschritt | | | | | | | |
| | | Aktion | + | + | - | + | - | + |
| | | Ereignis | + | + | - | - | - | + |
| | | Reaktion | + | + | + | + | - | + |
| | | Zustand | + | + | - | - | - | + |
| | Soll-Reaktion | | | | | | | |
| | | Angabe von Toleranzen | - | - | + | + | - | - |
| | Vor- und Nachbedingungen | | | | | | | |
| | | Vorbedingung | - | - | - | + | - | + |
| | | Nachbedingung | - | - | - | + | - | + |
| | Testdaten | | | | | | | |
| | | Wertbeschreibung | - | - | - | + | - | + |
| | | Signalbeschreibung | - | + | - | + | - | - |
| | | Liste | - | - | - | + | - | - |
| | | kontinuierliche Wert/Signale | - | + | - | + | - | - |
| | Nachrichtenaustausch | | | | | | | |
| | | Senden von Nachrichten | + | - | + | + | - | - |
| | | Empfangen von Nachrichten | + | - | + | + | - | - |
| | zeitliche Aspekte | | | | | | | |
| | | früheste Zeitangabe | - | - | - | - | - | - |
| | | späteste Zeitangabe | - | - | - | - | - | - |
| | | Zeitangabe über Dauer | - | - | - | - | - | - |
| | | Periodenangabe | - | - | - | - | - | - |
| | | absolute Zeitangabe | - | - | - | - | - | - |
| | | relative Zeitangabe | - | - | - | - | - | - |
| | höhere Kontrollstrukturen | | | | | | | |
| | | Wiederholung | - | + | - | + | - | - |
| | | Iteration durch Liste | - | - | + | + | - | - |
| | | Warteanweisung | + | - | + | + | - | + |
| | Parallelitäten | | | | | | | |
| | | Beschr. parallelen Abläufen | + | + | - | + | - | - |
| | 10 | 25 | 10 | 8 | 8 | 17 | 0 | 8 |
| Sprachelemente einer Testfallimplementierung | | | | | | | | |
| | Testobjekt | | | | | | | |
| | | Schnittstellendefinition | - | - | + | + | - | - |
| | Kontrollstrukturen | | | | | | | |
| | | Sequenzen | - | - | + | + | - | - |
| | | Schleifen | - | - | + | + | - | - |
| | | Verzweigungen | - | - | + | + | | |
| | Datentypen / Typisierung | | | | | | | |
| | | einfache Datentypen | - | + | + | + | - | - |
| | | komplexe Datentypen | - | - | + | + | | |
| | | Typisierung | - | - | + | + | - | - |
| | Protokollierungs-Mechanismen | | | | | | | |
| | | Aufzeichnungen | - | - | + | - | - | - |
| | | Protokollierung | - | - | + | - | - | - |
| | Operatoren | | | | | | | |
| | | arithmetische Operatoren | - | + | + | + | - | - |
| | | relationale Operatoren | - | + | + | + | - | - |
| | | logische Operatoren | - | + | + | + | - | - |
| | Zeit-Operationen | | | | | | | |
| | | Start / Stop von Timern | + | + | + | - | - | - |
| | | Ablauf von Timern | + | + | + | - | - | - |
| | | Zeitmessung | + | - | + | - | | |
| | Stream Handling | | | | | | | |
| | | Operationen auf Streams | - | - | + | - | - | - |
| | Operatoren für Nebenläufikeit | | | | | | | |
| | | Definition von Nebenläufigkeit | - | - | + | - | - | - |
| | | Fork, Join, Sync, ... | - | - | + | - | - | - |
| | Testergebnis | | | | | | | |
| | | Verdict (Pass, fail, abort, . . .) | + | + | + | + | - | - |
| | 9 | 19 | 4 | 7 | 19 | 11 | 0 | 0 |
| Mechanismen zur Verständlichkeit | | | | | | | | |
| | Erstellung neuer Sprachelemente | | | | | | | |
| | | Erweiterung der Syntax | - | - | - | - | - | - |
| | Workflow zur Erhöhung der Verst. | | | | | | | |
| | | Vorgaben zur Bennenung | - | - | - | - | - | - |
| | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| Pradigmen des Testfalls | | | | | | | | |
| | Aufbau eines Testfalls | | | | | | | |
| | | Sequenzen | + | + | + | + | + | + |
| | | Automaten | + | + | - | + | - | - |
| | 1 | 2 | 2 | 2 | 1 | 2 | 1 | 1 |
| | 22 | 48 | 16 | 17 | 28 | 30 | 1 | 9 |

**Abbildung 5.22:** Detaillierte Betrachtung der Sprachelemente im Vergleich

# Kapitel 6

# Eine Testfall-Spezifikationssprache für die Beschreibung von Testfällen im automobilen Bereich

Bei der Analyse der Domäne „Testen im Automobilen Bereich“ (Kapitel 2, S. 13) und der Untersuchung der Testfallbeschreibung (Kapitel 3, S. 59) wurden die benötigten Sprachelemente für eine Testfallbeschreibung herausgearbeitet. Weiterhin ist bekannt, dass für die Signalbeschreibung eine separate Lösung gefunden werden muss um diese abstrahiert bei der Testfallbeschreibung verwenden zu können.

In diesem Kapitel wird zuerst die Grundlage für die Definition der Sprache gelegt. Erst danach wird die Testfall-Spezifikationssprache definiert. Anschliessend wird für eine visuelle Darstellung der Testfallbeschreibung eine Transformation in die UML2 Aktivitätsdiagramme vorgestellt. Zuletzt werden besondere Bereiche aus der Definition der Sprache betrachtet, die ausserhalb der Sprachdefinition liegen, aber einen Einfluss auf die Sprache haben. Für diese Bereiche werden Lösungsvorschläge vorgestellt.

## 6.1 Zur Beschreibung von Sprachen

### 6.1.1 Syntax

Eine Sprache ist eine Teilmenge aller Wörter, die durch hintereinanderschreiben von Elementen (Zeichen bzw. Symbole) aus einem Alphabet gebildet werden (vgl. [Sch01b]

und Anhang G). Um diese Teilmenge zu beschreiben werden im Folgenden zwei Mechanismen betrachtet: Grammatiken und Automaten.

Eine Grammatik besteht aus vier Elementen. Dem Alphabet, den Variablen, den Regeln und einer Startvariable. Eine Regel besteht aus einer Transformation einer gegebenen Form in eine andere. Dabei wird auf die linke Seite einer Implikation der Regel die ursprüngliche Form dargestellt und auf die rechte Seite die Form nach der Umwandlung. Angefangen von der Startvariable wird durch anwenden der Regeln und der Verwendung von Variablen an der linken bzw. rechten Seite der Implikation, Wörter erzeugt, die nur noch durch die Elemente des Alphabets bestehen.

Je nachdem wie die Regeln zusammengesetzt sind ergibt sich eine Einteilung der erzeugten Sprache.

Unterliegen die Regeln z.B. keinerlei Einschränkung, so wird die erzeugte Sprache als Typ 0 bezeichnet. Enthält die Grammatik Regeln, bei denen die rechte Seite gleich oder größer als die linke Seite ist, so ist sie vom Typ 1. Eine mathematisch korrekte Definition der Grammatik und der Elemente (Alphabet, Symbole, Wörter, Sprachen) ist im Anhang G, 281 vorhanden.

In der Typ 2 Grammatik bzw. bei den kontextfreien Sprachen sind die Programmiersprachen einzuordnen. In diesem Umfeld wird auch die Testfall-Spezifikationssprache definiert.

Die zweite Möglichkeit Sprachen zu definieren sind Automaten. Allerdings werden bei den Automaten die Wörter der Sprachen nicht erzeugt, d.h. bei einem gegebenen Wort erkennt der Automat, ob das Wort zur Sprache gehört oder nicht.

Typ 3 Sprachen werden mit endlichen Automaten erkannt. Typ 2 Sprachen werden von nichtdeterministischen Kellerautomaten erkannt. Lineare nichtdeterministische Turingmaschinen erkennen Typ 1 und allgemeine Typ 0 Sprachen.

Abbildung 6.1 zeigt die Zuordnung der Automaten zu den Grammatiktypen.

### 6.1.2 Semantik

Die Semantik kann entweder informell oder formal beschrieben sein und legt die Bedeutung der Zeichenfolge fest, die durch die Syntax definiert wurde.

Eine informelle Beschreibung wird z.B. bei der Definition von UML verwendet, wo jedes Diagramm bzw. Element in natürlicher Sprache (Englisch) definiert ist.

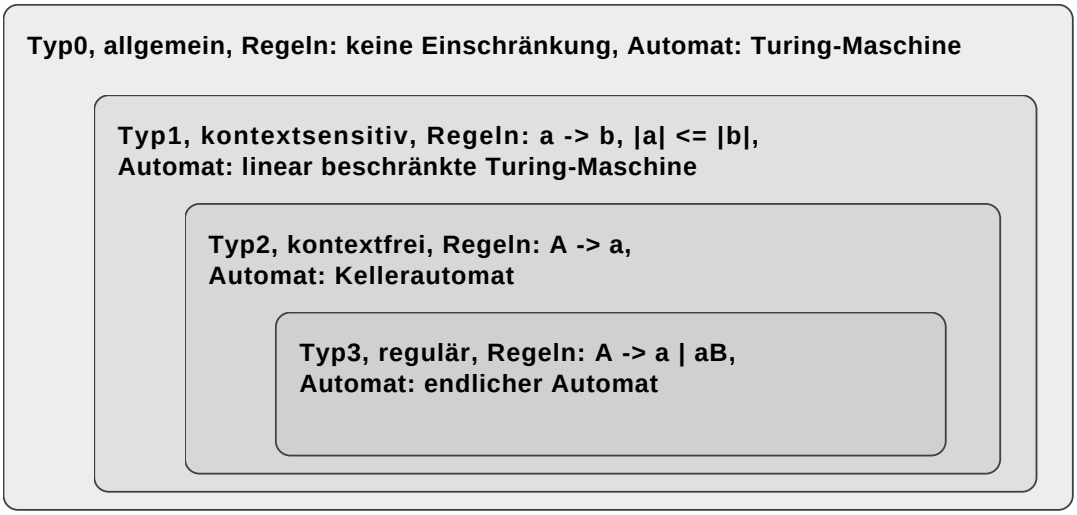



**Abbildung 6.1:** Die Chomsky Hierarchie für Grammatiken

Die formale Semantik zu beschreiben kann auf drei Arten erfolgen:

- Operationale Semantik

  Mit der operationalen Semantik wird die Programmausführung als ein System von Zustandsübergängen beschrieben. Durch die Definition erfolgt die abstrakte Beschreibung wie ein Programm auszuführen ist.
- Denotationelle Semantik

  Mit der denotationellen Semantik wird das Ergebniss der Programmausführung beschrieben. Dadurch werden nur die Effekte von Berechnungen, nicht der Weg, definiert. Die Modellierung erfolgt über mathematische Modelle.
- Axiomatische Semantik

  Mit der axiomatischen Semantik werden Prädikate beschrieben, die an bestimmten Punkten im Programm gelten sollen. Durch die Definition von Zusicherungen (Constraints) wird ebenfalls das Ergebnis von bestimmten Programmabschnitten spezifiziert. Dabei werden bzw. können nur (relevante) Teilaspekte wiedergegeben werden.

Eine formale Semantik mit einer dieser drei Arten zu definieren ist für eine ausführbare Sprache sinnvoll.

### 6.1.3 Zusammenfassung

Für die Definition der Testfall-Spezifikationssprache wird eine Möglichkeit für die Definition der Syntax und der Grammatik benötigt.

Die Syntax wird erzeugend definiert, d.h. durch eine Grammatik. Hier wird das EBNF verwendet. Anhang E (S. 273) gibt eine Einführung zu EBNF. Die Semantik erfolgt in natürlicher Sprache, da die Testfall-Spezifikationssprache keine ausführbare Semantik besitzen soll.

## 6.2 Entscheidungen für den Entwurf der Testfall-Spezifikationssprache

Die Sprachelemente basieren auf dem Inhalt und Umfang der Testfälle aus Kapitel 3, S. 59, aus dem Kapitel für die Anforderungen 4, S. 97 und dem Stand der Technik 5, S. 123. Abbildung 6.2 stellt den Einfluss der Kapitel auf die Testfall-Spezifikationssprache graphisch dar.

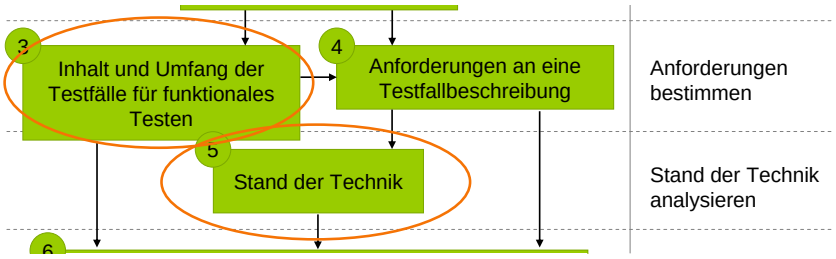



**Abbildung 6.2:** Einfluss der vorherigen Kapitel auf den Entwurf der Testfall-Spezifikationssprache

Inspiriert wird das Konzept von dem Schema für Testfälle in ATML (s. Abbildung 5.10), aus Kapitel Stand der Technik, S. 123. Konzeptionelle Entscheidungen bei der Testfallbeschreibung in dem ATML-Schema werden nicht berücksichtigt, falls sie Aufgrund der Eigenschaften (bzw. Einschränkungen) von XML entworfen wurden. Dieses Schema wird erweitert um die Erkenntnisse aus Kapitel Inhalt und Umfang der Testfälle für den funktionalen Steuergerätetest, S. 59 (Herleitung der Sprachelemente) und den Anforderungen aus dem Kapitel Anforderungen an die Testfallbeschreibung, S. 97 (Definition weiterer Sprachelemente), so dass insgesamt eine überarbeitete Grundstruktur vorliegt.

### Graphisch vs. textuell

Je komplizierter eine Modellierung ist, desto unübersichtlicher wird die graphische Darstellung. Kommen noch Aspekte wie Feinmodellierung hinzu, verliert die graphische Darstellung ihre Übersichtlichkeit und Verständlichkeit. Auf der anderen Seite

können einfache Modelle / Beschreibungen in textueller Form länger für die Übermittlung des Inhalts brauchen, als die äquivalente graphische Präsentation [Har01a].

So wird auf Basis einer textuellen Notation gearbeitet. Diese wird durch eine graphische Repräsentation ergänzt. Somit lassen sich die beiden Darstellungen wechselseitig einsetzen.

Beide Fälle helfen jedoch, bei zu großen Modellen, nicht die Lesbarkeit bzw. Verständlichkeit eines Modells zu erhöhen.

## Formalitätsgrad

Ob eine Sprache formal, semi-formal oder informell ist hat einen Einfluss auf die Eindeutigkeit: Mit der Formalität steigt die Eindeutigkeit der Beschreibung. Zugleich steigt der Aufwand, sich Gedanken über die Formulierung zu machen, da der Verfasser in der Sprache denken muss bzw. seine Gedanken niederschreiben muss.

Würde sich der Aufwand beim Lesen und damit Nachvollziehen des Inhalts in Grenzen halten, so wäre eine formale Beschreibungssprache fast geeignet. Jedoch ist die Nachvollziehbarkeit einer formalen Beschreibung mit dem Aufwand verbunden sich in der Sprache auszukennen. Für Benutzer, dessen Aufgabengebiet die Verwendung von so einer Sprache umfasst, sicherlich keine Herausforderung. Für Benutzer, wie einem Reviewer, ist dies allerdings nicht zumutbar.

So wird die Syntax formal gehalten, aber die Semantik in natürlicher Sprache definiert. Soweit notwendig werden Prosabeschreibungen zugelassen, um die Beschreibungsmächtigkeit der Sprache nicht einzugrenzen.

## Signalbeschreibung

Die erwarteten Sollergebnisse können durch die Eingabe- und Ausgabewerte definiert werden. Allerdings ist eine Modellierung der Signalwerte nach Einsatzgebiet auf unterschiedlichen Abstraktionslevel durchzuführen. Beispiel: Den Werteverlauf der Drosselklappe als eine Sinus Kurve zu modellieren ist für den Kenner verständlich. Allerdings ist die eigentliche Information, wieso diese nach dieser Kurve modelliert wurde nur implizit vorhanden (durch die Kurve). Eine explizite und damit abstrakte Modellierung wäre in Form von 'Die Drosselklappe wird auf und zu gemacht'.

So erfolgt die Signalbeschreibung in erster Linie durch eine Prosabeschreibung, die durch eine detaillierte formale Signalbeschreibung ergänzt werden soll.

### Werkzeugunterstützte Eingabe

Der Text muss strukturiert sein. Schlüsselwörter, Leerzeichen, Absätze und weitere Elemente zur visuellen Präsentation eines Textes erleichtern dem Leser die Nachvollziehbarkeit des Inhalts.

So wird die Sprache mit dem Ziel entworfen, sowohl werkzeugunterstützt als auch manuell die Beschreibung eingeben zu können. Das Ziel ist hier, so wenig wie möglich Abhängigkeiten zum Werkzeug zu erzeugen, so dass ausgedruckte Testfälle immer noch bearbeitet werden können.

## 6.3 Definition der Sprachelemente

Die Syntax der Testfall-Spezifikationssprache ist in folgende Unterteilung gegliedert:

- Kernelemente der Sprache, bestehend aus den Teilen:
  - Elemente eines Testfalls
  - Instanzen eines Testschritts
  - Kontrollstrukturen der Sprache
- Elemente zur Strukturierung
- Elemente zur Reaktionsauswertung
- Elemente für die zeitlichen Aspekte
- Hilfselemente der Sprache
- Elemente zur Anbindung an die Testfallimplementierung bestehend aus den Teilen:
  - Definition der Aktionen, Reaktionen, Ereignisse und Zustände
  - Atomare Aktionen, Reaktionen, Ereignisse und Zustände
  - Sprachelemente zur Funktionsaufrufen und Signalsetzungen

Im Folgenden wird die Syntax und Semantik der Sprache nach dieser Unterteilung dargestellt.

Jede Definition besitzt eine Syntax und eine Beschreibung. Die Beschreibung definiert die Semantik der Syntax. Hinweise und Anmerkungen erfolgen innerhalb der Beschreibung. Zusammenhängende Sprachelemente werden unter 'siehe auch' aufgelistet und sind zum Verständnis der Syntax und Semantik notwendig.

## 6.4 Syntax und Semantik der Testfall-Spezifikationssprache

| Name | Testfallbeschreibung |
|---|---|
| Beschreibung | Die Testfallbeschreibung definiert ein Grundgerüst für eine Testfallbeschreibung.<br>Die Testfallbeschreibung besteht aus nur optionalen Teilen. Die erste Hälfte definiert optionale *Aktionen* (→), *Ereignisse* (→), *Reaktionen* (→) und *Zustände* (→). Anschliessend können diese definierten Elemente in einem *Testfall* (→) verwendet werden. Im Gegensatz zur ersten und letzten Hälfte können mehrere Testfälle definiert werden. In der letzten Hälfte können die in der ersten Hälfte definierten Aktionen, Ereignisse, Reaktionen und Zustände durch atomare Anweisungen spezifiziert werden, um eine Verknüpfung zu einem Testskript zu ermöglichen. |
| Syntax | {*Referenz.*}<br>[**Aktionen Anfang**<br>{**Aktion** *Aktion.*}<br>**Ende Aktionen**.]<br>[**Ereignisse Anfang**<br>{**Ereignis** *Ereignis.*}<br>**Ende Ereignisse**.]<br>[**Reaktionen Anfang**<br>{**Reaktion** *Reaktion.*}<br>**Ende Reaktionen**.]<br>[**Zustände Anfang**<br>{**Zustand** *Zustand.*}<br>**Ende Zustände**.]<br>{*Testfall.*}<br>[**Definition Aktionen Anfang**<br>{*Aktionsdefinition.*}<br>**Ende Definition Aktionen**.]<br>[**Definition Ereignisse Anfang**<br>{*Ereignisdefinition.*}<br>**Ende Definition Ereignis**.]<br>[**Definition Reaktionen Anfang**<br>{*Reaktionsdefinition.*}<br>**Ende Definition Reaktionen**.]<br>[**Definition Zustände Anfang**<br>{*Zustanddefinition.*}<br>**Ende Definition Zustände**.] |
| Siehe auch | |

### 6.4.1 Kernelemente der Sprache

#### Elemente eines Testfalls

| Name | Testfall |
|---|---|
| Beschreibung | Ein Testfall beschreibt eine Menge von *Testschritten* (→) mit optionalen *Vorbedingungen* (→) und *Nachbedingungen* (→). Ein Testfall muss eine eindeutige ID besitzen und kann optional mit einem *Parameter* (→) und einer *Priorität* (→) angegeben werden. |
| Syntax | **Testfall** TesfallID [*Parameter*] [*Priorität*] **Anfang**<br>[**Beschreibung :** Beschreibung.]<br>[*Vorbedingung*.]<br>{*Testschritt*.}<br>[*Nachbedingung*.]<br>**Ende Testfall** |
| Siehe auch | *Testschritt* |

| Name | Vorbedingung |
|---|---|
| Beschreibung | Eine Vorbedingung ist eine Menge von *Zuständen* (→), die vor *Testschritten* (→) in einem *Testfall* (→) gelten müssen. |
| Syntax | **Vorbedingung Anfang**<br>{**Zustand** *Zustandsname*.}<br>**Ende Vorbedingung** |
| Siehe auch | Nachbedingung |

| Name | Testschritt |
|---|---|
| Beschreibung | Ein Testschritt ist eine Zusammenfassung der möglichen Schritte und wird explizit in keinem *Testfall* (→) verwendet. Ein Testschritt kann ein *Testfallschritt* (→), ein *Aktionsschritt* (→), ein *Nachrichtenschritt* (→), ein *Ereignisschritt* (→) oder eine *Kontrollstruktur* (→) sein.<br>Hinweis: Ein Element Testschritt taucht somit in keiner Testfallbeschreibung auf. |
| Syntax | *Testfallschritt* \| *Aktionsschritt* \| *Nachrichtenschritt* \| *Ereignisschritt* \| *Kontrollstruktur* |
| Siehe auch | *Testfallschritt*, *Aktionsschritt*, *Nachrichtenschritt*, *Ereignisschritt*, *Kontrollstuktur* |

| Name | Nachbedingung |
|---|---|
| Beschreibung | Eine Nachbedingung ist eine Menge von *Zuständen* (→), die nach *Testschritten* (→) in einem *Testfall* (→) gelten müssen. |
| Syntax | **Nachbedingung Anfang**<br>{**Zustand** *Zustandsname*.}<br>**Ende Nachbedingung** |
| Siehe auch | Vorbedingung |

| Name | Zustand |
|---|---|
| Beschreibung | Ein Zustand ist Vorgabe, die erfüllt werden muss. |
| Syntax | *ZustandID Zustandsname* |
| Siehe auch | *Vorbedingung*, *Nachbedingung* |

### Instanzen eines Testschritts

| Name | Aktionsschritt |
|---|---|
| Beschreibung | Ein Aktionsschritt definiert eine *Aktion* (→) oder eine *Aktionsgruppe* (→) gefolgt von der erwarteten *Reaktion* (→). Die *Aktion* kann optionale *Parameter* (→) enthalten und mit zeitlichen Verhalten (*Zeitverhalten* (→)) definiert werden. |
| Syntax | (**Aktion** *Aktion* {*Parameter*} {*Zeitverhalten*} \| Aktionsgruppe) [**führt zur** ( [**parallelen**] *Reaktionsschritt* \| *Reaktionsgruppe* \| *Reaktionsauswertung*)] |
| Siehe auch | |

| Name | Reaktionsschritt |
|---|---|
| Beschreibung | Ein Reaktionsschritt definiert die erwartete *Reaktion* (→) auf eine *Aktion* (→). Optional können *Parameter* (→) definiert werden und das *Zeitverhalten* (→) bestimmt werden. |
| Syntax | **Reaktion** *Reaktion* [*Parameter*] [*Zeitverhalten*] |
| Siehe auch | |

| Name | Nachrichtenschritt |
|---|---|
| Beschreibung | Ein Nachrichtenschritt definiert ein Austausch einer *Nachricht* (→) zwischen zwei *Steuergeräten* (→) mit Wertangabe und *Zeitverhalten* (→). |
| Syntax | (*Steuergerät* **sendet Nachricht** *Nachricht* **mit Wert** Wert **an** *Steuergerät* [*Zeitverhalten*]. \|<br>*Steuergerät* **empfängt Nachricht** *Nachricht* **mit Wert** Wert **von** *Steuergerät* [*Zeitverhalten*].) |
| Siehe auch | *Steuergerät*, *Nachricht* |

| Name | Ereignisschritt |
|---|---|
| Beschreibung | Ein Ereignisschritt definiert ein erwartetes *Ereignis* (→). Optional kann dem Ereignis *Parameter* (→) angegeben werden. |
| Syntax | **Ereignis** *Ereignis* [*Parameter*] |
| Siehe auch | *Ereignis* |

| Name | Aktion |
|---|---|
| Beschreibung | Ein Testschritt mit dem Ziel Eingaben beim Testobjekt zu definieren. |
| Syntax | *AktionID Aktionsname* |
| Siehe auch | *Aktionsschritt*, *Gruppe_von_Aktionen* |

| Name | Reaktion |
|---|---|
| Beschreibung | Ein Testschritt mit dem Ziel die Ausgaben vom Testobjekt zu definieren. |
| Syntax | *ReaktionID Reaktionsname* |
| Siehe auch | *Reaktionsschritt*, *Gruppe_von_Reaktionen* |

| Name | Ereignis |
|---|---|
| Beschreibung | Ein Testschritt, dass keine zeitliche Ausdehnung besitzt und zeitlich nicht vorhersagbar ist. |
| Syntax | *EreignisID Ereginisname* |
| Siehe auch | *Ereignisschritt* |

### Kontrollstrukturen der Sprache

| Name | Kontrollstruktur |
|---|---|
| Beschreibung | Eine Kontrollstruktur kann eine Warteanweisung, eine Wiederholung oder eine Iteration sein.<br>Hinweis: Ein Element Kontrollstruktur taucht somit in keiner Testfallbeschreibung auf. |
| Syntax | *Warteanweisung* \| *Wiederholung* \| *Iteration* |
| Siehe auch | *Warteanweisung*, *Wiederholung*, *Iteration* |

| Name | Warteanweisung |
|---|---|
| Beschreibung | Durch eine Warteanweisung wird eine zeitliche Verzögerung definiert oder das Warten auf ein Ereignis (→) vorgegeben. |
| Syntax | **Warte** (*Zeitangabe* \| **bis** *Ereignis*) |
| Siehe auch | *Ereignis* |

| Name | Wiederholung |
|---|---|
| Beschreibung | Eine Wiederholung definiert das n-malige Ausführen eines oder einer Menge von *Testschritten* (→). Alternativ kann Teil der *Testschritte* (→) solange ausgeführt werden, bis ein erwarteter *Ereignisschritt* (→) eintritt. |
| Syntax | **Wiederhole** ( **mit Iterator** *ID Zahl* **mal** \| **bis** *Ereignisschritt* ) **Anfang**<br>{*Testschritt*}.<br>**Ende Wiederhole** |
| Siehe auch | |

| Name | Iteration |
|---|---|
| Beschreibung | Ein oder mehrere *Testschritte* (→) werden mit Werten aus einer *Liste* (→) iterriert. Der Iterator ID kann als *Parameterangabe* (→) bei den Testschritten verwendet werden. |
| Syntax | **Iteriere mit Iterator** *ID* durch *Liste* **Anfang**<br>{*Testschritt*}.<br>**Ende Iteriere** |
| Siehe auch | |

### 6.4.2 Elemente zur Strukturierung

| Name | Testfallschritt |
|---|---|
| Beschreibung | Ein Testfallschritt besteht aus einem Aufruf eines *Testfalls* (→) mit optionaler *Parameter* (→) Angabe. |
| Syntax | **Testfallaufruf** *TestfallID* [*Parameter*] |
| Siehe auch | *Testfall* |

| Name | Aktionsgruppe |
|---|---|
| Beschreibung | Eine Aktionsgruppe ist eine Menge von *Testschritten* (→). Sie können optional für eine parallele Durchführung definiert werden und im zeitlichen Umfang optional spezifiziert werden. |
| Syntax | **Gruppe von Aktionen** [*Parallel*] [*Zeitverhalten*] **Anfang**<br>{*Testschritt.*}<br>**Ende Gruppe von Aktionen** |
| Siehe auch | *Aktion*, *Aktionsschritt* |

| Name | Reaktionsgruppe |
|---|---|
| Beschreibung | Eine Reaktionsgruppe ist eine Menge von *Reaktionsschritten* (→) oder *Kontrollstukturen* (→). Sie können optional für eine parallele Durchführung definiert werden und im zeitlichen Umfang optional spezifiziert werden. |
| Syntax | **Gruppe von Reaktionen** [*Parallel*] [*Zeitverhalten*] **Anfang**<br>{*Reaktionsschritt.* \| *Reaktionsgruppe.* \| *Kontrollstruktur.*}<br>**Ende Gruppe von Reaktionen** |
| Siehe auch | *Reaktion*, *Reaktionsschritt* |

| Name | Referenz |
|---|---|
| Beschreibung | Dient zum Verweis auf andere *Testfallbeschreibungen* (→). Ähnlich der Referenz Funktion von Programmiersprachen können mit diesem Element vorhandene Definitionen von *Aktionen* (→), *Reaktionen* (→), *Ereignissen* (→), *Zuständen* (→) und *Testfällen* (→) verwiesen werden, so dass sie in dieser *Testfallbeschreibung* (→) ebenfalls verwendet werden können. |
| Syntax | **Referenz** *TestfallbeschreibungID* |
| Siehe auch | |

