Vermeidung von Lücken in API-Verträgen

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat.
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

eingereicht beim Fach-Promotionsausschuss Informatik von

Jan Christian Krause

aus Buxtehude

November 2013

Gutachter:
Prof. Dr.-Ing. Wolfgang Menzel
Prof. Dr.-Ing. habil. Matthias Riebisch

Tag der Disputation:
30.09.2014

Für Sandra

Die vorliegende Arbeit ist das Ergebnis von mehr als fünfjähriger Forschung - mit unzähligen Höhen und auch Tiefen. Während dieser Zeit habe ich große Unterstützung von vielen Menschen erfahren, denen ich an dieser Stelle dafür herzlich danken möchte.

Ganz besonders und herzlich danke ich Wolfgang Menzel für die wunderbare Betreuung dieser Arbeit. Wolfgangs Tür stand stets für mich offen. Seine kritischen Fragen und wertvollen Hinweise während unserer Gespräche haben mich immer wieder auf große und kleine Lücken in meinen Theorien, Argumentationsketten, Texten und Planungen aufmerksam gemacht.

Ebenso besonders und herzlich möchte ich Matthias Riebisch danken, der die Rolle des Zweitgutachters dieser Dissertation übernommen hat. Herr Riebisch hat mir im Rahmen seiner Vorlesung "Modellbasierte Software-Entwicklung" die Möglichkeit zur Evaluierung meines Verfahrens mit studentischen Probanden gegeben. Ich danke ihm außerdem für die zahlreichen kritischen Fragen und Hinweise zur Durchführung dieser Evaluierung und der resultierenden schriftlichen Ausarbeitung.

Die nebenberufliche Anfertigung dieser Dissertation wäre nicht möglich gewesen ohne die großartige Unterstützung meines Arbeitgebers - der AKRA GmbH aus Hamburg. Deshalb schulde ich meinen Chefs Fabian von Borcke und Thomas Ochmann für das vielfach entgegengebrachte Verständnis, für ihre Geduld, Unterstützung und Offenheit besonders großen und herzlichen Dank.

Außerdem möchte ich mich besonders bei meinen Kollegen Axel Schmolitzky und Ascan Heydorn bedanken. Mit beiden hatte ich jeweils mehrere Gespräche, die äußerst prägender Natur für meine Arbeit waren. Axel verdanke ich zahlreiche Hinweise auf das von Meyer geprägte Vertragsmodell sowie auf dessen Metaphern Stille und Rauschen. Ascan hat mich neben zahlreichen methodischen Anregungen auch immer wieder darauf hingewiesen, wie wichtig das Projektmanagement für den erfolgreichen Abschluss dieses Lebensabschnittes ist und dass jedes Vorhaben irgendwann enden muss. Euch beiden vielen, vielen Dank dafür.

Kaum ein Konzept wird von einer einzelnen Person in isolierter Weise erdacht. Während der vergangenen fünf Jahre habe ich immer wieder Kollegen kennengelernt, die sich für meine Forschung interessierten. Sie haben durch Fragen, ihre persönlichen Einschätzungen und organisatorische Flexibilität wesentlich zur Reifung meines Konzeptes und damit zu dieser Dissertation beigetragen. In diesem Zusammenhang richte ich meinen herzlichen Dank an Christian Engelhardt, Timm Schwemann, Florian Stumpf, Alexander Bollens,

Rias Sherzad, Oliver Hankel, Gabriel Jacobsohn, Dirk Meier-Eickhoff und Patrick McCrae.

Großen Dank schulde ich ferner Martin Christof Kindsmüller für die Hinweise auf die Beratungsangebote der Universität Hamburg im Bezug auf empirische Versuchsdurchführungen und -auswertungen sowie für die Diskussion meines Exposés zur empirischen Evaluierung. Ebenso danke ich Hans Pinnschmidt für seine Beratung bei der Planung meiner statistischen Auswertungen. In diesem Zusammenhang möchte ich mich außerdem bei meinen Kollegen Matthias Kerzel und Dirk Meier-Eickhoff für ihre großartige Unterstützung bei der Durchführung der Probandenstudie bedanken.

Einen sehr wichtigen Baustein für meine empirische Arbeit verdanke ich Natalie Steinmetz und der seekda GmbH aus Innsbruck in Österreich: die Bereitstellung des seekda-Korpus. Diese Daten haben mir einen sehr realistischen Einblick in die Praxis der Benennung von Operationen geliefert und bilden das empirische Fundament meiner Arbeit.

Außerdem danke ich Klaus Lewerentz und Petra Hofstedt für die Organisation des großartigen Doktorandensymposiums auf der Konferenz Software Engineering 2012 in Berlin. Die hier geführten Gespräche haben mich bei der Gestaltung meines Dissertationsprojektes maßgeblich beeinflusst.

Auch in die Erstellung dieser schriftlichen Ausarbeitung sind viele Rückmeldungen eingeflossen. Für die kritische Durchsicht meines Manuskriptes danke ich Tatjana Trautmann, Sven Felix Oberquelle, Alexander Bollens, Florian Stumpf, Max Weichert, Timo Baumann, Dirk Meier-Eickhoff und Sebastian Iwanowski.

Viele Jahre vor Beginn der Arbeit an meiner Dissertation habe ich in einer, für mich sehr schwierigen, persönlichen Situation ausgesprochen viel Unterstützung von meiner früheren Klassenlehrerin Lisa Webber erfahren. Ohne sie wäre mein Lebensweg wahrscheinlich sehr viel anders verlaufen. Daher ist es mir eine Herzensangelegenheit, Lisa an dieser Stelle nochmals für alles zu danken.

Und last, but not least danke ich meiner Freundin Sandra Dammann für Ihre selbst- und vorbehaltlose Unterstützung während der langen Zeitspanne, über die sich dieses Projekt gestreckt hat. Liebe Sandra, ohne dich, dein Wirken und deine unzähligen aufbauenden Worte wäre diese Arbeit niemals entstanden. Vielen, vielen, vielen Dank dafür.

Buxtehude, im November 2013

Jan Christian Krause

Abstract

Many of modern complex software-systems are built of components. These components interact via defined interfaces with operations, so called Application Programming Interfaces (API). The requirements and offered services of an operation are described by its contract. If there are gaps in such a contract, serious misunderstandings between provider and consumer of a component are a probable consequence. Such gaps are called *silence*. In most cases the gaps refer to resources, business rules or possible error cases, which are invisible at the operation's signature. This thesis presents a systematic approach to identify silence.

The presented approach bases on the linguistic analysis of the operation's identifier in natural language. It is shown, that such identifiers often contain english verbs. With these verbs or, more precise, with these predicates one could infer required resources, rules or error cases for the corresponding contract. By applying the linguistic concept of thematic grids the operations are categorized. The verbs as well as the inferred contents are assigned to each of these categories. This thesis suggests such a lexicon which has been derived from empirical investigations. This approach is independent of a concrete natural language (to formulate the identifiers) or programming language (to formulate an operation's signature). In addition a protoypical software assistant is presented, which supports authors when formulating API-contracts.

This work also introduces an approach to measure the amount as well as the relevance of silence in API-contracts. It bases on systematically constructed test cases. By constructing test cases systematically a set of hints to expected contents could be derived. The amount and relevance of silence are quantified by comparing these expectations with an actual contract. The introduced lexicon is evaluated in a case study and a laboratory experiment. Both evaluations show that there is much space to complete API-contracts which have been created with traditional approaches as *javadoc*.

Keywords: Application Programming Interface, API, API-Documentation, API-Specification, Design by Contract, Interface, Specification

Zusammenfassung

Viele moderne Software-Systeme sind aus verschiedenen Komponenten aufgebaut. Diese Komponenten interagieren jeweils über definierte Schnittstellen mit Operationen, sogenannte Application Programming Interfaces (API). Deren Erwartungen und erbrachte Leistungen werden durch einen zur jeweiligen Operation gehörenden Vertrag spezifiziert. Enthält dieser Vertrag Lücken, so sind folgenschwere Missverständnisse zwischen dem Hersteller einer Komponente und deren Nutzern (im Sinne nutzender Programmierer) sehr wahrscheinlich. Diese Lücken werden als *Stille* bezeichnet und betreffen häufig an der Signatur unsichtbare Ressourcen, Regeln oder mögliche Ausnahmefälle. In dieser Arbeit wird ein systematisches Verfahren zur Identifikation dieser Lücken entwickelt.

Dieses Verfahren basiert auf einer linguistischen Analyse des natürlichsprachlichen Namens der Operation. Die vorliegende Arbeit zeigt, dass selbiger häufig englische Verben beinhaltet. Aufgrund des Verbs oder genauer des Prädikates des Namens kann auf notwendige Inhalte des zugehörigen Vertrages rückgeschlossen werden. Dazu werden Operationen gemäß dem linguistischen Konzept der thematischen Raster in Kategorien eingeteilt. Diesen Kategorien werden in einem Lexikon sowohl mögliche Verben als auch die geforderten Inhalte zugeordnet. Diese Arbeit enthält einen auf Basis empirischer Analysen entwickelten Vorschlag für die Initialisierung dieses Lexikons. Das Verfahren kann unabhängig von einer konkreten natürlichen Sprache (zur Formulierung der Bezeichner) oder Programmiersprache (zur Formulierung der Operationssignatur) eingesetzt werden. Es wird ferner ein prototypischer Software-Assistent vorgestellt, der Autoren bei der Formulierung von API-Verträgen auf Basis des neu entwickelten Ansatzes unterstützt.

Diese Arbeit stellt außerdem einen Ansatz zur Quantifizierung des Umfangs und der Relevanz von Lücken in einem API-Vertrag vor. Dieser Ansatz basiert auf systematisch konstruierten Testfällen zu einer Operation. Über die systematische Konstruktion von Testfällen kann ein Erwartungshorizont für API-Verträge konstruiert werden. Vor dem Hintergrund dieses Erwartungshorizontes an Inhalten erfolgt dann die Bewertung des tatsächlichen Vertrages. Abschließend wird das neue Verfahren im Rahmen einer Fallstudie und eines Probandenexperimentes evaluiert. Dabei zeigt sich, dass sich durch das entwickelte Lexikon ein hohes Optimierungspotential für API-Verträge erschließt, die auf Basis konventioneller Verfahren wie *javadoc* formuliert worden sind.

Schlagwörter: Application Programming Interface, API, API-Dokumentation, API-Spezifikation, Schnittstelle, Spezifikation, Vertragsmodell

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Nahezu jeder Programmierer wird im Laufe seines Berufslebens mit der Aufgabe konfrontiert, eine externe Komponente oder eine Bibliothek in das eigene Programm zu integrieren. Die Gründe für diese Form der Wiederverwendung sind vielfältig: Szyperski et al. argumentieren, dass die Entwicklung eines komponenten-orientierten Systems weniger risikobehaftet ist als die vollständige Eigenentwicklung. Der Komponentenhersteller trägt die Aufwände für Fehlerkorrekturen und Weiterentwicklung der Komponente - anders als bei einer vollständigen Eigenentwicklung. Gleichzeitig ist ein solches System aufgrund seines modularen Aufbaus eher an die eigenen Bedürfnisse anpassbar als ein monolithisches Standardsystem (vgl. Szyperski et al., 2002, Seite 4 - 7).

Viele Komponenten bieten zum Zweck der Integration in andere Programme eine Menge an speziellen Schnittstellen und Datenstrukturen an. Diese Menge wird oft als Application Programming Interface (API) bezeichnet. Robillards Untersuchungen zeigen, dass viele Programmierer die Handhabung eines für sie neuen API anhand dessen Beschreibung erschließen. Derartige API-Beschreibungen werden vorerst informell als Spezifikation oder Vertrag bezeichnet. Lücken in den Verträgen stellen eines der größten Hindernisse für das Verständnis eines neuen API dar. Gänzlich fehlende Teile (z.B. Abbildungen oder Begriffsdefinitionen), nicht vorhandene bzw. unpassende Beispiele oder auch Spezifikationslücken hemmen nach Robillard das Verständnis der Programmierer (vgl. Robillard, 2009, Seite 29 f.).

Zweck der Integration einer fremden Komponente oder einer Bibliothek ist immer eine Verhaltensänderung des eigenen Systems. Entweder erlangt es dadurch ein Verhalten, das es vor der Integration nicht besaß, oder bestehendes Verhalten wird optimiert. Verhalten bzw. Tätigkeiten werden in Programmen durch Operationen abgebildet. Daher sind Operationen ein essenzieller Bestandteil eines API. Nurvitadhi et al. zeigen die besondere Relevanz von Operationsbeschreibungen für das Verständnis eines API insgesamt (vgl. Nurvitadhi et al., 2003, Seite T3C-15 f.). Folglich führen Lücken in den API-Verträgen sehr wahrscheinlich zu Missverständnissen zwischen Benutzer (meist ein Programmierer) und Hersteller des API. Im Hinblick auf die Korrektheit des nutzenden Systems ist dies besonders heikel, wenn dessen Programmierer die Lücken eines API-Vertrages durch eigene Annahmen schließen. Aus falschen Annahmen können Fehler in ihren Programmen resultieren. Aber selbst im Falle zutreffender Annahmen drohen zukünftig höhere Aufwände bei der Aktualisierung der zum API gehörenden Komponente auf eine neuere Version. Deren Entwicklung könnte vom Hersteller ebenfalls Annahmen zur Schließung der Lücken fordern. Wenn sich diese Annahmen von denen der Programmierer unterscheiden, ist die Korrektheit der benutzenden Systeme nicht mehr gewährleistet. Auf diese Weise drohen zusätzliche Migrationsaufwände auf beiden Seiten.

Die systematische und möglichst vollständige Ableitung der relevanten fachlichen und technischen Inhalte eines API-Vertrages stellt somit eine große Herausforderung dar. Ziel dieser Arbeit ist die Entwicklung eines Verfahrens, um Lücken in API-Verträgen systematisch zu identifizieren. So sollen folgenschwere Missverständnisse zwischen Anbieter und Hersteller eines API vermieden werden.

1.2 Thesen

Die vorliegende Arbeit widmet sich der Frage: Wie können relevante Inhalte für die Beschreibung einer Operation systematisch und möglichst operationsspezifisch ermittelt werden? Systematisch bedeutet hier, dass die inhaltliche Gestaltung des Vertrages einem definierten Satz von Regeln folgt. Die Frage relevanter und irrelevanter Inhalte für die Beschreibung einer Operation wird in der Literatur seit Langem diskutiert[1].

Viele der bisher veröffentlichten Ansätze orientieren sich zur Bestimmung dieser Inhalte an der technischen Signatur der Operation, also an formalen Parametern, dem Rückga-

[1] Siehe beispielsweise Clements et al. (2003), Seite 228 - 237 oder Szyperski et al. (2002), Seite 55 - 57.

betyp und möglichen Ausnahmen bzw. Fehlernummern. Die Schwäche dieser Verfahren ist, dass sie relevante Aspekte, die nicht an der Signatur sichtbar sind, nicht entdecken (z.B. benötigte Ressourcen oder dem Resultat zugrunde liegende Formeln). Außerdem findet keine inhaltliche Kategorisierung von Operationen statt. Folglich müssen die Autoren der Spezifikation für ähnliche Operationen die notwendigen Inhalte stets aufs Neue ermitteln - trotz möglicher Überschneidungen. Dabei wäre ihnen wahrscheinlich bereits mit Hinweisen auf die entsprechenden Lücken sehr geholfen.

Die im Rahmen der vorliegenden Arbeit gewonnenen Erkenntnisse führen im Ergebnis zu den folgenden Thesen:

These 1 *Durch Analyse des Bezeichners einer Operation unter Berücksichtigung der Semantik von Teilen der natürlichen Sprache können relevante Vertragsbestandteile abgeleitet werden. Die daraus resultierenden Hinweise auf Lücken helfen dem Autor der API-Verträge bei deren Vervollständigung. Auf diese Weise wird deren Qualität insgesamt erhöht.*

These 2 *Operationen sind nach ihrem Zweck kategorisierbar. Aus der Kategorie einer Operation leiten sich die relevanten Inhalte für ihre Beschreibung ab. Je nach Kategorie können diese Inhalte differieren.*

These 3 *Aus der Kategorie einer Operation können spezifische, zu erwartende Fehlerfälle für diese Operation (Checked Exceptions) abgeleitet werden. Für diese Ableitung ist keine Analyse weiterer Quelltexte notwendig.*

These 4 *Sowohl der Umfang als auch die Relevanz von Lücken in Operationsbeschreibungen ist auf Basis von Testfällen quantifizierbar.*

Der in dieser Arbeit vorgeschlagene Ansatz nutzt den natürlichsprachlichen Namen einer Operation (auch *Bezeichner* genannt) zu deren Kategorisierung in einem vordefinierten Lexikon. In diesem Lexikon sind pro Kategorie sowohl technische Inhalte, wie mögliche Fehlerfälle oder benötigte Ressourcen, als auch fachliche Inhalte wie Geschäftsregeln oder Formeln etc. erfasst. Dieser Ansatz bietet gegenüber herkömmlichen Verfahren mehrere Vorteile:

- Da ausschließlich die Signatur der Operation analysiert wird (u.a. ihr Bezeichner), sind die Empfehlungen für zu beschreibende Inhalte sehr effizient berechenbar. Es müssen keine weiteren, aufwändigen Analysen von Quelltexten ausgeführt werden.

- Da dieses Verfahren nicht auf die Analyse von Programm-Quelltexten angewiesen ist, kann es bereits in sehr frühen Stadien der Systementwicklung zur Überprüfung des Entwurfes einer Operation eingesetzt werden. Dafür muss lediglich die Signatur der Operation existieren.

- Aufgrund der Ableitung der relevanten Inhalte aus dem natürlichsprachlichen Bezeichner der Operation sind das Verfahren und dessen Lexikon universell einsetzbar. Es kann mit jeder Programmiersprache verwendet werden, die natürlichsprachliche Bezeichner für Operationen unterstützt.

- Die im Lexikon erfassten Kategorien reduzieren die Menge aller dokumentierbaren Inhalte auf die für die Operation relevanten Inhalte. Auf diese Weise wird dem Autor das Studium umfangreicher Checklisten erspart.

- Das Prinzip der Definition von Operationskategorien in einem Lexikon ist sehr anschaulich. Das Lexikon kann daher ohne spezielle Kenntnisse einfach an die jeweilige Domäne angepasst werden.

- Die Kategorisierung und Analyse der Operation bedürfen sehr prägnanter Bezeichner. Derartige Bezeichner verbessern zusätzlich die Verständlichkeit des Programms und tragen so zur Senkung von Entwicklungs- und Wartungskosten bei.

1.3 Gang dieser Untersuchung

Zu Beginn werden für diese Arbeit grundlegende Begriffe in Kapitel 2 eingeführt und erläutert. Anschließend beschreibt das Kapitel 3 drei Szenarien, in denen die systematische Vermeidung von Stille erheblichen wirtschaftlichen Schäden vorbeugt. Danach widmet es sich auf Basis ausführlicher Literaturrecherchen der Frage, welche Leitlinien und Verfahren zur inhaltlichen Gestaltung von API-Verträgen existieren. Erfahrungsgemäß formulieren viele Programmierer und Software-Architekten ihre API-Verträge sehr abstrakt und lassen viele relevante Aspekte bewusst aus. Aus diesem Grund wird zusätzlich dargelegt, in welchen Situationen und unter welchen Bedingungen die Preisgabe von detaillierteren Angaben in einem Vertrag sinnvoll sein kann.

Darauf aufbauend führt Kapitel 4 in das grundlegende Konzept dieser Arbeit ein: die Beschreibung von Operationen auf Basis des linguistischen Konzeptes der thematischen Raster. Dieses Kapitel legt die theoretischen Grundlagen für den Aufbau des eingangs

erwähnten Lexikons von Operationskategorien. Des Weiteren beschreibt es die empirisch fundierte Gestaltung eines Katalogs an thematischen Rastern. Es stellt ferner das prototypische Werkzeug *iDocIt!* vor, welches diesen Ansatz unterstützt.

Abschließend wird eine Teilmenge des neu entwickelten Katalogs evaluiert. Zu diesem Zweck zeigt das Kapitel 5, wie die Vollständigkeit einer Operationsbeschreibung näherungsweise bewertet werden kann. Außerdem stellt dieses Kapitel einen neuen Ansatz vor, um einzelne Lücken bzgl. ihrer Relevanz zu quantifizieren und dadurch vergleichen zu können. Die eigentliche Evaluierung erfolgt im Rahmen einer Fallstudie und eines Laborexperimentes. Sie wird in Kapitel 6 detailliert beschrieben. Diese Arbeit schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Forschungsthemen in Kapitel 7.

Kapitel 2

Definition grundlegender Begriffe

Die systematische Analyse von API-Verträgen auf Lücken bedarf der präzisen Beschreibung dieser Verträge. Dieses Kapitel führt das hierfür notwendige Begriffsvokabular ein und erläutert es. Eine Übersicht aller eingeführten Begriffe und ihrer Beziehungen untereinander ist in der Abbildung 2.1 dargestellt.

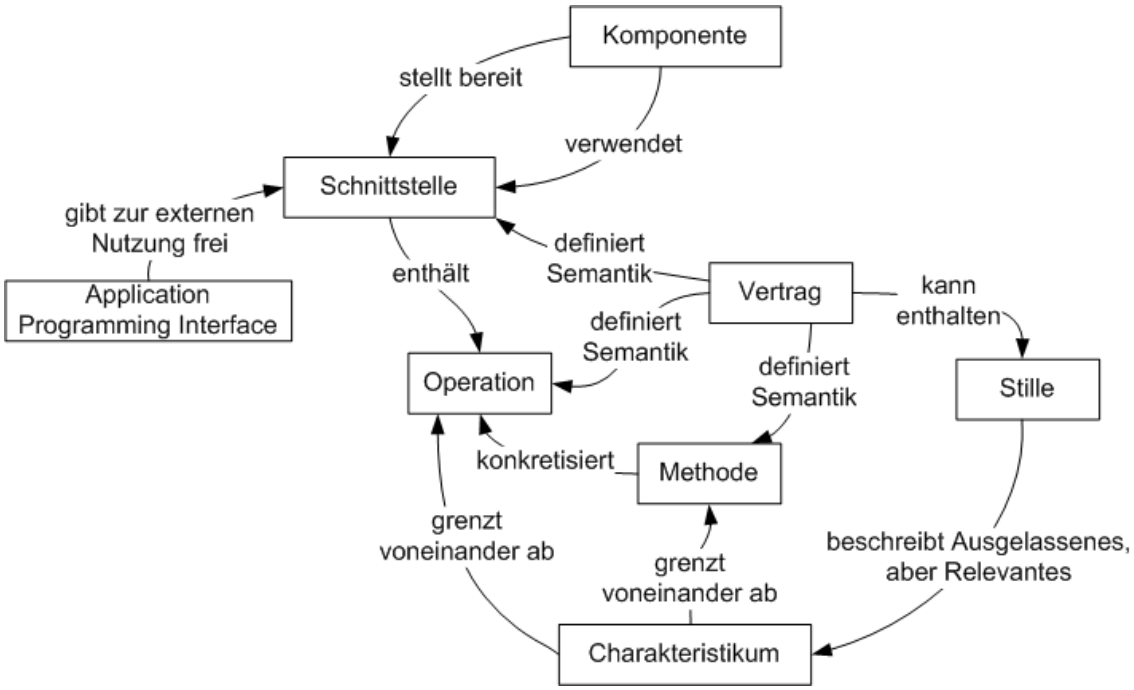


Abbildung 2.1: Beziehungen der grundlegenden Begriffe untereinander

2.1 Operation und Methode

In der software-technischen Literatur existiert eine Reihe von Begriffen zur Repräsentation von Tätigkeiten: z.B. Aktion (vgl. The Object Management Group Inc., 2011b, Seite 225), Aktivität (vgl. The Object Management Group Inc., 2011a,b, Seite 151 (a) und Seite 295 (b)), Methode (vgl. Goldberg und Robson, 1983, Seite 8), Routine, Unterprogramm, Prozedur, Funktion etc.[1] Je nach Autor bzw. Kontext sind diese Begriffe unterschiedlich belegt bzw. mit unterschiedlichen Eigenschaften versehen. Diese Problematik wird anhand des Begriffes Funktion anschaulich. In der Mathematik ist eine Funktion zustandslos. Das bedeutet, ihr Resultat ist eindeutig und hängt ausschließlich von den Belegungen ihrer Parameter ab. In der imperativen Programmierung hingegen kann eine Funktion Werte in globalen Variablen speichern und diese bei späteren Aufrufen verwenden. Eine solche Funktion besitzt folglich einen über mehrere Aufrufe hinweg existierenden Zustand, der das Resultat ebenfalls beeinflussen kann.

Ein gemeinsames, in der Softwaretechnik sehr häufig angewendetes Prinzip besteht in der Trennung zwischen Definition und tatsächlicher Realisierung einer Tätigkeit. Dieses findet sich beispielsweise bei Scheer in der Unterscheidung von drei Ebenen (vgl. Scheer, 1991, Seite 16):

- **Fachkonzept:** Was soll getan werden (Definition von Anforderungen)?
- **Datenverarbeitungskonzept:** Wie soll etwas getan werden (Programmentwurf und -spezifikation)?
- **Implementierungskonzept:** Mit welchen konkreten hardware- und software-technischen Ressourcen soll etwas getan werden?

Das Prinzip der Trennung zwischen der Definition von Anforderungen (was?) und deren Umsetzung (wie?) lässt sich auch auf Operationen übertragen. Eine derartige begriffliche Trennung ist bei Snyder zu finden. Snyder unterscheidet die Begriffe Operation und Methode. Demnach ist eine Operation ein abstraktes Konzept einer Tätigkeit. Sie definiert, was getan werden soll. Zu diesem Zweck besitzt eine Operation eine Signatur. Die Signatur enthält den Namen der Operation, ihre formalen Parameter inkl. Datentypen und das Resultat der Operation ebenfalls inkl. Datentyp (vgl. Snyder, 1991, Seite 6 f.). Die Datentypen geben die Menge aller zulässigen Werte des Parameters bzw. Resultates an. Diese Arbeit folgt Snyders Sichtweise. Daher wird eine Operation wie folgt definiert:

[1] Siehe beispielsweise Meyer (1988), Seite 35 oder Keller et al. (1992), Seite 8.

Definition 1 (Operation) *Eine Operation ist ein abstraktes Konzept zur Abbildung einer Tätigkeit in einem System. Sie besitzt eine Signatur. Die Signatur legt fest, welchen Namen die Operation trägt und welche formalen Parameter in welcher Anordnung sie besitzt. Die Signatur kann zusätzlich die möglichen Resultate, die zu erwartenden Fehlerfälle sowie geltende Vor- und Nachbedingungen festlegen.*

```
1 public abstract class Sorter {
   public abstract List<Integer> sortAsc (
3    Collection<Integer> unsorted);
  }
```

Listing 2.1: Beispiel für eine Operation zur Sortierung

Das Listing 2.1 gibt ein Beispiel für eine Operation zur Sortierung von ganzen Zahlen an. Nach Synder gehört zu einer Operation aber kein Programm. Sie kann folglich nicht ausgeführt werden. Dies unterscheidet sie von einer Methode. Eine Methode ist ein konkretes Konzept zur Abbildung einer Tätigkeit im Sinne einer Umsetzungsvariante. Sie bezieht sich immer auf eine Operation und definiert, wie diese Operation umgesetzt werden soll. Dies geschieht in Form eines zur Methode gehörigen Programms (vgl. Snyder, 1991, Seite 6). So ergibt sich die folgende Definition einer Methode für diese Arbeit (siehe die Beispiele in Listing 2.2):

Definition 2 (Methode) *Eine Methode ist ein konkretes Konzept zur Abbildung einer Tätigkeit in einem System. Sie definiert mittels eines Programms, wie die Vorgaben der zugehörigen Operation umgesetzt werden.*

2.2 Charakteristikum

In der realen Welt existiert eine Fülle von Tätigkeiten, die durch Operationen und Methoden abgebildet werden können (z.B. Sortierungen). Daher stellt sich die Frage, wie diese voneinander abgegrenzt werden können. Im Kontext einer Sortierung sind verschiedene Abgrenzungen denkbar. Jeder Sortierung liegt beispielsweise eine Ordnungsrelation zugrunde. Ferner kann sich das genaue Vorgehen zur Auswahl der miteinander zu vergleichenden Elemente unterscheiden. Unabhängig davon, auf welcher Ebene Tätigkeiten

```
public class QuickSorter extends Sorter {
 public List<Integer> sortAsc (Collection<Integer> unsorted){
     // Quicksort-Implementierung
 }
}

public class BubbleSorter extends Sorter {
 public List<Integer> sortAsc (Collection<Integer> unsorted){
     // Bubblesort-Implementierung
 }
}
```

Listing 2.2: Beispiel für zwei Methoden zur Sortierung

voneinander abgegrenzt werden - sie sind unterscheidbar. Diese Erkenntnis führt zum für diese Arbeit grundlegenden Begriff des Charakteristikums[2]:

Definition 3 (Charakteristikum) *Ein Charakteristikum ist ein Merkmal, das eine Operation oder eine Methode von anderen ihrer Art bzgl. ihrer Bedeutung und ihres Verhaltens durch einen von außen sichtbaren Effekt unterscheidet.*

Jedes Charakteristikum verfügt über eine Belegung. Im obigen Sortierungsbeispiel ist z.B. das Charakteristikum *Ordnungsrelation* mit der Belegung "Aufsteigende Ordnung der ganzen Zahlen" versehen worden (erkennbar am Zusatz "Asc" für "Ascending" im Bezeichner der Operation). Für jedes Charakteristikum ist außerdem die leere Belegung möglich. Auch diese ist das Resultat einer Entwurfsentscheidung. In diesem Fall lässt die Spezifikation bewusst eine Lücke, um Implementierungen Differenzierungsmöglichkeiten zu geben. Wichtig ist, dass die Signaturelemente, z.B. formale Parameter und das Resultat, ebenfalls Charakteristika der Operation sind.

