**Christopher Guggi**

# Konzeption einer objektorientierten Methodik zur transparenten Verknüpfung von kundenrelevanten Fahrzeugeigenschaften und Bauteileigenschaften

Diplomarbeit

Technische Universität Graz
Institut für Fahrzeugtechnik

Betreuer:
Univ.-Doz. Dr. techn. Mario Hirz
Dipl.-Ing. Johannes Mayr

Graz, 2014

Deutsche Fassung:

Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008

Genehmigung des Senates am 1.12.2008

## EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..

(Unterschrift)

Englische Fassung:

## STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..

date (signature)

## Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit der Konzeption einer objektorientierten Methodik zur transparenten Verknüpfung von
kundenrelevanten Fahrzeugeigenschaften und Bauteileigenschaften.
Zu Beginn dieser Arbeit werden die für die Umsetzung benötigten theoretischen Grundlagen aus dem Konfigurations- und Anforderungsmanagements und der Datenbanksysteme beschrieben. Im Anschluss daran folgt der Entwurf einer objektorientierten Methodik zur Lösung der Aufgabenstellung.
Mittels der generierten Methodik kann der hierarchische Aufbau des Fahrzeuges dargestellt werden. Durch den objektorientierten Ansatz können mit Hilfe von Klassen für die Fahrzeugelemente sämtliche Elemente Instanziiert werden. In einer weiteren, durch Vererbung der Fahrzeugelementsklasse erzeugten, generischen Visual Basic System Collection Klasse werden diese durch Zuweisung an die Elternknoten gesammelt.
Auf dieselbe Weise werden die Fahrzeugeigenschaften verwaltet. Neben der allgemeinen Klasse zum Sammeln aller Eigenschaften gibt es jedoch eine weitere, wo ebenfalls durch Vererbung der Eigenschaftsklasse die Abhängigkeiten definiert und gesammelt werden können.
In einer Klasse zur Verknüpfung von Fahrzeugelementen mit Eigenschaften werden die Instanziierten Elemente hinsichtlich ihres Einflusses und des damit verbundenen Erfüllungsgrades auf die Eigenschaften bewertet. Eine Collection davon wird wieder über Vererbung dieser Klasse und einer generischen System Collection angelegt. Mit den Bewertungsdaten in dieser Collection können sämtliche Analysen durchgeführt werden. Diese können Informationen über besonders einflussreiche Fahrzeugelemente liefern, bzw. wo Änderungen große Auswirkungen auf Kundeneigenschaften haben. Aber auch ein Variantenvergleich, zum Beispiel durch den Einsatz verschiedener Werkstoffe oder Technologien, kann damit erstellt werden. In einer Evaluierungsklasse werden die verschiedenen Funktionen zur Analyse gesammelt, wobei durch die Datensammlung in einer generischen Liste leicht weitere, je nach Bedarfsfall benötigte, Funktionen implementiert werden.

## Abstract

The present thesis deals with the conception of an object-oriented methodology for the transparent linking of customer-relevant vehicle properties with vehicle components.

At the beginning of this work, the required theoretical foundations from the configuration- and requirements management and from database systems are described. This introduction follows the design of an object-oriented methodology, providing a solution of the problem.

With this new method, the hierarchical structure of a vehicle can be recreated. All elements can be instantiated by using a class of vehicle elements.

They are collected by assignments to the parent node in a generic Visual Basic System Collection class, which was generated by inheritance of the vehicle element class.

In the same way, the properties of a new vehicle concept can be managed. In addition to the general property- and collecting class there are two further classes, which enable a definition of dependencies and a collection by inheritance of the property class.

In a class for linking vehicle elements with properties the instantiated elements can be evaluated in terms of their influence and the associated performance level on the properties. A collection of them is created as a function of inheritance of this class in a generic system collection. Analyses can be performed based on the data in this collection.

For example, data provide information about the vehicle elements, which have a major impact on certain customer characteristics. In addition, a comparison of different vehicle versions, for example by use of different materials or technologies, can be created. Finally, an evaluation class includes various analyzing functions.

# Inhaltsverzeichnis

# 1 Einleitung

Die subjektive Wahrnehmung von Fahrzeugeigenschaften ist ein wesentliches Kriterium für einen Fahrzeugkauf. Mittels einer transparenten Methodik soll von den Kundenanforderungen ausgehend ein Zusammenhang mit den Fahrzeugelementen hergestellt werden.
Die Methodik soll dazu dienen, diese Vielzahl von sich gegenseitig beeinflussenden Zusammenhängen zu verwalten, zu dokumentieren und zu gewichten. Es soll erkennbar werden, wie sich beispielweise Änderungen in der Verbindungtechnologie oder Änderungen bei der Werkstoffwahl auf Eigenschaften auswirken. Verschiedene Varianten von Fahrzeugen sollen dadurch anhand ihrer spezifischen Eigenschaften vergleichbar werden. Durch einen Bewertungsvorgang soll dies und eine Identifikation von besonders einflussreichen Fahrzeugelementen ermöglicht werden.

Durch die Kenntnis von besonders einflussreichen Bauteilen auf die Kundeneigenschaften sowie von den Beziehungen der Eigenschaften untereinander und der Möglichkeit die Auswirkungen verschiedener technischer Einflussfaktoren zu erkennen, kann die Erreichung einer möglichst hohen Befriedigung der Kundenbedürfnisse unterstützt werden.

# 2 Konfigurationsmanagement (in Anlehnung an Versteegen (2003) und Popp (2013))

## 2.1 Grundlagen

Konfigurationsmanagement enthält technische und administrative Vorschriften, um

- Konfigurationseinheiten zu identifizieren und deren physikalische und funktionale Eigenschaften zu dokumentieren
- Änderungen dieser Eigenschaften zu steuern
- Änderungs- und Implementierungsstatus aufzuzeichnen und veröffentlichen
- die Erfüllung der spezifizierten Anforderungen zu verifizieren

Dies ist Definition laut CMMI (Capability Maturity Model Integration) der Carnegie Mellon University Pittsburgh. Mit Hilfe des CMMI Modells sollen Unternehmen ihre Prozesse bewerten und optimieren können.
Als Konfigurationseinheit ist hierbei eine Kombination aus Hardware, Software, Dienstleistung oder jede andere Unterteilung davon zu verstehen auf die Konfigurationsmanagement angewendet werden soll (Versteegen, 2003, S.5).

ISO 9000 enthält keine Definition für Konfigurationsmanagement, aber Anforderungen welche denen des Konfigurationsmanagement entsprechen (Versteegen, 2003, S.6):

- „… der Lieferant muss alle Verfahren dokumentieren und dafür sorgen, dass das Design der Erzeugnisse den Anforderungen entspricht.
- …sämtliche Dokumente müssen vor ihrer Verwendung geprüft werden. Die aktuell gültigen Versionsstände der Dokumente sind in einer Stammliste einzutragen.
- …Änderungen an Dokumenten müssen geprüft und freigegeben werden. …“

DIN EN ISO 10007:1996 ist ein Leitfaden für Konfigurationsmanagement und die Definition dafür lautet (Versteegen, 2003, S.6):
„Konfigurationsmanagement ist eine Managementdisziplin, die über die gesamte Lebensdauer eines Erzeugnisses angewandt wird, um Transparenz und Überwachung seiner funktionellen und physischen Merkmale sicherzustellen. Hauptziel von Konfigurationsmanagement ist, die gegenwärtige Konfiguration eines Erzeugnisses, sowie den Stand der Erfüllung seiner physischen und funktionellen Forderungen zu dokumentieren und volle Transparenz herzustellen. Ein weiteres Ziel ist, dass jeder am Projekt mitwirkende zu jeder Zeit des Erzeugnislebenslaufs die richtige und zutreffende Dokumentation verwendet. Der Konfigurationsmanagement Prozess umfasst die folgenden Tätigkeiten:

- Konfigurationsidentifizierung,
- Konfigurationsüberwachung,
- Konfigurationsbuchführung,
- Konfigurationsauditierung.

Es gibt keine einheitliche Definition für das Konfigurationsmanagement, aber die meisten Beschreibungen beinhalten diese Tätigkeiten.

Die Konfigurationsidentifizierung umfasst folgende Tätigkeiten (Versteegen, 2003, S. 6ff):

- Definition der Erzeugnisstruktur
- Auswahl von Konfigurationseinheiten
- Dokumentation physischer und funktioneller Merkmale der Konfigurationseinheiten
- Aufstellung und Verwendung von Regeln zur Bezeichnung der Konfigurationseinheiten, deren Elemente und Zusammenstellungen von Dokumenten, Schnittstellen, Änderungen und Freigaben vor und nach der Realisierung

- Mittels formalisierter Vereinbarungen Einrichten von Bezugskonfigurationen. Diese bilden die aktuell gültige Konfiguration, bei Genehmigung von vorhandenen Änderungen.

Alle Änderungen sollten aber bei der erstmaligen Freigabe überwacht werden, wobei die Änderungen in einem Änderungsverfahren dokumentiert werden sollten. Diese Tätigkeiten fallen in die Konfigurationsüberwachung:

- Dokumentation von Änderungen
- Begründung von Änderungen
- Beurteilung der Auswirkungen der Änderungen
- Genehmigung oder Ablehnung der Änderung

Die Rückverfolgung von Änderungen auf die letzte Bezugskonfiguration ermöglicht die Konfigurationsbuchführung. Die dafür erforderlichen Aufzeichnungen sollten als Nebenprodukt der Identifizierung und Überwachung anfallen.

Damit die Erfüllung der vertraglich festgelegten Anforderungen sichergestellt werden kann, sollte vor der Annahme einer Bezugskonfiguration ein Konfigurationsaudit durchgeführt werden. Man kann zwischen zwei Arten unterscheiden:

- Funktionsbezogenes Konfigurationsaudit: Formale Prüfung einer Konfigurationseinheit, ob die in den Konfigurationsdokumenten festgelegten Anforderungen erreicht werden.
- Physisches Konfigurationsaudit: Formale Prüfung ob die aktuelle Konfiguration einer Konfigurationseinheit ihren Konfigurationsdokumenten entspricht.

Teilprozesse des Konfigurationsmanagements sind:

- Versionsverwaltung
- Konfigurationsverwaltung
- Releasemanagement

- Änderungsmanagement
- Buildmanagement
- Distributionmanagement

## 2.2 Ziele

Konfigurationsmanagement ist eine Sammlung von Richtlinien. Mit Hilfe dieser Richtlinien wird ein Prozess beschrieben, dessen Ziel die Optimierung und Regelung der Zusammenarbeit eines Teams ist. Im Großen und Ganzen kann dies in vier große Ziele zusammenfassen (Abbildung 1).

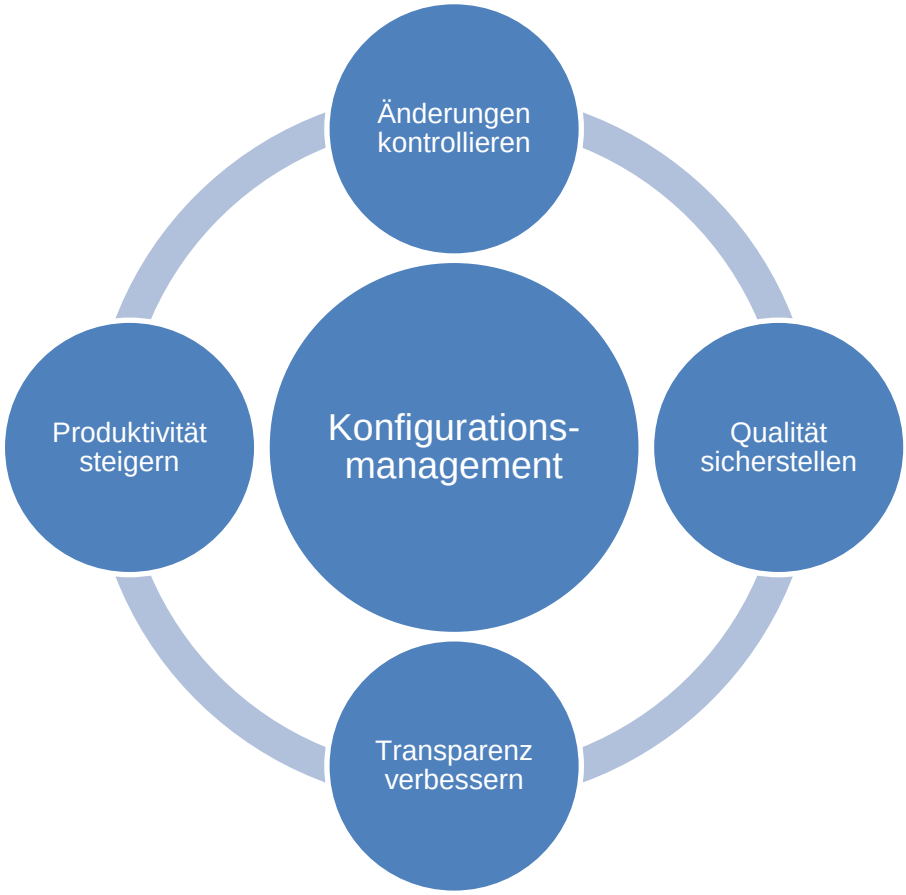


**Abbildung 1: Ziele des Konfigurationsmanagements (in Anlehnung an Popp, 2013, S.11)**

### 2.2.1 Änderungen kontrollieren

Bei einer nicht funktionierenden Kommunikation geraten Änderungen leicht außer Kontrolle. Solange Änderungen vor der Umsetzung mit allen Beteiligten besprochen werden, treten meist keine Schwierigkeiten auf. Je größer das Projekt bzw. die Anzahl der daran beteiligten Personen ist, umso schwieriger wird diese permanente

gegenseitige Abstimmung. Der Kommunikationsaufwand steigt überproportional mit jedem neuen Teammitglied, unter der Annahme, dass jeder irgendwann mit jedem kommuniziert (Abbildung 2).

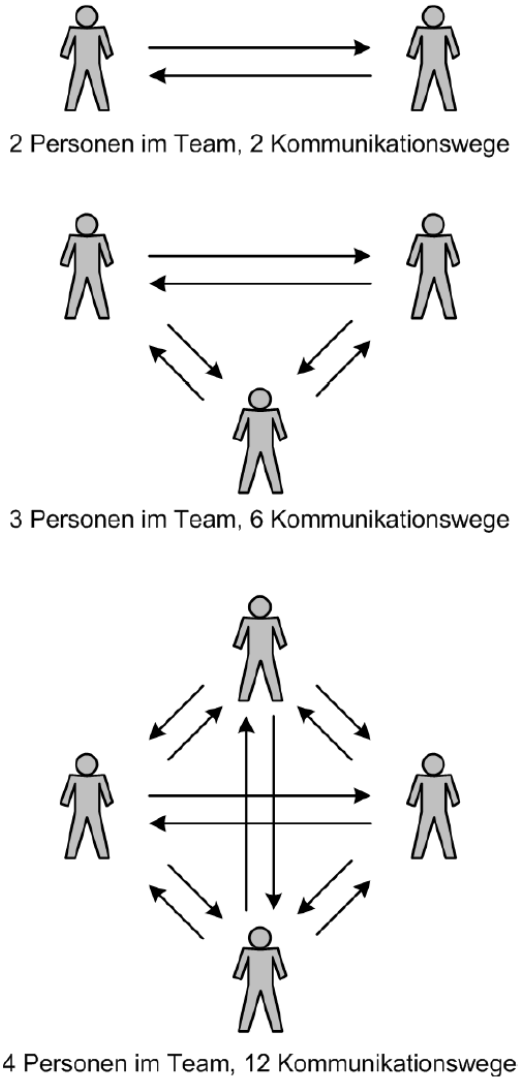


**Abbildung 2: Abstimmungsaufwand (Popp, 2013, S.12)**

Infolge dessen finden schon in Teams ab vier Personen Abstimmungen nicht oder nur unvollständig statt. Mithilfe des Konfigurationsmanagementprozesses bzw. dem Änderungsmanagement sollt diese Situation durch eine vereinbarte teaminterne Kommunikation verhindert werden.

Weiters soll damit neben der Vereinfachung der Kommunikation auch sichergestellt werden, dass die richtigen Änderungen vorgenommen werden und diese auch nachvollziehbar sind. Durchgeführte Änderungen sollen unter keinen Umständen

verloren gehen, z.B. bei paralleler Änderung eines Elementes durch mehrere Bearbeiter.

### 2.2.2 Qualität sicherstellen

Durch die resultierende Fehlervermeidung, Projektautomatisierung und dss Änderungsmanagement wird die Produktqualität verbessert und das erreichte Qualitätsniveau leichter dauerhaft gehalten.
Die Fehlervermeidung soll durch präventive Maßnahmen erreicht werden. Rein prozessbedingte Fehler, z.B. aufgrund einer falsch konfigurierten Testumgebung, können durch die Optimierung des Entwicklungsprozesses im Rahmen des Konfigurationsmanagement reduziert werden. Die durch das Konfigurationsmanagement veranlasste Durchführung regelmäßiger Tests, sowie deren automatische Auswertung, sind wirksame Mahnahmen zur Fehlervermeidung.
Ein Aspekt der dauerhaften Qualitätssicherung ist die mittels der Projektautomatisierung jederzeit zuverlässige und wiederholbare Herstellung der Produkte.

### 2.2.3 Produktivität steigern

Eine einfache Möglichkeit die Produktivität zu steigern ist, den Fokus der Teammitglieder soweit wie möglich auf ihre jeweiligen Tätigkeiten zu lenken. Alle anderen Tätigkeiten sollten so weit wie möglich reduziert werden. Konfigurationsmanagement unterstützt dies auf verschiedenste Weisen, z.B. mit der Versions- und Konfigurationsverwaltung wird eine übersichtliche Struktur geschaffen, wodurch zeitraubende Rückfragen im Team vermieden werden.

### 2.2.4 Transparenz verbessern

Transparenz wird vor allem mit dem Änderungsmanagement geschaffen. Für die Projektleitung ist damit leicht ersichtlich, welche Änderungen bereits durchgeführt wurden oder an welchen gerade gearbeitet wird. Der Ist-Stand des Projektes ist damit abrufbar. Diese Informationen lassen sich auf das Anforderungsmanagement übertragen, wobei Aussagen zur Erfüllung der Anforderungen möglich sind und ob Änderungen im Projekt für eine vollständige Erfüllung erforderlich sind.

## 2.3 Versionsverwaltung

Schon am Anbeginn der Softwareentwicklung zeigte sich, dass es vorteilhaft ist, nicht nur den aktuellen Stand sondern auch ältere zu speichern. Denn Software wird oft nach dem „Trial & Error- Prinzip“ entwickelt, weshalb nach erfolglosen Änderungen oft auf eine alte Version zurückgegriffen wird.
Mittels der Versionsverwaltung werden nach der Bearbeitung die neue als auch die alte Version gespeichert, wobei diese Verwaltung aber nur Versionen voneinander unabhängiger Konfigurationseinheiten betrifft. Meist werden jedoch Konfigurationen aus einer Vielzahl von Konfigurationseinheiten gebildet und sollten daher in Gruppen gegliedert werden. Diese Gruppierung erfolgt in der Konfigurationsverwaltung.

Vorteile der Versionierung:

- Wiederherstellung eines alten Standards
- Wiederherstellung bei irrtümlichem Löschen
- Vergleich zweier Versionen um festzustellen, welche Änderungen durchgeführt wurden
- Paralleles arbeiten an zwei unterschiedlichen Versionen der selben Konfiguration

## 2.4 Definition der Konfigurationseinheiten

Um den Konfigurationsmanagementprozess einzuführen ist als erster Schritt die Definition der Konfigurationseinheiten erforderlich. Eine Konfigurationseinheit kann eine Konfiguration, d.h. ein komplettes Subsystem, oder auch nur ein einzelner Bauteil sein. Alle Instanzen der Konfigurationseinheiten zusammen ergeben die Konfiguration bzw. das zu erstellende Produkt. Der simpelste Fall wäre, dass mehrere Konfigurationseinheiten eine Konfiguration bilden. Falls dies nicht ausreichend ist, wie es häufig der Fall ist, dann ist das Bilden einer hierarchischen Struktur notwendig (Abbildung 3). Alle anderen Projektbestandteile, die nicht dem Produkt zuordenbar sind, fallen nicht unter die Konfigurationseinheiten, wie z.B. Hilfsmittel zur Projektdurchführung wie Konstruktionszeichnungen und Pläne. Mittels eines Auswahlprozesses erfolgt die Festlegung, welche Dokumente unter das Konfigurationsmanagement fallen und welche nicht. Währenddessen erfolgt auch eine sinnvolle Strukturierung.

Am Start des Projektes kann dieser Prozess als erster Entwurf gesehen werden. Im weiteren Projektverlauf wird bei Einführung neuer Elemente dieser Prozess, nach einer Überprüfung der Auswirkungen auf das Konfigurationsmanagement, eventuell erneut ausgeführt werden.

Die Wichtigkeit dieser Auswahl von Konfigurationseinheiten ist von sehr großer Bedeutung.

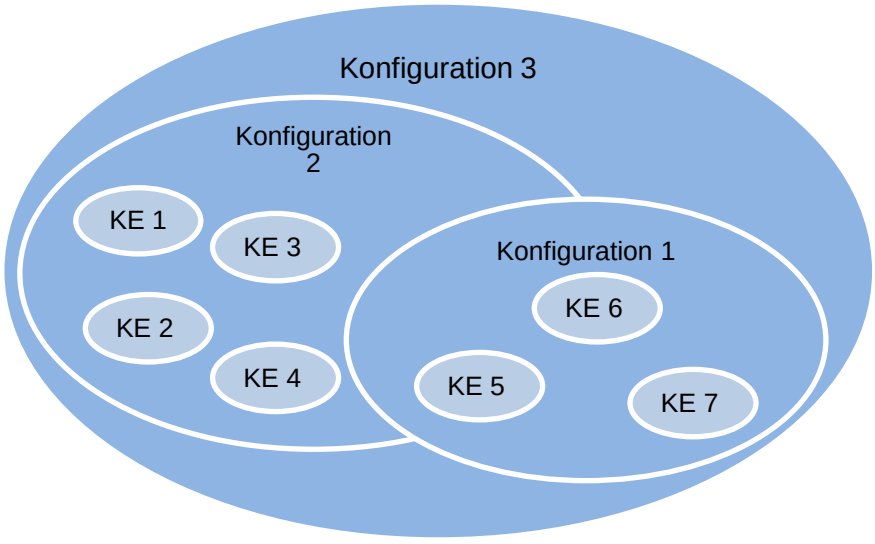


**Abbildung 3: Zusammenhang Konfigurationen-Konfigurationseinheiten (in Anlehnung an Versteegen, 2003, S.11)**

Bei einer unvollständigen Auswahl der wichtigen Teile des Produktes kann eine vollständige Auslieferung nicht garantiert werden. Werden hingegen unnötige Teile hinzugefügt, dann erzeugt dies einen unnötigen Aufwand welcher durch Konfigurationsmanagement eigentlich vermieden werden soll.

## 2.5 Konfigurationsmanagementplan

Nach der Erfassung aller Konfigurationseinheiten erfolgt deren Dokumentation. Die Dokumentation sollte alle Entscheidungen beinhalten, welchen den Konfigurationsmanagementprozess des Projektes betreffen.
Laut Standard Institute of Electrical and Electronics Engineers (IEEE-828-1998) sollte der Plan aus folgenden Abschnitten bestehen:

- “Einleitung: Erläutert die Ziele des Plans und die wichtigsten Begriffe
- Management: Beschreibt, wer für welche Konfigurationsmanagementaktivitäten verantwortlich ist, und wie die Einbindung anderer Organisationseinheiten erfolgt
- Aktivitäten: Dokumentiert die zentralen Aufgaben und Verfahren des Konfigurationsmanagements
- Zeitplanung: Legt die zeitliche Abfolge der Konfigurationsmanagementaktivitäten fest
- Ressourcen: Dokumentiert den Bedarf an Werkzeugen und Personal
- Pflege: Beschreibt, in welcher Form der Plan während eines laufenden Projektes geändert werden kann, z.B. um auf geänderte Anforderungen zu reagieren“

Der Aufwand der Erstellung eines Konfigurationsmanagementplans fällt als Gesamtes meist nur im ersten Projekt an. Große Teile davon können in Folgeprojekten wiederverwendet werden.

## 2.6 Beschreibung der Konfigurationseinheiten

Die minimal vorhandenen Informationen einer Konfigurationseinheit im Konfigurationsplan sind deren Name sowie eine kurze inhaltliche Beschreibung. Weitere dokumentierte Informationen zu den Eigenschaften der Einheit sind durchaus sinnvoll.
Die inhaltliche Beschreibung sollte nur aus ein, zwei bis drei Sätzen bestehen. Jedes Projektmitglied sollte Sinn und Zweck der Konfigurationseinheit schnell erfassen können.
Der Name jeder Konfigurationseinheit sollte folgende Anforderungen erfüllen:

- Jede Instanz der Konfigurationseinheit muss anhand ihres Namens eindeutig identifizierbar sein. Beispielsweise bietet sich in größeren Projekten die Einführung einer fortlaufenden Nummer für bestimmte Konfigurationseinheiten an.
- Der Name der Konfigurationseinheit sollte auf die direkt übergeordnete Konfiguration oder die übergeordnete Konfigurationseinheit hinweisen.
- Aus dem Namen der Konfigurationseinheit sollten Beziehungen zwischen anderen Elementen ersichtlich sein.

Weitere sinnvolle Beschreibungen könnten z.B. hinsichtlich Anforderungen, Konfigurationsdaten sowie Werkzeugen angeführt werden.

## 2.7 Definition der Projektstruktur

Nach der Beschreibung der Konfigurationseinheiten erfolgt die Festlegung der Projektstruktur. Laut Conway’s Law hat die Struktur der Projektorganisation starken Einfluss auf die entwickelte Systemstruktur. Sobald ein Projekt auf mehrere Abteilungen aufgeteilt wird, erfolgt die Abgrenzung der Beteiligten. Jeder möchte sich vor dem Einfluss anderer Teilbereiche abschirmen. Dadurch entstehen Subsysteme, Komponenten und Schnittstellen, welche sich nicht an den Erfordernissen oder an

der Funktionalität einer sauberen Architektur orientieren, sondern an der Projektorganisation. Sind bei der Umsetzung eines Projektes drei Abteilungen beteiligt, so findet sich diese Randbedingung in der Form von drei künstlich zurechtgeschnittenen Subsystemen unter Umständen wieder auch wenn auch aus Architektursicht zwei Subsysteme die optimale Lösung gewesen wäre.
Durch organisatorische Maßnahmen kann hier im Projekt Abhilfe geschaffen werden, wobei die Verantwortlichkeiten am Inhalt und nicht am Umfeld ausgerichtet werden (Popp, 2013, S. 31f).

## 2.8 Verwaltung der Konfigurationseinheiten

Die Verwaltung der Konfigurationseinheiten ist eine der zentralen Rollen des Konfigurationsmanagements. Eine kontrollierte Durchführung und Nachvollziehbarkeit aller Änderungen soll damit sichergestellt werden.
In einer Softwarebibliothek werden alle Instanzen der Konfigurationen und Einheiten gespeichert. Die ist die zentrale Ablage des Projektes worauf alle Beteiligten Zugriff haben. In dieser Bibliothek müssen alle identifizierten Konfigurationseinheiten verwaltet werden.
Im Konfigurationsmanagementprozess muss diese Bibliothek folgende Datensicherheitsanforderungen gewährleisten:

- Sicherstellung der Datenverfügbarkeit
- Gewährleistung der Integrität, vor allem wenn mehrere Benutzer gleichzeitig Änderungen durchführen
- Nachvollziehbarkeit aller durchgeführten Änderungen, sowie Zurücknahme ungewollter Änderungen.