### 6.4.3 Elemente zur Reaktionsauswertung

| Name | Reaktionsauswertung |
|---|---|
| Beschreibung | Eine Reaktionsauswertung beinhaltet eine Menge von *Reaktionsauswertungsschritten* (→). |
| Syntax | **Reaktionsauswertung Anfang**<br>{*Reaktionsauswertungsschritt.*}<br>**Ende Reaktionsauswertung** |
| Siehe auch | *Reaktionsauswertungsschritt* |

| Name | Reaktionsauswertungsschritt |
|---|---|
| Beschreibung | Ein Reaktionsauswertungsschritt wird über eine Wenn-Dann Klausel definiert: Wenn ein *Reaktionsschritt* (→) erfolgt oder ein Wert (*LWert* (→)) einem erwarteten Wert (*RWert* (→)) entspricht, dann soll entweder ein *Verdict* (→) gesetzt werden oder die *Priorität* (→). Alternativ kann ein Testfallaufruf erfolgen. Zusätzlich kann durch ein oder mehrere „und“ Verknüpfungen weitere Verdicts bzw. Prioritäten gesetzt werden.<br>*Hinweis*: Mehrere Verdicts zu setzen oder mehrere Prioritäten festzulegen ergibt keinen Sinn, ist syntaktisch aber ermöglicht. Es empfiehlt sich immer ein Verdict bzw. eine Priorität festzulegen, damit die Eindeutigkeit gewährleistet ist. |
| Syntax | **Wenn** (*Reaktionsschritt* \| *LWert* **=** *RWert*) **dann** (**Verdict:** *Verdict* \| **Priorität:** *Priorität* \| *Testfallschritt*) {**und** (**Verdict:** *Verdict* \| **Priorität:** *Priorität*)} |
| Siehe auch | *Verdict*, *Priorität* |

| Name | Verdict |
|---|---|
| Beschreibung | Ein Verdict kann mit pass oder fail definiert werden. |
| Syntax | **pass** \| **fail** |
| Siehe auch | |

### 6.4.4 Elemente für die zeitlichen Aspekte

| Name | Zeitverhalten |
|---|---|
| Beschreibung | Mit dem Zeitverhalten werden *Zeiten* (→) definiert. Die Zeiten können mit „und“ verknüpft werden. Alernativ zu den Zeitangaben kann das Zeitverhalten zu einem Ereignis in Bezug gesetzt werden. |
| Syntax | (**mit Zeitverhalten** *Zeitaspekt* {**und** *Zeitaspekt*} \| **bis** *Ereignisschritt*) |
| Siehe auch | |

| Name | Zeitaspekt |
|---|---|
| Beschreibung | Mit dem Zeitaspekt werden zeitliche Aspekte definiert. Hier sind *frühestens* und *spätestens* als relativer Bezug, sowie die Angabe über die *Dauer* und die zyklische Wiederholung möglich. Durch Kombination der Zeiten mittels „und“ ist die Definition aller möglicher zeitlichen Aspekte möglich.<br>*Hinweis*: Die mehrfache Verwendung eines zeitlichen Aspekts ergibt keinen Sinn, ist jedoch syntaktisch möglich. Die Verknüpfung mittels „und“ ist nur dann sinnvoll, wenn unterschiedliche zeitliche Aspekte in Verbindung gesetzt werden sollen. |
| Syntax | **frühestens** *Zeitangabe* \| **spätestens** *Zeitangabe* \| **dauer** *Zeitangabe* \| **Zykluszeit** *Zeitangabe* |
| Siehe auch | |

| Name | Zeitangabe |
|---|---|
| Beschreibung | Die Zeitangabe ist für die Definition des zeitlichen Verhaltens (*Zeitaspekt* (→)) notwendig. |
| Syntax | Zahl *Zeiteinheit* |
| Siehe auch | *Zeiteinheit*, *Warte* |

| Name | Zeiteinheit |
|---|---|
| Beschreibung | Definiert die möglichen Einheiten einer zeitlichen Angabe.<br>ns steht für Nanosekunden.<br>us steht für Mikrosekunden.<br>ms steht für Millisekunden.<br>s steht für Sekunden.<br>m steht für Minuten.<br>h steht für Stunden.<br>d steht für Tage. |
| Syntax | **ns** \| **us** \| **ms** \| **s** \| **min** \| **h** \| **d** |
| Siehe auch | *Zeitangabe* |

### 6.4.5 Hilfselemente der Sprache

| Name | Parameter |
|---|---|
| Beschreibung | Parameter definieren einen Parameterwert oder eine *Liste* (→) von Parameterwerten. Die Verwendung erfolgt nur dort, wo (optionale) Parameterangaben möglich sind. |
| Syntax | ( **mit Parameter** *ParameterID* \| **mit Parameterliste** *Liste*) |
| Siehe auch | |

| Name | Priorität |
|---|---|
| Beschreibung | Die Priorität definiert die Priorität mittels der *Prioritätsangabe* (→). Die Verwendung erfolgt nur dort, wo (optionale) Prioritätsangaben möglich sind. |
| Syntax | **mit Priorität** *Prioritätsangabe* |
| Siehe auch | *Prioritätsangabe* |

| Name | Liste |
|---|---|
| Beschreibung | Die Liste definiert mindestens ein *Listen Element* (→). Optional können mehrere *Listen Elemente* angegeben werden. |
| Syntax | *ListenElement* {**,** *ListenElement*} |
| Siehe auch | *ListenElement* |

| Name | ListenElement |
|---|---|
| Beschreibung | Ein ListenElement beinhaltet einen eindeutigen Namen *NameID* (→). |
| Syntax | *NameID* |
| Siehe auch | |

| Name | Steuergerät |
|---|---|
| Beschreibung | Das Element Steuergerät definiert einen *Steuergerätenamen* (→). |
| Syntax | *Steuergerätename* |
| Siehe auch | |

| Name | Steuergerätname, NameID, LWert, RWert, TestfallbeschreibungID, Signalname, Signalverlauf |
|---|---|
| Beschreibung | Die Elemente Steuergerätename, NameID, LWert, RWert und TestfallbeschreibungID bestehen aus einem String. Des Weiteren werden Signalname und Signalverlauf ebenfalls über einen String definiert. |
| Syntax | STRING |
| Siehe auch | |

| Name | Zahl, Prioritätsangabe |
|---|---|
| Beschreibung | Die Elemente Zahl und Prioritätsangabe bestehen aus einem Integer.<br>Anmerkung: INTEGER ist in EBNF definiert als<br>INTEGER = DIGIT { DIGIT }, wobei DIGIT durch die Ziffern definiert ist.<br>FLOAT ist definiert als<br>FLOAT = INTEGER "." DIGIT { DIGIT } |
| Syntax | [-] ( INTEGER \| FLOAT ) |
| Siehe auch | |

## 6.4.6 Anbindung an die Testfallimplementierung

### Definition der Aktionen, Reaktionen, Ereignisse und Zustände

Um eine (halbautomatisierte) Erstellung von Testfallimplementierungen zu ermöglichen ist eine Definition der vier Grundelemente Aktionen, Reaktionen, Eregnisse und Zustände notwendig.

| Name | Aktionsdefinition |
|---|---|
| Beschreibung | Eine Aktionsdefinition besteht aus einer oder mehreren Definitionen von *Aktionen* (→). Dabei kann eine Aktion durch *atomare Aktionen* (→) definiert werden oder die Definition erfolgt durch eine vorhandene Aktion, wobei in diesem Fall *Parameter* (→) und/oder das *Zeitverhalten* (→) definiert werden können.<br>*Hinweis*: Die Definition einer Aktion über eine Aktion (, d.h. Rekursion) ist dann sinnvoll, wenn Parameter und/oder Zeitverhalten bestimmter Aktionen wohldefinierte Werte besitzen sollen. |
| Syntax | **Def Aktion** *Aktion* [*Parameter*]<br>**:=** (*Aktion* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareAktionen.*}<br>**Ende Def Aktion**) |
| Siehe auch | *Atomare Aktion* |

| Name | Reaktionsdefinition |
|---|---|
| Beschreibung | Eine Reaktionsdefinition besteht aus einer oder mehreren Definitionen von *Reaktionen* (→). Dabei kann eine Reaktion durch *atomare Reaktionen* (→) definiert werden oder die Definition erfolgt durch eine vorhandene Reaktion, wobei in diesem Fall *Parameter* (→) und/oder das *Zeitverhalten* (→) fest definiert werden können. |
| Syntax | **Def Reaktion** *Reaktion* [*Parameter*]<br>(**:=** *Reaktion* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareReaktionen.*}<br>**Ende Def Reaktion**) |
| Siehe auch | *Atomare Reaktion* |

| Name | Ereignisdefinition |
|---|---|
| Beschreibung | Eine Ereignisdefinition besteht aus einer oder mehreren Definitionen von *Ereignissen* (→). Dabei kann eine Ereignis durch *atomare Ereignisse* (→) definiert werden oder die Definition erfolgt durch ein vorhandenes Ereignis, wobei in diesem Fall *Parameter* (→) und/oder das *Zeitverhalten* (→) fest definiert werden können.<br>*Hinweis*: Bei dem Zeitverhalten kann keine *dauer* verwendet werden, da per Definition ein Ereginis keine Dauer besitzt. |
| Syntax | **Def Ereignis** *Ereignis* [*Parameter*]<br>(**:=** *Ereignis* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareEreignis.*}<br>**Ende Def Ereignis**) |
| Siehe auch | *Atomares Ereignis*, *Zeitverhalten* |

| Name | Zustandsdefinition |
|---|---|
| Beschreibung | Eine Zustandsdefinition besteht aus einer oder mehreren Definitionen von *Zuständen* (→). Dabei kann ein Zustand durch *atomare Zustände* (→) definiert werden oder die Definition erfolgt durch einen vorhandenen Zustand, wobei in diesem Fall *Parameter* (→) und/oder das *Zeitverhalten* (→) fest definiert werden können. |
| Syntax | **Def Zustand** *Zustand* [*Parameter*]<br>(**:=** *Zustand* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareZustand.*}<br>**Ende Def Zustand**) |
| Siehe auch | *Atomarer Zustand* |

### Atomare Aktionen, Reaktionen, Ereignisse und Zustände

Die atomaren Sprachelemente zu den Aktionen, Reaktionen, Ereignissen und Zuständen bilden die Koppelebene zu der Testfallimplementierung ab.

| Name | Atomare Aktion |
|---|---|
| Beschreibung | Nicht mehr untergliederte Aktion. Kann nur mit Bibliotheksaufrufen und/oder Signalsetzungen definiert werden. |
| Syntax | *Bibliotheksaufruf* \| *Signalsetzung* |
| Siehe auch | |

| Name | Atomare Reaktion |
|---|---|
| Beschreibung | Nicht mehr untergliederte Reaktion. Kann nur mit Bibliotheksaufrufen und/oder Signalsetzungen definiert werden. |
| Syntax | *Bibliotheksaufruf* \| *Signalsetzung* |
| Siehe auch | |

| Name | Atomares Ereignis |
|---|---|
| Beschreibung | Nicht mehr untergliedertes Ereignis. Kann nur mit Bibliotheksaufrufen und/oder Signalsetzungen definiert werden. |
| Syntax | *Bibliotheksaufruf* \| *Signalsetzung* |
| Siehe auch | |

| Name | Atomarer Zustand |
|---|---|
| Beschreibung | Nicht mehr untergliederter Zustand. Kann nur mit Bibliotheksaufrufen und/oder Signalsetzungen definiert werden. |
| Syntax | *Bibliotheksaufruf* \| *Signalsetzung* |
| Siehe auch | |

#### Sprachelemente zur Funktionsaufrufen und Signalsetzungen

Die Funktionsaufrufe und Signalsetzungen werden wie folgt festgelegt. Im übernächsten Abschnitt (Abschnitt 6.6, S. 216) werden beide Definition für die Ankopplung an vorhandene Testskripte umdefiniert (Werkzeugabhängige Definition der Sprachelemente).

| Name | Bibliotheksaufruf |
|---|---|
| Beschreibung | Aufruf einer Funktion in einer Bibliothek, die bei der Testfallimplementierung zur Verfügung gestellt wird. |
| Syntax | **Funktionsaufruf:** *Bibliotheksfunktion* |
| Siehe auch | |

| Name | Signalsetzung |
|---|---|
| Beschreibung | In der Signalsetzung wird der Verlauf eines Signals definiert. |
| Syntax | **Signalsetzung:** *Signalname* **=** *Signalverlauf* |
| Siehe auch | |

Die Definition des Signalnamen und des Signalverlaufs erfolgt über einen String (siehe vorherige Abschnitte).

## 6.5 Abbildung der Sprache im UML2 Aktivitätsdiagramm

In diesem Abschnitt wird eine Transformation der Testfall-Spezifikationssprache in eine graphische Darstellung spezifiziert.

Eigenschaften, die eine graphische Grundlage für die Testfallbeschreibung bieten müssen, sind bei jedem Graphen vorhanden: Knoten (Aktionen) und Kanten (Übergänge).

Doch sind unterschiedliche Gewichtungen der Darstellungsformen ausschlaggebend für die Verwendung der UML2 Aktivitätsdiagramme gewesen.

So besitzen z.B. Petri-Netze token-orientiertes Verhalten, so dass das (token-basierte) Schalten der Übergänge ermöglicht wird. Mit dieser Eigenschaft versehen, werden Petri-Netze zur visuellen Darstellung von Deadlockeigenschaften (Livelockfreiheit) oder zur Untersuchung von Nebenläufigkeiten herangezogen. Bei der Beschreibung von Testfällen besteht keine Notwendigkeit zur Untersuchung dieser Eigenschaften und somit besteht auch kein Bedarf für Tokens.

Ein weiteres oft verwendetes Diagramm sind Zustandsdiagramme. Sie besitzen als Knoten Zustände und die Übergänge werden durch Ereignisse ausgelöst. Ereignisbasiertes und zustand-basiertes Verhalten lassen sich damit gut abbilden. Die Testfallbeschreibung könnte damit abgebildet werden, doch gibt es eine genauere Darstellungsform für die Testfallbeschreibung.

Bei der Testfallbeschreibung mit der Testfall-Spezifikationssprache stehen Aktionen im Vordergrund. Im UML2 Aktivitätsdiagramm stehen ebenfalls die Aktionen im Mittelpunkt. Um diesen herum kann der Kontrollfluss definiert und der Objektfluss angeordnet werden. Innerhalb der Diagramme sind rekursiv weitere Aktivitäsdiagramme möglich.

Im Folgenden werden zuerst zwei Formen für die Erweiterung der UML2 Diagramme vorgestellt und darauf folgend die Abbildung definiert.

### 6.5.1 Lightweight Extensions zur Erweiterung der UML

Für die Darstellung der Testfall-Spezifikationssprache in UML2 Aktivitätsdiagramme gibt es zwei Ansätze. Entweder wird die UML2 an der Metasprache durch neue Elemente erweitert oder man verwendet die von der UML2 angebotenen Möglichkeiten zur Erweiterung bzw. Anpassung der UML2 an die eigenen Bedürfnisse.

Der erste Weg hat den Vorteil, dass man neue Symbole/Piktogramme einführen kann, die die neudefinierten Elemente besser wiedergeben, als die Standardsymbole der UML. Das hat aber den Nachteil, dass vorhandene UML Werkzeuge mit dieser Art der Erweiterung nicht umgehen können. Man nennt diese Art der Erweiterung heavyweight extension.

Der zweite Weg ist der Gegensatz dazu: Die lightweight extension. Hier werden mittels Stereotypen, Tagged Values und Constraints die Elemente der UML an die eigenen Bedürfnisse angepasst. Zusammen formen sie ein sogenanntes „UML Profil“. Der Vorteil ist, dass man mit gängigen UML Werkzeugen die Diagramme erstellen und bearbeiten kann. Der Nachteil ist, dass die Symbole gleich aussehen und sich nur in der Bezeichnung unterscheiden.

Durch die Bearbeitung der angepassten Diagramme mit herkömmlichen UML Werkzeugen eignet sich der zweite Weg, d.h. eine Anpassung ohne das Metamodell zu verändern.

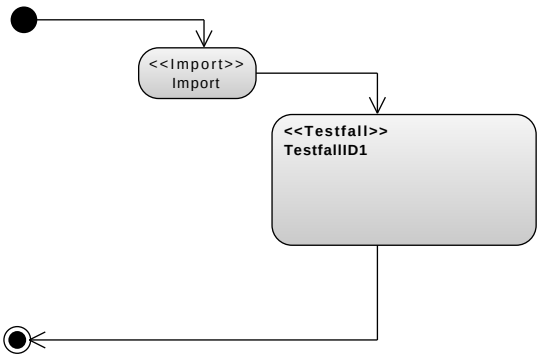
### 6.5.2 Das Grundgerüst der Sprache

Per Konvention gilt: In den Erläuterungen werden die Sprachelemente der Testfall-Spezifikationssprache *kursiv* geschrieben. Sprachelemente der UML2 Aktivitätsdiagramme werden von dem Präfix „UML2 AD Element“ - wobei Element auch entfallen kann - eingeleitet und werden *abgeschrägt* geschrieben.

Für die Abbildung wird die gleiche Unterteilung der Sprachelemente verwendet, die bei der Definition vorgenommen wurde. Einige Sprachelemente benötigen keine eigene Darstellung, so dass die Syntax im Folgenden nicht abgebildet wurde. Sie werden entweder durch Tagged Values dargestellt oder sind durch die Darstellung eines anderen - übergeordneten - Elements abgedeckt.

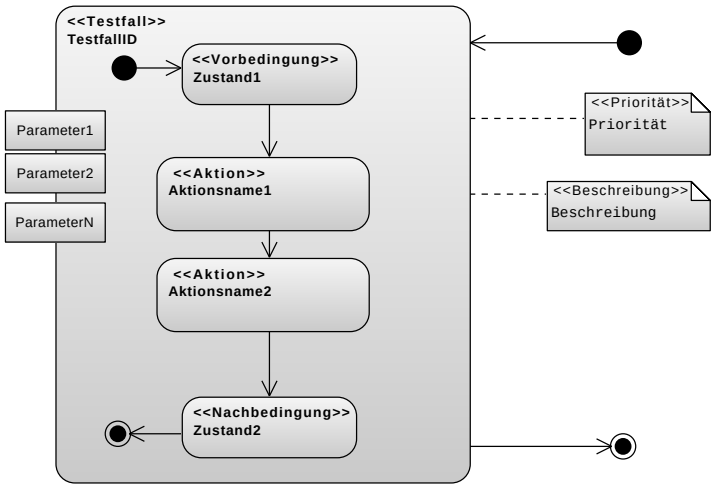
Die Definition der verwendeten Elemente der UML2 Aktivitätsdiagramme ist im Anhang D vorhanden.

Das UML2 Aktivitätsdiagramm wird im Folgenden mit „UML2 AD“ abgekürzt.

| Name | Testfallbeschreibung |
|---|---|
| Syntax der Testfall-Spezifikationssprache | {*Referenz.*}<br>**[Aktionen Anfang**<br>{**Aktion** *Aktion.*}<br>**Ende Aktionen**.]<br>**[Ereignisse Anfang**<br>{**Ereignis** *Ereignis.*}<br>**Ende Ereignisse**.]<br>**[Reaktionen Anfang**<br>{**Reaktion** *Reaktion.*}<br>**Ende Reaktionen**.]<br>**[Zustände Anfang**<br>{**Zustand** *Zustand.*}<br>**Ende Zustände**.]<br>{*Testfall.*}<br>**[Definition Aktionen Anfang**<br>{*Aktionsdefinition.*}<br>**Ende Definition Aktionen**.]<br>**[Definition Ereignisse Anfang**<br>{*Ereignisdefinition.*}<br>**Ende Definition Ereignis**.]<br>**[Definition Reaktionen Anfang**<br>{*Reaktionsdefinition.*}<br>**Ende Definition Reaktionen**.]<br>**[Definition Zustände Anfang**<br>{*Zustanddefinition.*}<br>**Ende Definition Zustände**.] |
| UML2 AD Darstellung |  |
| Erläuterung | Die Testfallbeschreibung ist das UML2 *Aktivitätsdiagramm* selbst. Das UML2 AD Element *Initialknoten* stellt den Anfang der Testfallbeschreibung dar. Das UML2 AD Element *Endknoten* stellt das Ende der Testfallbeschreibung dar. Zwischen Ihnen können die Darstellungen zu den Sprachelementen *Referenz*, *Testfall* und die Darstellungen für die Definitionen der Sprachelemente *Aktion*, *Reaktion*, *Ereignis* und *Zustand* vorkommen. |

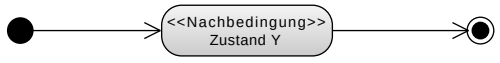
### 6.5.3 Kernelemente der Sprache

#### Elemente eines Testfalls

| Name | Testfall |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Testfall** TesfallID [*Parameter*] [*Priorität*] **Anfang**<br>[**Beschreibung :** Beschreibung.]<br>[*Vorbedingung*.]<br>{*Testschritt*.}<br>[*Nachbedingung*.]<br>**Ende Testfall** |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Testfall wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Testfall* dargestellt. Ein Testfall beginnt bei dem Initialknoten, der den Einstieg darstellt. Dem Initialknoten folgt optional eine Vorbedingung. Vorbedingung bzw. Nachbedingung werden durch eine Sequenz von Zuständen mit den Stereotypen *Vorbedingung* bzw. *Nachbedingung* dargestellt. Die Testschritte werden durch eine Sequenz von Testschritten gebildet, die durch das jeweilige Stereotyp für den Testschritt gekennzeichnet ist.<br>Hinweis: *Testschritt* ist abstrakt und wird in der Darstellung durch die konkrete Ausprägung repräsentiert (siehe *Testschritt*).<br>Das Ende eines Testfall stellt der Endknoten dar.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

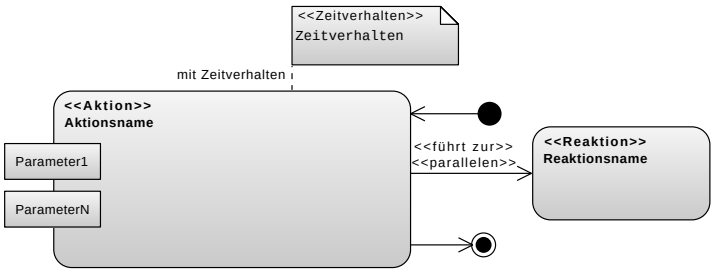
| Name | Vorbedingung |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Vorbedingung Anfang**<br>{**Zustand** *Zustandsname*.}<br>**Ende Vorbedingung** |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Vorbedingung wird durch eine Sequenz von *Zuständen* dargestellt. Jeder *Zustand* in der Vorbedingung wird durch das UML2 AD Element Aktion mit dem Stereotypen *Vorbedingung* dargestellt.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

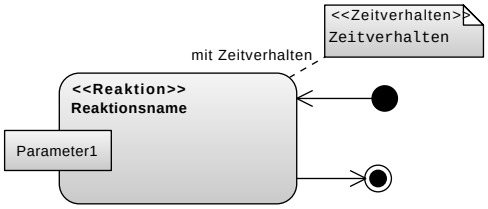
| Name | Testschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *Testfallschritt* \| *Aktionsschritt* \| *Nachrichtenschritt* \| *Ereignisschritt* \| *Kontrollstruktur* |
| UML2 AD Darstellung | keine UML 2 AD Darstellung, da abstrakt zu sehen. |
| Erläuterung | Ein Testschritt ist abstrakt definiert (siehe Syntax), das bedeutet es besitzt keine UML2 AD Darstellung, sondern die Darstellung erfolgt durch die konkrete Ausprägung.<br>Hinweis: Das Element Testschritt taucht niemals in einer Testfallbeschreibung auf. |

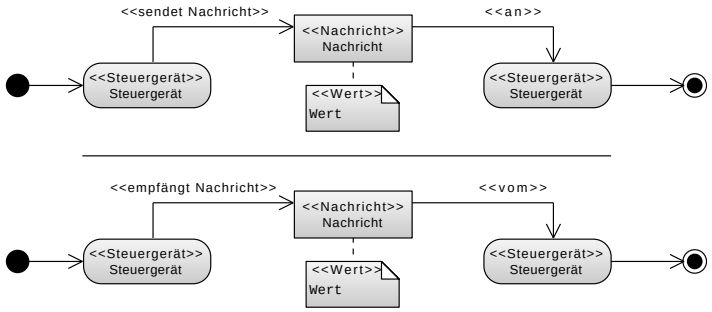
| Name | Nachbedingung |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Nachbedingung Anfang**<br>{**Zustand** *Zustandsname*.}<br>**Ende Nachbedingung** |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Nachbedingung wird durch eine Sequenz von *Zuständen* dargestellt. Jeder *Zustand* in der Nachbedingung wird durch das UML2 AD Element Aktion mit dem Stereotypen *Nachbedingung* dargestellt.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

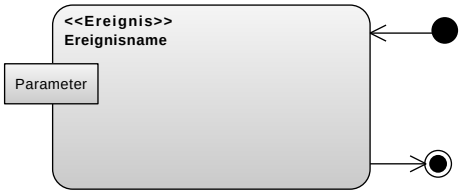
| Name | Zustand |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *ZustandID Zustandsname* |
| UML2 AD Darstellung | keine UML 2 AD Darstellung, da abstrakt zu sehen. |
| Erläuterung | Ist ein UML2 AD Aktionselement mit den Stereotypen *Vorbedingung* bzw. *Nachbedingung* (siehe *Vorbedingung*, *Nachbedingung*). |

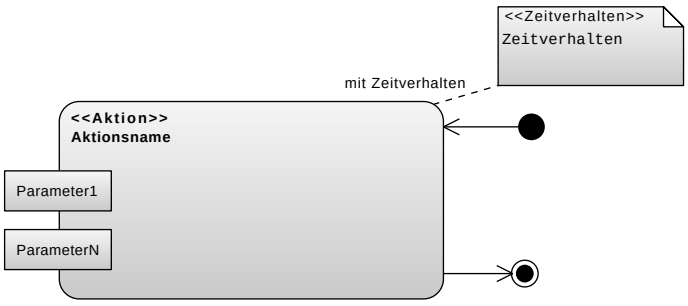
### Instanzen eines Testschritts

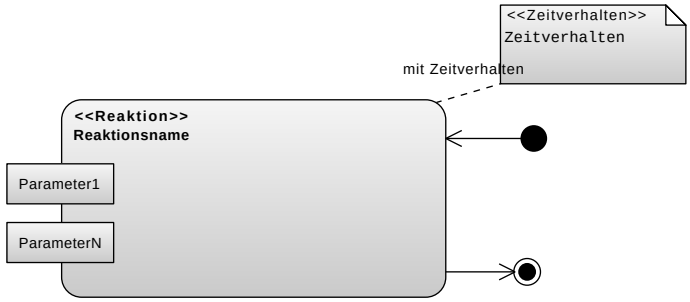
| Name | Aktionsschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | (**Aktion** *Aktion* {*Parameter*} {*Zeitverhalten*} \| Aktionsgruppe) [**führt zur** ( [**parallelen**] *Reaktionsschritt* \| *Reaktionsgruppe* \| *Reaktionsauswertung*)] |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Aktionsschritt, welches mit einer *Aktion* definiert wird, wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Aktion* dargestellt. Der Name der *Aktivität* beinhaltet den Aktionsnamen. Der *Aktivität* kann ein UML2 AD Element *Constraint* angefügt werden um das *Zeitverhalten* zu definieren (siehe *Zeitverhalten*). Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden. Das Schlüsselwort *führt zur* wird durch das UML2 AD Element *Kontrollfluss* mit dem Stereotypen *führt zur* dargestellt. Das optionale Schlüsselwort *parallelen* wird an das gleiche UML2 AD Element *Kontrollfluss* wie das *führt zur* (an zweiter Position) angeführt. Das Ende des UML2 AD Elements *Kontrollfluss* bildet die Verknüpfung zu einer UML2 AD Darstellung der Sprachelemente Reaktionsschritt, Reaktionsgruppe oder Reaktionsauswertung.<br>Ein Aktionsschritt, welches mit einer *Aktionsgruppe* definiert wird, wird durch ein UML2 AD Element *structured Element* dargestellt. Näheres unter der Darstellungsdefinition für *Aktionsgruppe*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Reaktionsschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Reaktion** *Reaktion* [*Parameter*] [*Zeitverhalten*] |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Reaktionsschritt wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Reaktion* dargestellt. Der Name der *Aktivität* beinhaltet den Reaktionsnamen. Der *Aktivität* kann ein UML2 AD Element *Constraint* angefügt werden um das *Zeitverhalten* zu definieren (siehe *Zeitverhalten*). Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Nachrichtenschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | (*Steuergerät* **sendet Nachricht** *Nachricht* **mit Wert** Wert **an** *Steuergerät* [*Zeitverhalten*]. \| *Steuergerät* **empfängt Nachricht** *Nachricht* **mit Wert** Wert **von** *Steuergerät* [*Zeitverhalten*].) |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Nachrichtenschritt wird durch eine UML2 AD *Sequenz* dargestellt. Das erste Element in der Sequenz ist ein UML2 AD Element *Aktion* mit dem Stereotypen *Steuergerät*. Anschliessend folgt ein UML2 AD Element *Objektknoten* mit dem Stereotypen *Nachricht*. Die beiden UML2 AD Elemente verbindet ein UML2 AD Element *Objektfluss* mit dem Stereotypen *sendet Nachricht* bzw. *empfängt Nachricht*. Der Name des UML2 AD *Objektknotens* enthält den Namen der *Nachricht*. Der Wert der *Nachricht* wird durch ein UML2 AD *Constraint* mit dem Stereotypen *Wert* dargestellt. Vom UML2 AD *Objektknoten* das Empfänger- bzw. das Sender-Steuergerät der Nachricht durch einen UML2 AD Element *Objektfluss* verbunden. Das Empfänger- bzw. das Sender-Steuergerät wird durch ein UML2 AD Element *Aktion* mit dem Stereotypen *Steuergerät* dargestellt.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Ereignisschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Ereignis** *Ereignis* [*Parameter*] |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Ereignisschritt wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Ereignis* dargestellt. Der Name der *Aktivität* ist der Ereignisname. Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