[2] Ein Charakteristikum ist vergleichbar mit der von Liskov und Wing beschriebenen *beweisbaren Eigenschaft* (provable property) in der Subtyp-Anforderung (vgl. Liskov und Wing (1994), Seite 1812). Der Begriff *beweisbar* ist zum Zweck der heuristischen Vervollständigung von API-Verträgen allerdings etwas abzuschwächen. Beispielsweise unterscheiden sich Algorithmen auf Ebene von Methoden häufig ausschließlich im Zeit- oder Speicherplatzbedarf. Dies sind keine auf Basis der Signatur und deren Vor- und Nachbedingungen formal *beweisbaren* Eigenschaften (selbstverständlich sind Komplexitätsbetrachtungen unter Kenntnis des Algorithmus möglich). Dennoch sind sie von außerhalb "beobachtbar". Die Bezeichnung als abgrenzende Eigenschaft für ein Charakteristikum gilt für den Zweck dieser Arbeit als hinreichend.

2.3 Komponente

Nun stellt sich die Frage, auf welche Weise Operationen und Methoden eines Systems in ein anderes integriert werden können. Szyperski spricht in diesem Zusammenhang von Komponenten[3]. Beide grenzen sich aufgrund verschiedener Eigenschaften voneinander ab. Nach Szyperski weist eine Komponente die folgenden Eigenschaften auf (vgl. Szyperski et al., 2002, Seite 36 f.):

- Eine Komponente ist eine unabhängig auslieferbare Einheit.
- Eine Komponente dient der Zusammensetzung (im Sinne von Wiederverwendung) mit anderen Komponenten durch Dritte.
- Eine Komponente ist (von außen betrachtet) zustandslos.

In Anlehnung an diese Eigenschaften wird der Begriff einer Komponente im Rahmen dieser Arbeit wie folgt definiert[4]:

Definition 4 (Komponente) *Eine (Software-)Komponente ist eine in sich geschlossene Auslieferungseinheit. Sie bietet ihre Operationen über spezifizierte Schnittstellen[5] an. Zweck einer Komponente ist die Verwendung dieser Operationen durch andere Komponenten. Verwendet eine Komponente K_2 eine andere Komponente K_1, so wird K_1 als Abhängigkeit von K_2 bezeichnet.*

Im Kontext der Programmierung in Java ist als Komponente beispielsweise eine JAR-Datei vorstellbar. Eine solche Datei enthält kompilierte Java-Klassen. JAR-Dateien sind demnach eine unabhängig auslieferbare Einheit (erste Eigenschaft einer Komponente). JAR-Dateien können in das eigene Java-Programm per Angabe im Klassenpfad (bzw. *classpath*) eingebunden werden. Auf diese Weise ist es möglich, die Klassen aus der JAR-Datei im eigenen Programm zu nutzen (zweite Eigenschaft einer Komponente). Eine JAR-Datei ist außerdem zustandslos (dritte Eigenschaft einer Komponente). Das bedeutet, dass

[3] Szyperski unterscheidet die Begriffe Komponente und Objekt. Im Gegensatz zu Komponenten könnten Objekte instanziiert werden und verfügten über eine eindeutige Identität. Ferner verfüge ein Objekt über einen Zustand und kapsele diesen (vgl. Szyperski et al. (2002), Seite 37 f.). Im weiteren Verlauf dieser Untersuchung wird auf den Begriff des Objektes kein weiterer Bezug genommen. Daher wird er an dieser Stelle auch nicht gesondert definiert.

[4] Vgl. dazu die Definition des Begriffes “Component” in Szyperski und Pfister (1996), Seite 130.

[5] Eine Definition des Begriffes “Schnittstelle” erfolgt in Abschnitt 2.4.

ihr Inhalt durch die nutzenden Programme nicht verändert wird. Alle ausgelieferten JAR-Dateien einer bestimmten Komponentenversion sind identisch - unabhängig davon, von welchem System sie genutzt werden.

2.4 Application Programming Interface

Gemäß der Definition aus dem vorherigen Abschnitt verfügt eine Komponente über spezifizierte Schnittstellen. Nach Szyperski ist eine Schnittstelle das Mittel, um sich mit einer Komponente zu verbinden (vgl. Szyperski et al., 2002, Seite 50). Laut Hohpe und Woolf existiert aus technischer Sicht eine große Bandbreite von Möglichkeiten zur Interaktion zwischen den Komponenten A und B über eine Schnittstelle (Datenübergabe jeweils von A nach B) (vgl. Hohpe und Woolf, 2008, Seite 41 f.):

- **Per Datei:** A schreibt eine Datei mit Datensätzen, welche von B gelesen wird.
- **Per Datenbank:** A schreibt Datensätze in eine Datenbank, die von dort durch B gelesen werden.
- **Per Aufruf einer (entfernten) Operation:** A ruft eine (entfernte) Operation von B auf, übergibt Datensätze als Parameter und startet damit eine Tätigkeit innerhalb von B.
- **Per Nachricht:** A und B sind an einem gemeinsamen Nachrichtenverteiler angeschlossen. A sendet die Datensätze über diesen Verteiler an B.

In dieser Arbeit wird unter einer Schnittstelle die technische Realisierung in Form eines Operationsaufrufes verstanden. Für diese Interpretation wird eine Schnittstelle in Anlehnung an Szyperski wie folgt definiert (vgl. Szyperski et al., 2002, Seite 50):

Definition 5 (Schnittstelle) *Eine Schnittstelle ist eine benannte Menge von benannten Operationen und den zugehörigen Verträgen.*[6]

Szyperski teilt Ausprägungen von Schnittstellen in zwei verschiedene Kategorien ein. Zum einen bietet eine Komponente Schnittstellen zur Nutzung durch andere Komponenten an (provided interfaces). Zum anderen kann eine Komponente Anforderungen an ihre

[6] Die Definition des Begriffes “Vertrag” folgt in Abschnitt 2.5.

Laufzeitumgebung stellen. Diese Anforderungen werden in Form von geforderten Schnittstellen formuliert (required interface). Die Laufzeitumgebung muss dann eine Implementierung für die geforderte Schnittstelle bereitstellen (vgl. Szyperski et al., 2002, Seite 42 - 45).

Szyperski nimmt für diese Unterscheidung eine Blackbox-Perspektive auf die Komponente ein. Aus einer Glassbox-Perspektive wird aber noch eine dritte Art von Schnittstellen erkennbar. Eine Komponente kann auch solche Schnittstellen beinhalten, die nur für den internen Gebrauch, aber nicht den Gebrauch durch externe Komponenten gedacht sind (internal interfaces). Im o.g. Fall einer Java-Komponente als JAR-Datei sind solche Schnittstellen in Form von Java-Interfaces ebenfalls von außen sichtbar.[7] Da diese Arbeit auf die Ableitung relevanter Inhalte für die Integration von Komponenten fokussiert ist, werden die angebotenen Schnittstellen begrifflich besonders abgegrenzt. Dies führt zum Begriff des Application Programming Interface und zur folgenden Definition[8]:

Definition 6 (Application Programming Interface) *Das Application Programming Interface (API) einer Komponente K ist die Menge aller angebotenen Schnittstellen von K inkl. der in den Operationssignaturen dieser Schnittstellen referenzierten Datentypen. Zusätzlich umfasst das API von K die Spezifikationen aller enthaltenen Schnittstellen, Operationen und Datentypen. Der Zweck eines API ist die explizite Benennung von zur externen Nutzung freigegebenen Operationen und Datentypen aus K.*

2.5 Vertrag

Bisher ist im Kontext von Schnittstellen und Operationen stets von zugehörigen Spezifikationen gesprochen worden. Der Begriff des Vertrages wurde sehr informell gebraucht. In diesem Abschnitt wird dieser Begriff in der Art konkretisiert, wie sie Meyer mit dem Prinzip *Design by Contract* geprägt hat (vgl. Meyer, 1992, Seite 2 - 4). Meyer beschreibt als Ausprägung eines Vertrages die Formulierung von Vor- und Nachbedingungen sowie Invarianten (vgl. Meyer, 1992, Seite 5 f. und Seite 11 - 13).

[7] Dieses Problem kann für JAR-Dateien durch eine Namenskonvention für Paketnamen entschärft werden. Im Eclipse-Umfeld werden Klassen und Schnittstellen zur internen Nutzung in Paketen mit dem Namenszusatz ".internal." abgelegt. Detaillierte Informationen dazu sind unter `http://wiki.eclipse.org/Naming_Conventions` zu finden (letzter Abruf: 01.03.2013).

[8] Diese Definition basiert auf der Abgrenzung der Begriffe Signature, Interface und API von Clements et al. in Clements et al. (2003), Seite 245 f.

Zur Abgrenzung von Vertragsinhalten wird im Prinzip aber immer dasselbe Kriterium angewendet: Vertragsinhalte dürfen nur von außen sichtbare Effekte betreffen. Diese "Effekte" werden im Rahmen dieser Arbeit unter dem Begriff des Charakteristikums subsummiert. Deshalb erweitern die folgenden Definitionen die Formulierung von Meyer um den Begriff des Charakteristikums:

Definition 7 (Vertrag einer Operation) *Der Vertrag V_o einer Operation o ist eine Beschreibung welche die zu erbringende Dienstleistung sowie die Rechte und Pflichten des Konsumenten und des Anbieters von o festschreibt. Die Charakteristika von o können Bestandteile des Vertrages sein.*

Definition 8 (Vertrag einer Methode) *Der Vertrag V_m einer Methode m ist dem Vertrag der zugehörigen Operation V_o untergeordnet. Er kann V_o um zusätzliche Charakteristika von m erweitern. Außerdem kann er die enthaltenen "freien" Charakteristika mit einer Belegung versehen.*[9]

Das bereits bekannte Beispiel der Operation `sortAsc` ist in den Listings 2.3 und 2.4 jeweils um einen Operations- bzw. Methodenvertrag ergänzt worden. Erkennbar wird das Charakteristikum *Ordnungsrelation* bereits auf Ebene der Operation festgeschrieben. Den eingesetzten Algorithmus definiert jeweils der Vertrag der Methode.[10]

[9] Ein Charakteristikum wird als "frei" bezeichnet, wenn es über keine Belegung verfügt.

[10] Sehr häufig wird mit einem Vertrag eine möglichst implementierungsunabhängige Beschreibung des Konzeptes hinter einer Operation assoziiert. Diese Einschränkung wird an dieser Stelle explizit noch nicht getroffen. Für eine detaillierte Diskussion dieses Aspektes wird auf Kapitel 3 verwiesen.

```java
/**
 * Sortiert die ganzen Zahlen in "unsorted" aufsteigend
 * gemäß ihrer mathematischen Ordnungsrelation. Der
 * Algorithmus zur Sortierung ist abhängig von der
 * Implementierung.
 *
 * @param unsorted Was sortiert werden soll.
 * @return "unsorted" aufsteigend sortiert
 */
public abstract List<Integer> sortAsc (
  Collection<Integer> unsorted);
```

Listing 2.3: Beispiel für einen Vertrag einer Operation zur Sortierung

```java
/**
 * Nutzt C.A.R. Hoares Quicksort-Algorithmus.
 *
 * @param unsorted Was sortiert werden soll.
 * @return "unsorted" aufsteigend sortiert
 */
public List<Integer> sortAsc (
  Collection<Integer> unsorted){
    // Quicksort-Implementierung
}
```

Listing 2.4: Beispiel für einen Vertrag einer Methode zur Sortierung

```
/**
 * Generiert und verschickt eine E-Mail an die übergebene
 * E-Mail-Adresse.
 *
 * @param recipient
 *  E-Mail-Adresse des Empfängers (darf nicht null
 *       sein).
 * @return True, wenn die E-Mail verschickt worden ist.
 * @throws SendingException
 */
public boolean sendMail
    (String recipient) throws SendingException;
```

Listing 2.5: Beispiel für einen mit Stille behafteten Vertrag

2.6 Stille

Diese Arbeit behandelt Verfahren zur Ableitung relevanter Inhalte für den Vertrag einer Operation. Meyer hat in diesem Zusammenhang zwei sehr anschauliche Metaphern geprägt, die diese Problemstellung charakterisieren: Noise und Silence (vgl. Meyer, 1985, Seite 7). Übertragen auf die Identifikation relevanter API-Vertragsinhalte lassen sich die Begriffe als Stille ("silence") und Rauschen ("noise") ins Deutsche übersetzen. Stille herrsche überall dort, wo der Vertrag bzgl. relevanter Inhalte schweige. Rauschen bezeichne hingegen eine Angabe im Vertrag, die irrelevant für die von der Operation bearbeitete Aufgabe sei. Auch wenn Rauschen ein relevantes Problem bei der Gestaltung von API-Verträgen darstellt, konzentriert sich diese Arbeit auf die Identifikation von Stille. Daher konkretisiert die folgende Definition ausschließlich diesen Begriff:

Definition 9 (Stille) *Ein Vertrag V zu einer Operation o enthält Stille bezüglich des Charakteristikums c, wenn c relevant für o ist und nicht in V erwähnt wird.*

Ein Beispiel für einen Stille enthaltenden Vertrag ist im Listing 2.5 angegeben. Die beschriebene Operation verschickt eine E-Mail an eine E-Mail-Adresse. Der Vertrag schweigt sich aus über die Fälle, in denen eine *SendingException* geworfen wird. So können weder der Architekt noch der Programmierer ein differenziertes Systemverhalten im Fehlerfall

modellieren. Sollte der Fehler in einer syntaktisch falschen E-Mail-Adresse im Formalparameter bestehen, so ist es vielleicht ausreichend, eine Fehlermeldung auf der grafischen Oberfläche anzuzeigen. Der Benutzer kann diese Eingabe ggf. korrigieren. Sollte der Mail-Server nicht verfügbar sein, so ist diese Art der Fehlerbehandlung unzureichend. In diesem Fall ist ein Fehler in der Laufzeitumgebung zu korrigieren (z.B. durch den zuständigen Administrator).

Aber auch zu anderen Charakteristika schweigt der Vertrag. So ist beispielsweise nicht beschrieben, mit welchen Mitteln die Nachricht versendet wird. Wird von der Laufzeitumgebung beispielsweise ein bestimmter Eintrag in einer Konfigurationsdatei oder eine gesetzte System-Eigenschaft bzw. Umgebungsvariable erwartet (z.B. zur Angabe des zu verwendenden Mail-Servers)?[11] Selbst wenn diese Charakteristika nicht an dieser Stelle festgelegt werden sollen, so müsste beschrieben werden, dass sich Methoden bzgl. dieses Charakteristikums unterscheiden können.

Ähnlich verhält es sich mit dem Inhalt der versendeten Mail. Was genau wird hier in welchem Format verschickt? Sollten bestimmte Inhalte in der Operation fest programmiert sein, so ist die Beschreibung des Inhalts wichtig für die Entscheidung, ob die Operation in einem bestimmten Anwendungsfall verwendet werden kann. Alternativ kann der Inhalt auch aus einer bestimmten Vorlage-Datei geladen werden. Aber auch diese Angabe beschreibt eine Anforderung an die Laufzeitumgebung, deren Erfüllung vor Produktivsetzung der Operation sichergestellt werden muss. Außerdem resultiert aus der Nichtverfügbarkeit dieser Datei eine neue mögliche Fehlersituation, auf die das System reagieren muss.

[11] Eine derartige Konfiguration ist in Form der Konfigurationseinträge (Properties) *mail.protocol.host*, *mail.protocol.port* und *mail.protocol.user* für Implementierungen der Java Mail API-Spezifikation vorgesehen. Diese Einträge werden in der Konfigurationsdatei erwartet. Für weitere Informationen siehe Oracle Corp. (2013b), Abschnitt “Overview”.

Kapitel 3

Gestaltung von API-Verträgen

In Kapitel 1 ist die Identifikation von Stille in API-Verträgen motiviert worden, um Missverständnisse zwischen Benutzern und Hersteller einer Komponente zu vermeiden. Als Ausgangspunkt für die weitere Untersuchung ist diese allgemeine Situationsbeschreibung allerdings noch nicht hinreichend. Daher skizziert der erste Abschnitt dieses Kapitels drei Szenarien, in denen Stille in API-Verträgen zu erheblichen wirtschaftlichen Schäden führen kann.

Die systematische Vermeidung von Stille setzt zudem die Erwartung bestimmter Inhalte sowie Leitlinien zu deren Auswahl voraus. Erfahrungsgemäß scheinen viele Programmierer und Software-Architekten intuitiv zur größtmöglichen Abstraktion bei der Formulierung von API-Verträgen zu neigen. Aus diesem Grund wird im zweiten Abschnitt recherchiert, welche Erwartungshaltungen bzgl. der Inhalte von API-Verträgen in der Literatur veröffentlicht worden sind. Anschließend prüft der dritte Abschnitt, an welchen Orten und in welcher Form API-Verträge niedergeschrieben werden sollten. Danach werden im vierten Abschnitt aktuelle Ansätze zur Gewährleistung der Vollständigkeit von API-Verträgen dargestellt. Den Schluss bildet eine Zusammenfassung der gewonnenen Erkenntnisse.

3.1 Szenarien zur Vermeidung von Stille

Im Rahmen dieser Arbeit konnten drei verschiedene Szenarien identifiziert werden, in denen der Vermeidung von Stille eine bedeutende Rolle zukommt. Auf Basis dieser Szenarien können später die besonders relevanten Arten von fehlenden Inhalten und Möglichkeiten

zu ihrer Identifikation untersucht werden. In allen drei Szenarien dient die Vollständigkeitsprüfung der Qualitätskontrolle vorliegender API-Verträge.[1]

3.1.1 Qualitätskontrolle eines Herstellers

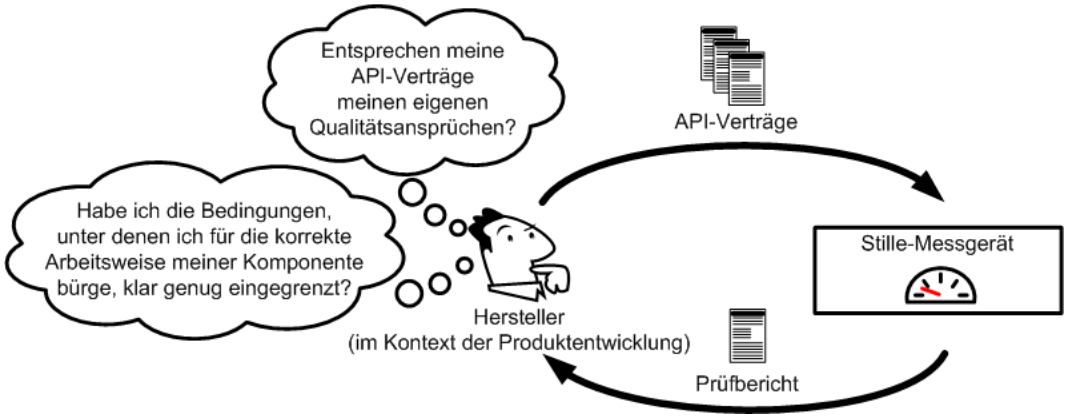


Abbildung 3.1: Grafische Darstellung des 1. Szenarios

Szenario 1 (Hersteller führt Qualitätskontrolle durch) *Der Hersteller h kontrolliert das API seiner Komponente komp vor der Freigabe der nächsten Version auf Einhaltung seiner internen Qualitätsstandards. Zu diesen Standards zählt u.a. die Vollständigkeit der API-Verträge. Durch die Vollständigkeitsprüfung erreicht h eine gute Reputation seiner Komponente unter Programmierern und somit unter potentiellen Käufern. Des Weiteren erhält h durch vollständige Verträge Rechtssicherheit durch Schutz vor Schadensersatzansprüchen seiner Kunden. h garantiert die korrekte Arbeitsweise seiner Komponente nämlich nur unter bestimmten Betriebsbedingungen. Diese Bedingungen müssen in den API-Verträgen von komp eindeutig dargelegt werden. h liegen alle in komp enthaltenen Quelltexte und API-Verträge vollständig vor. Dieses Szenario ist in Abbildung 3.1 dargestellt.*[2]

[1] Die dargestellten Fragestellungen der Szenarioteilnehmer können auf mehrere Szenarien zutreffen. Aus Gründen der Übersichtlichkeit ist jede Fragestellung in nur einem besonders charakteristischen Szenario dargestellt worden.

[2] Im Rahmen der Erstellung dieser Abbildung ist die Grafik "MC900239195.WMF" aus der Microsoft Clip Art and Media-Gallerie verwendet worden (`http://office.microsoft.com/de-de/images/`; letzter Abruf am 23.06.2013). Nutzung mit Genehmigung von Microsoft.

3.1.2 Qualitätskontrolle eines Auftragnehmers

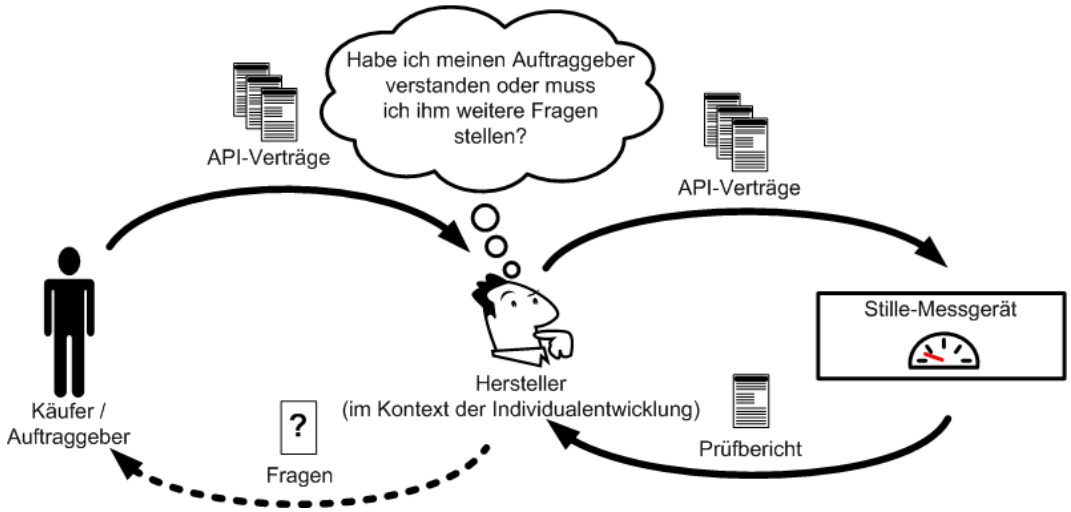


Abbildung 3.2: Grafische Darstellung des 2. Szenarios

Szenario 2 (Auftragnehmer führt Qualitätskontrolle durch) *Der Auftragnehmer a prüft die API-Verträge der von ihm zu entwickelnden Komponente komp auf Vollständigkeit. Er hat die API-Verträge von seinem Auftraggeber als Spezifikation erhalten. Auf diese Weise stellt a sicher, dass die von ihm entwickelte Komponente alle Anforderungen des Auftraggebers erfüllt. Nur auf Basis einer vollständigen Spezifikation kann er wirklich alle Aufwände zur Implementierung der Anforderungen in seiner Angebotskalkulation berücksichtigen. a liegen die API-Verträge für komp als Spezifikation des Entwicklungsauftrages vor. Quelltexte von komp existieren noch nicht, weil komp erst im Rahmen des Auftrages implementiert werden soll. Juristisch gesehen handelt es sich bei den API-Verträgen dieses Szenarios um einen Bestandteil des zwischen Auftraggeber und Auftragnehmer geschlossenen Werkvertrages nach § 631 BGB. Dieses Szenario ist in Abbildung 3.2 dargestellt.*[3]

3.1.3 Qualitätskontrolle eines Käufers

Szenario 3 (Käufer führt Qualitätskontrolle durch) *Der Käufer k prüft das API der Komponente komp, deren Kauf er erwägt, auf Vollständigkeit. Auf diese Weise stellt*

[3] Im Rahmen der Erstellung dieser Abbildung ist die Grafik "MC900239195.WMF" aus der Microsoft Clip Art and Media-Gallerie verwendet worden (`http://office.microsoft.com/de-de/images/`; letzter Abruf am 23.06.2013). Nutzung mit Genehmigung von Microsoft.

k sicher, dass weitere Kompatibilitätsprüfungen von komp gegenüber seinen fachlichen Anforderungen und seiner Laufzeitumgebung sinnvoll durchgeführt werden können. Außerdem prüft er die Sorgfalt des Lieferanten. k liegen alle API-Verträge vor. k hat aber keinen Zugriff auf die Quelltexte von komp.[4] *Juristisch gesehen handelt es sich bei den API-Verträgen dieses Szenarios um einen Bestandteil des zwischen Auftraggeber und Auftragnehmer geschlossenen Kaufvertrages nach § 433 BGB. Dieses Szenario ist in Abbildung 3.3 dargestellt.*[5]

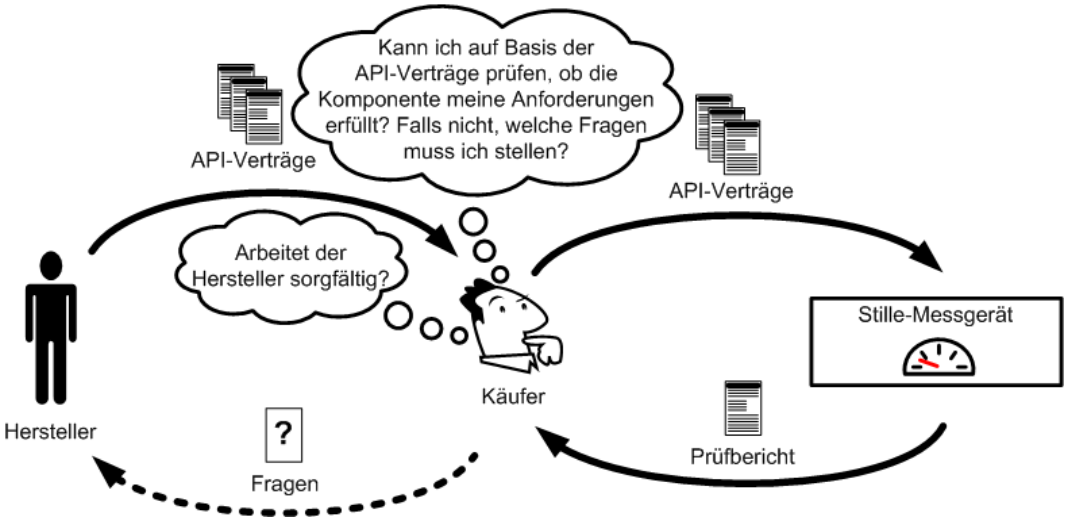


Abbildung 3.3: Grafische Darstellung des 3. Szenarios

Aus diesen Szenarien können verschiedene Erkenntnisse gewonnen werden. Szenario 1 illustriert die besondere Relevanz von Betriebsbedingungen einer Komponente, die in deren API-Verträgen spezifiziert werden müssen. Dazu gehören z.B. benötigte Ressourcen wie Datenbanken, (Konfigurations-)Dateien, Umgebungsvariablen etc. Szenario 2 zeigt die besondere Relevanz von fachlichen Regeln oder Algorithmen bzw. die Benennung der beteiligten Fremdkomponenten, deren Integration den späteren Implementierungsaufwand maßgeblich determiniert. Auch gehören mögliche Fehlerzustände und deren entsprechende Behandlung in diese Kategorie. Die Notwendigkeit der Spezifikation sowohl von Be-

[4] Es gibt Lizenzmodelle, bei denen den Käufern einer Komponente auch die entsprechenden Quelltexte ausgeliefert werden (z.B. bei der GNU Public Licence (GPL)). Da dies allerdings nicht für den allgemeinen Fall aller Lizenzen gilt, wird in diesem Szenario von der Nichtverfügbarkeit der Quelltexte ausgegangen.

[5] Im Rahmen der Erstellung dieser Abbildung ist die Grafik "MC900239195.WMF" aus der Microsoft Clip Art and Media-Gallerie verwendet worden (`http://office.microsoft.com/de-de/images/`; letzter Abruf am 23.06.2013). Nutzung mit Genehmigung von Microsoft.

triebsbedingungen als von Verhalten in Form von fachlichen Regeln, Algorithmen und Fehlerzuständen wird ebenfalls durch Szenario 3 demonstriert.

Die Szenarien zeigen außerdem eine weitere wichtige und herausfordernde Einschränkung für ein Verfahren zur Identifikation von Stille. Nur der Hersteller aus Szenario 1 verfügt über den Zugriff auf die Quelltexte der Komponente, anders als der Auftragnehmer aus Szenario 2 oder der Käufer aus Szenario 3. Somit sollte das zu entwickelnde Verfahren ohne Kenntnis des Quelltextes einer Implementierung eine Vollständigkeitsprüfung durchführen können. Ein solches Verfahren hätte zusätzlich den Vorteil, bereits in sehr frühen Entwicklungsphasen einer Komponente zur Überprüfung der getroffenen Entwurfsentscheidungen bzgl. des API einsetzbar zu sein. Dies begünstigt schließlich deren kostenminimale Korrektur.

3.2 Prinzipien zur Vertragsgestaltung

API-Verträge sind öffentliche Dokumente. Daher stellt sich bei ihrer Erstellung immer die Frage nach den Inhalten, die preisgegeben werden. Die Formulierung von API-Verträgen kann deshalb im Grundsatz als Auswahlproblem formuliert werden. Aus einer Kandidatenmenge von denkbaren Inhalten sind bestimmte Inhalte auszuwählen und im Vertrag der Operation niederzuschreiben. Aus welchen Inhalten wird die Kandidatenmenge gebildet und welche Kandidaten gehören in einen API-Vertrag? Diese Fragestellung führt zur Recherchefrage 1.