Die Konfigurationsverwaltung befasst sich auch mit der Definition von Rollen und Verantwortlichkeiten:

- Berechtigung zur Erstellung neuer Konfigurationseinheiten

- Berechtigung zur Gruppierung von Konfigurationseinheiten
- Berechtigung der Änderung an Konfigurationseinheiten
- Berechtigung der Änderung an Konfigurationen

## 2.9 Releasemanagement

Die inhaltliche und zeitliche Planung der Auslieferungszustände wird im Releasemanagement geregelt. Unter Baselining wird im Releasemanagement das Einfrieren der Releasekonfiguration verstanden, auch als Erstellen von Bezugskonfigurationen benennbar. Damit wird das Nachvollziehen der Konfigurationen und Konfigurationseinheiten der Releases zu einem späteren Zeitpunkt sichergestellt.

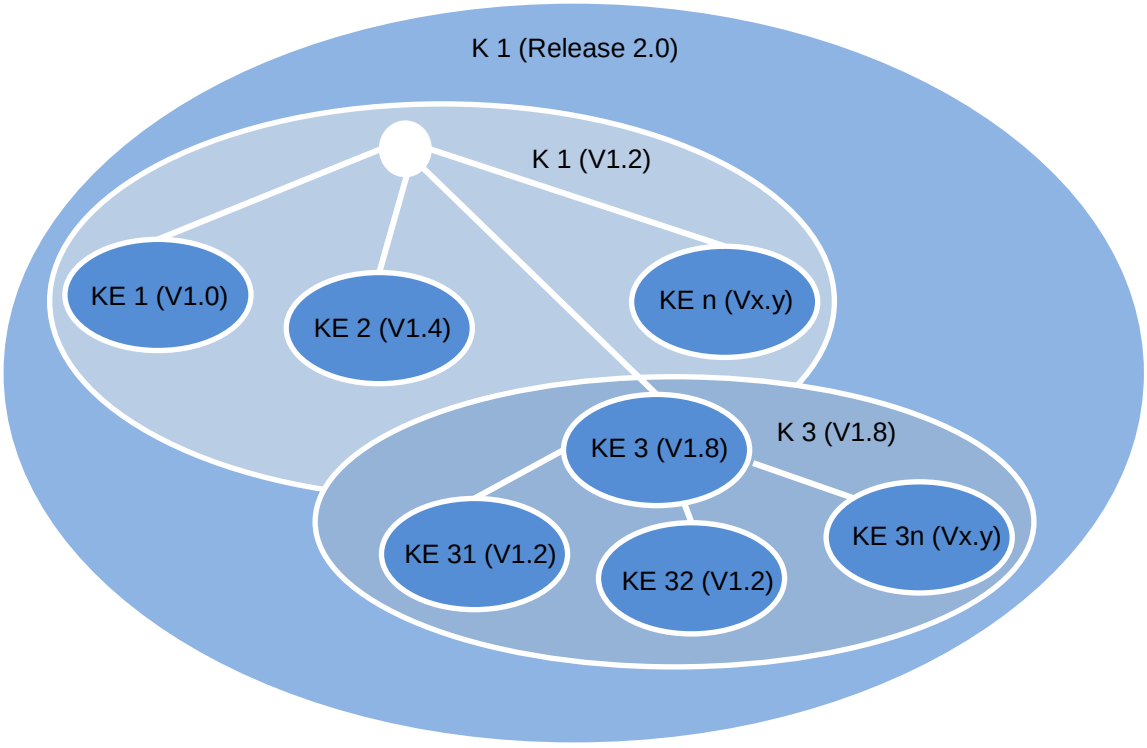


**Abbildung 4: Releasebestandteile (in Anlehnung an Versteegen, 2003, S.12)**

Die Gesamtkonfiguration K 1 im Release 2.0 (Abbildung 4) besteht aus der Konfiguration K 1 der Version 1.2 und K 3 V1.8. Beim Einfrieren des Releases werden alle Elemente der Konfigurationen mit einer Releasekennung versehen, in diesem Beispiel „Release 2.0“. Somit kann ein Release eindeutig nachvollzogen werden, und es lässt sich sofort erkennen in welchen Releases eine bestimmte Version einer Konfigurationseinheit verwendet wurde.

## 2.10 Änderungsmanagement

Das Änderungsmanagement befasst sich mit der Umsetzung definierter Prozesse zur Änderung von Konfigurationen und Konfigurationseinheiten. Der einfachste Änderungsprozess beinhaltet, dass jede Person zu jedem beliebigen Zeitpunkt Änderungen einarbeiten kann. Aufgrund der Aufbewahrung aller alten Versionen sind die Änderungen zwar nachvollziehbar, aber unter Änderungsmanagement wird mehr verstanden. Änderungen sollen über Änderungsanträge in den Änderungsprozess eingebunden werden. Jeder Antrag ist von der darüber zu entscheidenden Stelle zu prüfen und zu genehmigen oder abzulehnen.

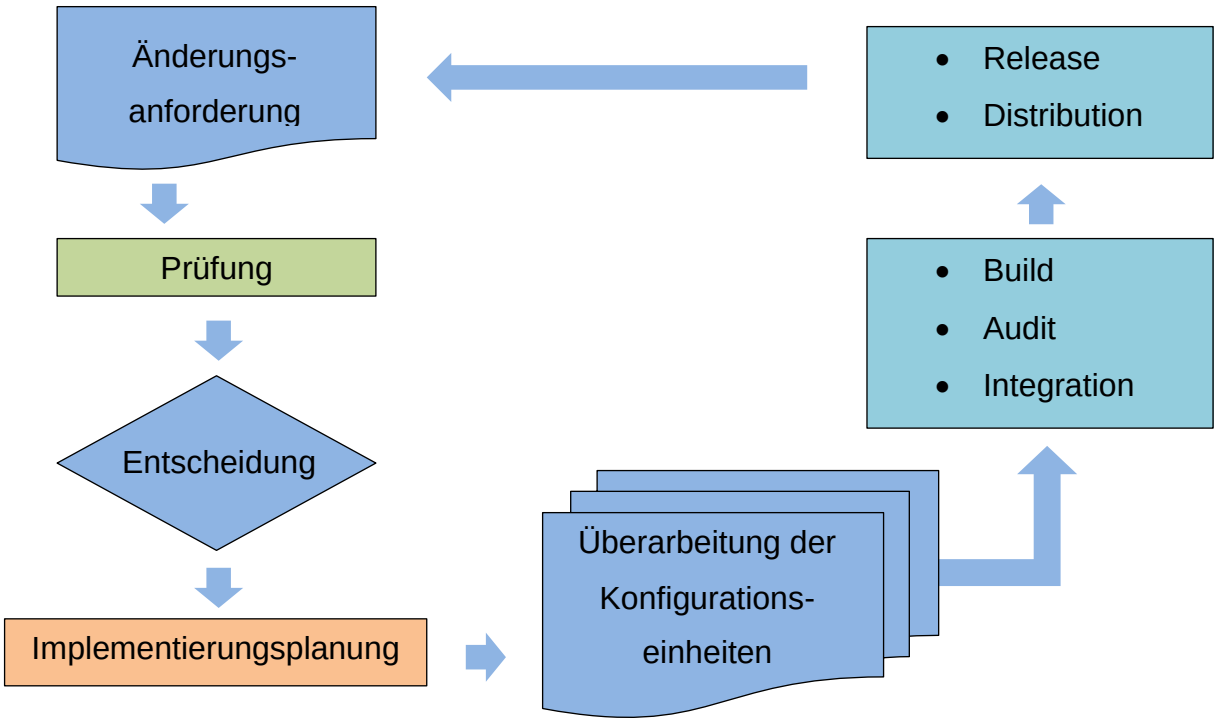


**Abbildung 5: Änderungsprozess (in Anlehnung an Versteegen, 2003, S.14)**

Danach folgt nicht sofort die Umsetzung der Änderung, sondern die Planung der Implementierung. Nach der Planung erfolgt die Überarbeitung der Konfigurationseinheiten und die Erzeugnisse werden zusammengebaut, getestet, freigegeben und verteilt (Abbildung 5).
Durch eine Priorisierung der Änderungen sowie der Koordination der Umsetzung kann z.B. verhindert werden, dass zwei kritische Änderungen an einer Konfigurationseinheit kurz hintereinander von unterschiedlichen Bearbeitern durchgeführt werden.

## 2.11 Buildmanagement

Im Buildmanagement soll mit Hilfe eindeutig identifizierter Konfigurationseinheiten ein Erzeugnis reproduzierbar dargestellt werden. Die Produktion des Erzeugnisses erfolgt mittels eines dokumentierten Buildprozesses. Die dafür erforderlichen Werkzeuge, sowie deren Einstellungen, werden ebenfalls darin dokumentiert (Abbildung 6).

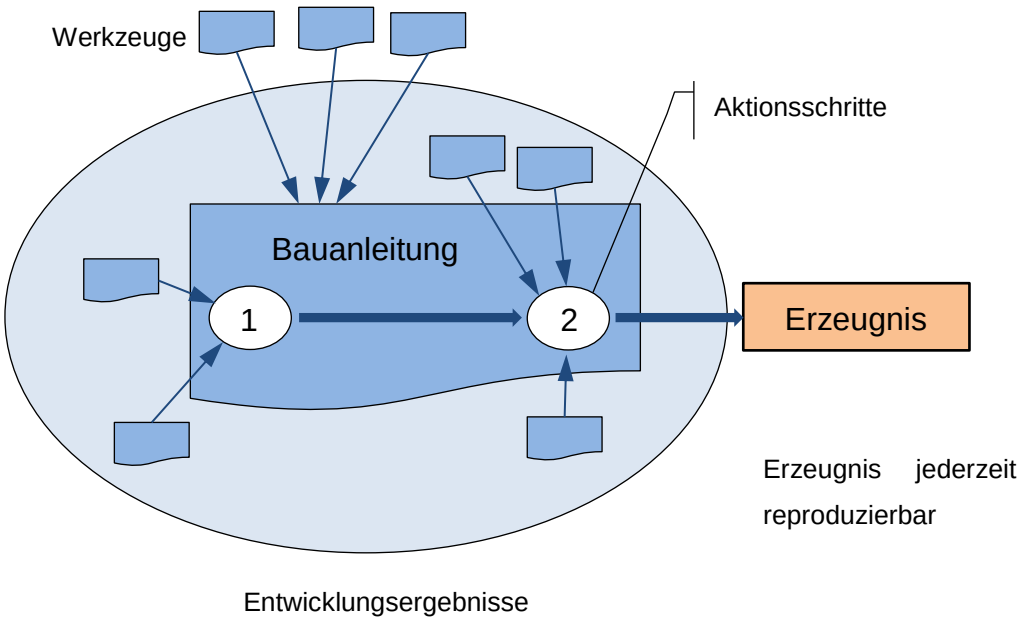


**Abbildung 6: Buildprozess (in Anlehnung an Versteegen, 2003, S.16)**

## 2.12 Distributionsmanagement

Das Distributionsmanagement dient dazu, dass Erzeugnisse zum richtigen Zeitpunkt an einem definierten Ort verfügbar sind.

Möglichkeiten der Verteilung:

- Push-Technologie: Erzeugnisse werden aktiv an Einsatzorte verteilt
- Pull-Technologie: Kunde holt sich die zu verteilenden Einheiten

## 2.13 Audit

Mit Hilfe von Audits soll überprüft werden, ob die Konfigurationseinheiten die an sie gestellten Anforderungen erfüllen. Dies kann von funktionalen Prüfungen des Systems mittels Anwender- oder Integrationstests bis hin zu Überprüfung, ob eine durchgängige Verwendung von Dokumentvorlagen eingehalten wurde, reichen.
Die Gemeinsamkeit aller Audits ist der Vergleich eines Istzustands mit einem vorher festgelegten Sollzustand. Bei einer Erfüllung des Soll verlief der Audit erfolgreich. Wird das Soll nicht erreicht, kann der Audit nach einer Nachbesserung der Einheit wiederholt werden.
Im Konfigurationsmanagementhandbuch ist ein Auditplan zu dokumentieren, wobei Teilnehmerkreis, Zeitpunkt und genaues Ziel festzuhalten sind.

# 3 Anforderungsmanagement (in Anlehnung an Jochem & Landgraf, 2011)

## 3.1 Grundlagen

Das Anforderungsmanagement befasst sich mit der Integration der Kundenwünsche in die Produkt- bzw. Systementwicklung.
Bei komplexen Systemen ist die Integration der Kundenanforderungen eine große Herausforderung. Neben der Erfassung dieser gehört der Einfluss des Systems auf diese beurteilt, dokumentiert und verwaltet. Eine systematische Vorgehensweise ist dabei unabdingbar.
Für den Erfolg eines Projektes sind dessen frühe Phasen der Entwicklung bereits ausschlaggebend. Eine ausreichende Definition der Projektziele sowie die Einbindung der Kunden, präzise Anforderungen oder eine kontrollierte Implementierung von Änderungswünschen können für einen Erfolg verantwortlich sein. Das Anforderungsmanagement befasst sich mit den Anforderungen von der frühen Entwicklungsphase bis hin zur Projektplanung und -durchführung, sowie mit der Wartung.

Eine Anforderung wird vom Standard Institute of Electrical and Electronics Engineers (IEEE) wird wie folgt definiert (IEEE 610.12-1990):

Eine Anforderung ist:

- Eine Bedingung oder Eigenschaft, die ein System oder eine Person benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen;
- Eine Bedingung oder Eigenschaft, die ein System oder eine Systemkomponente aufweisen muss, um einen Vertrag zu erfüllen oder einem Standard, einer Spezifikation oder einem anderen formell auferlegten Dokument zu genügen;
- Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft, wie in den beiden oberen Punkten definiert.

Anforderungen laut dieser Definition können sowohl Wünsche und Ziele der Kunden als auch Eigenschaften und Bedingungen des zu entwickelnden Systems, welches ein Problem lösen oder ein Ziel erreichen soll, sein.

Eine weitere Definition für das Anforderungsmanagement lautet wie folgt (Jochem & Landgraf, 2011, S. 18f):

Als Anforderungsmanagement wird:

- Die Anforderungskontrolle, -verfolgung und -verwaltung sowie
- Die Steuerung und Kontrolle des Prozesses und der Zielsetzungen definiert.

Nach Rupp (2007) untergliedert sich Anforderungsmanagement in:

- Die Anforderungsanalyse (Erheben, Dokumentieren, Prüfen und Verwalten von Anforderungen) und
- Das Anforderungsmanagement (Maßnahmen, die zur Unterstützung der Anforderungsanalyse und der Weiterverwendung der Anforderungen beitragen, wie zum Beispiel ein Simulationsmodell für die Weiterverwendung).

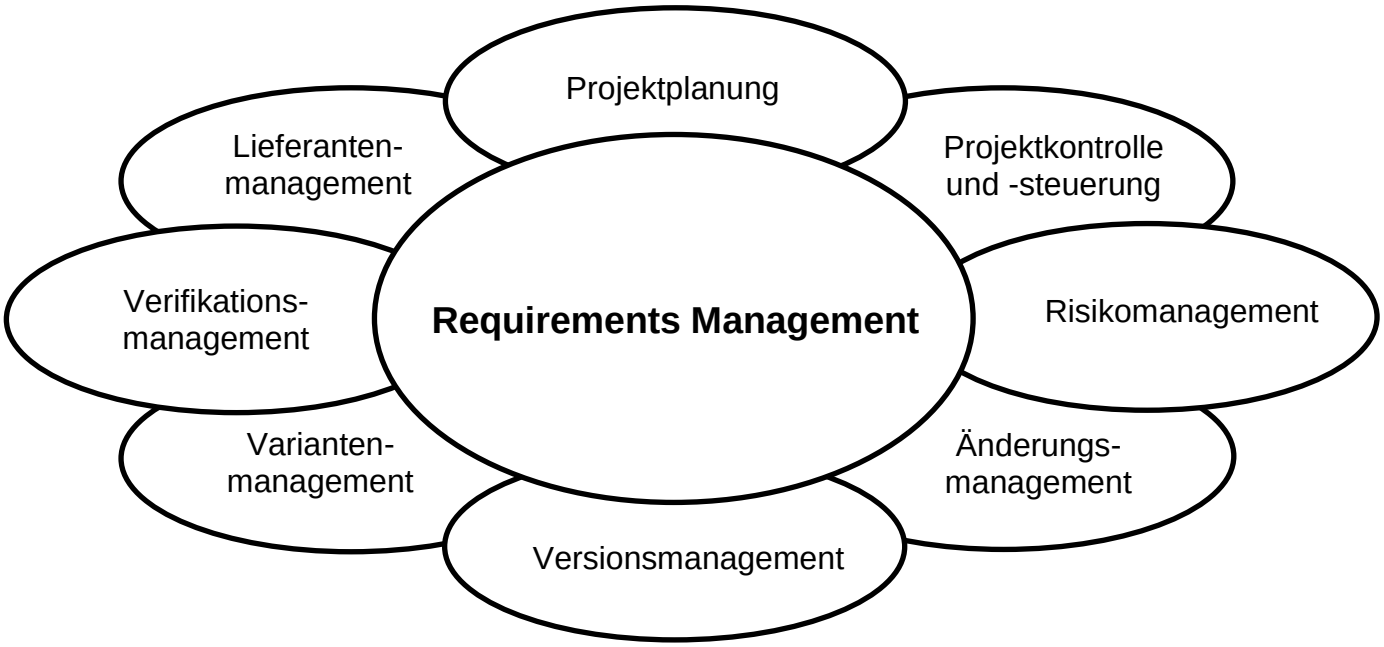


**Abbildung 7: Einflussbereich Anforderungsmanagement (Jochem & Landgraf, 2011, S.20)**

Das Anforderungsmanagement umfasst somit jegliche Aktivitäten, welche sich mit Anforderungen bei der Entwicklung von Systemen und bei der Umsetzung von Projekten beschäftigen.

Andere Managementbereiche wie Projektplanung, Projektkontrolle und -steuerung, das Verifikationsmanagement und Lieferantenmanagement werden durch das Anforderungsmanagement im Produktlebenszyklus unterstützt, koordiniert und mit Informationen versorgt (Abbildung 7). Diese Informationen sind für fundierte Entscheidungen im Projektverlauf von Bedeutung.
Durch die steigende Komplexität an die Anforderungen an ein System sowie die in den Vordergrund rückende Kontrolle von Qualität, Kosten und Zeit, wird ein systematisches Anforderungsmanagement immer wichtiger.
Anforderungen lassen sich in verschiedene Klassifikationen unterteilen. Es kann zwischen Prozess- und Produktanforderungen unterschieden werden (siehe Abbildung 8). Eine Differenzierung in funktionale und nicht funktionale Anforderungen der Produkte ist möglich, welche sich weiter untergliedern lassen. Die Prozessanforderungen umfassen die Bedürfnisse sowie die Rahmenbedingungen auf Kunden- und Lieferantenebene der Geschäftsprozesse.
Die Qualitätsanforderungen beziehen sich auf Anforderungen an die Güte der Produkte, des Prozesses oder an die daran beteiligten Personen.
Das Sammeln von Anforderungen ist in der Praxis oft mit Problemen verbunden, denn generell besitzen die Kunden keine vollständigen Spezifikationen die nur noch analysiert werden müssen. Hierbei ist es wichtig eine hohe Qualität der Anforderungsspezifikation zu erreichen.

Eine Methodik zur Pflege und Entwicklung von Anforderungen ist, Anforderungen zu ermitteln, beschreiben, analysieren und einem Projekt oder einer Projektphase zuzuteilen. Brauchbare Anforderungen sind vollständig, korrekt, konsistent, testbar, verständlich, notwendig, eindeutig und umsetzbar.

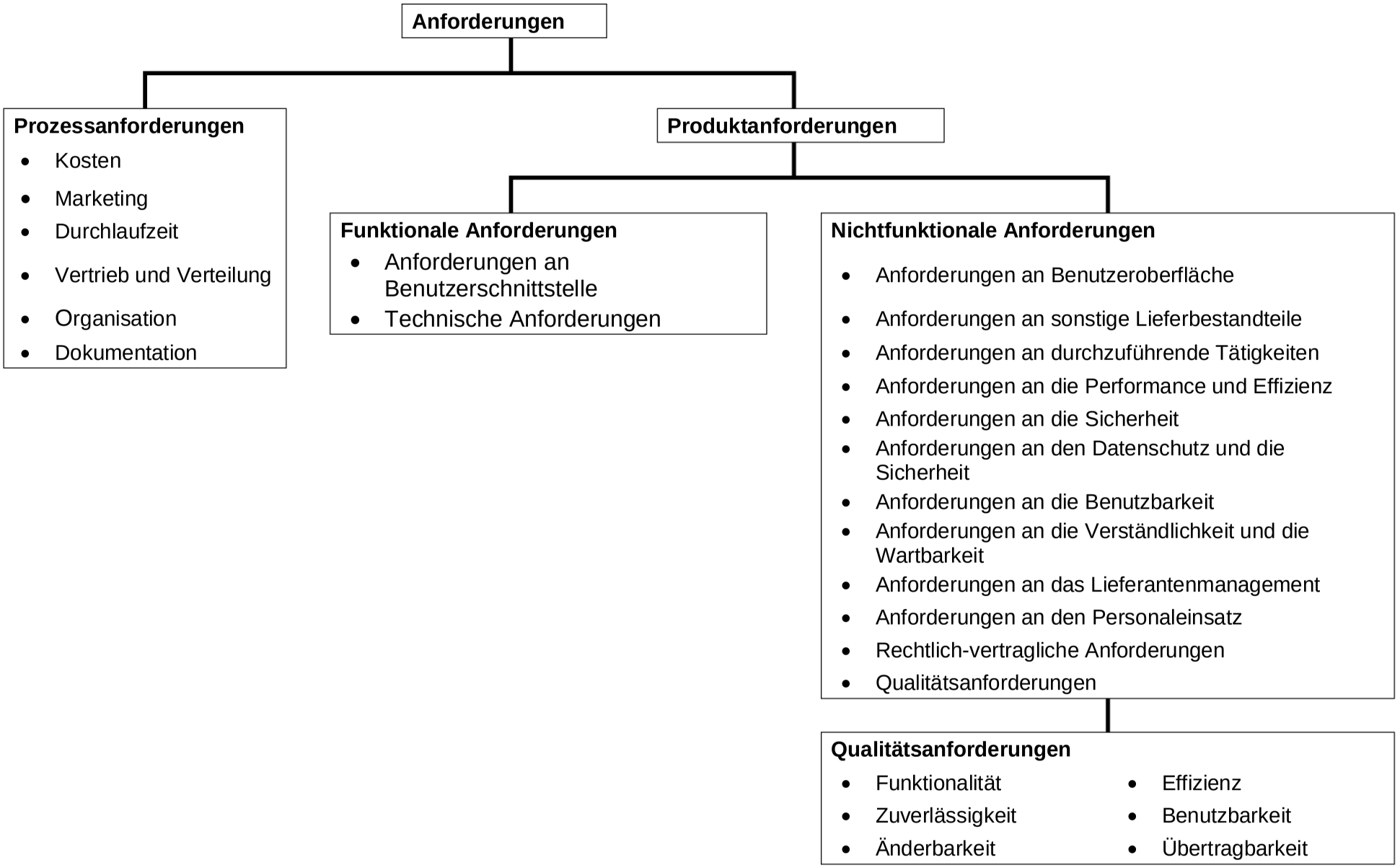


**Abbildung 8: Klassifikation von Anforderungen (in Anlehnung an Jochem & Landgraf, 2011, S. 21)**

Sämtliche Wünsche der Stakeholder sollen durch die Methodik des Anforderungsmanagements erfasst, bewertet und verbunden werden. Durch Kommunikation mit dem Kunden sollen über einen iterativen Prozess Lösungs- und Systemanforderungen formuliert werden. Die Lösungsanforderungen sollen die Basis der Produktanforderungen darstellen, welche wiederum in Funktionen und Eigenschaften transformiert werden.
Im Anforderungsmanagement ist eine strikte Differenzierung zwischen Lösungen und Anforderungen erforderlich. Zuerst sollen Bedürfnisse und Nutzen des Kunden identifiziert werden (z.B. im Lastenheft). Anschließend folgt die Generierung von Lösungen, Pflichtenheft und weiterer produktspezifischen Entwicklungen (Abbildung 9). Eine Kombination von Lösungs- und Anforderungsperspektive führt zu einer nur schwer lösbaren Struktur.

Trennung der Spezifikationen im Anforderungsmanagement in zwei Kategorien (Jochem & Landgraf, 2011, S. 23):

- Anforderungsspezifikation: Beschreibt die Erwartung an die Lösung, deckt die Marktforderungen ab, wird als Lastenheft bezeichnet.
- Systembeschreibung oder Lösungsspezifikation: Beschreibt die Realisation der Lösung, deckt die Produktanforderungen und Teile der Komponentenanforderung ab, wird als Pflichtenheft bezeichnet.

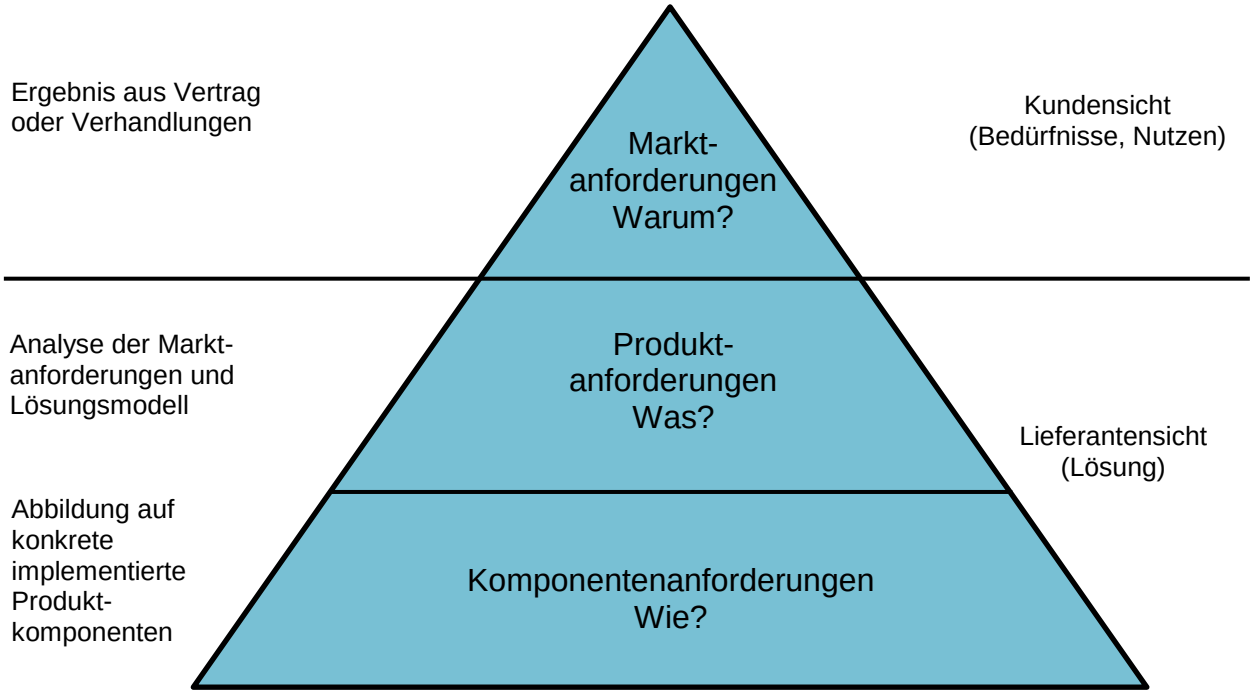


**Abbildung 9: Sichtweisen der Anforderungen (in Anlehnung an Jochem & Landgraf, 2011, S.23)**

Der Best-Practice-Requirements-Management- und Engineering-Prozess (Abbildung 10) zeigt eine mögliche Strukturierung des Anforderungsmanagements. Begonnen wird mit einer Anforderungsermittlung und Anforderungsspezifikation. In der Anforderungsermittlung werden die Projektvision generiert, die Systemgrenzen definiert sowie die Anforderungen gesammelt. Danach werden die Anforderungen spezifiziert und mit den Attributen und deren Abnahmekriterien verknüpft. Im selben Schritt wird eine Anforderungsanalyse dazu durchgeführt. In dieser werden eine Prozesvisualierung durchgeführt und die Risiken dazu analysiert.