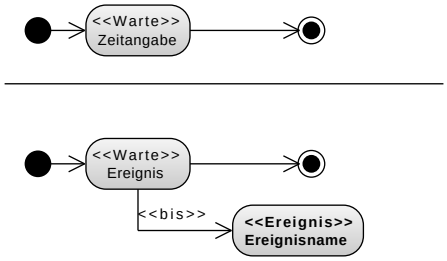
| Name | Aktion |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *AktionID Aktionsname* |
| UML2 AD Darstellung |  |
| Erläuterung | Erläuterung siehe *Aktionsschritt*. |

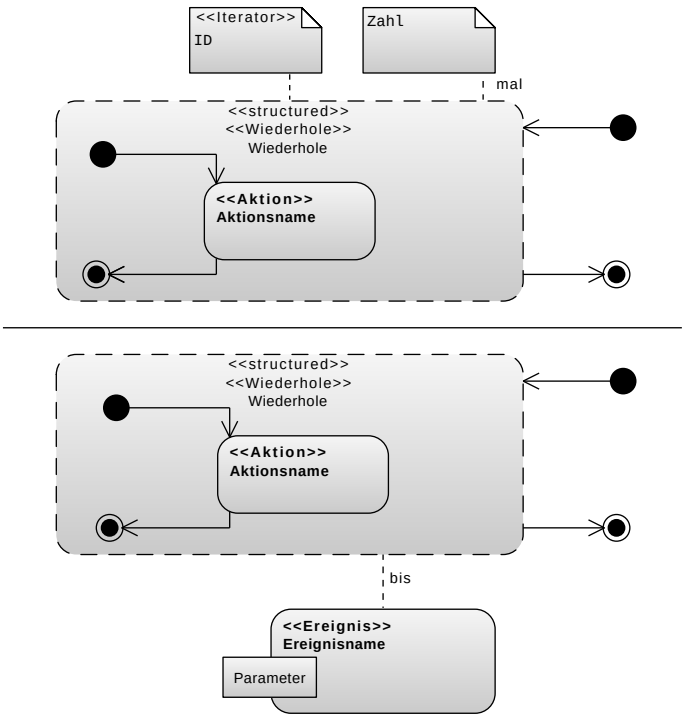
| Name | Reaktion |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *ReaktionID Reaktionsname* |
| UML2 AD Darstellung |  |
| Erläuterung | Erläuterung siehe *Reaktionsschritt*. |

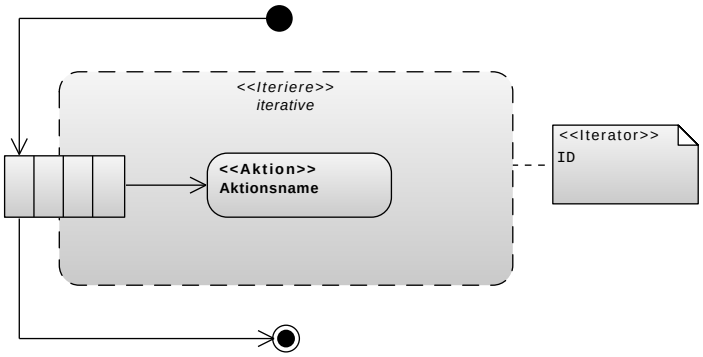
| Name | Ereignis |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *EreignisID Ereginisname* |
| UML2 AD Darstellung | keine UML 2 AD Darstellung, da abstrakt zu sehen. |
| Erläuterung | Erläuterung siehe *Ereignisschritt*. |

### Kontrollstrukturen der Sprache

| Name | Kontrollstruktur |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *Warteanweisung* \| *Wiederholung* \| *Iteration* |
| UML2 AD Darstellung | keine UML 2 AD Darstellung, da abstrakt zu sehen. |
| Erläuterung | Eine Kontrollstruktur ist abstrakt definiert (siehe Syntax), das bedeutet es besitzt keine UML2 AD Darstellung, sondern die Darstellung erfolgt durch die konkrete Ausprägung.<br>Hinweis: Das Element Kontrollstruktur taucht niemals in einer Testfallbeschreibung auf. |

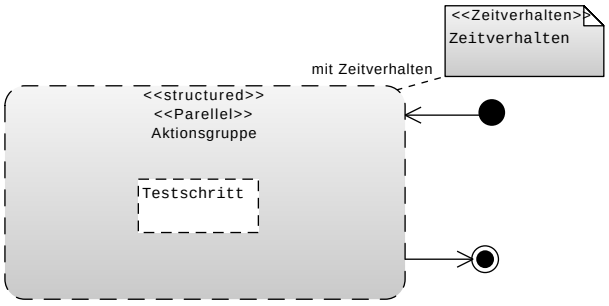
| Name | Warteanweisung |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Warte** (*Zeitangabe* \| **bis** *Ereignis*) |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Warteanweisung wird durch ein UML2 AD Element *Aktion* mit dem Stereotypen *Warte* dargestellt. Die Zeitangabe wird im Namen der *Aktion* dargestellt. Bei einer Warteanweisung mit einem Ereignis lautet der Name der *Aktion* Ereignis. Das Ereignis wird als eine UML2 AD *Aktivität* mit dem Stereotypen *Ereignis* dargestellt und der Name der *Aktivität* ist der *Ereignisname*. Die beiden UML2 AD Elemente verbindet ein UML2 AD *Kontrollfluss* mit dem Stereotypen *bis*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

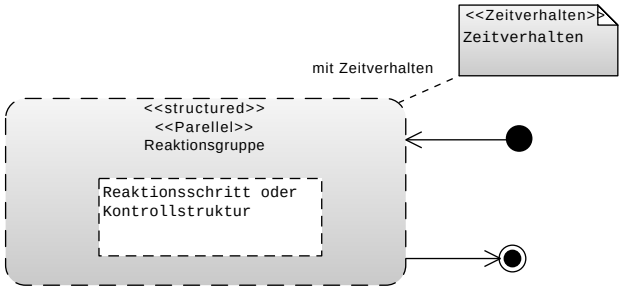
| Name | Wiederholung |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Wiederhole** ( **mit Iterator** *ID Zahl* **mal** \| **bis** *Ereignisschritt* ) **Anfang**<br>{*Testschritt*}.<br>**Ende Wiederhole** |
| UML2 AD Darstellung |  |
| Erläuterung | Die Wiederholung wird mit einem UML2 AD Element *Loop Node* mit dem Stereotypen *Wiederhole* dargestellt. Innerhalb der *Loop Node* werden die *Testschritte* dargestellt, wobei der erste *Testschritt* durch ein UML2 AD Element *Initialknoten* innerhalb der *Loop Node* mit einem UML2 AD Element *Kontrollfluss* verbunden ist. Ein UML2 AD Element *Endknoten* ist mit dem letzten *Testschritt* durch ein UML2 AD Element *Kontrollfluss* verbunden. Das UML2 AD Element Loop Node kann entweder ein UML2 AD *Constraint* mit der *Zahl* der Wiederholungen angehängt sein, oder eine Darstellung eines *Ereignises*. Beide Möglichkeiten sind durch ein UML2 AD Element *Constraint* verbunden, wobei bei der Zahl der Name des Constraints *mal* lautet und bei dem Ereignis *bis*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Iteration |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Iteriere mit Iterator** *ID* durch *Liste* **Anfang**<br>{*Testschritt*}.<br>**Ende Iteriere** |
| UML2 AD Darstellung |  |
| Erläuterung | Die Iteration wird durch ein UML2 AD Element *Expansion Region* mit dem Stereotypen *Iteriere* dargestellt. Die *Liste* wird durch ein UML2 AD *Expansion Node* dargestellt, welches an dem *Expansion Region* anhängt. Der *Iterator* wird durch ein UML2 AD *Constraint* mit dem Stereotypen *Iterator* dargestellt, der an dem *Expansion Region* angehängt ist. Der Inhalt des *Constraints* in die *ID*. Innerhalb der *Expansion Region* werden die *Testschritte* dargestellt.<br>Die *Liste* wird in dem *Expansion Node* durch UML2 AD *Tagged Values* dargestellt. Die *Iteration* wird durch die *Expansion Region* mit einem UML2 AD *Objektfluss* betreten und verlassen. Innerhalb der *Expansion Region* wird ausgehend von der *Expansion Region* der erste Testschritt durch ein UML2 AD *Objektfluss* verbunden.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Objektfluss*. |

### 6.5.4 Elemente zur Strukturierung

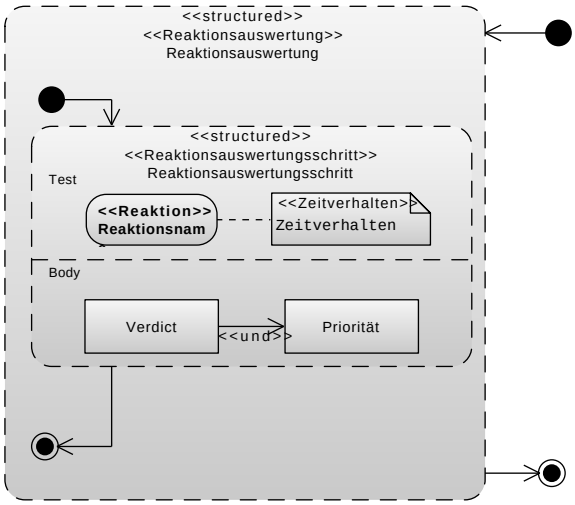
| Name | Testfallschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Testfallaufruf** *TestfallID* [*Parameter*] |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Testfallschritt wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Testfallaufruf* dargestellt. Der Name der Aktivität ist die TestfallID und optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

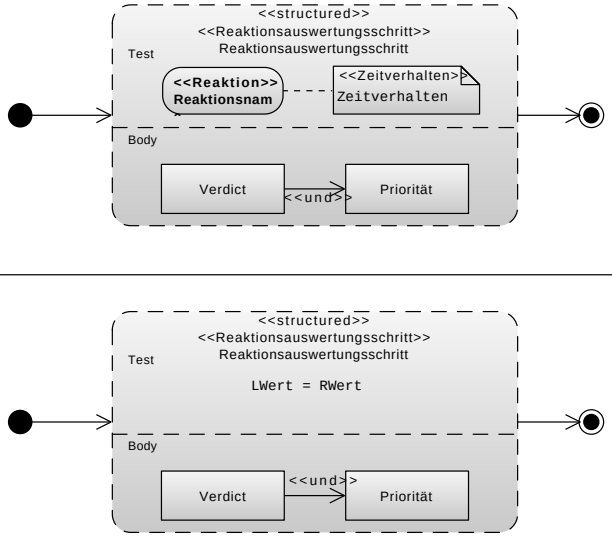
| Name | Aktionsgruppe |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Gruppe von Aktionen** [*Parallel*] [*Zeitverhalten*] **Anfang**<br>{*Testschritt*.}<br>**Ende Gruppe von Aktionen** |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Aktionsgruppe wird durch ein UML2 AD Element *Structured Activity Node* dargestellt mit dem Stereotypen *Aktionsgruppe*. Das optionale Sprachelement *Parallel* wird durch den Stereotyp *Parallel* dargestellt, das sich in zweiter Position befindet. Ein optionales Zeitverhalten wird durch ein UML2 AD Element *Constraint* dargestellt (siehe Darstellung *Zeitverhalten*). In dem *Structured Activity Node* kann ein oder mehrere *Testschritte* dargestellt werden (im Bild durch ein weißes Rechteck mit Inschrift *Testschritt* kenntlich gemacht).<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Reaktionsgruppe |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Gruppe von Reaktionen** [*Parallel*] [*Zeitverhalten*] **Anfang**<br>{*Reaktionsschritt.* \| *Reaktionsgruppe.* \| *Kontrollstruktur.*}<br>**Ende Gruppe von Reaktionen** |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Reaktionsgruppe wird durch ein UML2 AD Element *Structured Activity Node* dargestellt mit dem Stereotypen *Reaktionsgruppe*. Das optionale Sprachelement *Parallel* wird durch den Stereotyp *Parallel* dargestellt, das sich in zweiter Position befindet. Ein optionales Zeitverhalten wird durch ein UML2 AD Element *Constraint* dargestellt (siehe Darstellung *Zeitverhalten*). In dem *Structured Activity Node* kann ein oder mehrere *Reaktionsschritte* oder *Kontrollstrukturen* dargestellt werden (im Bild durch ein weißes Rechteck mit Inschrift *Reaktionschritt oder Kontrollstruktur* kenntlich gemacht).<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Referenz |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **import** *TestfallbeschreibungID* |
| UML2 AD Darstellung |  |
| Erläuterung | Referenz wird durch ein UML2 AD Element *Aktion* mit dem Stereotypen *Referenz* dargestellt.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

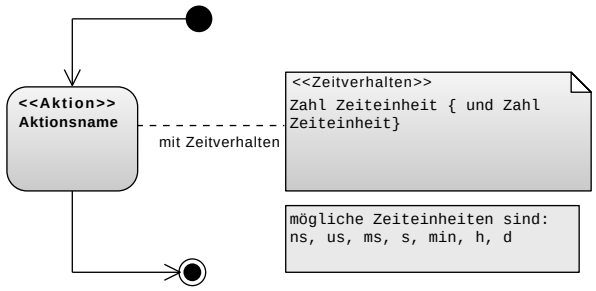
### 6.5.5 Elemente zur Reaktionsauswertung

| Name | Reaktionsauswertung |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Reaktionsauswertung Anfang**<br>{*Reaktionsauswertungsschritt*.}<br>**Ende Reaktionsauswertung** |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Reaktionsauswertung wird durch ein UML2 AD Element *Sequence Node* mit dem Stereotypen *Reaktionsauswertung* dargestellt. Innerhalb der *Sequence Node* werden die *Reaktionsauswertungsschritte* dargestellt. Ein Initialknoten innerhalb der *Sequence Node* stellt den Anfang dar. Ein Endknoten innerhalb der *Sequence Node* stellt das Ende der *Reaktionsauswertung* dar. Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Reaktionsauswertungsschritt |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Wenn** (*Reaktionsschritt* \| *LWert* **=** *RWert*) **dann** (**Verdict:** *Verdict* \| **Priorität:** *Priorität* \| *Testfallschritt*) {**und** (**Verdict:** *Verdict* \| **Priorität:** *Priorität*)} |
| UML2 AD Darstellung |  |
| Erläuterung | Ein Reaktionsauswerungsschritt wird durch ein UML2 AD Element *Conditional Node* mit dem Stereotypen Reaktionsauswertungsschritt dargestellt. Der Test Bereich der UML2 AD *Conditional Node* enthält die UML2 AD Darstellung für die *Reaktion* bzw. den Wertevergleich (*LWert = RWert*). Im Body Bereich wird das *Verdict* bzw. die *Priorität* durch ein UML2 AD *Objektknoten* dargestellt. Mehrere Objektknoten werden durch das UML2 AD Element *Objektfluss* mit dem Stereotypen *und* verbunden dargestellt.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Verdict |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **pass** \| **fail** |
| UML2 AD Darstellung | |
| Erläuterung | |

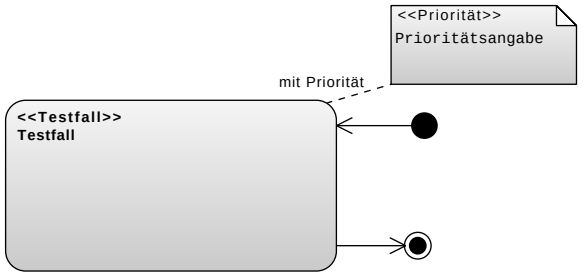
### 6.5.6 Elemente für die zeitlichen Aspekte

| Name | Zeitverhalten |
|---|---|
| Syntax der Testfall-Spezifikationssprache | (**mit Zeitverhalten** *Zeitaspekt* {**und** *Zeitaspekt*} \| **bis** *Ereignisschritt*) |
| UML2 AD Darstellung |  |
| Erläuterung | Das Zeitverhalten wird mit einem UML2 AD Element *Constraint* mit dem Stereotypen *Zeitverhalten* dargestellt. Die Verbindung erhält den Namen „mit Zeitverhalten“. Innerhalb des Constraints können *Zeitangaben* gemacht werden. Eine *Zeitangabe* besteht aus einer Zahl gefolgt von einer Zeiteinheit. Mehrere Zeitangaben können durch das Schlüsselwort „und“ verknüpft werden. Eine Zeiteinheit kann sein: ns, us, ms, s, min, h, d (siehe Definition *Zeiteinheit* im vorherigen Abschnitt). |

Die Elemente Zeitaspekt, Zeitangabe und Zeiteinheit sind in der Darstellung vom Zeitverhalten zu finden.

### 6.5.7 Hilfselemente der Sprache

| Name | Parameter |
|---|---|
| Syntax der Testfall-Spezifikationssprache | ( **mit Parameter** *ParameterID* \| **mit Parameterliste** *Liste*) |
| UML2 AD Darstellung |  |
| Erläuterung | Parameter sind ein Teil von dem UML2 AD Element *Aktivität*. Die UML2 AD *Aktivitätsparameter* können einzeln oder beliebig oft vorkommen. Die Schlüsselwörter „mit Parameter“ und „mit Parameterliste“ entfallen bei der UML2 AD Darstellung. |

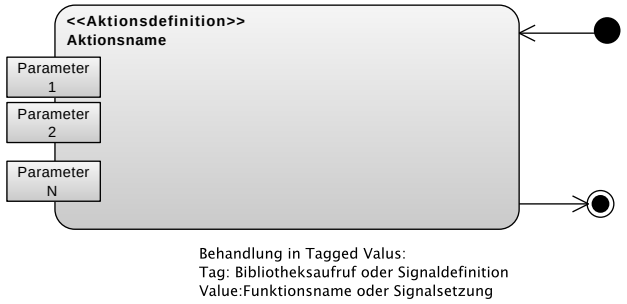
| Name | Priorität |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **mit Priorität** *Prioritätsangabe* |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Priorität wird durch ein UML2 AD *Constraint* mit dem Stereotypen *Priorität* dargestellt. Die Verbindung trägt den Namen „mit Priorität“. |

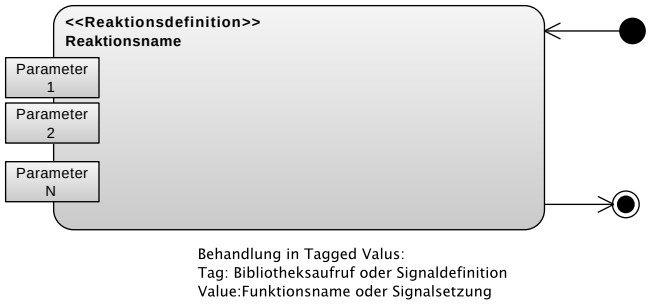
| Name | Liste |
|---|---|
| Syntax der Testfall-Spezifikationssprache | *ListenElement* {**,** *ListenElement*} |
| UML2 AD Darstellung | keine eigene UML 2 AD Darstellung vorhanden. |
| Erläuterung | Die Liste taucht in der UML2 AD Darstellung nicht visuell auf. Tagged Values werden eingesetzt. |

Die Elemente ListenElement, Steuergerät, sowie Steuergerätname, NameID, LWert, RWert, TestfallbeschreibungID, Signalname, Signalverlauf, Zahl und Prioritätsangabe benötigen keine visuelle Darstellung.

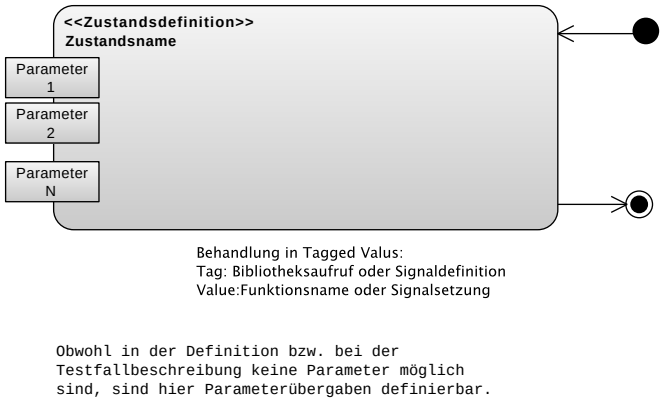
### 6.5.8 Anbindung an die Testfallimplementierung

#### Definition der Aktionen, Reaktionen, Ereignisse und Zustände

| Name | Aktionsdefinition |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Def Aktion** *Aktion* [*Parameter*]<br>**:=** (*Aktion* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareAktionen*.}<br>**Ende Def Aktion**) |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Aktionsdefinition wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Aktionsdefinition* dargestellt. Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden. Falls die Definition durch eine andere *Aktion* erfolgt, wird innerhalb der *Aktivität* eine *Aktion* mit gesetztem Parameter bzw. Zeitverhalten dargestellt. Falls die Definition durch *AtomareAktion* erfolgt, wird die Darstellung extern nicht sichtbar. Die UML2 AD *Tagged Values* werden für die Darstellung verwendet. In den Tag Bereich steht das Schlüsselwort *Bibliotheksaufruf* oder *Signaldefinition* geschrieben. In dem Value Bereich der *Funktionsaufruf* bzw. die *Signalsetzung*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Reaktionsdefinition |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Def Reaktion** *Reaktion* [*Parameter*]<br>(**:=** *Reaktion* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareReaktionen.*}<br>**Ende Def Reaktion**) |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Reaktionsdefinition wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Reaktionsdefinition* dargestellt. Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden. Falls die Definition durch eine andere *Reaktion* erfolgt, wird innerhalb der *Aktivität* eine *Aktion* mit gesetztem Parameter bzw. Zeitverhalten dargestellt. Falls die Definition durch *AtomareReaktion* erfolgt, wird die Darstellung extern nicht sichtbar. Die UML2 AD *Tagged Values* werden für die Darstellung verwendet. In den Tag Bereich steht das Schlüsselwort *Bibliotheksaufruf* oder *Signaldefinition* geschrieben. In dem Value Bereich der *Funktionsaufruf* bzw. die *Signalsetzung*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Ereignisdefinition |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Def Ereignis** *Ereignis* [*Parameter*]<br>(**:=** *Ereignis* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareEreignis.*}<br>**Ende Def Ereignis**) |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Ereignisdefinition wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Ereignisdefinition* dargestellt. Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden. Falls die Definition durch eine andere *Ereignis* erfolgt, wird innerhalb der *Aktivität* eine *Aktion* mit gesetztem Parameter dargestellt. Falls die Definition durch *AtomaresEreignis* erfolgt, wird die Darstellung extern nicht sichtbar. Die UML2 AD *Tagged Values* werden für die Darstellung verwendet. In den Tag Bereich steht das Schlüsselwort *Bibliotheksaufruf* oder *Signaldefinition* geschrieben. In dem Value Bereich der *Funktionsaufruf* bzw. die *Signalsetzung*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

| Name | Zustandsdefinition |
|---|---|
| Syntax der Testfall-Spezifikationssprache | **Def Zustand** *Zustand* [*Parameter*]<br>(**:=** *Zustand* [*Parameter*] [*Zeitverhalten*] \| **Anfang** {*AtomareZustand*.}<br>**Ende Def Zustand**) |
| UML2 AD Darstellung |  |
| Erläuterung | Eine Zustandsdefinition wird durch ein UML2 AD Element *Aktivität* mit dem Stereotypen *Zustandsdefinition* dargestellt. Optional können Parameter mittels dem UML2 AD *Aktivitätsparameter* angegeben werden. Falls die Definition durch eine andere *Zustand* erfolgt, wird innerhalb der *Aktivität* eine *Aktion* mit gesetztem Parameter bzw. Zeitverhalten dargestellt. Falls die Definition durch *AtomarerZustand* erfolgt, wird die Darstellung extern nicht sichtbar. Die UML2 AD *Tagged Values* werden für die Darstellung verwendet. In den Tag Bereich steht das Schlüsselwort *Bibliotheksaufruf* oder *Signaldefinition* geschrieben. In dem Value Bereich der *Funktionsaufruf* bzw. die *Signalsetzung*.<br>Ein und ausgehende Kanten sind vom Typ UML2 AD Element *Kontrollfluss*. |

### Atomare Aktionen, Reaktionen, Ereignisse und Zustände

Diese Gruppe der Sprachelemente benötigen keine visuelle Darstellung. Wie im vorherigen Abschnitt über die Definition der Aktionen, Reaktionen, Ereignisse und Zustände deutlich wird, werden Tagged Values für die Abbildung eingesetzt.

**Sprachelemente zu Funktionsaufrufen und Signalsetzungen**

Diese Gruppe der Sprachelemente benötigen keine visuelle Darstellung. Wie im vorherigen Abschnitt über die Definition der Aktionen, Reaktionen, Ereignisse und Zustände deutlich wird, werden Tagged Values für die Abbildung eingesetzt.

## 6.6 Vorschlag zur feingranuralen Definition einiger Sprachelemente

In diesem Abschnitt wird die Beschreibung der Aktionen, Reaktionen, Ereignisse und Zustände, die Sprachelemente zur Anbindung an die Testfallimplementierung und die Beschreibung eines Signalverlaufs näher betrachtet.

### 6.6.1 Benennung der Aktionen, Reaktionen, Ereignisse und Zustände

Für die Benennung der Aktion, Reaktionen, Ereignisse und Zustände ist Freitext vorgesehen. Dieser Freiheitsgrad ist zwingend notwendig, um alle denkbaren Testfälle erstellen zu können.

Für eine einheitliche Formulierung ist es sinnvoll diesen Freiheitsgrad der Freitexte einzuschränken und dabei trotzdem alle denkbaren Testfälle erstellen zu können. Falls genügend Formalisierung erreicht werden kann, ist eine automatisierte Transformation dieser Bereiche ebenfalls möglich (z.B. in eine andere Sprache).

Ein denkbares Konzept ist der Einsatz von kontrollierter Sprache (vgl. Kapitel Stand der Technik, Abschnitt 5.3.6, S. 165). Die vier Elemente besitzen Grundlegende Eigenschaften, die hier zur Einschränkung verwendet werden können. Beispiel: Der Zustand kann in den meisten Fällen durch Einsatz von 'ist' formuliert werden, wie: „Klemme 15 ist an“.

### 6.6.2 Definiton der Aktionen, Reaktionen, Ereignisse und Zustände

Die atomaren Aktionen, Reaktionen, Ereignisse und Zustände sollten bei einer Anbindung an ein Werkzeug zur Testdurchführung mehr als String-Handling sein.

Für die Definition in EBNF ist es ausreichend die atomaren Aktionen mit Strings darzustellen. Bei der Ankopplung der Testfälle an Testskripts sind fehlerhafte Eingaben mühselig zu finden. Um eine automatisierte Überprüfung der Eingabe zu erleichtern wird hier ein regulärer Ausdruck für einen Parser definiert. Dieser kann durch ein Constraint (lightweight Extension) den atomaren Ausdrücken hinzugefügt werden.

Für Funktionen gilt:

```
\w* \w+ \s*? (.*) \s*? ;?
```

Dieser reguläre Ausdruck wird an folgenden Beispielen verdeutlicht:

```
1 String setParam (Parameter1, Parameter2)

3 callSetting(Parameter1, Parameter2)
```

Es ist möglich aber nicht notwendig einen Rückgabewert zu definieren. Der Funktionsname ist notwendig. Nach dem Funktionsnamen kann ein Leerzeichen (oder mehrere) erfolgen, muss aber nicht. Vorhanden sein muss folgend eine Klammer (auf) mit beliebigem Inhalt mit einer darauffolgenden notwendigen Klammer zu. Optional kann ein Semikolon angegeben werden oder nicht. Hier ebenfalls mit keinem, einem oder mehreren Leerzeichen zwischen Klammer und Semikolon.

Für die Signalsetzung wird die Gleichung beibehalten. Allerdings wird die rechte Seite der Gleichung wie folgt definiert.

### 6.6.3 Signalbeschreibung nach IEEE 1641

Für die Signalbeschreibung wird der Standard IEEE 1641 [Ins06] verwendet, die eine Darstellung der Signalverläufe u.a. in XML definiert. Der Standard ist für den Einsatz im Elektrik/Elektronik Bereich spezifiziert und eignet sich damit für die Beschreibung von Signalverläufen für Steuergerätetests.

## 6.7 Zusammenfassung

In diesem Kapitel wurde ein Entwurf für eine Testfall-Spezifikationssprache, basierend auf den Erkenntnissen der vorherigen Kapitel, erstellt.

Die Testfall-Spezifikationssprache besitzt eine formale Syntax, die durch eine informelle Semantik ergänzt wurde. Unabhängig davon ist mit den Definitionen der Grundelemente die Testfallbeschreibung abgeschlossen, d.h. die Funktionsaufrufe und Signalsetzungen bilden die atomaren Aufrufe.

Zudem ist eine graphische Repräsentation der Sprache als ein Profil zu dem UML2 Aktivitätsdiagramm definiert.

Der Aufbau der Testfall-Spezifikationssprache ist durch die unterschiedlichen Ansätze in Kapitel Stand der Technik, S. 123 inspiriert, im Besonderen durch den kommenden Standard ATML.

# Kapitel 7

# Werkzeugunterstützte Eingabe von Testfallbeschreibungen

Die Implementierung des Konzepts soll einen Editor ergeben, der den Testdesigner bei der Testfallerstellung unterstützt. Für die Erstellung eines solchen Editors gibt es unterschiedliche Ansätze. In diesem Kapitel wird die Herleitung des Editors auf Basis des Konzepts aufgezeigt.