Recherchefrage 1 *Welche Inhalte gehören in einen API-Vertrag?*

3.2.1 Das Geheimnisprinzip

Brooks gibt eine Näherung in Form des Project Workbook an. Das Project Workbook ist eine strukturierte Sammlung aller Dokumente, die während der Laufzeit eines Projektes erstellt werden. Dazu gehören z.B. die Definitionen der Projektziele, Spezifikationen, technische Standards oder administrative Memoranden. Für das Anlegen eines Project Workbook sprechen nach Brooks zwei Argumente. Zum einen konserviert es sämtliches Wissen zum Projekt. Dieses Wissen ist damit zu späteren Zeitpunkten abrufbar, um z.B. Handbücher zu schreiben oder die Gründe für einen bestimmten Systementwurf nachzuvollziehen. Zum anderen kann mit Hilfe des Project Workbook gewährleistet werden,

dass jeder Projektmitarbeiter Zugriff auf alles für ihn relevante Wissen bzgl. des Projektes erhält. Jeder Programmierer verfügt über eine eigene Kopie des vollständigen Project Workbook. Aufgrund der Öffentlichkeit ihrer Arbeit arbeiten die Mitarbeiter zudem sorgfältiger (vgl. Brooks Jr., 1995, Seite 75 - 78). Außerdem können sie sich durch gegenseitige Prüfung ihrer Arbeit unterstützen. Beides trägt zur Qualitätssicherung der gesamten Software bei (vgl. Brooks Jr., 1995, Seite 271).

Die von Brooks ins Feld geführte Konservierung von Wissen erlaubt es, zu einem späteren Zeitpunkt aufkommende Fragen zu beantworten. Insofern lässt sich die Kandidatenmenge etwas abstrakter auch als die Menge aller Antworten auf Fragen bzgl. des entwickelten Systems bezeichnen. Nach Brooks' Ansatz werden Antworten auf alle möglichen Fragen im Vertrag veröffentlicht. Dieses Prinzip ist in Abbildung 3.4 dargestellt.

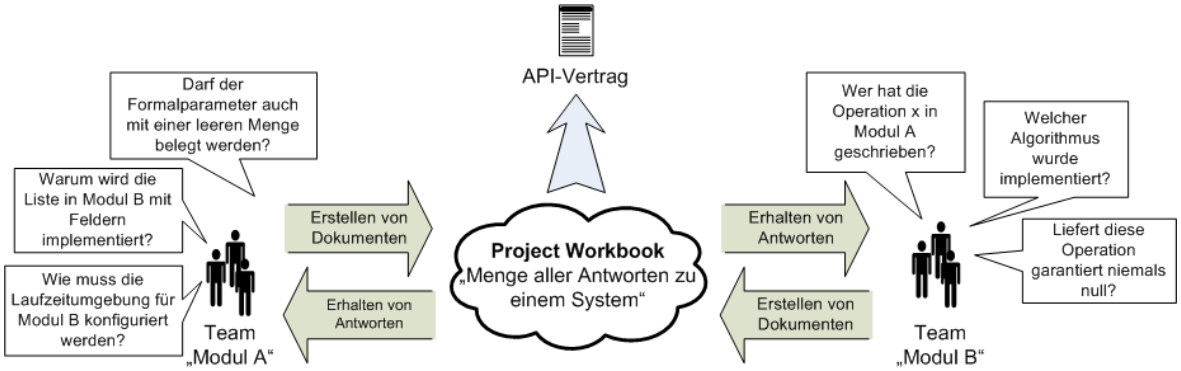


Abbildung 3.4: Kommunikation zwischen Teams über das Project Workbook

Allerdings birgt die in Brooks' Konzept beschriebene, vollständige Transparenz auch einen gewichtigen Nachteil in sich. Dieser tritt oft bei der Änderung eines bestehenden Programmteils in Erscheinung. Parnas argumentiert, dass gute Programmierer ihr Wissen nutzen, um die bestmöglichen und effizientesten Programme zu schreiben (vgl. Parnas, 1971, Seite 341 f.). Es besteht nach Parnas somit die Gefahr, dass die Programmierer des Moduls A aufgrund ihres derzeitigen Wissens Annahmen über Modul B treffen und es entsprechend ihren Annahmen benutzen. Sobald die Programmierer von Modul B die Arbeitsweise ihres Moduls aber ändern, droht Modul A nicht mehr korrekt zu funktionieren. Es besteht folglich eine schwer zu identifizierende Kopplung auf Basis von Annahmen zwischen beiden Modulen (vgl. Parnas, 1971, Seite 339). In der Jubiläumsauflage seines Klassikers "The Mythical Man-Month" revidiert Brooks seine frühere Position und bestätigt Parnas' These (vgl. Brooks Jr., 1995, Seite 271 f.).

Diese Problematik wird im Folgenden anhand eines kleinen Beispiels veranschaulicht. Angenommen, Modul B stellt die Operation *findeKunden* bereit. Diese Operation liefert eine Liste von Kunden-Objekten für einen Nachnamen, der als Parameter übergeben wird. Die aktuelle Methode dieser Operation sortiert die Kunden nach ihrem Vornamen. Da Modul A eben diese Sortierung benötigt, wird die Liste dort in der gelieferten Form direkt weiterverarbeitet. Zu einem späteren Zeitpunkt wird von der Projektleitung eine neue Anforderung an Modul B formuliert. Die Erfüllung dieser Anforderung setzt eine Operation zur Suche voraus, die für einen gegebenen Nachnamen die betreffenden Kunden-Objekte sortiert nach deren Kundennummer liefert. Die Programmierer von Modul B ändern die Sortierung der Methode *findeKunden* entsprechend. Nun funktioniert Modul A aber nicht mehr wie ursprünglich entworfen. Modul A muss die benötigte Sortierung nach dem Vornamen jetzt selbst herstellen. In diesem Beispiel treffen die Programmierer von A die Annahme, dass die Sortierung in B nicht mehr geändert wird. Die Programmierer von B hingegen nehmen an, dass niemand das Implementierungsdetail "Sortierung" als unveränderlich voraussetzt.

Aus diesem Grund schlägt Parnas vor, Antworten auf Fragen zum Projekt nicht frei, sondern kontrolliert und beschränkt zur Verfügung zu stellen. Um dies zu erreichen, erstellt der Architekt während des Systementwurfes für jedes Modul eine "externe Spezifikation". Diese Spezifikation ist öffentlich, sodass andere Module auf den hier festgeschriebenen Charakteristika aufbauen können. Diejenigen Charakteristika, die sich zu einem späteren Zeitpunkt ändern könnten, dürfen nicht in die externe Spezifikation aufgenommen werden (vgl. Parnas, 1971, Seite 342). Wie in Abschnitt 2.2 dargelegt, resultieren Charakteristika aus Entwurfsentscheidungen. Parnas empfielt, Module derart zu schneiden, dass möglichst wenige Module von einer Entwurfsentscheidung abhängig sind (idealerweise nur ein einziges). Dieses Prinzip halte die externen Spezifikationen so knapp wie möglich und reduziere damit die Kopplung der Module untereinander auf ein Minimum. Es wird als Geheimnisprinzip bezeichnet ("principle of information hiding") (vgl. Parnas, 1972, Seite 1056 und 1058).

3.2.2 Abgrenzung weiterer Entwurfsprinzipien

In der softwaretechnischen Literatur sind über die vergangenen Jahrzehnte neben dem Geheimnisprinzip weitere Prinzipien als Leitlinien für den Entwurf langlebiger Softwaresysteme publiziert worden. Martin subsummiert die Anwendung derartiger Prinzipien

unter dem Begriff *agile design* (vgl. Martin, 2003, Seite 94). Er hebt insbesondere die folgenden fünf Entwurfsprinzipien besonders hervor:

- Single-Responsibility-Prinzip: Jede Klasse darf nur eine einzige Verantwortung haben (vgl. Martin, 2003, Seite 95).[6]
- Open-Closed-Prinzip: Module müssen offen für Erweiterungen und geschlossen für Modifikationen sein.[7]
- Liskov'sches-Substitutionsprinzip: Eine Instanz einer abgeleiteten Klasse muss sich bzgl. der gemeinsamen Operationen genauso wie eine Instanz der Basisklasse verhalten.[8]
- Interface-Segregation-Prinzip: Schnittstellen dürfen nur Operationen anbieten, die ihre Nutzer benötigen, und keine darüber hinaus (vgl. Martin, 2003, Seite 137 f.).
- Dependency-Inversion-Prinzip: Module einer höheren Ebene dürfen nicht von Modulen einer niedrigeren Ebene abhängen. Beide Module benötigen eine gemeinsame Abstraktion. Abstraktionen dürfen nur auf Abstraktionen basieren und nicht auf Details von einzelnen Modulen (vgl. Martin, 2003, Seite 127).

Diese Art von Entwurfsprinzipien geben dem Programmierer bzw. Software-Architekten Hilfestellung bei der Aufteilung des Quelltextes, z.B. bei der Auswahl der in einer Klasse zu kapselnden Operationen bzw. Methoden. Insofern resultieren auch aus diesen Prinzipien Entwurfsentscheidungen. Die getroffenen Entwurfsentscheidungen spiegeln sich selbstverständlich in den Belegungen der Charakteristika einer Operation wider. Damit haben sie direkten Einfluss auf die Gestaltung des API-Vertrages.

All diesen Prinzipien ist gemein, dass sie Leitlinien für die Anordnung des Quelltextes im Gesamtsystem vorgeben. Sie stellen daher vorrangig eine Begründung für den gewählten Entwurf dar und weniger die Entwurfsentscheidung an sich. Folglich nehmen sie eine eher dokumentierende und weniger spezifizierende Rolle ein. Der Fokus dieser Arbeit liegt aber auf der Vollständigkeit des Vertrages, also der Spezifikation (und nicht der Dokumentation) einer Operation. Es werden daher keine weiteren dieser Entwurfsprinzipien mit Bezug zur Vertragsgestaltung im Detail diskutiert.

[6] Martin spricht anstelle der Verantwortung von einem einzigen Grund zur Veränderung (vgl. Martin, 2003, Seite 97).

[7] Vgl. Martin (2003), Seite 99 f. und Meyer (1988), Seite 23 - 25

[8] Vgl. Liskov und Wing (1994), Seite 1812 und Martin (2003), Seite 111 f.

Der Leser mag einwenden, dass dieser Umstand auch auf das Geheimnisprinzip zutrifft. Das ist soweit auch korrekt. Da die Anwendung des Geheimnisprinzips den Quelltext aber derart strukturiert, dass die Verträge minimal werden, hebt es sich in der Form seines Einflusses auf Vertragsinhalte beispielsweise von den oben genannten *agile Design*-Prinzipien stark ab. Aus diesem Grund bildet das Geheimnisprinzip eine Ausnahme.

3.2.3 Zusicherungen und Erwartungen

Das Geheimnisprinzip von Parnas bildet eine wichtige Grundlage für die Gestaltung von Verträgen. Es begründet die Existenz des Vertrages als externe Spezifikation. Allerdings hilft das Geheimnisprinzip allein noch nicht, konkrete Inhalte für den Vertrag auszuwählen. Es plädiert lediglich dafür, genau und ausschließlich diejenigen Inhalte über Modul B an Programmierer anderer Module zu geben, die sie zur Benutzung von B benötigen (vgl. Parnas, 1971, Seite 342).

Zur Charakterisierung von Klassen und Operationen schlägt Meyer die Metapher des Vertrages vor. Gemäß der Definition aus Abschnitt 2.5 ergeben sich Vor- und Nachbedingungen sowie Invarianten aus Verhandlungen zwischen den Benutzern einer Operation und ihrem Anbieter (vgl. Meyer, 1992, Seite 2 - 4). Die Benutzer der Operation verpflichten sich, die Vorbedingungen zu erfüllen. Der Anbieter der Operation hingegen gewährleistet die Gültigkeit von Invarianten und Nachbedingungen (vgl. Meyer, 1992, Seite 9). Ein Beispiel für einen Vertrag nach Meyer enthält das Listing 3.1 in einer an die Programmiersprache Eiffel angelehnten Notation.

Dieser Vertrag beschreibt die Erwartungen und Zusicherungen der Operation *meldeAn* im Kontext des Beobachter-Entwurfsmusters.[9] Über diese Operation können sich Beobachter bei einem Subjekt registrieren. Nach erfolgreicher Registrierung eines Beobachters benachrichtigt das Subjekt ihn nach Änderung des eigenen Zustandes. Als Vorbedingung wird erwartet, dass die Operation niemals mit einem leeren Beobachter als Aktualparameter aufgerufen wird (siehe Zeile 8). Die Benutzer müssen dies sicherstellen. Im Gegenzug sichert das Subjekt den Aufrufern als Nachbedingung zu, dass der übergebene *Beobachter* nach Aufruf von *meldeAn* in der Menge der Beobachter des Subjekts enthalten ist (siehe Zeile 10). Außerdem garantiert das Subjekt, dass der Zustand nie der leeren Referenz entspricht (Invariante, siehe Zeile 12).

[9] Für eine ausführliche Darstellung des Beobachter-Musters siehe Gamma et al. (2004), Seite 287 - 300.

```
class Subjekt
feature
  zustand : Zustand
  ...
  meldeAn(beob: Beobachter)
    -- Registriert den Beobachter beob
    -- an diesem Subjekt.
    require beob /= Void
    do ...
    ensure beob ∈ beobachter
    end -- Methode meldeAn
  invariant zustand /= Void
end -- Klasse Subjekt
```

Listing 3.1: Beispiel für einen Vertrag auf Operationsebene mit Vor- und Nachbedingungen

Vor- und Nachbedingungen sowie Invarianten charakterisieren den Zustand des Programms - vor und nach Aufruf einer Operation. Der Begriff Zustand umfasst die Aktualparameter sowie die Werte der von der Operation aus zugreifbaren globalen Variablen. Andere Autoren fassen den Vertragsbegriff aber deutlich weiter. Helm et al. argumentieren, dass weitere Spezifikationen in objektorientierten Programmen auf Klassen- bzw. Modulebene von besonderer Relevanz sind (vgl. Helm et al., 1990, Seite 169). Den Vertragsbegriff von Meyer erweitern sie einerseits um die Zusicherung des Aufrufs bestimmter Operationen anderer Objekte. Andererseits schlagen sie vor, die verhaltenstechnischen Abhängigkeiten von Objekten untereinander explizit durch den Vertrag zu definieren (vgl. Helm et al., 1990, Seite 171).

Eine weitere Ergänzung des Vertragsbegriffes offenbart sich auf der Ebene einer Komponente. Gemäß der Definition einer Komponente in Abschnitt 2.3 sind Komponenten in sich geschlossene Auslieferungseinheiten. Damit besteht nach Szyperski die Notwendigkeit der Anforderungspezifikation der Komponente an ihre Laufzeitumgebung. Auf diese Weise könne der Betreiber der Laufzeitumgebung sicherstellen, dass die Komponente ordnungsgemäß funktioniert (vgl. Szyperski et al., 2002, Seite 44 f.). Ólafsson und Bryan konkretisieren diese Anforderung in Form von Schnittstellen anderer Komponenten, für die in der Laufzeitumgebung eine Implementierung verfügbar sein müsse. Diese Schnitt-

stellen werden als “required interfaces” bezeichnet (vgl. Ólafsson und Bryan, 1996, Seite 162 - 164).

3.2.4 Verhandlungen zwischen Benutzer und Anbieter

Insgesamt ist erkennbar, dass sich Verträge in ihrer grundsätzlichen Struktur gleichen - unabhängig davon, auf welcher Ebene im System sie geschlossen werden. Diese Analogie ist in Abbildung 3.5 dargestellt. Es gibt immer zwei Vertragspartner: Benutzer und Anbieter. Verhandlungsgegenstände sind immer die Zusicherungen und Erwartungen bzgl. der Ein- und Ausgaben gemäß dem EVA-Prinzip (**E**ingabe, **V**erarbeitung und **A**usgabe) auf der jeweiligen Systemebene.[10] Der Benutzer muss sicherstellen, dass die Eingaben den Anforderungen des Vertrages entsprechen. Selbiges gilt für den Anbieter bzgl. der Ausgaben.

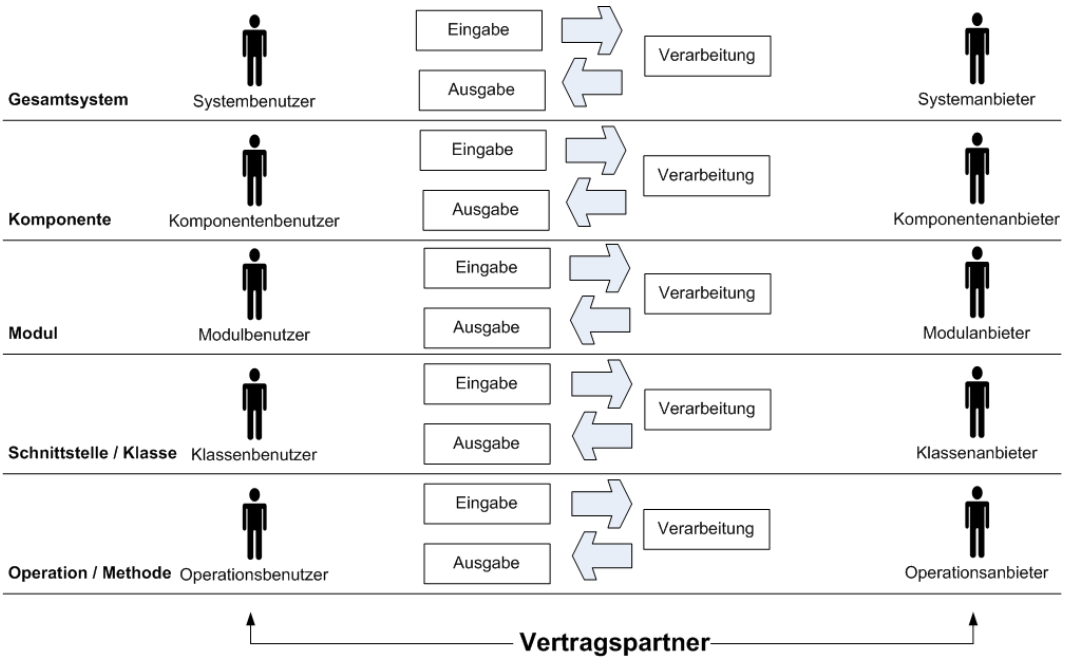


Abbildung 3.5: Analogie des Vertragsmodells auf den verschiedenen Systemebenen

Aufgrund der strukturellen Gleichheit der Verträge zwischen den verschiedenen Ebenen besteht auf allen Ebenen ein Grundsatzkonflikt zwischen Benutzer und Anbieter,

[10] Die strukturelle Gleichheit der Verträge auf den verschiedenen Systemebenen ist nicht allein auf objektorientierte Programme beschränkt. Aus diesem Grund wird das EVA-Prinzip hier als eines der grundlegendsten Prinzipien der Datenverarbeitung z.B. gegenüber Interaktionen zwischen Objekten bevorzugt.

den es zu verhandeln gilt. Meyer skizziert diesen Konflikt auf Ebene der Operation wie folgt: Der Benutzer hat im Allgemeinen ein natürliches Interesse an möglichst schwachen Vorbedingungen. Der Anbieter muss in diesem Fall eine Vielzahl verschiedener Eingaben auf definierte Weise verarbeiten können. Dies spart Implementierungsaufwände und damit Kosten auf Seiten des Benutzers. Diese Aufwände sind vom Anbieter zu leisten. Der Anbieter wiederum strebt meist danach, diese Aufwände zu vermeiden. Er favorisiert folglich strenge Vorbedingungen und verlagert damit Implementierungsaufwände auf die Seite des Benutzers. Bzgl. der Nachbedingungen verhält es sich genau umgekehrt. Der Benutzer verhandelt nach Möglichkeit strenge Nachbedingungen, wogegen der Anbieter ein Interesse an möglichst weichen Nachbedingungen hat. Auch hier sind die gegensätzlichen Interessenlagen mit der Vermeidung von Aufwänden zu erklären (vgl. Meyer, 1992, Seite 9).

Je höher die verhandelte Systemebene liegt, desto mehr droht insbesondere dem Anbieter ein Dilemma. Wie eben dargelegt, strebt er nach möglichst strengen Vorbedingungen. Auf Komponentenebene schließen diese Vorbedingungen auch Anforderungen an die Laufzeitumgebung der Komponente ein. Bei der Spezifikation benötigter Ressourcen im Vertrag, z.B. in Form des Datenbanksystems eines bestimmten Herstellers in einer bestimmten Version, droht allerdings eine Verletzung des Geheimnisprinzips. Der Benutzer der Komponente erhält in diesem Fall über den Vertrag Kenntnis von einer Entwurfsentscheidung des Anbieters, z.B. die Nutzung einer bestimmten Datenbank. Es droht die Gefahr, dass der Benutzer dieses Wissen als Annahme in seinem Programm kodiert, z.B. in Form direkter Manipulation des Zustandes der benötigten externen Komponenten. Im Falle der angesprochenen Datenbank könnte eine solche Manipulation beispielsweise in einem direkten Verbindungsaufbau und der Absendung einzelner UPDATE-Anweisungen bestehen. Der Anbieter hat drei verschiedene Möglichkeiten, die Verletzung des Geheimnisprinzips zu entschärfen:

1. Abwägung zwischen heutigen und zukünftigen Einsparungen.

2. Herbeiführung eines unvollständigen Vertrages.

3. Explizite Beschränkung der Nutzung.

Zum einen kann der Anbieter die von seiner Komponente benötigten anderen Komponenten selbst mitliefern, z.B. die erwähnte benötigte Datenbank. Im schlimmsten Fall muss er die Abhängigkeiten seiner Komponente sogar selbst implementieren. Dann hat er heute

die notwendigen Implementierungsaufwände zu tragen. Der Anbieter wahrt dadurch aber das Geheimnisprinzip und zugleich die eigene Freiheit zur zukünftigen Änderung dieser und aller darauf basierenden Entwurfsentscheidungen. Alternativ kann er heute Implementierungsaufwände sparen bzw. sie auf die Seite des Benutzers delegieren - indem er sich die Bereitstellung der benötigten Komponenten vertraglich zusichern lässt. Möchte er in Zukunft aber seine Entwurfsentscheidung ändern, so schlagen dann höhere Aufwände zur Änderung, zum Test des Programms sowie ggf. zur Datenmigration zu Buche.

Die zweite Möglichkeit besagt, dass der Anbieter die Abhängigkeiten von den benötigten im Vertrag verschweigen und ihre Verfügbarkeit in der Laufzeitumgebung trotzdem voraussetzen kann. In diesem Fall würde die Komponente des Anbieters in der vertragskonformen Laufzeitumgebung aber nicht funktionieren. In Deutschland regelt das Bürgerliche Gesetzbuch diesen Sachverhalt in rechtlicher Hinsicht:

1. Im Falle einer Standardsoftware ein Sachmangel der Komponente (Sache), da sie nicht die im Kaufvertrag vereinbarte Beschaffenheit hat (siehe § 434 BGB).

2. Im Falle einer Individualsoftware ein Sachmangel der Komponente (Werk), da sie nicht die im Werkvertrag vereinbarte Beschaffenheit hat (siehe § 633 BGB).

Dem Verkäufer bzw. Unternehmer drohen in diesem Fall Nacherfüllungsforderungen, der Rücktritt des Käufers bzw. Bestellers vom Kauf- bzw. Werkvertrag, Kaufpreisminderung und Schadensersatzansprüche (siehe § 437 BGB und § 635 BGB). Die dritte Möglichkeit bedeutet die Nennung der benötigten Komponente im Vertrag. Allerdings wird dort jegliche anderweitige Benutzung explizit untersagt. Das Hosting-Unternehmen CloudFare weist beispielsweise auf die Variabilität der Attributreihenfolge in JSON-Antworten (**J**ava**S**cript **O**bject **N**otation) seiner *Hosting Provider API* hin[11] [12]:

> [...] The precise order of the fields in the JSON responses, as described in the examples in this document, may change and should not be relied upon. [...]

Eine andere Variante ist in der API-Spezifikation des Rahmenwerks *Eclipse Plug-in Development Environment* zu finden. Über spezielle Annotationen können Klassen und Schnittstellen mit Beschränkungen versehen werden, z.B. als nicht durch Benutzer zu implementieren (bei Schnittstellen) oder nicht durch Benutzer direkt zu instanziieren (bei

[11] CloudFare Inc. (2012), Abschnitt "1.2 - General Issues Checklist".
[12] Für eine detaillierte Beschreibung des Datenformates JSON wird auf Crockford (2006) verwiesen.

Klassen) (vgl. IBM Corp. und andere, ohne Jahr, Abschnitt “Reference” > “Defining API Restrictions”).

Natürlich ist es in jedem Fall möglich, dass die Benutzer sich über die vom Vertrag definierten Beschränkungen hinwegsetzen. Dieses Verhalten kann wiederum zu den von Parnas beschriebenen impliziten (auf Annahmen beruhenden) Kopplungen führen. Hier ist allerdings einzuwerfen, dass auch ohne Vertrag die Möglichkeit zur Beobachtung des Verhaltens der Komponente besteht. Auch die Beobachtungsergebnisse können von Benutzern verwendet werden und damit zu impliziten Kopplungen führen. Unter diesen Rahmenbedingungen und bei derartig vorgehenden Programmierern sind implizite Kopplungen wahrscheinlich unvermeidbar.

Zusammenfassend bleibt festzustellen, dass das Geheimnisprinzip eine wichtige Leitlinie für den Entwurf von Schnittstellen sowie für das Abgrenzen von Modulen darstellt. Seine Abschwächung kann in Einzelfällen dennoch notwendig und sinnvoll sein. Parnas begründet das Geheimnisprinzip mit der Verwertung verfügbaren Wissens durch gute Programmierer. Die von Parnas identifizierte Gefahr impliziter, also auf Annahmen beruhender, Kopplungen von verschiedenen Modulen untereinander resultiert folglich vorrangig aus der Verwendung des Wissens aufgrund von dessen freier Verfügbarkeit. Durch explizite Beschränkung der Verwendungsmöglichkeiten von Vertragsinhalten kann der Anbieter die Übernahme der Verantwortung für derartige Kopplungen vermeiden. Schließlich ist anzunehmen, dass gute Programmierer auch verantwortungsbewusste Programmierer sind. Es besteht keine Notwendigkeit, verantwortungsbewusste Programmierer vor sich selbst zu schützen.

3.2.5 Realisierung und Nutzung

Der bisherige Vertragsbegriff bedeutet vor allem für die Anbieter ein Dilemma. Ein Vertrag entspricht der Spezifikation eines abstrakten Konzeptes. Von dessen Realisierungen wird dabei aufgrund des Geheimnisprinzips weitgehend abstrahiert.[13] Es liegt in der Natur einer abstrakten Spezifikation, dass sie die Gemeinsamkeiten der möglichen Realisierungen beschreibt. Sofern mehrere Realisierungen einer Spezifikation im Wettbewerb stehen, haben die jeweiligen Anbieter ein großes Interesse an der Darstellung der Besonderheiten ihrer Realisierung. Nur so können sie potentielle Benutzer zum Einsatz ihrer Komponente

[13] Eine Ausnahme bilden hier die im Vertrag aufgeführten Abhängigkeiten von der Laufzeitumgebung.

animieren. Allerdings droht bei derartigen Darstellungen eine Verletzung des Geheimnisprinzips. Dieses Dilemma führt zur Recherchefrage 2.

Recherchefrage 2 *Gibt es verschiedene Arten von API-Verträgen? Falls ja, welche Arten sind das?*

Cheesman und Daniels schlagen zur Lösung dieses Konfliktes die Unterscheidung zwischen Nutzungsverträgen und Realisierungsverträgen im Kontext von Komponenten vor (siehe Abbildung 3.6 [14]). Dieses Konzept lässt sich auf die anderen Ebenen im Systementwurf übertragen (Operation / Methode, Schnittstelle / Klasse, Modul und System) (vgl. Cheesman und Daniels, 2001, Seite 16 - 21):

- **Nutzungsvertrag:** Ein Nutzungsvertrag wird zwischen dem Benutzer und einer Konzeptspezifikation geschlossen. Er beschreibt abstrakte Beziehungen und Eigenschaften. Sein Fokus liegt folglich auf den Gemeinsamkeiten verschiedener Realisierungsmöglichkeiten.

- **Realisierungsvertrag:** Ein Realisierungsvertrag wird zwischen dem Anbieter und der Konzeptspezifkation selbst geschlossen. Er beschreibt konkrete Beziehungen und Eigenschaften. Der Realisierungsvertrag erweitert immer mindestens einen Nutzungsvertrag. Er legt die Charakteristika der jeweiligen Realisierung dar.

Ein Beispiel für einen Nutzungsvertrag stellt die Beschreibung der Schnittstelle *java.util.List* des *Java Collections Framework* dar.[15] Dieser Vertrag erläutert das Konzept einer Liste. Eine Liste ist demnach eine geordnete Sammlung. Der Benutzer kann ein Element an einer bestimmten Position in die Liste einfügen. Er hat außerdem die Möglichkeit, Elemente der Liste über ihren Positionsindex zu lesen und nach bestimmten Elementen in der Liste zu suchen. Eine Liste erlaubt Duplikate (vgl. Oracle Corp., 2013a, Vertrag der Schnittstelle *java.util.List*).

Das *Java Collections Framework* bietet u.a. die folgenden zwei Klassen als Implementierungen einer Liste: *java.util.ArrayList* und *java.util.LinkedList*. Bereits die Bezeichner

[14] Die hier dargestellte Abbildung 3.6 basiert auf der Abbildung in Cheesman und Daniels (2001), Seite 18.

[15] Für detaillierte Informationen zum Java Collections Framework wird auf dessen Homepage `http://docs.oracle.com/javase/7/docs/technotes/guides/collections/index.html` verwiesen (letzter Abruf: 16.04.2013).