Nach der Definition und Spezifikation der Anforderungen werden diese in der Anforderungsverifikation und -validierung geprüft und falls erforderlich korrigiert und abschließend mit dem Kunden vereinbart.

Während dieser gesamten Prozesskette werden in der Anforderungskontrolle/-verfolgung/-verwaltung die Risiken bewertet, Anforderungen kontrolliert, verfolgt und verwaltet, sowie Änderungen gesteuert.

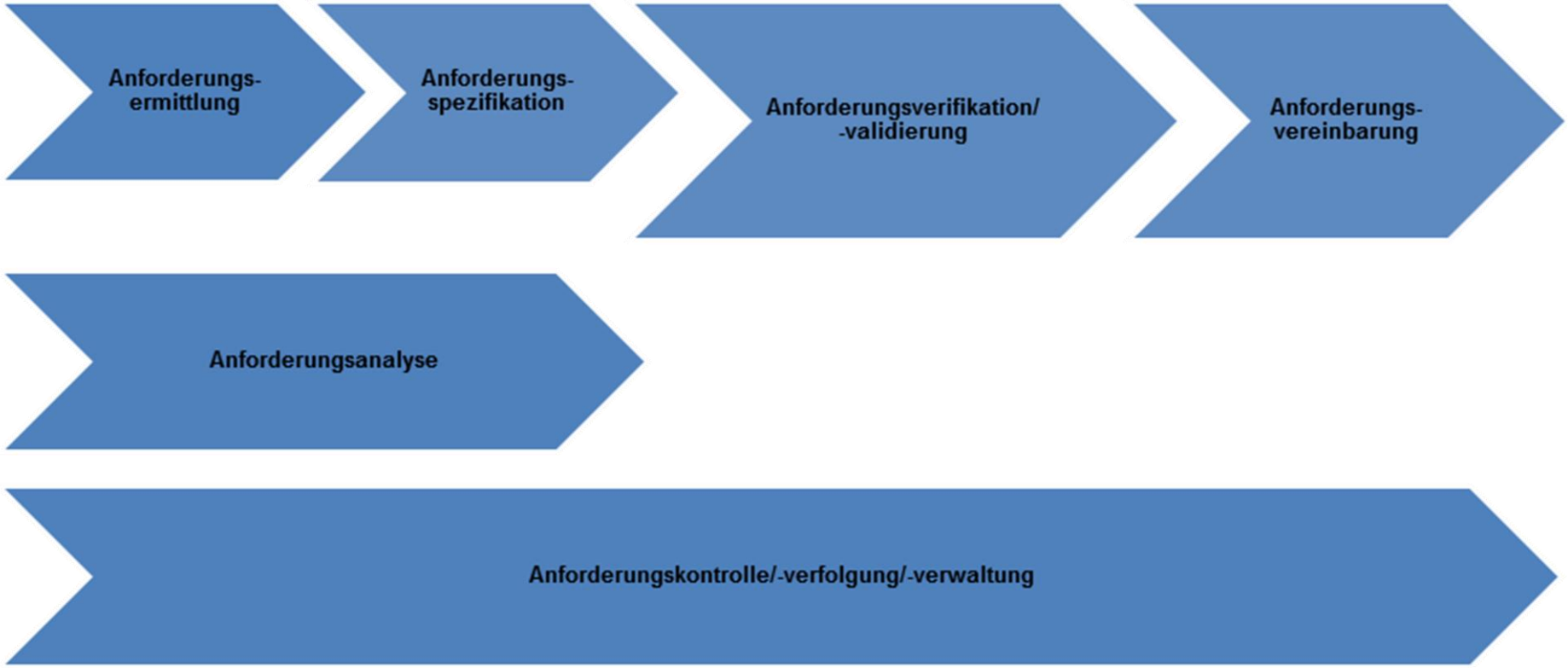


**Abbildung 10: Best-Practice-Requirements-Management- und Engineering-Prozess (in Anlehnung an Landgraf, 2009, S.69)**

## 3.2 Funktionsorientiertes Anforderungsmanagement

### 3.2.1 Einleitung

Eine Verringerung der Komplexität sowie die systematische Wiederverwendung sind die Ziele des funktionsorientieren Anforderungsmanagements.
In der Vergangenheit wurde die Anpassung der Anforderungen durch bereits bestehende Lastenhefte erreicht. Der Nachteil bestand darin, dass keine systematische Wiederverwendung möglich war, d.h. Funktionen, die in einem Nachfolgeprojekt nicht verwendet wurden, entfielen und standen somit für eine zukünftige Wiederverwendung nicht mehr zur Verfügung (Abbildung 11).

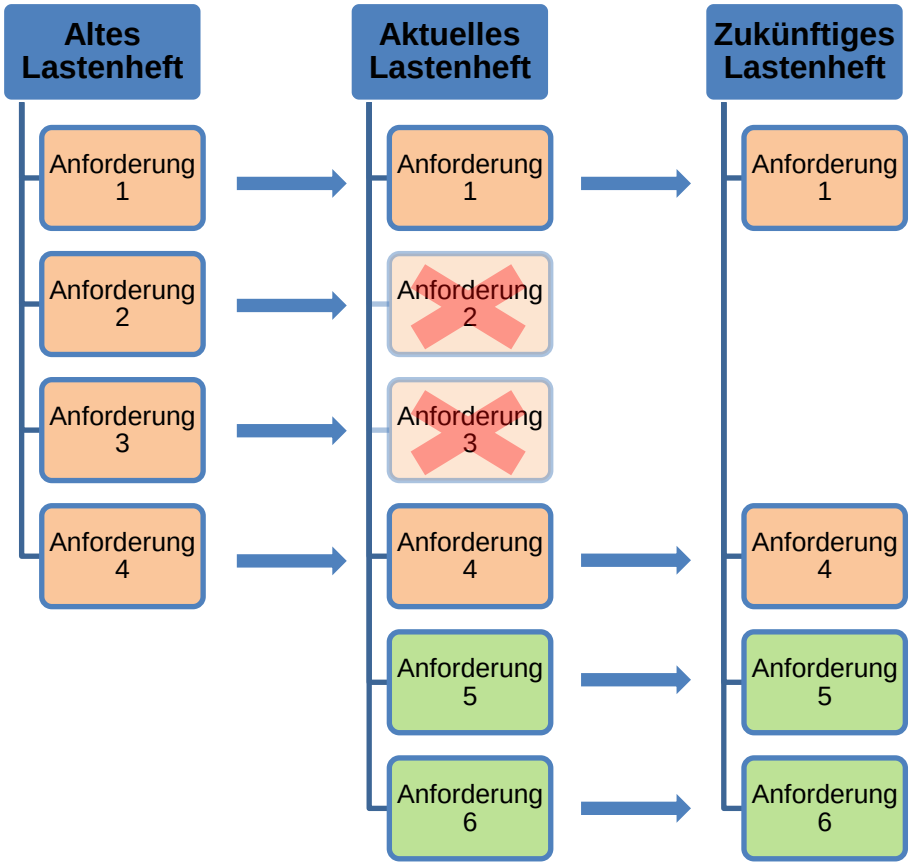


**Abbildung 11: Wiederverwendung von Lastenheften**

In der Automobilindustrie wurden daher in den letzten Jahren Anforderungspools angelegt. Durch diese Ablegung in Datenbanken stehen die Anforderungen zur Wiederverwendung zur Verfügung.

Aufgrund der Gliederung eines Fahrzeugs in Systeme/Module/Bauteile findet diese Gliederung auch bei der Anlegung der Anforderungspools statt. Bei der Definition eines Lastenheftes für eine neue Komponente werden die dafür relevanten Anforderungen aus dem Pool herangezogen. Aufgrund der sehr hohen Anzahl von Anforderungen an ein Fahrzeug, wobei die Komplexität und Anzahl der Anforderungen weiter steigt, ist es erforderlich die Anforderungen über Funktionen zu clustern (Abbildung 12).

Eine Verknüpfung der Anforderung mit den einzelnen Funktionen führt dazu, dass alle Anforderungen für eine Funktion herausgefiltert werden können. Dabei werden alle Anforderungen an eine Funktion erfasst, auch wenn sie aus verschiedenen Domänen (Systemen/Modulen/Bauteilen) stammen.

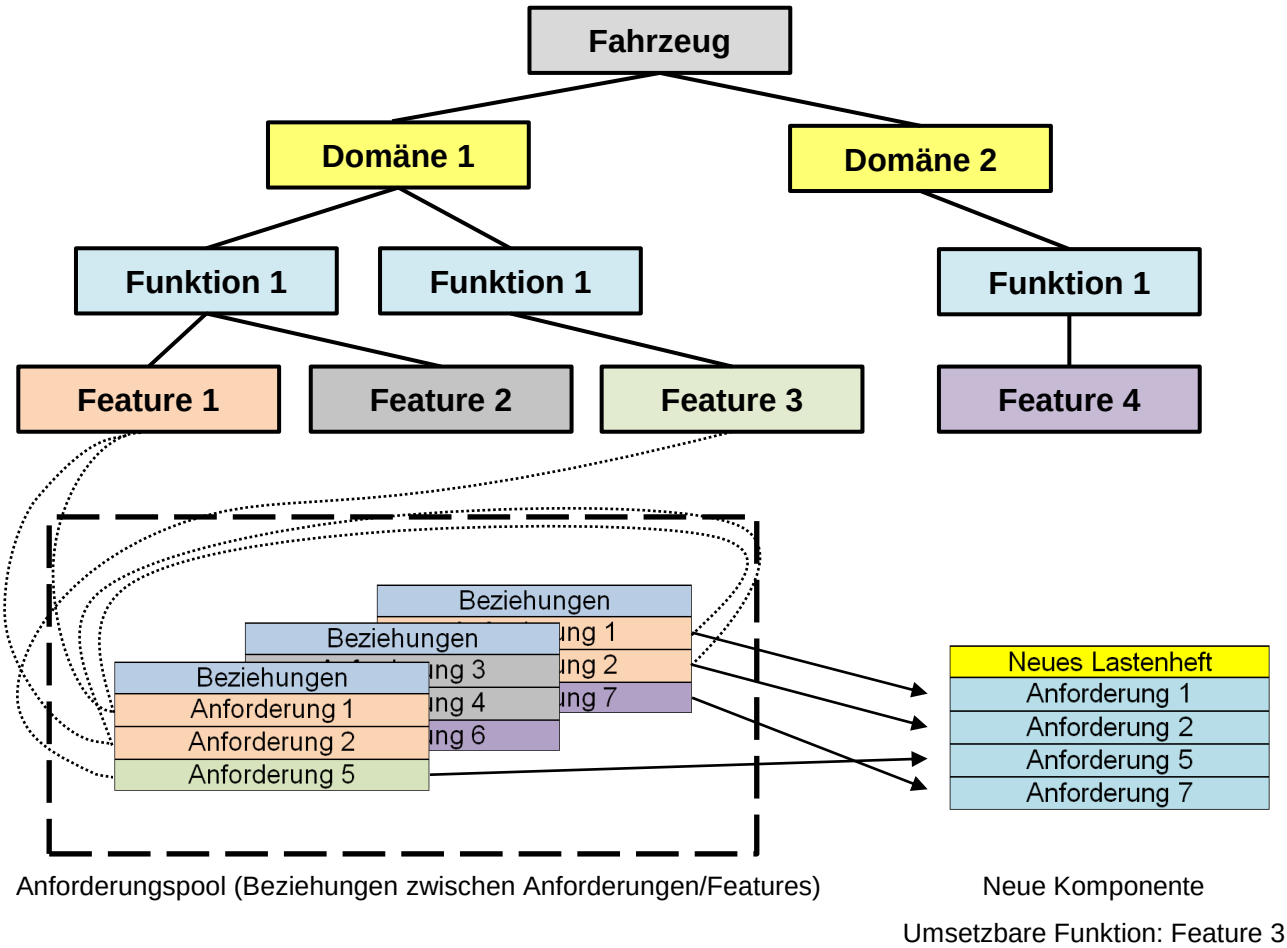


**Abbildung 12: Anforderungen funktionsorientiert gegliedert**

Mittels dieser Strukturierung kann Transparenz geschaffen werden. Woraus besteht eine Funktion, und aus welchen Anforderungen setzt sie sich zusammen? Der Funktionsbaum unterstützt mit seiner hierarchischen Struktur somit die Entscheidung, welche Anforderungen für eine neue Komponente gelten sollen. Durch die damit aufgezeigten Beziehungen und Randbedingungen ist es möglich zu erkennen, wie sich Funktionen gegenseitig beeinflussen, nur gemeinsam umgesetzt werden können oder gegenseitig ausschließen. Auch die von Änderungen betroffen Beziehungen zwischen Anforderungen bzw. deren Auswirkungen auf Abhängigkeiten können damit einfach ermittelt werden.
Vor der Erstellung eines Funktionsnetzes müssen Funktionen und Anforderungen miteinander verknüpft werden. Ebenfalls werden Beziehungen zwischen Funktionen und auch zwischen den Anforderungen aufgestellt. Nach der Erstellung des Funktionsbaumes werden für jede neue Komponente die umzusetzenden Funktionen in einem neuen Lastenheft bestimmt. Die Auswahl der Anforderungen erfolgt automatisch über die Zusammenhänge der Beziehungen zwischen den Funktionen und Anforderungen. Beim Hinzufügen von neuen Funktionen müssen diese im Funktionsbaum integriert und mit den bereits vorhandenen Funktionen vernetzt werden. Danach folgt die die Spezifikation von Anforderungen.
Diese Struktur erhöht die Anzahl wiederverwendbarer Anteile. Über die Zielfunktion wird auf Basis des Funktionsbaums über die erforderlichen Anforderungen entschieden.

Während des Ablaufs wird über folgende Punkte entschieden (Jochem & Landgraf, 2011, S. 68):

- Grobentscheidung für bestimmte Funktionen auf Kundenanforderungsbasis und der vom Kunden gewünschten Ziele.
- Funktionsentscheidung auf Basis technologischer Punkte.
- Konkretisierung der Funktionen auf Basis der Technologieentscheidung. Daraus ergeben sich die umzusetzenden Funktionen mit den relevanten Teilfunktionen. Die daran verknüpften Anforderungen werden einfach übernommen.

### 3.2.2 Software „Doors“

Die Anforderungsmanagementsoftware Doors (**D**ynamic **O**bject **O**riented **R**equirements **S**ystem) ist im Automotive-Sektor weit verbreitet. Die Verwaltung von Daten sowie der Datenaustausch zwischen Hersteller und Zulieferer werden mit diesem Tool unterstützt. Die Einsatzmöglichkeiten sind sehr vielfältig und reichen von der Dokumentation der Anforderungen, der hierarchischen Strukturierung bis hin zum Versionsmanagement und zur Änderungsverfolgung. Die Erstellung von nützlichen Dokumenten im Änderungsmanagement, wie z.B. Lastenheften wird ebenfalls unterstützt.

Jede Anforderung in Doors wird als eigenes Objekt betrachtet und bekommt eine eigene ID zugewiesen. Da im Anforderungsmanagement jede Anforderung nachweisbar sein muss, werden für den vollständigen Erfüllungsnachweis Means of Compliance (MOC)/Nachweisverfahren genutzt. In Doors werden somit während der Spezifikation für jede einzelne Anforderung im Attribut MOC die entsprechenden Nachweisarten festgelegt. Sämtliche Nachweisverfahren werden in einem Nachweismodul aufgelistet. Darin wird auch jeder Anforderung das zugehörige Testverfahren zugewiesen (Abbildung 13). Nicht testbare Anforderungen werden im MOC z.B. durch Berechnungen, Simulationen oder Annahmen nachgewiesen.

Mittels Analyseoptionen und Filtermöglichkeiten können Übersichten erstellt werden, welche Anforderungen ein Nachweisverfahren abdeckt oder welche Testfälle eine Anforderung benötigt.

Aufgrund der erforderlichen Zuordnung von Testarten/-ebenen wird bereits in einer sehr frühen Phase ans Testen gedacht, wobei Vereinbarungen mit dem Kunden getroffen werden können. Ein Vorteil der Verknüpfung von Anforderungen zu Testfällen ist auch, dass der Nachweisstatus der Anforderungen während des Projektes überprüft werden kann. Weiters ist bei Änderungen der Anforderungen leicht ersichtlich, welche Testfälle davon betroffen sind (Jochem & Landgraf, 2011, S.70ff).

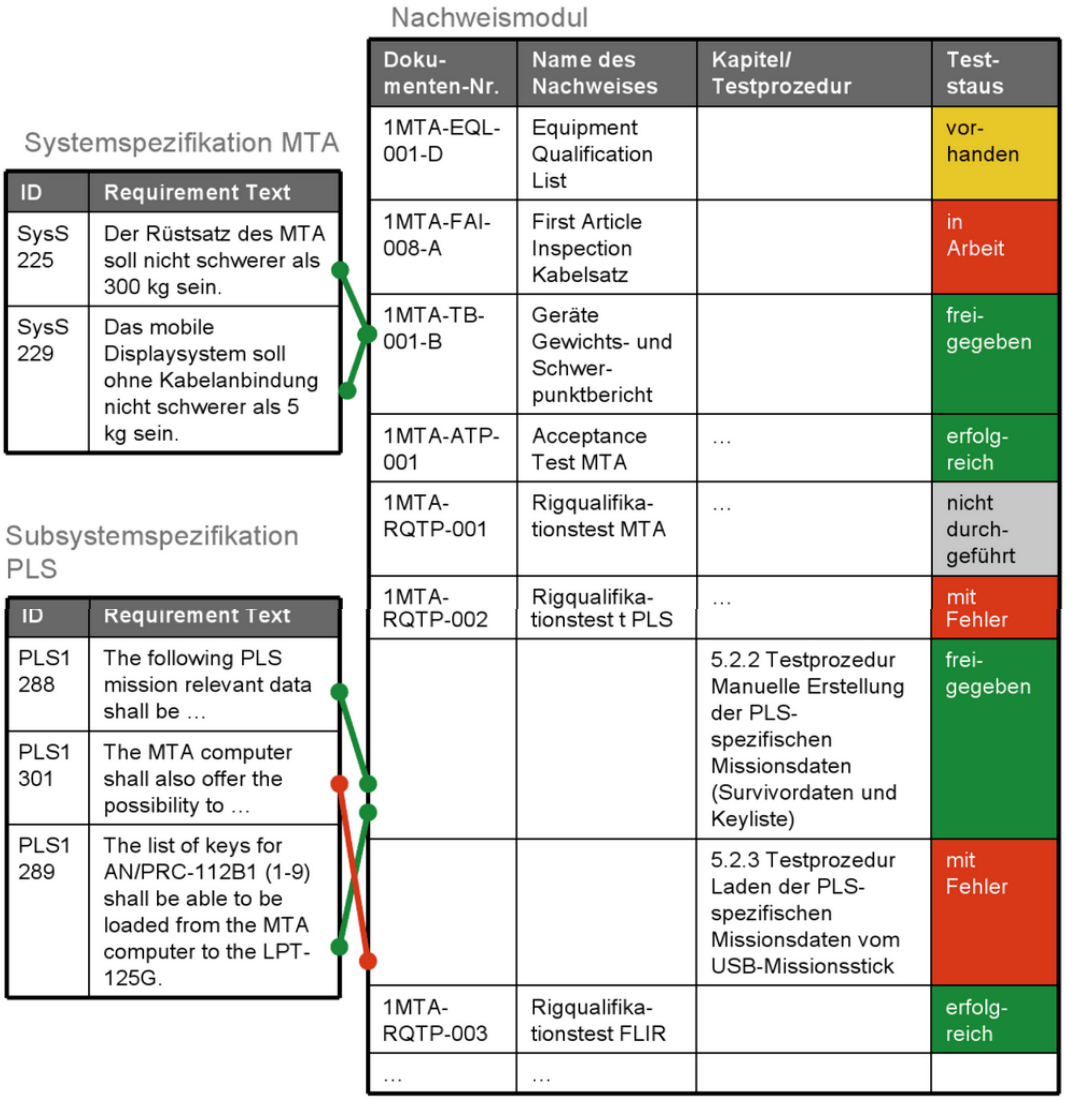


**Abbildung 13: Doors Anforderungs-Nachweisverfahren-Verknüpfung (Jochem & Landgraf, 2011, S.71)**

## 3.3 Anforderungsmanagementsystem

### 3.3.1 Einleitung

Bei der Entwicklung von komplexen Produkten steht man vor der Herausforderung verschiedene Anforderungen einer Vielzahl von Stakeholdern befriedigen zu müssen. Um diese Anforderungsmenge in adäquate Produktmerkale und Lösungsmöglichkeiten übersetzen zu können, ist die richtige Strukturierung, Identifizierung und Analyse notwendig.

Aufgrund unterschiedlicher Interpretationen von Anforderungen, wobei die Ursache dafür oft in den unterschiedlichen Fachkenntnissen der jeweiligen Stakeholder liegt, treten oft Verständnisprobleme im Umgang mit Anforderungen, bezogen auf Bedeutung und Inhalt, auf. Ein weiteres Problem kann auch einfach die große Menge an zu berücksichtigenden Anforderungen sein. Die Lösung dafür liegt in einer strukturierten Vorgehensweise. In der Planung ist darauf zu achten, dass die Anforderungen vollständig vorliegen und alle Stakeholder berücksichtig wurden.

Eine weitere Steigerung der Komplexität im Planungsprozess wird durch gegenseitige Abhängigkeiten der Anforderungen verursacht. Beziehungen zwischen Anforderungen entstehen, wenn diese miteinander interagieren oder ein gegenseitiger Einfluss aufeinander vorhanden ist. Diese Beziehungen haben einen starken Einfluss auf das Ergebnis des Entwicklungsprozesses.

Wenn aufgrund von unterschiedlichen Zielen der Stakeholder Beziehungen entstehen, oder aufgrund von technischen Zusammenhängen, dann spricht man von Wirkbeziehungen. Diese müssen bei der Entscheidung über Produktmerkmale berücksichtigt werden, um den bestmöglichen Lösungskompromiss der gestellten Anforderungen zu finden. Unter den Anforderungen gibt es sogenannte Präzisierungsbeziehungen. Diese entstehen bei der Abgabe durch auf verschiedenen Abstraktionsebenen befindlichen Anforderungen, oder wenn bereits festgelegte Anforderungen erst im weiteren Planungsverlauf präzisiert werden. Derartige Beziehungen müssen bei der Erstellung der Strukturierung ebenfalls berücksichtigt werden.

Diese genannten Punkte führen zu Unsicherheiten und Unstimmigkeiten im Planungsprozess. Daher werden oft Entscheidungen auf Basis von Erfahrungen aus

abgeschlossenen Projekten getroffen und nur zum Teil aufgrund der erhobenen Anforderungen. Dies kann aber den Erfolg des Planungsergebnisses gefährden.

### 3.3.2 Formalisierung von Anforderungen

Das Ziel in der Formalisierung von Anforderungen liegt in der Strukturierung ihres Inhalts und in der Dokumentation ohne Informationsverluste. Ein mögliches Template dafür wurde vom SFB 696 (Sonderforschungsbereich 696 TU Dortmund) entwickelt (Abbildung 14).


**Abbildung 14: Template zur Abbildung von Einzelanforderungen (Jochem & Landgraf, 2011, S.80)**

In der Kategorie *Bezug* werden die Kernanforderungen definiert. Die anderen Kategorien erfassen erweiterte Informationen der Anforderung. Eine Anforderung kann anhand der Komponenten *Bezugsobjekt, Eigenschaft* und *Wert* definiert werden. Eine beispielhafte Anforderung wie „Das System soll eine hohe Verfügbarkeit haben" lässt sich folgendermaßen überführen. „Bezugsobjekt-System“, „Eigenschaft-Verfügbarkeit“ und „Wert-hoch“. Um den Wert zu definieren wurden die

Skalentypen *nominal, ordinal, Intervall, Verhältnis* und *absolut* eingeführt. Der geringste Typ ist der nominale Wert und der höchste der absolute Wert. Je höher der ausgewählte Skalentyp ist, desto höher ist der Informationsgehalt der Anforderung.
Um die Beziehungen zwischen Anforderungen und ihren drei Komponenten (Bezugsobjekt, Eigenschaft, Wert) darzustellen, wurde eine Ontologie entwickelt.

### 3.3.3 Ontologie

Eine Ontologie besteht aus einer Klassenhierarchie, zusammengesetzt aus den relevanten Begriffen der betrachteten Domäne und dessen Verbindungen.

Die Klassenhierarchie besteht hier aus folgenden Hauptklassen (Abbildung 15):

- Anforderungssatz
- Bezugsobjekt
- Eigenschaft
- Wert

Mittels der Oberklasse-Anforderungssatz werden im Zuge einer Anforderungsabgabe Entities erzeugt. Jede einzelne davon repräsentiert eine einzelne Anforderung. Ausgehend davon führt eine „hat_Bezugsobjekt“-Relation zum nächsten Oberklasse-Bezugsobjekt. Auf gleiche Weise führen weitere Relationen zu den Klassen *Eigenschaft* und **Wert**. Mittels dieser Relationen sollen ausschließlich sinnvolle Beziehungen und Kombinationen zwischen den Klassen definiert werden. Alle potenziellen Anforderungen sollen durch diese Kombinationen der Komponenten erfasst werden können.
Weitere definierte Relationen dieser Ontologie heißen „besteht_aus“, „wird_präzisiert_von“ und „beeinflusst“. Die Relation „besteht_aus“ legt fest, aus welchen anderen Objekten das jeweilige Bezugsobjekt bestehen kann. Dies ermöglicht die Definition einer Produktstruktur während des Planungsprozesses. Weiters ist in der Relation die Information enthalten, welches Bezugsobjekt ein

anderes präzisieren kann. Die „wird_präzisiert_von“ Relation beschreibt, welche Eigenschaft oder welcher Wert eine andere Eigenschaft oder einen anderen Wert präzisieren kann. Die Information, wie die Erfüllung einer Anforderung die Erfüllung einer anderen beeinflusst, wird in der Relation „beeinflusst“ gespeichert.

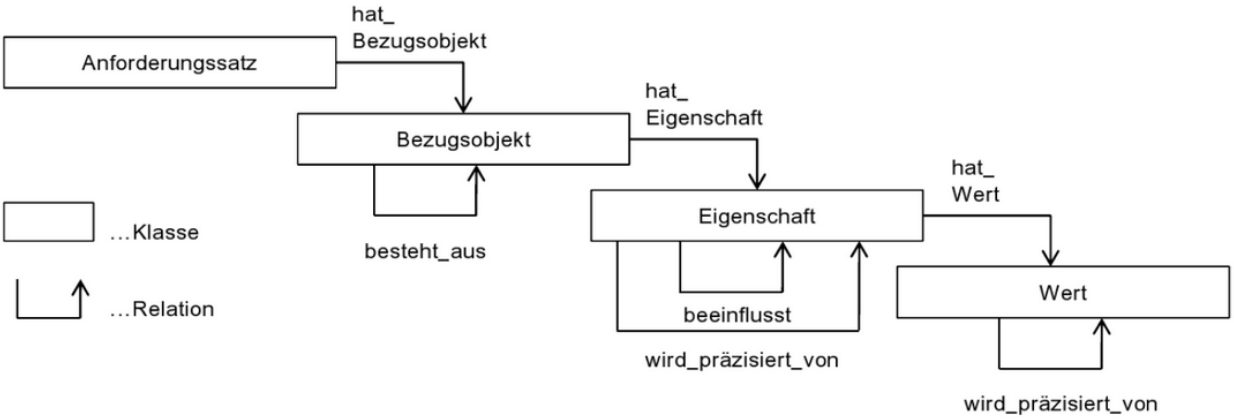


**Abbildung 15: Anforderungsontologie (Jochem & Landgraf, 2011, S.82)**

Die Ontologie dient somit als Bibliothek der Anforderungen und ihren Beziehungen zueinander.