Im ersten Schritt werden vorhandene Implementierungswerkzeuge auf ihre Einsetzbarkeit hin bewertet und ausgewählt. Anschliessend folgt eine Vorstellung der Implementierung des Editors. Eine Beispielanwendung des entwickelten Editors schliesst das Kapitel ab.

## 7.1 Auswahl des Werkzeugs für die Implementierung

Bei der Auswahl des Implementierungswerkzeugs werden zuerst Kriterien definiert, mit denen die Werkzeuge bewertet werden. Anschliessend erfolgt eine Vorstellung der Implementierungswerkzeuge mit einer Bewertung aller Werkzeuge und Auswahl eines Werkzeugs für die Implementierung.

### 7.1.1 Bewertungskriterien für die Implementierungswerkzeuge

Für die Auswahl des Implementierungswerkzeugs werden Bewertungskriterien definiert.

- Implementierungsaufwand/-dauer

  Aus der Implementierung des Editors ist der Implementierungsaufwand zu betrachten.

- Verwendete Programmiersprache

  Weiterhin wird die verwendete Programmiersprache für die Implementierung des Editors betrachtet.

  Dieses Kriterium ist ein Indikator für die Eingliederung des Editors in eine vorhandene Toollandschaft des Unternehmens.

- Mächtigkeit des Werkzeugs

  Die Konzeptumsetzung umfasst den Entwurf, die Implementierung und die Erstellung des Editors. Dieses Kriterium dient zur Bewertung der Unterstützung dieser Schritte.

  Konkret sollen zwei Fragen mit diesem Kriterium betrachtet werden: Ob man mit dem Werkzeug eine domänenspezifische Sprache erstellen kann (Design und Implementierung) und ob man einen Editor für die so erzeugte Sprache erstellen kann.

  Dieses Kriterium hat einen direkten Einfluss auf den Implementierungsaufwand.

- Unterstützung der Eingabe

  Der erstellte Editor soll den Testdesigner bei der Erstellung des Editors unterstützen.

  Dieses Kriterium wird im nachfolgenden Abschnitt genauer spezifiziert.

### Anforderungen an den Editor

Das Kriterium „Unterstützung der Eingabe“ für das Implementierungswerkzeug ist eine Anforderung an den (zu erstellenden) Editor, den der Testdesigner bei der Testfallbeschreibung verwenden soll.

Dabei ist das primäre Ziel des Editors, Fehler bei der Eingabe zu minimieren und eine einheitliche Darstellung der Testfallbeschreibungen über mehrere Testdesigner hinweg zu erzielen. Die Unterstützung der Eingabe wird durch folgende Kriterien definiert:

- Verwaltung der Projektdokumente

  Ein Testdesigner arbeitet meist gleichzeitig an mehreren Projekten. Der Editor soll in der Lage sein, die unterschiedlichen Projektdokumente zu verwalten.

- Verwaltung der Aktionen, Reaktionen, Ereignisse und Zustände

  Da die wesentlichen Sprachelemente bei der Testfall-Spezifikationssprache die Aktionen, Reaktionen, Ereignisse und Zustände sind und auf diesen basierend die Testfälle erstellt werden, ist eine Verwaltung der vorhandenen bzw. erstellten Aktionen, Reaktionen, Ereignisse und Zustände notwendig.

- Hervorhebung der Schlüsselwörter

  Um die Eingabe zu erleichtern, ist es sinnvoll, dass die Sprachelemente der Testfall-Spezifikationssprache hervorgehoben werden, um damit eine visuelle Trennung der Sprachelemente der Sprache und dem erstellten Inhalt des Testdesigners zu ermöglichen. (Im Fachjargon *Syntax Highlighting* genannt.)

- Vorgaben / Templates

  Für immer wiederkehrende Textbausteine sollen Templates erstellt werden können.

- Eingabeunterstützung

  Bei der Eingabe sollen die nächstmöglichen Eingabemöglichkeiten aufgezeigt werden, um die Arbeit des Testdesigners zu erleichtern.

- Workflowunterstützung

  Der Workflow bei der Erstellung eines Testfalls soll unterstützt werden.

### 7.1.2 Implementierungswerkzeuge

Die Auswahl der möglichen Implementierungswerkzeuge für die Umsetzung des Konzepts sind übersichtlich.

Im Folgenden werden

- DSL Werkzeuge von Microsoft
- MSP von Jetbrains
- Xtext Erweiterung für Eclipse und die
- Eigenentwicklung

betrachtet, mit denen textuelle domänenspezifische Sprachen erstellt werden können. Implementierungswerkezuge mit denen visuelle (graphische) domänenspezifische Sprachen erstellt werden können wie z.B. MetaEdit+ werden nicht in Betracht gezogen.

Es folgt eine kurze Vorstellung der Werkzeuge.

### DSL Werkzeuge von Microsoft

Das ursprünglich für die Entwicklung von domänenspezifischen Sprachen gedachte Projekt[1] wird aktuell in Richtung Datenbanken und Datenverarbeitung gelenkt und befindet sich momentan noch in der Entwicklung[2].

Das Projekt besitzt eine eigene Modellierungssprache, die *M* genannt wird (Stand 11/2008[3]). *M* basiert auf einem XML-Dialekt[4]. Die zweite Komponente, Quadrant genannt, ist eine visuelle Möglichkeit auf die Daten zuzugreifen und die dritte Komponente ist für die Datenhaltung (Repository) zuständig. Die Entwicklung ist auf die .NET Plattform ausgerichtet.

Bei der Modellierungssprache *M* können domänenspezifische Datenmodelle erstellt werden (MSchema), eine domänenspezifische Grammatik (MGrammar) und ein abstraktes Datenmodell (MGraph) für die in MSchema erstellten Datenmodelle.

Eine praktische Erfahrung, besonders mit der Grammatik, konnte nicht gesammelt werden, da die Vorabversionen nicht zum Laufen gebracht werden konnten. Trotzdem ist das Projekt wegen der Anbindung an die .NET Plattform erwähnenswert. In [WKK09] wird bei der Evaluierung des Mittels auf die vielen fehlenden Features für die Erstellung einer DSL hingewiesen. Jedoch ist mittels dieses Werkzeugs erfolgreich domänenspezifische Sprachen erstellt worden [DGH09].

### MSP von Jetbrains

Das in Java geschriebene MSP (Meta Programming System)[5] ist ebenfalls in der Entwicklung und in einer Vorabversion erhältlich. Es hat eine visuelle Oberfläche für die Entwicklung von textuellen DSLs.

Im ersten Schritt werden in der Syntax Knoten (Nodes) definiert. Durch Constraints können die möglichen Eingaben spezifiziert werden, die ein Node haben kann (z.B. nur Zahlen oder nur Zeichen) oder die Beziehung (z.B. Referenzierung) zu anderen Nodes.

[1] http://www.heise.de/ix/meldung/Microsoft-gewaehrt-ersten-Einblick-in-Visual-Studio-2010-208811.html
[2] http://en.wikipedia.org/wiki/Oslo_(Microsoft)
[3] http://www.microsoft.com/germany/msdn/events/archiv/technicalsummit08/-library.aspx?id=msdn_de_30139
[4] http://www.heise.de/newsticker/meldung/Microsoft-bringt-neue-Programmiersprache-M-210879.html
[5] http://www.jetbrains.com/mps/index.html

In den weiteren Schritten wird das aussehen im Editor festgelegt und - falls notwendig - werden Datentypen spezifiziert. Beim erstellen der Syntax sieht man den starken Zusammenhang zu dem Unterbau, den Javaklassen.

Eine Bewertung auf praktischen Erkenntnissen konnte nicht vollständig erfolgen. Das Hauptproblem bei MSP lag in dem enormen Speicherverbrauch des (in der Vorabversion befindlichen) Werkzeugs während der Grammatikdefinition, so dass umfangreiche Sprachdefinitionen nach einer gewissen Zeit nicht mehr möglich waren.

### Xtext Erweiterung für Eclipse

Ebenfalls in Java entwickelt ist die Xtext Erweiterung[6] für Eclipse[7].

Mit Xtext werden textuelle domänenspezifische Sprachen entwickelt. Das Werkzeug begleitet durchgehend den Entwurf, die Implementierung und Erstellung eines Editors. Der Entwurf erfolgt in einer EBNF ähnlichen Modellierungssprache. Anschliessend kann durch einen Generator die notwendige Implementierung und der Editor erstellt werden. Weitere Bestandteile sind wie bei MSP, die Erstellung von Datentypen und Constraints, um bei der Eingabe nur bestimmte Eingabewerte zu erlauben. Das Werkzeug hat eine Anbindung zur Codegenerierung mittels der Templatesprache Xpand[8].

Praktische Erfahrungen konnten aufgrund der guten Dokumentation und eines durchdachten Workflows schnell gesammelt werden, so dass die Implementierung des Editors auf Xtext basiert.

Der erzeugte Editor erfüllt alle Anforderungen an einen Editor (siehe vorheriger Abschnitt) bis auf die Workflowunterstützung. Die Verwaltung der Projektdokumente und der Sprachelemente muss explizit definiert werden (siehe nächster Abschnitt).

### Eigenentwicklung

Eine Eigenentwicklung würde die meiste Zeit in Anspruch nehmen, da ein Framework (d.h. Umgebung mit notwendigen Werkzeugen) entwickelt werden müsste um die eigentliche Sprache zu entwickeln ( vgl. [ZS09]). Zum Beispeil wird für die korrekte Erkennung der Sprache ein Parser benötigt. Das Framework würde also in Grundzügen wie bei Xtext und MSP aus einem Parser bestehen, der die DSL in ein

[6] http://www.eclipse.org/Xtext/
[7] http://www.eclipse.org
[8] http://wiki.eclipse.org/Xpand

semantisches Modell überführt. Das semantische Modell wird u.a. zur Codegenerierung oder zur syntaktischen Analyse (Korrektheit) benutzt. Zusätzliche Funktionen, wie die Erstellung des Editors, würden die Erweiterung des Frameworks durch weitere Bestandteile bedeuten[9].

Vorhandene lexikalische Parser für das Parsen der DSL würden verwendet werden, aber die Anpassungen der vorhandenen Bibliotheken in ein Framework würde zeitlich die Entwicklung der Sprache übersteigen. Weiterhin müsste ein Workflow und Tools definiert bzw. erstellt werden, um die erstellte Grammatik in einen Editor zu überführen.

Eine Eigenentwicklung ist nur dann sinnvoll, wenn die domänenspezifische Sprache in einen Gesamtkontext, wie einer vorhandenen Toollandschaft eines Unternehmens, eingebettet werden soll.

### 7.1.3 Fazit

| | Oslo / Microsoft | MSP / Jetbrains | Xtext / Eclipse | Eigenentwicklung |
|---|---|---|---|---|
| Implementierungsaufwand | mittel | mittel | gering | hoch |
| Verwendete Programmiersprache | .NET | Java | Java | beliebig |
| Mächtigkeit des Werkzeugs | k. A. | mittel | hoch | abhängig von der Implementierung |
| Unterstützung der Eingabe | k. A. | k. A. | sehr gut | abhängig von der Implementierung |

Legende: “k. A.”: es konnten keine Angaben gemacht werden.

**Tabelle 7.1:** Vergleich vorhandener Implementierungswerkzeuge für die Entwicklung einer textuellen DSL

Tabelle 7.1 zeigt eine Übersicht über die Bewertung der untersuchten Implementierungswerkzeuge.

[9] Angelehnt an das umfangreiche, noch nicht erschienene Buch über domänenenspezifische Sprachen unter http://martinfowler.com/dslwip/

Xtext für Eclipse fällt wegen des durchgehend unterstützten Workflows bei der Erstellung von textuellen DSL positiv auf. Die Grammatik basiert auf einer EBNF ähnlichen Syntax, so dass die im Kapitel 6 (Eine Testfall-Spezifikationssprache für die Beschreibung von Testfällen im automobilen Bereich) vorgestellte Grammatik im Grundaufbau verwendet werden kann. Ein weiteres Argument für Xtext ist die gute Dokumentation und die Möglichkeit mittels einer Templatesprache Codes zu generieren.

Der erzeugte Editor erfüllt - wie erwähnt - fast alle Anforderungen und muss nur marginal erweitert werden. Die Erweiterungsmöglichkeiten sind jedoch hier aufgrund von Eclipse und seinen Erweiterungen fast unbegrenzt möglich solange Java Technologien zum Einsatz kommen.

## 7.2 Implementierung der Testfall-Spezifikationssprache in Xtext

Wie in jedem anderen Entwicklungsframework gibt es auch bei Xtext ein bestimmtes Vorgehen bei der Entwicklung. In diesem Abschnitt wird zunächst dieses Vorgehen aufgezeigt. Anschliessend folgt eine Auflistung wichtiger Ergebnisse bei der Implementierung. Zuletzt wird die Transformation der Testfall-Spezifikationssprache in u.a. UML2 Aktivitätsdiagramme unter Benutzung der Templatesprache Xpand vorgestellt.

### 7.2.1 Vorgehen bei der Entwicklung des Editors

Das Vorgehen entspricht dem Vorgehen aus der Dokumentation des Projekts [xte09][10].

Abbildung 7.1 zeigt die erzeugten Komponenten durch das Impelementierungswerkzeug: Ein Parser/Lexer, ein Serializier, ein Linker, Codeteile für die Verlinkung zu den EMF Komponenten und Codeteile für die Erstellung einer Erweiterung für den Eclipse Editor.

Ohne weit in die Erstellung von (Programmier-)sprachen einzudringen: Parser werden für die syntaktische Analyse verwendet, um die Eingabesprache in ein abstraktes Modell wie einem abstrakten Syntaxbaum zu überführen. Für die lexikalische Analyse eines Parsers wird ein sogenannter Lexer verwendet. Der Parser wird z.B. bei der Codegenerierung durch einen Compiler oder durch einen Interpreter benutzt.

[10] http://www.eclipse.org/Xtext/documentation/0_7_2/xtext.pdf

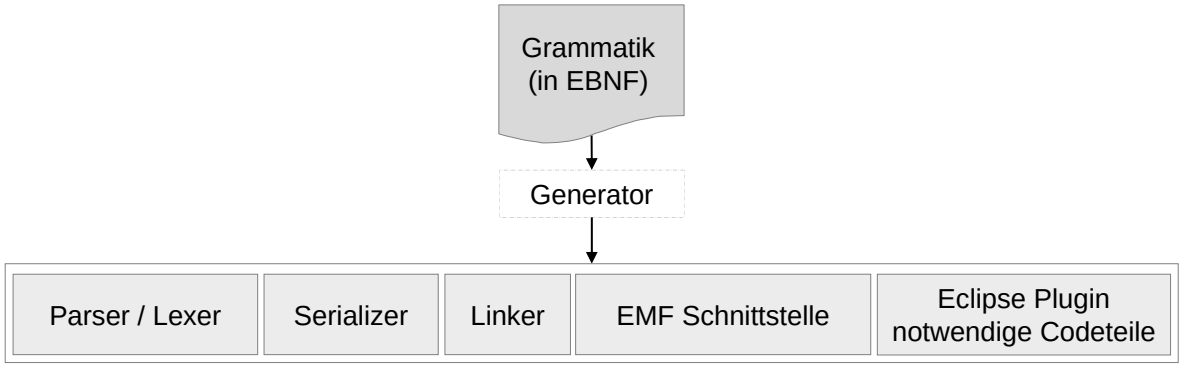



**Abbildung 7.1:** Der Workflow von Xtext

Hier wird der Parser/Lexer dazu verwendet, die textuelle DSL einzulesen, zu parsen und ein abstraktes Model davon zu erstellen (hier: abstrakter Syntaxbaum, AST). Gleichzeitig wird die syntaktische Korrektheit überprüft. Das abstrakte Modell wird intern im Editor verwendet und basiert auf einem Eclipse eigenen Format, der EMF[11] (Eclipse Modeling Framework). Hierfür benötigte Codeteile werden ebenfalls vom Generator erzeugt.

Der Serializier wandelt das (veränderte) Modell wieder in eine textuelle Form, das der definierten Grammatik entspricht.

Der Linker wird benutzt um eine inhaltliche Konsistenz im abstrakten Modell zu halten, so dass nicht erlaubte Crossreferenzen nicht entstehen können.

Die letzte darzustellende Komponente sind die Codeteile, die für eine Erweiterung in Eclipse benötigt werden.

Abbildung 7.2 zeigt das Zusammenwirken dieser generierten Komponenten.

Implementierungsdetails: Der Generator wird mittels dem MWE (Modeling Workflow Engine) [12] gesteuert bzw. konfiguriert, das im weitesten Sinne Java Klassen instantiiert und Methoden aufruft. Zusätzliche Komponenten (wie z.B. eigene Parseranpassungen) können hier in dem Gesamtworkflow des Generators eingebettet werden.

[11] http://www.eclipse.org/modeling/emf
[12] http://wiki.eclipse.org/Modeling_Workflow_Engine_(MWE)

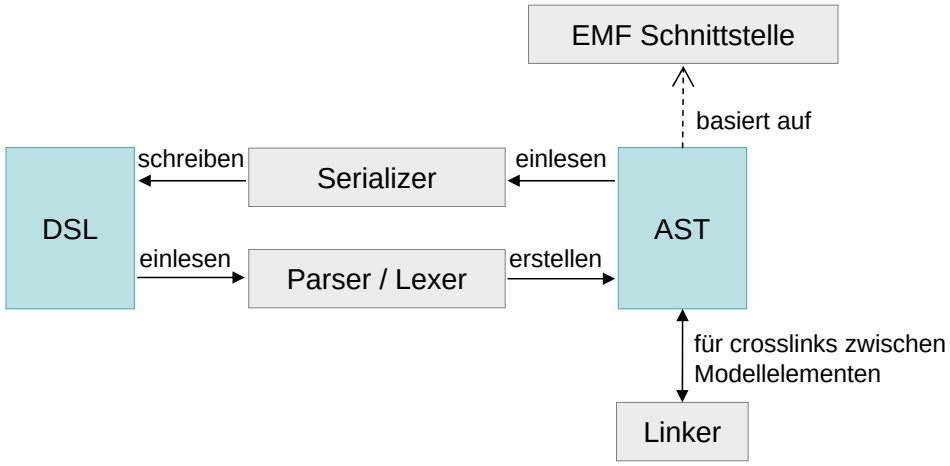



**Abbildung 7.2:** Das Zusammenwirken der erzeugten Komponenten

### 7.2.2 Ergebnisse der Implementierung

Die Vorstellung der Ergebnisse wird auf die Grammatik für den Generator (Eingabe) und den erstellten Editor (Ausgabe) eingeschränkt, da die Auflistung der einzelnen Änderungen der Codeteile nicht mehr zweckmäßig wäre. Einzig die Codeteile für die Transformation der Testfall-Spezifikationssprache sind im Rahmen der Vorstellung relevant, welches im nächsten Abschnitt erfolgt.

#### Die Grammatik für den Generator

Der Generator erzeugt aus einer EBNF ähnlichen Syntax die notwendigen Komponenten. Das Listing 7.1 zeigt einen Auszug aus der Syntaxdefinition. Die Abweichungen beinhalten Informationen für den Linker (siehe oben), so dass Instanzen und Crossreferenzen im abstrakten Modell richtig erzeugt werden können.

```
1 ...
  Testfall :
3 "Testfall" name=ID ( parameter=Parameter )? ( prioritaet=Prio )? "↩
     Anfang"
    ("Beschreibung:" STRING '.')*
5   (vorbed+=Vorbedingung '.')?
    (testschritt+=Testschritt '.')+
7   (nachbed+=Nachbedingung '.')?
  "Ende Testfall" ;
9
Testschritt :
11 testfallschritt+=Testfallschritt  |  aktionschritt+=Aktionschritt | ↩
     nachricht+=Nachrichtenschritt | ereignis+=Ereignisschritt  | ↩
```

```
    kontrollstruktur+=Kontrollstruktur;

Testfallschritt :
 "Testfallaufruf" testfall+=[Testfall] ( parameter=Parameter )? ;
...
```

**Listing 7.1:** Auszug aus der Syntaxdefinition der Testfall-Spezifikationssprache

Anhang F beinhaltet die vollständige Sprachdefinition, wie sie für XText verwendet wird und basiert dabei auf der Syntaxdefinition aus dem vorherigen Kapitel (S. 173).

### Der erzeugte Editor

Die Codeteile, die aus dem Generator erzeugt wurden um eine Erweiterung für Eclipse zu erstellen, erstellen eine Erweiterung, welches

- Syntax Highlighting
- Navigation in der Sprache
- Autovervollständigung
- Outline der Sprache und
- Templates für die Sprache

in Eclipse hinzufügt. Das bedeutet, dass alle Anforderungen an einen Editor aus dem vorherigen Kapitel somit erfüllt sind und der Editor noch einige Besonderheiten mehr aufweist.

Abbildung 7.3 zeigt den Editor in dem, editorunterstützt, ein Testfall erstellt wurde.

Der mittlere Bereich ist für die Erstellung der Testfälle und Definition der Sprachelemente. Der linke Teil entspricht dem Explorer und wird für die Verwaltung der Projektdokumente verwendet. Der rechte Teil ist der sogenannte Outliner, d.h. hier wird angezeigt, welche Sprachelemente, welche Testfälle in dem momentanen offenen Dokument definiert wurden. Weiterhin werden die Inhalte der Testfälle und der Definitionen baumartig strukturiert dargestellt. Im nächsten Abschnitt wird eine beispielhafte Benutzung des Editors mit einem definierten Vorgehen aufgezeigt.

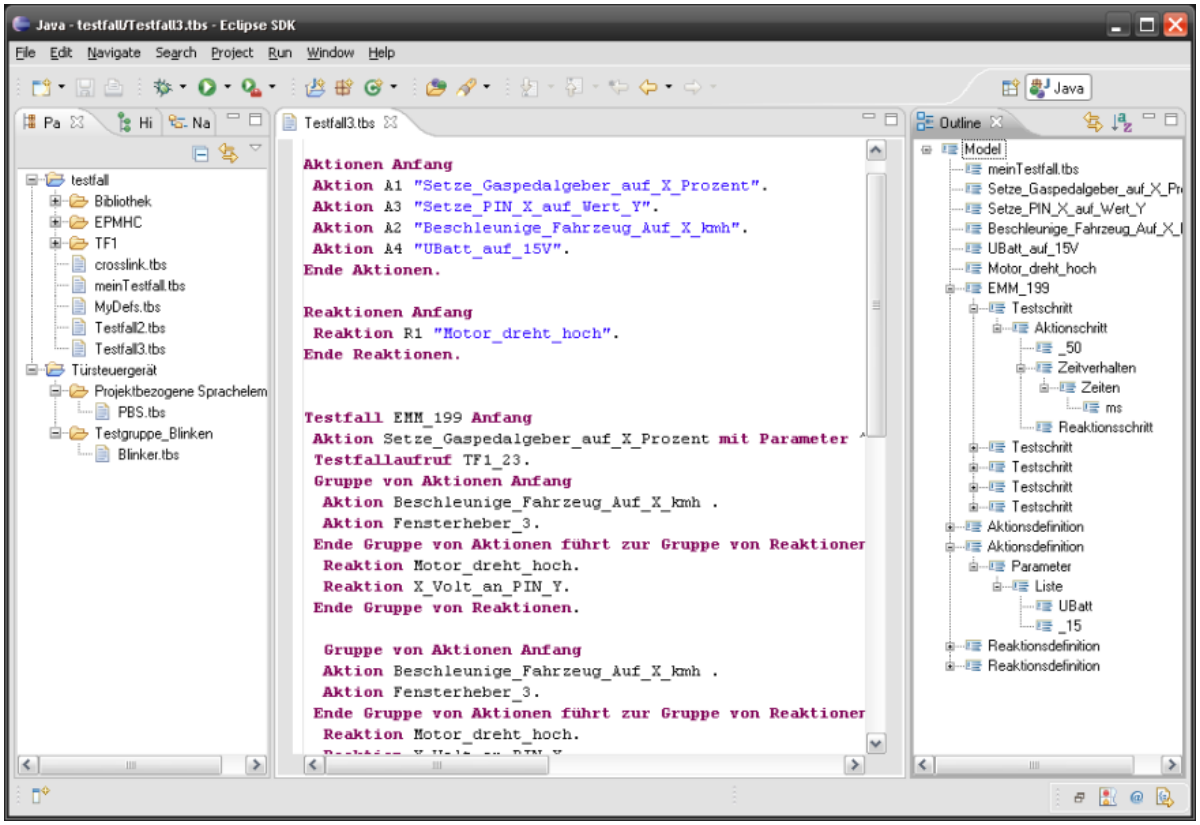



**Abbildung 7.3:** Der erstellte Editor für die Testfall-Spezifikationssprache

### 7.2.3 Transformation der Testfallbeschreibung

Bei der Definition der Testfall-Spezifikationssprache wurden Konzepte zur Transformation in UML2 Aktivitätsdiagramme und die Überführung in Testskripte vorgestellt (vgl. 6.5, S. 192). In diesem Abschnitt werden Implementierungsdetails zu diesen Transformationen vorgestellt. Zuerst folgt ein Abschnitt über die theoretisch möglichen Implementierungen, um anschliessend von diesen Möglichkeiten die praktisch umgesetzten vorzustellen.

#### Die Theorie

**Überführung in eine andere Darstellung** Es gibt zwei Möglichkeiten die Testfall-Spezifikationssprache in eine andere Darstellung (z.B. UML2 Aktivitätsdiagramme) zu überführen. Zuerst die wünschenswerte mit seinen Vor- und Nachteilen.

Die elegantere Möglichkeit wäre die Transformation innerhalb Eclipse durchzuführen um den selben Sachverhalt aus unterschiedlichen Sichten zu betrachten (vgl. Separation of concerns in [Dij82]). Die grundlegende Idee ist: Innerhalb von Eclipse wird durch einen Reiterwechsel eine Transformation angetriggert, womit die aktuelle Sicht in die andere überführt werden kann. Da z.B. bei den UML2 Aktivitätsdiagrammen wie im vorherigen Kapitel beide Sichten 1:1 gemappt sind, wäre eine konsistente Modellierung in textueller und graphisher Form möglich. So können Testdesigner, die textuell besser ihre Gedanken verfassen und Testdesigner, die die graphische Sicht mehr anspricht, besser kommunizieren.

Die zweite Möglichkeit ist, ausserhalb von Eclipse zu arbeiten: Ein Programm übernimmt dabei die Transformation in eine andere Darstellung. Konkret soll die Transformation in die UML2 Aktivitätsdiagramm Darstellung behandelt werden.

Für die Umwandlung würde man die Testfall-Spezifikationssprache in das, von der OMG[13] definierte, XMI[14] Format überführen. XMI (XML Metadata Interchange) wurde definiert, um den Austausch von UML Diagrammen über Herstellertools hinweg zu standardisieren und definiert ein XML Schema.

**Codegenerierung aus der Testfallbeschreibung** Ausser den beiden Umwandlungsmöglichkeiten der Sprache in eine andere Darstellung, gibt es noch die Codegenerierung. Im Unterschied zur Umwandlung der Darstellung fliessen bei der Codegenerierung zusätzliche Informationen ein bzw. die Information werden transformiert.

[13] http://www.omg.org
[14] http://www.omg.org/technology/documents/modeling_spec_catalog.htm

Die Anwendung wird mit folgendem Beispiel deutlich: Die Testfall-Spezifikationssprache wird in ein Testskript transformiert, der durch den Tester (vgl. der Rollen 2.5.3, 32) - falls notwendig - angepasst wird und der so (halb)automatisch erstellte Testkript wird von dem Tester durchgeführt.

In der Theorie würde es ausreichen, wenn alle notwendigen Informationen für eine automatische Codegenerierung vorliegen um somit die Testskripte automatisch ausführen zu können. In der Praxis muss der Tester dennoch manuell bearbeiten bzw. erstellen, da die Testskripte für jedes Testsystem unterschiedlich kalibriert werden müssen.

### Die Praxis

Die Transformation wird mit der Templatesprache Xpand programmiert.

Die Transformation mittels Xpand soll an einem einfachen Beispiel deutlich gemacht werden. Dazu wird die erste Hälfte der Testfallbeschreibung betrachtet, d.h die Definition der Aktionen, Reaktionen, Ereignisse und Zustände soll in ein Visual Basic Testskript transformiert werden.

Listing 7.2 zeigt die Definitionen, wie sie in der Testfall-Spezifikationssprache erstellt wurde.

```
1 Aktionen Anfang
   Aktion A1 "Fensterheber_hoch".
3  Aktion A2 "Blinker_an".
   Aktion A3 "UBatt_auf_X_Volt".
5  Aktion A4 "UBatt_auf_8_Volt".
  Ende Aktionen.
7
  Reaktionen Anfang
9  Reaktion R1 "Fenster_bewegt_sich_nach_oben".
   Reaktion R2 "feste_zeit".
11 Ende Reaktionen.
```

**Listing 7.2:** Eingabe für die Codegenerierung

Listing 7.3 zeigt das Ergebnis: Die Transformation in die Testskripte.

```
1 '#Language "WWB-COM"

3 Option Explicit

5 ' Library:          Aktionen
  '
7 ' Description:  Aktionen für die Testbeschreibung
```

```
'
' History:
'
'  Funktion für die Aktion Blinker_an
Sub Blinker_an()
  'Bitte Code vervollständigen!
End Sub

'  Funktion für die Aktion Fensterheber_hoch
Sub Fensterheber_hoch()
  'Bitte Code vervollständigen!
End Sub

'  Funktion für die Aktion UBatt_auf_8_Volt
Sub UBatt_auf_8_Volt()
  'Bitte Code vervollständigen!
End Sub

'  Funktion für die Aktion UBatt_auf_X_Volt
Sub UBatt_auf_X_Volt()
  'Bitte Code vervollständigen!
End Sub
...
```

**Listing 7.3:** Erzeugter Testkript von dem Templateskript (Ausschnitt)

Für die Umwandlung wird das Templateskript in Listing 7.4 verwendet.

```
«IMPORT tbs»;

«EXTENSION templates::Extensions»

«DEFINE main FOR Model-»
«FILE "Aktionslib.txt"-»
'#Language "WWB-COM"

Option Explicit

' Library:          Aktionen
'
' Description:  Aktionen für die Testbeschreibung
'
' History:
'
«FOREACH this.aktion.typeSelect(Aktion).sortBy(e|e.name) AS e-»
'  Funktion für die Aktion «e.name»
Sub «e.name»«"".javaPrintSkript()»()
  'Bitte Code vervollständigen!
End Sub

«ENDFOREACH-»
«ENDFILE-»
```

```
«ENDDEFINE»
```

**Listing 7.4:** Das Templateskript für die Transformation in einen Testskript

In Xpand ist es erlaubt, Javacodeteile auszuführen. Diese müssen allerdings (sinnigerweise) Strings zurück liefern. Zeile 19 benutzt eine extern definierte Java Methode, die aus dem Templateskript aufgerufen wird.