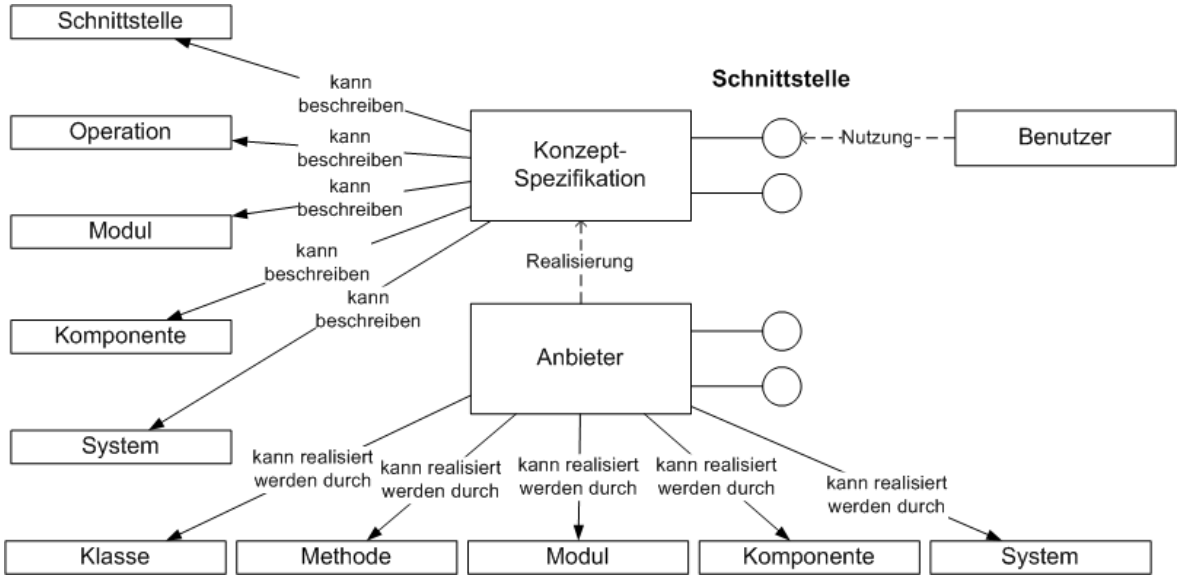


Abbildung 3.6: Schematische Darstellung der Nutzungs- und Realisierungsbeziehung zwischen Benutzer und Anbieter

beider Klassen geben Aufschluss über ihre jeweilige Realisierung des Konzeptes einer Liste. Die *ArrayList* basiert auf einem Feld, dessen Größe sich je nach Bedarf verändert. Das Lesen bzw. Hinzufügen von Elementen oder die Berechnung der Größe erfolgt in konstanter Zeit (vgl. Oracle Corp., 2013a, Vertrag der Klasse *java.util.ArrayList*). Es ist deutlich erkennbar, dass dieser Vertrag nicht zur Erläuterung des Konzeptes einer Liste beiträgt. Vielmehr charakterisiert er detailliert eine bestimmmte Realisierung einer Liste, welche in diesem Fall auf einem Feld basiert. Des Weiteren gibt der Vertrag nicht-funktionale Zusicherungen bzgl. der Laufzeitkomplexität einzelner Operationen. Auch der Vertrag der *java.util.LinkedList* enthält einzelne Details der zugehörigen Realisierung. Diese Implementierung basiert auf einer doppelt-verketteten Liste. Zur Optimierung verwenden deren Methoden jeweils den Start oder das Ende der Liste als Ausgangspunkt - je nachdem, was dichter an der übergebenen Position liegt (vgl. Oracle Corp., 2013a, Vertrag der Schnittstelle *java.util.LinkedList*). Auch dieser Vertrag erläutert nicht das Konzept einer Liste. Er kennzeichnet die konkrete Implementierung als doppelt verkettete Liste mit den bekannten Eigenschaften. Außerdem nennt er eine interne Optimierung dieser Implementierung. Sie wählt in Abhängigkeit vom übergebenen Index die Start- oder die Ende-Referenz für die Ausführung der jeweiligen Operation aus.

Beide letztgenannten Verträge stammen aus der Feder eines Anbieters. Verstoßen sie gegen das Geheimnisprinzip? Streng genommen ja. Schließlich werden Implementierungs-

details herausgegeben, die nicht zwingend zur korrekten Nutzung der Realisierung nötig sind. Diese ist bereits durch den zugehörigen Nutzungsvertrag hinreichend dargelegt worden. Aber Realisierungsverträge dienen gemäß ihrer Definition nicht der Erläuterung eines Konzeptes, sondern der Charakterisierung einer Realisierung. Auf Basis dieser Charakterisierung hat der Benutzer die Möglichkeit, unter mehreren möglichen Realisierungen für seinen Anwendungsfall die am besten passende auszuwählen. Daher kann das Geheimnisprinzip zu diesem Zweck bei Realisierungsverträgen etwas abgeschwächt werden, sodass derlei Abgrenzungen möglich sind. Auch in diesem Fall steht der Anbieter in einem Konflikt zu sich selbst. Er strebt natürlicherweise nach der größtmöglichen Freiheit, Entwurfsentscheidungen zu einem späteren Zeitpunkt ändern zu können. Gleichzeitig muss er bestimmte Charakteristika seiner Realisierung preisgeben, um Benutzer zu werben. Letztendlich muss diese Frage nach gründlicher fallspezifischer Abwägung entschieden werden.

3.2.6 Abgrenzung zur API-Dokumentation

Gemäß den vorherigen Abschnitte definieren Nutzungs- und Realisierungsverträge, welche Leistung der Benutzer erhält und welche Eigenleistung in Form von Bereitstellung von Parametern und Ressourcen er erbringen muss. Sie spezifizieren die Operation. Sehr eng mit dem Begriff der Spezifikation geht die Dokumentation einher. Nach Clements et al. wird eine Schnittstelle durch ihre Spezifikation dokumentiert (vgl. Clements et al., 2003, Seite 226). Dennoch gibt es Inhalte, die eher einem erläuternden Handbuch als einem rechtsverbindlichen Vertrag zuzuordnen sind. Dazu gehören Richtlinien zur Nutzung eines API und entsprechende (Quelltext-)Beispiele ebenso wie eine Begründung für den gewählten Entwurf (vgl. Clements et al., 2003, Seite 232 f.).

Kramer argumentiert, dass diese Inhalte den Umfang der Verträge stark vergrößerten. Somit besteht die Gefahr, dass derartige Inhalte vom eigentlichen Kern des Vertrages ablenken (vgl. Kramer, 1999, Seite 147 f.). Außerdem charakterisieren sie weder die erbrachte Leistung noch definieren sie Strukturen zwecks zukünftiger Planungssicherheit. Im Kontext dieser Arbeit werden derartige Inhalte unter dem Begriff API-Dokumentation zusammengefasst. Eine schematische Übersicht der Bestandteile einer Komponente gibt die Abbildung 3.7. Im Folgenden werden ausschließlich API-Verträge und keine API-Dokumentationen behandelt. Auch diese Abgrenzung wird mit dem Untersuchungsziel eines vervollständigten Vertrages und damit einer qualitativ höherwertigen Spezifikation begründet.

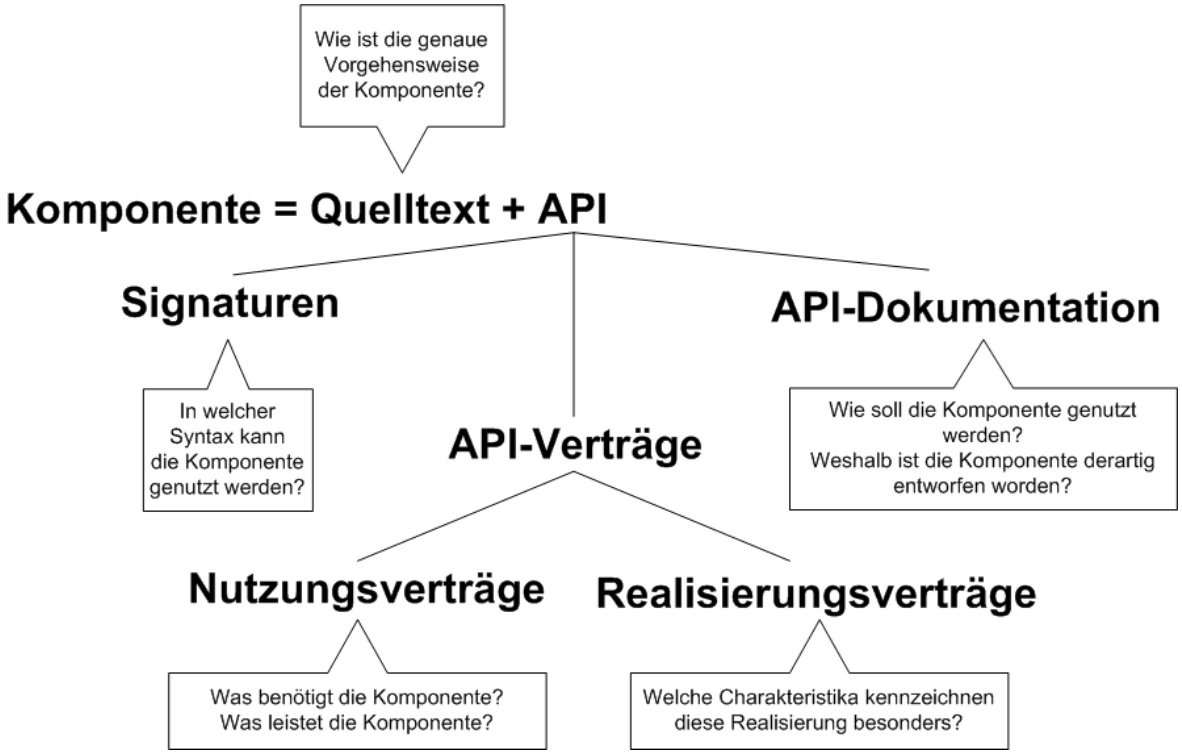


Abbildung 3.7: Schematische Übersicht der Bestandteile einer Komponente

3.3 Ausprägungen von API-Verträgen

Der vorangegangene Abschnitt hat drei wichtige Prinzipien zur Auswahl von Inhalten eines API-Vertrages dargestellt und diese diskutiert (das Geheimnisprinzip, das Prinzip von Zusicherungen und Erwartungen sowie die Unterscheidung zwischen Nutzung und Realisierung). Offen ist bisher der konkrete Aufbau von API-Verträgen geblieben. Diesen Aspekt adressiert die Recherchefrage 3:

Recherchefrage 3 *In welcher Form werden an welchem Ort welche Inhalte der API-Verträge hinterlegt?*

Zuerst werden die möglichen Formen von API-Verträgen betrachtet. Auf Basis der publizierten Arbeiten können die folgenden vier Grundformen für Inhalte von API-Verträgen identifiziert werden:

- **Formal:** Spezifikation in einer wohl definierten und maschinell interpretierbaren Form, z.B. Vor- und Nachbedingungen in *Eiffel* (vgl. Meyer, 1992, Seite 5 - 11) oder in der *Object Constraint Language* (OCL) (vgl. The Object Management Group Inc., 2012, Seite 8 und Seite 179 f.).

- **Natürlichsprachlich:** Spezifikation in formatfreier, natürlicher Sprache, z.B. die Zusammenfassung der von einer Methode angebotenen Leistung in einem *javadoc*-Methodenkommentar (vgl. Smith und Kramer, 2003, Abschnitt "Method Specification").

- **Grafisch:** Spezifikation mittels definierter, grafischer Elemente, z.B. durch ein UML-Zustandsdiagramm [16]

- **Hybrid:** Spezifikation, die Inhalte von mindestens zwei anderen Grundformen beinhaltet, z.B. ein Methodenkommentar in *javadoc* enthält eine formale Struktur (mittels Tags wie @param, @return etc.) und natürlichsprachliche Erläuterungen (z.B. Bedeutung eines Formalparameters).

Aber wo sollen die Vertragsinhalte in den beschriebenen Formen hinterlegt werden? Gemäß dem Ansatz von Brooks müssten die API-Verträge in Dokumentform als Teil des Project Workbook organisiert werden. Knuth hingegen schlägt vor, das ausführbare Programm und die es betreffende Dokumentation aus derselben Quelle zu erzeugen.[17] Auf diese Weise ist die Konsistenz zwischen Programm und Dokumentation einfacher zu gewährleisten. Zur Demonstration seines Ansatzes entwickelte Knuth das Programm *WEB*. WEB verarbeitet nicht nur API-Verträge und -Dokumentation, sondern auch innerhalb des Programms befindliche Kommentare. Knuth unterscheidet bei der Verarbeitung der Quelle zwei verschiedene Prozesse: weaving und tangling. Weaving bezeichnet die Erzeugung eines Dokuments, welches alle Kommentare in aufbereiteter Form enthält. Tangling hingegen meint die Erzeugung des ausführbaren Programms (vgl. Knuth, 1992, Seite 101 f.). Knuths Ansatz ist mittlerweile für viele Programmiersprachen in sehr ähnlicher Weise umgesetzt worden (vgl. Meier-Eickhoff, 2011, Seite 29 - 36).[18]

[16] Eine detaillierte Beschreibung der UML-Zustandsdiagramme findet sich in The Object Management Group Inc. (2011b), Seite 535 - 596. An dieser Stelle sind ausschließlich die in The Object Management Group Inc. (2011b), Seite 593 - 595 spezifizierten grafischen Elemente des Zustandsdiagramms gemeint.

[17] Obwohl Knuth explizit von Dokumentation spricht, wird sein Konzept für diese Arbeit ebenfalls auf API-Verträge übertragen.

[18] Die Bachelor-Thesis von Dirk Meier-Eickhoff ist im Rahmen dieses Dissertationsprojektes betreut worden.

```
/**
 * Returns the blue component in the range 0-255
 * in the default sRGB space.
 *
 * @return the blue component.
 * @see getRGB()
 */
public int getBlue();
```

Listing 3.2: Beispiel für einen javadoc-Vertrag auf Operationsebene

Friendly und Kramer beschreiben mit *javadoc* ein ähnliches Werkzeug, welches im Java-Umfeld verbreitet ist.[19] *javadoc* identifiziert die API-Verträge als speziell ausgezeichnete Kommentare direkt im Java-Quelltext (siehe das Beispiel in Listing 3.2[20]). Im Unterschied zu Knuths *WEB*-System verarbeitet *javadoc* keine Kommentare aus dem Methodenrumpf, sondern nur solche aus dem Methodenkopf. Das Werkzeug erzeugt aus den hinterlegten Kommentaren ein Hypertext-Dokument im HTML-Format (vgl. Friendly, 1995, Seite 152).

Bisher sind größtenteils sehr allgemeine Prinzipien zur Auswahl von Vertragsinhalten diskutiert worden. Die Angabe konkreter Inhalte gestaltet sich für Außenstehende oft schwierig, weil dafür detaillierte Kenntnisse des jeweiligen Systems und seines fachlichen Kontextes notwendig sind. Trotzdem haben Smith und Kramer für die Erstellung von API-Verträgen mit *javadoc* eine Reihe detaillierter inhaltlicher Anforderungen veröffentlicht.[21] In dieser Übersicht findet sich allerdings keine Unterscheidung zwischen Nutzungs- und Realisierungsverträgen. Aus diesem Grund sind die *javadoc*-Anforderungen in Tabelle 3.1 mit entsprechender Zuordnung zu den beiden Vertragsarten dargestellt.

Besonders erwähnenswert erscheint die geforderte Beschreibung des Algorithmus, weil dadurch eine Verletzung des Geheimnisprinzips sehr wahrscheinlich ist. Aus dem von Smith und Kramer gegebenen Beispiel der Berechnung eines Hashwertes kann entnommen werden, dass die detaillierte Darlegung des Algorithmus zur exakten Beschreibung des

[19] Für eine ausführliche Beschreibung von *javadoc* wird auf die Arbeiten von Friendly (Friendly (1995)) und Kramer (Kramer (1999)) verwiesen.

[20] Oracle Corp. (2013a), Vertrag der Methode *getBlue()* der Klasse *java.awt.Color*.

[21] Für eine detaillierte Darstellung dieser Anforderungen wird auf Smith und Kramer (2003) verwiesen.

Ergebnisses dient. Dies ist besonders bei mathematischen Formeln einleuchtend, die mit natürlicher Sprache oft nur umständlich zu beschreiben sind.

Ein weiterer, sehr interessanter Aspekt ist die Angabe von erlaubten Verhaltensunterschieden zwischen verschiedenen Implementierungen desselben Nutzungsvertrages. Als Beispiel verweisen Smith und Kramer auf den Vertrag der Methode *write()* der Java-Klasse *java.io.FileOutputStream*. Hier wird darauf hingewiesen, dass sowohl die Java Virtual Machine als auch das Betriebssystem die geschriebenen Daten in einem Puffer speichern können. Je nach Implementierung kann es dazu kommen, dass Ausnahmen für Schreibfehler erst nach Aufruf der Methoden *flush* oder *close* geworfen werden. Diese Freiheitsgrade gelten für alle Implementierungen und sind daher dem gemeinsamen Nutzungsvertrag zuzuordnen. Clements et al. weisen im Kontext einer Schnittstelle zusätzlich auf die Definition von Konfigurationsparametern und ihre Bedeutung hin (vgl. Clements et al., 2003, Seite 232). Derartige Angaben gehören in einen Realisierungsvertrag.

3.4 Gewährleistung von Vollständigkeit

Die vorangegangenen Abschnitte haben den aktuellen Forschungsstand in Bezug auf Entwurfsprinzipien, Form, Ablageort und die Inhalte von API-Verträgen dargestellt. Basierend auf den beschriebenen Erkenntnissen untersucht dieser Abschnitt, wie derzeit Stille in API-Verträgen vermieden wird:

Recherchefrage 4 *Welche Ansätze zur Vermeidung von Stille existieren derzeit?*

3.4.1 Prüflistenbasierte Verfahren

Als erste Kategorie von Ansätzen lassen sich die prüflistenbasierten Verfahren identifizieren. Nach Grochla et al. ist eine Prüfliste ein Katalog von Fragen zum Erkennen, Aufnehmen und Analysieren von Schwachstellen. Eine Prüfliste dient dem Abgleich des IST-Zustandes mit dem SOLL-Zustand des Untersuchungsgegenstandes. Außerdem hilft eine Prüfliste bei der Bewertung der identifizierten Abweichungen und gibt erste Anhaltspunkte für Maßnahmen zur Verbesserung des IST-Zustandes (vgl. Grochla et al., 1986, Seite 25). Unter einer Schwachstelle ist in diesem Zusammenhang Stille im API-Vertrag zu verstehen. Im Kontext der Vollständigkeitsprüfung eines API-Vertrages enthält die Prüfliste die Inhalte, die im geprüften API-Vertrag erwartet werden. Zusätzlich kann die

	Nutzungsvertrag					**Realisierungsvertrag**				
	S	K	M	Kl/S	O/M	S	K	M	Kl/S	O/M
Konkretisierung von Vorbedingungen										
Gültiger Wertbereich von Formalparametern					X					X
Verhalten bei leerer Belegung von Formalparametern (z.B. `null`)					X					X
Abhängigkeiten vom Betriebssystem bzw. zur Laufzeitumgebung)						X	X	X	X	X
Sicherheitsbeschränkungen				X	X				X	X
Referenzen auf ext. Spezifikationen	X	X	X	X		X	X	X	X	
Konkretisierung von Nachbedingungen										
Erwartetes Verhalten					X					X
Zustandsänderungen					X					X
Referenzen auf ext. Spezifikationen	X	X	X	X		X	X	X	X	
Sonstige Charakteristika										
Angewandter Algorithmus										X
Zulässige Unterschiede im Verhalten zwischen Implementierungen					X					
Beschreibung des abgebildeten Modellaspektes	X	X	X	X	X	X	X	X	X	X
Mögliche Zustände und Zustandsübergänge				X					X	
Ursachen zu erwartender Ausnahmen					X					X

Tabelle 3.1: Inhaltliche Anforderungen an mit javadoc erstellte API-Verträge (Legende: S = System, K = Komponente, M = Modul, Kl / S = Klasse / Schnittstelle, O / M = Operation / Methode)

Prüfliste auch die Strukturierung der Vertragsinhalte vorgeben. Der Autor geht das Schema der Prüfliste sukzessive durch und füllt es gemäß dem zu spezifizierenden System, der Komponente, dem Modul, der Klasse bzw. Schnittstelle oder der Operation bzw. Methode aus.

Prüflistenbasierte Verfahren geben dem Autoren folglich ein Raster im Sinne eines Blanko-Formulars vor, welches dieser zu vervollständigen hat. Clements et al. beschreiben ein derartiges Formular zur Dokumentation von Schnittstellen sehr detailliert. Dieses ermöglicht die Beschreibung aller von der Schnittstelle bereitgestellten Operationen inkl. ihrer Datentypen und der jeweiligen Bedeutungen. Besondere Relevanz hat die Darstellung eventueller Nutzungsbeschränkungen, vorhandener Konfigurationsparameter, Abhängigkeiten von der Laufzeitumgebung und möglicher Fehlersituationen (vgl. Clements et al., 2003, Seite 228 - 233). Das Kapitel A des Anhangs enthält die von Clements et al. vorgeschlagene Prüfliste, konvertiert in die von Grochla et al. angegebene Struktur (vgl. Grochla et al., 1986, Seite 29). Analog können auch die von *javadoc* geforderten und in Tabelle 3.1 dargestellten Inhalte in eine ähnlich geartete Prüfliste konvertiert werden.

3.4.2 Quelltextbasierte Verfahren

Als zweite Kategorie können die quelltextbasierten Verfahren abgegrenzt werden. Einige Prüfpunkte der oben vorgestellten Prüflisten haben einen dynamischen Charakter. Ihre genaue Anzahl und Ausprägung ist abhängig vom zum API-Vertrag gehörigen Programm. Als Beispiele sind hierfür die Liste von deklarierten Formalparametern, von Ausnahmen und deren Bedeutung oder die Liste der vom Programm bereitgestellten Konfigurationsoptionen zu nennen. Quelltextbasierte Verfahren extrahieren aus dem Quelltext die konkreten Ausprägungen dieser Prüfpunkte, z.B. eine Liste aller akzeptierten Konfigurationsoptionen einer Operation.

Forward und Lethbridge haben beobachtet, dass die Dokumentation vieler Programme meist nicht mehr dem aktuellen Entwicklungsstand des Quelltextes entspricht (vgl. Forward und Lethbridge, 2002, Seite 29). Rabkin und Katz argumentieren, dass dies u.a. vom Programm akzeptierte Optionen zur Konfiguration betrifft. Sie zeigen, dass die Dokumentationen von Konfigurationsoptionen in den von ihnen untersuchten Opensource-Projekten viele Inkonsistenzen aufweisen. Beispielsweise sind genutzte Optionen gar nicht dokumentiert oder dokumentierte Optionen werden nicht genutzt (vgl. Rabkin und Katz, 2011, Seite 139). Rabkin und Katz stellen einen Algorithmus vor, der eine Näherung für die Menge der existierenden Konfigurationsoptionen und deren jeweiligen Wertebereich

berechnet. Auf diese Weise kann der Vertragsautor bei der Erstellung der Dokumentation und beim Finden von Fehlern unterstützt werden (vgl. Rabkin und Katz, 2011, Seite 136).

Buse und Weimer weisen darauf hin, dass dieselbe Problematik auch bei der Dokumentation von Bedingungen besteht, die zum Werfen von Ausnahmen führen. Sie schlagen einen Algorithmus vor, mit dem natürlichsprachliche Beschreibungen dieser Bedingungen generiert werden können (vgl. Buse und Weimer, 2008, Seite 273 - 276). Sridhara et al. geben einen Algorithmus an, der zusammenfassende natürlichsprachliche Beschreibungen zu den ausgeführten Instruktionen einer Java-Methode generiert (vgl. Sridhara et al., 2011, Seite 101 f.).

Alle bisher vorgestellten quelltextbasierten Verfahren analysieren den Quelltext der geprüften Methoden. Das von Burn et al. entwickelte Programm *Checkstyle* kontrolliert neben der Einhaltung sprachlicher Konventionen die Vollständigkeit des zugehörigen *javadoc*-Vertrags ausschließlich auf Basis der Operationssignatur. *Checkstyle* prüft u.a., ob zu jedem deklarierten Formalparameter, jeder deklarierten Ausnahme und dem Resultat (falls vorhanden) eine Beschreibung im Vertrag enthalten ist (vgl. Burn et al., (ohne Jahr, Abschnitt “Documentation”, “Standard Checks”, “Javadoc Comments”).

3.4.3 Diskussion

In diesem Abschnitt sind zwei Kategorien von Ansätzen zur Vermeidung von Stille in API-Verträgen vorgestellt worden: prüflistenbasierte Verfahren und quelltextbasierte Verfahren.

Die prüflistenbasierten Verfahren können sehr einfach und ohne komplexe technische Hilfsmittel angewendet werden (die Existenz der Prüfliste vorausgesetzt). Sie sind außerdem sehr universell für jede inhaltliche Anforderung an einen Vertrag einsetzbar. Da prüflistenbasierte Verfahren nicht den Zugriff auf Quelltexte voraussetzen, sind sie für den Einsatz in allen drei vorgestellten Szenarien geeignet. In manchen Fällen sind zum vollständigen Ausfüllen einer Prüfliste jedoch besondere Kenntnisse der Implementierung notwendig. Das ist beispielsweise bei der Prüfung der aufgeführten Konfigurationsoptionen in einem Realisierungsvertrag der Fall. Dies mag auf den ersten Blick als Nachteil erscheinen. Aber selbst in einem solchen Fall macht die Prüfliste den Autor auf möglicherweise offene Punkte aufmerksam. Er kann diese Punkte dann mit einem Experten klären und den Vertrag entsprechend gestalten. Somit sind prüflistenbasierte Verfahren in allen drei beschriebenen Szenarien einsetzbar.

Nachteilig an Prüflisten ist, dass sie manuell angewendet werden müssen. Dies kann sehr zeitaufwändig sein. Außerdem ist die konsistente Anwendung besonders bei existierenden Verträgen schwierig zu gewährleisten (siehe die zuvor zitierte Arbeit von Forward und Lethbridge). Des Weiteren sind die Prüflisten oft sehr allgemein gehalten. Daher setzt ihre Anwendung einen hohen Erfahrungsschatz des Autors voraus. Parnas weist zusätzlich auf die Gefahr hin, dass Prüflisten auf Basis von Annahmen über die später damit zu beschreibenden Systeme getroffen werden. Es kann daher passieren, dass diese Annahmen für ein konkretes System nicht zutreffen.[22] Es besteht die Möglichkeit, dass diese Autoren die Abhängigkeiten anderer, vom eigenen Modul abhängiger Module redundant im eigenen Vertrag erfassen. Oder sie definieren einfach irgendwelche plausiblen Abhängigkeiten - nur um den Punkt der Prüfliste auszufüllen (vgl. Parnas, 1971, Seite 341).

Beispielsweise verfügt nicht jedes Java-Programm über Abhängigkeiten vom Betriebssystem, obwohl die *javadoc*-Prüfliste einen solchen Prüfpunkt enthält.[23] Einzelne Autoren könnten sich in dieser Situation dazu gezwungen fühlen, das vorgegebene Schema auszufüllen - auch, wenn keine direkten Abhängigkeiten vom Betriebssystem existieren.

Quelltextbasierte Verfahren liefern aufgrund ihrer Automatisierung in sehr kurzer Zeit konsistente und reproduzierbare Hinweise auf Lücken in API-Verträgen. Damit eignen sie sich insbesondere während der Wartung und Weiterentwicklung einer Komponente zur Gewährleistung der Vollständigkeit der API-Verträge. Sie passen daher gut für den Einsatz im Dienste des Herstellers aus Szenario 1.

Der größte Nachteil quelltextbasierter Verfahren besteht in ihrer Abhängigkeit von der Verfügbarkeit des Quelltextes. Der Auftragnehmer und der Käufer aus den Szenarien 2 und 3 haben allerdings keine Möglichkeit zur Einsicht in die Quelltexte. Damit sind quelltextbasierte Verfahren für sie nicht einsetzbar. Des Weiteren entstehen die Quelltexte meist während der Implementierungsphase eines Projektes. Daher steht das Feedback der quelltextbasierten Verfahren erst ab einem späten Zeitpunkt des Projektes zur Verfügung. Die Verfügbarkeit der Hinweise wäre in diesen Phasen allerdings von sehr großem Vorteil, da Anpassungen eines Programms in späteren Entwicklungsphasen tendenziell mit höheren Kosten verbunden sind als in früheren Phasen (vgl. Boehm, 1981, Seite 39 f.). Nachteilig an diesen Verfahren ist außerdem ihre jeweils sehr hohe Spezialisierung auf eine

[22] Parnas verwendet dafür in Parnas (1971) den Begriff *Documentation Standards*. Dieser Begriff ist hier mit dem Begriff der Prüfliste gleichgesetzt worden.

[23] Für die Forderung der Dokumentation von Abhängigkeiten vom Betriebssystem oder zur Plattform vgl. Smith und Kramer (2003), Abschnitte “Package Specification”, “Class / Interface Specification” oder “Method Specification”.

bestimmte Art von Charakteristika. Es ist derzeit kein Algorithmus bekannt, der beispielsweise zur Überprüfung der im Vertrag genannten Konfigurationsoptionen und gleichzeitig zur Überprüfung der dort genannten Fehlerfälle eingesetzt werden kann.

Auch der hohe Automatisierungsgrad dieser Verfahren ist mit Skepsis zu beurteilen. Schließlich muss der Autor des Vertrages die Ausgaben dieser Verfahren mit Bedacht prüfen, um seine Interessen bezüglich des Geheimnisprinzips zu wahren. Rabkin und Katz beschreiben beispielsweise, dass einzelne von ihrem Verfahren entdeckte Konfigurationsoptionen scheinbar nur für den Test der Komponente vorgesehen sind (vgl. Rabkin und Katz, 2011, Seite 135 f.). Es ist daher unwahrscheinlich, dass der Hersteller der betrachteten Komponente seinen Kunden die Verfügbarkeit dieser Optionen vertraglich garantieren möchte. Die vertragliche Zusicherung der Ausgaben quelltextbasierter Verfahren muss im Einzelfall genau abgewogen werden.