### 3.3.4 Strukturierung von Anforderungen in Gruppen

Durch eine Vielzahl von zu berücksichtigenden Stakeholdern und deren Anforderungen ist ein mehrdimensionales Strukturierungsmodell erforderlich. Die Auswahl der Anzahl an Strukturierungsdimensionen sowie die Gestaltung der Unterteilung dieser in Kategorien und Unterkategorien erzeugt Räume, in welche die Anforderungen eingeordnet werden können. Die Dimensionen und Kategorien sind so auszuwählen, dass sie intralogistischen Anlagen zuweisbar sind. Von besonderer Bedeutung sind die Kriterien Vermeidung von Doppelungen und Unabhängigkeit der Klassen. Daher sollten die Kategorien der Dimensionen keine zu große Ähnlichkeit aufweisen. Weiters sollten auch Kategorien und Dimension weitgehend voneinander unabhängig sein. Die Einordnung einer Anforderung in eine Kategorie einer

Dimension darf nicht aufgrund von Ergebnissen einer Anforderungsstrukturierung einer anderen Kategorie erfolgen.

Der Stakeholder soll durch Auswahl aus den definierten Dimensionen seine Anforderung einer entsprechenden zuweisen können. Danach werden die Kategorien und Unterkategorien der jeweiligen Dimension angezeigt, in welche die Anforderung einsortiert werden kann. Durch diesen Zuordnungsprozess werden potenzielle Konflikte aufgezeigt, falls eine Anforderung unterschiedlichen Kategorien zugewiesen wurde.

Grundgedanke bei diesem Modell ist, dass eine Anforderung nicht allein betrachtet werden sollte (Abbildung 16). Eine ganzheitliche Betrachtung ist erforderlich, von der Anforderungserfüllung der Stakeholder bis hin zur resultierenden Kundenzufriedenheit. Abgesehen vom Produkt selbst dürfen auch zusätzliche produktbegleitende Dienstleitungen, wie z.B. Wartung und Service, bei langlebigen und komplexen Systemen nicht außer Acht gelassen werden. Daher fällt auch das erweiterte Produkt in den betrachteten Bereich der Anforderungserfassung, -strukturierung und -bewertung.

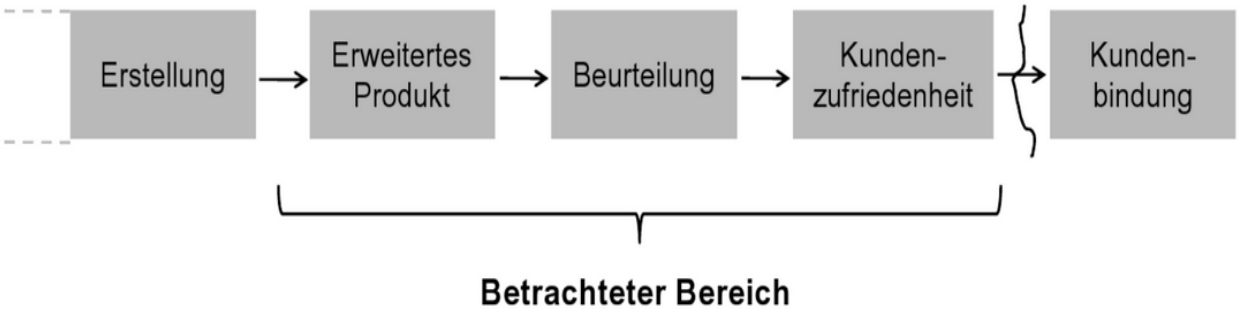


**Abbildung 16: Grundgedanke des Strukturmodells (Jochem & Landgraf, 2011, S.85)**

### 3.3.5 Anforderungspriorisierung

Bei einer großen Anzahl abgegebener Anforderungen kommen häufig sich ausschließende und konkurrierende Anforderungen vor. Daher ist eine Gewichtung nötig, um Prioritäten festlegen zu können. Die Gewichtung im Template (Abbildung 14) des SFB 696 beinhaltet drei Teilgewichte, welche zusammen das Gewicht der Anforderung ergeben.
Eine andere Möglichkeit der Gewichtung ist die Conjoint-Analyse. Dabei wird eine Rangfolge von Produktbündeln aufgestellt. Daraus werden die Anforderungsgewichte abgeleitet.
Die AHP (Analytial Hierarchy Process) Methode berechnet die Anforderungsgewichte mithilfe des paarweisen Vergleichs indem Alternativen priorisiert werden. Vorrausetzung für den AHP ist das Vorhandensein der Hierarchiestruktur der gegebenen Anforderungen (Jochem & Landgraf, 2011, S. 91).

### 3.3.6 Systemnutzen

Mit der Ontologie ist es möglich die Einzelanforderungen zu strukturieren und in ihrer Sinnhaftigkeit und Vollständigkeit zu prüfen. Die nachfolgende Strukturierung nach Gruppen mittels eines mehrdimensionalen Modells im Rahmen von Dimensionen und Kategorien unterstützt die Überprüfung, ob alle Stakeholderanforderungen im Planungsprozess berücksichtig wurden, oder ob Informationsdefizite vorliegen. Anhand der einzelnen Dimensionen und Kategorien können deren zugehörige Anforderungen angezeigt werden, wodurch kontrolliert werden kann, ob jeder Stakeholder bzw. dessen Anforderungen untersucht wurden. Dieses Modell beinhaltet nicht die Beziehungen der Anforderungen zueinander, das Wissen darüber befindet sich in der Ontologie. Daher ist es Aufgabe der Ontologie, die Beziehungen zwischen Anforderungen zu identifizieren. Bei beiden Schritten ist der Abstraktionslevel der Anforderungen zu beachten. Es besteht das Risiko, dass Anforderungen in einem oberen Level des Modells verbleiben und eine Einordnung in tiefere Ebenen nicht erfolgt. Eine Erschwerung der Überprüfung der Vollständigkeit

der Anforderung innerhalb der Kategorie wäre die Folge. Durch eine Analyse von Präzisierungsbeziehungen kann dieses Problem behoben werden (Abbildung 9).

Mit Hilfe eines derartigen Anforderungsmanagementsystems wird die Eingangsgröße für Methoden der Qualitätssicherung (v.a. QFD; Quality-Function-Deployment) optimiert, wobei die Voraussetzung dafür eine atomare Form der Anforderungen ist.
Die Formalisierung, Zusammenhangsanalyse in der Ontologie und Strukturierung der Anforderungen dienen als Vorbereitung für ihre spätere Umsetzung im Rahmen einer QFD. Diese Strukturierung ist aufgrund der großen Menge an zu berücksichtigenden Anforderungen essentiell, da ansonsten ungeeignete Resultate, wegen der auf unterschiedlichen Detailierungsebenen befindlichen Anforderungen, auftreten würden.

# 4 **Fahrzeugeigenschaften** (in Anlehnung an Eckstein (2010))

Kraftfahrzeuge unterliegen einer großen Zahl an Anforderungen (Abbildung 17), wobei sich viele Abhängigkeiten und Gegensätze dieser Randbedingungen ergeben. Diese spezifischen Anforderungen ergeben sich durch den Endkunden, den Hersteller oder durch gesetzliche Richtlinien und müssen in einem stimmigen Gesamtkonzept verwirklicht werden. Hierbei können Zielkonflikte auftreten, siehe Abbildung 18.

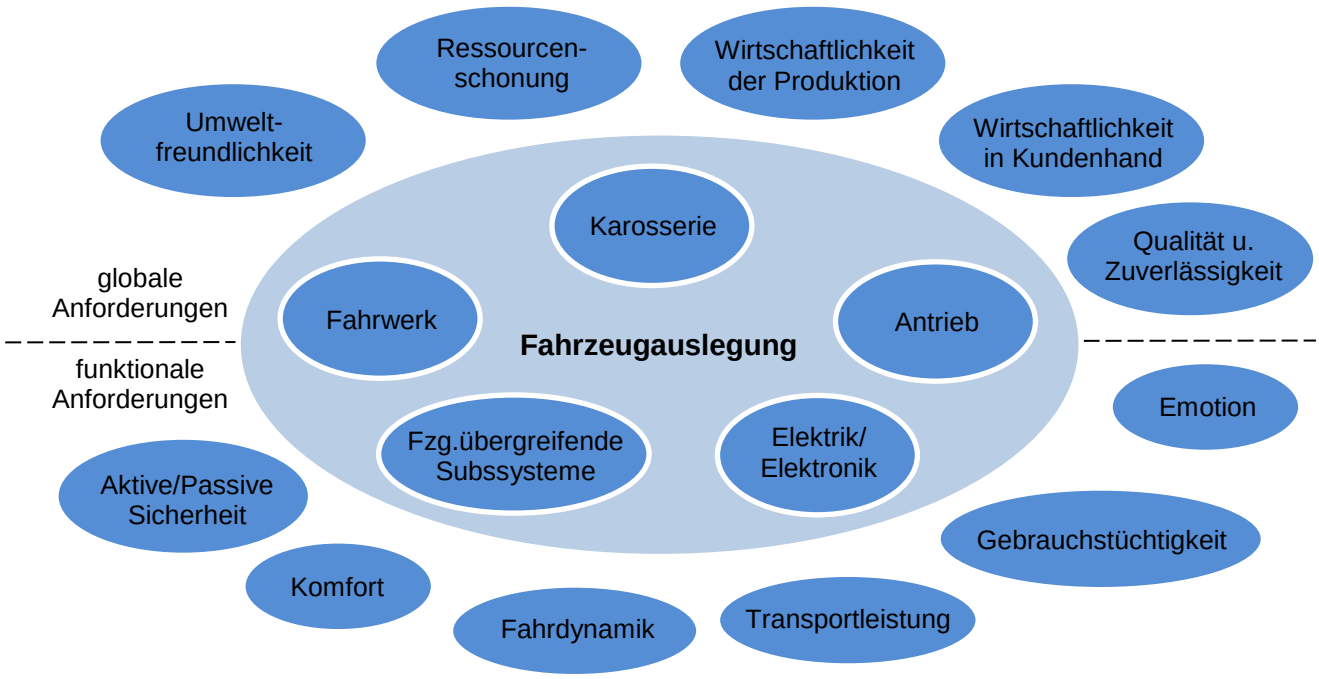


**Abbildung 17: Grundsätzliche Anforderungen (in Anlehnung an Eckstein, 2010, S.13)**

Um die Zufriedenheit der Kunden sicherzustellen, muss auf deren Anforderungen an ein Fahrzeug eingegangen werden. Diese Anforderungen variieren unter den Kunden, in Abhängigkeit von ihrer Umwelt bzw. ihrem Lebensraum. Diese Unterschiede lassen sich in Anforderungssätze zusammenfassen, welche die Eigenschaften und technischen Merkmale eines Fahrzeuges beeinflussen. Bei der Entwicklung eines Fahrzeugkonzeptes müssen diese Einflüsse von Beginn an berücksichtigt werden.

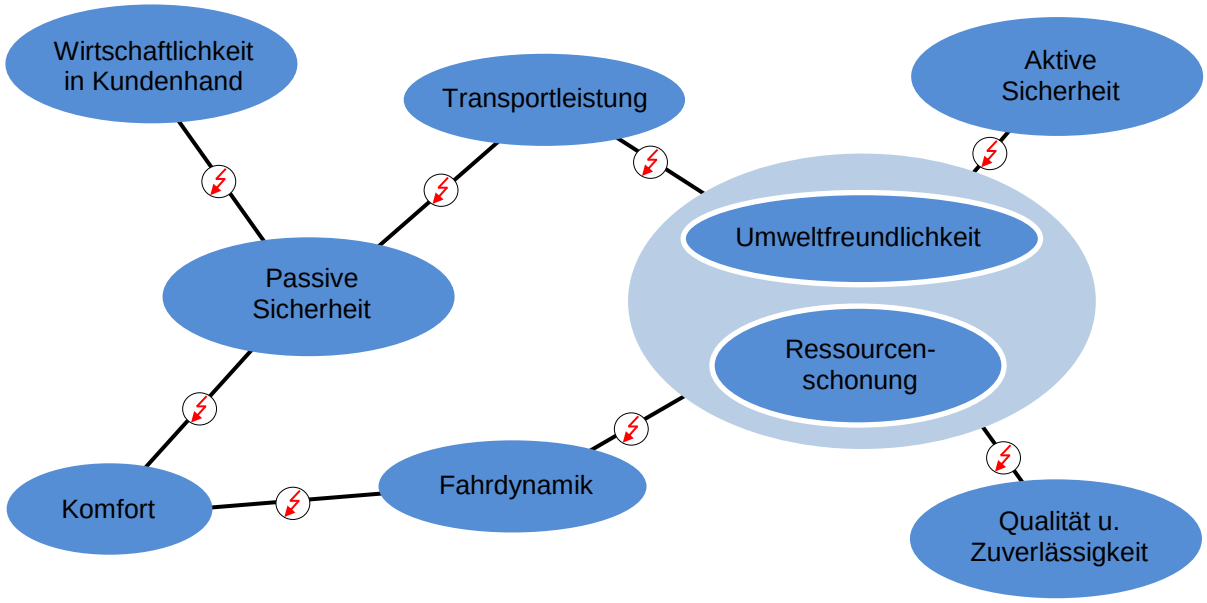


**Abbildung 18: Zielkonflikte (in Anlehnung an Eckstein, 2010, S.13)**

Bei der Aufstellung der vom Kunden gewünschten Eigenschaften eines Fahrzeuges können bereits die ersten Zielkonflikte auftreten. Die gegensätzlichen Anforderungen müssen bereits in der Konzeptphase berücksichtigt werden. Zudem ist aufgrund der Wechselwirkung vieler Eigenschaften die Anforderungsspezifikation sehr komplex. Weiters kommt hinzu, dass die Entwicklung vieler, auf sich gegenseitig Einfluss nehmender Systeme in verschieden Abteilungen des Unternehmens erfolgt. Jede Abteilung hat ihre eigene Sicht auf ihr Subsystem, während nur wenige einen Überblick über die Interaktion sämtlicher Systeme bewahren. Diese Problemstellungen werden im Konfigurations- und Änderungsmanagement behandelt.

# 5 Datenbanken (in Anlehnung an Geisler, 2011)

## 5.1 Grundlagen

Unter einer Datenbank versteht man ein verteiltes, integriertes Computersystem welches Nutz- und Metadaten enthält. Daten die von Benutzern angelegt werden und aus denen Informationen abstrahiert werden können sind Nutzdaten. Metadaten helfen die Nutzdaten in der Datenbank zu strukturieren und werden oft als Daten über den eigentlichen Daten gesehen (Abbildung 19).
Zur Verwaltung einer Datenbank werden Datenbankmanagementsysteme (DBMS) verwendet, welche den Zugriff auf die Daten regeln. DBMS können aus mehreren Programmen bestehen, welche zusammenarbeiten.

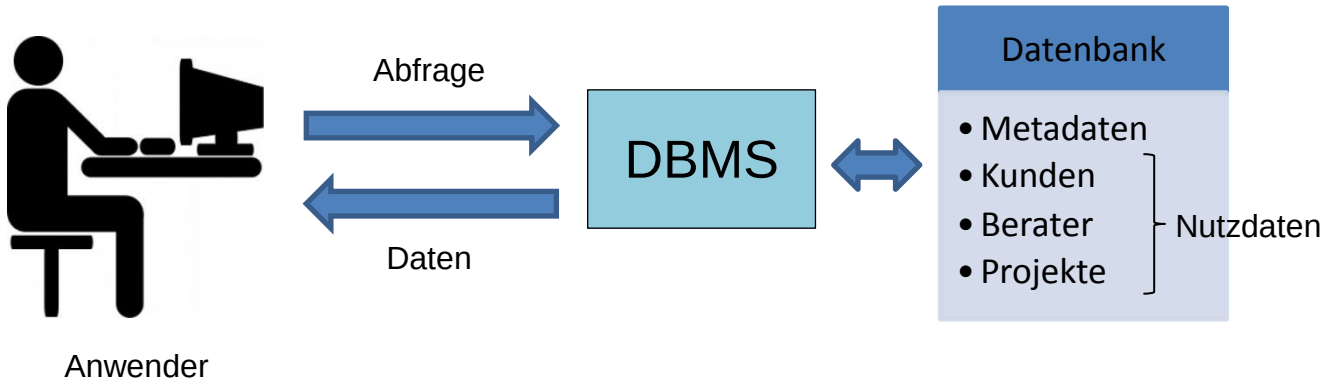


**Abbildung 19: Datenbankmanagementsystem (in Anlehnung an Geisler, 2011, S.22)**

Bei einem schlechten Datenbankdesign besteht das Problem, dass redundante Daten auftreten können. Dieses mehrfache Auftreten von Daten in der Datenbank verursacht einen erhöhten Speicherbedarf und Inkonsistenzen.
Man spricht von Dateninkonsistenzen, wenn verschiedene Versionen einer Datei existieren und dadurch Konflikte entstehen, welche Version die aktuellste bzw. gültige ist. Werden Datensätze aus einer Datei abgerufen und man erhält unterschiedliche Informationen, dann fehlt den Daten die Datenintegrität. Fehler durch Dateninkonsistenzen werden auch Datenanomalien genannt. Ändert sich der Wert eines Datenfeldes, so sollte die Änderung an einer einzigen Stelle erfolgen müssen.

## 5.2 Datenanomalien

- Änderungs-Anomalie
  Bei Speicherung derselben Daten an verschiedenen Stellen muss die Gewährleistung existieren, dass die Daten auch an allen gespeicherten Stellen geändert werden.

- Einfüge-Anomalie
  Diese liegt bei Nichterfassung bestimmter Daten, oder bei wenn Daten nicht gleichzeitig erfasst werden können, vor.

- Lösch-Anomalie
  Eine Lösch-Anomalie tritt auf, wenn beim Löschen bestimmter Daten andere Daten, die nicht gelöscht werden sollten, ebenfalls mitgelöscht werden.

Datenanomalien können in schlecht entworfenen Datenbanksystemen auftreten und sind das Ergebnis von Datenredundanz.

## 5.3 Datenbanksystem

Datenbanksysteme bestehen nicht nur aus einem Datenbankmanagementsystem, sondern auch aus zahlreichen anderen Komponenten die in eine Beschreibung des Datenbanksystems mit einbezogen und untersucht werden müssen.
Moderne Datenbanksysteme überwachen sogar die Beziehungen einzelner Datensätze in der Datenbank. Das heißt, es ist nicht möglich Datensätze der Datenbank zu löschen, von denen andere Datensätzen abhängig sind.

Ein Datenbanksystem besteht aus vier Schichten (Abbildung 20):

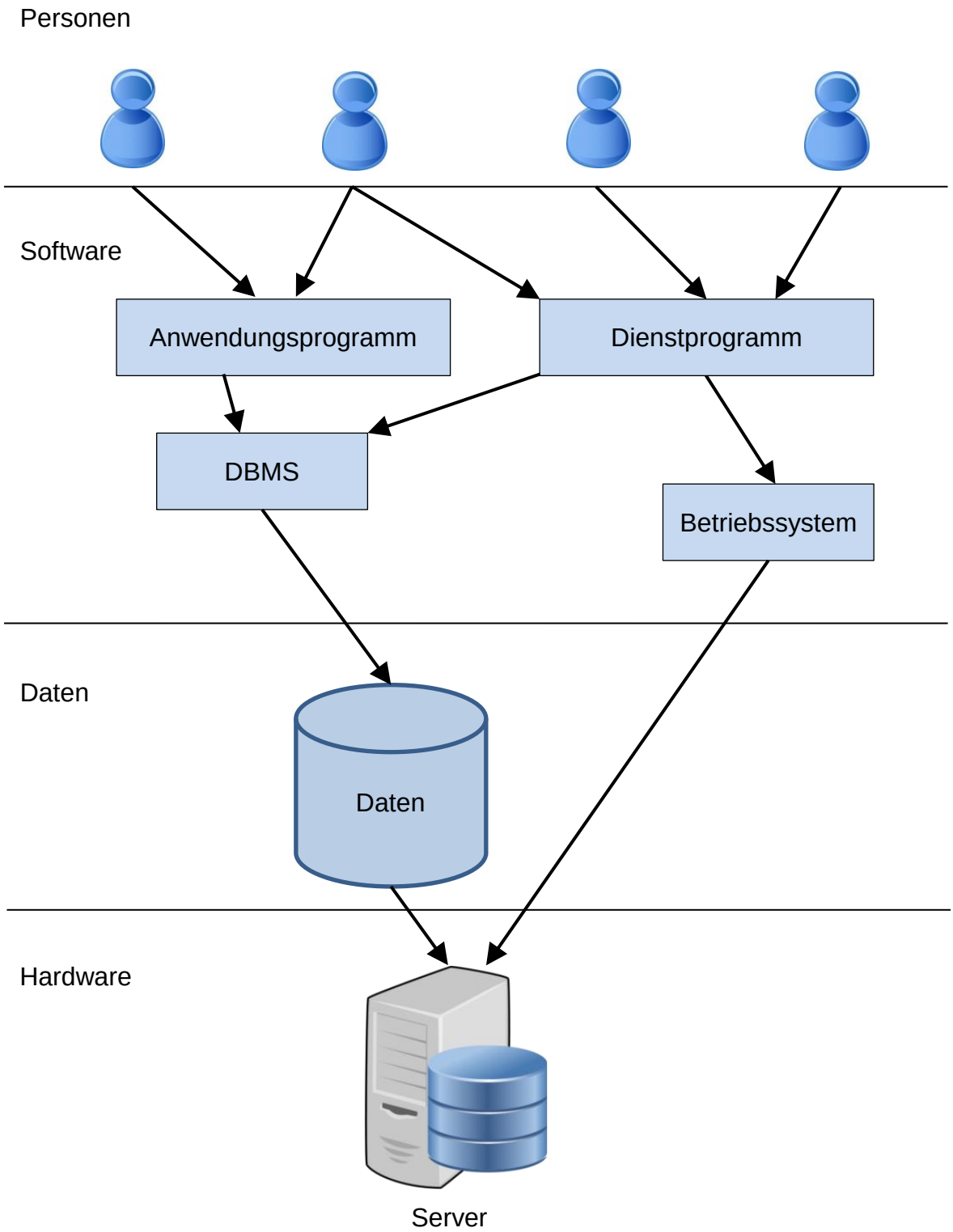


**Abbildung 20: Datenbanksystem (in Anlehnung an Geisler, 2011, S.44)**

- Hardware

  Auf dieser Schicht basiert das ganze System. Darunter werden alle physikalischen Geräte, aus denen das System aufgebaut ist, verstanden. Dazu zählt der Datenbankserver auf dem das Datenbankmanagementsystem ausgeführt wird und der die Daten speichert. Weiters gehören auch der Clientrechner, die Kabel, Hubs, Router, Bridges, Firewalls, usw., die Clientrechner und Server verbinden, dazu. Auch Peripheriegeräte wie Drucker, Speichermedien mit Sicherungskopien zählen zur Hardware.

- Daten

  Die Daten sind die Fakten die im Datenbanksystem gespeichert werden. Die einzige Komponente, welche auf die Daten zugreift ist, das Datenbankmanagementsystem. Üblicherweise wird der den Datendateien zugewiesene Speicherplatz vom Betriebssystem verwaltet. Bei großen kommerziellen Lösungen kann die Verwaltung der Datenspeicherung auch auf das DBMS umgelagert werden.

- Software

  Eine wichtige Komponente ist natürlich auch die Software für ein funktionierendes Datenbanksystem. Das Kernelement hierbei ist das DBMS, welches die Verwaltung der Daten übernimmt. Der Anwender der Datenbank kommt mit den Datenbankanwendungsprogrammen in Kontakt, welche als Windows-Anwendung oder auch als Webanwendung dargestellt werden. Mit den Dienstprogrammen können Wartungsarbeiten am Datenbanksystem, wie Datensicherung oder Überprüfung der Datenbank, durchgeführt werden. Der reibungslose Betrieb soll damit sichergestellt werden. Die Dienstprogramme werden in der Regel nur von Datenbankadministratoren, Datenbankdesignern und Programmierern verwendet. Das Betriebssystem des Severs und der Clientrechner gehören auch in diese Kategorie. Das Betriebssystem liefert in der Regel die Basis, auf der die Datenbankanwendungen und das DBMS aufbauen.

- Personen

  Die Personen die im Datenbanksystem arbeiten können in funktionale Gruppen unterteilt werden.

  Anwender: Verwenden nur Anwendungsprogramme mit denen Daten erfasst und ausgewertet werden.

  Datenbankadministratoren: Verwalten das Datenbanksystem und gewährleisten ein fehlerfreies, zuverlässiges System. Sorgen für genügend Speicherplatz, legen neue Benutzer an, vergeben Berechtigungen und führen Datensicherungen durch.

  Datenbankdesigner und Programmierer: Legen die Strukturen in der Datenbank an und tragen dafür Sorge, dass die Strukturen robust sind, und dass die Anwender auf einfache Weise darauf zugreifen können.

## 5.4 Datenbankmanagementsystem-Funktionen

Das DBMS stellt den zentralen Teil eines Datenbanksystems dar. Es ist das einzige Element, welches direkt auf die Daten Zugriff hat. Diese zentrale Rolle des DBMS wurde gewählt, da nur ein Programm in dieser Position alle Funktionen erfüllen kann, die notwendig sind um die Konsistenz und Integrität der Daten zu sichern. Für den Anwender der Datenbank sind die meisten Funktionen davon unsichtbar, da sie im Hintergrund ausgeführt werden (Abbildung 21).

- Verwaltung der Sicherheit

  Ein DBMS muss die Sicherheit der gespeicherten Daten gewährleisten, d.h. der Zugriff bestimmter Benutzer auf bestimmte Datenbankobjekte muss gewährleistet bzw. verweigert werden können. Zugriff bedeutet hier auch welche Art der Aktion ausgeführt werden kann, z.B. lesen, schreiben oder löschen.

- Verwaltung mehrerer Anwender

  Ist das DBMS nicht für den gleichzeitigen schreibenden Zugriff mehrerer Benutzer auf einen Datensatz ausgelegt, dann kann dies zur Korruption der Datenbank führen. Vernünftige DBMS verhindern dies indem der Datensatz beim ersten schreibenden Benutzerzugriff gesperrt wird und somit kein weiterer Benutzer schreibend darauf zugreifen kann.


**Abbildung 21: DBMS-Funktionen (in Anlehnung an Geisler, 2011, S. 49)**

- Datensicherung/Wiederherstellung

  Hardwarebedingte Ausfälle aber auch Benutzerbedingte Fehler machen eine Datensicherung unumgänglich. Redundante Festplattensysteme bieten zwar vor Hardwareausfällen Schutz aber nicht vor Benutzerfehlern. Moderne Datenbanksysteme beinhalten Tools, welche es ermöglichen die Datenbank zu sichern oder auch wiederherzustellen.

- Sprachen für Datenbankzugriff

  Meist wird vom Hersteller des Datenbanksystems eine Abfragesprache bereitgestellt. Über diese kann der Anwender mit der Datenbank kommunizieren. Die eigentliche Kommunikation findet meist über SQL statt, obwohl die Datenbankanwendungen meist grafische Benutzeroberflächen besitzen. Die getätigten Eingaben werden intern in SQL Befehle umgesetzt und an die Datenbank gesendet.

- Bereitstellung von Kommunikationswegen

  Die Bereitstellung von Kommunikationswegen zur Verständigung der Clientrechner mit dem Datenbankserver ist ein wichtiger Punkt. Dafür wird üblicherweise ein Kommunikationsprotokoll implementiert.