In Listing 7.3, dem Ergebnis, sind Kommentare zu sehen, dass die Funktion vervollständigt werden soll. Das bedeutet, die in diesem Beispiel erstellten Skripte müssen manuell angepasst werden bevor sie lauffähig sind (halbautomatische Testskripterstellung).

## 7.3 Arbeiten mit dem Editor

In diesem Abschnitt soll eine Referenzangabe über die Nutzung des Editors gemacht werden[15]. Sie ist in zwei Teile unterteilt: Der erste Teil beantwortet die Frage „Wie gehe ich bei der Erstellung eines Testfalls mit der Testfall-Spezifikationssprache vor?“. Der zweite Teil beantwortet die Frage „Welche Punkte muss ich bei der Strukturierung der Dokumente im Editor beachten?“

### 7.3.1 Workflow zur Erstellung von Testfallbeschreibungen

Abbildung 7.4 zeigt, wie eine Testfallbeschreibung erstellt werden sollte.

Zuerst werden die Aktionen, Reaktionen und falls bekannt die Ereignisse und Zustände definiert. Diese möglichen Ein- und Ausgaben vom Testobjekt können von einer Gruppe von Leuten[16] durchgesehen werden. Hier wird schon inhaltlich, bevor der Testfall entworfen wird, die möglichen Angaben überprüft und ein gemeinsames Verständnis geschaffen. Anschliessend erfolgt die Definition der Testfälle durch den Testdesigner. Bei der Erstellung der Testfälle werden notwendige zusätzliche Aktionen, Reaktionen, Ereignisse oder Zustände definiert. Nach diesem Schritt erfolgt ebenfalls ein Review des Inhalts. Diesmal werden die Testfälle durchgesehen, genauso wie eventuell neu hinzugekommene Definitionen der Sprachelemente. Die optio-

[15] Eine Referenzimplementierung ist eine Implementierung einer Spezifikation, um zum einen deren praktischen Nutzen zu liefern und gleichzeitig eine Empfehlung, ein Anhaltspunkt und/oder eine Vorgabe zu machen, wie die Ersteller der Spezifikation sich die Implementierung vorgestellt haben.

[16] Hier ist absichtlich von einer „Gruppe von Leuten“ die Rede ohne klar zu definieren, welche Rollen nach 2.5.3 vorhanden sein müssen.

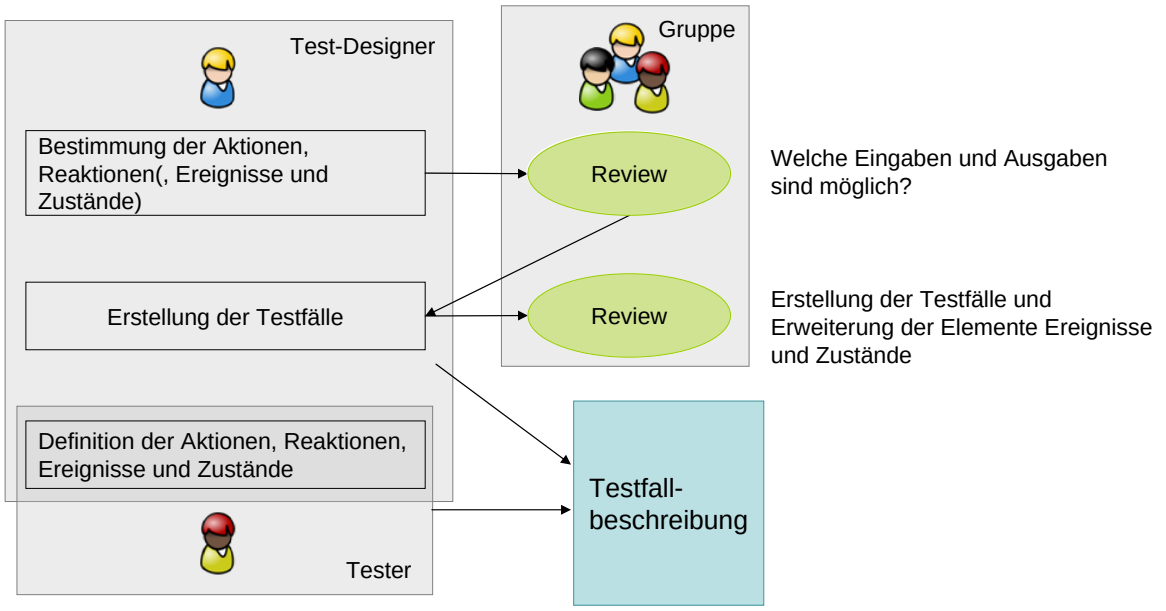



**Abbildung 7.4:** Workflow zur Beschreibung der Testfälle mittels dem Editor

nale Definition der Aktionen, Reaktionen, Ereignisse und Zustände für eine automatische Testkripterstellung kann vom Tester erfolgen oder durch den Testdesigner. Das Ergebnis ist eine Testfallbeschreibung.

### 7.3.2 Strukturierung der Testfallbeschreibung im Editor

Abbildung 7.5 zeigt die Grundlegende Struktur eines jeden Testfalldokuments[17].
Sie besteht - wie im vorherigen Kapitel 6 vorgestellt - aus drei Abschnitten: Einem Abschnitt über die Bestimmung möglicher Aktionen, Reaktionen, Ereignisse und Zustände. Einem Abschnitt aus den Testfällen und einem Abschnitt über die Definition der vier Sprachelemente.
In jedem Testfalldokument sind die drei Abschnitte optional, da eine Möglichkeit für einen Verweis vorhanden ist. Das bedeutet, dass in einem Tesfalldokument auf ein anderes Testfalldokument verwiesen werden kann bzw. der Inhalt dieser importiert werden kann (vgl. Anhang F über die vollständige Syntax der Sprache für den Editor). Wird der Inhalt eines Dokuments importiert, so stehen alle seine Elemente zur Verfügung (siehe Abbildung 7.5).

Abbildung 7.6 zeigt die Abhängigkeit der drei Abschnitte voneinander. Demzufolge können die bestimmten Aktionen, Reaktionen, Ereignisse und Zustände einzeln

[17]Ein Testfalldokument ist ein Dokument in dem Editor, der auf der rechten Seite - der Navigation - zu finden ist.

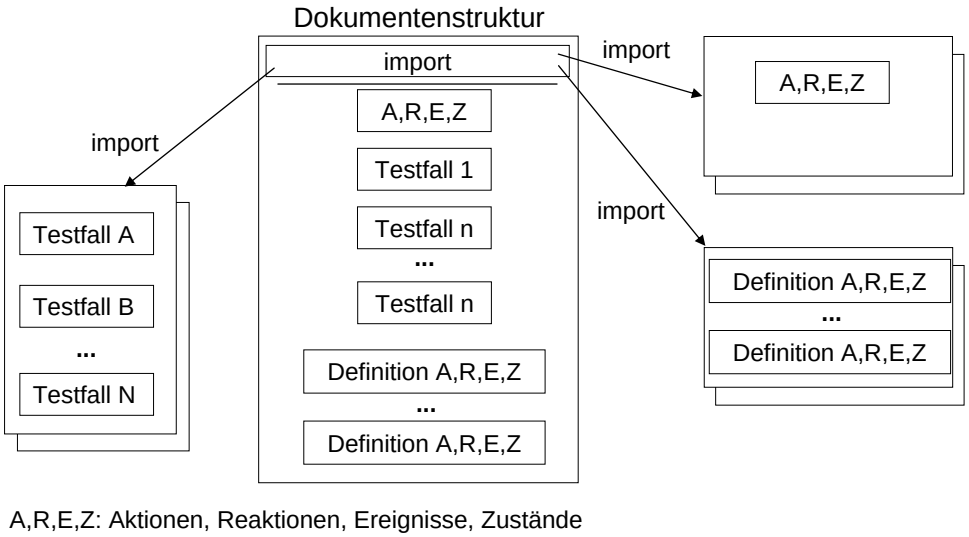



**Abbildung 7.5:** Verweismöglichkeiten des Testfalldokuments auf andere Testfalldokumente

in einem Testfalldokument ohne Verweis (Import) stehen, da ihre Abhängigkeit zu der Spezifikation rein inhaltlicher Natur ist. Jedoch können Testfälle ohne diese nicht existieren, d.h. in einem Testfalldokument, wo Testfälle erstellt werden müssen, muss ein Import der vier Sprachelemente gemacht werden. Anderenfalls ist eine Erstellung der Testfälle unter Benutzung der vier Sprachelemente nicht möglich. Das gleiche gilt für die Definition der Sprachelemente. Auch hier ist eine Abhängigkeit vorhanden, so dass vor der Definition ein Import erfolgen muss. Zu sehen ist weiterhin, dass zwischen den Testfällen und der Definition der Sprachelemente keine Relation vorhanden ist, d.h. sie sind unabhängig voneinander[18].

Wie erwähnt wird aus Abbildung 7.5 deutlich, dass jedes der drei Abschnitte aus anderen Dokumenten importiert werden kann. Sinnvoll ist hier, die drei Abschnitte in Unterdokumente aufzuteilen und auf diese bei Bedarf zu verweisen.

## 7.4 Zusammenfassung

Der Mehrwert der Testfall-Spezifikationssprache wird erst durch die Eingabeunterstützung durch den Editor deutlich: Fehler bei der Eingabe sind nicht möglich bzw. werden durch den Editor gekennzeichnet (Syntaktische Überprüfung) und eine einheitliche Darstellung der Testfälle ist gewährleistet.

[18]Es existiert zwar eine indirekte Abhängigkeit über die Bestimmung der Sprachelemente, jedoch kann ein Testfall ohne die Definitionen weitergegeben werden. Umgekehrt genauso.

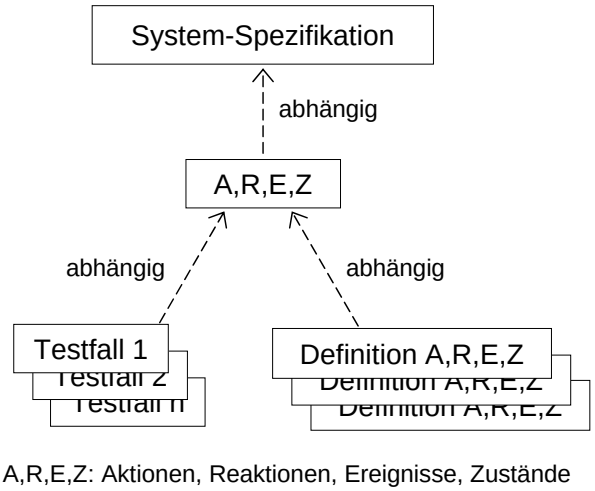



**Abbildung 7.6:** Abhängigkeit der drei Abschnitte in einem Testfalldokument

Der Editor kann die unterschiedlichen Projekte eines Testdesigners verwalten. Abstriche gibt es bei der Unterstützung des Workflows. Dies wird momentan durch externe Vorgaben (Dokumente) ermöglicht (angelehnt an den Workflow der Referenznutzung auf Abschnitt 7.3) und ist nicht Bestandteil des Editors.

Abbildung 7.7 zeigt die editorunterstützte Eingabe. Zu sehen ist, dass der Testdesigner die Deklaration „Unterstützung_HV_Batterie_anfordern_X“ sich aufzeigen lassen will. Diese kann in dem gleichen Testfalldokument sein oder - wie in diesem Beispiel - in einem extern verwiesenen Dokument. In diesem Fall würde der Editor das Dokument öffnen und an die Stelle der Deklaration springen.

Der Editor ermöglicht eine kontrollierte Eingabe, d.h. während der Eingabe mit dem Editor wird überwacht, ob die eingegebene Testfallbeschreibung der definierten Syntax entspricht. Weicht diese von der Form ab, wird eine Warnung ausgegeben. So entspricht jede Eingabe der Syntax und ist unabhängig von dem Testdesigner.

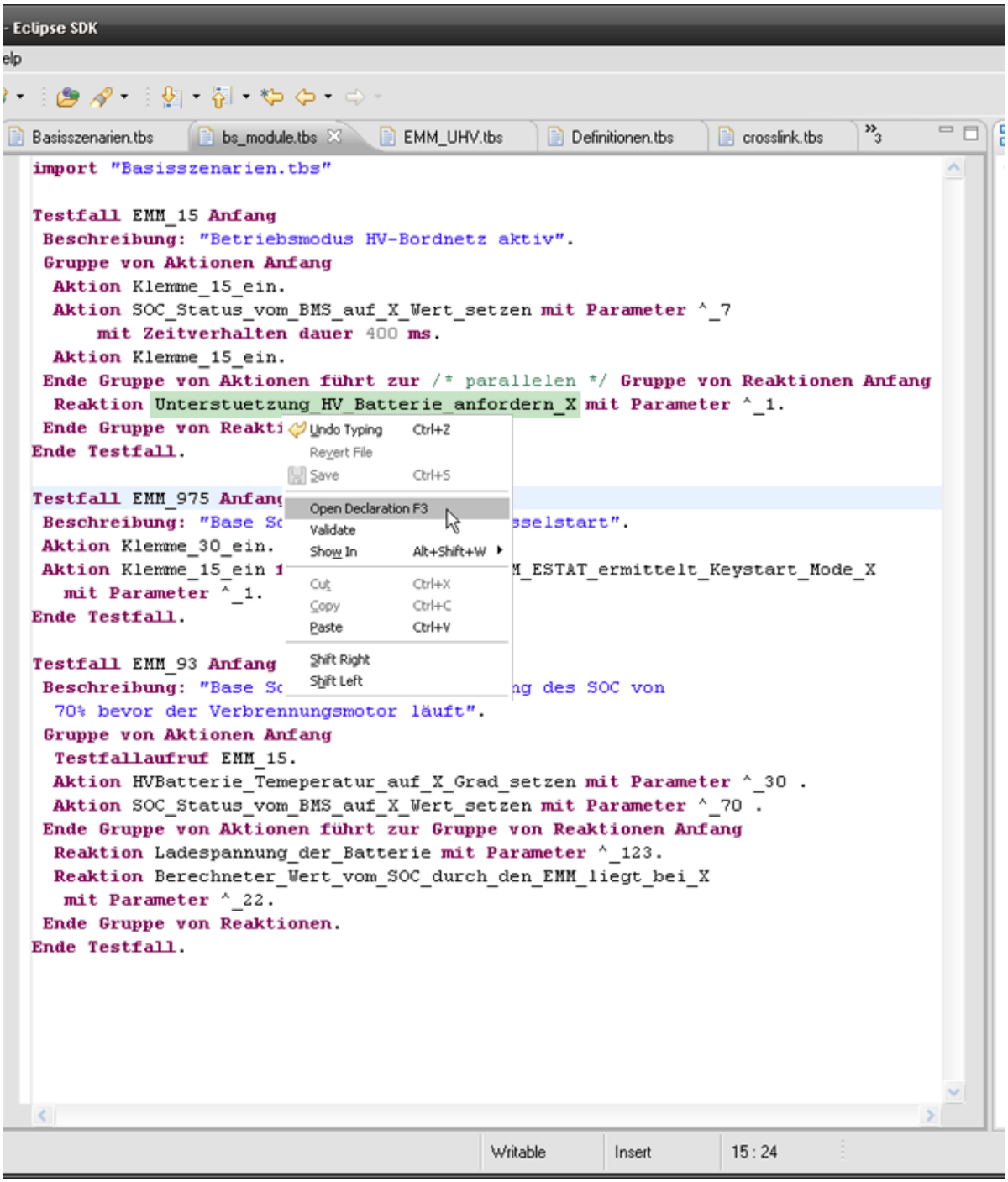



**Abbildung 7.7:** Editorunterstützte Eingabe der Testfallbeschreibung

# Kapitel 8

# Beschreibung von funktionsorientierten Steuergerätetests

In diesem Kapitel wird vorgestellt, wie mit der Testfall-Spezifikationssprache Testfälle spezifiziert werden.

## 8.1 Beschreibung von Testfällen

### 8.1.1 Formulierung der Funktionsvorschrift in natürlicher Sprache

Für die Beschreibung der Funktionsvorschrift wird das Beispiel aus Kapitel 5, S. 123 verwendet. Hier wird u.a. die natürliche Sprache zur Beschreibung von Testfällen benutzt und an einem Beispiel gezeigt wie das aussehen würde. Dieses Beispiel wird hier nochmal herangezogen:

```
1
  Vorbedingungen
3 Als Vorbedingung für den Test, muss das Testsystem in einen
  vordefinierten Zustand gebracht werden. Dazu wird zunächst
5 die Stromversorgung für das SUT (Steuergeräteverbund)
  eingeschaltet. Danach wartet man 5 Sekunden auf die Initialisierung
7 der Steuergeräte, bevor man die nächsten Schritte einleitet.
  Am Schluss wird der Zündschlüssel noch in die entsprechende
9 Stellung gebracht, da das Richtungsblinken wie mit der Realität
  konform nur bei eingeschalteter Zündung geprüft werden kann.
11
  Aktionen
13 Nach den Initialisierungsschritten des Testsystems wird nun
```

```
als Aktion der (simulierte) Blinkschalter in die Stellung
Linksblinken gebracht.

Erwartete Ergebnisse
Zunächst wird in diesem Test Überprüft, ob die Betätigung des
Blinkschalters in Form eines CAN C Signals innerhalb von 20ms
vom MRSM erfasst und anschließend durch ZGW auf CAN B innerhalb
von 20ms geleitet wird. Danach wird betrachtet, ob das Signal
in weniger als 15ms vom SAM_H empfangen wird. Zudem wird zeitgleich
kontrolliert, dass die rechten Blinkleuchten während des Tests
inaktiv bleiben. Das SAM_H soll nach Erhalt des Signals innerhalb
von 30ms die hinten linken Blinkleuchten ansteuern. Gleichzeitig
soll das Signal noch innerhalb von 15ms an SAM_V, TSG_VL sowie AAG
weitergeleitet werden. Diese steuern daraufhin innerhalb von 30ms
die an ihnen angeschlossenen Blinkleuchten an. Zusätzlich soll vor
der Ansteuerung der Anhängerblinkleuchten noch das Vorhandensein
eines Anhängers überprüfen. In diesem Testfall ist ein Anhänger
in der simulierten Umgebung vorhanden.
```

**Listing 8.1:** Beispiel einer Testfallbeschreibung in der natürlichen Sprache

### 8.1.2 Beschreibung mittels der Testfall-Spezifikationssprache

#### 1. Schritt: Aktionen, Reaktionen, Ereignisse und Zustände werden deklariert

Im ersten Schritt werden die vier Gruppen von Elementarschritten deklariert.

```
Aktionen Anfang
 Aktion A1 "Zuendschluessel_in_Position_bringen".
 Aktion A2 "Blinkerschalter_betaetigen".
Ende Aktionen.

Reaktionen Anfang
 Reaktion R1 "MRSM_erhaelt_Nachricht".
 Reaktion R2 "SAM_H_erhaelt_Nachricht".
 Reaktion R3 "SAM_V_erhaelt_Nachricht".
 Reaktion R4 "TSG_VL_erhaelt_Nachricht".
 Reaktion R5 "AAG_erhaelt_Nachricht".
 Reaktion R6 "Rechte_Blinker_bleiben_inaktiv".
Ende Reaktionen.

Zustaende Anfang
 Zustand Z1 "Stromversorgung_fuer_SUT_ist_eingeschaltet".
Ende Zustaende.
```

**Listing 8.2:** Aktionen, Reaktionen, Ereignisse und Zustände werden deklariert

Wie zu sehen, sind die einzelnen Elemente als Freitext vorhanden. Hier kann statt „Steuergerät erhält Nachricht“ auch „Steuergerät empfängt Nachricht von Steuergerät 2“ verwendet werden. Der Text ist hier je nach ermessen unterschiedlich zu formulieren.

### 2. Schritt: Der Testfall wird erstellt

Der Testfall kann nun mittels der vorgegebenen Grundelemente spezifiziert werden.

```
Testfall TF1 Anfang
 Vorbedingung Anfang
  Zustand Stromversorgung_fuer_SUT_ist_eingeschaltet.
 Ende Vorbedingung.

 Warte 5 s.

 Aktion Zuendschluessel_in_Position_bringen.
 Aktion Blinkerschalter_betaetigen führt zur
  Gruppe von Reaktionen Parallel Anfang
   Reaktion Rechte_Blinker_bleiben_inaktiv.
   Gruppe von Reaktionen Anfang
    Reaktion MRSM_erhaelt_Nachricht mit Zeitverhalten frühestens 19 ↩
       ms und spätestens 21 ms.
    Gruppe von Reaktionen Parallel mit Zeitverhalten frühestens 29 ms↩
        und spätestens 31 ms Anfang
     Reaktion TSG_VL_erhaelt_Nachricht.
     Reaktion SAM_V_erhaelt_Nachricht.
     Reaktion AAG_erhaelt_Nachricht.
    Ende Gruppe von Reaktionen.
   Ende Gruppe von Reaktionen.
  Ende Gruppe von Reaktionen.

Ende Testfall.
```

**Listing 8.3:** Beschreibung des Blinkertests in der Testfall-Spezifikationssprache

### 3. Schritt: Optionale Definition der vier Grundelemente

Der Testfall ist spezifiziert. Optional kann nun die Spezifikation auf Signalebene erfolgen. Hierfür müssen die aus dem ersten Schritt deklarierten Aktionen, Reaktionen, Ereignisse und Zustände definiert werden.

```
Definition Aktionen Anfang
 Def Aktion Zuendschluessel_in_Position_bringen Anfang
  Signalsetzung: "Klemme_15" = "3".
```

```
 Ende Def Aktion.

 Def Aktion Blinkerschalter_betaetigen Anfang
  Signalsetzung: "CAN_C.MRSM.SCHALTER.Blinkerschalter_POS" = "2".
 Ende Def Aktion.
Ende Definition Aktionen.

Definition Reaktionen Anfang
 Def Reaktion AAG_erhaelt_Nachricht Anfang
  Signalsetzung: "CAN_B.AAG.SCHALTER.Blinkerschalter_POS" = "0.333".
 Ende Def Reaktion.
Ende Definition Reaktionen.
```

**Listing 8.4:** Definition einiger Aktionen und Reaktionen für den Blinkertest

## 8.2 Der Einsatz in einem Pilot-Projekt

Der Einsatz in Unternehmens-Projekten zeigt die Vorteile der Sprache auf, aber auch dessen Herausforderungen unter realen Bedingungen.

Der erste Einsatz erfolgt in einem Pilotprojekt. Dafür wird üblicherweise die Rahmenbedingungen identifiziert und Maßnahmen definiert um die Sprache einsetzen zu können.

Eine Herausforderung war, die Sprache in die vorhandene Werkzeuglandschaft einzugliedern. DOORS von Telelogic wird bei den Projekten nicht nur für die Aufzeichnung und Verwaltung von den Anforderungen und System-Spezifikationen hergenommen, das Werkzeug wird auch für die Verwaltung der Testfallspezifikationen eingesetzt.

Für die erste Pilotierung war es notwendig, die Kommunikation mit DOORS zu ermöglichen. So entstand ein Übersetzer für den Transport der Testfälle zwischen DOORS und den in Kapitel 7, S. 219 erstelltem Werkzeug. Dies erfolgte Aufgrund der formalen Syntax der Sprache und der vorgegebenen festen Strukturen reibungslos.
Durch den Übersetzter der DOORS Testfall-Strukturen in die Testfall-Spezifikationssprache standen auf Anhieb alle vorhanden DOORS Testfälle in der Struktur der Testfall-Spezifikationssprache zur Verfügung.

Der umgekehrte Weg, die Testfälle aus der eigenen Sprache in das DOORS zu transportieren benötigte umfangreiche Maßnahmen. Hier ist der Übersetzer ähnlich der aufgebaut wie im vorherigen Fall, jedoch sind im Moment politische und organisatorische Herausforderungen die im DOORS vorhandenen Erweiterungen benutzen zu

dürfen vorhanden.

Trotzdem war für das Pilotprojekt die ersten Grundlage gelegt, sodass ein intensiver Einsatz der Sprache und des Werkzeugs möglich sind. So konnten die Test-Designer ihre Testfälle werkzeugunterstützt eingeben. Die erste Evaluierung hat gezeigt, dass die bisherigen Testfallbeschreibungen in DOORS ebenfalls in der Testfall-Spezifikationssprache erstellt werden können. Die Eingabe erfolgt etwas schneller, was auf das Werkzeug zurückzuführen ist.

Bei den Rücksprachen mit den Test-Designern und Experten im Unternehmen kamen vor allem Aspekte bezüglich der Definitionen von den Aktionen, Reaktionen, Ereignissen und Zuständen auf. Diese sollten in dem Werkzeug, welches als Machbarkeitsstudie aus dieser Arbeit entstanden ist, besser verwaltet werden können. Vor allem dann, wenn die Anzahl der zu überprüfenden Anforderungen ständig wächst.

So bedarf vor allem das Werkzeug eine Anpassung in Richtung Verwaltung der Definitionen und Beschreibung der einzelnen Aktionen, Reaktionen, Ereignisse und Zustände (wie in Kapitel 6, S. 173 dargestellt.).

Zukünftige Arbeiten bei der Pilotierung richtet sich auf die Transformation der Kundenspezifischen Testfall-Strukturen auf die Testfall-Spezifikationssprache, sowie eine automatisierte Ableitung der Testskripte aus den Testfall-Spezifikationen.

## 8.3 Zusammenfassung und Bewertung

Die Testfall-Spezifikationssprache erleichtert vor allem die Arbeit des Test-Designers. Dieser hat durch die einzelnen Schritte eine kleine Methodik bei der Erstellung der Testfälle. So kann er aus der Spezifikation Aktionen, Reaktionen, Ereignisse und Zustände ermitteln, um anschliessend diese zu einem Testfall zusammenzufassen.

Die ermittelten Grundelemente können in Gruppen validiert und ein gemeinsamen Verständnis geschaffen werden, was gemeint ist und wie man am Besten diese formuliert.

Die Definition der vier Grundelemente ermöglicht bei der Testfallimplementierung eine weitgehende Automatisierung. Stehen diese Informationen zu Beginn der Testfall-Spezifikation nicht zur Verfügung, ist die Testfallbeschreibung immer noch gültig und kann jederzeit ergänzt werden.

Werden die Abläufe kompliziert, verliert sich die Beschreibung zum Teil in den Schlüsselelementen. Hier hilft eine werkzeugunterstützte Eingabe und Verarbeitung. Nach

einiger Zeit ist jedoch aufgrund der immer wiederkehrenden Strukturen selbst für das menschliche Auge die Verständlichkeit gewährleistet. Die natürliche Sprache stellt weiterhin sicher, dass die einzelnen Schlüsselwörter und ihre Kombination immer verständlich bleiben.

Das Eingabewerkzeug muss über den Stand einer Machbarkeitsstudie hinaus und muss um Funktionen für die Verwaltung der Sprachelemente erweitert werden.

Deutlich ist an der Testfallbeschreibung 8.4 zu erkennen, dass die einzelnen Elemente (Aktionen und Reaktionen) unterschiedlich formuliert werden können und dieser Freiheitsgrad ebenfalls eingeschränkt werden muss.

Abbildung 8.1 zeigt die Abdeckung der Bewertungskriterien durch die Testfall-Spezifikationssprache.

| | | Gewichtung | TFSpecSpr |
|---|---|---|---|
| Anforderungen aus dem Inhalt und Umfang eines Testfalls | | | |
| | | | |
| | Inhalt und Umfang eines Testfalls (Testfallbeschr.) | 4 | 24 |
| | Unterstützung Testfallparadigmen | 4 | 1 |
| | | | 100 |
| Anforderungen an die Notation | | | |
| | Darstellungsform | 1 | textuell, graphisch |
| | Formalismus | 2 | formal |
| | | | 4 |
| Anforderungen an die Beschreibungstechnik | | | |
| | Werkzeugunterstützte Eingabe | 1 | ja |
| | | | 1 |
| Anforderungen aus dem Prozess und der Rollen | | | |
| | Sprachelemente für Verlinkung zu Testfällen bzw. | 1 | ja |
| | Sprachelemente für Platzhalter | 1 | ja |
| | Sprachelemente für Templates | 1 | ja |
| | Sprachelemente für Reaktionsauswertung | 1 | ja |
| | Sprachelemente für Hierarchisierung | 2 | ja |
| | Darstellung von Metainformationen | 2 | ja |
| | Workflow zur Ableitung von Testfallimplementierungen | 1 | ja |
| | | | 9 |
| | | | 114 |

**Abbildung 8.1:** Abdeckung der Bewertungskriterien durch die Testfall-Spezifikationssprache

# Kapitel 9

# Zusammenfassung und Ausblick

## Zusammenfassung

Ziel war es eine Sprache für den Testdesigner zur Verfügung zu stellen, mit dem er Testfälle beschreiben kann, die er mittels seines Wissens und seiner Erfahrung ableitet. Die Sprache sollte dabei lesbar, verständlich und eindeutig sein, um wieder verwendbare Testfallbeschreibungen erstellen zu können.