Abschließend ist noch ein weiterer offener Punkt bei beiden Kategorien von Verfahren festzustellen: die fehlende Unterstützung bei der Integration von fachlichen Bezügen in den Vertrag. Smith und Kramer fordern für Methoden-Spezifikationen in *javadoc* lediglich, dass das erwartete oder gewünschte Verhalten der Operation festgeschrieben ist. Des Weiteren verlangen sie die Angabe verwendeter Algorithmen (siehe Smith und Kramer, 2003, Abschnitt “Method Specification”). Insbesondere die erste Forderung bedarf sehr viel Interpretation bzgl. des Aufbaus und der erwarteten Inhalte einer solchen Beschreibung. Selbstverständlich schließt diese Formulierung fachliche Bezüge z.B. zu umgesetzten Geschäftsregeln oder gesetzlichen Vorgaben nicht aus, aber explizit erwähnt sind sie auch nicht. Ähnlich verhält es sich mit der geforderten Beschreibung von Algorithmen. Auch hier sind fachliche Bezüge zu beispielsweise Berechnungsvorschriften denkbar, werden aber nicht explizit erwähnt. Ein quelltextbasiertes Verfahren, welches derartige Bezüge aufdeckt, konnte gar nicht recherchiert werden. Die fehlenden fachlichen Bezüge überraschen insofern, als ein Programm oft erst dadurch eine für Menschen vorstellbare und vor allem auf Korrektheit verifizierbare Bedeutung erhält.

3.5 Schlussfolgerungen

Das vorangegangene Kapitel hat einen Überblick über den Stand der Forschung zur Gestaltung von API-Verträgen gegeben. Die Tabelle 3.2 gibt die aus der Literatur gewonnenen Erkenntnisse bzgl. der vier aufgeworfenen Recherchefragen kurz zusammengefasst wieder.

Insgesamt lässt sich sagen, dass in der Literatur ein breiter Konsens bzgl. der Inhalte eines Vertrages besteht: gegenseitige Zusicherungen der Vertragspartner (Benutzer und Anbieter), z.B. bzgl. Wahrung zulässiger Wertebereiche von Formalparametern und Resultaten oder Bereitstellung benötigter Ressourcen etc. Auch die Auffassungen über das Vorgehen bei der Erstellung bzw. der Bearbeitung eines Vertrages sind sehr ähnlich. Alle vorgestellten Arbeiten liefern dem Autor des Vertrages ein, teilweise vorausgefülltes, Blanko-Formular. Dieses Formular fordert beispielsweise die Benennung der deklarierten Formalparameter, existierender Vor- und Nachbedingungen, möglicher Ausnahmen oder benötigter Ressourcen. Es basiert entweder auf Prüflisten oder auf Quelltextanalysen. Der Autor hat die Aufgabe, Interessenkonflikte zwischen den Vertragsparteien zu verhandeln und die Verhandlungsergebnisse anschließend durch Ausfüllen bzw. Ergänzung des Formulars schriftlich zu fixieren.

Sehr spärlich ist die Unterstützung bzw. Anleitung des Autoren bei der konkreten Zusammensetzung der operationsabhängigen Teile dieses Formulars - sowohl in technischer als auch in fachlicher Hinsicht. Einzelne Vorschläge zu technischen Angaben können auf Basis der Operationssignatur erzeugt werden. Damit fordert das Formular, falls jeweils vorhanden, Angaben zu Formalparametern, dem Resultat oder deklarierten Ausnahmen. Ohne Zugriff auf den Quelltext der Implementierung bleibt dem Autoren weitere Unterstützung beim Aufbau des Vertrages verwehrt, z.B. dem Auftragnehmer aus Szenario 2 oder dem Käufer aus Szenario 3. Aber dieses Problem betrifft auch den Hersteller aus Szenario 1, sobald dieser Nutzungsverträge z.B. für Schnittstellen oder Operationen erstellen möchte. Auch in diesen Fällen besteht nicht die Möglichkeit, den Quelltext einer Implementierung zwecks Analyse heranzuziehen.

Daher wird die Ableitung eines Formulars ohne Zugang zum Quelltext von Methoden als Forschungslücke identifiziert, welches neben technischen auch fachliche Vertragsinhalte zur Operation abdeckt. Ein derartiges Verfahren wäre sowohl für den Hersteller, den Auftragnehmer als auch den Käufer aus den vorgestellten Szenarien nutzbar.

Recherchefrage 1: Welche Inhalte gehören in einen API-Vertrag?
Die Verträge sind so knapp wie möglich zu halten. Jeder Vertragsinhalt muss zur Spezifikation der zu erbringenden Leistung unerlässlich sein (Geheimnisprinzip). Die Leistung wird durch Vorbedingungen und Nachbedingungen spezifiziert. Bei Realisierungsverträgen kann das Geheimnisprinzip abgeschwächt werden (siehe Antwort auf Recherchefrage 2). In diesem Fall ist die zulässige Verwendung der zusätzlich preisgegebenen Angaben explizit zu reglementieren. Verhandlungsgegenstände sind konkret z.B. die Bereitstellung von Ressourcen oder die Einhaltung von Wertebereichsbeschränkungen.
Recherchefrage 2: Gibt es verschiedene Arten von API-Verträgen? Falls ja, welche Arten sind das?
Unterschieden werden Nutzungs- und Realisierungsverträge. Ein Nutzungsvertrag sichert dem Benutzer Leistungen nach einem abstrakten Konzept zu. Mittels eines Realisierungsvertrages garantiert ein Anbieter eine konkrete Implementierung eines Nutzungsvertrages. Dabei können auch wichtige Details der Implementierung beschrieben werden, welche sie besonders kennzeichnen (Abschwächung des Geheimnisprinzips).
Recherchefrage 3: In welcher Form werden an welchem Ort welche Inhalte der API-Verträge hinterlegt?
API-Verträge bestehen meist aus einer Mischung von formalen, natürlichsprachlichen oder grafischen Inhalten. Ihre Bestandteile stehen direkt im Quelltext und können programmatisch zu einem Dokument zusammengesetzt werden. Die Tabelle 3.1 gibt eine Übersicht über konkrete Vorschläge für Vertragsinhalte.
Recherchefrage 4: Welche Ansätze zur Vermeidung von Stille existieren derzeit?
Zu unterscheiden sind prüflistenbasierte und quelltextbasierte Verfahren. Erstere sind sehr universell einsetzbar und haben daher einen sehr allgemeinen Charakter. Sie müssen aber manuell angewendet werden und bieten ein hohes Risiko von Inkonsistenzen im API-Vertrag. Letztere sind sehr zuverlässig, aber gleichzeitig hochgradig spezialisiert. Sie benötigen zudem Zugriff auf den Quelltext.

Tabelle 3.2: Beantwortung der Recherchefragen des Kapitels 3

Kapitel 4

Identifikation von Stille

Die im vorherigen Kapitel dargelegte Recherche hat ergeben, dass der Ausgangspunkt für die Vervollständigung des API-Vertrages oft die Signatur der jeweiligen Operation in Kombination mit einer Prüfliste ist. So wird systematisch und ohne zusätzliche Analysen von Quelltexten oder anderen Spezifikationsdokumenten die Vollständigkeit eines Vertrages überprüft. Neben der jeweils eingesetzten Programmiersprache findet natürliche Sprache in Spezifikationen auf eingeschränkte Weise Verwendung und zwar in Form von meist fachlichen Bezeichnern.

Brooks vermutet, dass diese Bezeichner ein Schlüssel zum Verständnis eines Programms sind (vgl. Brooks, 1983, Seite 551 f.). Gellenbeck und Cook bestätigen diese Vermutung bzgl. der Bezeichner von Prozeduren und Variablen (vgl. Gellenbeck und Cook, 1991, Seite 74). Auch mehrere praktische Ratgeber für Programmierer stellen die besondere Rolle von Bezeichnern und die erforderliche Sorgfalt bei ihrer Auswahl heraus.[1] Aus diesen Beobachtungen resultiert die Forschungsfrage 1:

Forschungsfrage 1 *Können Bezeichner auch bei der Vollständigkeitsprüfung eines API-Vertrages helfen?*

4.1 Valenz eines Operationsbezeichners

Seit Langem besteht in der Linguistik ein weitgehender Konsens darüber, dass das Verb bzw. das Prädikat ein wichtiger Schlüssel zum Verständnis der Struktur und der Bedeu-

[1] Siehe dazu z.B. Hunt und Thomas (2003), Seite 234 - 236 oder Ottinger (2009), Seite 17 f.

tung eines Satzes ist.[2] [3] Sätze bestehen danach aus zusammengehörigen Gruppen von Wörtern. Eine solche Wortgruppe bildet eine strukturelle Einheit und wird als Phrase oder Konstituente bezeichnet (vgl. Müller, 2008, Seite 3).

Eine Phrase besteht immer aus einem Kopf und einem Rest. Der Kopf ist dasjenige Wort, das die wichtigsten Eigenschaften wie Aufbau oder Anwesenheit bestimmter anderer Elemente bestimmt (vgl. Müller, 2008, Seite 10). Phrasen können innerhalb des Satzes verschoben oder (unter bestimmten Bedingungen) ausgelassen werden, ohne dass der Satz seine grammatikalische Gültigkeit verliert (vgl. Müller, 2008, Seite 3 - 5). Aus der Wortart des Kopfes leitet sich der Typ der Phrase ab, z.B. wird eine Phrase mit einem Verb als Kopf Verbalphrase genannt (vgl. Kürschner, 2008, Seite 158 - 169). Auch Verben können daher in der Rolle des Phrasenkopfes die weitere Struktur des Rests bestimmen. Sie fordern andere Phrasen an bestimmten Stellen bzw. mit bestimmten grammatischen Eigenschaften im Satz.[4] Das deutsche Verb *suchen* fordert das gesuchte Objekt im Akkusativ, z.B. in Satzbeispiel 1.

(1) Suche den Datensatz mit der Nummer in der Kartei.

Diese Eigenschaft wird als Valenz bezeichnet (vgl. Kürschner, 2008, Seite 81). Sie begründet hierarchische Abhängigkeiten innerhalb des Satzes. Die Präpositionalphrase *mit der Nummer* hängt beispielsweise vom Substantiv *Datensatz* ab. In der Linguistik existieren zwei verschiedene Kategorien von Grammatiken zur Abbildung dieser hierarchischen Struktur: Konstituenten- und Dependenzgrammatiken. Beide Kategorien lassen sich anhand der Kenntnis des Phrasenbegriffs voneinander abgrenzen. In Konstituentengrammatiken können auch Phrasen, also zusammengehörige Wörter, einen Knoten bilden (vgl. Müller, 2008, S. 15 - 20). In einer Dependenzgrammatik hingegen repräsentiert jeder Knoten genau ein Wort (vgl. Tesnière, 1980, S. 28).[5]

Zur Veranschaulichung dieser hierachischen Struktur wird das Satzbeispiel einmal als Konstituentenstruktur (siehe Abbildung 4.1a) und einmal als Dependenzstruktur (siehe

[2] Vgl. Tesnière (1980), S. 29 und Chomsky (1986), Seite 42 f. Trotz des angesprochenen Konsenses existiert eine Reihe verschiedener Grammatikmodelle. Eine prägnante Übersicht einschl. Angaben zu weiterführender Literatur findet sich in Müller (2008), Seite 14 f.

[3] Eine prägnante Übersicht zu frühen Studien zur Rolle des Verbs in natürlichen Sprachen findet sich in Foucault (1971), Seite 131 - 136.

[4] Vgl. Kürschner (2008), Seite 81 und Tesnière (1980), Seite 161

[5] Tesnière verwendet den Begriff *Term* anstelle des Begriffs *Wort*.

Abbildung 4.1b) dargestellt. Die Bedeutungen der Kantenbeschriftungen der Dependenzdarstellung sind in Tabelle 4.1 angegeben.[6]

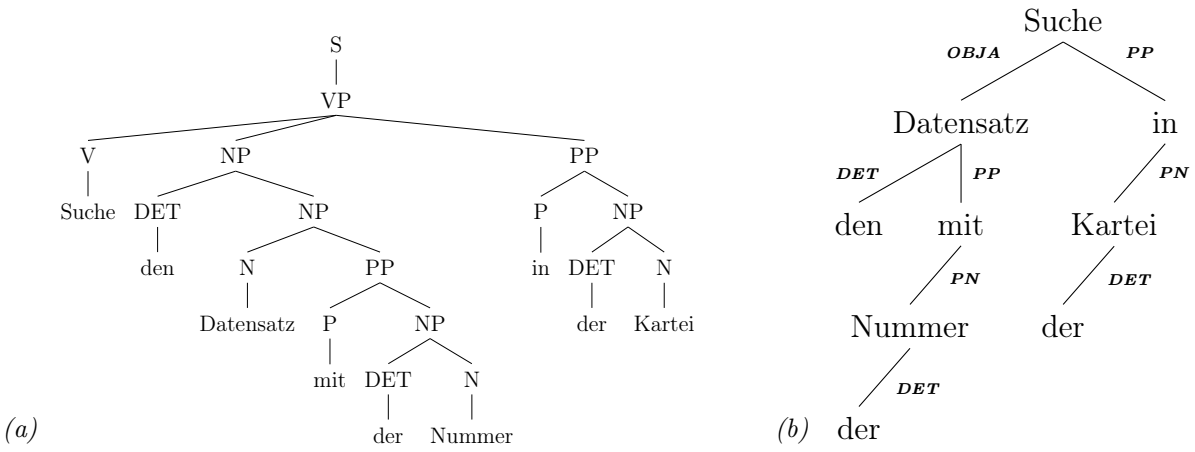


Abbildung 4.1: Konstituentenstruktur (a) und Dependenzstruktur (b)

Kanten- bzw. Knotenbeschriftungen:	**Bedeutung:**
S	Satz
VP	Verbalphrase
V	Verb
NP	Nominalphrase
N	Nomen
PP	Präpositionalphrase
P	Präposition
OBJA	Akkusativobjekt
DET	Determiner eines Nomens (meist bestimmte oder unbestimmte Artikel sowie alle Arten attributiver Pronomen)
PP	Präpositionalphrase
PN	Komplement einer Präposition oder eines Nomens (meist Nomen oder Pronomen)

Tabelle 4.1: Bedeutungen der Kantenbeschriftungen

[6] Die angegebenen Beschriftungen der Dependenzdarstellung sind im Handbuch zum am Arbeitsbereich *Natürlichsprachliche Systeme* des Department Informatik der Universität Hamburg entwickelten Constraint Dependency Grammar-System (CDG) definiert worden (siehe Foth (2004)).

Eine detaillierte Diskussion der verschiedenen Grammatiktheorien würde den Rahmen dieser Arbeit sprengen. Sie ist aber auch gar nicht notwendig, denn für die Zwecke dieser Arbeit ist die Erkenntnis der hierachischen Abhängigkeiten innerhalb des Satzes völlig hinreichend. Schließlich weisen Operationsbezeichner weit weniger komplexe grammatikalische Strukturen auf als natürlichsprachliche Sätze wie Satzbeispiel 1. Ottinger beispielsweise empfiehlt einfache Verbalphrasen als Bezeichner für Operationen, z.B. `postPayment`, `deletePage` oder `save` (vgl. Ottinger, 2009, Seite 25). Es ist anzunehmen, dass die Eigenschaft der Valenz trotz der einfacheren inneren Struktur auch in derartig gewählten Bezeichnern zu beobachten ist. Im Falle von `deletePage` stellt sich beispielsweise die Frage, wo die *Page* gelöscht wird oder woran die zu löschende *Page* erkannt werden kann.

Nun wäre es möglich, alle durch die Valenz eines Operationsbezeichners geforderten Phrasen in den Bezeichner mit aufzunehmen, z.B. `deletePageWithMoreThan1000WordsFromGivenDocument`. Es ist allerdings anzunehmen, dass derartige Bezeichner die Lesbarkeit des Programms aufgrund ihrer Länge stark beeinträchtigen. Sinnvoll hingegen erscheint es, diese Inhalte im Vertrag der Operation zu hinterlegen. Somit wirkt die Valenz des Bezeichners einer Operation bis in ihren Vertrag. Aus diesem Grund wird der Begriff der Valenz für diese Arbeit wie folgt definiert:

Definition 10 (Valenz eines Operationsbezeichners) *Die Valenz eines Operationsbezeichners bezeichnet seine Eigenschaft, die Beschreibung bestimmter Charakteristika im zugehörigen Vertrag zu fordern.*

Der Bezeichner bildet folglich eine Hilfestellung bei der Ableitung der notwendigen Vertragsinhalte. Diese Erkenntnis führt zu den folgenden zwei neuen Fragestellungen:

1. Wie kann die Valenz eines Operationsbezeichners präzise beschrieben werden?
2. Auf welche Weise ist ihre Kenntnis bei der Konstruktion von API-Verträgen nutzbar?

4.2 Thematische Rollen und Raster

Bei Sätzen, die ähnliche Vorgänge beschreiben, lassen sich generische Muster im inhaltlichen Satzaufbau erkennen. Gruber beschreibt eine vom Verb abgeleitete vorlexikalische Ebene, welche die Interpretation eines Satzes determiniert (vgl. Gruber, 1962, Seite 2 - 5).

Fillmore hat sechs verschiedene Fälle für Phrasen identifiziert, die weniger die Position der Substantive im Satz, sondern vielmehr deren Bedeutung bzw. Rolle im Geschehen ausdrücken. Beispielsweise beschreibt der Lokativ den Ort oder die Richtung einer Handlung. Das Instrumentum hingegen charakterisiert das eingesetzte Mittel einer Handlung (vgl. Fillmore, 1968, Seite 24 f.).

Die von Fillmore identifizierten Fälle sind eine Art semantischer Typ, der in verschiedenen Ausprägungen von Sätzen immer wiederkehrt. Beispielsweise erfordert die Spezifikation der Tätigkeit *suchen* für einen Dritten immer mindestens das gesuchte Objekt und einen Suchraum. Diese durch die Valenz eines Verbs geforderten Phrasen tragen verschiedene Bezeichnungen: Argument (vgl. Löbner, 2002, Seite 101 f.), Ergänzung (vgl. Kürschner, 2008, Seite 81), Theta-Rolle (vgl. Chomsky, 1986, Seite 42) oder thematische Rolle (vgl. Dowty, 1989, Seite 69). Im Folgenden wird der Begriff thematische Rolle von Dowty verwendet.[7]

Eine thematische Rolle tritt entweder obligatorisch (muss zwingend vorhanden sein) oder fakultativ (kann je nach Kontext vorhanden sein) auf (vgl. Kürschner, 2008, Seite 81). In der Linguistik stehen die thematischen Rollen ebenfalls in hierarchischen Beziehungen zueinander (analog zu den Abhängigkeiten der Wörter bzw. Phrasen eines Satzes). Für das bereits bekannte Satzbeispiel 1 sind diese Beziehungen in Abbildung 4.2 dargestellt.[8]

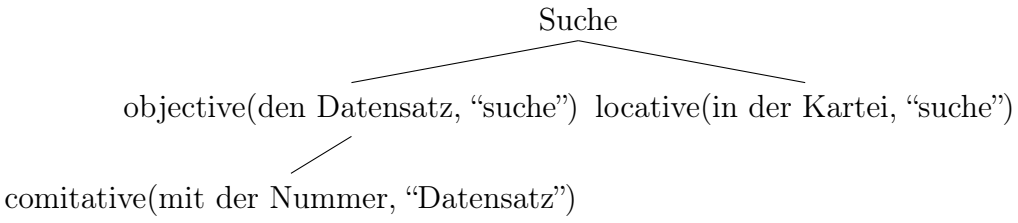


Abbildung 4.2: Hierarchische Beziehungen thematischer Rollen des Beispielsatzes

[7] Vgl. Dowty (1989), Seite 76. Dowty unterscheidet zwischen den individuellen thematischen Rollen eines Verbs und daraus über verschiedene Verben generalisierbaren thematischen Rollentypen. Diese Unterscheidung ist für diese Arbeit nicht notwendig. Daher wird im Folgenden immer nur von thematischen Rollen gesprochen.

[8] Zur Erklärung der gewählten Notation in Abbildung 4.2: Der Name der thematischen Rolle steht jeweils vor der Klammer. Das erste Argument innerhalb der Klasse ist die Phrase, welche die thematische Rolle belegt. Als zweites Argument wird das Wort angegeben, auf welches sich die thematische Rolle bezieht.

Nach Fillmore wird die thematische Rolle AGENTIVE mit derjenigen Phrase belegt, von der die durch das Verb beschriebene Handlung ausgeht (vgl. Fillmore, 1968, Seite 24).[9] Da Satzbeispiel 1 (wie die meisten Operationsbezeichner) die Struktur eines Imperativsatzes hat, ist der AGENTIVE hier nur implizit enthalten (das vom Sprecher adressierte Objekt bzw. die adressierte(n) Person(en)). Vom Verb hängen direkt zwei weitere thematische Rollen ab: der Ort der Handlung (bei Fillmore LOCATIVE genannt) und das Objekt, welches behandelt wird (bei Fillmore OBJECTIVE genannt) (vgl. Fillmore, 1968, Seite 25). Die Phrase *mit der Nummer* bezieht sich hingegen nicht auf das Verb, sondern auf den Datensatz. Die Nummer ist schließlich eines seiner Attribute. Bildlich gesprochen "begleitet" sie "ihren" Datensatz. Diese thematische Rolle wird als COMITATIVE bezeichnet. Stolz et al. detaillieren diese Rolle noch etwas weiter: sie unterscheiden den ACCOMPANEE (derjenige, der begleitet wird, also der Datensatz) und den COMPANION (derjenige, der begleitet, also die Nummer) (vgl. Stolz et al., 2006, Seite 17).

Für den Zweck der Formulierung von API-Verträgen ist es notwendig, das linguistische Konzept einer thematischen Rolle etwas abzuändern. Schließlich sind die durch die thematischen Rollen abgebildeten Angaben im Kontext des API-Entwurfes keine Bestandteile des Operationsbezeichners, sondern des Vertrages. Die notwendigen Veränderungen betreffen die folgenden vier Aspekte:

- Thematische Rollen werden nicht nur mit Phrasen des Bezeichners belegt, sondern auch mit Abschnitten aus dem zur betrachteten Operation gehörenden API-Vertrag.

- Das Verb bzw. Prädikat des Operationsbezeichners wird mit der thematischen Rolle *ACTION* assoziiert.[10] Im API-Vertrag wird diese Rolle durch eine zusammenfassende Beschreibung der Operation belegt.

- Die hierarchische Struktur der thematischen Rollen wird eingeebnet. Alle thematischen Rollen beziehen sich immer auf das Verb. Zusätzlich sind aber auch Bezüge zu anderen thematische Rollen möglich.[11] Diese Einebnung ist zulässig, weil die innere Struktur der Bezeichner weitaus weniger komplex ist als die eines natürlichsprachlichen Satzes.

[9] Sowohl Fillmore als auch Stolz et al. (später im Text) verwenden nicht den Begriff der thematischen Rolle, sondern den des Falls. Für die Zwecke dieser Arbeit wird in diesem konkreten Beispiel aus Gründen der Lesbarkeit stets von thematischen Rollen gesprochen.

[10] In den linguistischen Konzepten belegt das Prädikat selbst keine thematische Rolle.

[11] Dieser Sonderfall wird in Abschnitt 4.3.2 behandelt.

- Eine thematische Rolle kann mehrfach in einem Vertrag belegt werden. Angenommen die thematische Rolle *SOURCE* beschreibe den Suchraum der Datensatzsuche. Dann könnte die Operation in mehreren verschiedenen *SOURCEs* suchen.

Somit ergibt sich die folgende Definition einer thematischen Rolle für diese Arbeit:

Definition 11 (Thematische Rolle) *Eine thematische Rolle repräsentiert die Menge aller zulässigen Vertragsbestandteile, die ein bestimmtes, vom Bezeichner gefordertes Charakteristikum der Operation beschreiben.*

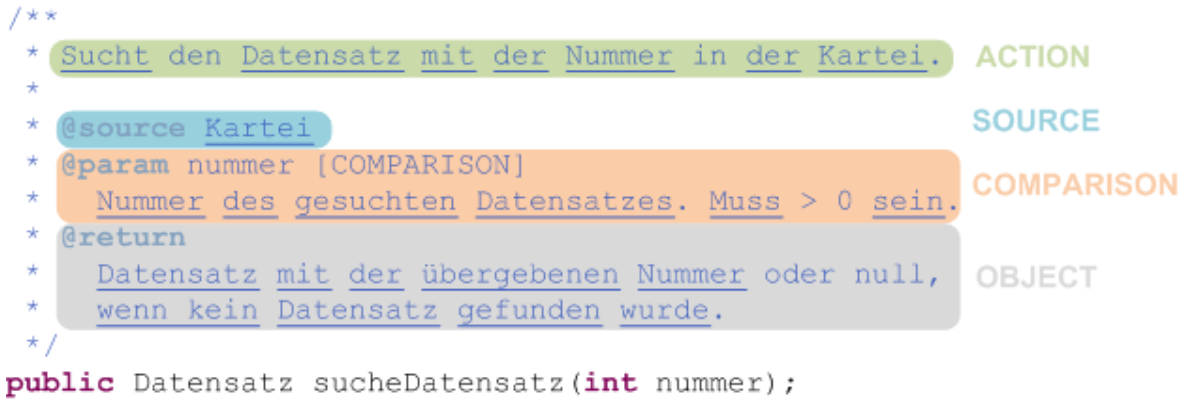


Abbildung 4.3: Korrespondierender API-Vertrag zu Satzbeispiel 1

Zur Veranschaulichung des abgewandelten Konzeptes thematischer Rollen ist ein zum Satzbeispiel 1 passender API-Vertrag in Abbildung 4.3 dargestellt. Für dieses Beispiel sind bereits thematische Rollen für software-technische Anwendungsfälle verwendet worden: ACTION, SOURCE, COMPARISON und OBJECT. Ihre Verwendung ist jeweils farblich hervorgehoben. Die Rolle SOURCE repräsentiert den Suchraum, also in diesem Fall die Kartei. Die Rolle COMPARISON beschreibt das Kriterium, anhand dessen das gesuchte OBJECT identifiziert werden kann (hier die Nummer). Das gesuchte Objekt selbst, hier der Datensatz, wird durch die Rolle OBJECT repräsentiert.[12]

Im Vergleich des vorherigen Satzbeispiels mit dem obigen Beispiel wird der Charakter einer thematischen Rolle im Kontext dieser Arbeit deutlich. Sie kennzeichnet jeweils Vertragsabschnitte, die für unterschiedliche Operationen sehr ähnliche Bedeutungen ausdrücken. Beispielsweise wäre der Relativsatz *der wie eine Rose duftet* keine gültige Belegung

[12] Die hier genannten thematischen Rollen sind der Arbeit von Girardi und Ibrahim entnommen worden (vgl. del Rosario Girardi Gutiérrez und Ibrahim, 1995, Seite 255). Eine detaillierte Beschreibung der Rollen erfolgt in Abschnitt 4.5.1. An dieser Stelle geht es in erster Linie um die Darstellung der Verwendung thematischer Rollen im API-Vertrag.

der thematischen Rolle COMPARISON, weil ein Datensatz nicht über die Eigenschaft eines Duftes verfügt. Es besteht hier eine gewisse Ähnlichkeit zum Konzept eines Datentyps. Ein Datentyp beschreibt die Menge der zulässigen Werte einer Variablen und der zulässigen Operationen auf diesen Werten (vgl. Louden, 2003, Seite 193 f.). Analog dazu beschreibt eine thematische Rolle die Menge aller Phrasen bzw. Vertragsabschnitte, die bestimmte Charakteristika der jeweiligen Operation repräsentieren. Es ist darauf hinzuweisen, dass die für eine thematische Rolle gewählte Belegung das Ergebnis einer Entwurfsentscheidung ist. Diese Entwurfsentscheidung ist vom menschlichen Autoren des Vertrages als Belegung ausgewählt worden. Eine Belegung kann nicht algorithmisch aus einem thematischen Raster abgeleitet werden.

Thematisches Raster: Suchende Operation		
Verben:	suchen, finden, filtern	
Them. Rolle:	*Bedeutung:*	*Obl. / Fak.*
ACTION	Suche	Obl.
OBJECT	Wird gesucht	Obl.
COMPARISON	Identifiziert das oder die OBJECTs	Obl.
SOURCE	Wo gesucht wird	Obl.