  Unter Middleware wird eine Softwarekomponente bezeichnet, welche als Zwischenstück von Datenbank und Datenbankanwendung fungiert und den Anwendungszugriff auf die Datenbank abstrahiert. Mit dieser einzigen Datenbankanwendung kann auf verschiedene Datenbanksysteme zugegriffen werden. Erreicht wird dies durch Verwendung datenbankspezifischer Treiber.

- Wahrung der Integrität

  Mittels des DBMS werden Regeln definiert, welche die Integrität gewährleisten. Damit werden die Datenredundanz und die Datenkonsistenz maximiert.

- Datenumwandlung/Präsentation

  Mittels der Datenumwandlung wird zwischen logischen und physikalischen Datentypen unterschieden, d.h. zwischen Eingabe- und Speicherformat. Durch die Umwandlung und Speicherung der logischen Datentypen als physikalische erreicht das DBMS Datenunabhängigkeit.

- Datenträgerverwaltung

  Wie vorhin erwähnt ist das DBMS das einzige Programm, welches auf die Daten Zugriff hat. Das DBMS erzeugt auf der Festplatte die Strukturen, um die Daten effizient und performant speichern zu können. Der Anwender ist somit

von der Aufgabe befreit, Funktionen zu programmieren, Daten auf die Festplatte zu schreiben oder zu lesen.

- Metadatenverwaltung
  Eine der wichtigsten Aufgaben des DBMS ist die Verwaltung der Metadaten, welche die gespeicherten Daten beschreiben. Die Metadaten der Datenbank sollte nie manuell geändert werden, sondern nur über die dafür vorgesehenen Funktionen. Da das DBMS diese Verwaltung übernimmt, ist der Anwendungsprogrammierer von der physikalischen Speicherung der Daten komplett abgekoppelt. Er greift lediglich auf die Repräsentation der Strukturen in Form von Tabellen und Feldern zu.

## 5.5 Datenbankmodelle

Für die Entwicklung einer Datenbank ist es von essentieller Bedeutung, die verwendeten Datenstrukturen und die sie verbindenden Beziehungen an das verwendete DBMS anzupassen, um dieses optimal nutzen zu können.
Unter einem Datenbankmodell wird eine abstrahierte Darstellung von Daten und den zwischen den Daten befindlichen Beziehungen verstanden. Datenbankmodelle können in konzeptionelle und implementative Modelle unterteilt werden.

Konzeptionelle Modelle befassen sich mit der logischen Struktur der Daten. Die erfassenden Daten sowie die zwischen ihnen bestehenden Beziehungen sollen erfasst und verstanden werden. Die Frage hierbei lautet hierbei eher, was in die Datenbank aufgenommen werden soll, und nicht wie dies erfolgen soll.

Implementative Modelle beschäftigen sich mit den Fragen, wie die Daten in der Datenbank gespeichert werden, wie die erzeugenden Strukturen und deren Beziehungen zueinander aussehen.

### 5.5.1 Hierarchischen Datenbanken

Die hierarchische Datenbank war eines der ersten Datenbankmodelle. Die Idee dafür entstand im Rahmen des Apollo-Weltraumprogramms. Größere Fertigungseinheiten sollten aus kleineren aufgebaut werden.

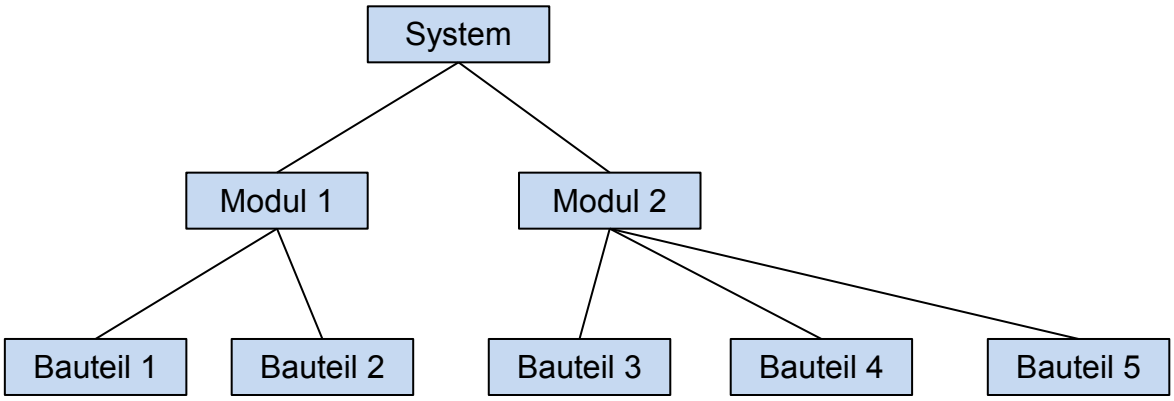


**Abbildung 22: Hierarchische Struktur**

Elemente in einer hierarchischen Struktur werden als Knoten bezeichnet. Der oberste Knoten, der den Ursprung der Struktur bildet, wird als Wurzelknoten bezeichnet. Dem Wurzelknoten sind alle anderen Knoten untergeordnet. Diese untergeordneten Knoten werden als Kindknoten bezeichnet. Damit die Kindknoten den Elternknoten richtig zugeordnet werden können, wird zusätzlich ein Grad angegeben. Modul 1 ist z.B. für den Knoten System ein Kindknoten ersten Grades. Ein Elternknoten kann beliebig viele Kindknoten besitzen. Ein Kindknoten kann aber maximal nur einen Elternknoten haben.

Über die hierarchische Datenstruktur wird die Datenintegrität erzwungen, denn ein Kindknoten muss immer einen Elternknoten besitzen, worüber er immer referenziert werden kann. Aufgrund der Verwaltung der Datentypen innerhalb der hierarchischen Datenbank wird Datenunabhängigkeit erreicht. Dieses Datenmodell ist vor allem zur Verwaltung großer Datenmengen, welche in 1:n Beziehungen zueinander stehen, und einer großen Anzahl an durchgeführten Transaktionen gut geeignet.

Das schwierige Management von hierarchischen Datenbanken sowie die komplexe Implementierung dieser Systeme haben dazu geführt, dass sie in den frühen 1980ern von anderen Datenbanksystemen abgelöst wurden. Programmierer und Datenbankadministratoren müssen eine genaue Vorstellung der physikalischen Struktur bei diesem Modell besitzen. Eine Umorganisation der Struktur erfordert, dass alle Anwendungsprogramme an diese Änderung angepasst werden müssen. Die hierarchische Datenbank besitzt somit keine strukturelle Unabhängigkeit. Das Finden bestimmter Daten erfordert die Navigation des Programmierers auf einem Pfad durch den Baum. Alle Elternknoten müssen durchlaufen werden, bevor auf einen Kindknoten zugegriffen werden kann. Der Anwendungsprogrammierer muss dafür die Datenbankstruktur genau kennen. Eine Änderung der hierarchischen Struktur führt dazu, dass die programmierten Dateizugriffspfade nicht mehr stimmen und die Anwendungsprogramme geändert werden müssen. Diese strukturelle Abhängigkeit hat Auswirkungen bis zum Anwender des Datenbanksystems, denn auch dieser muss denn genauen Pfad der von ihm benötigten Daten aus der Datenbank wissen.

Die hierarchische Struktur begünstigt auch Delete-Anomalien. Beim Löschen eines Unterbaums werden auch alle Knoten unterhalb des Wurzelknotens des Wurzelbaums gelöscht. Dadurch können versehentlich Daten gelöscht werden, wenn sie sich irgendwo im zu löschenden Unterbaum befinden.
Im hierarchischen Datenmodell können nur 1:n Beziehungen dargestellt werden. Ein Elternknoten kann beliebig viele Kindknoten besitzen, aber ein Kindknoten kann nur einen Elternknoten besitzen.

Das Ende der hierarchischen Datenbanken wurde durch das Fehlen vernünftiger Standards eingeläutet. Auf einem bestimmten System entwickelte Datenbankanwendungen konnten nur unter großem Aufwand auf ein Datenbanksystem eines anderen Herstellers portiert werden. Dieser Aufwand war fast so groß wie die Neuentwicklung des Systems.

### 5.5.2 Netzwerk-Datenbanken

Aufgrund der Hauptkritikpunkte des hierarchischen Datenbankmodells, der Unfähigkeit m:n Beziehungen darzustellen, sowie das Fehlen einheitlicher Standards, wurde das Netzwerk-Datenbankmodell entwickelt.
Die Einführung einer standardisierten Sprache zur Verwaltung und Manipulation der in der Datenbank gespeicherten Daten sollte erfolgen. Eine Unabhängigkeit des verwendeten DBMS sollte erreicht werden. Diese verabschiedeten Standards wurden von den Herstellern aber recht frei interpretiert, sodass die Vereinfachungen nicht erreicht wurden.

Das Netzwerkmodell ähnelt in vielen Bereichen dem hierarchischen Datenbankmodell. Der entscheidende Unterschied liegt aber darin, dass ein Kindknoten mehr als nur einen Elternknoten besitzen kann. Es wurde daher auch eine neue Terminologie eingeführt. Die Beziehung zwischen zwei Elementen wird als Menge bezeichnet. Die Elternknoten werden Besitzer der Menge und die Kindknoten als Mitglied der Menge bezeichnet. D.h. ein einzelnes Element kann gleichzeitig Mitglied mehrerer Mengen sein. Über diese Eigenschaft wird eine der größten Einschränkungen des hierarchischen Datenbankmodells, die Beschränkung auf 1:n Beziehungen, überwunden. Ein flexiblerer Zugriff auf die Daten der Datenbank ist dadurch möglich, da jedes Mengenmitglied mehr als einen Besitzer haben kann und somit unterschiedliche Suchwege existieren.

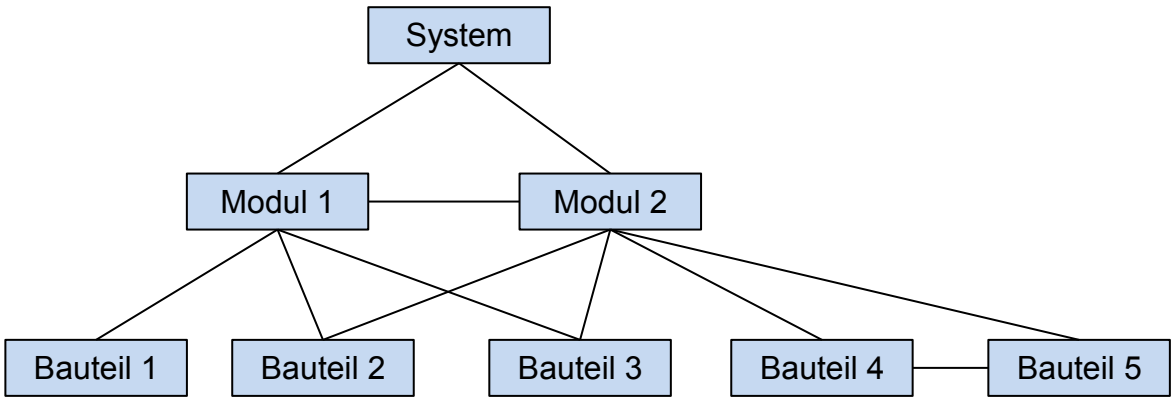


**Abbildung 23: Netzwerk-Datenbankmodell**

Die Integrität der Datenbank wird verbessert indem es nicht möglich ist ein Mengenmitglied ohne Besitzer zu definieren. Im Vergleich zum hierarchischen Modell muss hier der Anwendungsentwickler nicht mehr die komplette physikalische Struktur kennen. Er muss nur den Teil, welcher für seine Lösung relevant ist, kennen.
Auch wenn die definierten Standards für Netzwerk-Datenbankmodelle nicht komplett von den Herstellern implementiert wurden, ist der Import auf ein anderes DBMS dennoch leichter möglich als mit hierarchischen Datenbanken.

Das Netzwerkdatenmodell erfordert aber auch die Navigation zu den gewünschten Daten der Datenbank, wie das hierarchische Modell. Die Komplexität ist hierbei aber geringer. Entwickler und Anwender müssen auch hier die Struktur der Daten kennen, wobei aber nicht Kenntnisse der kompletten Struktur erforderlich sind, sondern nur über den für sich relevanten Teil. D.h. auch in diesem Modell tritt eine strukturelle Abhängigkeit auf, die es erfordert, dass bei Strukturänderungen der Datenbank auch alle Anwendungsprogramme angepasst werden müssen.

### 5.5.3 Relationale Datenbanken

Wie bei dem hierarchischen und Netzwerk-Datenbankmodell ersichtlich wurde, ist die strukturelle Abhängigkeit ein großes Problem. Auch kleine Änderungen an der Struktur der Datenbank haben zur Folge, dass die auf die Datenbank zugreifenden Anwendungsprogramme ebenfalls geändert werden müssen.
Daraus folgte, dass Spezialisten benötigt wurden, um die Datenbank effektiv nutzen zu können. Für die meisten der Anwender war es zu kompliziert den Überblick über die Struktur der Datenbank zu bewahren. Der Ruf nach Ad-hoc Abfragemöglichkeiten wurde immer lauter. Bereits 1970 wurde das relationale Datenbankmodell entwickelt, aber die Hardware war für die praktische Umsetzung noch nicht so weit entwickelt. Relationale Datenbanken gehören heute zu den am weitest verbreiteten Datenbanksystemen.

Das Datenbankmanagementsystem wird beim relationalen Datenbankmodell RDBMS genannt. Das RDBMS enthält dieselben grundlegenden Funktionen wie das DBMS des hierarchischen und Netzwerk-Datenbanksystem, aber auch einige Erweiterungen. Eine komplette Kapselung der physikalischen Datenspeicherung wird mit dem RDBMS erreicht. Der Anwender braucht sich hierbei nur mit der logischen Struktur der Daten zu beschäftigen.

Das grundlegende Konzept der relationalen Datenbank ist die Tabelle. Die Zeilen der Tabelle repräsentieren die Datensätze und die Spalten die Datenfelder. Eine Zelle der Tabelle beinhaltet den Wert, welcher ein bestimmter Datensatz in einem bestimmten Datenfeld enthält.
Beziehungen zwischen Tabellen werden über Primär-/Fremdschlüssel hergestellt. In einer der beiden Tabellen existiert ein Primärschlüssel der jeden Datensatz eindeutig kennzeichnet. Der Primärschlüssel kann beispielweise eine fortlaufende Nummer sein. In der anderen der beiden Tabellen existiert der Fremdschlüssel, dies ist der Verweis auf den Primärschlüssel der ersten Tabelle. Die Tabelle mit dem Primärschlüssel wird als Mastertabelle bezeichnet und die mit dem Fremdschlüssel als Detailtabelle.

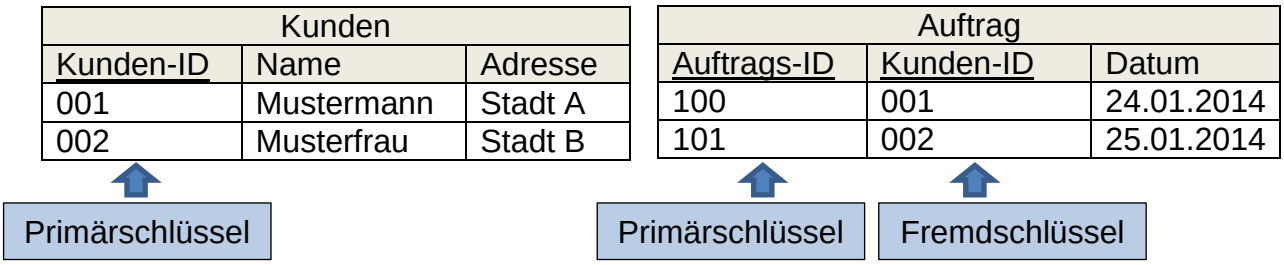


**Abbildung 24: Relationale Datenbank**

Der größte Vorteil der relationalen Datenbank liegt in der strukturellen Unabhängigkeit. Im Vergleich zur hierarchischen und Netzwerk-Datenbank gibt es hierbei keine Struktur durch die man sich navigieren muss, um an die Daten in der Datenbank zu gelangen. Der Zugriff des RDBMS auf die Daten ändert sich nicht bei Änderungen in der Datenbank. Bei Änderungen der Baumstruktur in hierarchischen

und Netzwerk-Datenbanken ändert sich auch der Zugriffspfad, über den das DBMS die Daten erreicht.
Durch die Kapselung der physischen Struktur der Datenspeicherung durch das RDBMS muss sich der Datenbankdesigner keine Gedanken über die physikalische Speicherung der Daten machen. Er kann sich ausschließlich den logischen Konzepten der Datenbank widmen.

Die strukturelle Unabhängigkeit sowie die Datenunabhängigkeit machen es einfach, eine relationale Datenbank zu entwerfen und verwalten. Diese Flexibilität zeigt sich auch in der Möglichkeit von Ad-hoc Abfragen. Die physikalische Komplexität der Datenbank wird mittels den RDBMS komplett vor den Datenbankdesignern und Anwender versteckt. Ein RDMS benötigt aber mehr Ressourcen als die DBMS der hierarchischen und Netzwerk-Datenbanken.

## 5.6 Datenbankanwendungen

### 5.6.1 Grundlagen

Die Schichtenarchitektur ist ein weit verbreitetes Strukturierungsmodell. Die einzelnen Schichten lassen sich physikalisch auf verschiedene Hardwarebereiche aufteilen. Je nach Anzahl der Hardwarebausteine spricht man von ein-, zwei- und n-schichtigen Datenbankanwendungen.

Datenbankanwendungen verbinden folgende drei Schichten:

- Präsentationsschicht
  Beinhaltet Funktionen zu Darstellung der Daten und Dateneingabe. Elemente davon sind Userinterfaces und Berichte.

- Geschäftsschicht
  Dient der Implementierung der Geschäftsregeln. Diese legen Regeln für die Daten in der Datenbank fest, z.B. für die Überprüfung von Wertebereichen.

Funktionen zur Durchführung von Berechnungen oder Abfragen werde hier ebenfalls bereitgestellt.

- Datenschicht
  Diese stellt Funktionen zur Suche, Speicherung, Ansicht und Integrität der Daten bereit.

### 5.6.2 Einschichtige Datenbankanwendung

Damit werden Anwendungen bezeichnet, in der Präsentations-, Geschäfts- und Datenbankschicht auf einem Computer in einem einzigen Programm realisiert werden (Abbildung 25). Diese Anwendungen greifen direkt auf die Datendateien, welche auf der lokalen Festplatte liegen, zu. Es erfolgt eine direkte Implementierung der Datenschicht in die Anwendung.

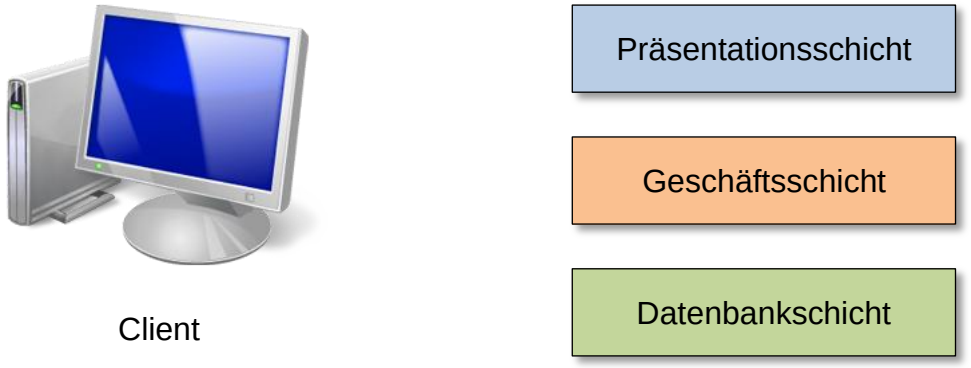


**Abbildung 25: Einschichtige Datenbankanwendung**

### 5.6.3 Zweischichtige Datenbankanwendung

Eine weit verbreitete Client-Server-Architektur ist die zweischichtige Datenbankanwendung. Die Clients greifen hierbei über ein Netzwerk auf einen zentralen Server zu. Die Client-Programme fordern die Daten vom Server an. Der

Server koordiniert die Zugriffe mehrerer Clients und liefert die Daten an diese zurück. Funktionen zur Benutzersteuerung und Datensicherung sind im Datenserver implementiert.
Die Datenbankanwendung kann mittels zwei verschiedener Modelle realisiert werden.

#### 5.6.3.1 Intelligenter Server

Die Geschäftsschicht wird hierbei auf dem Server implementiert. Auf dem Client läuft nur die Präsentationsschicht ab (Abbildung 26).
Der Vorteil beim intelligenten Server ist, dass die umfangreichen Berechnungen mit den Daten direkt von diesem durchgeführt werden können. D.h. dieses System ist vor allem dann in Betracht zu ziehen, wenn die Clientrechner nicht genügend Ressourcen für die Berechnungen zur Verfügung haben.
Der Server muss für die zentrale Bearbeitung sehr große Ressourcen besitzen, da es sonst bei vielen Benutzern zu einem Engpass kommen kann. Die Netzwerkbenutzung ist nicht optimal, da Daten vom Server zum Client und bei jedem Bearbeitungsschritt vom Client zum Server zurückgesendet werden müssen. Bei manchen Aufgabenstellungen muss diese Datenübertragung zum Server sogar mehrfach erfolgen.

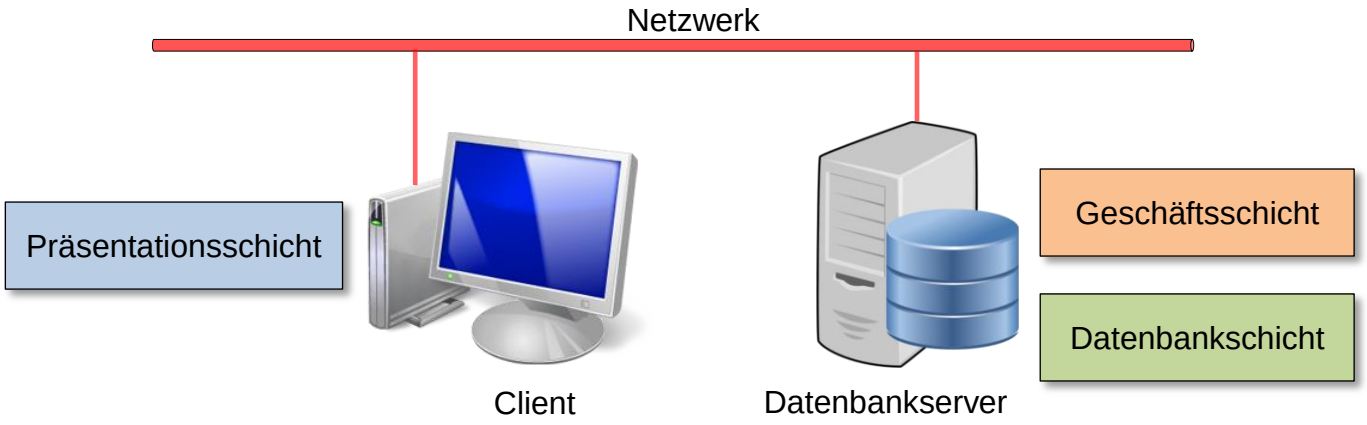


**Abbildung 26: Intelligenter Server**

#### 5.6.3.2 Intelligenter Client

Die Präsentationsschicht und die Geschäftsschicht werden hier auf dem Client implementiert. Der Datenbankserver enthält nur die Datenschicht (Abbildung 27).
Der Vorteil dieses Modell besteht darin, dass die Daten nur einmal zum Client übermittelt werden müssen und dort bearbeitet werden können. Dieses Prinzip wird von den meisten Client-Server-Datenbankanwendungen verwendet.
In der Praxis hingegen wird dieses Modell jedoch nicht so strikt gehandhabt. Die Geschäftsschicht wird meist zwischen Client und Server aufgeteilt, sodass datenintensive Funktionen am Server ausgeführt werden. Damit kann die Netzwerklast verringert werden. Um die Serverbelastung zu reduzieren werden Funktionen mit kleineren Datenmengen von den Clients ausgeführt.

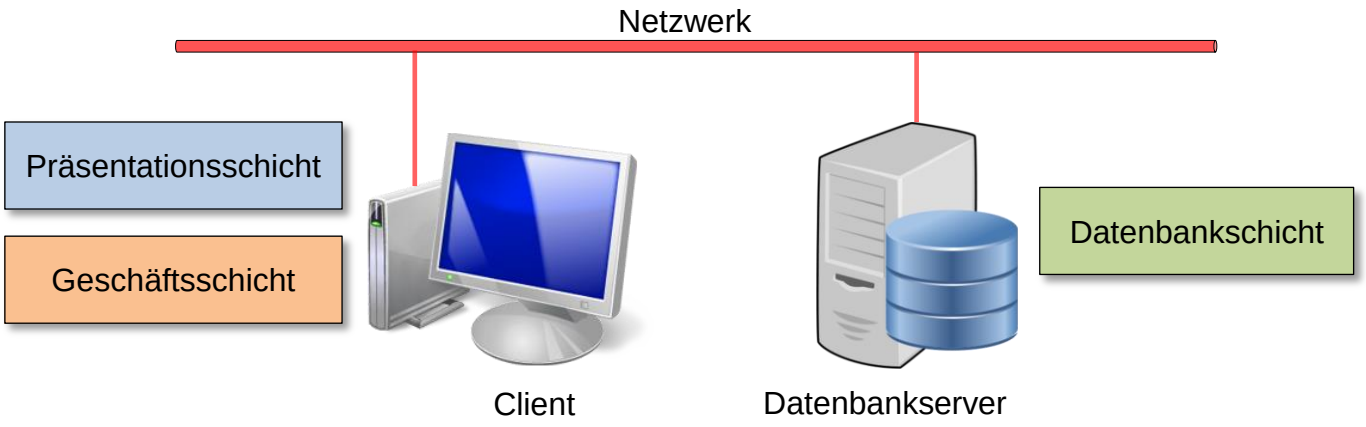


**Abbildung 27: Intelligenter Client**

### 5.6.4 n-schichtige Datenbankanwendung

Bei der n-schichtigen Datenbankanwendung werden alle drei Schichten strikt getrennt. Jede Schicht wird auf einem eigenen Rechner ausgeführt. Auf dem Clientrechner läuft somit nur die Präsentationsschicht. Daneben gibt es noch je einen Server für die Geschäfts- und Datenbankschicht. Der Server für die Geschäftsschicht wird Anwendungsserver genannt (Abbildung 28).

Die Trennung der Daten- und Geschäftsschicht macht eine größere Skalierbarkeit der Anwendung möglich, da bei Benötigung zusätzlicher Ressourcen ein Anwendungs- oder Datenbankserver hinzugefügt werden kann.