Die Arbeit ist in zwei Teile geteilt: Der erste Teil beschäftigte sich mit der Analyse der Anwendungsdomäne (Kapitel 2, 3 und 4). Die Steuergeräte sind eingebettete, reaktive Systeme mit Echtzeiteigenschaften und die Erfüllung einer Funktionalität über mehrere Steuergeräte spielt eine immer größere Rolle. Durch die Analyse werden Elemente für die Beschreibung von funktionalen Steuergerätetests im Automobilen Umfeld herausgearbeitet. Weiterhin wird festgestellt, dass sich vorhandene Grundkonzepte (Automaten, XML, UML Diagramme, ...) als Grundlage hernehmen lassen und durch Elemente für funktionale Steuergerätetests erweitert werden können, so dass ein - an das Anwendungsgebiet angepasstes - Beschreibungsmittel entsteht.

Die Anforderungen werden durch die Anwendungsfälle ermittelt. Zuerst werden die Anwendungsfälle für die Testfallbeschreibung identifiziert. Dabei werden Ermittlungstechniken wie Interviews oder Brainstorming angewandt, um die Anwendungsfälle der Stakeholder[1] festzustellen. Durch zusätzliche Tätigkeiten im Umfeld von Gremienarbeiten werden die Anwendungsfälle auch aus anderen Unternehmen ermittelt. So sind die Anwendungsfälle für die Testfallbeschreibung gut umrissen. Aus

[1] Personen, die sich mit der Testfallbeschreibung auseinandersetzen müssen oder Rollen, die durch ihre Aufgabentätigkeit auf den Inhalt der Testfallbeschreibung angewiesen sind.

diesen Anwendungsfällen werden im zweiten Schritt die Anforderungen abgeleitet. Bei der Analyse der Anwendungsfälle und Ableitung der Anforderungen wurde festgestellt, dass einige der Anwendungsfälle - wie z.B. wieder verwendbare Testfallbeschreibungen zu erstellen - nicht durch die Notation aufgegriffen werden können. Dadurch wurden die Anforderungen an die Testfallbeschreibung konkret.

Im Stand der Technik wurde aufgezeigt, dass keins der vorhandenen Testfallbeschreibungsmöglichkeiten auf der Ebene der Spezifikation ansetzen, d.h. sie vermitteln nicht den Gedanken was man mit dem Testfall erreichen will, sondern bieten Möglichkeiten zur Beschreibung auf der Umsetzungsebene.

Der zweite Teil basiert auf den Ergebnissen des ersten Teils. Hier wird eine Sprache entworfen, die an der natürlichen Sprache angelehnt ist und von der Syntax her nur Sprachelemente besitzt, die für die Beschreibung notwendig sind. So tauchen keine Elemente auf wie sie in Programmiersprachen vorkommen (Klammer, Semikolon,..). Besonders bei der Bewertung gegen die Anforderungen fällt auf, dass keine Elemente der Testfallimplementierungsebene zu finden sind. Das kann nur durch eine komplett neu definierte Syntax erreicht werden (Domänenspezifische Sprache). Die Sprachelemente sind angelehnt auf die Erkenntnisse der Analyse und den Sprachelementen aus den Mitteln im Stand der Technik, insbesondere auf dem Aufbau der Testfallbeschreibung nach ATML.

Durch die Testfall-Spezifikationssprache wurden mehrere Ziele erreicht: Die Sprache ist formal genug gehalten um automatisiert diese in andere Abbildungen zu überführen. Sie besitzt jedoch genug Freiraum um beliebige Gedankengänge im Rahmen der funktionalen Steuergerätetests festzuhalten.

Mit der Abbildung der Testfall-Spezifikationssprache in eine graphische Sicht (Untermenge der UML2 Aktivitätsdiagramme) ist mit einer werkzeugunterstützten Eingabe ein Arbeiten auf sowohl der textuellen als auch der graphischen Sicht möglich.

Weiterhin bietet die Sprache Möglichkeiten der Transformation. Die Testfallbeschreibung kann in die vorhandenen proprietären Formate der Unternehmen überführt werden. Umgekehrt können die proprietären Formate in die Sprache der Testfall-Spezifikationssprache umgewandelt werden. Für einen Dienstleister, der in unterschiedlichen Toollandschaften arbeiten muss, ein nicht zu unterschätzender Vorteil.

## Ausblick

Für lesbare, verständliche und eindeutige Testfallbeschreibungen, wie sie gefordert wurden, sind noch Arbeiten an der Sprache und naheliegenden Bereichen offen.

Die Anwendungsfälle bzw. die Wünsche der Stakeholder und Anforderungen an eine Testfallbeschreibung können weitgehendst nicht allein durch die Notation gedeckt werden, sondern die Herleitung des Testfalls besitzt einen entscheidenden Einfluss bei der Erfüllung der Anwendungsfälle.

Das Engineering eines Testfalls (Testfallerstellung, Testverfahren oder Prüfverfahren genannt) muss bis auf die Formulierung der Einzelschritte nach einem Workflow erfolgen: Von der Erstellung der Struktur eines Testfalls bis zur Benennung der einzelnen Testschritte haben alle Elemente der Beschreibung einen entschiedenen Einfluss wenn verständliche und wieder verwendbare Testfallbeschreibungen erstellt werden sollen. Die Notation ist hier nur eine wichtige Säule bei der Testfallerstellung.

Ein weiterer Schwerpunkt, der betrachtet werden sollte, ist die Herleitung an sich: Was im Moment durch einen mentalen Vorgang (vgl. Anhang B, S. 263) durch den Test-Designer erledigt wird, ist ein unerforschter Bereich. Hier ist gefordert, diese mentalen Vorgänge zu kanalisieren, zu bündeln und ein Vorgehen oder praktische Anleitungen zu ermitteln, womit der Test-Designer einen generischen Ansatz für die Erstellung der Testfälle hat. Angesichts der Komplexität der Steuergeräte, der unterschiedlichen Domänen im Fahrzeug und der Unmenge an möglichen Testfällen (vgl. Kapitel 2, S. 13) ist dies ein schweres Unterfangen.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **OEM** | Original Equipment Manufacturer |
| **E/E** | Elektrik/Elektronik |
| **ECU** | Electronic Control Unit |
| **CAN** | Controller Area Network |
| **LIN** | Local Interconnect Network |
| **HIL** | Hardware-in-the-Loop |
| **MIL** | Model-in-the-loop |
| **SIL** | Software-in-the-loop |
| **SUT** | System Under Test |
| **WLAN** | Wireless Local Area Network |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **CSP** | Communicating Sequential Process |
| **UML** | Unified Modeling Language |
| **DSL** | Domain Specific Language |
| **XML** | Extensible Markup Language |
| **EBNF** | Erweiterte Backus-Naur-Form |

# Anhang A

# IEEE 829: Software Test Documentation

## A.1 Dokumente beim Testprozess

Um eine Eingrenzung und Abgrenzung der Arbeit zu ermöglichen werden die entstehenden Dokumente im Testprozess untersucht. Daraus ergibt sich eine Einordnung der eigenen Arbeit in die vorhandene Dokumentenlandschaft.

Üblicherweise werden Dokumente in einem Unternehmen für die eigenen Bedürfnisse und die angepassten Prozesse ebenfalls angepasst erstellt. Dabei werden unterschiedliche Inhalte, sei es wegen Nichtgebrauch einiger Teile oder um eine Überorganisation zu vermeiden, zu einem Dokument zusammengefasst.

Um bestimmen zu können, welche Dokumenteninhalte wie verlagert werden und welche Inhalte überhaupt existieren sollten, wird der IEEE Standard 829 (IEEE Standard for Software Test Documentation [Iee98]) als Grundlage genommen.

## A.2 Testdokumentation nach IEEE 829

Abbildung A.1 zeigt den Zusammenhang zwischen den im Standard beschriebenen Dokumenten, zu den Dokumenten auf denen aufgebaut wird und den Prozessen, die

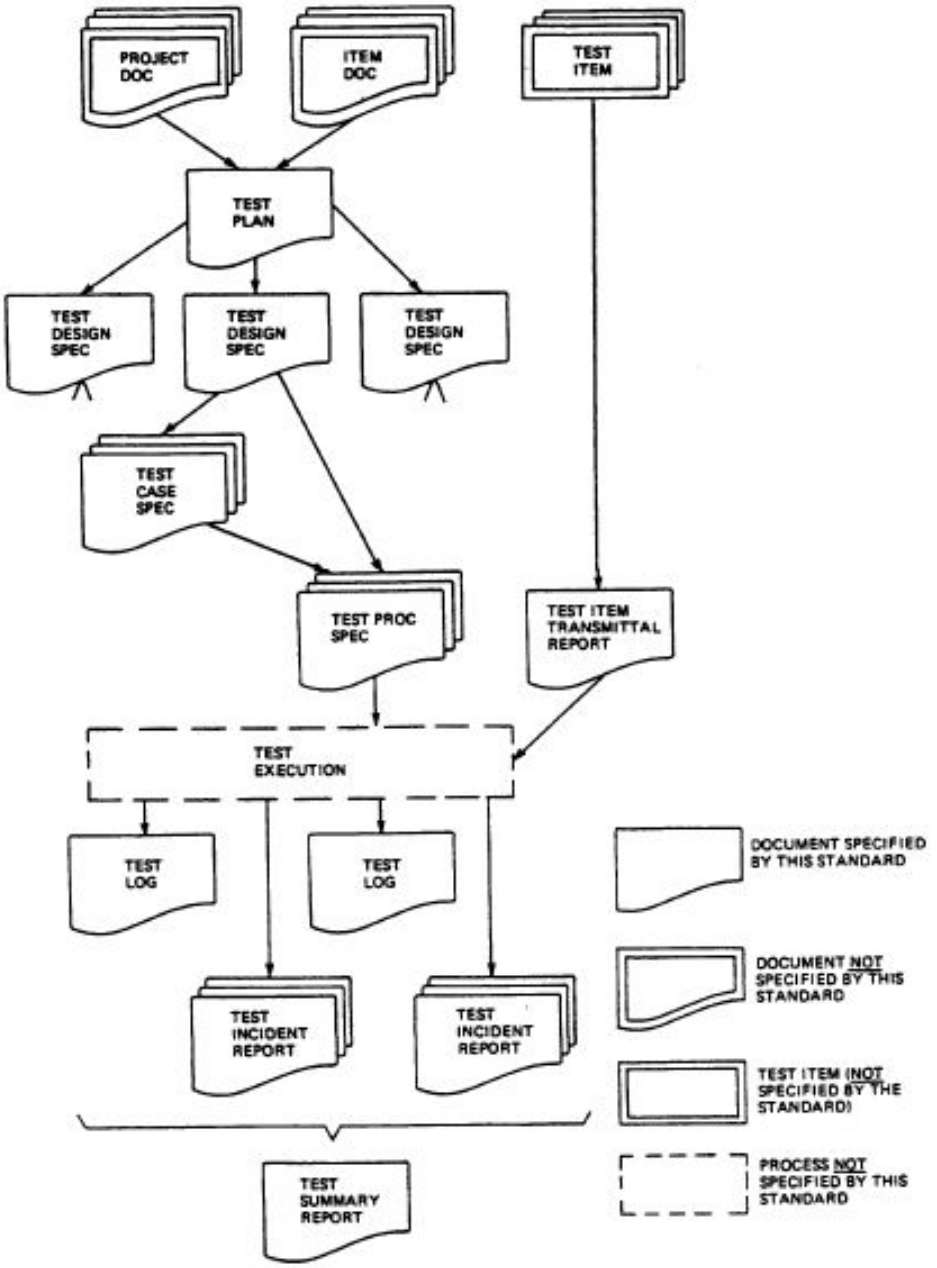



**Abbildung A.1:** Zusammenhang zwischen in dem Standard IEEE 829 definierten Dokumenten, den Prozessen und den benötigten, nichtdefinierten Dokumenten [vgl. [Iee98]]

diese Dokumente verwenden [1].

Der Standard umfasst drei Teilbereiche: Ein Dokument zur Testplanung, drei Dokumente für die Testspezifikation und vier Dokumente für den Testreport.

- Testplan

  Der Testplan schreibt den Scope, die Vorgehensweise, die Ressourcen und den Ablauf der Testaktivitäten vor. Die zu testenden Gegenstände, die zu testenden Eigenschaften, die auszuführenden Testaufgaben, die Verantwortlichen für die Aufgaben und das Risiko werden festgehalten.
- Test design specification

  Die Test design specification verfeinert die Test-Vorgehensweise und identifiziert die abzudeckenden Eigenschaften durch den Entwurf und zugeordneten Tests. Ausserdem sind Testfälle und Test-Prozeduren, sofern sie für das Testen notwendig sind und die Kriterien für ein Pass/Fail festgelegt.
- Test case specification

  In der test case specification werden die zu benutzenden Eingabewerte und die erwarteten Ausgabewerte dokumentiert. In dem Testfall werden Constraints festgehalten, die in der Test-Prozedur auftauchen[2]. Die Testfälle sind vom test design getrennt um die Verwendung der Testfälle in mehreren Test-Designs und anderen Situationen zu ermöglichen.
- Test procedure specification

  In der test procedure specification sind alle notwendigen Schritte für ein funktionierendes System und die Ausführung des Testfalls beinhaltet um die zugeordneten test designs zu verwirklichen. Die Test-Prozedur ist von der test design specification getrennt, da diese mit der Absicht erstellt wird Schritt für Schritt befolgt zu werden ohne in Details einzugehen.
- Test item transmittal report

  Die test item transmittal report beinhaltet zu überführende Test-Items für den Fall, dass unterschiedliche Entwickler und Testgruppen involiert sind oder ein offizieller Start der Testausführung erwünscht ist.
- Test log

  Das test log wird durch eine Testgruppe verwendet um aufzuzeichnen was während der Testausführung passiert ist.
- Test incident report

  Das test incident report beschreibt jedes Ereignis, welches während der Testausführung aufgetreten ist und eine Nachprüfungen erfordert.

[1] Die Original Bennenung der Dokumente aus dem Standard wird beibehalten um durch die eigene Übersetzung keine Interpretation einzubringen

[2] Hier wird die Test-Prozedur als der Vorgang bzw. Ablauf definiert, wo der Testfall verwendet wird

- Test summary report

  Das Test summary report fasst die Testaktivitäten mit einer oder mehreren test design specifications zusammen.

# Anhang B

# Mentales Modell

Mit dem Begriff des mentalen Modells wird die Tatsache festgehalten, dass jeder Mensch eine innere Vorstellung (Modell) über Sachen aufbaut, mit denen er sich beschäftigt.

## B.1 Begriffsdefinition

Es gibt zahlreiche Abhandlungen zum Begriff 'Mentales Modell'. In dieser Arbeit soll folgende Definition gelten, die sich an der Interaktion der Menschen mit Systemen anlehnt: „people's views of the world, of themselves, of their own capabilities, and of the tasks that they asked to perform, or topics they are asked to learn, depend heavily on the conceptualizations that they bring to the task. *In interacting with the environment, with others, and with the artifacts of technology, people form internal, mental models of themselves and on the things with which they are interacting.*"[1] [GS83].

## B.2 Bezug zum Testen

Der Bezug vom mentalen Modell zum Testen wird in einer Reihe von Quellen genannt, die hier zitiert werden sollen.

[1] Kursiver Text bei den Zitaten ist zur Hervorhebung und nicht im Original vorhanden.

„*In principal, any form of software testing can be seen as model based. The tester always forms a mental model of the system under test before engaging in activities such as test case design.*“ [HKO06]

„*Testing is a process in which we create mental models of the environment, the program, human nature, and the tests themselves.* Each model is used either until we accept the behavior as correct or until the model is no longer sufficient for the purpose. Unexpected test results always force a revision of some mental model, and in turn may lead to a revision of whatever is being modeled. The revised model may be more detailed, which is to say more complicated, or more abstract, which is to say simpler. The art of testing consists of creating, selecting, exploring, and revising models. Our ability to go through this process depends on the number of different models we have at hand and their ability to express a program's behavior.“ [Bei90]

„Test engineers use these specification documents to gain an approximate understanding of the intended behavior. *That is to say, they build a mental model of the system.* This mental model is then used to derive test cases for the implementation, or system under test (SUT): input and expected output. Obviously, this approach is implicit, unstructured, not motivated in its details and not reproducible.“ [BJK05]

Ein Testfall besteht demnach aus einer Reihe von Eingabewerten, die diese Modelle 'durchlaufen' und den dazugehörigen Ausgabewerten:

„*While some argue that because of these implicit mental models all testing is necessarily model-based* [Bin99], the idea of model-based testing is to use explicit behavior models to encode the intended behavior. Traces of these models are interpreted as test cases for the implementation: input and expected output. The input part is fed into an implementation (the system under test, or SUT), and the implementation's output is compared to that of the model, as reflected in the output part of the test case.“ [BJK05]

„We understand human nature and its suceptibility to error. This understanding leads us to create three models: a model of the environment, a model of the program, and a model of the expected bugs. From these models we create a set of tests, which are then executed“ [Bei90]

„Our model of the environment includes our beliefs regarding such things as the workings of the computer's instruction set, operating system macros and commands, and what a higher-order language statement will do“ [Bei90]

Das das Expertenwissen zur Testfallherleitung nicht ersetzt werden kann, wird wie folgt beschrieben:

„It is likely that for test case generation, fully automatic abstraction is not possible

but that test engineers must provide the abstraction mechanism with domain and application specific knowledge.“ [BJK05]

# Anhang C

# Testidee bei einigen klassischen Testverfahren

Es gibt Testverfahren, die die innere Struktur berücksichtigen und solche Verfahren, die rein auf das spezifizierte Verhalten aufsetzten. Letztere werden hier näher betrachtet. In [Lig02] ist ein Gesamtüberblick über die Basis-Testverfahren, sowie ihre Vorstellung vorhanden.

Im Folgenden wird die Testidee zu den Testverfahren Äquivalenzklassen, Grenzwertanalysen, zustandsbasiertes Testen und Ursachen-Wirkungs-Graphen und Expertenwissen aufgezeigt.

- Äquivalenzklassen

  Die Idee bei der Äquivalenzklassen ist, dass eine Menge von Werten das gleiche Verhalten bzw. Ausgabe erzeugen. So können diese Wertemengen bzw. -bereiche zu Äquivalenzklassen zusammengefasst werden. Ein Wert repräsentiert bei der Testdurchführung die Klasse in der er eingeordnet wurde.

- Grenzwertanalysen

  Eine spezielle Betrachtung bei den Äquivalenzklassen sind die Grenzwerte, d.h. die Randwerte, bei denen das Verhalten immernoch dem Verhalten der jeweiligen Klasse entsprechen muss. So werden für diese Randwerte Testfälle erstellt.

- Zustandsbasiertes Testen

  Die Erstellung des Testfalls basiert auf einem Zustandsgraphen, der entweder in der Spezifikation vorhanden ist oder explizit modelliert wird. Auf Basis dieser Zustandsgraphen werden Testfälle abgeleitet. Dabei ist der Testfall ein Pfad im Graphen.

- Ursachen-Wirkungs-Graph

  Der Ursachen-Wirkungs-Graph stammt aus dem Gedanken, dass die Spezifikationen in natürlicher Sprache verfasst sind. Um dieser informellen Spezifikation eine formale Struktur zu geben von dem man anschliessend Testdaten ableiten kann, werden im ersten Schritt die Ursachen (oder Stimuli bzw. Aktionen) in der Spezifikation gekennzeichnet. Anschliessend werden die (Aus-)wirkungen kenntlich gemacht. Angefangen von den (Aus-)Wirkungen werden rückfolgend die Ursachen identifiziert und die Ursachen in Beziehung gesetzt. Dabei entsteht ein Ursachen-Wirkungs-Graph aus Elementen wie NOT, AND, OR und exklusive und inklusive Abhängigkeiten.

  Dieser Graph kann in eine Entscheidungstabelle überführt werden und aus dieser Tabelle können Testdaten abgeleitet werden. Der Ursachen-Wirkungs-Graph betrachtet dabei keine zeitlichen Aspekte, sondern dient der rein logischen Verknüpfung der Ursachen auf die Wirkungen.

- Expertenwissen

  Das Expertenwissen umfasst jede Art von Testfallherleitung, die rein auf dem Wissen und der Erfahrung des Experten bzw. desjenigen der die Testfälle ableitet ruht. Ein dokumentiertes Vorgehen ist dabei nicht vorhanden, sondern ist geprägt durch die Raffinesse des Testfallentwicklers. Die Testidee ist hier nicht durch das Testverfahren zu erkennen.

# Anhang D

# Das UML2 Aktivitätsdiagramm

Die UML2 wurde in Kapitel 5 (*Stand der Technik*, S. 123) untersucht und ein Überblick über die dreizehn Diagrammtypen vorgestellt.

In diesem Kapitel wird die Notation der UML2 Aktivitätsdiagramme dargestellt, soweit sie für das Verständnis der Definitionen in Kapitel 6 notwendig sind. Für eine detailierte Darstellung der UML2 Notation wird auf die Spezifikation [Obj09b] verwiesen[1]. Abbilung D.1 zeigt die möglichen Notationselemente der UML2 Aktivitätsdiagramme.

Für die Definition der Testfall-Spezifikationssprache in Kapitel 6, S. 173 wird nur ein Teil der Elemente verwendet. Diese werden im Folgenden für das Verständnis der Diagramme erklärt. Abbilung D.2 fast diese Elemente in einem Aktivitätsdiagramm zusammen.

- Initialknoten und Endknoten

  Der Initialknoten (schwarz gefüllter Kreis) ist der Einstiegsknoten in ein Aktivitätsdiagramm. Aus diesem kann ein Kontrollfluss (Pfeil) hinauslaufen. Der Endknoten (schwarz gefüllter Kreis mit Ring) ist der Ausstiegsknoten aus einem Aktivitätsdiagramm. In diesen kann ein Kontrollfluss (Pfeil) hineinlaufen.
- Kontrollfuss

  Der Kontrollfluss (durchgezogener Pfeil) verbindet zwei Elemente der Aktivitätsdiagramme miteinander und stellt den Kontrollfluss dar.
- Objektfluss

[1] Gegebenenfalls mit [Obj09a] zu verwenden, die die Definition der Grundelemente (Metasprache) beinhaltet, die in [Obj09b] verwendet werden.

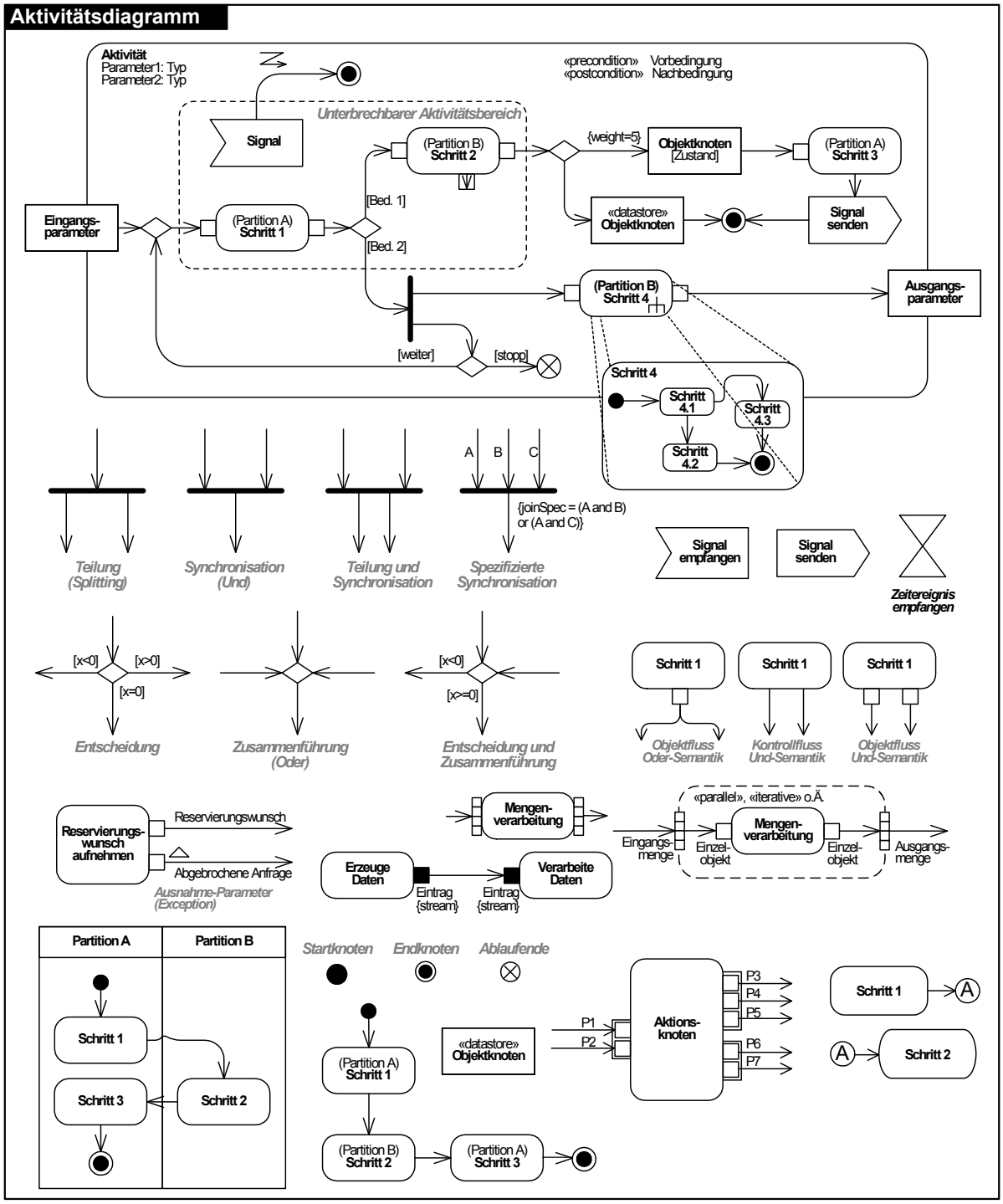



**Abbildung D.1:** Übersicht aller Notationselemente der UML2 Aktivitätsdiagramme [oose.de]

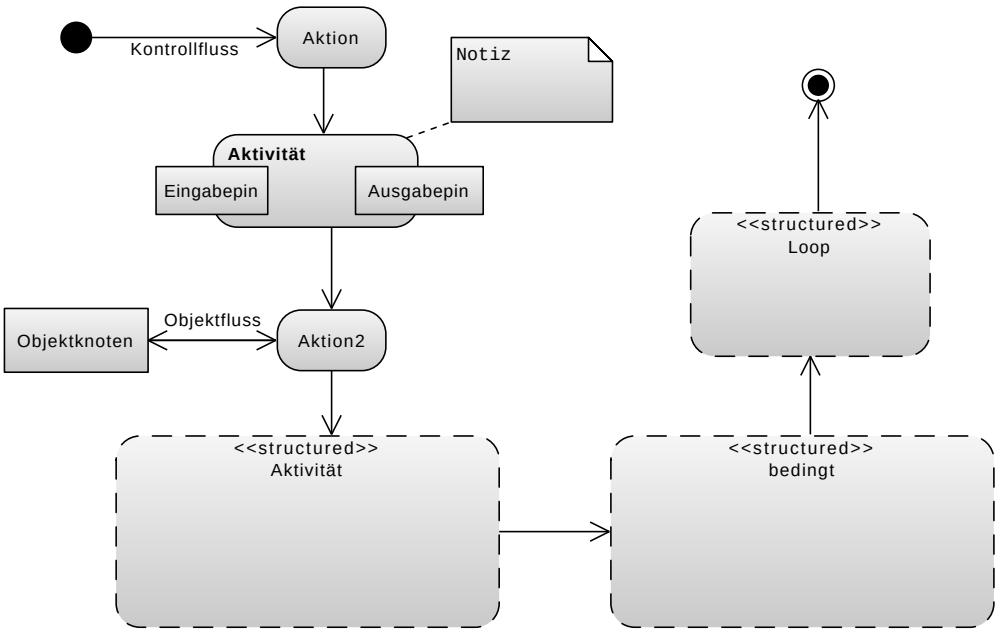



**Abbildung D.2:** Verwendete Elemente in einem Aktivitätsdiagramm

Der Objektfluss hat die gleiche visuelle Darstellung wie der Kontrollfluss. Jedoch stellt sie für Objektknoten den ein- bzw. ausgehenden Fluss dar.

- Aktion

  Eine Aktion konvertiert eine Menge von Eingaben in eine Menge von Ausgaben, wobei beide leere Mengen sein können. Die visuelle Darstellung der Eingaben und Ausgaben können durch Pins an den Aktionen dargestellt werden.

  Die Spezifikation definiert grundlegende Aktionen, wie Operationsaufrufe und Event- und Signalhandling.

- Aktivität

  Das Element Aktivität kann alle Element der Aktivitätsdiagramme beinhalten und definiert somit selbst eine Aktivität. Wie das Aktivitaetsdiagramm selbst kann auch dieser Ein- und Ausgabepins enthalten.

- Notizen

  Jedem Element kann ein Notizelement (comment) angeknüpft werden. Der Kommentar ist Freitext, kann aber auch durch die OMG spezifizierte Contraintsprache wie der Object Constraint Language (OCL) befüllt werden.