Tabelle 4.2: Vereinfachtes thematisches Raster einer suchenden Operation

Mit Hilfe thematischer Rollen allein kann die Valenz eines Operationsbezeichners jedoch nicht beschrieben werden. Es bedarf noch einer Zuordnung der thematischen Rollen zu den jeweiligen Bezeichnern, genauer gesagt zu ihrem Verb (in Funktion eines Prädikates). Zu diesem Zweck eignet sich das Konzept des thematischen Rasters. Nach Carnie ordnet ein thematisches Raster einem Verb eine Menge von thematischen Rollen zu. Es kann auch mehrere, inhaltlich ähnliche Verben umfassen. Aus dieser Beschreibung geht die genaue Bedeutung der thematischen Rollen für das jeweilige Verb hervor (vgl. Carnie, 2013, Seite 232 - 236). Es definiert für jede Rolle, ob sie obligatorisch oder fakultativ verwendet wird. So ergibt sich die folgende Definition des thematischen Rasters für diese Arbeit:

Definition 12 (Thematisches Raster) *Ein thematisches Raster repräsentiert eine Kategorie von Operationen. Es ordnet einer Menge von bedeutungsgleichen Verben eine Menge thematischer Rollen zu. Für jede Rolle ist hinterlegt, ob sie obligatorisch oder fakultativ auftritt.*

Die Tabelle 4.2 gibt ein vereinfachtes thematisches Raster zu Satzbeispiel 1 an.

```
/**
 * Liefert null, falls der übergebene Nachname nicht
 * gefunden wurde.
 *
 * @param nachname
 *     Darf nicht leer sein
 * @return Gefundene Kunden oder null
 * @throws IllegalArgumentException
 *  Wenn nachname leer ist
 */
public List<Kunde> sucheKundenFuerNachname_1
    (String nachname);
```

Listing 4.1: Vertrag der Operation sucheKundenFuerNachname

4.3 Stilleidentifikation mit thematischen Rastern

4.3.1 Konzept

Wie in Abschnitt 3.4.1 dargestellt, erfolgt die Unterstützung bei der Konstruktion von API-Verträgen oft prüflistenbasiert. Häufig erhält der Autor des Vertrages Unterstützung durch ein Assistenten-System. Die Aufgabe des Assistenten besteht in der Zusammenstellung der geforderten Vertragsinhalte zu einer bestimmten Operation auf Basis einer hinterlegten Prüfliste. Das daraus resultierende Blanko-Formular enthält beispielsweise die Parameter und die deklarierten Ausnahmen der Operation. Zur Erstellung des Vertrages füllt der Autor dieses Formular aus. In dieser Arbeit wird vorgeschlagen, diese bewährte Vorgehensweise um thematische Raster zu erweitern.[13] Die Abbildung 4.4 stellt den zugehörigen Prozess schematisch dar. Dieser Prozess wird im Folgenden anhand der in Listing 4.1 angegebenen Java-Operation `sucheKundenFuerNachname_1` veranschaulicht.[14]

Grundsätzlich ist für die Anwendung thematischer Raster ein Katalog erforderlich, in dem selbige dokumentiert sind. Ein solcher Katalog wird wahrscheinlich niemals allge-

[13] Das in diesem Abschnitt vorgestellte Konzept ist zum Teil bereits in den Beiträgen von Krause (2010) und Krause (2011) veröffentlicht worden.

[14] Im Positivfall liefert diese Operation die Liste der gefundenen Kunden. Da dies bereits aus dem Bezeichner hervorgeht, ist diese redundante Beschreibung im Vertrag ausgelassen worden.

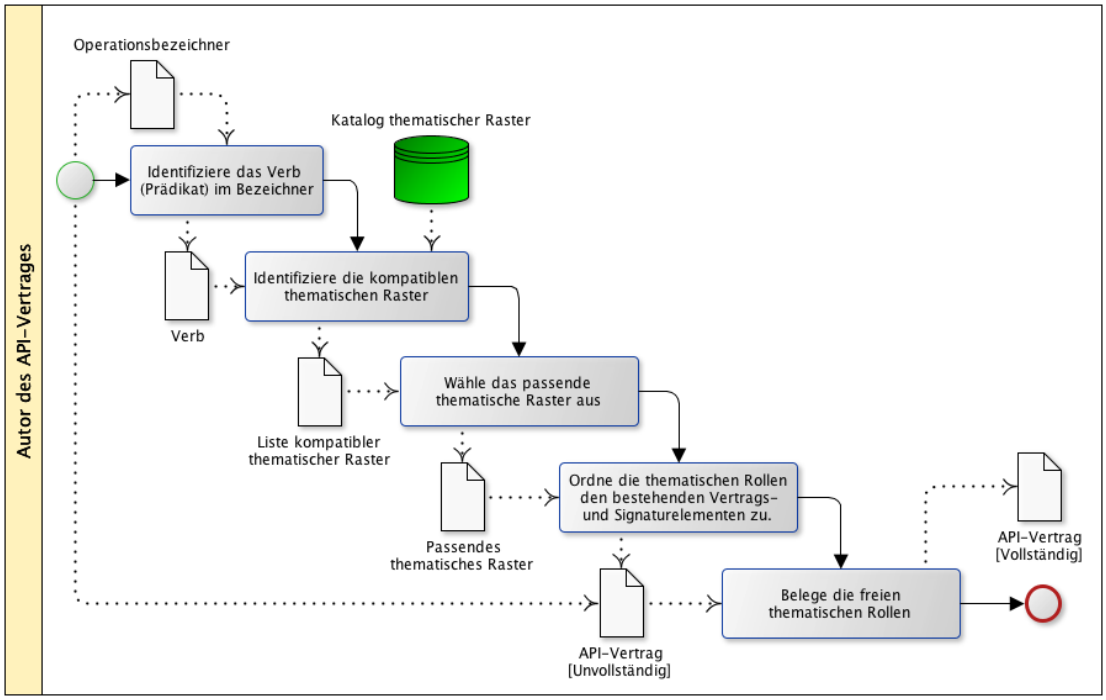


Abbildung 4.4: Prozess zur Identifikation von Stille mit thematischen Rastern (dargestellt in der Business Process Modeling Notation [BPMN])

meingültig sein und für alle Anwendungsfälle passen. Der praktische Einsatz dieses Verfahrens dürfte daher domänenspezifische Anpassungen erfordern. Einen Ausgangspunkt für die Definition eines individuellen Kataloges liefert diese Arbeit im Kapitel C des Anhangs. Die dort verzeichneten thematischen Raster sind auf empirische Weise im Rahmen der vorliegenden Arbeit erhoben worden.

Der dargestellte Prozess erhält als Eingabe den Bezeichner und den Vertrag der betrachteten Operation. Ziel ist es, Stille in diesem Vertrag zu identifizieren und so zu seiner Vervollständigung beizutragen. Der erste Schritt besteht in der Identifikation des Verbs bzw. Prädikates im Bezeichner. Anschließend werden die kompatiblen thematischen Raster aus dem Katalog herausgesucht. Ein thematisches Raster ist genau dann kompatibel zu einem Verb, wenn es dieses Verb in der Menge seiner zugeordneten Verben enthält. Dabei sind drei verschiedene Fälle möglich:

1. Das Verb ist zu keinem thematischen Raster kompatibel.
2. Das Verb ist zu mehreren thematischen Rastern kompatibel.
3. Das Verb ist zu genau einem thematischen Raster kompatibel.

Im ersten Fall bringen die thematischen Raster keine Verbesserung gegenüber der konventionellen signaturorientierten Vorgehensweise. Dann ist es ratsam, ein neues thematisches Raster auf Basis des Vertrages der betreffenden Operation zu definieren. Dieses Raster kann bei wiederholter Anwendung gegebenenfalls modifiziert werden.[15] Sollte eine Kompatibilität zu mehreren thematischen Rastern bestehen, so ist das am besten für die Operation passende auszuwählen. Für den im Anhangskapitel C vorgestellten Katalog trifft das beispielsweise für das englische Verb *to add* zu. Dieses Verb ist zu den thematischen Rastern *berechnende Operation* (mathematische Addition), *zusammenführende Operation* (Kombination zweier Objekte zu einem Objekt) und *deponierende Operation* (Ablage eines existierenden Objektes an einem Ziel) kompatibel. Der Idealfall besteht in genau einem kompatiblen thematischen Raster, welches die Operation korrekt abbildet. Aufgrund des Verbs *suche* im Bezeichner der Beispieloperation kann auf das in Tabelle 4.2 angegebene thematische Raster einer suchenden Operation geschlossen werden.

$$\begin{aligned} &\text{Suche} \\ &\quad [\ \text{Kunde}\ ]_{\text{OBJECT}} \\ &\quad [\ \text{Kundennummer / Id}\ ]_{\text{COMPARISON}} \\ &\quad [\ ?\ ]_{\text{SOURCE}} \end{aligned} \tag{4.1}$$

Nach Auswahl des passenden Rasters ordnet der Autor den thematischen Rollen die bestehenden Vertrags- bzw. Signaturelemente zu (siehe Formel 4.1). Sobald eine thematische Rolle nicht mit einer Phrase versehen werden kann, ist dies ein Indiz für Stille. Im gegebenen Beispiel sind keine Angaben des Vertrages zur SOURCE enthalten. Es ist nun am Autor, über das weitere Vorgehen zu entscheiden. Er hat mehrere Möglichkeiten:

1. Er kann die thematische Rolle ignorieren und sie nicht belegen. Diese Möglichkeit bietet sich beispielsweise an, wenn die betreffende Rolle für die jeweilige Operation nicht relevant oder kein Unterscheidungsmerkmal für Implementierungen ist.

2. Er kann die thematische Rolle als nicht spezifiziert kennzeichnen. Damit stellt die Rolle ein potentielles Unterscheidungsmerkmal für zukünftige Implementierungen dar. Dies bietet sich vorwiegend in Nutzungsverträgen an.

[15] Aus Gründen der Einfachheit ist die Behandlung dieses Sonderfalls nicht in Abbildung 4.4 berücksichtigt worden.

3. Er kann unter Berücksichtigung des Geheimnisprinzips und der Vertragsart (Realisierungs- oder Nutzungsvertrag) eine Phrase zur Belegung der Rolle formulieren. Diese Option empfiehlt sich meist für Realisierungsverträge.

Die erste Möglichkeit dient dem Fall, dass ein thematisches Raster nicht vollständig auf eine Operation passt. Da thematische Raster durchaus domänenspezifisch sein können, ist dieser Fall nicht auszuschließen. Beispielsweise wäre es denkbar, bei einer suchenden Operation nach einer thematischen Rolle MAP zu fragen. Im Fall einer Wegsuche kann der Autor diese Rolle mit dem verwendeten Kartenmaterial bzw. dessen Quelle oder Erzeugungsverfahren belegen. Im Fall der Suche eines Datensatzes in einer Datenbank oder Datei ergibt die Spezifikation des Kartenmaterials keinen Sinn und wird deshalb ausgelassen.

Alternativ dazu kann der Autor eine Entwurfsentscheidung als nicht spezifiziert kennzeichnen. Dies ist üblicherweise in Nutzungsverträgen der Fall. In solch einer Situation haben die Programmierer der Implementierungen in ihren Realisierungsverträgen die Möglichkeit, Unterschiede ihrer eigenen zu den anderen Implementierungen herauszuarbeiten. Sie müssen dort Belegungen für die nicht spezifizierten thematischen Rollen formulieren. Von der letztgenannten Pflicht handelt die dritte Handlungsmöglichkeit des Autors. Er kann die betreffende thematische Rolle auch mit einer Belegung versehen. Hierbei steht er vor der Herausforderung, zwischen zwei verschiedenen Interessen abzuwägen:

1. Freiheit zur späteren Änderung seiner Entwurfsentscheidungen (Wahrung des Geheimnisprinzips).

2. Explizite Definition der Abhängigkeiten seiner Komponente zur Laufzeitumgebung (Schutz vor Forderungen seiner Vertragspartner).

Da thematische Rollen mit Phrasen belegt werden, ist bei der Vereinigung der beiden Interessen die Wahl einer passenden Formulierung ausschlaggebend. Die Abbildung 4.5 zeigt drei Phrasen zur Belegung der thematischen Rolle SOURCE im Beispiel des Listings 4.1. Diese Phrasen lassen sich analog zu den drei bereits erwähnten Abstraktionsebenen von Scheer abgrenzen (siehe Abschnitt 2.1):

1. **Fachkonzept:** Beschreibung der SOURCE auf einer sehr konzeptuellen bzw. fachlichen Ebene (Mengendefinition der betroffenen Kunden). Eine derartige Belegung ist oft das Ergebnis der Anforderungsaufnahme mit der Fachabteilung. Sie ist daher prädestiniert für den Einsatz in einem Nutzungsvertrag.

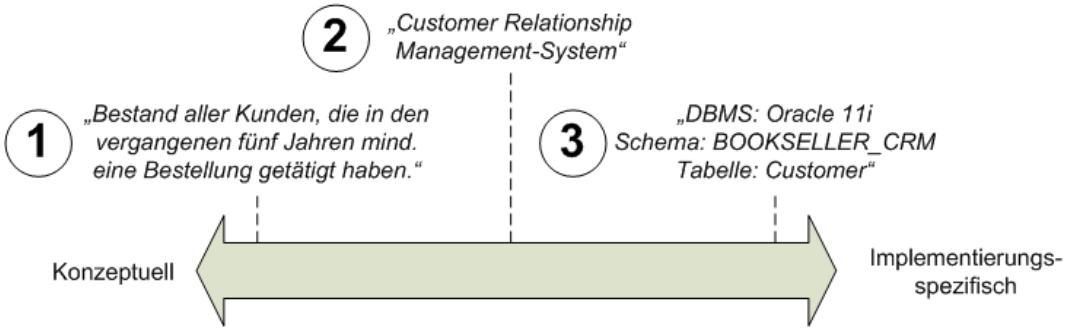


Abbildung 4.5: Phrasen zur Belegung der thematischen Rolle SOURCE einer suchenden Operation

2. **Datenverarbeitungskonzept:** Angabe eines konkreten Systems als Suchraum (genauer dessen Kundendatenbestand).Trotz der Systemangabe bleibt der konkret genutzte Speicher versteckt, z.B. das genutzte Datenbankschema. Diese Variante ist daher sehr gut für einen Realisierungsvertrag geeignet.

3. **Implementierungskonzept:** Angabe einer Tabelle in einem Schema eines konkreten Datenbankmanagementsystems (DBMS).

Die Variante 1 beschreibt die grundsätzliche Anforderung, nämlich die Suche nach Kunden, die in einem bestimmten Zeitraum Bestellungen getätigt haben. Durch Variante 2 wird eine Abhängigkeit der eigenen Komponente von diesem System gekennzeichnet. Gleichzeitig wird dem Leser des Vertrages deutlich, wo er Kundendaten ergänzen, ändern oder löschen muss, um Einfluss auf das Suchergebnis zu erzielen. Diese Angabe ist beispielsweise für den Entwurf von Testfällen relevant. Variante 3 impliziert, dass zur Änderung des Suchraumes der Zugriff auf diese Tabelle notwendig ist. Das bedeutet, dass alle mit der eigenen Komponente interagierenden fremden Komponenten diesen Zugriff implementieren müssen. Sollte der Wechsel des DBMS oder eine Änderung der betreffenden Tabellen notwendig werden, so kann dies aufgrund der Vielzahl anderer Systeme, die ebenfalls geändert werden müssen, zeitaufwändig und damit teuer werden. Die Verletzung des Geheimnisprinzips wiegt in diesem Fall folglich schwer. Deshalb sollten derartige Belegungen grundsätzlich vermieden werden.

Die Angabe eines konkreten DBMS kann jedoch in Form einer Abhängigkeit sinnvoll sein. In diesem Fall sollte die Nutzung des DBMS gegebenenfalls durch weitere Phrasen eingeschränkt werden, z.B. “Dieses DBMS dient der internen Datenhaltung dieser Kom-

```
/**
 * Liefert null, falls der übergebene Nachname nicht
 * gefunden wurde.
 *
 * @param nachname [COMPARISON]
 *     Darf nicht  leer sein
 * @return [OBJECT] Gefundene Kunden oder null
 * @throws IllegalArgumentException
 *  Wenn nachname leer ist
 * @suchraum Bestand aller Kunden, die in den
 *  vergangenen fünf Jahren mind. eine Bestellung
 *  getätigt haben.
 * @thematicgrid Suchende Operation
 */
public List<Kunde> sucheKundenFuerNachname_2
    (String nachname);
```

Listing 4.2: Vervollst. Nutzungsvertrag der Operation sucheKundenPerNachname

ponente. Es ist anderen Systemen untersagt, seine Inhalte direkt zu lesen oder gar zu ändern bzw. zu erweitern."

Das Listing 4.2 enthält einen vervollständigten Vorschlag für einen Nutzungsvertrag des Beispiels.[16] Aus diesem Vertrag wird die vorgeschlagene Integration von thematischen Rollen in *javadoc* und ähnlichen Sprachen zur Gestaltung von Verträgen ersichtlich. Da für die Signaturelemente bereits *javadoc*-Tags existieren, wird die thematische Rolle in eckigen Klammern hinter dem Tag bzw. dem vom Tag adressierten Bezeichner notiert. Auf diese Weise können Parameter und Resultate beschrieben werden (z.B. COMPARISON oder OBJECT). Im Falle von Nicht-Signaturelementen wird aus dem Namen der thematischen Rolle ein eigenes *javadoc*-Tag erzeugt (z.B. SOURCE).

Bei genauer Betrachtung fällt außerdem auf, dass die Beschreibung des Parameters *nachname* etwas gekürzt worden ist. Die Intention hinter der Phrase "Nummer des gesuchten Kunden" ist wahrscheinlich die explizite Kennzeichnung des Parameters als Such-

[16] Im Positivfall liefert diese Operation die Liste der gefundenen Kunden. Da dies bereits aus dem Bezeichner hervorgeht, ist diese redundante Beschreibung im Vertrag ausgelassen worden.

```
public void bestaetigeBestellung(String benutzer,
    int bestellungNr);
```

Listing 4.3: Signatur der Operation approveOrder

kriterium. Dies wird aber bereits durch die thematische Rolle COMPARISON ausgedrückt und kann daher entfallen. Eine gewisse Redundanz ist allerdings nach wie vor nicht von der Hand zu weisen. Schließlich geht die Verwendung des Parameters in diesem Beispiel bereits aus dem Bezeichner der Operation hervor. Durch die Verknüpfung der thematischen Rolle mit dem Parameter wird diese Schlussfolgerung zusätzlich noch einmal explizit gekennzeichnet. Die explizite Kennzeichnung entspricht einem verbreiteten Vorgehen, welches u.a. in die Konventionen zur Gestaltung von *javadoc*-Verträgen Einzug gefunden hat (vgl. Oracle Corp., ohne Jahr, Abschnitt "Tag Conventions", "Required Tags"). Zusätzlich ist das gewählte thematische Raster dieser Operation mittels des Tag *@thematicgrid* angegeben worden. Diese Angabe unterstützt zusätzlich die Interpretation des Vertrages durch den Leser.

Abschließend ist zu ergänzen, dass sich selbst für vermeintlich „technische" Rollen fachliche Bezüge gemäß der von Scheer angegebenen Dreiteilung finden lassen. Zur Demonstration wird an dieser Stelle dem Abschnitt 4.6.2 und dem Anhang C.1 vorgegriffen, welche die thematische Rolle USERNAME dokumentieren. Wie die Bezeichnung der Rolle vermuten lässt, handelt es sich hierbei um einen Benutzernamen. Im Beispiel der Operation *bestaetigeBestellung* wird der Parameter *benutzer* mit dieser thematischen Rolle verknüpft (siehe Listing 4.3). Über diese Operation kann die Bestellung mit der übergebenen *bestellungNr* als bestätigt markiert werden. Die Belegung der thematischen Rolle USERNAME ist nun mit verschiedenen Phrasen möglich:

1. "Eine Bestellung muss von einer Führungskraft der 4. Ebene oder höher (Stelle aus einem Organigramm) bestätigt werden." (Fachkonzept)

2. "Eine Bestellung muss von einem Abteilungsleiter (Rolle aus einem Use Case-Diagramm) bestätigt werden." (Datenverarbeitungskonzept)

3. "Eine Bestellung muss von einem Mitglied der Benutzergruppe 'abteilungsleiter' aus dem Active-Directory-Verzeichnisdienst bestätigt werden." (Implementierungskonzept)

Anhand dieser Beispiele ist erkennbar, dass die vermeintlich "technische" Rolle USERNAME Bezüge zu fachlichen Belegungen hat. In diesem Fall ist nur eine bestimmte Personengruppe der Aufbauorganisation befugt, eine Bestellung zu bestätigen: die Abteilungsleiter. An dieser Stelle wird deutlich, wie viel Ermessensspielraum der Autor bei der Wahl von Phrasen zur Formulierung eines konkreten Vertrages hat.

4.3.2 Rasterbasierte Regeln

Der vorherige Abschnitt zeigt, wie aus dem Verb des Bezeichners Rückschlüsse auf einzelne Charakteristika der jeweiligen Operation gezogen werden können. Zusätzlich ist es unter Einbeziehung der bereits belegten thematischen Rollen möglich, weitere Charakteristika zu identifizieren. Im Beispiel aus Listing 4.2 ist die thematische Rolle OBJECT z.B. mit der zurückgelieferten Kundenliste belegt worden. Diese Liste enthält potentiell mehr als einen Kunden und ist als Liste eine sortierbare Datenstruktur. Daher stellt die Sortierung der Liste ebenfalls ein relevantes Charakteristikum dar. Im Fall einer suchenden Operation hängt die Empfehlung der thematischen Rolle ORDERING demnach vom Numerus (Singular oder Plural) des Kopfes der Nominalphrase ab, mit der die thematische Rolle OBJECT belegt worden ist. Hat der Phrasenkopf den Numerus Plural, so ist nach dem ORDERING zu fragen.

Folglich hängen die vom thematischen Raster empfohlenen Rollen nicht allein vom Verb des Bezeichners ab, sondern auch von bereits formulierten Vertragsabschnitten (in diesem Fall der Beschreibung der aus dem Aufruf der Operation resultierenden Kundenliste als OBJECT). Rasterbasierte Regeln dienen der Steuerung derartiger dynamischer Empfehlungen. Jeder thematischen Rolle ist in einem thematischen Raster genau eine rasterbasierte Regel zugeordnet. Formal betrachtet ist eine rasterbasierte Regel eine zweistellige Funktion, die einen Wahrheitswert liefert. Als Argumente erhält diese Funktion immer den aktuell bestehenden Vertrag V und die Signatur *Sig* der betreffenden Operation. Liefert sie *WAHR* als Resultat, so ist die Notwendigkeit einer Belegung der betreffenden thematischen Rolle zu empfehlen, ansonsten nicht. Die rasterbasierte Regel R zur thematischen Rolle ORDERING im thematischen Raster *suchende Operation* ist wie in Formel 4.2 angegeben definiert.

$$R_{(V,Sig)}^{\text{Suchende Operation, ORDERING}} = istPlural(OBJECT, V, Sig) \tag{4.2}$$

Der linke Teil der Formel vor dem Gleichheitszeichen gibt die Signatur der rasterbasierten Regel an. Aus ihr geht hervor, auf welches thematische Raster und auf welche Rolle sich die Regel bezieht. Außerdem sind hier die Parameter der Regel sichtbar (der Vertrag V und die Signatur *Sig*). Der rechte Teil nach dem Gleichheitszeichen beschreibt die Definition der Regel. In der Definition hat die Funktion *istPlural* Verwendung gefunden. Sie ist dreistellig und prüft, ob der Kopf der Belegung der übergebenen thematischen Rolle (erstes Argument) im übergebenen Vertrag (zweites Argument) für die übergebene Signatur (drittes Argument) den Numerus Plural hat.

Der Leser mag sich fragen, weshalb das Argument *Sig* erforderlich ist. Schließlich kann der Numerus allein aus der sprachlichen Formulierung der Rollenbelegung abgeleitet werden. *Sig* dient einer kleinen Optimierung. Nicht jede Datenstruktur, die mehrere Elemente enthalten kann, ist sortierbar. Dies ist z.B. für mathematische Mengen der Fall, denn Mengen besitzen keine Ordnung. Nähme man an, die Operation `sucheKundenFuerNachname_2` liefere keine Kundenliste, sondern eine Kundenmenge, so dürfte das thematische Raster nicht nach dem ORDERING der Kunden fragen. Schließlich verfügt eine Menge gemäß ihrer mathematischen Definition über keine Ordnung. Somit droht aufgrund der geforderten Sortierung dieser Menge ein Widerspruch innerhalb des Vertrages. Durch Betrachtung der Signatur ist jede rasterbasierte Regel in der Lage, derartige Datenstrukturen zu erkennen und mit in die Empfehlungsbildung einzubeziehen.

Dieses Beispiel lässt sich wie folgt verallgemeinern: Die Funktion $R_{(V,Sig)}^{TRA,TRO}$ gibt an, ob die thematische Rolle TRO auf Basis des Vertrages V und der Signatur *Sig* für das thematische Raster TRA empfohlen werden soll oder nicht. Im obigen Beispiel ist TRO durch ORDERING und TRA durch *suchende Operation* zu ersetzen. Diese Überlegung führt zur folgenden Definition einer rasterbasierten Regel:

Definition 13 (Rasterbasierte Regel) *Eine rasterbasierte Regel R des thematischen Rasters TRA für die thematische Rolle TRO ist eine zweistellige Funktion, die einen Wahrheitswert liefert. Sie steuert die Empfehlung von TRO auf Basis des Vertrages V (erstes Argument) zur Signatur Sig (zweites Argument). Liefert R WAHR, so ist TRO zu empfehlen, ansonsten nicht.*

Die im Rahmen dieser Arbeit identifizierten Funktionen zur Definition rasterbasierter Regeln sind in Tabelle 4.3 zusammengestellt.[17] Für detaillierte Beispiele zu rasterbasierten

[17] Die Signaturen der hier beschriebenen Funktionen sind aufgrund der vorteilhafteren softwaretechnischen Realisierung immer mit denselben Argumenten angegeben, auch wenn nicht jedes Argument in jeder Funktion benötigt wird.

Funktion:	**Bedeutung:**
$existiert_{(\text{thematische Rolle},V,Sig)}$	Prüft, ob für die thematische Rolle eine Belegung im Vertrag V enthalten ist.
$istPlural_{(\text{thematische Rolle, V, Sig})}$ $istSingular_{(\text{thematische Rolle, V, Sig})}$	Prüft auf Basis von *Sig*, ob der Numerus des Kopfes der Belegung der jeweiligen thematischen Rolle Singular bzw. Plural ist.
$immer_{(\text{thematische Rolle, V, Sig})}$	Liefert immer *wahr*.
$hatAttribute_{(\text{thematische Rolle, V, Sig})}$	Prüft auf Basis von *Sig*, ob der Datentyp des mit der thematischen Rolle in V belegten Signaturelementes über Attribute verfügt. Ist die thematische Rolle nicht mit einer Klasse verknüpft, so wird $falsch$ geliefert.

Tabelle 4.3: Identifizierte Funktionen für rasterbasierte Regeln

Regeln wird auf den Katalog thematischer Raster in Anhang C.2 verwiesen. Die Abbildung 4.6 stellt die Erweiterung des Prozesses zur Generierung von Empfehlungen um die Anwendung rasterbasierter Regeln dar. Neu ist hier die Aktivität zur Ergänzung der auf Basis bereits beschriebener Vertragsinhalte abgeleiteten thematischen Rollen.

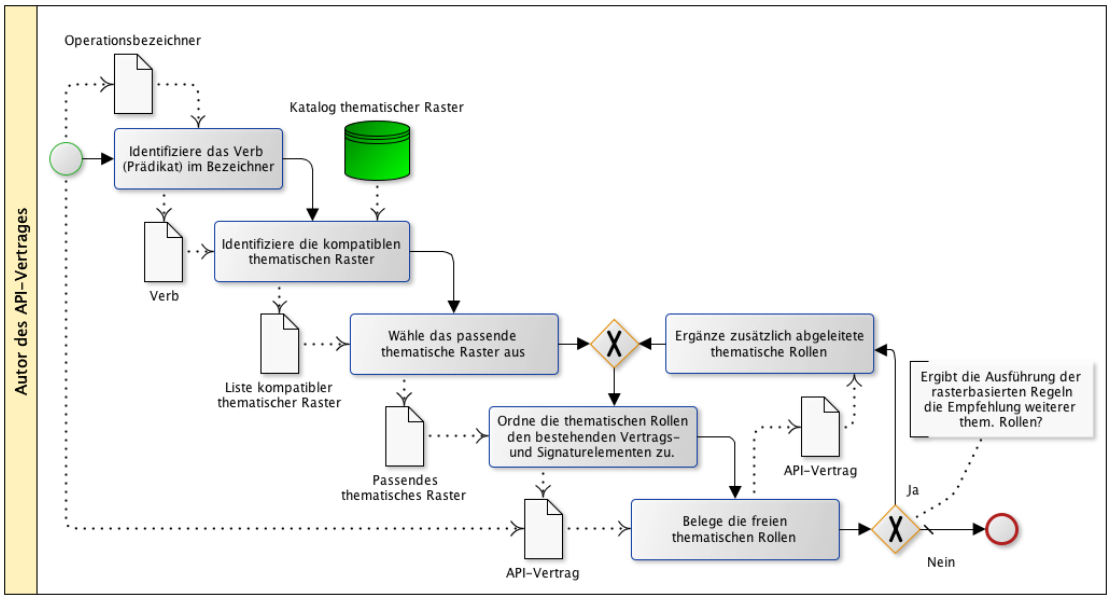


Abbildung 4.6: Prozess zur Identifikation von Stille mit thematischen Rastern inkl. rasterbasierter Regeln (dargestellt in der BPMN)

4.3.3 Identifikation von Ausnahmen und Fehlern

Meyer unterscheidet zwei verschiedene Arten von Fehlern, die während der Ausführung einer Operation auftreten können: die Ausnahme und den Fehler. Eine Ausnahme bezeichnet das Auftreten einer nicht normalen Situation während der Ausführung der Operation. Bei einem Fehler hingegen genügt ein Teil des Programms nicht der Spezifikation (vgl. Meyer, 1988, Seite 147 f.). In Java werden darüber hinaus zwei verschiedene Arten von Ausnahmen unterschieden: geprüfte Ausnahmen (Checked Exceptions) und Laufzeitausnahmen (Runtime Exceptions). Nach Bloch ist eine geprüfte Ausnahme zu verwenden, sobald die ursächlichen Bedingungen für die Ausnahme im Einflussbereich des Operationsbenutzers liegen. Dies ist z.B. bei vielen Ein-/Ausgabe-Fehlern der Fall (z.B. aufgrund fehlender Zugriffsrechte auf eine Datei). Eine geprüfte Ausnahme muss in der Signatur der Operation deklariert werden. So kann der Compiler bzw. Interpreter bereits prüfen, ob Benutzer der Operation diese Ausnahmen auch behandelten. Laufzeitausnahmen hingegen weisen auf einen Programmierfehler hin und sind daher nicht Bestandteil der Operationssignatur (vgl. Bloch, 2008, Seite 244 f.).