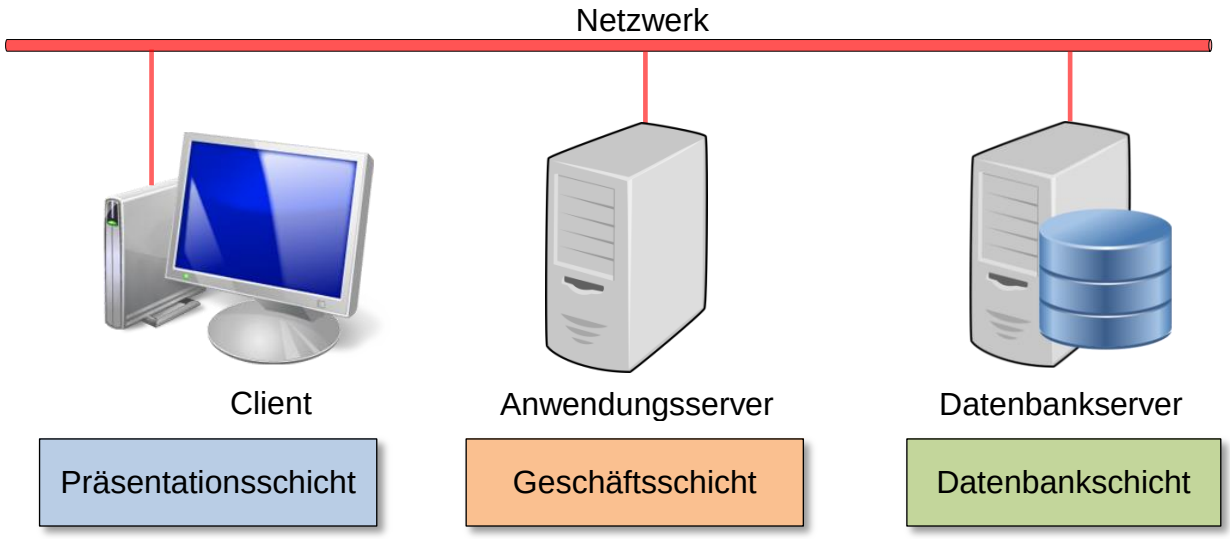


**Abbildung 28: n-schichtige Anwendung**

## 5.7 Middleware

Aus einem Anwendungsprogramm heraus wird normalerweise nicht direkt auf eine Datenbank zugegriffen. Für den Datenbankzugriff wird eine Zwischenschicht, sozusagen eine Kopplung zwischen Datenbank und Anwendungsprogramm, benutzt. Dadurch ergibt sich der Vorteil, dass in der Anwendung nur über einen logischen Namen auf die Datenbank zugegriffen wird. Die Middleware ist somit eine datenunabhängige Schnittstelle für die Programmierung. Bei Änderungen in der Datenbank muss lediglich eine Änderung in dieser Zwischenschicht erfolgen. Das Anwendungsprogramm kann unverändert bleiben (Abbildung 29).

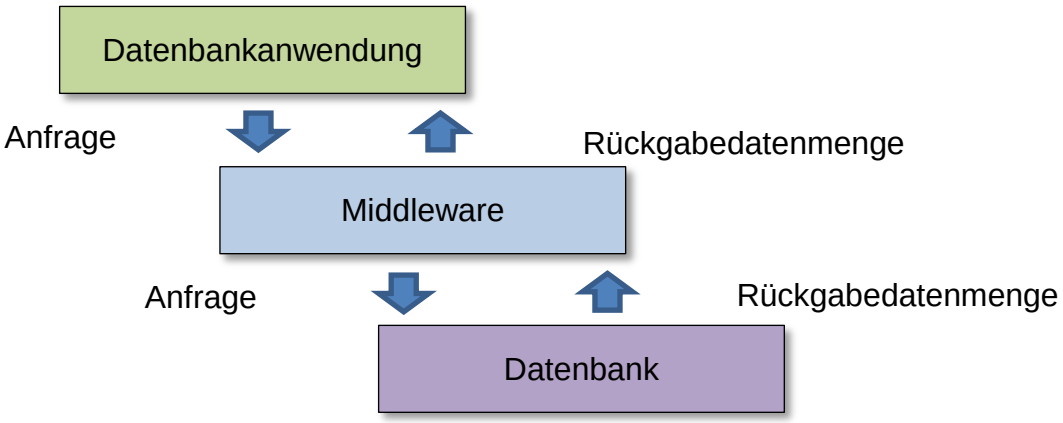


**Abbildung 29: Middleware**

### 5.7.1 ADO (ActiveX Data Objects)

ADO wurde von Microsoft entwickelt und soll einen universellen Datenzugriff zur Verfügung stellen. Dieser universelle Datenzugriff wird durch OLE DB (Object Linking and Embedding, Database) und ADO als Programmierschnittstelle ermöglicht.

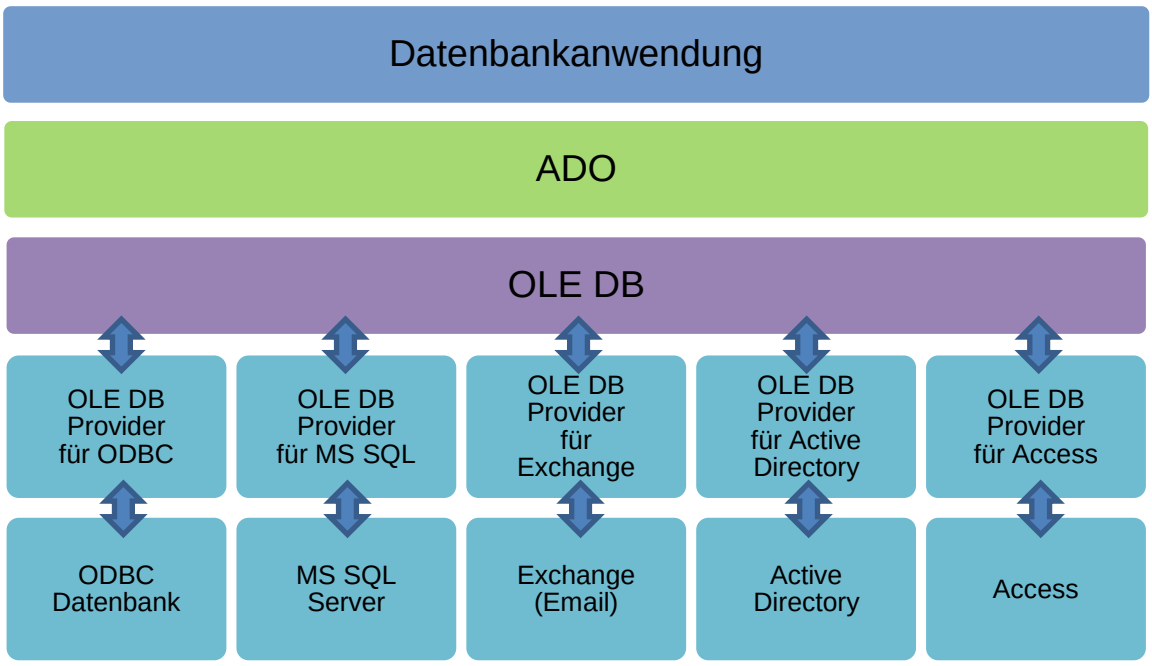


**Abbildung 30: ADO-Architektur (in Anlehnung an Geisler, 2011, S.79)**

ADO stützt sich auf datenbankspezifische Treiber, welche OLE DB Provider heißen. Für Datenbanken, für die es keine OLE DB Provider gibt, bietet ADO die Möglichkeit über ODBC zuzugreifen, wobei dies langsamer ist als der direkte Zugriff über den jeweilige OLE DB Provider.
ADO besteht aus mehreren DLLs, wobei verschiedene Microsoft Produkte jeweils auf verschiedene ADO-Versionen aufbaut. Die ADO-Version richtet sich somit nach Installationsreihenfolge und installierten Software am Rechner.
Der Vorteil von ADO besteht darin, dass OLE DB Provider nicht nur für die bekannten Datenbanksysteme existieren, sondern auch für andere Anwendungen wie beispielsweise MS Excel. (Geisler, 2011, S. 79ff)

### 5.7.2 ADO.NET

Genau wie ADO kann die grundlegend überarbeitete ADO.NET Version auf verschiedene, auch nicht relationale, Datenquellen zugreifen. Bei der Entwicklung von ADO waren Client-Server Anwendungen weit verbreitet. Durch eine derartige Struktur, z.B. im LAN von Unternehmen, war eine zentrale Annahme von ADO, dass eine ständig eindeutige Verbindung zwischen Client und Server vorhanden ist. Mittlerweile sind aber viele Anwendungen für webbasierte Intra- und Internetverbindungen konzipiert. Dabei wird im Gegensatz zur Client-Server Architektur der Datenaustausch über zustandslose http-Protokolle abgewickelt. ADO unterstützt daher nur noch eine vorwärtsgerichtete, lesende Navigation durch Datenmengen, womit der Netzwerkverkehr und Ressourcenverbrauch auf dem Datenbankserver wichtig ist.
ADO.NET bietet aufgrund der neuen Architektur nun auch weit mehr Möglichkeiten bei der Arbeit mit zwischengespeicherten Daten. Es gibt in der Architektur eine Unterteilung zwischen .NET DataProvider und .NET DataSet (Abbildung 31). Alles für die Kommunikation benötigte ist in .NET DataProvider implementiert, und alle für die Arbeit mit den Daten auf dem Client in .NET DataSet. Zu den Hauptaufgaben des .NET DataProvider gehören das Befüllen der .NET DataSets, und darin vorgenommene Änderungen in die Datenbank zu schreiben. (Geisler, 2011, S. 81f)

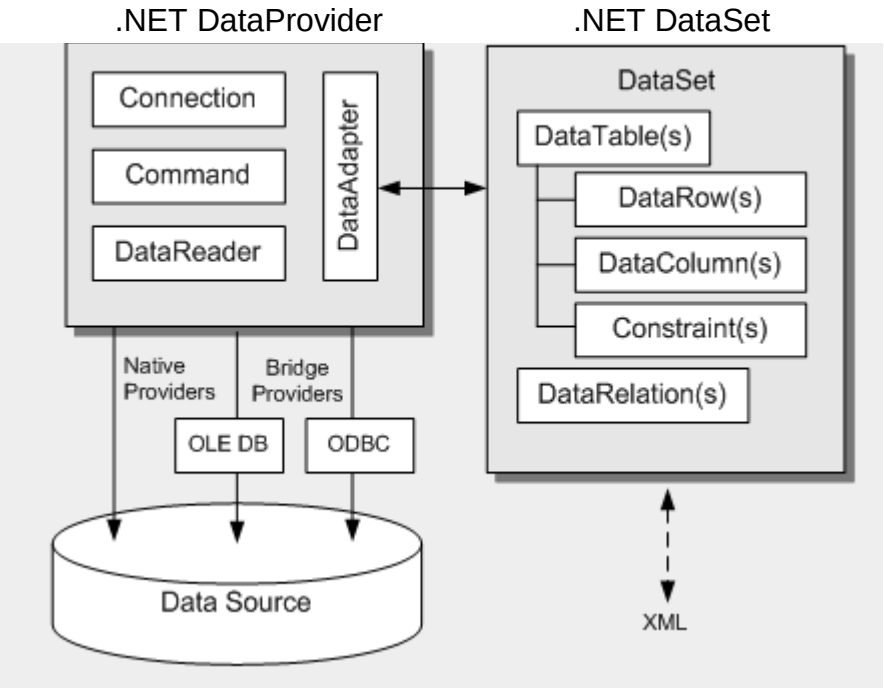


**Abbildung 31: ADO.NET Architektur (in Anlehnung an Geisler, 2011, S. 82)**

## 5.8 Normalisierung

Unter Normalisierung versteht man einen Prozess, welcher zur Erstellung einer effizienten Tabellenstruktur einer relationalen Datenbank benutzt wird. Damit sollen die in der Datenbank vorhandenen Datenredundanzen reduziert und somit Anomalien vermieden werden. Datenredundanzen, die nicht durch das Datenbanksystem eingeführt wurden, können natürlich nicht eliminiert werden, sie können nur in kontrollierbare Bahnen gelenkt werden.

In Abbildung 32 sieht man in der oberen Tabelle, dass die zu verwaltenden Daten eine Datenredundanz, durch mehrfaches auftreten der Anrede „Herr“, erzeugen. Werden die Anreden in eine eigene Tabelle verschoben und den Kunden lediglich ein Fremdschlüssel zu der Tabelle zugewiesen, so wird die Redundanz besser kontrollierbar. Es muss dann nur mehr der Fremdschlüssel in der Kundentabelle eingegeben werden, die Wahrscheinlichkeit von Tippfehler ist somit geringer, und

Änderungen der Anrede sind einfacher, da sie nur mehr an einer Stelle geändert werden müssen.

| Kunden | | |
|---|---|---|
| Kunden-ID | Anrede | Name |
| 101 | Herr | Mustermann |
| 102 | Frau | Musterfrau |
| 103 | Herr | Mannmuster |
| 104 | Herr | Mann |

Redundanz

| Kunden | | |
|---|---|---|
| Kunden-ID | Anrede-ID | Name |
| 101 | 1 | Mustermann |
| 102 | 2 | Musterfrau |
| 103 | 1 | Mannmuster |
| 104 | 1 | Mann |

| Anreden | |
|---|---|
| Anrede-ID | Anrede |
| 1 | Herr |
| 2 | Frau |

**Abbildung 32: Datenredundanz**

Der Nachteil der Normalisierung liegt darin, dass der Aufwand steigt, um die benötigten Daten wiederzugeben. Um die Anrede der Kunden zu erhalten, muss die Beziehung zwischen der Tabelle Kunden und Anreden nämlich aufgelöst werden. Dies bedingt einen erhöhten Programmieraufwand, sowie eine erhöhte Komplexität des Datenmodells. Das Datenbanksystem benötigt somit mehr Ressourcen für die Auflösung der Beziehungen. Das System ist unter Umständen nicht mehr so performant wie bei der Lösung mit einer Tabelle. Hierbei gilt es, zwischen Performance und Stabilität des Datenbankmodells abzuwägen.

In der Praxis wird daher das Datenbankmodell komplett normalisiert und danach an gezielten Stellen, wo die Performance Priorität hat, wieder denormalisiert. An diesen Stellen wird somit bewusst eine höhere Datenredundanz zugelassen.

Die meist verwendeten Normalformen, worauf die Normalisierung beruht, sind die erste, zweite, dritte und die Boyce-Codd-Normalform. Die einzelnen Normalformen bauen aufeinander auf, d.h. eine Datenbank in dritter Normalform befindet sich auch automatisch in erster und zweiter. Die in der Praxis meist verwendeten Anwendungen beruhen auf Datenbanken, deren Datenmodell sich in der dritten

Normalform befindet. Wie vorhin erwähnt, ist es nicht immer erstrebenswert eine möglichst hohe Normalform zu verwenden, da das Datenmodell damit immer komplexer wird und es somit schwieriger wird, Daten aus der Datenbank auszulesen.

### 5.8.1 Erste Normalform

Eine Zuordnung von mehreren Datensätzen zu einem Datensatz wird als Wiederholungsgruppe bezeichnet. Relationale Datenbanken dürfen keine Wiederholungsgruppen besitzen, jede einzelne Zeile einer relationalen Tabelle muss eine eigene Entität darstellen.
Eine Tabelle befindet sich in erster Normalform, wenn es keine Wiederholgruppen gibt, alle Schlüsselattribute vorhanden sind, und alle Nicht-Schlüsselattribute vom Primärschlüssel oder einem Teil davon abhängen.
Jedes in der Tabelle enthaltene Attribut ist elementar.

| Schüler-ID | Klasse | Name | Klassenvorstand | LV-ID | LV-Titel |
|---|---|---|---|---|---|
| 101 | 11 | Hans H. | Huber | 311<br>305<br>301 | Mathematik<br>Statik<br>Dynamik |
| 102 | 12 | Peter P. | Muster | 305 | Statik |
| 103 | 13 | Franz F. | Marko | 301 | Dynamik |

| Schüler-ID | Klasse | Name | Klassenvorstand | LV-ID | LV-Titel |
|---|---|---|---|---|---|
| 101 | 11 | Hans H. | Huber | 311 | Mathematik |
| 101 | 11 | Hans H. | Huber | 305 | Statik |
| 101 | 11 | Hans H. | Huber | 301 | Dynamik |
| 102 | 12 | Peter P. | Muster | 305 | Statik |
| 103 | 13 | Franz F. | Marko | 301 | Dynamik |

**Abbildung 33: Beispiel Überführung 1.NF**

### 5.8.2 Zweite Normalform

Die erste Normalform schützt nicht vor Delete-, Insert- und einigen Arten der Update-Anomalien. Deshalb ist eine Überführung in die zweite Normalform sinnvoll.
Eine Tabelle befindet sich in der zweiten Normalform, wenn sie sich in der ersten befindet und es keine teilweisen Abhängigkeiten gibt. D.h. jedes Nicht-Schlüsselattribut ist voll funktional abhängig von einem Primärschlüssel. Tabellen mit einem Primärschlüsselattribut sind somit automatisch in der zweiten Normalform, wenn sie sich in der ersten befinden.
Jedes Attribut hängt vollständig von einem Schlüssel ab oder ist selbst ein Schlüssel.

| Schüler-ID | Klasse | Name | Klassenvorstand |
|---|---|---|---|
| 101 | 11 | Hans H. | Huber |
| 102 | 12 | Peter P. | Muster |
| 103 | 13 | Franz F. | Marko |

| LV-ID | LV-Titel |
|---|---|
| 311 | Mathematik |
| 305 | Statik |
| 301 | Dynamik |

| Schüler-ID | LV-ID |
|---|---|
| 101 | 311 |
| 101 | 305 |
| 101 | 301 |
| 102 | 305 |
| 103 | 301 |

**Abbildung 34: Beispiel Überführung 2.NF**

Aus dem Schlüssel Schüler-ID lässt sich in dem Beispiel eindeutig auf Name, Klasse und Klassenvorstand schließen, sowie aus Schlüssel LV-ID auf den LV-Titel (Abbildung 34).

### 5.8.3 Dritte Normalform

Die zweite Normalform hat noch eine Anfälligkeit gegen Update-Anomalien vor allem auf transitiv und funktional abhängige Felder.
Eine Tabelle befindet sich in der dritten Normalform, wenn sie sich in der zweiten Normalform befindet und keine transitiven oder funktionalen Abhängigkeiten

vorhanden sind. Aus keinem Nichtschlüsselattribut darf ein anderes Nichtschlüsselattribut folgen.

Jedes Nicht-Schlüsselattribut muss nichttransitiv vom Primärschlüssel abhängen.

Dadurch, dass in der dritten Normalform keinen Datenanomalien auftreten können, ist sie die gebräuchlichste Normalform.

| Schüler-ID | Name | Klasse |
|---|---|---|
| 101 | Hans H. | 11 |
| 102 | Peter P. | 12 |
| 103 | Franz F. | 13 |

| LV-ID | LV-Titel |
|---|---|
| 311 | Mathematik |
| 305 | Statik |
| 301 | Dynamik |

| Schüler-ID | LV-ID |
|---|---|
| 101 | 311 |
| 101 | 305 |
| 101 | 301 |
| 102 | 305 |
| 103 | 301 |

| Klasse | Klassenvorstand |
|---|---|
| 11 | Huber |
| 12 | Muster |
| 13 | Marko |

**Abbildung 35: Beispiel Überführung 3.NF**

Zum Beispiel: Aus dem Nichtschlüsselattribut Klasse (Abbildung 34) folgt aber das Attribut Klassenvorstand. Daher werden diese beiden Attribute in eine neue Relation gebracht (Abbildung 35).

# 6 Konzept

Im Zuge der Diplomarbeit sollte für einen Industriepartner eine objektorientierte Methodik zur transparenten Verknüpfung von kundenrelevanten Fahrzeugeigenschaften und Bauteileigenschaften konzipiert werden.
Durch diese Verknüpfung sollen die Auswirkungen von Änderungen an Fahrzeugelementen, insbesondere in Hinsicht auf deren zur Herstellung angewandten Technologien, auf die Fahrzeugeigenschaften erkennbar werden. Der Vergleich verschiedener Fahrzeugvarianten soll ermöglicht werden.
Mit der Methodik soll auch eine Wissensdatenbank angelegt werden. Die Dokumentation der Abhängigkeiten und des Zustandekommens von Eigenschaften, sowie die Dokumentation von vorgenommenen Änderungen an Fahrzeugelementen sollen eine spätere Nachvollziehbarkeit gewährleisten.
Die Umsetzung der Aufgabenstellung soll in VB.NET erfolgen.

Die Aufgaben der Methodik wurden in folgende Teilbereiche gegliedert:

- Verwaltung der hierarchischen Fahrzeugstruktur
- Verwaltung der Fahrzeugeigenschaften und Erfassung ihrer Wechselwirkungen
- Verknüpfung von Fahrzeugeigenschaften mit Fahrzeugelementen
    - Einführung eines Bewertungsvorganges
- Dokumentationsmöglichkeit bei sämtlichen Schritten

## 6.1 Bisherige Lösung

Für die bisherige Lösung beim Industriepartner wurde der Einfluss von Modulen auf bestimmte Fahrzeugeigenschaften bewertet und eine Matrix aufgestellt (Abbildung 36).

Im Anschluss erfolgte die Bewertung der Matrix *Kundeneigenschaften-Fahrzeugeigenschaften*. Den vorhin erwähnten Fahrzeugeigenschaften werden die vom Kunden wahrgenommenen Eigenschaften gegenübergestellt, und der Einfluss der Fahrzeugeigenschaften darauf bewertet (Abbildung 37).

Um einen Variantenvergleich erstellen zu können, wurde eine Basiswertung geschaffen, wobei die Kundeneigenschaften ein Rating bekamen. Mit diesem Faktor wurden die jeweiligen Kundeneigenschaften-Fahrzeugeigenschaften-Bewertungen aus Abbildung 37 aufgewertet, siehe Abbildung 38. Die Summe davon wird mit der Modul-Fahrzeugeigenschaftsbewertung (aus Abbildung 36, in dieser Abbildung ist die Reihenfolge der Eigenschaften anders) multipliziert. Die ∑a dieser erhaltenen Werte wird mit den vorhin aufgewerteten Werten der Kundeneigenschaften-Fahrzeugeigenschaftenmatrix multipliziert, und dann zeilenweise für jede Kundeneigenschaft eine Summe gebildet. Die Summe für jede Kundeneigenschaft wird mit der Gesamtsumme dieser Werte aller Kundeneigenschaften, wobei diese noch mit der Gesamtpunktezahl ∑b dividiert wird, dividiert. Somit wird ein Basiswert für jede Kundeneigenschaft erhalten.

Für verschiedene Varianten werden die nun vorhin als ∑a erhaltenen Werte mit einem Faktor aufgewertet und die Rechnung erneut durchgeführt (Abbildung 40).

Matrix 1

| Modul | Modul | Str. | Modulbenennung | mobility | stiffness | Stiffness: static | Stiffness: dinamic | Stiffness: bending | |
|---|---|---|---|---|---|---|---|---|---|
| 21 | 1 | 1 | Aufbau | 1 | 1 | 1 | 1 | 1 | |
| 2100 | 100 | 2 | Rohbau (Biw) | 1 | 1 | 1 | 1 | 1 | |
| 2101 | 101 | 2 | Karosse (painted body) | 1 | 1 | 1 | 1 | 1 | |
| 210102 | 10102 | 3 | Vorderwagen (front end) | 1 | 1 | 1 | 1 | | |
| 210104 | 10104 | 3 | Unterbau mitte / Boden mitte (chassis middle / floor center) | 1 | 1 | 1 | | 1 | |
| 210105 | 10105 | 3 | Hinterbau (End body) | 1 | 1 | 1 | | 1 | |
| 210106 | 10106 | 3 | Zelle (Gerippe, Dach, Heckabschluss) (Cell) | 1 | 0 | | | | |
| 210108 | 10108 | 3 | Lack- und Nahtversiegelung | 1 | 1 | | | 1 | |
| 210110 | 10110 | 3 | Ueberrollbuegel | 1 | 1 | 1 | 1 | | |
| 2102 | 102 | 2 | Exterieur (Exterior) | 1 | 0 | | | | |

**Abbildung 36: NES**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| performance | | | | 1 | | 1 | 2 | | 1 | 2 | 2 | 1 | | |
| quality | | 1 | 1 | 1 | | | 2 | 2 | 1 | 2 | 2 | 1 | | |
| consumer's economy | | 1 | 1 | 1 | 1 | | 2 | 1 | 3 | 1 | 2 | 2 | | |
| comfort | | | | | 2 | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | |
| information & Entertainment | | | | | | 2 | | | 1 | 3 | | | | |
| safety | | 1 | 1 | 1 | 2 | 2 | | | | | 3 | 3 | 1 | |
| styling | | 1 | 1 | 1 | | 2 | | 2 | 2 | | 2 | 2 | | |
| environment | | | | | 2 | | 2 | 2 | 1 | 1 | 1 | 1 | 1 | |
| roominess | | | | | | 2 | | 1 | | | 1 | 1 | | |
| vehicle dynamics | | | | | 1 | | 1 | 2 | | | | | 1 | |

Matrix 2

| Modul | Modul | Str. | Modulbenennung | corrosion | corrosion: metal | corrosion: plastic | drivetrain | ergonomics | mobility | aerodynamics | assembility: ease of assembly | thermal | cabin cooling | cabin heating | control |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2101 | 101 | 2 | Karosse (painted body) | 2 | 2 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 210102 | 10102 | 3 | Vorderwagen (front end) | 2 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 210104 | 10104 | 3 | Unterbau mitte / Boden mitte (chassis middle / floor center) | 2 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 210105 | 10105 | 3 | Hinterbau (End body) | 2 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 210106 | 10106 | 3 | Zelle (Gerippe, Dach, Heckabschluss) (Cell) | 2 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Abbildung 37: NES_Sortiert**

| | My_Concept (Rating 10 - 1, 10 is best) |
|---|---|
| performance | 10 |
| quality | 8 |
| consumer's economy | 10 |
| comfort | 7 |
| information & Entertainment | 5 |
| safety | 8 |
| styling | 10 |
| environment | 3 |
| roominess | 10 |
| vehicle dynamics | 7 |

∑b

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 0 | 0 | 0 | 20 | 0 | 2 | | | | | |
| | | | | 16 | 16 | 16 | 0 | 0 | | | | | | |
| | | | | 20 | 20 | 20 | 20 | 0 | 1 | | | | | |
| | | | | 0 | 0 | 0 | 7 | 14 | | | | | | |
| | | | | 0 | 0 | 0 | 0 | 5 | | | | | | |
| | | | | 16 | 16 | 16 | 8 | 8 | | | | | | |
| | | | | 20 | 20 | 20 | 0 | 10 | | | | | | |
| | | | | 0 | 0 | 0 | 3 | 0 | | | | | | |
| | | | | 0 | 0 | 0 | 0 | 10 | | | | | | |
| | | | | 0 | 0 | 0 | 14 | 0 | 1 | | | | | |

=10*2

| Modul | Modul | Str. | Modulbenennung | ∑ Modulaktivitäten | corrosion | corrosion: metal | corrosion: plastic | drivetrain | ergonomics | mobility |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Produktqualifizierung | 3068 | 72 | 72 | 72 | 72 | 47 | 54 |
| | | | Prio | | 1 | 1 | 1 | 1 | 1 | 1 |
| 21 | 1 | 1 | Aufbau | 4172 | 144 | 144 | 144 | 0 | 94 | 108 |
| 2100 | 100 | 2 | Rohbau (Biw) | 3134 | 72 | 72 | 0 | 0 | 94 | 108 |
| 2101 | 101 | 2 | Karosse (painted body) | 2552 | 72 | 72 | 0 | 0 | 94 | 108 |
| 210102 | 10102 | 3 | Vorderwagen (front end) | 1526 | 72 | 72 | 0 | 0 | 0 | 108 |
| 210104 | 10104 | 3 | Unterbau mitte / Boden mitte (chassis middle / floor center) | 1192 | 72 | 72 | 0 | 0 | 0 | 108 |