- Objektknoten

  Ein Objektknoten (Rechteck) ist ein Knoten für eine Objektdarstellung. Objekte sind Daten, die von Aktionen, Aktivitäten und anderen verarbeitenden Knoten verwendet werden können. Die Verbindung zwischen diesen verarbeitenden Knoten und dem Objektknoten erfolgt durch einen Objektfluss-Pfeil.

- Strukturierte Aktivitätsknoten

  Aus dem Aktivitätselement sind mehrere spezielle Aktivitätselemente definiert. So sind strukturierte Aktivitätsknoten und Aktivitätsknoten für bedingte Verarbeitung und Schleifen vorhanden. Sie sind durch Stereotypen gekennzeichnet.

# Anhang E

# Extended Backus-Naur Form

Die Definition der Syntax erfolgt in der Extended Backus-Naur Form (EBNF). Die EBNF ist eine Metasprache [ISO96], d.h. sie ist eine Sprache um Sprachen zu definieren. Die EBNF ist so aufgebaut, dass sie leicht erweitert werden kann, so dass dadurch viele Derivate entstanden sind. Im Folgenden erfolgt eine kompakte Einführung in die EBNF, die in der ISO/IEC 14977 ( [ISO96]) definiert ist. Für eine detailierte Definition von EBNF wird auf die ISO Norm verwiesen.

Die EBNF Spezifikation selbst ist Top-Down aufgebaut, wo ausgehend vom Begriff die Syntax auf die einzelnen Elemente der Metasprache heruntergebrochen wird. In natürlicher Sprache wird zusätzlich die Semantik definiert. Damit eine formale Definition möglich ist, wird in EBNF einer Reihe von Zeichen eine Semantik hinterlegt. Anschliessend können diese Zeichen benutzt werden um die eigentliche Sprache zu definieren.

Durch die formale Definition der Syntax in EBNF werden die syntaktischen Teile der Sprache (z.B. nicht-terminale Zeichen) aufgezeigt. Weiterhin ist dargestellt, welche Sequenz von Zeichen ein Satz der Sprache sind und wie die syntaktische Strukturen der Sätze in der Sprache aussehen.

Listing E.1 zeigt einen vereinfachten Auszug der Sprachdefinition aus dem Anhang F mit den wichtigsten Zeichenfolgen der EBNF:

```
1 Testfallbeschreibung = {Testfall '.'} ;

3 Testfall = "Testfall" Name  "Anfang"
    {testschritt+=Testschritt '.'}
5  "Ende Testfall" ;

7 Name = {"A" ... "Z"};
```

```
9  Testschritt = Testfallschritt  |  Aktionschritt |  Kontrollstruktur;

11 Aktionschritt = "Aktion" aktion [Parameter] "führt zur" ↩
       Reaktionsschritt;
   ...
```

**Listing E.1:** Beispiel einer Syntaxdefinition in EBNF

Wie EBNF ist die Definition der Sprache ebenfalls Top-Down aufgebaut. So ist der Einstiegspunkt die Regel „Testfallbeschreibung“. Sie ist definiert (zu erkennen an dem Zeichen „=“) durch ein oder mehrere Elemente des Testfalls. Die Syntax-Regel wird abgeschlossen mit dem Endzeichen „;“. In Anführungszeichen umschlossene Zeichen sind Zeichen der definierten Sprache.

Anschliessend wird die Syntax-Regel für „Testfall“ definiert, die ebenfalls mit dem Definitionsymbol „=“ beginnt. Der Kern enthält eine oder mehrere Definitionen eines Testschritts. Auch hier wird die Regel durch das Endzeichen „;“ abgeschlossen.

Der Testschritt wird mit alternativen Syntaxregeln definiert. Das Symbol für alternative Regeln ist | . So werden sukzessive alle nicht-terminale Zeichen bis zu den Terminalen definiert. Die Regel für *Name* zeigt eine Definition, die nur noch aus terminalen Zeichen besteht.

Die wichtigsten Elemente für das Verständnis der Syntaxregeln sind:

- Terminalzeichen der zu definierenden Sprache werden in Anführungszeichen umschlossen.

```
"Testfall"
"Aktion"
```

- Eckige Klammern stellen optionale Angaben dar.

```
[Optionale Aussagen]
```

- Die in geschweifte Klammern angegebene Aussage kann mehrmals vorkommen. Sie muss jedoch mindestens einmal vorkommen.

```
{Wiederkehrende Aussagen}
```

- Runde Klammern dienen der Gruppierung von Aussagen.

```
(Gruppe von Aussagen)
```

- | definiert alternative Aussagen.

```
(Aussage |  ... |  Aussage N)
```

# Anhang F

# Syntax der Testfall-Spezifikationssprache

Die Syntax ist in einem EBNF Derivat definiert um die Vorraussetzung für das Eclipse Plugin xtext zu erfüllen.

```
Testfallbeschreibung:
 //import
 (impor+=Import)*
 //Aktionen
 ("Aktionen" "Anfang"
   ("Aktion" aktion+=Aktion '.')+
 "Ende Aktionen.")*
 //Ereginisse
 ("Ereignisse" "Anfang"
   ("Ereignis" ereignis+=Ereignis '.')+
 "Ende Ereignisse.")*
 //Reaktionen
 ("Reaktionen" "Anfang"
   ("Reaktion" reaktion+=Reaktion '.')+
 "Ende Reaktionen.")*
  //Zustaende
 ("Zustaende" "Anfang"
   ("Zustand" zustand+=Zustand '.')+
 "Ende Zustaende.")*
 //Testfall
 (Testfall+=Testfall '.')*
  //Aktion
 ("Definition Aktionen" "Anfang"
  (aktionsdefinition+=Aktionsdefinition '.')+
  "Ende Definition Aktionen.")*
  //Ereignis
 ("Definition Ereignisse" "Anfang"
  (ereignisdefinition+=Ereignisdefinition '.')+
```

```
29   "Ende Definition Ereignis.")*
     //Reaktion
31  ("Definition Reaktionen" "Anfang"
     (reaktionsdefinition+=Reaktionsdefinition '.')+
33   "Ende Definition Reaktionen.")*
     //Zustand
35  ("Definition Zustaende" "Anfang"
     (zustanddefinition+=Zustanddefinition '.')+
37   "Ende Definition Zustaende.")*;

39 //Uniform Resource Identifier
   Import : 'import' importURI=STRING;
41
   Parameter :
43  ("mit Parameter" Parameter=ID |"mit Parameterliste" Liste=Liste);

45 Prio :
    "mit Prioritaet" (STRING|INT);
47
   Testfall :
49  "Testfall" name=ID ( parameter=Parameter )? ( prioritaet=Prio )? "↩
       Anfang"
      ("Beschreibung:" STRING '.')*
51    (vorbed+=Vorbedingung '.')?
      (testschritt+=Testschritt '.')+
53    (nachbed+=Nachbedingung '.')?
    "Ende Testfall" ;
55
   Testschritt :
57  testfallschritt+=Testfallschritt  |  aktionschritt+=Aktionschritt | ↩
       nachricht+=Nachrichtenschritt | ereignis+=Ereignisschritt  | ↩
       kontrollstruktur+=Kontrollstruktur;

59 Testfallschritt :
    "Testfallaufruf" testfall+=[Testfall] ( parameter=Parameter )? ;
61
   Liste :
63     "(" listOfArgs+=LElement (',' listOfArgs+=LElement)* ")";

65 LElement : name=ID;

67 //VOR- und NACHBEDINGUNG
   Vorbedingung :
69  "Vorbedingung" "Anfang"
    ("Zustand" zustand+=[Zustand] '.')+
71 "Ende Vorbedingung" ;

73 Nachbedingung :
    "Nachbedingung" "Anfang"
75  ("Zustand" zustand+=[Zustand] '.')+
   "Ende Nachbedingung";
77
```

```
Zustand : zustandID=ID name=STRING;

//AKTION
Aktion :
 aktionID=ID name=STRING;

Aktionschritt :
 ( "Aktion" aktion=[Aktion]( parameter=Parameter )? (zeitverhalten+=Zeitverhalten)? | aktionsgruppe+=Aktionsgruppe ) ("führt zur" ("parallelen")? (reaktion+=Reaktionsschritt | rgruppe+=Reaktionsgruppe | rauswertung+=Reaktionsauswertung))?;

Aktionsgruppe :
 "Gruppe von Aktionen" ("Parallel")? (zeitverhalten+=Zeitverhalten)? "Anfang"
  (testschritt+=Testschritt '.')+
"Ende Gruppe von Aktionen";

Reaktionsgruppe :
 "Gruppe von Reaktionen" ("Parallel")? (zeitverhalten+=Zeitverhalten)? "Anfang"
 (reaktion+=Reaktionsschritt '.' | rgruppe+=Reaktionsgruppe '.' | kontrollstruktur+=Kontrollstruktur '.')+
"Ende Gruppe von Reaktionen";

//NACHRICHTEN
Nachrichtenschritt :
 sg+=Steuergeraet "sendet Nachricht" nachricht+=STRING "mit Wert" wert+=STRING "an" sg2+=Steuergeraet (zeitverhalten+=Zeitverhalten)? '.'|
 sg+=Steuergeraet "empfängt Nachricht" nachricht+=STRING "mit Wert" wert+=STRING "von" sg2+=Steuergeraet (zeitverhalten+=Zeitverhalten)? '.';

Steuergeraet : STRING;

//ZEITASPEKTE
enum Zeiteinheit :
   Nanosekunden = "ns" | Mikrosekunden = "us" | Millisekunden = "ms" | Sekunden = "s" | Minuten = "min" | Stunden = "h" | Tage = "d";

Zeitangabe : INT zeiteinheit=Zeiteinheit;

Zeiten :
 "frühestens" zeitangabe+=Zeitangabe | "spätestens" zeitangabe+=Zeitangabe | "dauer" zeitangabe+=Zeitangabe | "Zykluszeit" zeitangabe+=Zeitangabe;

Zeitverhalten :
 "mit Zeitverhalten" zeiten+=Zeiten ("und" weiterezeiten+=Zeiten)? | "bis" ereignis+=Ereignisschritt;
```

```
115
  Ereignisschritt :
117  "Ereignis" ereignis+=[Ereignis] ( parameter=Parameter )? ;

119 Ereignis : ereignisID=ID name=STRING;

121 //KONTROLLSTRUKTUREN
  Kontrollstruktur :
123  Warteanweisung | Wiederholung | Iteration;

125 Warteanweisung:
    "Warte" (zeitangabe=Zeitangabe | "bis" ereignis=[Ereignis] );
127
  Wiederholung :
129  "Wiederhole" ( "mit Iterator" ID anzahl+=INT "mal" | "bis" ereignis↩
       +=Ereignisschritt ) "Anfang"
   (testschritt+=Testschritt '.')+
131 "Ende Wiederhole";

133 Iteration :
   "Iteriere" "mit Iterator" ID durch Liste  "Anfang"
135  (testschritt+=Testschritt '.')+
  "Ende Iteriere";
137
  //REAKTION
139 Reaktionsschritt:
   "Reaktion" reaktion=[Reaktion] ( parameter=Parameter )? (↩
       zeitverhalten+=Zeitverhalten)?;
141
  Reaktion : reaktionID=ID name=STRING;
143
  Verdict : "pass" | "fail";
145
  Prioritaet : INT;
147
  RAuswertung :
149  "Wenn" (reaktion+=Reaktionsschritt | lwert+=STRING '=' rwert+=STRING↩
       )? "dann" ("Verdict:" verdict+=Verdict | "Priorität:" prio+=↩
      Prioritaet | testfallschritt+=Testfallschritt) ( "und" ("Verdict↩
      :" verdict+=Verdict | "Priorität:" prio+=Prioritaet))?;

151 Reaktionsauswertung :
   "Reaktionsauswertung" "Anfang"
153   (wertung+=RAuswertung '.')?
  "Ende Reaktionsauswertung";
155
  //Koppelebene zur Testfallimplementierung
157 Bibliotheksaufruf :
   "Funktionsaufruf:" bibliotheksfunktion=STRING;
159
  Signalsetzung :
161  "Signalsetzung:" signalname=STRING "=" signalverlauf=STRING;
```

```
AtomareAktionen :
 bibliotheksfunktion=Bibliotheksaufruf | signalsetzung=Signalsetzung;

AtomareReaktionen :
 bibliotheksfunktion=Bibliotheksaufruf | signalsetzung=Signalsetzung;

AtomareEreignis :
 bibliotheksfunktion=Bibliotheksaufruf | signalsetzung=Signalsetzung;

AtomareZustand :
 bibliotheksfunktion=Bibliotheksaufruf | signalsetzung=Signalsetzung;

Aktionsdefinition :
 "Def Aktion" aktion=[Aktion] ( parameter=Parameter )? (
 ":=" AktionRef=[Aktion] ( parameter=Parameter )? (zeitverhalten+=Zeitverhalten)? |
 "Anfang"
  (atom+=AtomareAktionen '.')+
 "Ende Def Aktion");

Reaktionsdefinition :
 "Def Reaktion" reaktion=[Reaktion] ( parameter=Parameter )? (
 ":=" reaktionRef=[Reaktion] ( parameter=Parameter )? (zeitverhalten+=Zeitverhalten)? |
 "Anfang"
  (atom+=AtomareReaktionen '.')+
 "Ende Def Reaktion");

Ereignisdefinition :
 "Def Ereignis" ereignis=[Ereignis] ( parameter=Parameter )? (
 ":=" ereignisRef=[Ereignis] ( parameter=Parameter )? (zeitverhalten+=Zeitverhalten)? |
 "Anfang"
  (atom+=AtomareEreignis '.')+
 "Ende Def Ereignis");

Zustanddefinition :
 "Def Zustand" zustand=[Zustand] ( parameter=Parameter )? (
 ":=" zustandRef=[Zustand] ( parameter=Parameter )? (zeitverhalten+=Zeitverhalten)? |
 "Anfang"
  (atom+=AtomareZustand '.')+
 "Ende Def Zustand");
```

**Listing F.1:** Syntaxdefinition der Testfall-Spezifikationssprache

# Anhang G

# Theoretische Grundlage für die Definition der Syntax

## G.1 Alphabet, Zeichen, Wörter und Sprache

Der Begriff *Alphabet* bezeichnet eine endliche, nicht-leere Menge. Die Elemente des Alphabeten heissen *Zeichen* oder *Symbole*. Beispiel: Die Menge der natürlichen Zahlen $\mathbb{N}$ ist ein Alphabet. Die Elemente 1, 2, 3 usw. sind Zeichen der Menge $\mathbb{N}$.

Aus dem Alphabet werden Wörter gebildet, die sich durch Hintereinanderschreiben der Zeichen aus dem Alphabet bilden lassen. Die Menge aller möglichen Wörter aus dem Alphabet $\Sigma$ bezeichnet man als $\Sigma^*$, wobei das leere Wort $\varepsilon$ Teil dieser Menge ist. Anderenfalls wird sie als $\Sigma^+$ bezeichnet.

Eine *Sprache A* ist eine Teilmenge der Menge aller Wörter $\Sigma^*$, die aus den Zeichen des Alphabets gebildet werden können. Mathematisch wird dieser Sachverhalt definiert als: $A \subset \Sigma^*$. Die bekannten Mengenoperationen können auf die Teilmengen angewandt werden. Ferner werden diese Teilmengen auf gruppentheoretische Grundlagen definiert, was in dieser Einführung verzeichnet wird.

Wichtig ist, dass diese Sprachen durch zwei Mechanismen definiert werden können: Grammatiken und Automaten. Die Grammatiken erzeugen die Sprache, wohingegen die Automaten ein vorgegebenes Wort einer Sprache zuordnen können. Im Folgenden werden die Grammatiken und Automaten vorgestellt.

## G.2 Grammatiken

In Abschnitt 6.1 über die Beschreibung von Sprachen wurden die Grammatiken in einer verständlichen Form kompakt eingeführt: Eine Grammatik besteht aus vier Elementen. Dem Alphabeten, den Variablen, den Regeln und einer Startvariable. Die formale Definition der Grammatik lautet:

**Definition G.1 (Grammatik)** *Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, das folgende Bedingungen erfüllt. $V$ ist eine endliche Menge, die Menge der Variablen. $\Sigma$ ist eine endliche Menge, das Terminalalphabet. Es muss gelten: $V \cap \Sigma = \emptyset$. $P$ ist die endliche Menge der Regeln oder Produktionen. Formal ist $P$ eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$. $S \epsilon V$ ist die Startvariable.*

*Seien $u, v \in (V \cup \Sigma)^*$. Wir definieren die Relation $u \Rightarrow_G v$ (in Worten: u geht unter G unmittelbar über in v), falls u und v die Form haben*

$$u = xyz$$
$$v = xy'z \text{ mit } x, z \in (V \cup \Sigma)^*$$

*und $y \rightarrow y'$ eine Regel in P ist. Falls kar ist, welche Grammatik G gemeint ist, so schreiben wir einfach $u \Rightarrow v$ anstatt $u \Rightarrow_G v$.*

*Die von G dargestellte (erzeugte, definierte) Sprache ist*

$$L(G) = w \in \Sigma^* | S \Rightarrow_G^* w$$

*Hierbei ist $\Rightarrow_G^*$ die reflexive und transitive Hülle von $\Rightarrow_G$ (...).*

*Eine Folge von Wörtern $(w_0, w_1, ..., w_n)$ mit $w_0 = S, w_n \in \Sigma^*$ und $w_0 \Rightarrow w_1 \Rightarrow ... \Rightarrow w_n$ heisst Ableitung von $w_n$. Ein Wort $w \in (V \cup \Sigma)^*$, das also noch Variablen enthält - wie es typischerweise im Verlauf einer Ableitung auftritt - heisst auch Satzform. [Sch01b]*

### Die Chomsky-Hierarchie

Das Hauptaugenmerkt liegt bei den Grammatiken auf den Regeln. Aufgrund unterschiedlicher Zusammensetzungen bei den Regeln, lassen sich die Grammatiken in eine Hierarchie einordnen. Abbildung 6.1 auf Seite 175 zeigt die visuelle Aufteilung. Im Folgenden die formale Definition:

**Definition G.2 (Chomsky-Hierarchie)** *Jede Grammatik ist zunächst automatisch vom Typ 0. Das heisst, bei Typ 0 sind den Regeln keinerlei Einschränkungen auferlegt. (Man spricht auch von allgemeinen Phrasenstrukturgrammatiken).*

*Eine Grammatik ist vom Typ 1 oder kontextsensitiv, falls für alle Regeln $w_1 \rightarrow w_2$ in $P$ gilt: $|w_1| \leq |w_2|$.*

*Eine Typ 1-Grammatik ist vom Typ 2 oder kontextfrei, falls für alle Regeln $w_1 \rightarrow w_2$ in $P$ gilt, dass $w_1$ eine einzelne Variable ist, d.h. $w_1 \in V$.*

*Eine Typ 2-Grammatik ist vom Typ 3 oder regulär, falls zusätzlich gilt: $w_2 \in \Sigma \cup \Sigma V$, d.h. die rechten Seiten von Regeln sind entweder einzelne Terminalzeichen oder ein Terminalzeichen gefolgt von einer Variablen.*

*Eine Sprache $L \subseteq \Sigma^*$ heisst vom Typ 0 (Typ 1, Typ 2, Typ 3), falls es eine Typ 0 (Typ 1, Typ 2, Typ 3)-Grammatik $G$ gibt mit $L(G) = L$. [Sch01b]*

## G.3 Automaten

Wie bei der Einführung der Automaten erwähnt, können Automaten ein Wort erkennen bzw. akzeptieren, falls das Wort einer gegebenen Sprache gehört.

Die Automaten dienen die unterschiedlichen Typen der Grammatik zu erkennen.

### Endlicher Automat

Der endliche Automat erkennt die regulären Sprachen und ist definiert als:

**Definition G.3 (Endlicher Automat)** *Ein (deterministischer) endlicher Automat $M$ wird spezifiziert durch ein 5-Tupel*

$$M = (Z, \Sigma, \delta, z_0, E).$$

*Hierbei bezeichnet $Z$ die Menge der Zustände und $\Sigma$ ist das Eingabealphabet. $Z \cap \Sigma = \emptyset$. $Z$ und $\Sigma$ müssen - wie schon der Name sagt - endliche Mengen sein. $z_0 \in Z$ ist der Startzustand, $E \subseteq Z$ ist die Menge der Endzustände und $\delta : Z \times \Sigma \rightarrow Z$ heisst die Überführungsfunktion. [Sch01b]*

Der Zustandsgraph ist eine bildhafte Darstellung des endlichen Automaten. So kann man das „akzeptieren“ eines Wortes als den Begin von dem Startzustand sehen, wo die einzelnen Zeichen den Zustandsübergang bewirken. Erreicht man nach dem letzten Zeichen des Wortes einen Endzustand, so ist das Wort ein Teil der Sprache.

In [Sch01b] wird der Beweis geliefert, dass jede durch endliche Automaten erkennbare Sprache regulär ist und das eine Sprache regulär ist, genau dann wenn sie durch einen endlichen Automaten erkannt wird. Somit bilden beide Arten (Grammatik und Automat) die gleiche Menge ab.

## Kellerautomat

Kellerautomaten sind eine Erweiterung der endlichen Automaten, das die kontextfreien Sprachen erkennt. Hierfür wird der endliche Automat um einen *Speicher* erweitert. Die formale Definition lautet:

**Definition G.4 (Kellerautomat)** *Ein (nichtdeterministischer) Kellerautomat (engl.: pushdown automaton, kurz: PDA) wird angegeben durch ein 6-Tupel*

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#).$$

*Hierbei sind:*

- $Z$ *die endliche Menge der Zustände,*
- $\Sigma$ *das Eingabealphabet,*
- $\Gamma$ *das Kelleralphabet,*
- $\delta : Z \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow P_e(Z \times \Gamma^*)$ *die Überführungsfunktion,*
  *(Hierbei bedeutet $P_e$ die Menge aller endlichen Teilmengen)*
- $z_0 \in Z$ *der Startzustand,*
- $\# \in \Gamma$ *das unterste Kellerzeichen. [Sch01b]*

Die Erklärung der Überführungsfunktion an einem Beispiel:
Die Funktion $\delta(z, aA) \ni (z', B_1...B_k)$ wäre die Überführung vom Zustand $z$ in Zustand $z'$, wenn das Zeichen $a$ gelesen wird und - zusätzlich zum endlichen Automaten - im „Keller“ anschliessend das Kellerzeichen $A$ durch $B_1...B_k$ ersetzt wird.

Eine Erweiterung der Definition für den Kellerautomaten in Richtung deterministisch erfolgt, um Wörter der deterministisch kontextfreien Sprachen akzeptieren zu können.

## Turingmaschine

Schliesslich erfolgt eine weitere Definition: Die Turingmaschine. Mit der Turingmaschine wird die Beschränkung des Kellerautomaten beim Speicher (Last-in, First-Out) aufgehoben. Die Definition lautet:

**Definition G.5 (Turingmaschine)** *Eine Turingmaschine (kurz: TM) ist gegeben durch ein 7-Tupel*

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E).$$

*Hierbei sind:*

- $Z$ *die endliche Zustandsmenge,*
- $\Sigma$ *das Eingabealphabet,*
- $\Gamma \ni \Sigma$ *das Arbeitsalphabet,*
- $\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times L, R, N$ *im deterministischen Fall (bzw.* $\delta : Z \times \Gamma \rightarrow P(Z \times \Gamma \times L, R, N)$ *im nichtdeterministischen Fall) die Überführungsfunktion,*
- $z_0 \in Z$ *der Startzustand*
- $\square \in \Gamma - \Sigma$ *das Blank,*
- $E \subseteq Z$ *die Menge der Endzustände. [Sch01b]*

Auch hier die Erklärung der Überführungsfunktion an einem Beispiel:
Die Funktion $\delta(z, a) = (z', b, x)$ bzw. $\delta(z, a) \ni (z', b, x)$ sagt aus: Wenn sich die Turingmaschine im Zustand $z$ befindet und unter dem Schreib-Lesekopf das Zeichen $a$ steht, so geht die Turingmaschine im nächsten Schritt in den Zustand $z'$ über und - wieder als unterschied zum endlichen Automaten - auf den Platz von $a$ $b$ auf das Band und führt danach eine Kopfbewegung $x \in L, R, N$ aus. Wobei $L$ für links, $R$ für rechts und $N$ für neutral steht.

Die so definierte, allgemeine Turingmaschine erkennt Typ 0 Sprachen.

Eine Einschränkung der Definition der Turingmaschine erfolgt auf die Definition der linear beschränkten, nichtdeterministischen Turingmaschinen[1], die Typ 1 Sprachen erkennen. Für eine detailierte (formale) Erklärung wird hier auf [Sch01b] verwiesen.

[1] Informell: Die Speicherplatzgröße der Turingmaschine wird auf die Wortgröße (Anzahl der Zeichen im Wort) der Sprache begrenzt

# Literaturverzeichnis


[AD94] ALUR, RAJEEV und DAVID L. DILL: *A theory of timed automata*. Theoretical Computer Science, 126:183–235, 1994.

[ADA09] ADAC: *ADAC Pannenstatistik 2009*. Informationen aus der FAHRZEUGTECHNIK, 2009.

[AMBD04] ABRAN, ALAIN, JAMES W. MOORE, PIERRE BOURQUE und ROBERT DUPUIS (Herausgeber): *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, California, 2004.

[Bal98] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik, Teil 2: Softwaremanagement, Software-Qualitaetssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998.

[BB79] BÜNTING, K.-D. und H. BERGENHOLTZ: *Einführung in die Syntax*. Athenäum, Königstein/Ts., 1979.

[Bei90] BEIZER, B.: *Software testing techniques*. Van Nostrand Reinhold Co., New York, NY, USA, 2 Auflage, 1990.

[BG08] BANDYOPADHYAY, ARITRA und SUDIPTO GHOSH: *Using UML Sequence Diagrams and State Machines for Test Input Generation*. In: *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, Seiten 309–310, 2008.

[BHR84] BROOKES, S. D., C. A. R. HOARE und A. W. ROSCOE: *A Theory of Communicating Sequential Processes*. J. ACM, 31(3):560–599, July 1984.

[BHS07] BAERO, THOMAS, JOCHEN HAGEL und ERIC SAX: *Effizientes Testen durch optimierte Prozesse*. E&E Kompendium, Seiten 263–265, 2006/2007.

[Bin99] BINDER, ROBERT V.: *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Object Technology Series. Addison, 1999.

[BJK05] BROY, MANFRED, BENGT JONSSON und JOOST-PIETER KATOEN: *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer, August 2005.

[BK04] BERKENKÖTTER, KIRSTEN und RAIMUND KIRNER: *Real-Time and Hybrid Systems Testing*. In: *Model-Based Testing of Reactive Systems*, Seiten 355–387, 2004.

[BN03] BROEKMAN, BART und EDWIN NOTENBOOM: *Testing Embedded Software*. Addison-Wesley, 2003.

[Boe79] BOEHM, B. W.: *Guidelines for Verifying and Validating Software Requirements and Design Specifications*. In: SAMET, P. A. (Herausgeber): *Euro IFIP 79*, Seiten 711–719. North Holland, 1979.

[Boe86] BOEHM, B: *A spiral model of software development and enhancement*. SIGSOFT Softw. Eng. Notes, 11(4):14–24, 1986.

[Boe06] BOEHM, BARRY: *A view of 20th and 21st century software engineering*. In: *ICSE '06: Proceedings of the 28th international conference on Software engineering*, Seiten 12–29, New York, NY, USA, 2006. ACM.

[Bro97] BROY, MANFRED: *Refinement of Time.* In: *ARTS*, Seiten 44–63, 1997.

[BSS05] BÄRO, T., E. SAX und S. SCHMERLER: *Erhöhung der Testtiefe durch HiL-Testing*. Simulations- und Testmethoden für Software in Fahrzeugsystemen (Proceedings der Jahrestagung der ASIM/GI-Fachgruppe 4.5.5 'Simulation technischer Systeme'), 2005.

[BSV06] BURMESTER, SVEN, RENATE STÜCKA und DR. ALEXANDER VOSS: *Werkzeugkopplung im Testprozess*. Hanser Automotive, 2006.

[BV03] BIRKHÖLZER, T. und J. VAUPEL: *IT-Architekturen*. VDE Verlag, 2003.

[BvdBK98] BROY, M., M. VON DER BEECK und I. KRÜGER: *SOFTBED: Problemanalyse für das Großverbundprojekt ?Systemtechnik Automobil - Software für eingebettete Systeme?*, 1998.

[Bär08] BÄRO, THOMAS: *Prozesse der Validierung von Steuergeräten im Kontext bestehender Standards automobiler Softwareentwicklung*. Doktorarbeit, Techn. Univ. Karlsruhe, 2008.

[Büs03] BÜSSOW, ROBERT: *Model Checking Combined Z and Statechart Specifications*. Doktorarbeit, Technische Universität Berlin, 2003.

[CAC+95] COURCOUBETIS, ALUR, R. ALUR, C. COURCOUBETIS, N. HALBWACHS, T. A. HENZINGER, P. H. HO, X. NICOLLIN, A. OLIVERO, J. SIFAKIS und S. YOVINE: *The Algorithmic Analysis of Hybrid Systems*. Theoretical Computer Science, 138:3–34, 1995.

[Cal07] CALAMÉ, JENS R.: *Adaptive Test Case Execution in Practice*. Technical Report SEN-E0703, Centrum voor Wiskunde en Informatica, June 2007.

[CF98] CHARLIER, BAUDOUIN LE und PIERRE FLENER: *Specifications Are Necessarily Informal or: Some More Myths of Formal Methods*. The Journal of Systems and Software, 40(3):275–??, March 1998.

[CF05] CONRAD, MIRKO und INES FEY: *Modell-basierter Test von Simulink/Stateflow-Modellen*. In: *Kolloqium "Testen im System - und Software-Life-Cycle", Technische Akademie Esslingen, 29.-30.Nov 2005, S.278-298*, 2005.