Thematische Raster können notwendige geprüfte Ausnahmen auf generische Weise ermitteln. Dazu ist für jede belegte thematische Rolle zu fragen: Wie reagiert die Operation, wenn das durch die Belegung der Rolle beschriebene Charakteristikum nicht verfügbar ist? Auf diese einfache Weise können Lücken des Vertrages in Bezug auf die Fehlerbehandlung bei Nichtverfügbarkeit von Ressourcen identifiziert werden, z.B. nicht verfügbare Dateien, Datenbanken oder Fremdsysteme.

Anhand des vorherigen Beispiels der Operation zur Kundensuche wird das genaue Vorgehen veranschaulicht (siehe das vereinfachte thematische Raster einer *suchenden Operation* aus Tabelle 4.2 und die Operation `sucheKundenFuerNachname_2` aus Listing 4.2). Die verwendeten thematischen Rollen lauten ACTION, OBJECT, COMPARISON und SOURCE. Die Rolle ACTION ist in diesem Fall Teil des Programms selbst und damit permanent verfügbar. Wäre sie nicht verfügbar, so läge eine Laufzeitausnahme oder ein Fehler vor. Das OBJECT wird durch den gefundenen Datensatz repräsentiert. Entweder ein Datensatz wird gefunden oder nicht - beide Fälle stellen keine geprüfte Ausnahme dar. Daher ist auch hier keine Ausnahme bzw. kein Fehler zu behandeln. Der Fall des nicht verfügbaren COMPARISON stellt eine geprüfte Ausnahme dar und wird bereits durch die `IllegalArgumentException` behandelt.[18] Diese wird geworfen, falls der

[18] Die Ausnahme `java.lang.IllegalArgumentException` ist per Vererbungshierarchie als Laufzeitausnahme deklariert. Im vorliegenden Beispiel-API liegt die Wahl der Aktualparameter allerdings im

```
/**
 * Liefert null, falls der übergebene Nachname nicht
 * gefunden wurde.
 *
 * @param nachname [COMPARISON] Darf nicht leer sein
 * @return [OBJECT] Gefundene Kunden oder null
 * @throws IllegalArgumentException
 *   [ERROR COMPARISON] Wenn nachname leer ist
 * @throws IllegalStateException
 *  [ERROR SOURCE] Wenn die Menge der zu
 *  durchsuchenden Kunden nicht verfügbar ist.
 * @source Bestand aller Kunden, die in den vergangenen
 *   fünf Jahren mind. eine Bestellung getätigt haben.
 * @thematicgrid Suchende Operation
 */
public List<Kunde> sucheKundenFuerNachname_3
    (String nachname);
```

Listing 4.4: Vervollst. Nutzungsvertrag der Operation sucheKundenPerNachname

übergebene Nachname leer ist. In diesem Fall wird ein ungültiger Aktualparameter als Nichtverfügbarkeit der Belegung der zugehörigen thematischen Rolle interpretiert.

Etwas anders verhält es sich im Fall der SOURCE. Als SOURCE werden je nach Implementierung verschiedene Speicher verwendet. Es ist durchaus denkbar, dass einer dieser Speicher nicht verfügbar ist. Sofern die Abhängigkeit von diesem Speicher im Vertrag definiert ist, ist es die Aufgabe der Laufzeitumgebung, die Verfügbarkeit des Speichers sicherzustellen. Die daraus resultierende geprüfte Ausnahme dient dazu, diese Situation anzuzeigen. Das Listing 4.4 gibt den um diese geprüfte Ausnahme ergänzten Nutzungsvertrag für die Beispieloperation an.[19]

Einflussbereich des Benutzers (Aufrufers). Aus diesem Grund wird in diesem Fall von einer geprüften Ausnahme gesprochen.

[19] Der Leser mag einwenden, dass in diesem Beispiel die thematische Rolle ACTION im Vertrag nicht explizit gekennzeichnet worden ist. Davon ist aus Gründen der Lesbarkeit abgesehen worden. Wie in Abschnitt 4.2 beschrieben, wird diese Rolle mit den einleitenden Sätzen des Vertrages und dem Operationsbezeichner belegt.

4.3.4 Voraussetzungen für thematische Raster

Nach der Darstellung der Vervollständigung von API-Verträgen mittels thematischer Raster stellt sich die Frage, welche Voraussetzungen für den Einsatz dieses Verfahrens erfüllt sein müssen. Im Rahmen dieser Arbeit konnten zwei verschiedene identifiziert werden:

1. Es muss mind. ein Verb im Operationsbezeichner enthalten sein.
2. Das Verb muss semantisch gehaltvoll genug sein, um die Operation einem thematischen Raster zuordnen zu können.

Die erstgenannte Voraussetzung ist sehr elementar. Trotzdem gibt es plausible Fälle, in denen sie auf den ersten Blick verletzt wird. Meist geschieht dies durch Bezeichner, welche gänzlich ohne Verb stehen. Derartige Konstellation können beispielsweise bei den berechnenden Operationen der Klasse `java.lang.Math` des *Java Development Kits* beobachtet werden (für einen Ausschnitt der Operationssignaturen dieser Klasse siehe Listing 4.5) (vgl. Oracle Corp., 2013a, Vertrag der Klasse `java.lang.Math`). Die Bezeichner der angegebenen Beispiele benennen jeweils das berechnete Resultat, also das Signum (signum) oder den Logarithmus (log).

```
1 public class Math {
    [...]
3   public static float signum(float f);
    public static double log(double a);
5   [...]
  }
```

Listing 4.5: Auszug einzelner Operationssignaturen aus der Klasse `java.lang.Math`

Weitere Beispiele in Java sind die konvertierenden Operationen `asList` der Klasse `java.lang.Arrays` und `toString` der Klasse `java.lang.Object`. Erstere überführt ein Array in eine äquivalente Liste und liefert diese zurück (vgl. Oracle Corp., 2013a, Vertrag der Schnittstelle `java.util.Arrays`). Letztere liefert eine Repräsentation des jeweiligen Objektes, auf dem sie aufgerufen wird, als Zeichenkette (vgl. Oracle Corp., 2013a, Vertrag der Klasse `java.lang.Object`). In allen Fällen muss der Aufrufer der Operation jeweils auf das Verb rückschließen. Sofern dies möglich ist, kann das passende thematische Raster ausgewählt werden. Andernfalls deutet das fehlende Verb auf eine

nicht eindeutige Zuständigkeit der Operation hin. In diesem Fall wäre es ratsam, den Entwurf der Operation derartig zu überarbeiten, dass eine Zuordnung zu mind. einem thematischen Raster möglich ist.[20]

Das Konzept der thematischen Raster ist in den vorherigen Abschnitten auf Basis der Annahme beschrieben worden, dass Operationsbezeichner als Imperativsätze zu interpretieren sind. Da die zuletzt genannten Beispiele allesamt kein Verb aufweisen, stellt sich die Frage, ob die Analogie zu einem natürlichsprachlichen Satz und damit die Anwendbarkeit thematischer Raster nach wie vor gilt. Die Antwort lautet: ja. In der Linguistik werden Auslassungen von Konstituenten im Satz, auf die mit Kenntnis des Kontextes rückgeschlossen werden kann, als Ellipsen bezeichnet (vgl. Kürschner, 2008, Seite 212). In der gesprochenen Sprache treten derartige Auslassungen häufig auf. Bowman klassifiziert derartige gesprochene Äußerungen trotz der Auslassungen als Satz (genauer gesagt als *minor sentence*), weil sie sowohl über eine Satzmelodie verfügen als auch über eine Sprechpause am Ende (wie reguläre Sätze mit Subjekt und Prädikat bzw. Prädikat im Imperativ auch) (vgl. Bowman, 1966, Seite 18). In den von Bowman aufgezeichneten Äußerungen finden sich sowohl Auslassungen des Verbs bzw. Hilfsverbs als auch des Subjektes (vgl. Bowman, 1966, Seite 53 f.). Im Folgenden sind die Bezeichner der zuvor genannten Beispiele noch einmal mit Angabe der elliptischen Auslassungen in Klammern aufgeführt:[21]

(2) (***compute***) log

(3) (***compute***) signum

(4) (***get the given array***) as list

(5) (***convert the object***) to string

Einen weiteren Sonderfall bilden Bezeichner mit mehreren Prädikaten. Ein fiktives Beispiel für einen solchen Fall ist in Listing 4.6 angegeben. In diesem Fall sind die zwei

[20] An dieser Stelle ist selbstverständlich nicht gemeint, dass sich der Entwurf der Operationen nach dem Katalog der thematischen Raster richten muss. Trotzdem ist es erfahrungsgemäß elementar für ein wartbares Programm, dass seine Operationen über eine eindeutig erkennbare Zuständigkeit verfügen. Diese kann dann auch gegebenenfalls in einem neu zu definierenden thematischen Raster abgebildet werden.

[21] Bei der Ableitung des Satzbeispiels 4 ist das semantisch wenig reichhaltige Verb *to get* gewählt worden, um eine zum tatsächlichen Bezeichner konsistente Formulierung zu erhalten. Selbstverständlich wäre ein Verb wie *to convert*, welches die durchgeführte Umwandlung in eine Liste ausdrückt, vorzuziehen. In diesem Fall müsste der Bezeichner allerdings in `toList` umformuliert werden.

```
public static void generateAndSendInvoice(Order order){
 PDFDocument invoice = generateInvoice(order);
 sendInvoice(invoice, order.getCustomer().getEMailAddress());
}
```

Listing 4.6: Methode mit mehreren Prädikaten im Bezeichner

Verben *to generate* und *to send* durch die Konjunktion *and* verbunden. Es bestehen zwei Alternativen, um einen solchen Fall zu behandeln:

1. Kombination der thematischen Raster für jedes einzelne Prädikat.

2. Umformulierung des Bezeichners, sodass er nur ein einziges Prädikat enthält.

Gemäß der ersten Variante muss der Vertrag zur Operation `generateAndSendInvoice` die thematischen Raster *erzeugende Operation* und *sendende Operation* miteinander kombinieren. Das bedeutet, dass die obligatorischen thematischen Rollen beider Raster mit Belegungen zu versehen sind. Die fakultativen Rollen beider Raster können bei Bedarf ebenfalls belegt werden. Sollte eine thematische Rolle in beiden Rastern vorkommen, so ist ihre Belegung unter Berücksichtigung aller durch die Prädikate ausgedrückten Tätigkeiten zu formulieren. In jedem Fall wird eine solche Formulierung für die thematische Rolle ACTION notwendig sein, denn diese thematische Rolle ist in jedem durch diese Arbeit dokumentierten Raster enthalten. Ein Beispiel für eine solche zusammengefasste Belegung ist in Listing 4.7 angegeben.

Alternativ dazu kann der Autor den Bezeichner auch systematisch umformulieren. Dazu bestehen zwei Möglichkeiten:

1. **Wahl eines neuen Prädikates:** Der Autor formuliert ein neues Verb, welches alle zuvor im Operationsbezeichner genannten Prädikate einschließt. Im genannten Beispiel könnten das Erstellen und Verschicken einer Rechnung unter *in Rechnung stellen* subsummiert werden. Der modifizierte Bezeichner könnte in diesem Fall z.B. `charge` lauten. Als Ausgangsbasis für die Auswahl eines neuen Verbs kann ein Lexikon an domänenspezifischen Verben dienen. In diesem konkreten Beispiel ergibt sich aus dem neu gewählten Prädikat die Frage, wofür die Rechnung gestellt wird. Daher

```
/**
 * Versendet ein Rechnungsdokument im PDF-Format an den Auftrag-
 * geber der übergebenen Bestellung. Dieses PDF-Dokument wird
 * nicht weiter persistiert, sondern dient nur dem Versand. Der
 * Versand selbst erfolgt per E-Mail an die E-Mail-Adresse des
 * Auftraggebers.
 * [...]
 */
}
```

Listing 4.7: Beispiel für eine zusammengefasste Belegung der thematische Rolle ACTION

kann der Bezeichner in diesem konkreten Fall um die Bestellung zu `chargeOrder` ergänzt werden.[22]

2. **Wahl eines primären Prädikates:** Der Autor wählt das primäre Prädikat aus dem Operationsbezeichner aus. Als *primär* gilt jenes Prädikat, welches die finale Aktion der Operation beschreibt (oder anders formuliert: die letzte Aktion, welche auf einer konzeptuellen Ebene zur Beschreibung der Operation erforderlich ist). Die Effekte der verbleibenden Prädikate werden durch Adjektive ausgedrückt. Im gegebenen Beispiel wäre das primäre Prädikat *to send*, weil das Versenden der Rechnung in diesem Beispiel die letzte konzeptuell auszuführende Tätigkeit ist. Der umformulierte Bezeichner lautet entsprechend `sendGeneratedInvoice`.

Neben dem vollständigen Mangel an Prädikaten bzw. der Existenz mehrerer Prädikate ist zu Beginn dieses Abschnittes eine weitere, wichtige Voraussetzung dieses Ansatzes identifiziert worden: der semantische Gehalt des Prädikates bzw. der Prädikate. Diese Problematik lässt sich am bereits bekannten Beispiel der Operation `sucheKundenFuerNachname` demonstrieren. Denn alternativ ist z.B. auch der Bezeichner `gibKundenFuerNachname` für diese Operation denkbar. Schließlich erhält der Aufrufer eine Liste von Kunden für den übergebenen Nachnamen. Insofern ist diese Formulierung sprachlich durchaus stimmig zur Operation. Diese Variante entspricht in etwa der in englischsprachigen Bezeichnern gebräuchlichen Formulierung auf Basis des Verbs *to get*, z.B. `getCustomerByLastname`.

[22] Die Anwendbarkeit der zuletzt beschriebenen Modifikation ist sehr spezifisch für das neu gewählte Prädikat *to charge* und kann nicht für jeden Fall verallgemeinert werden.

Die in den Beispielen verwendeten Verben *geben* und *to get* sind beide jeweils sehr allgemein. Das bedeutet, sie beschreiben eine Vielzahl möglicher Operationen. Es ist für einen menschlichen Leser anhand dieser Verben nicht erkennbar, ob die betreffende Operation das zurückgelieferte Resultat neu erzeugt, es aus einem Speicher liest, es nach einer mathematischen Formel berechnet oder es anhand einer Geschäftsregel ableitet. Da eine Unterscheidung bzw. Abgrenzung verschiedener Operationen mit einem derart allgemeinen Prädikat für einen Menschen nicht möglich ist, können auch thematische Raster hier auf Basis des Operationsbezeichners nicht helfen. Die Frage der Zuständigkeit der betreffenden Operation kann in einem solchen Fall letztendlich nur unter Hinzunahme des Quelltextes beantwortet werden. Daher geht mit dem Einsatz thematischer Raster die Anforderung an die Programmierer zur Verwendung semantisch gehaltvoller Verben einher.

4.3.5 Diskussion

Anhand der drei Beispiele der Operation `sucheKundenFuerNachname` ist zuvor gezeigt worden, wie Stille in einem Vertrag auf Basis des Bezeichners der Operation identifiziert werden kann. Nach Ergänzung des Vertrages gibt das vorgestellte Verfahren gegebenenfalls weitere Hinweise auf potentielle Stille. Da es ausschließlich auf Basis der Bezeichner und des Vertrages funktioniert, ist es weitgehend unabhängig von einer konkreten Programmiersprache. Es benötigt zudem keinen Zugriff auf die zugrunde liegenden Quelltexte. Damit ist es sowohl für Nutzungsverträge (auf Ebene einer Operation) als auch für Realisierungsverträge (auf Basis einer Methode) einsetzbar. Außerdem können Verträge mit diesem Verfahren bereits zu einem sehr frühen Zeitpunkt der Entwicklung geprüft werden. Dies ermöglicht eine ebenfalls frühe Ergänzung des Vertrages und folglich eine potentielle Kostenersparnis im Vergleich zu einer späteren Korrektur. Des Weiteren kann Stille nicht nur im Vertrag, sondern auch in der Signatur selbst identifiziert werden. Dies wäre z.B. bei ausgelassenen Parametern in der Rolle eines COMPARISON im obigen Beispiel der Fall.

Das Verfahren ist aber nicht für jede beliebige Operation nutzbar. Sein Einsatz bedarf der Erfüllung mehrerer Grundvoraussetzungen. Zum einen muss der Bezeichner der Operation mindestens ein Verb beinhalten oder sich auf ein Verb zurückführen lassen. Die Überprüfung dieser Anforderung anhand einer repräsentativen Stichprobe spiegelt sich in Forschungsfrage 2 wider.

Forschungsfrage 2 *Werden verschiedenartige Verben in signifikanter Häufigkeit zur Bildung von Operationsbezeichnern verwendet?*

Die zweite wichtige Voraussetzung besteht in der Verwendung mind. eines semantisch gehaltvollen Verbs. Dieses muss mind. einem thematischen Raster zugeordnet werden können. Diese Zuordnung wiederum setzt die Definition der thematischen Rollen und Raster in einem Katalog voraus. Nur mittels eines solchen Kataloges kann Stille im Vertrag auf Basis von Bezeichnern ermittelt werden. Dieser Gedanke führt zur Recherchefrage 5 und zur Forschungsfrage 3.

Recherchefrage 5 *Existieren thematische Rollen bzw. Raster im Kontext software-technischer Arbeiten, die für die Gestaltung von API-Verträgen verwendet werden können?*

Forschungsfrage 3 *Welche thematischen Rollen und Raster sind für die Gestaltung von API-Verträgen im Speziellen notwendig?*

Aber auch im Falle der Erfüllung aller genannten Voraussetzungen bestehen beim vorgestellten Ansatz einige offene Punkte. Löbner weist darauf hin, dass die in der Linguistik identifizierten thematischen Rollen und ihre Abgrenzung untereinander zwischen verschiedenen Autoren variieren. Gleichwohl gibt er eine Menge thematischer Rollen an, über die ein weitgehender Konsens besteht (vgl. Löbner, 2002, Seite 112 f.). Gelhausen führt an, dass es zusätzlich an einem standardisierten Vorgehen zur Ermittlung thematischer Rollen und zum Test der gefundenen Rollen mangelt (vgl. Gelhausen, 2010, Seite 82 - 85). Bezogen auf die Vollständigkeit eines API-Vertrages kann dieser Mangel dazu führen, dass die beteiligten Parteien mit unterschiedlichen thematischen Rastern zu unterschiedlichen Einschätzungen bzgl. der Vollständigkeit des Vertrages kommen. Diese Überlegung führt zu der Frage, ob ein derartiger Katalog ein gesättigtes Niveau erreichen kann (siehe Forschungsfrage 4).

Forschungsfrage 4 *Wie skalieren die ermittelten thematische Rollen und Raster für API-Verträge?*

Die Abhängigkeit der Qualität der Empfehlungen vom semantischen Gehalt des Operationsbezeichners, genauer gesagt seines Prädikates, ist anhand eines Vergleiches der Bezeichner `sucheKundenFuerNachname` und `gibKundenFuerNachname` dargelegt worden. Ein weiterer offener Punkt ist, dass zwar einzelne thematische Rollen, nicht aber

deren Kardinalität bzw. Belegung abgeleitet werden können. Es ist nach wie vor erforderlich, dass sich der Autor beispielsweise zur Anzahl der nach Kunden durchsuchten Systeme bzw. deren Namen äußert.

4.4 Eignung der thematischen Raster für API-Verträge

4.4.1 Ziele dieser Untersuchung

Das Ziel dieser ersten Untersuchung besteht in der Beantwortung der Forschungsfrage 2. Es gilt zu klären, ob Verben in der beschriebenen Form überhaupt zur Bildung von Operationsbezeichnern verwendet werden. Eine wichtige Grundlage für die Spezifität der Empfehlungen thematischer Raster für eine bestimmte Operation ist der semantische Gehalt des Verbs in deren Bezeichner. Das bedeutet, das verwendete Verb muss die zugehörige Operation möglichst stark von anderen Operationen abgrenzen. Angenommen die Bezeichner enthielten immer dasselbe Verb. Dann könnten thematische Raster zwar eingesetzt werden, aber es wäre nicht möglich, bestimmte Raster von vornherein auszuschließen. Der Programmierer müsste in diesem Fall alle im Katalog verzeichneten Raster prüfen, um das passende auszuwählen. Deshalb gilt es herauszufinden, ob und wie viele verschiedene Verben in welcher Häufigkeit in Operationsbezeichnern verwendet werden.

Dieser Abschnitt beschreibt die hierzu durchgeführte empirische Querschnittstudie.[23] Eine Querschnittstudie ist eine empirische Forschungsmethode. Der Begriff *empirische Forschungsmethode* bezeichnet das planmäßige und systematische Gewinnen von Erfahrung zu Forschungszwecken.[24] Empirische Untersuchungen erheben zur Beschreibung des Untersuchungsgegenstandes Daten verschiedener Dimensionen in Variablen (z.B. die natürliche Sprache, in welcher der Vertrag verfasst worden ist). Eine Variable ist ein Symbol, das durch alle möglichen Ausprägungen der jeweiligen Dimension ersetzt werden kann. Tritt nur eine Ausprägung einer Dimension auf, so handelt es sich um eine Konstante (vgl. Bortz und Döring, 2006, Seite 2). Als Beispiel ist die Variable *Sprache* denkbar. Sie kann demnach durch die Ausprägungen *Deutsch*, *Englisch* etc. ersetzt werden. Im Kontext empirischer Untersuchungen lassen sich verschiedene Arten von Variablen unterscheiden (vgl. Bortz und Döring, 2006, Seite 3):

[23] Die in diesem Abschnitt vorgestellten Ergebnisse sind teilweise bereits in dem Konferenzbeitrag von Krause (2010) vorgestellt worden.

[24] Vgl. Laatz (1993), Seite 9 und Dudenredaktion (2009), Begriff “Empirie”.

- Beobachtete Veränderungen von **abhängigen Variablen** werden in Experimenten mit der Manipulation von **unabhängigen Variablen** erklärt.

- Wenn eine dritte Variable die Auswirkung einer unabhängigen Variable auf eine abhängige verändert, bezeichnet man die dritte Variable als **Moderatorvariable**.

- Sofern eine unabhängige Variable eine abhängige Variable nicht direkt, sondern indirekt über eine dritte Variable beeinflusst, wird diese dritte Variable **Mediatorvariable** genannt.

- Wenn Moderatorvariablen vorsorglich erhoben werden, um ihre Einflüsse gegebenenfalls zu analysieren, bezeichnet man sie als **Kontrollvariablen**.

Kontrollvariablen enthalten vorsorglich erhobene Daten, um gegebenenfalls weitere Einflüsse zu erkennen bzw. auszuschließen (vgl. Bortz und Döring, 2006, Seite 3).

Eine Querschnittstudie erhebt in Form einer oder mehrerer Stichproben einmalig, zeitgleich und auf ähnliche Weise bestimmte Merkmale einer Population (vgl. Baltes, 1967, Seite 11). Auf diese Weise können eventuell vorhandene Unterschiede bzgl. der betrachteten Merkmale zwischen den Stichproben aufgezeigt werden. Querschnittstudien zählen daher zur Menge der Ex-Post-Facto-Studien. Diese Kategorie von Verfahren untersucht die Beziehungen der betrachteten Merkmale untereinander. Im Unterschied zu einem Experiment kann der Forscher bei Ex-Post-Facto-Untersuchungen diese Merkmale im Nachhinein nicht verändern bzw. kontrollieren. Entweder ist dies grundsätzlich nicht möglich (z.B. die Änderung des Geschlechts eines Probanden) oder die Auswirkungen sind bereits eingetreten (z.B. die Modellierung einer Operationssignatur). Daher sind auf Basis von Ex-Post-Facto-Studien keine Rückschlüsse auf Ursache-Wirkungs-Beziehungen möglich (vgl. Kerlinger, 1979, Seite 580 - 582). Da erfahrungsgemäß sehr viele Bezeichner in englischer Sprache formuliert werden, konzentriert sich auch diese Untersuchung ausschließlich auf englische Verben.

4.4.2 Das seekda-Korpus

Zuallererst ist zu klären, auf Basis welcher Daten die Querschnittstudie durchgeführt werden soll. Notwendig ist eine möglichst umfangreiche Menge an Operationssignaturen. In der Linguistik wird eine solche Sammlung gesprochener bzw. geschriebener Texte für Analysezwecke als Korpus bezeichnet (vgl. Carnie, 2013, Seite 12). Ein repräsentatives Korpus

sollte Signaturen aus möglichst vielen verschiedenen Ländern und Programmiersprachen umfassen.

In dieser Untersuchung wird eine umfangreiche Stichprobe von Beschreibungen real existierender Web Service-APIs betrachtet. Diese Beschreibungen müssen im standardisierten Format der Web Service Description Language (WSDL) vorliegen. Die in WSDL formulierten Verträge sind strukturell mit *javadoc*-Verträgen vergleichbar. Nach Bean wird das Konzept eines Web Service häufig bei der Integration von Anwendungen eingesetzt. Dafür spricht u.a. seine Unabhängigkeit von einer konkreten Programmiersprache (vgl. Bean, 2010, Seite 32 - 34 und Seite 43 - 48). Aufgrund dieser Unabhängigkeit von einer bestimmten Technologie ist zu erwarten, dass Web Service APIs eine aussagekräftige Grundlage für das zu bestimmende Korpus bilden.

Das Unternehmen seekda GmbH aus Innsbruck betreibt eine hausinterne Suchmaschine für Web Service APIs. Der Index dieser Suchmaschine enthält WSDL-Beschreibungen von Web Services aus dem Internet. Er wird mittels sog. Crawler-Programme aufgebaut. Diese Programme durchsuchen das Internet selbstständig nach WSDL-Beschreibungen und indexieren diese automatisch. Für die weiteren Untersuchungen wird das Abbild dieses Index vom 11.09.2009 verwendet.[25]

Korpus-Attribut:	**Wert:**
Anzahl verschiedener Top-Level-Domains der Service-Anbieter	89
Anzahl der Service-Anbieter	8.947
Anzahl der WSDL-Beschreibungen	17.453
Anzahl der WSDL-Porttypes (Schnittstellen)	24.471
Anzahl der Operationen	186.741

Tabelle 4.4: Auswahl quantitativer Attribute des seekda-Korpus

Die Tabelle 4.4 zeigt eine Auswahl relevanter quantitativer Attribute des seekda-Korpus. Die Anzahl der Internet-Top-Level-Domains (z.B. DE, CH, FR etc.) wird als Indiz für eine hohe Abdeckung von Web Services aus verschiedenen Ländern interpretiert. Außerdem kann der Umfang des Korpus mit 186.741 Operationen inkl. Bezeichner als repräsentativ angesehen werden.

[25] Dieses Abbild ist von der seekda GmbH freundlicherweise bereitgestellt worden.

4.4.3 Versuchsdurchführung

Die WSDL-Beschreibungen des seekda-Korpus liegen in Form von Dateien vor. Jede Datei enthält genau eine WSDL-Beschreibung. Aus diesen gilt es nun die Verben der Operationsbezeichner zu extrahieren. Auf Basis des folgenden Procedere wird jeder Bezeichner in eine der vier in Tabelle 4.5 dargestellten Kategorien eingeteilt:

1. Extrahiere alle Operationsbezeichner aus den WSDL-Beschreibungen. Ein Operationsbezeichner ist der Wert des Attributes *name* aller *wsdl:operation*-Elemente.

2. Füge vor jedem Großbuchstaben aller Bezeichner jeweils ein Leerzeichen ein. Hier gilt die Annahme, dass jedes neue Wort in den Bezeichnern mit einem Großbuchstaben beginnt. Diese Annahme wird durch den Umstand gestützt, dass diese Namenskonvention in vielen Programmierrichtlinien Anwendung findet.[26] Ergänze außerdem am Ende eines jeden Bezeichners einen Punkt (Zeichen für Satzende).

3. Bestimme die Wortart aller Wörter mit Hilfe eines Part-of-Speech-Tagger.[27] Das PennTreebank-Tagset definiert die Menge der möglichen Wortarten.

4. Sammele alle Wörter, die als Verb im Infinitiv oder als finites Verb im Präsens erkannt worden sind.[28]

5. Überprüfe in einem Lexikon für das Englische, ob die identifizierten Verben auch dort als Verb verzeichnet sind. Entferne alle Wörter, für die das nicht zutrifft.

Bzgl. dieses Vorgehens sind ein paar Anmerkungen notwendig. Die erste betrifft die Wortart einzelner Wörter. Sie ist in einigen Fällen nur bei Kenntnis des kompletten Satzes bestimmbar, so z.B. in den Bezeichnern `email invoice` und `check email`. Beide Bezeichner enthalten das Wort `email` - als Verb (1. Beispiel) und als Substantiv (2. Beispiel). Um Fehlklassifikationen durch den Part-of-Speech-Tagger zu erkennen, werden die

[26] Derartige Vorgaben finden sich beispielsweise für Java in (Oracle Corp., 1999, Kapitel 9. Naming Conventions) oder für C# in (Microsoft Corp., ohne Jahr, Kapitel Naming Guidelines, Abschnitt Capitalization Styles).

[27] Ein Part-of-Speech-Tagger ist ein Programm, welches Wörtern im Satz die entsprechende Wortart zuordnet.

[28] Die einzelnen Tags haben die folgenden Bedeutungen: Verb in Basisform (VB), Verb in der ersten oder zweiten Person Singular Präsens (VBP) und Verb in der dritten Person Singular Präsens (VBZ). Eine Übersicht aller Tags des Penn Treebank Tagset ist in Marcus et al. (1993) enthalten.