=72*2

∑a

**Abbildung 38: Funktion marktgetrieben**

| | | My_Concept (Rating 10 - 1, 10 is best) | Variante zu Basis | Basis Wertung | Variante | | | | |
|---|---|---|---|---|---|---|---|---|---|
| performance | | 10 | 103% | 8,58 | 8,84 | 0 | 0 | 0 | 20 |
| quality | | 8 | 103% | 8,82 | 9,12 | 16 | 16 | 16 | 0 |
| consumer's economy | | 10 | 103% | 14,47 | 14,90 | 20 | 20 | 20 | 20 |
| comfort | | 7 | 103% | 6,69 | 6,90 | 0 | 0 | 0 | 7 |
| information & Entertainment | | 5 | 102% | 2,73 | 2,79 | 0 | 0 | 0 | 0 |
| safety | | 8 | 103% | 10,47 | 10,77 | 16 | 16 | 16 | 8 |
| styling | | 10 | 104% | 7,08 | 7,35 | 20 | 20 | 20 | 0 |
| environment | | 3 | 103% | 3,08 | 3,16 | 0 | 0 | 0 | 3 |
| roominess | | 10 | 103% | 9,04 | 9,29 | 0 | 0 | 0 | 0 |
| vehicle dynamics | | 7 | 102% | 7,02 | 7,14 | 0 | 0 | 0 | 14 |

| Modul | Modul | Str. | Modulbenennung | | | Modulaktivitäten | corrosion | corrosion: metal | corrosion: plastic | drivetrain |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | proz. Verbesserung | | | | 102% | 101% | 111% | 100% |
| | | | Produktqualifizierung | | Aufwertefaktor | 3068 | 72 | 72 | 72 | 72 |
| | | | Prio | | | | 1 | 1 | 1 | 1 |
| | | | Summenpunkte Original | | | | 6264 | 5904 | 864 | 1584 |
| | | | Summenpunkte Variante | | | | 6396 | 5971 | 958 | 1584 |
| 21 | 1 | 1 | Aufbau | | 1,1 | 4589 | 158 | 158 | 158 | 0 |
| 2100 | 100 | 2 | Rohbau (Biw) | | 1,1 | 3447 | 79 | 79 | 0 | 0 |
| 2101 | 101 | 2 | Karosse (painted body) | | 1,1 | 2807 | 79 | 79 | 0 | 0 |
| 210102 | 10102 | 3 | Vorderwagen (front end) | | 1 | 1526 | 72 | 72 | 0 | 0 |
| 210104 | 10104 | 3 | Unterbau mitte / Boden mitte (chassis middle / floor center) | | 1 | 1192 | 72 | 72 | 0 | 0 |

**Abbildung 39: Variantenvergleich**

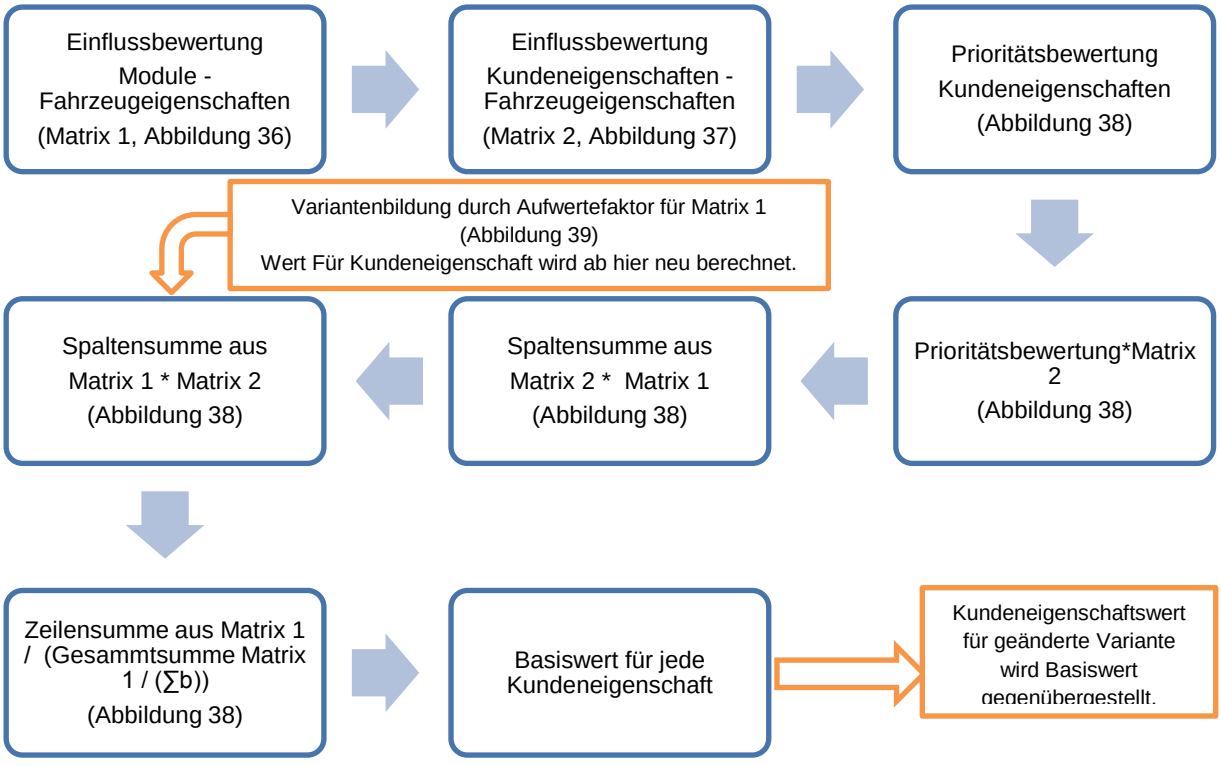


**Abbildung 40: Flussdiagramm bestehendes Konzept**

## 6.2 Neue Methodik

Mit der zu konzipierenden Methodik soll es möglich sein, die Kundenanforderungen transparent mit den Elementen des Fahrzeuges zu verknüpfen. Mit der bisherigen Methodik wurden die Kundeneigenschaften in zehn Bereiche gegliedert, wobei Erweiterungen aufgrund der komplexen Zuweisung der Module einen großen Aufwand bedeuteten. Projektspezifische Anpassungen gestalteten sich daher schwierig, und die Transparenz des Zustandekommens der Werte war kaum gegeben, was bei Variantenvergleiche Gefahren mit sich brachte

.

Das Gesamtfahrzeug besitzt eine hierarchische Struktur, bestehend aus Systemen, Modulen und Bauteilen. Dieser Struktur sollen die Kundenanforderungen bzw. Eigenschaften zugeordnet werden.

Da der Einfluss verschiedener Fahrzeugelemente nicht auf jede Eigenschaft gleich groß ist, bzw. teilweise gar nicht vorhanden ist, erfordert dies eine Berücksichtigung im Konzept. Es wird daher ein Relevanzfaktor eingeführt. Der Relevanzfaktor soll definieren wie groß der Einfluss des jeweiligen Fahrzeugelementes auf die gerade betrachtete Eigenschaft ist.

Nach dieser Definition erfolgt eine Bewertung, mit welcher Güte das Element die Anforderungen, in Bezug auf die gerade betrachtete Eigenschaft, erfüllt. Dieser Faktor wird im folgenden Erfüllungsgrad bezeichnet.

Mit diesem Bewertungsvorgang können nun Variantenvergleiche, zum Beispiel bei Verwendung einer anderen Verbindungtechnologie oder anderen Werkstoffen, durchgeführt werden. Durch die Kombination der beiden Faktoren können nun verschiedene Auswertungen vorgenommen werden.

Somit kann beispielsweise herausgefiltert werden, welche Elemente von großer Bedeutung für eine Eigenschaft sind. Eine Erhöhung des Erfüllungsgrads dieser Elemente ist daher am effektivsten, um die Erfüllung einer Eigenschaft zu verbessern.

Das durchschnittliche Produkt aus Relevanz und Erfüllungsgrad für jede Eigenschaft ist hingegen gut geeignet um verschiedene Varianten, und die Auswirkungen derer Unterschiede zu vergleichen.

Weiters sollen auch die wechselseitigen Beziehungen der Eigenschaften erfasst werden können. Die Speicherung erfolgt in einem Datenbanksystem. So können bei sämtlichen Schritten zur Nachvollviehbarkeit Informationsunterlagen, Dokumentationen und Literatur angehängt werden, was im Sinne des Konfigurationsmanagements zum Aufbau der Transparenz beiträgt.

## 6.3 Entity Relationship Model

Ein Fahrzeug besteht aus mehreren Systemen. Die Systeme sollen den Fahrzeugen zuordenbar sein, wobei ein System in mehreren Fahrzeugen zu Verwendung kommen kann. Dies ist über eine n:m-Beziehung möglich. Die Systeme bestehen aus Modulen, welche wiederum aus Bauteilen bestehen. Da Module und Bauteile mehrfach in verschiedenen übergeordneten Elementen verbaut werden können tritt hier ebenfalls eine n:m-Beziehung auf.
Die Fahrzeuge haben viele Eigenschaften, wobei diese über einen Auswahlprozess zuordenbar sein sollen. Die Eigenschaften stehen in Wechselwirkungen zueinander (Abbildung 41). Während der Definition dieser Beziehungen soll die Möglichkeit bestehen, Literatur und sonstige Unterlagen zu den Fahrzeugeigenschaften beizufügen.

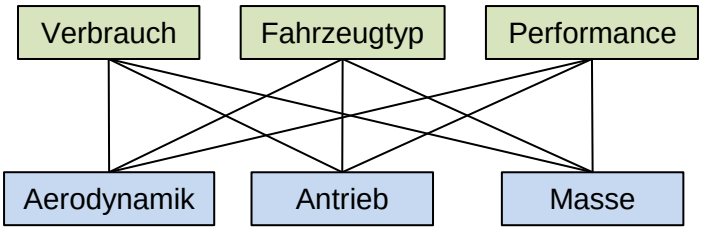


**Abbildung 41: Bsp. Fahrzeugeigenschaften n:m**

Die Eigenschaften werden den Systemen/Modulen/Bauteilen über die Entität Connection zugewiesen. Grundsätzlich besteht zwischen den Elementen und Eigenschaften eine m:n-Beziehung; jedem Element sollen alle Eigenschaften zuweisbar sein und allen Eigenschaften sollen mit jedem Element verknüpfbar sein.

Gehören zur n:m-Beziehung aber weitere Attribute, wie Relevanz und Bewertung, so wird bereits im ER-Modell ein eigener Entitätstyp gebildet, womit zwei getrennte 1:n-Beziehungen entstehen.

Die Verknüpfung von Elementen und Eigenschaften erfordert somit eine Definition der Relevanz, wie stark der Einfluss des Elementes auf diese Eigenschaft ist, und des Erfüllungsgrades, die Bewertung mit welcher Güte das Element seine Anforderungen, auf die jeweilige Eigenschaft bezogen, erfüllt. Weiters soll währenddessen die Möglichkeit bestehen, Dokumentationsunterlagen anzuhängen, welche beispielsweise die Entscheidung der Bewertung begründen.

In Abbildung 42 werden die Entitäten mit ihren Attributen und Beziehungen zueinander dargestellt. Die Entitäten werden über entsprechende Beziehungstypen, welche vorhin erläutert wurden, miteinander verknüpft.

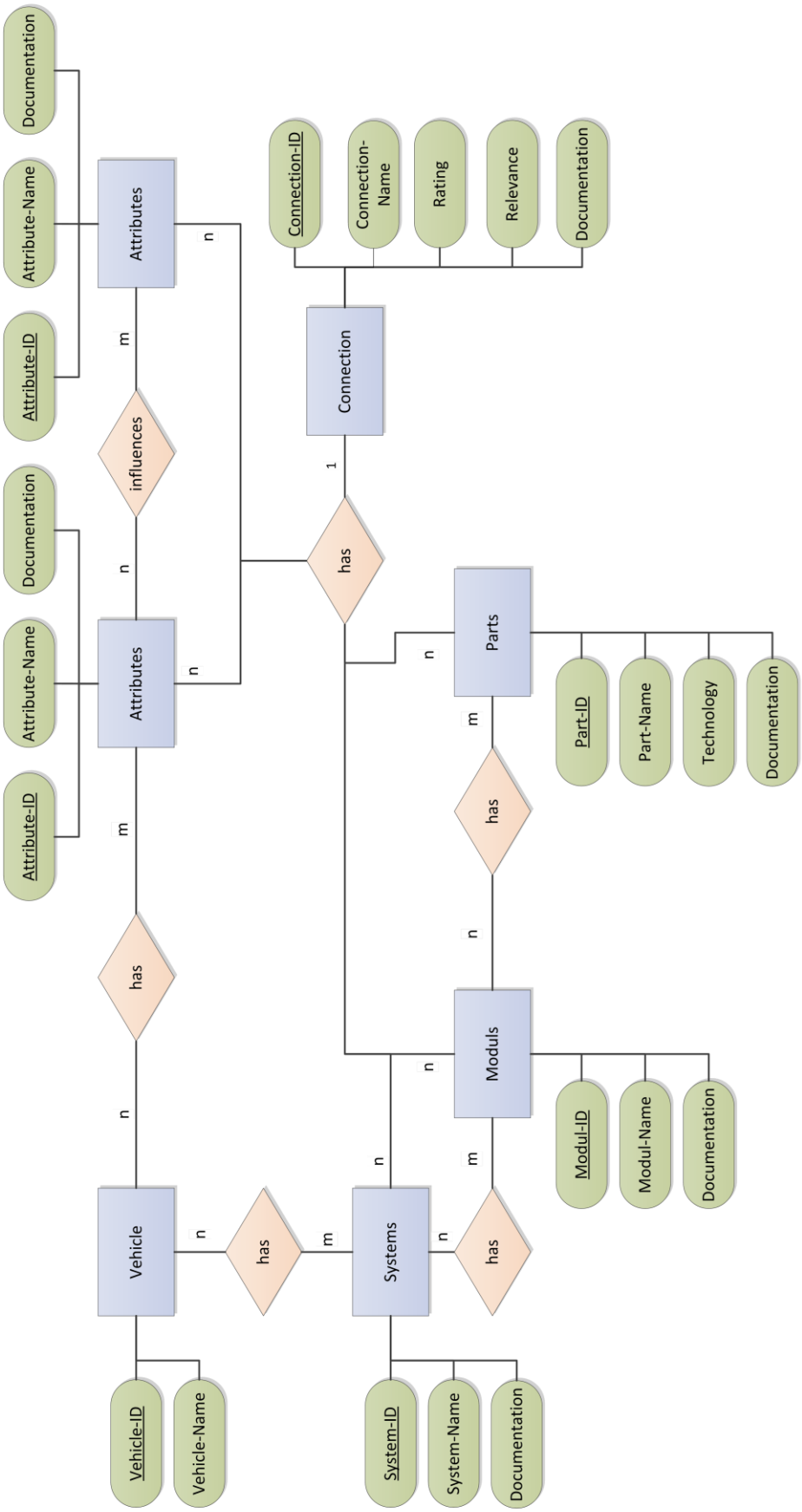


**Abbildung 42: ERM Konzept**

## 6.4 Relationales Modell

Das ERM wird durch eine Aufteilung der Datenbank in drei Bereiche umgesetzt.

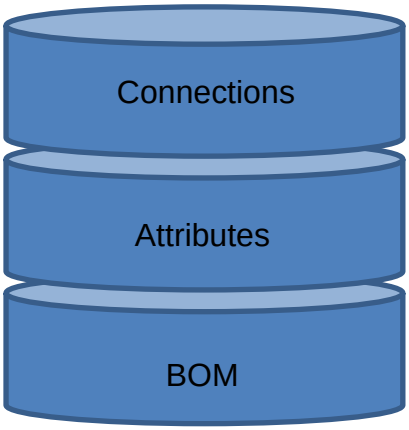


**Abbildung 43: Relationales Modell**

In der BOM erfolgt die Speicherung der Struktur des Fahrzeuges bestehend aus den Systemen, Modulen und Bauteilen. Im zweiten Bereich erfolgen die Definitionen der Eigenschaften und deren Beziehungen zueinander. Die Verknüpfung von Eigenschaften und Fahrzeugelementen sowie das dabei durchgeführte Rating fallen in den dritten Teilbereich.

### 6.4.1 BOM

Da n:m-Beziehungen in den meisten relationalen Datenbanken nicht direkt umgesetzt werden können, werden diese aufgelöst und über eine weitere Datenbanktabelle realisiert. Diese Tabelle enthält die beiden Primärschlüssel als Fremdschlüssel und bildet die n:m-Beziehung mittels zwei 1:n-Beziehungen nach.

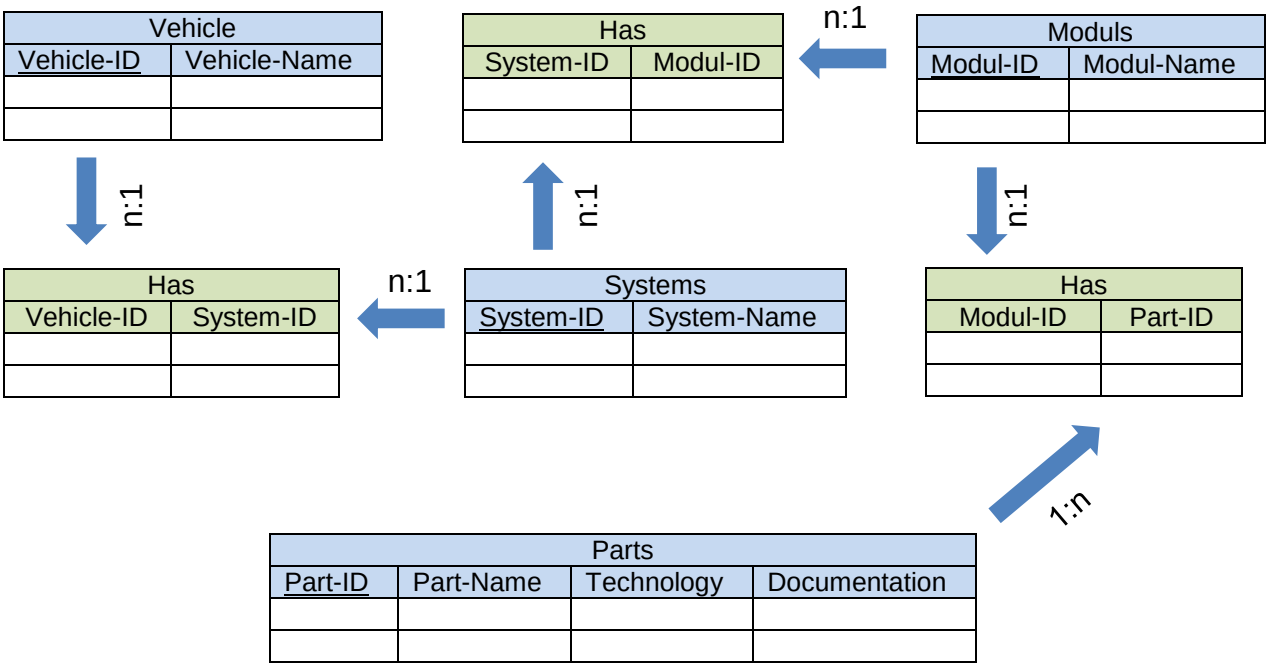


**Abbildung 44: BOM Relational**

### 6.4.2 Attributes

Neben der Definition der Eigenschaften lässt sich über eine weitere Tabelle festlegen, welche Eigenschaften aufeinander Einfluss haben.

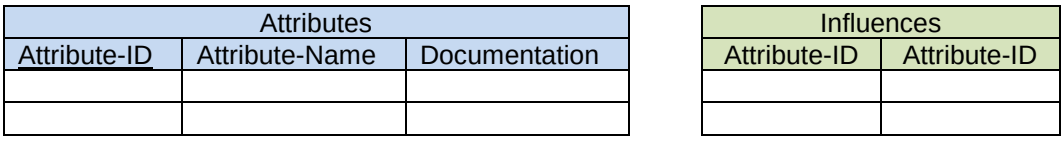


**Abbildung 45: Attributes Relational**

### 6.4.3 Connections

Hier erfolgt die Definition, welches Element mit welcher Eigenschaft im Zusammenhang steht. Dabei werden auch die Relevanz und die Bewertung definiert.

| Connections | | | | | |
|---|---|---|---|---|---|
| Connection-ID | System-ID<br>Modul-ID<br>Part-ID | Attribute-ID | Relevance | Rating | Documentation |
| | | | | | |
| | | | | | |

**Abbildung 46: Connections Relational**

## 6.5 Objektorientierte Definition der Klassen in VB.NET

Aufgrund der auftretenden Gleichartigkeit der Entitäten und der Wunsch nach einer einfachen Anwendbarkeit bietet sich ein objektorientierter Ansatz an.
In der objektorientierten Programmierung werden die Interna eines Objektes durch Datenkapselung und Trennung von Spezifikationen und Implementierung einer Methode verborgen. Durch das Verbergen der Objektinterna wird die Datensicherheit und Änderungsfreundlichkeit unterstützt. Die Datenelemente können nicht von anderen Objekten verändert werden und Änderungen in den internen Strukturen haben keine Auswirkungen auf die anderen Objekte. Es folgt dadurch eine Erhöhung der Flexibilität gegenüber Änderungen und Erweiterungen.
Ähnliche Objekte, die sich nur in ihren Attributwerten unterscheiden, müssen nur einmal beschrieben werden. Diese statische Beschreibung gleichartiger Objekte erfolgt in einer Klasse. Mittels Vererbung können die Klassen ihre Eigenschaften an Unterklassen übertragen.

Zur Umsetzung des Konzeptes wurden in VB.NET folgende Klassen definiert:

- Item: Zur Instanziierung von Systemen, Modulen, Bauteilen
- Attribute: Zur Instanziierung von Eigenschaften
- AttributeDependency: Zur Instanziierung von Eigenschaftsbeziehungen
- ConnectionItemAttribute: Zur Instanziierung von Fahrzeugelement-Eigenschaftsverknüpfungen und deren Bewertung
- Evaluation: Funktionen zur Auswertung

Für jede Klasse zur Instanziierung von Elementen gibt es eine eigene Collection Klasse, welche die Properties in eine generische System Collection Klasse vererbt bekommt. Mittels der List(Of T)-Klasse, wird eine generische Liste dieser Klassen erstellt. Die List(Of T)-Klasse stellt Methoden zum Durchsuchen, Sortieren und Bearbeiten dieser typisierten Liste von Objekten zur Verfügung und kann einfach in einem DataGridView visualisiert werden.

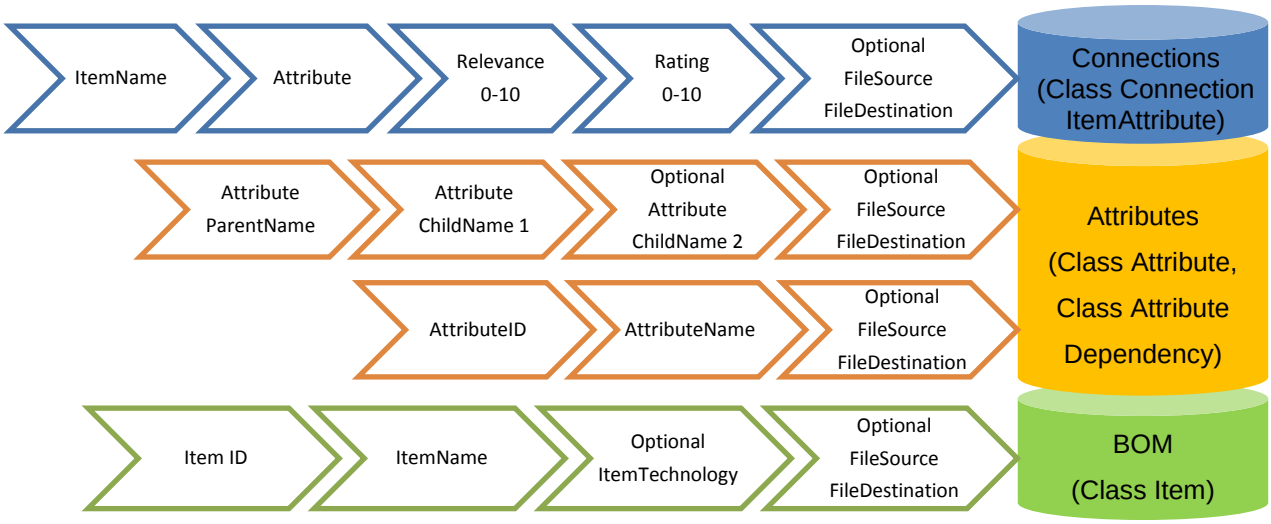


**Abbildung 47: VB.NET Konzept**

Aus den Item-Attribute Verknüpfungen können nun folgende Auswertungen durchgeführt werden:

- Such nach verknüpften Items oder Attributes.
- Such nach einer Item-Attribute Verknüpfung.
- Suche nach Rating- oder Relevance-Werten, die größer kleiner oder gleich einem Referenzwert sind.
- Suche nach Werten die größer kleiner oder gleich dem Produkt von Rating und Relevance sind.
- Berechnung der durchschnittlichen Eigenschaftserfüllung, die dem Variantenvergleich dient.

Diese Funktionen befinden sich in der Class Evaluation.

Bei jeder Instanziierung besteht die Möglichkeit der Angabe eines Quell- und Zielpfades, womit Dateien an einen entsprechenden Ort der Datenbank kopiert werden können.

### 6.5.1 BOM-Elemente

Elemente der Stückliste enthalten laut ERM folgende Attribute (Abbildung 42):

- ID: Primärschüssel des Elements
- Name: Benennung des Elements
- Technologie: Herstellungsverfahren (z.B. verwendete Verbindungstechnologie), verwendeter Werkstoff…
- Dokumentation: Informationen zum jeweiligen Element oder zur Technologie.

Die Systeme, Module und Bauteile aus denen das Fahrzeug besteht werden nun einheitlich mit dem Begriff „Item“ bezeichnet, da sie alle Attribute desselben Typs besitzen. In Anhang A.1 wird die Instanziierung neuer Fahrzeugelemente anhand von Beispielen beschrieben. Der Anwender hat eine ID und den Namen des Fahrzeugelements zu definieren. Optional kann auch eine verwendete Technologie definiert werden. Sollen Dokumentationsunterlagen hinterlegt werden, so muss die Dateiquelle und der neue Speicherort, wohin die Datei kopiert wird, angeführt werden.

Um eine Collection der Elemente aufstellen zu können, und sie dabei ihrem Elternknoten zuzuweisen, wurde eine weitere Klasse definiert (Anhang A.2). Die Klasse wird mittels Vererbung der Class Item in eine List(Of T) erstellt. Damit wird eine generische Liste mit allen Eigenschaften der Class Item gebildet.
Der User gibt den Namen der Collection und das Item, welchen den Elternknoten bildet an, siehe Beispiel in Anhang A.2. System1 besteht aus den Modulen 1 und 2.
Ein nachträgliches Bearbeiten einer Collection erfolgt mit den List(Of T)-Methoden (Anhang B). Das Entfernen eines bestimmten Bauteils oder aller Elemente ist im Beispiel von Anhang A.2 beschrieben.

### 6.5.2 Attributes

Die Eigenschaften enthalten laut ERM folgende Attribute (Abbildung 42):

- ID: Primärschüssel der Eigenschaft
- Name: Benennung der Eigenschaft
- Dokumentation: Informationen zur jeweiligen Eigenschaft und zu deren Einfluss auf andere Eigenschaften bzw. Abhängigkeiten von anderen Eigenschaften.

Die Class Attribute dient der Instanziierung der Eigenschaften. Durch Definition der ID, des Attributnamens und einer optionalen Dokumentationsbeilage werden die Eigenschafften definiert (Bsp. Anhang A.3).

Die Collection der Attribute erfolgt gleich wie bei den Fahrzeugelementen, durch Vererbung der Class Attribute in eine generische System Collection, siehe Bsp. Anhang A.4. Der Anwender definiert eine neue AttributeCollection und fügt dieser seine Instanziierten Attribute zu.

#### 6.5.2.1 Attribute-Dependencies

Wie bereits vorhin erwähnt sollen auch die gegenseitigen Abhängigkeiten der Eigenschaften erfasst werden.
Mit der Class AttributeDependency können diese definiert werden. Der Anwender Instanziiert eine neue Attributabhängigkeit, wobei er die Elterneigenschaft und dessen in Beziehung stehende Eigenschaften angibt (Beispiel Anhang A5).