[CFGK05] CONRAD, MIRKO, INES FEY, MATTHIAS GROCHTMANN und TORSTEN KLEIN: *Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei DaimlerChrysler*. Inform., Forsch. Entwickl., 20(1-2):3–10, 2005.

[CM09] CUADRADO, JESÚS SÁNCHEZ und JESÚS GARCÍA MOLINA: *Building Domain-Specific Languages for Model-Driven Development*. IEEE SOFTWARE, 2009.

[CMM+00] CARRINGTON, DAVID, IAN MACCOLL, JASON MCDONALD, LEESA MURRAY und PAUL STROOPER: *From Object-Z Specifications to ClassBench Test Suites*. Journal on Software Testing, Verification and Reliability, 10(2), 2000.

[CNP08] CARVER, JEFFREY C., NACHIAPPAN NAGAPPAN und ALAN PAGE: *The Impact of Educational Background on the Effectiveness of Requirements Inspections: An Empirical Study*. IEEE Transactions on Software Engineering, 34(6):800–812, 2008.

[Con04] CONRAD, M.: *Modell-basierter Test eingebetteter Software im Automobil - Auswahl und Beschreibung von Testszenarien*. Doktorarbeit, Techn. Univ. Berlin, 2004.

[Con09] CONRAD, MIRKO: *Testing-based translation validation of generated code in the context of IEC 61508*. Form. Methods Syst. Des., 35(3):389–401, 2009.

[CS03] CONRAD, MIRKO und ERIC SAX: *Mixed Signals*. In: *Testing Embedded Software*, Seiten 229 – 249. Addison Wesley, 2003.

[Dan09] DANISCH, RUBEN: *ESP wird Pflicht*. ATZ online Nachrichten, 17.03.2009.

[DGH09] DANTRA, RUSKIN, JOHN GRUNDY und JOHN HOSKING: *A domain-specific visual language for report writing using Microsoft DSL tools*. In: *VLHCC '09: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2009.

[DGN02] DAI, ZHEN RU, JENS GRABOWSKI und HELMUT NEUKIRCHEN: *Timed TTCN-3 - A Real-time Extension for TTCN-3*. In: *TestCom '02: Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV*, Seiten 407–424, 2002.

[Dij78] DIJKSTRA, EDSGER W.: *On the foolishness of "natural language programming"*. circulated privately, 1978.

[Dij82] DIJKSTRA, EDSGER W.: *On the role of scientific thought*. In: *Selected Writings on Computing: A Personal Perspective*, Seiten 60–66. Springer-Verlag, 1982.

[DN08] DANG, THAO und TARIK NAHHAL: *Using Disparity to Enhance Test Generation for Hybrid Systems*. In: *TestCom '08 / FATES '08: Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems*, Seiten 54–69, Berlin, Heidelberg, 2008. Springer-Verlag.

[Dre06] DREIER, RICO: *Verteilte Ausführung hybrider System-Modelle auf heterogenen Rechnerplattformen*. Doktorarbeit, Publikation FZI, 2006.

[FB89] FELBER, HELMUT und GERHARD BUDIN: *Terminologie in Theorie und Praxis*. G. Narr, 1989.

[FK03] FREY, GEORG und STEFAN KOWALEWSKI: *Entwurf einer Richtlinie zur Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik*. In: *Tutorial, Entwurf komplexer Automatisierungssysteme, Braunschweig*, Seiten 15–23, 2003.

[fN01] NORMUNG, DEUTSCHES INSTITUT FÜR: *Klassifikationssysteme; Erstellung und Weiterentwicklung von Klassifikationssystemen*. Beuth-Verlag, 1987-01.

[Fow09] FOWLER, MARTIN: *A Pedagogical Framework for Domain-Specific Languages*. IEEE Software, 26:13–14, 2009.

[Gau05] GAUS, WILHELM: *Dokumentations- und Ordnungslehre : Theorie und Praxis des Information-Retrieval*. Springer, Berlin [u.a.], 5., überarb. Aufl. Auflage, 2005.

[GCF+06] GROSSMANN, JÜRGEN, MIRKO CONRAD, INES FEY, ALEXANDER KRUPP, KLAUS LAMBERG und CHRISTIAN WEWETZER: *TestML – A Test Exchange Language for Model-based Testing of Embedded Software*. In: *Automotive Software Workshop*, San Diego, 1 Januar 2006.

[Gev06a] GEVATTER, HANS-JÜRGEN: *Handbuch der Mess- und Automatisierungstechnik im Automobil*. Springer, 2006.

[Gev06b] GEVATTER, HANS-JÜRGEN: *Handbuch der Mess- und Automatisierungstechnik im Automobil*. Springer, 2., vollständig bearbeitete Auflage Auflage, 2006.

[GM06] GROSSMAN, JÜRGEN und WOLFGANG MÜLLER: *A Formal Behavioral Semantics for TestML*. In: *In Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (IEEE-ISoLA)*, Seiten 19–26, Paphos, Cyprus, November 2006. IEEE.

[Gra00] GRABOWSKI, JENS: *TTCN-3 - A new Test Specification Language for Black-Box Testing Of Distributed Systems*. In: *Proceedings of the 17th International Conference and Exposition on Testing Computer Software (TCS'2000), Theme : Testing Technology vs.Testers' Requirements, Washington D.C*, 2000.

[Gra02] GRABOWSKI, JENS: *TTCN-3 ... und danach? - Gedanken und Vorschläge zur Weiterentwicklung von TTCN-3*. Presentation, Testing Technologies IST GmbH, Berlin, 11. Januar 2002, Januar 2002.

[Gra09] GRAUNITZ, BJÖRN: *Wachsender Markt für Automotive-HMI-Systeme*. Elektronik automotive, 06.08.2009.

[Gra10a] GRAUNITZ, BJÖRN: *Markt für automobile Audio-Systeme verdoppelt sich bis zum Jahr 2016*. Elektronik automotive, 16.03.2010.

[Gra10b] GRAUNITZ, BJÖRN: *Die Automotive-Industrie im Umbruch*. Elektronik automotive, 18.01.2010.

[GRDNPG97] GREGORIO-RODRIGREZ, CARLOS, LUIS FERNANDO LLANA DIAZ, MANUEL NUNEZ und PEDRO PALAO-GOSTANZA: *Testing Semantics for a Probabilistic-Timed Process Algebra*. In: *ARTS '97: Proceedings of the 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software*, Seiten 353–367, London, UK, 1997. Springer-Verlag.

[GS83] GENTNER, DEDRE und ALBERT L. STEVENS: *Mental models*. Routledge, 1983.

[GS05] GROCHTMANN, MATTHIAS und LINDA SCHMUHL: *Systemverhaltensmodelle zur Spezifikation bei der modellbasierten Entwicklung von eingebetteter Software im Automobil*. In: *Dagstuhl Workshop Modellbasierte Entwicklung Eingebetteter Systeme I*, 2005.

[Gud03] GUDDAT, ULRICH: *Automatisierte Tests von Telematiksystemen im Automobil*. Doktorarbeit, Tübingen University, 2003.

[GWWH00] GRABOWSKI, JENS, ANTHONY WILES, COLIN WILLCOCK und DIETER HOGREFE: *On the Design of the New Testing Language TTCN-3*. In: *TestCom '00: Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems*, Deventer, The Netherlands, 2000. Kluwer, B.V.

[Hal90] HALL, ANTHONY: *Seven Myths of Formal Methods*. IEEE Software, 7:11–19, 1990.

[Har01a] HARTMANN, KNUT: *Text-Bild-Beziehungen in multimedialen Dokumenten : eine Analyse aus Sicht von Wissensrepräsentation, Textstruktur und Visualisierung*. Doktorarbeit, Univ. Magdeburg, Fak. für Informatik, 2001.

[Har01b] HARTMANN, NICO: *Automation des Tests eingebetteter Systeme am Beispiel der Kraftfahrzeugelektronik*. Doktorarbeit, Techn. Univ. Karlsruhe, 2001.

[HCDG+02] HUDAK, J., S. COMELLA-DORDA, D. GLUCH, G. LEWIS und C. WEINSTOCK: *Model-Based Verification: Abstraction Guidelines*. Technischer Bericht 2002-TN-011, CMU/SEI, 2002.

[HE07] HEISSING, BERND und METIN ERSOY: *Fahrwerkhandbuch Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik, Perspektiven: Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik, Perspektiven*. Vieweg+Teubner Verlag, 2007.

[Hen96] HENZINGER, T. A.: *The Theory of Hybrid Automata*. In: *Proceedings of LICS*, Seiten 278–292. IEEE Computer Society Press, 1996.

[HH08] HHN, REINHARD und STEPHAN HPPNER: *Das V-Modell XT: Grundlagen, Methodik und Anwendungen (eXamen.press)*. Springer Publishing Company, Incorporated, 2008.

[HKO06] HARTMAN, ALAN, MIKA KATARA und SERGEY OLVOVSKY: *Choosing a Test Modeling Language: A Survey.* In: BIN, EYAL, AVI ZIV und SHMUEL UR (Herausgeber): *Haifa Verification Conference*, Band 4383 der Reihe *Lecture Notes in Computer Science*, Seiten 204–218. Springer, 2006.

[Hor05] HORSTMANN, MARC: *Verflechtung von Test und Entwurf für eine verlässliche Entwicklung eingebetteter Systeme im Automobilbereich*. Doktorarbeit, Technische Universität Braunschweig, 2005.

[Hut06] HUTTER, ALEXANDER: *Eine Systematik zur Erstellung virtueller Steuergeräte für Hardware-in-the-Loop-Integrationstests*. Doktorarbeit, Techn. Univ. München, 2006.

[IAB97] IABG: *Entwicklungsstandard für IT-Systeme des Bundes - Vorgehensmodell*. IABG, Allgemeiner Umdruck Nr. 250/1 - 1997.

[IAB09] IABG: *V-Modell XT, Grundlagen des V-Modells, 1.Teil*, Februar 2009. http://v-modell.iabg.de.

[iDuVD94] VDE (DKE), DEUTSCHE ELEKTROTECHNISCHE KOMMISSION IM DIN UND: *Leittechnik; Regelungstechnik und Steuerungstechnik; Allgemeine Grundbegriffe*, Band 1. 1994.

[Iee91] IEEE: *IEEE 610-1991: IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1991.

[Iee98] IEEE: *Std 829-1998: IEEE Standard for Software Test Documentation*. IEEE, 1998.

[IES07] IESE, FRAUNHOFER: *TTCN-3*. http://www.softwarekompetenz.de, 2007.

[Ins05] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS: *Standard Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML*, Stand 2005.

[Ins06] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS: *IEEE Guide for the Use of IEEE Std 1641, Standard for Signal and Test Definition* , 2006.

[ISO96] *International Standard: Information technology – Syntactic metalanguage – Extended BNF*. ISO/IEC 14977: 1996(E), First Edition, Dezember 1996.

[IST07] ISTQB: *ISTQB Test Glossar*. http://www.imbus.de/glossary/glossary.pl, 2007.

[KEGZ08] KAISER, OLIVER S., HEINZ EICKENBUSCH, VERA GRIMM und AXEL ZWECK: *Zukunft des Autos*. Technischer Bericht, Zukünftige Technologien Consulting (ZTC) der VDI Technologiezentrum GmbH, 2008.

[KH08] KRAAS, ALEXANDER und MARTIN HACKENBERG: *XML-based Representation of Test Cases for Distributed Systems*. In: *Model-based Testing in Practice*, 2008.

[Kho02] KHOUMSI, AHMED: *New results for testing distributed real-time reactive systems using a centralized method*. In: *Proceedings of the IASTED International Conference APPLIED INFORMATICS, Innsbruck, Austria*. ACTA Press, 2002.

[Kno07] KNOLLMANN, VOLKER: *UML-basierte Testfall- und Systemmodelle für die Eisenbahnleit- und -sicherungstechnik*. Doktorarbeit, Braunschweig, Techn., Univ., 2007.

[Kno08] KNORRA, ULRICH: *Zunehmende Komplexität erschwert die Integration bei Fahrerassistenzsystemen*. ATZ online Nachrichten, 05.11.2008.

[Koy91] KOYMANS, RON: *(Real) Time: A Philosophical Perspective.* In: *REX Workshop*, Seiten 353–370, 1991.

[KP08] KIFFE, GERHARD und DR. ANDREA PAGGEL: *EXAM – Methodik zur Darstellung, Durchführung und Auswertung von plattformunabhängigen, wiederverwendbaren Testfällen*. In: *2. AutoTest Test von Hard- und Software in der Automobilentwicklung*, 2008.

[KRRS05] KONSCHAK, THOMAS, RAINER RASCHE, DR. PETER RISSLING und THOMAS SUSSEBACH: *Testprozesse bei der BMW Group*. Hanser Automotive, 2005.

[KS04] KIM, HYE YEON und FREDERICK T. SHELDON: *Testing Software Requirements with Z and Statecharts Applied to an Embedded Control System*. Software Quality Journal, 12(3):231–264, 2004.

[KSS04] KUHLEN, RAINER, THOMAS SEEGER und DIETMAR STRAUCH: *Grundlagen der praktischen Information und Dokumentation : Bd. 1: Handbuch zur Einführung in die Informationswissenschaft und -praxis*. Saur, München [u.a.], 5 Auflage, 2004.

[KST04] KING, PETER, PATRICK SCHMITZ und SIMON THOMPSON: *Behavioral reactivity and real time programming in XML: functional programming meets SMIL animation*. In: *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*, Seiten 57–66, New York, NY, USA, 2004. ACM.

[Kus06] KUSCHE, FLORIAN: *Eine Systematik zur E/E-Architekturunabhängigen Beschreibung von HIL-Echtzeit-Integrationstests*. Doktorarbeit, Universität Ulm, 2006.

[Kut09] KUTHER, THOMAS: *Neue EU-Verordnung für verbesserten Fußgängerschutz macht Bremsassistenten verpflichtend*. Elektronik Praxis, 26.11.2009.

[Lam01] LAMBERG, KLAUS: *Methodik zur systematischen Bereitstellung von HIL-Testsystemen für Kfz-Steuergeräte*. Doktorarbeit, Technische Universität München, 2001.

[Laz05] LAZIC, LJUBOMIR: *The software testing challenges and methods*. In: *ICCOM'05: Proceedings of the 9th WSEAS International Conference on Communications*, Seiten 1–17, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[LBB+01] LEDRU, Y., L. DU BOUSQUET, P. BONTRON, O. MAURY, C. ORIAT und M.-L. POTET: *Test Purposes: Adapting the Notion of Specification to Testing*. In: *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, Seite 127, Washington, DC, USA, 2001. IEEE Computer Society.

[Leh04] LEHMANN, ECKARD: *Time partition testing : systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen*. Doktorarbeit, Techn. Univ. Berlin, 2004.

[Lem05] LEMMER, KARSTEN: *Mensch-Maschine-Interaktion*. Vieweg, 2005.

[Ley05] LEY, MARTIN: *Kontrollierte Textstrukturen - Ein (linguistisches) Informationsmodell für die Technische Kommunikation*. Doktorarbeit, Justus-Liebig-Universität Gießen, 2005.

[LG98] LAUBER, R. und P. GÖHNER: *Prozessautomatisierung 1, 3. Auflage*, 1998.

[Lig02] LIGGESMEYER, PETER: *Software-Qualit : Testen, Analysieren und Verifizieren von Software*. Spektrum, 2002. LIG p 02:1 1.Ex.

[Liu00] LIU, CHANG: *Platform-independent and tool-neutral test descriptions for automated software testing*. In: *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, Seiten 713–715, New York, NY, USA, 2000. ACM.

[Mar08] MARWEDEL, PETER: *Eingebettete Systeme*. Springer, October 2008.

[Mar09] MARTIN, ROBERT C.: *Clean Code: A handbook of agile software craftsmanship*. Prentice Hall, 2009.

[MB08] MALER, ODED und GRÉGORY BATT: *Approximating Continuous Systems by Timed Automata*. In: *FMSB '08: Proceedings of the 1st international workshop on Formal Methods in Systems Biology*, Seiten 77–89, Berlin, Heidelberg, 2008. Springer-Verlag.

[MHS05] MERNIK, MARJAN, JAN HEERING und ANTHONY M. SLOANE: *When and how to develop domain-specific languages*. ACM Computing Surveys (CSUR), 37(4):316–344, 2005.

[MSBT04] MYERS, GLENFORD J., COREY SANDLER, TOM BADGETT und TODD M. THOMAS: *The Art of Software Testing, Second Edition*. Wiley, June 2004.

[MST10] MATTHIES, DR. GREGOR, DR. KLAUS STRICKER und DR. JAN TRAENCKNER: *Zum E-Auto gibt es keine Alternative*. Bain & Company, 2010.

[Mül07] MÜLLER, CHRISTIAN: *Durchgängige Verwendung von automatisierten Steuergeräte-Verbundtests in der Fahrzeugentwicklung*. Doktorarbeit, Techn. Univ. Karlsruhe, 2007.

[NSL08] NAVET, NICOLAS und FRANCOISE SIMONOT-LION: *Automotive Embedded Systems Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 2008.

[Obj09a] OBJECT MANAGEMENT GROUP: *UML Infrastructure Specification v2.2*. Technischer Bericht, OMG, 2009.

[Obj09b] OBJECT MANAGEMENT GROUP: *UML Superstructure Specification v2.2*. Technischer Bericht, OMG, 2009.

[Ole08] OLEJNICZAK, ROBERT: *Systematisierung des funktionalen Tests eingebetteter Software*. Dissertation, Technische Universität München, München, 2008.

[OMG05] OMG: *UML Testing Profile Specification*. OMG, 2005.

[PAD+98] PELESKA, JAN, PETER AMTHOR, SABINE DICK, OLIVER MEYER, MICHAEL SIEGEL und CORNELIA ZAHLTEN (Herausgeber): *Testing Reactive Real-Time Systems, 5th International Symposium, FTRTFT'98, Lyngby, Denmark, September 14-18, 1998, Proceedings*, Band 1486 der Reihe *Lecture Notes in Computer Science*. Springer, 1998.

[Pat82] PATZAK, GEROLD: *Systemtechnik, Planung komplexer innovativer Systeme. Grundlagen, Methoden, Techniken*. Springer Verlag, 1982.

[Pat05a] PATING, SUSANNE: *Die Evolution von Modellierungssprachen*. Doktorarbeit, Otto-von-Guericke-Universität, 2005.

[Pat05b] PATTON, RON: *Software Testing (2nd Edition)*. Sams, 2005.

[PCG05] POHLHEIM, H., M. CONRAD und A GRIEP: *Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences*. In: *SAE 2005 World Congress and Exhibition, April 2005, Detroit, MI, USA*, 2005.

[PJ04] PICKIN, SIMON und JEAN-MARC JÉZÉQUEL: *Using UML Sequence Diagrams as the Basis for a Formal Test Description Language*. In: *Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK,*

Band 2999 der Reihe *Lecture Notes in Computer Science*, Seiten 481–500. Springer, 2004.

[PP04] PRENNINGER, WOLFGANG und ALEXANDER PRETSCHNER: *Abstractions for Model-Based Testing*. In: *Proceedings Test and Analysis of Component-based Systems*, 2004.

[PP05] PILONE, DAN und NEIL PITMAN: *UML 2.0 in a Nutshell*. O'Reilly, 2005.

[Pre03] PRETSCHNER, WALTER ALEXANDER: *Zum modellbasierten funktionalen Test reaktiver Systeme*. Dissertation, Technische Universität München, München, 2003.

[PvV98] POL, MARTIN und ERIK VAN VEENENDAAL: *Structured Testing of Information Systems*. Kluwer, 1998.

[Rei06] REIF, KONRAD: *Automobilelektronik*. Vieweg, 2006.

[RGTL09] ROBERT, SYLVAIN, SÉBASTIEN GÉRARD, FRANÇOIS TERRIER und FRANÇOIS LAGARDE: *A Lightweight Approach for Domain-Specific Modeling Languages Design*. In: *SEAA '09: Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, 2009.

[Ros01] ROSENBERGER, RALF: *Zur Generierung von Verhaltensmodellen für gemischt analog/digitale Schaltungen auf Basis der Theorie dynamischer Systeme*. Doktorarbeit, Technische Universität Darmstadt, Oktober 2001.

[Roy87] ROYCE, W. W.: *Managing the development of large software systems: concepts and techniques*. In: *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, Seiten 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[RR86] REED, GEORGE M. und A. W. ROSCOE: *A Timed Model for Communicating Sequential Processes*. In: KOTT, LAURENT (Herausgeber): *ICALP*, Band 226 der Reihe *Lecture Notes in Computer Science*, Seiten 314–323. Springer, 1986.

[RS09] REINER, JÜRGEN und MARCUS SCHAPER: *Enorme Sparpotenziale in der Fahrzeugelektronik*. AUTOMOBIL ELEKTRONIK, 5 2009.

[Rup07] RUPP, CHRIS: *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser Fachbuchverlag, 2007.

[SA98] SIMMONS, REID und DAVID APFELBAUM: *A task description language for robot control*. In: *in Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 1998.

[Sax08] SAX, ERIC: *Automatisiertes Testen Eingebetteter Systeme in der Automobilindustrie*. Carl Hanser Verlag, 2008.

[SBG06] SCHIEFERDECKER, INA, ECKARD BRINGMANN und JÜRGEN GROSSMANN: *Continuous TTCN-3: testing of embedded control systems*. In: *SEAS*

*'06: Proceedings of the 2006 international workshop on Software engineering for automotive systems*, Seiten 29–36, New York, NY, USA, 2006. ACM Press.

[Sch98] SCHWITTER, ROLF: *Kontrolliertes Englisch für Anforderungsspezifikationen*. Doktorarbeit, Institut für Informatik (IFI) der Universität Zürich, 1998.

[Sch01a] SCHIENMANN, BRUNO: *Kontinuierliches Anforderungsmanagement: Prozesse- Techniken- Werkzeuge*. Pearson Education, 2001.

[Sch01b] SCHÖNING, UWE: *Theoretische Informatik - kurzgefasst.* Spektrum-Akademischer Vlg, March 2001.

[Sch03] SCHMID, HERMANN: *Konzeption einer pragmatischen Testmethodik für den Test von eingebetteten Systemen*. Doktorarbeit, Universität Ulm, 2003.

[Sch07] SCHOLZ, PETER: *Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung (Xpert.press)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[SCS97] SINGH, HARBHAJAN, MIRKO CONRAD und SADEGH SADEGHIPOUR: *Test Case Design Based on Z and the Classification-Tree Method*. In: *ICFEM '97: Proceedings of the 1st International Conference on Formal Engineering Methods*, Seite 81, Washington, DC, USA, 1997. IEEE Computer Society.

[SG03] SCHIEFERDECKER, INA und DAI GRABOWSKI: *The UML 2.0 Testing Profile and its Relation to TTCN-3*, 2003.

[Slo02] SLOANE, A.: *Post-design Domain-Specific Language Embedding: A Case Study in the Software Engineering Domain*. In: *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, 2002.

[Spi01] SPITZER, BERNHARD: *Modellbasierter Hardware-in-the-Loop Test von eingebetteten elektronischen Systemen*. Doktorarbeit, Institut für Technik der Informationsverarbeitung (ITIV), 2001.

[Sre02] SREENIVAS, ASHOK: *Towards formal test specifications*. In: HANEBERG, DOMINIK, GERHARD SCHELLHORN und WOLFGANG REIF (Herausgeber): *Proceedings, Workshop on Tools for System Design and Verification (FM-TOOLS), Reisensburg, Germany*, Nummer 2002–11 in *Technical Report*, Seiten 75–80. Universität Ulm, Fakultät für Informatik, Juli 2002.

[SZ04] SCHÄUFFELE, JÖRG und THOMAS ZURAWKA: *Automotive Software Engineering*. Vieweg, October 2004.

[Tab07] TABELING, PETER: *Softwaresysteme und ihre Modellierung: Grundlagen, Methoden und Techniken*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[TD09] TRIPAKIS, STAVROS und THAO DANG: *Modeling, Verification and Testing using Timed and Hybrid Automata*. In: *Model-Based Design of Heterogeneous Embedded Systems*. CRC Press, 2009.

[Tes09] TESTAUSTAUSCHFORMAT, HIS-TAF EXPERTENGRUPPE: *Anforderungen an ein standardisiertes Testaustauschformat - TAF*. Technischer Bericht 1.1, HIS-TAF Expertengruppe Testaustauschformat, 2009.

[Tet09] TETZNER, ELKE: *Nutzerfreundliche Modellierung mit hybriden Systemen zur symbolischen Simulation in CLP*. Doktorarbeit, Universität Rostock, Fakultät für Informatik und Elektrotechnik, 2009.

[Unb08] UNBEHAUEN, HEINZ: *Regelungstechnik I. Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*. Vieweg Verlag, 15. Aufl. Auflage, October 2008.

[UPL06] UTTING, MARK, ALEXANDER PRETSCHNER und BRUNO LEGEARD: *A taxonomy of model-based testing*. Working Paper, April 2006.

[vDKV00] DEURSEN, ARIE VAN, PAUL KLINT und JOOST VISSER: *Domain-Specific Languages: An Annotated Bibliography*. ACM SIGPLAN Notices, 35:26–36, 2000.

[vL00] LAMSWEERDE, AXEL VAN: *Formal specification: a roadmap*. In: *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, Seiten 147–159, New York, NY, USA, 2000. ACM.

[Vol09] VOLLMER, ALFRED: *Gerüstet für Driver-in-the-Loop*. AUTOMOBIL ELEKTRONIK, 10 2009.

[WB05] WÖRN, HEINZ und UWE BRINKSCHULTE: *Echtzeitsysteme*, Band 1 der Reihe *eXamen-press*. Springer Verlag, Institut für Prozessrechentechnik, Automation und Robotik, Universität Karlsruhe (TH), 1 Auflage, Februar 2005.

[Wil06] WILDEMANN, HORST: *In- und Outsourcingstrategien in der Automobil- und -Zuliefererindustrie*. In: *Innovative Kooperationsnetzwerke*. DUV, 2006.

[WKK09] WACHTEL, EUGEN, MARCO KUHRMANN und GEORG KALUS: *A Domain Specific Language for Project Execution Models*. In: FISCHER, STEFAN, ERIK MAEHLE und RÜDIGER REISCHUK (Herausgeber): *Beiträge der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Seiten 2986–3000. Gesellschaft für Informatik e.V. (GI), 2009.

[WR06] WALLENTOWITZ, HENNING und KONRAD REIF (Herausgeber): *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen*. Vieweg, Wiesbaden, 2006.

[WRDZ07] WIESE, DIPL.-ING. MATTHIAS, PROF. DR. HANS-CHRISTIAN REUSS, DR. RÜDIGER DORN und DR. ROLF ZÖLLER: *Systematischer Test modellbasiert entwickelter Steuergeräte*. 26. Treffen der GI-Arbeitsgruppe

"Test, Analyse und Verifikation von Software" (TAV) der Gesellschaft für Informatik (GI), 2007.

[WWD+97] WEIKUM, GERHARD, DIRK WODTKE, ANGELIKA KOTZ DITTRICH, PETER MUTH und JEANINE WEISSENFELS: *Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR.* Inform., Forsch. Entwickl., 12(2):61–71, 1997.

[Wym07] WYMAN, OLIVER: *Innovationsmanagement in der Automobilindustrie*. Technischer Bericht, Oliver Wyman, 2007.

[Wüs63] WÜSTENECK, K. D.: *Zur philosophischen Verallgemeinerung und Bestimmung des Modellbegriffes*. Deutsche Zeitschrift für Philosophie, Heft 12, 1963.

[xte09] *Xtext User Guide*, 2008 - 2009.

[Zei84] ZEIGLER, BERNARD P.: *Multifacetted modelling and discrete event simulation*. Academic Press, New York, 1984.

[ZEKB09] ZIMMERMANN, FABIAN, ROBERT ESCHBACH, JOHANNES KLOOS und THOMAS BAUER: *Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems*. In: *12th European Workshop on Dependable Computing, EWDC 2009, Toulouse, France*, 2009.

[Zha08] ZHAO, YIPING: *Evaluierung der Testbeschreibungsmittel auf ihre Eignung im funktionalen Steuergerät-Test*. Master's Thesis (Diplomarbeit), Technische Universität Ilmenau, 2008.

[ZS09] ZDUN, U. und M. STREMBECK: *Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Development*. In: *Proc. of the 14th European Conference on Pattern Languages of Programs (EuroPLoP)*, Irsee Monastery, Germany, July 2009.

# Lebenslauf

## *Persönliche Daten*

| | |
|---|---|
| Geburtsdatum: | 24. Juni 1976 |
| Geburtsort: | Nürnberg |
| Familienstand: | ledig |
| Staatsangehörigkeit: | deutsch |

## *Schulbildung*

| | |
|---|---|
| 1982 – 1986 | Grundschule Fürth |
| 1986 – 1987 | Grund- und Hauptschule in Fürth |
| 1987 – 1997 | Dürer Gymnasium in Nürnberg |

## *Studium*

| | |
|---|---|
| 1997 – 1999 | Studium der BWL an der FAU Erlangen-Nürnberg |
| 1999 – 2006 | Studium der Informatik an der FAU Erlangen-Nürnberg |
| 2006 – 2010 | Promotion an der TU Kaiserslautern |

## *Fremdsprachen*

| | |
|---|---|
| Englisch | gut |
| Türkisch | Muttersprache |

## *Kenntnisse*

| | |
|---|---|
| Betriebssysteme | Linux, Windows |
| Programmiersprachen | C++, Python, C, PHP, Shell-Skript Programmierung, HTML |
| Datenbanken | MySQL, PostgreSQL |

## *Berufliche Tätigkeiten*

| | |
|---|---|
| 2006 – 2010 | Test-Ingenieur bei der MB-technology GmbH |