Die Collection der Abhängigkeiten erfolgt analog zu den vorigen Collection-Klassen, siehe Beispiel Anhang A.6.

### 6.5.3 Connections

Die Verknüpfung von Fahrzeugelementen mit Eigenschaften enthält laut ERM folgende Attribute (Abbildung 42):

- ID: Primärschüssel der Connection
- Name: Benennung der Connection
- Relevanz: Einfluss des Fahrzeugelementes auf die Eigenschaft (Bewertung mittels Punktesystem 0-10 Punkte)
- Rating: Erfüllungsgrad der eigenschaftsabhängigen Elementanforderungen (Bewertung mittels Punktesystem 0-10 Punkte)
- Dokumentation: Informationen zum Zusammenhang der Elementes und der Eigenschaft, sowie der Bewertung von Rating und Relevanz.

In Anhang A.7 wird eine derartige Verknüpfungserstellung beschrieben. Der Anwender definiert die Verknüpfung indem er das Item und ein davon abhängiges Attribute angibt. Zur Bewertung können der Wert für Relevanz und Rating einfach aus einem Dropdown Menü ausgewählt werden.
Die Collection der Connections erfolgt analog zu den vorigen Collection-Klassen, siehe Beispiel Anhang A.8.
Treten nun beispielsweise Änderungen in der Verbindungstechnologie auf, welche Einfluss auf die Bewertung haben, so wird eine neue Collection mit den geänderten Bewertungen angelegt.

### 6.5.4 Evaluation

In dieser Klasse befinden sich die Funktionen zur Auswertung. Der Einfluss von Änderungen bestimmter Elemente auf Eigenschaften soll erkennbar werden. Daher sind Funktionen zur Filterung der Bewertungsfaktoren gefragt.
Die Erstellung weiterer spezifischer Funktionen gestaltet sich aufgrund der Verwendung einer generischen System Collection einfach.

Die definierten Funktionen benötigen jeweils die Angabe der zu durchsuchenden ConnectionCollection. Daher ist ein Variantenvergleich verschiedener Fahrzeuge schnell durch Änderung des Collectionnamens durchführbar.

#### 6.5.4.1 Max-/Min-/Is-Value

Zur Auswertung der vorgenommenen Bewertung sind die Unterschreitung sowie Überschreitung bestimmter Werte von Relevanz, Ratings (Erfüllungsgrad) und des Produktes dieser beiden Werte von Interesse. Weiters kann auch die Suche nach einem bestimmten Wert von einem dieser drei Faktoren erwünscht sein.

Das Herausfiltern von Element-Eigenschaftsverknüpfungen erfolgt durch Definition folgender Parameter:

- welche ConnectionCollection soll durchsucht werden
- nach welchem Valuetype soll gesucht werden (Rating, Relevance, Rating*Relevance)
- soll nach Wertebereichen größer/kleiner/ist gleich, bezogen auf den Referenzwert, gesucht werden?
- Referenzwert

Der Valuetype und Wertebereich kann durch Verwendung der Enum-Anweisung bei der Funktion einfach ausgewählt werden.

In Anhang A.9.1 befindet sein ein Beispiel einer Suche, deren Ergebnis durch Verwendung der List(Of T)-Klasse schnell und übersichtlich in einem DataGridView angezeigt werden kann.
Der User greift auf die Funktion FindValue der Evaluation-Klasse zu und durchsucht damit die ConnectionConnection1 nach Relevanzwerten größer 4.
Um Änderungen zwischen verschiedenen Varianten festzustellen, ist bei der Wertabfrage nur der Name der ConnectionCollection zu ändern.

#### 6.5.4.2 Find Item and/or Attribute

Mit der Funktion in Anhang A.9.2 können Fahrzeugelemente, Eigenschaften oder Verknüpfungen von Elementen und Eigenschaften herausgefiltert werden.
Der Anwender hat die zu durchsuchende ConnectionCollection sowie Fahrzeugelement und/oder Eigenschaft anzugeben (Beispiel Anhang A.9.2).

#### 6.5.4.3 Average AttributeFullfillment

Eine weitere Vergleichsmöglichkeit zwischen verschiedenen Varianten ergibt sich aus dem durchschnittlichen Gesamterfüllungswert einer Eigenschaft. Die Relevanz sämtlicher Elemente multipliziert mit dem Erfüllungsgrad, geteilt durch die Anzahl der Elemente, ergibt den durchschnittlichen Erfüllungsgrad einer Eigenschaft. Der Unterschied verschiedener bewerteter Varianten, welche je eine ConnectionCollection bilden, lässt sich hierbei wieder leicht, nur durch Änderung des ConnectionCollection-Namen, ermitteln.
In Anhang A.9.3 wird die Funktion anhand eines Beispiels beschrieben. Die zu durchsuchende ConnectionCollection und der Eigenschaftsname, deren durchschnittlicher Gesamterfüllungsgrad gesucht wird, sind anzugeben.

# 7 Zusammenfassung

Mit diesem objektorientierten Konzept ist eine Darstellung des Fahrzeugaufbaus, die Verwaltung der Eigenschaften und deren Beziehung zueinander sowie der Variantenvergleich von Fahrzeugen, bzw. das Erkennen von Auswirkungen auf die Fahrzeugeigenschaften aufgrund von Änderungen, möglich.

Der spätere Nutzen des Konzeptes liegt in der Erkennung von Haupteinflussträger und derer Wechselwirkungen auf Eigenschafen, in der Wissensspeicherung und der daraus resultierenden gezielten Erreichung geforderter Fahrzeugeigenschaften.

Die Möglichkeit der Dokumentation in sämtlichen Schritten soll im Sinne des Konfigurationsmanagements die Transparenz gewährleisten und auch der Wissensspeicherung dienen.

Im Gegensatz zur bisherigen Methodik wird hierbei auch eine Vereinfachung für den Anwender durch den objektorientierten Ansatz der Anwendung geschaffen. Der aus nur zwei Einflussfaktoren bestehende Bewertungsvorgang erhöht die Transparenz des Zustandekommens einer Eigenschaftserfüllung, wodurch die Auswirkungen von Änderungen leichter nachvollziehbar werden.

# Anhang A: VB.NET Code

## A.1 Class Item

Vorgang der Item-Instanziierung:


**Abbildung 48: Item-Instanziierung**

```
Public Class Item

    Private m_ItemID As Integer
    Private m_ItemName As String
    Private m_ItemTechnology As String

    Public Property ItemID As Integer
        Get
            Return m_ItemID
        End Get
        Set(ByVal value As Integer)
            m_ItemID = value
        End Set
    End Property
    Public Property ItemName As String
        Get
            Return m_ItemName
        End Get
        Set(ByVal value As String)
            m_ItemName = value
        End Set
    End Property
    Public Property ItemTechnology As String
        Get
            Return m_ItemTechnology
        End Get
        Set(ByVal value As String)
            m_ItemTechnology = value
        End Set
    End Property

    Public Sub New(ByVal ItemID As Integer, _
                   ByVal ItemName As String, _
                   Optional ByVal ItemTechnology As String = "", _
                   Optional ByVal FileSource As String = "", _
```

```
                    Optional ByVal FileDestination As String = "")

        m_ItemID = ItemID
        m_ItemName = ItemName
        m_ItemTechnology = ItemTechnology

        If FileSource = "" Or FileDestination = "" Then
        Else
            System.IO.File.Copy(FileSource, FileDestination, True)
        End If

    End Sub

End Class
```

Beispiele zur Instanziierung von Elementen:

```
        Dim system1 As New Item(014, "SystemName1")
        Dim modul1 As New Item(049, "ModulName1")
        Dim bauteil1 As New Item(005, "BauteilName1", "schweißen", _
                                 C:\Documents\..., A:\Documents\...)
        Dim bauteil2 As New Item(006, "BauteilName2")
```

### A.2 Class ItemCollection

Damit wird das Erstellen einer Auflistung von Items ermöglicht.

```
Public Class ItemCollection

    Inherits List(Of Item)

    Public Sub New(ByVal Item As Item)
    End Sub

End Class
```

Beispiele zu Erstellung von Item-Collections:

```
        Dim system1_collection As New ItemCollection(system1)
        system1_collection.Add(modul1)
        system1_collection.Add(modul2)

        Dim modul1_collection As New ItemCollection(modul1)
        modul1_collection.Add(bauteil1)
        modul1_collection.Add(bauteil2)

        Dim modul2_collection As New ItemCollection(modul2)
```

```
modul1_collection.Add(bauteil3)
modul1_collection.Add(bauteil4)
```

Entfernt Bauteil1:

```
modul1_collection.Remove(bauteil1)
```

Entfernt alle Elemente aus der Collection:

```
modul1_collection.Clear()
```

### A.3 Class Attribute

Vorgang der Attribute-Instanziierung:


**Abbildung 49: Attribute-Instanziierung**

```
Public Class Attribute

    Private m_AttributeID As Integer
    Private m_AttributeName As String

    Public Property AttributeID As Integer
        Get
            Return m_AttributeID
        End Get
        Set(ByVal value As Integer)
            m_AttributeID = value
        End Set
    End Property
    Public Property AttributeName As String
        Get
            Return m_AttributeName
        End Get
        Set(ByVal value As String)
            m_AttributeName = value
        End Set
    End Property

    Public Sub New(ByVal AttributeID As Integer, _
                   ByVal AttributeName As String, _
```

```
                    Optional ByVal FileSource As String = "", _
                    Optional ByVal FileDestination As String = "")

        m_AttributeID = AttributeID
        m_AttributeName = AttributeName

        If FileSource = "" Or FileDestination = "" Then
        Else
            System.IO.File.Copy(FileSource, FileDestination, True)
        End If

    End Sub

End Class
```

Beispiel der Attribute-Instanziierung:

```
        Dim attribute1 As New Attribute(001, "AttributName1")
        Dim attribute2 As New Attribute(002, "AttributName2")
```

### A.4 Class AttributeCollection

Damit wird das Erstellen einer Auflistung von Attributen ermöglicht.

```
Public Class AttributeCollection

    Inherits List(Of Attribute)

    Public Sub New()
    End Sub

End Class
```

Beispiel des Erstellens einer Attribute-Collection:

```
        Dim AllAttributes As New AttributeCollection()
        AllAttributes.Add(attribute1)
        AllAttributes.Add(attribute2)
```

## A.5 Class AttributeDependency

Vorgang einer Abhängigkeitsinstanziierung:

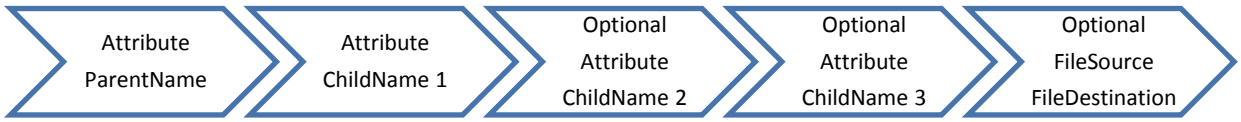


**Abbildung 50: Dependency-Instanziierung**

```
Public Class AttributeDependency

    Private m_DependencyID As Integer

    Public Property DependencyID As Integer
        Get
            Return m_DependencyID
        End Get
        Set(ByVal value As Integer)
            m_DependencyID = value
        End Set
    End Property
    Public Property ParentName As String
    Public Property Child1Name As String
    Public Property Child2Name As String
    Public Property Child3Name As String


    Public Sub New(ByVal DependencyID As Integer, _
                   ByVal ParentAttribute As Attribute, _
                   ByVal ChildAttribute1 As Attribute, _
                   Optional ByVal ChildAttribute2 As Attribute = Nothing,_
                   Optional ByVal ChildAttribute3 As Attribute = Nothing,_
                   Optional ByVal FileSource As String = "", _
                   Optional ByVal FileDestination As String = "")

        m_DependencyID = DependencyID
        ParentName = ParentAttribute.AttributeName
        Child1Name = ChildAttribute1.AttributeName
        Child2Name = ChildAttribute2.AttributeName
        Child3Name = ChildAttribute3.AttributeName

        If FileSource = "" Or FileDestination = "" Then
        Else
            System.IO.File.Copy(FileSource, FileDestination, True)
        End If
    End Sub
End Class
```

Beispiel einer Abhängigkeits-Instanziierung:

```
Dim attribute1_dependencies As New AttributeDependency(4578,
    attribute1, attribute2, attribute4, attribute3)
```

### A.6 Class AttributeDependencyCollection

Damit wird das Erstellen einer Auflistung von Attribute-Abhängigkeiten ermöglicht.

```
Public Class AttributeDependencyCollection

    Inherits List(Of AttributeDependency)

    Public Sub New()
    End Sub

End Class
```

Beispiel des Erstellens einer Abhängigkeits-Collection:

```
Dim DependencyCollection As New AttributeDependencyCollection()
DependencyCollection.Add(attribute1_dependencies)
```

### A.7 Class Connection

Vorgang einer Item-Attribut-Connection-Instanziierung:

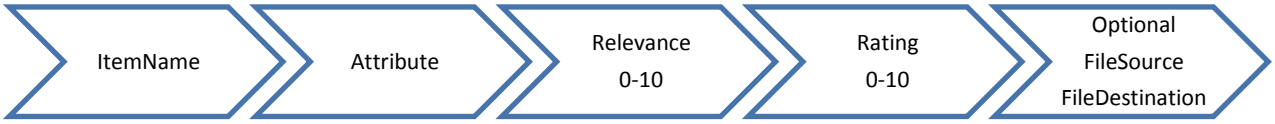


**Abbildung 51: Connection-Instanziierung**

```
Public Class ConnectionItemAttributes
    Private m_ConnectionID As Integer
    Public Property ConnectionID As Integer
        Get
            Return m_ConnectionID
        End Get
```

```
        Set(ByVal value As Integer)
            m_ConnectionID = value
        End Set
    End Property
    Public Property ItemName As String
    Public Property ConnectedAttribute As String
    Public Property sRelevance As Integer
    Public Property sRating As Integer

    Enum RelevanceEnum As Integer
        Zero = 0
        One
        Two
        Three
        Four
        Five
        Six
        Seven
        Eight
        Nine
        Ten
    End Enum
    Enum RatingEnum As Integer
        Zero = 0
        One
        Two
        Three
        Four
        Five
        Six
        Seven
        Eight
        Nine
        Ten
    End Enum

    Public Sub New(ByVal ConnectionID As Integer, _
                   ByVal Item As Item, _
                   ByVal Attribute As Attribute, _
                   ByVal Relevance As RelevanceEnum, _
                   ByVal Rating As RatingEnum, _
                   Optional ByVal FileSource As String = "", _
                   Optional ByVal FileDestination As String = "")

        m_ConnectionID = ConnectionID
        ItemName = Item.ItemName
        ConnectedAttribute = Attribute.AttributeName
        sRelevance = Relevance
        sRating = Rating

        If FileSource = "" Or FileDestination = "" Then
        Else
            System.IO.File.Copy(FileSource, FileDestination, True)
```

```
        End If
    End Sub
End Class
```

Beispiel einer Verknüpfung eines Items mit einer Eigenschaft:

```
Dim ItemAttConnection1 As New ConnectionItemAttributes(45454, bauteil1,
attribute2, ConnectionItemAttributes.RelevanceEnum.Seven,
ConnectionItemAttributes.RatingEnum.Six)
```

### A.8 Class ConnectionCollection

Damit wird das Erstellen einer Auflistung von Item-Attribute-Verknüpfungen ermöglicht.

```
Public Class ConnectionCollection

    Inherits List(Of ConnectionItemAttributes)

    Public Sub New()
    End Sub

End Class
```

Beispiel der Erstellung einer Sammlung von Connections:

```
        Dim ConnectionCollection As New ConnectionCollection()
        ConnectionCollection.Add(ItemAttConnection1)
```

### A.9 Class Evaluation

#### A.9.1 Function FindValue

Suchvorgang:


**Abbildung 52: FindValue**

```
Enum Valuetype
    Rating
    Relevance
    RelevanceMultiplyRating
End Enum

Enum ValueRange
    SmallerThan
    BiggerThan
    IsExactly
End Enum

Public Shared Function FindValue( _
                ByVal ConnectionCollection As ConnectionCollection, _
                ByVal Valuetype As Valuetype, _
                ByVal ValueRange As ValueRange, _
                ByVal Value As Integer) As List(Of
            ConnectionItemAttributes)

    Select Case ValueRange
        Case Evaluation.ValueRange.SmallerThan
         Select Case Valuetype
            Case Evaluation.Valuetype.Rating
                FindValue = _
                    (From ConnectionItemAttributes In ConnectionCollection _
                     Where ConnectionItemAttributes.sRating < Value).ToList
            Case Evaluation.Valuetype.Relevance
                FindValue = _
                    (From ConnectionItemAttributes In ConnectionCollection _
                    Where ConnectionItemAttributes.sRelevance < Value).ToList
            Case Evaluation.Valuetype.RelevanceMultiplyRating
                FindValue = _
                    (From ConnectionItemAttributes In ConnectionCollection _
                     Where (ConnectionItemAttributes.sRelevance *
                     ConnectionItemAttributes.sRating) < Value).ToList
            Case Else
                FindValue = Nothing
         End Select

        Case Evaluation.ValueRange.BiggerThan
            Select Case Valuetype
            Case Evaluation.Valuetype.Rating
                FindValue = _
                    (From ConnectionItemAttributes In ConnectionCollection _
                    Where ConnectionItemAttributes.sRating > Value).ToList
            Case Evaluation.Valuetype.Relevance
                FindValue = _
                    (From ConnectionItemAttributes In ConnectionCollection _
                    Where ConnectionItemAttributes.sRelevance > Value).ToList
            Case Evaluation.Valuetype.RelevanceMultiplyRating
                FindValue = _
                    (From ConnectionItemAttributes In ConnectionCollection _
                     Where (ConnectionItemAttributes.sRelevance *
                     ConnectionItemAttributes.sRating) > Value).ToList
            Case Else
                FindValue = Nothing
         End Select

        Case Evaluation.ValueRange.IsExactly
```

```
            Select Case Valuetype
                Case Evaluation.Valuetype.Rating
                    FindValue = _
                        (From ConnectionItemAttributes In ConnectionCollection _
                        Where ConnectionItemAttributes.sRating = Value).ToList
                Case Evaluation.Valuetype.Relevance
                    FindValue = _
                        (From ConnectionItemAttributes In ConnectionCollection _
                        Where ConnectionItemAttributes.sRelevance = Value).ToList
                Case Evaluation.Valuetype.RelevanceMultiplyRating
                    FindValue = _
                        (From ConnectionItemAttributes In ConnectionCollection _
                         Where (ConnectionItemAttributes.sRelevance *
                         ConnectionItemAttributes.sRating) = Value).ToList
                Case Else
                    FindValue = Nothing
               End Select

            Case Else
                FindValue = Nothing
        End Select
    End Function
```

Beispiel der Suche nach Elementen mit einem Relevanzwert>4, wobei das Ergebnis in einem DataGridView angezeigt wird:

```
DataGridView1.DataSource = Evaluation.FindValue(ConnectionCollection1,
Evaluation.Valuetype.Relevance, Evaluation.ValueRange.BiggerThan, 4)
```

| ConnectionCollection1 | | | |
|---|---|---|---|
| ItemName | ConnectedAttribute | Relevance | Rating |
| Bauteil1 | Eigenschaft 1 | 1 | 4 |
| Bauteil 2 | Eigenschaft 2 | 7 | 8 |

| DataGridView1 | | | |
|---|---|---|---|
| ItemName | ConnectedAttribute | Relevance | Rating |
| Bauteil 2 | Eigenschaft 2 | 7 | 8 |

### A.9.2 Function FindItemAndOrAttribute

Suchvorgang:

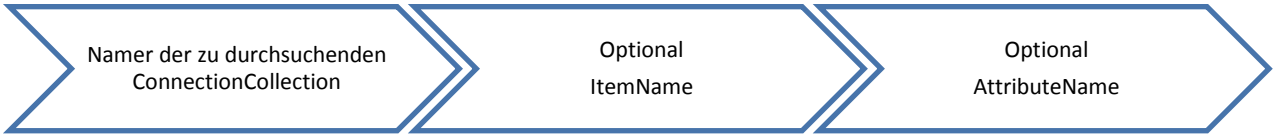


**Abbildung 53: FindObject**

```
Public Shared Function FindItemAndOrAttribute(
                      ByVal ConnectionCollection As ConnectionCollection, _
                      Optional ByVal ItemName As String = "", _
                      Optional ByVal AttributeName As String = "") _
                      As List(Of ConnectionItemAttributes)

        If AttributeName = "" Then
            FindItemAndOrAttribute = (From ConnectionItemAttributes _
                                      In ConnectionCollection _
                                      Where ConnectionItemAttributes.ItemName = _
                                      ItemName).ToList

        ElseIf ItemName = "" Then
            FindItemAndOrAttribute = (From ConnectionItemAttributes _
                                      In ConnectionCollection _
                                   Where ConnectionItemAttributes.ConnectedAttribute = _
                                      AttributeName).ToList

        ElseIf AttributeName = "" And ItemName = "" Then
            FindItemAndOrAttribute = Nothing

        Else
            FindItemAndOrAttribute = (From ConnectionItemAttributes _
                                      In ConnectionCollection _
                                   Where ConnectionItemAttributes.ConnectedAttribute = _
                                      AttributeName And _
                                      ConnectionItemAttributes.ItemName = _
                                      ItemName).ToList
        End If

    End Function
```

Beispiel der Suche nach einer Verknüpfung zwischen Bauteil1 und Attribute2 in der ConnectionCollection1, wobei das Ergebnis in einem DataGridView angezeigt wird:

```
DataGridView2.DataSource =
Evaluation.FindItemAndOrAttribute(ConnectionCollection1, "bauteil1",
"AttributeName2")
```

### A.9.3 Function AttributeFullfillment

Suchvorgang:


**Abbildung 54: AttributeFullfillment**

```
Public Shared Function AttributeFullfillment( _
                ByVal ConnectionCollection As ConnectionCollection, _
                ByVal AttributeName As String) As Integer

   Dim queryResults = From ConnectionItemAttributes _
       In ConnectionCollection
       Where ConnectionItemAttributes.ConnectedAttribute = AttributeName
       Select ConnectionItemAttributes.sRating, _
       ConnectionItemAttributes.sRelevance

    Dim ValueList As New List(Of Integer)

    For Each result In queryResults
        ValueList.Add(result.sRating * result.sRelevance)
    Next

    Dim Value As Integer = ValueList.Average
    AttributeFullfillment = Value

End Function
```

Beispiel der Anzeige des durchschnittlichen Erfüllungsgrades der Eigenschaft Attribute2, aus der ConnectionCollection1, in einer Textbox:

```
TextBox1.Text = Evaluation.AttributeFullfillment(ConnectionCollection1,
"AttributeName2")
```

## Anhang B: List(Of T)-Methoden

| Name | Beschreibung |
|---|---|
| Add | Fügt am Ende der List(Of T) ein Objekt hinzu. |
| Clear | Entfernt alle Elemente aus der List(Of T). |
| Contains | Bestimmt, ob sich ein Element in List(Of T) befindet. |
| ConvertAll(Of TOutput) | Konvertiert die Elemente in der aktuellen List(Of T) in einen anderen Typ und gibt eine Liste der konvertierten Elemente zurück. |
| CopyTo(T()) | Kopiert die gesamte List(Of T) in ein kompatibles eindimensionales Array, wobei am Anfang des Zielarrays begonnen wird. |
| Exists | Bestimmt, ob die List(Of T) Elemente enthält, die mit den vom angegebenen Prädikat definierten Bedingungen übereinstimmen. |
| Find | Sucht nach einem Element, das die durch das angegebene Prädikat definierten Bedingungen erfüllt, und gibt das erste Vorkommen im gesamten List(Of T) zurück. |
| FindAll | Ruft alle Elemente ab, die die vom angegebenen Prädikat definierten Bedingungen erfüllen. |
| FindLast | Sucht nach einem Element, das die durch das angegebene Prädikat definierten Bedingungen erfüllt, und gibt das letzte Vorkommen im gesamten List(Of T) zurück. |
| Remove | Entfernt das erste Vorkommen eines angegebenen Objekts aus List(Of T). |
| RemoveAll | Entfernt alle Elemente, die die vom angegebenen Prädikat definierten Bedingungen erfüllen. |
| ToArray | Kopiert die Elemente der List(Of T) in ein neues Array. |

Erweiterungsmethoden:

| Name | Beschreibung |
|---|---|
| All(Of T) | Bestimmt, ob alle Elemente einer Sequenz eine Bedingung erfüllen. (Durch Enumerable definiert.) |
| Any(Of T)(Func(Of T, Boolean)) | Überladen. Bestimmt, ob ein Element einer Sequenz eine Bedingung erfüllt. (Durch Enumerable definiert.) |
| Average(Of T)(Func(Of T, Int32)) | Überladen. Berechnet den Durchschnitt einer Sequenz von Int32-Werten, die durch den Aufruf einer Transformationsfunktion für jedes Element der Eingabesequenz ermittelt werden. (Durch Enumerable definiert.) |
| Count(Of T) | Überladen. Gibt die Anzahl der Elemente in einer Sequenz zurück. (Durch Enumerable definiert.) |
| GroupBy(Of T, TKey)(Func(Of T, TKey)) | Überladen. Gruppiert die Elemente einer Sequenz entsprechend einer angegebenen Schlüsselauswahlfunktion. (Durch Enumerable definiert.) |
| OrderBy(Of T, TKey)(Func(Of T, TKey)) | Überladen. Ruft für jedes Element einer generischen Sequenz eine Transformationsfunktion auf und gibt den niedrigsten Ergebniswert zurück. (Durch Enumerable definiert.) |
| Reverse(Of T) | Überladen. Sortiert die Elemente einer Sequenz in aufsteigender Reihenfolge nach einem Schlüssel. (Durch Enumerable definiert.) |
| Where(Of T)(Func(Of T, Int32, Boolean)) | Überladen. Filtert eine Sequenz von Werten nach einem Prädikat. (Durch Enumerable definiert.) |

# Abbildungsverzeichnis

## Literaturverzeichnis

Eckstein L., 2010: *Strukturentwurf von Kraftfahrzeugen*, 1. Neuauflage, fka – Forschungsgesellschaft Kraftfahrwesen mbH Aachen, Aachen.

Geisler F., 2011: *Datenbanken Grundlagen und Design*, 4. Auflage, mitp-Verlag, Heidelberg.

IEEE 610.12-1990: *Standard Institute of Electrical and Electronics Engineers*, in Pohl K., 2008: *Requirements Engineering, Grundlagen, Prinzipien, Techniken,* 2. korrigierte Auflage, dpunkt-Verlag, Heidelberg.

IEEE-828-1998: *Standard Institute of Electrical and Electronics Engineers-Standard for Software Configuration Management Plan*, in Popp G., 2013: *Konfigurationsmanagement mit Subversion, Maven und Redmine*, 4. aktualisierte und erweiterte Auflage, dpunkt-Verlag, Heidelberg.

Jochem R., Landgraf K., 2011: *Anforderungsmanagement in der Produktentwicklung, Komplexität reduzieren, Prozesse optimieren*, *Qualität sichern,* 1. Auflage, Symposion Publishing GmbH, Düsseldorf.

Landgraf K., 2009: *ISYPROM Forschungspapier 400-01-01, Anforderungsmanagement*.

Popp G., 2013: *Konfigurationsmanagement mit Subversion, Maven und Redmine*, 4. aktualisierte und erweiterte Auflage, dpunkt-Verlag, Heidelberg.

Rupp C., 2007: *Requirements-Engineering und –Management. Professionelle, iterative Anforderungsanalyse für die Praxis*, 4. Aktualisierte und erw. Auflage, Carl Hanser Verlag, München.

Versteegen G., Weischedel G., 2003: *Konfigurationsmanagement*, 1. Auflage, Springer-Verlag, Münschen.