Kategorie:	**Verb enthalten:**	**Verb erkannt:**
Korrekt klassifizierter Bezeichner	Ja	Ja
Nicht klassifizierter Bezeichner	Ja	Nein
Falsch klassifizierter Bezeichner	Nein	Ja
Korrekt ignorierter Bezeichner	Nein	Nein

Tabelle 4.5: Fehlerklassen bei der Extraktion von Operationsbezeichnern aus dem seekda-Korpus

gefundenen Verben im Lexikon gegengeprüft. Nur ein als Verb markiertes Wort, welches auch im Lexikon als Verb verzeichnet ist, wird als Verb gezählt.

Die zweite Anmerkung betrifft die zu erwartenden Klassifikationen der Wortarten durch den Tagger. Der Stanford Tagger basiert auf einem probabilistischen Ansatz. Das verwendete Modell ist größtenteils auf englischen Texten, das heißt überwiegend mit Sätzen in Subjekt-Prädikat-Objekt-Struktur trainiert worden. Im Gegensatz dazu besitzen Bezeichner sehr häufig die Struktur eines Imperativsatzes (mit implizitem Subjekt, wie z.B. in `search for the file on the hard disk`). Dies führt zu einer hohen Anzahl von Tagger-Fehlern und in der Folge davon zu vielen nicht korrekt klassifizierten Bezeichnern.[29] Zum Beispiel wird der Bezeichner `process XML` durch den Tagger mit $\texttt{process}_{NN}$ $\texttt{XML}_{NNP}$ ausgezeichnet. Beide Wörter sind als Substantive markiert worden, obwohl *process* in diesem Kontext ein Verb ist. Erklären lässt sich diese Ausgabe damit, dass in einem typischen Deklarativsatz vor dem Verb erst das Subjekt erscheint und dieses entweder ein Substantiv oder Personalpronomen ist. Daher ist die Wahrscheinlichkeit für derartige Markierungen relativ groß.

Um die nicht klassifizierten Bezeichner trotzdem zu einem möglichst großen Anteil für die Untersuchung nutzen zu können, wird ein kleiner Trick angewendet. Nach dem ersten Durchlauf des obigen Procedere werden alle nicht klassifizierten Bezeichner an ihrem Anfang um das Personalpronomen *we* ergänzt. Der Tagger liefert anschließend für das obige Beispiel we_{PRP} $process_{VBZ}$ XML_{NNP}. *process* ist folglich als Verb identifiziert worden.[30] In einem zweiten Durchlauf wird das obige Procedere erneut auf die derartig

[29] Eine anschauliche Darstellung der Verwendung impliziter Subjekte in Imperativsätzen findet sich in Winograd (1983), Seite 493 f.

[30] Die Markierung des Verbs *process* mit dem Tag VBZ ist zwar falsch, weil es sich hier nicht um die dritte Person Singular, sondern um die erste Person Plural handelt. Aber die Klassifikation als Verb ist für diese Analyse hinreichend. Das korrekte Tag für diesen Fall lautet VBP.

modifizierten Bezeichner angewendet. Abschließend werden die in beiden Durchläufen identifizierten Verben zusammengeführt und ausgewertet.

Neben dem seekda-Korpus werden die folgenden zusätzlichen Ressourcen eingesetzt:

- **Stanford Part-of-Speech-Tagger Version 1.6:** eines der derzeit leistungsfähigsten Programme zur Erkennung von Wortarten in einem Satz. Im Rahmen dieser Untersuchung wird das in der Datei *bidirectional-wsj-0-18.tagger* mitgelieferte statistische Modell verwendet (vgl. Toutanova et al., 2003, Seite 173 f.).
- **PennTree Tagset:** Klassifikationssystem von Wortarten[31]
- **WordNet Version 3.0:** Englisches Lexikon, in dem zu einem Wort die Wortart, Synonyme (Wörter mit gleicher Bedeutung), Antonyme (Wörter mit gegensätzlicher Bedeutung) etc. erfasst sind.[32]

4.4.4 Ergebnisse

Zur Beantwortung der Forschungsfrage 2 ist die Klärung zweier verschiedener Aspekte notwendig. Zum einen gilt es herauszufinden, ob Verben überhaupt eine Rolle in Operationsbezeichnern spielen. Sollten Verben in den Bezeichnern vorkommen, so ist zum anderen zu prüfen, welche Verben das genau sind.

Das seekda-Korpus umfasst 161 Bezeichner, die einen Punkt enthalten (z.B. `clients.updateClient`). Der Stanford Tagger hat diese Bezeichner deshalb als zwei Sätze interpretiert. Aus diesem Grund werden sie bei der weiteren Analyse nicht berücksichtigt. Daher umfasst die Menge der betrachteten Bezeichner 186.580 Elemente.[33] Insgesamt sind Verben in 150.188 Fällen, das heißt in ca. 80.5% der Bezeichner nachgewiesen worden.[34] Da die Anzahl der nicht klassifizierten und der korrekt ignorierten Bezeichner relativ hoch ist, sind die betreffenden Daten noch einmal stichprobenartig im Detail gesichtet worden. Diese Analyse führte zur Identifikation der folgenden vier verschiedenen Ursachen für korrekt ignorierte Bezeichner:

[31] Das Penn Treebank Tagset wird in Marcus et al. (1993) detailliert vorgestellt und diskutiert.

[32] Miller gibt eine sehr gute und prägnante Vorstellung von WordNet in Miller (1995).

[33] Der in Krause (2010) veröffentlichte Wert für die ausgelassenen Bezeichner weicht aufgrund eines früheren Messfehlers von dem hier genannten Wert um 30 Bezeichner ab.

[34] Diese Betrachtung lässt die 13.180 nicht klassifizierten Bezeichner aus, weil in ihnen nur durch WordNet, aber nicht durch den Stanford Tagger Verben gefunden worden sind. Bei Berücksichtigung dieser Bezeichner steigt der Anteil von Bezeichnern mit Verben auf 150.188 + 13.180 = 163.368 oder 87.6%.

- Der Bezeichner ist nicht in englischer Sprache formuliert worden.
- Der Bezeichner enthält kein Verb (z.B. `optional info` oder `from csv to xml`).
- Der Bezeichner ist nicht nach der Camel-Case-Konvention formuliert worden (z.B. `getserverversion`).[35]
- Das Verb des Bezeichners ist abgekürzt worden (z.B. `mkdir` für `make directory` oder `crmget inbox folder` für `get inbox folder of customer relationship management system`).[36]

Die hohe Anzahl der nicht klassifizierten Bezeichner ist auf den Tagger zurückzuführen. Trotz der Ergänzung des Personalpronomens *we* hat der Tagger in vielen Fällen falsche Tags gewählt. So lieferte er für den Bezeichner *we message query data* die Markierungen we_{PRP} $message_{NN}$ $query_{NN}$ $data_{NNS}$ oder für den Bezeichner *we order* die Markierungen we_{PRP} $order_{NN}$.[37] Die Verben *to message* und *to order* sind in diesen Fällen als Substantive erkannt worden. Vermutlich könnte die Erkennungsqualität hier durch ein speziell auf Imperativsätzen trainiertes Tagger-Modell weiter verbessert werden. Aufgrund der hohen Anzahl von Bezeichnern mit Verben und des Umfangs der Stichprobe ist der erste zu klärende Aspekt der Forschungsfrage 2 trotzdem zu bejahen: In der überwiegenden Zahl der Fälle werden Verben zur Bildung von Operationsbezeichnern verwendet. Es bleibt nun noch die Frage, welche Verben dies sind.

In den 150.188 betrachteten Bezeichnern konnten 830 verschiedene englische Verben identifiziert werden.[38] Die Tabelle 4.6 enthält die 30 häufigsten Verben absteigend sortiert nach ihrer absoluten Häufigkeit. Es fällt auf, dass die Häufigkeit der Verben exponentiell abnimmt. Das häufigste Verb *to get* wird ca. 12,3-mal so oft verwendet wie das zweithäufigste Verb *to add*. Aber das zweithäufigste Verb *to add* kommt beispielsweise nur ca. 1,2-mal so häufig vor wie das dritthäufigste Verb *to create*. Es scheint, als gelte das

[35] Die Camel-Case-Notation ist bei der Anwendung mehrerer Programmiersprachen verbreitet, z.B. Java. Sie besagt, dass jedes neue Wort in einem Bezeichner mit einem Großbuchstaben beginnen muss.

[36] Die Leerzeichen in den hier angegebenen Bezeichnern sind der besseren Lesbarkeit wegen eingeführt worden.

[37] Die einzelnen Tags haben die folgenden Bedeutungen: Personalpronomen (PRP), Substantiv im Singular (NN) und Substantiv im Plural (NNS). Eine Übersicht aller Tags des Penn Treebank Tagset ist in Marcus et al. (1993) enthalten.

[38] Detaillierte Angaben zu den Klassifikationen der Bezeichner sind im Anhang in Tabelle B.1 angegeben.

Rang	**Verb**	**Häufigkeit**
1.	get	74.790
2.	add	6.105
3.	create	4.929
4.	delete	4.769
5.	update	4.398
6.	send	3.176
7.	list	2.658
8.	check	1.944
9.	remove	1.861
10.	is	1.845
11.	load	1.762
12.	search	1.742
13.	run	1.727
14.	find	1.367
15.	echo	1.146
16.	submit	1.041
17.	save	1.018
18.	wait	949
19.	modify	944
20.	do	913
21.	validate	841
22.	register	836
23.	insert	728
24.	edit	701
25.	query	553
26.	retrieve	539
27.	process	535
28.	change	526
29.	test	524
30.	describe	500

Tabelle 4.6: Die 30 häufigsten Verben des seekda-Korpus

Zipf'sche Gesetz auch für die Verteilung der Verben in Operationsbezeichnern. Dieses Gesetz besagt, dass das Produkt aus Rang und Häufigkeit eines Wortes in einem Korpus annähernd konstant ist (vgl. Zipf, 1949, Seite 24).

Unabhängig davon, welcher exponentiellen Verteilungsfunktion die Verteilung der Verben im Detail genügt, ist Zipfs Erklärung für die Ursache der Verteilung trotzdem anwendbar.[39] Er führt die Verteilung von Wörtern auf das Prinzip des geringsten Aufwands bei der Wortwahl zurück. Nach Zipf strebt der Sprecher nach einer Sprache mit möglichst wenigen Wörtern, mit denen er seine Intention ausdrücken kann (vgl. Zipf, 1949, Seite 20 f.). Im vorliegenden Fall ist das Verb *to get* für jede Operation passend, die ein Resultat liefert. Die Verwendung dieses Verbs reduziert folglich den Aufwand der Programmierer zur Formulierung eines passenden Bezeichners. Auch die Betrachtung der weiteren Verben bestätigt diese Theorie. Viele der beobachteten Verben treten häufig in softwaretechnischen Zusammenhängen auf, z.B. in Programmiersprachen (*get* (1), *run* (13) oder *do* (20)), in Abfragesprachen wie SQL (*delete* (4), *update* (5) oder *describe* (30)) oder Namen für Konsolenkommandos (*find* (14), *echo* (15), *edit* (24) oder *test* (29)).[40]

4.4.5 Diskussion

Offensichtlich nutzen Programmierer existierende Fachterminologien zur Auswahl der Verben. Auch auf diese Weise reduzieren sie den Aufwand zur Bildung eines Bezeichners, weil der Bezeichner nicht komplett neu erdacht werden muss. Die Kategorien werden dabei vermutlich durch die Erfahrung der Programmierer mit Werkzeugen, Kommandos oder Programmen gebildet. So ergibt sich außerdem eine hohe Wahrscheinlichkeit zur korrekten Interpretation des Bezeichners durch Kollegen. Alle diese Implementierungen verfügen über einen Namen und verhalten sich auf eine spezielle Weise. Sobald der Programmierer einen Namen für eine neue Operation sucht, wählt er Teile des Namens einer möglichst ähnlichen Implementierung. Zusammenfassend bleibt festzustellen, dass mit 830 Verben eine hohe Bandbreite verschiedenartiger Verben in Operationsbezeichnern nachgewiesen worden ist.

Für die bemerkenswerte Häufigkeit von *to get* können mehrere Faktoren ursächlich sein. Es wäre möglich, dass Programmierer aus reiner Bequemlichkeit immer dasselbe Verb verwenden. Alternativ ist denkbar, dass Englisch für viele Programmierer nicht die

[39] Einen Überblick verschiedener Exponentialverteilungen und eine Methodik zur Ermittlung der exponentiellen Verteilungsfunktion hinter einer Stichprobe finden sich z.B. in Clauset et al. (2009).

[40] Die geklammerten Nummern geben den Rang des Verbs in Tabelle 4.6 an.

Muttersprache ist. Daher ist es wahrscheinlich, dass sie im Englischen über einen geringeren Wortschatz verfügen als in ihrer Muttersprache. Deshalb wählt der Programmierer ein möglichst generisches Verb für den Bezeichner. Man könnte dieser Position entgegenhalten, dass er in diesem Fall Verben seiner Muttersprache mit Hilfe eines Wörterbuches ins Englische übersetzen könnte. Neben dem Aufwand des Nachschlagens ist eine Übersetzung aber zusätzlich mit dem Risiko der wörtlichen und damit potentiell missverständlichen Verbwahl verbunden. Dies wird am Beispiel einer konvertierenden Operation deutlich. Das deutsche Verb *umwandeln* kann wörtlich mit *to convert* (im Sinne von *verwandeln*) oder mit *to translate* (im Sinne von *übersetzen*) übersetzt werden. Diese Übersetzungen können bei einem Kollegen völlig unterschiedliche Erwartungshaltungen gegenüber der betreffenden Operation wecken, z.B. die Umwandlung von Daten aus einem in ein anderes Dateiformat oder die Übersetzung von Sätzen aus einer in eine andere Sprache.

Eine weitere Erklärung für die häufige Verwendung von *to get* ist, dass es für den Programmierer oft schwierig zu sein scheint, eine bestimmte Operation zur Wahl des Bezeichners zu kategorisieren. Mehrere Autoren berichten, dass ein ähnliches Phänomen beim Spracherwerb von Kleinkindern unabhängig von der jeweiligen Landessprache zu beobachten ist. Die ersten rund 100 Wörter im Wortschatz von Kindern im Alter unter drei Jahren sind zum überwiegenden Teil Substantive. Die wenigen vorhandenen Verben beschreiben vornehmlich sichtbare Handlungen wie *rennen*, *gehen* oder *werfen*. Obwohl die Mütter in ihrer Wortwahl häufig Verben wie *sehen* oder *denken* verwendeten, finden diese kaum Einzug in den Wortschatz des Kindes.[41]

Snedeker und Gleitmann vermuten, dass der Lerner beim Erlernen einer Sprache Konzepte mit den gelernten Vokabeln verknüpft, z.B. das Konzept eines Hauses mit der Vokabel *Haus*. Diese Verknüpfung ist für Objekte der realen Welt (bezeichnet durch Substantive) leichter als für Handlungen (bezeichnet durch Verben). Beobachtbares kann demnach nur durch Beobachtung gelernt werden - unabhängig vom Alter und von der kognitiven Entwicklung des Lernenden (vgl. Snedeker und Gleitmann, 2004, Seite 266). Deshalb ist zu vermuten, dass thematische Raster Programmierer bei der Bildung eines präzisen Bezeichners signifikant unterstützen. Schließlich repräsentieren sie Kategorien von Operationen inkl. zugeordneter Verben. Der Programmierer kann für eine konkrete Operation aus dem Katalog thematischer Raster dasjenige auswählen, welches seiner Operation am ähnlichsten ist. Damit werden die Operationen vor dem Hintergrund des Kataloges zu etwas

[41] Eine sehr gute und kompakte Übersicht über die verschiedenen Positionen zur Erklärung der geringen Anzahl von Verben findet sich in Snedeker und Gleitmann (2004), Seite 257 - 261.

"Beobachtbarem". Auf diese Weise wird durch thematische Raster ein elementares Prinzip des Spracherwerbs auf die Formulierung von Bezeichnern für Operationen übertragen.

Nichtsdestotrotz scheint die Anwendbarkeit thematischer Raster auf den ersten Blick in Frage gestellt. Das mit großem Abstand am häufigsten verwendete Verb ist gleichzeitig das semantisch ärmste. Dieser Aspekt wurde bereits in Abschnitt 4.3.4 diskutiert. Auf den zweiten Blick und unter dem Eindruck der zuvor zitierten Erkenntnisse von Snedeker und Gleitmann offenbart sich jedoch eine Herausforderung, die u.a. durch thematische Raster direkt angegangen wird. Während der Forschung an dieser Thematik konnte kein systematischer und zusammenhängender Ansatz zur Konstruktion von Bezeichnern für verschiedene Typen von Operationen aufgefunden werden. Daher ist davon auszugehen, dass Programmierer sowohl während ihrer Ausbildung als auch während ihres Berufslebens nicht an die systematische Verwendung ihrer Muttersprache in den eigenen Programmen herangeführt werden. Die Wahl natürlichsprachlicher Formulierungen im Programm obliegt vielmehr ihrer Intuition. Daher kann die beobachtete Häufigkeit von *to get* wohl kaum als Argument gegen die Anwendung thematischer Raster verwendet werden. Thematische Raster ermöglichen ja gerade aufgrund ihres Angebotes an spezifischen Verben die sprachliche Abgrenzung von Operationen zueinander. Somit ist die häufige Verwendung von *to get* vielmehr ein Beleg für die Notwendigkeit, weitere Kapazitäten in die Erforschung dieser Thematik zu investieren.

4.5 Thematische Raster in der Softwaretechnik

Im vorangegangenen Abschnitt ist gezeigt worden, dass englische Verben in einer großen Vielfalt in Operationsbezeichnern genutzt werden. Damit sind die zwei wichtigsten Voraussetzungen für den Einsatz thematischer Raster erfüllt. Dieser Abschnitt untersucht, ob in der software-technischen Literatur bereits Kategorien existieren, die als thematische Rollen bzw. Raster verwendet werden können. Es werden verschiedene publizierte Ansätze mit entsprechendem Bezug vorgestellt und auf in dieser Arbeit verwendbare Kategorien überprüft. Die so aufgefundenen Kategorien werden als Grundlage für die spätere Ableitung der thematischen Raster genutzt (siehe Forschungsfrage 3).

4.5.1 Indexierung von Software-Komponenten

Girardi und Ibrahim stellen das ROSA-System (**R**euse **o**f **S**oftware **A**rtifacts) vor, das passende Software-Komponenten aufgrund natürlichsprachlicher Suchanfragen finden soll.

Als Wissensbasis nutzt ROSA einen Index, den es automatisch auf Basis der natürlichsprachlichen Dokumentationen der Komponenten erstellt (vgl. del Rosario Girardi Gutiérrez und Ibrahim, 1995, Seite 249 f.). Zur Strukturierung des Index schlagen Girardi und Ibrahim 13 thematische Rollen vor, die Software-Komponenten charakterisieren. Diese Rollen sind mit ihren Originalbezeichnungen in Tabelle 4.7 dargestellt.[42] Jede thematischen Rolle ist ein Attribut der indexierten Komponente und jede Rollenbelegung entspricht dem Wert des Attributes (vgl. die Grammatik in del Rosario Girardi Gutiérrez und Ibrahim, 1995, Seite 258).

Thematische Rolle	**Beschreibung**
ACTION	Die von der Komponente angebotene Hauptfunktion.
AGENT	Die Komponente, welche die Funktionalität anbietet.
COMPARISON	Die Entität, welche mit dem Hauptobjekt verglichen wird.
CONDITION	Die Prämisse, unter der die Hauptfunktion ausgeführt wird.
DESTINATION	Das Ziel des Hauptobjektes der Komponente.
DURATION	Dauer der ACTION.
GOAL	Das Hauptobjekt, Ziel oder Resultat der ACTION.
INSTRUMENT	Die Entität, mit der die ACTION durchgeführt wird.
LOCATION	Die Entität, an der die ACTION durchgeführt oder die als Medium für die ACTION genutzt wird.
MANNER	Der Modus, in dem die ACTION durchgeführt wird.
PURPOSE	Der Zweck der Komponente oder der ACTION.
SOURCE	Der Ursprung der Hauptobjektes.
TIME	Zeitpunkt, zu dem die ACTION durchgeführt wird.

Tabelle 4.7: Die 13 von Girardi und Ibrahim identifizierten thematischen Rollen

In der Arbeit von Girardi und Ibrahim findet sich jedoch keine Ableitung von thematischen Rastern. Sie konzentriert sich vorwiegend auf die Extraktion von Rollenbelegungen aus den natürlichensprachlichen Dokumentationen. Dabei ist jede thematische Rolle für jede Software-Komponente fakultativ. Funktional unterschiedliche Komponenten können auf diese Weise in einer einheitlichen Struktur indexiert werden. Der Indexierer fordert keine speziellen thematischen Rollen für einen bestimmten Komponententyp.

[42] Vgl. del Rosario Girardi Gutiérrez und Ibrahim (1995), Seite 255. Die Beschreibungstexte der einzelnen Rollen sind vom Autor dieser Arbeit ins Deutsche übersetzt worden.

4.5.2 UML-Modellextraktion aus narrativen Texten

Gelhausen hat untersucht, inwieweit sich semi-formale Modelle aus natürlichsprachlichen Texten systematisch ableiten lassen. Zu diesem Zweck schlägt er den Formalismus SENSE (The Software Engineer's Natural-language Semantics Encoding) zur Abbildung der Bedeutung natürlicher Sprache aus Sicht eines Software-Ingenieurs vor (vgl. Gelhausen, 2010, Seite 89). SENSE schlüsselt die Bedeutung eines Satzes u.a. in eine Menge von thematischen Rollen auf (vgl. Gelhausen, 2010, Seite 97). Gelhausen gibt zu diesem Zweck 69 thematische Rollen an (vgl. Gelhausen, 2010, Seite 227 - 229). Außerdem wird eine systematische Vorgehensweise zur Konstruktion der thematischen Rollen vorgeschlagen (vgl. Gelhausen, 2010, Seite 82 f.). In der Arbeit von Gelhausen findet sich keine Verknüpfung thematischer Rollen mit thematischen Rastern. Ähnlich zur Arbeit von Girardi und Ibrahim setzt auch Gelhausen eine vollständige natürlichsprachliche Beschreibung im Erzählstil (narrativ) voraus. Dieser Text wird nicht auf Vollständigkeit und Konsistenz geprüft.

4.5.3 Konstruktion von Anwendungsfallbeschreibungen

Rolland und Achour schlagen ein Verfahren zur strukturierten Erstellung von textuellen Beschreibungen von Anwendungsfällen vor (vgl. Rolland und Achour, 1998, Seite 126). Bei der Beschreibung eines Anwendungsfalls komme den einzelnen Aktionen und damit den entsprechenden Verben eine besondere Rolle zu (vgl. Rolland und Achour, 1998, Seite 130). Die Ableitung eines speziellen Satzes von thematischen Rollen findet sich in dieser Arbeit jedoch nicht. Dennoch geben Rolland und Achour basierend auf Fillmores Tiefenkasus zwei Kategorien von thematischen Rastern an (vgl. Rolland und Achour, 1998, Seite 136):[43]

- **Action Clause Semantic Patterns:** thematische Raster für atomare Aktionen
- **State Clause Semantic Patterns:** thematische Raster zur Beschreibung statischer Beziehungen zwischen zwei Objekten, z.B. ein Zustand (state), eine Verortung (localisation) oder ein Besitzverhältnis (ownership).

Als einziges konkretes thematisches Raster wird eine *Communication Action* mit den thematischen Rollen AGENT, OBJECT, SOURCE und DESTINATION angegeben. Der

[43] Im zitierten Beitrag von Rolland und Achor werden thematische Raster als *Clause Semantic Patterns* bezeichnet.

AGENT ist der agierende Teilnehmer des Anwendungsfalls, der das OBJECT von einer SOURCE an die DESTINATION übermittelt. Daneben existiert noch ein sehr abstraktes thematisches Raster zur Abbildung beliebiger Aktionen mit den thematischen Rollen AGENT und OBJECT. Der AGENT verrichtet die Aktion mit oder an dem OBJECT (vgl. Rolland und Achour, 1998, Seite 137). Eine strukturierte Aufzählung konkreter Verben zu diesen thematischen Rastern wird nicht angegeben. Ebenso finden sich keine weiteren Beschreibungen von thematischen Rastern.

4.5.4 Diskussion

Wie in diesem Abschnitt dargelegt worden ist, haben thematische Rollen durchaus Einzug in verschiedene Forschungsvorhaben mit software-technischen Anwendungen gefunden. Bzgl. der Recherchefrage 5 ist die Existenz thematischer Rollen folglich zu bejahen. Die Arbeiten von Girardi, Ibrahim und Gelhausen liefern eine Fülle verschiedener Rollen, auf deren Basis die Definition thematischer Raster zur Identifikation von Stille in API-Verträgen erfolgen kann.

Trotzdem ist eine spezielle Untersuchung der in API-Verträgen genutzten thematischen Rollen erforderlich. Bereits das in Abschnitt 4.3.2 erweiterte thematische Raster einer *suchenden Operation* enthält eine Rolle, die in keiner der vorgestellten Sammlungen enthalten ist: ORDERING. Diese Beobachtung lässt darauf schließen, dass für die sehr spezielle Domäne der Spezifikation von Operationen weitere spezifische thematische Rollen erforderlich sind. Im Gegensatz zu den Rollen wurden thematische Raster aus software-technischer Sicht bisher nur in sehr rudimentärer Form von Rolland und Achour beschrieben. Mit Blick auf Recherchefrage 5 ist daher zu resümieren, dass bisher keine thematischen Raster als Grundlage für die Gestaltung von API-Verträgen veröffentlicht worden sind.

4.6 Identifikation thematischer Rollen für API-Verträge

Wie der vorangegangene Abschnitt gezeigt hat, kommen thematische Rollen bereits in verschiedenen software-technischen Problemstellungen zum Einsatz. Trotzdem ist noch keine Aussage zur Eignung der bisher diskutierten Sätze an thematischen Rollen für die Identifikation von Stille in API-Verträgen möglich. Dieser Abschnitt beschreibt die systematische Konstruktion eines Satzes thematischer Rollen für genau diesen Zweck. Die

hier konstruierten thematischen Rollen bilden die Grundlage für die spätere Ableitung thematischer Raster zur Beantwortung der Forschungsfrage 3.

4.6.1 Vorgehensweise

Die thematischen Rollen, sowohl von Gelhausen als auch von Girardi und Ibrahim, sind auf Basis der Analyse natürlichsprachlicher Sätze identifiziert worden. Dieses Vorgehen erscheint bei thematischen Rollen für API-Verträge nicht sinnvoll. Wie das Beispiel ORDERING zeigt, können die gesuchten thematischen Rollen einen sehr speziellen Charakter haben. Spezifische Rollen für API-Verträge sind daher wahrscheinlich nicht in derartigen Sätzen zu identifizieren. Aus diesem Grund erscheint es sinnvoll, erneut eine Querschnittstudie auf Basis des seekda-Korpus durchzuführen. Dabei erfolgt die Identifikation spezieller thematischer Rollen auf Basis der Operationen. Es wird angenommen, dass jedes Signaturelement der Operation eine thematische Rolle belegt bzw. Teil ihrer Belegung ist. Selbiges gilt für möglicherweise vorhandene Kommentare der Operationen. Die von Girardi und Ibrahim vorgeschlagenen Rollen bilden eine echte Untermenge der Rollen von Gelhausen. Daher werden die 69 Rollen von Gelhausen als Startmenge verwendet. Das genaue Vorgehen ist wie folgt:

1. Erstelle eine Liste aller 186.741 Operationen des seekda-Korpus und entferne Dubletten.

2. Mische die Liste zufällig und spalte sie in Pakete mit je 50 Operationen auf. Zu einer Operation gehören ihr Bezeichner, die vollst. WSDL-Input- sowie -Output-Message und die definierten -Faults (Ausnahmen).

3. Annotiere jedes Signaturelement jeder Operation manuell mit genau einer thematischen Rolle (definiere gegebenenfalls eine neue, passende Rolle). Falls vorhanden, so annotiere die Kommentare derartig, dass jede Phrase Teil der Belegung genau einer thematischen Rolle ist.

4. Identifiziere weitere Angaben, die zur Konzeption mindestens eines Testfalls des Web Service notwendig sind. Ein Beispiel für eine solche Angabe ist der Speicherort, von dem ein erzeugtes Objekt zwecks Prüfung gelesen werden kann. Annotiere auch diese Angaben mit einer thematischen Rolle.

5. Erfasse pro Paket die Anzahl der neu identifizierten thematischen Rollen. Analysiere so lange neue Pakete, bis der Rollenzuwachs drei Mal 5% oder weniger beträgt.

Paket	$\sum$ Rollen	Neue Rollen	Zuwachsrate	Ign. Operationen
0	69	69	0	0
1	84	15	0,22	1
2	90	6	0,07	2
3	95	5	0,06	1
4	97	2	0,02	2
5	97	0	0,00	2
6	98	1	0,01	5

Tabelle 4.8: Identifikation thematischer Rollen mit wachsender Korpusgröße

Der Umfang der Analysepakete von je 50 Operationen und das Abbruchkriterium der Zuwachsrate von 5% sind auf Basis von Erfahrungswerten gewählt worden. Die Annotation der thematischen Rollen erfolgte durch den Verfasser dieser Arbeit.

4.6.2 Thematische Rollen für API-Verträge

Das Abbruchkriterium der Analyse ist nach der Auswertung von 300 Operationen, also sechs Paketen zu je 50 Operationen, erfüllt worden. 13 Operationen wurden bei der Analyse ignoriert, weil sie nicht über einen englischsprachigen Bezeichner verfügen. Die Analyse ergab die Identifikation von 29 neuen thematischen Rollen, sodass der Gesamtumfang der bekannten thematischen Rollen für software-technische Anwendungsgebiete von 69 auf 98 Rollen ansteigt. Die kumulative Zahl der identifizierten Rollen in Abhängigkeit von der Größe des Korpus ist in Abbildung 4.7 dargestellt. Ergänzend dazu gibt die Tabelle 4.8 die Rollenzuwachsrate und die Anzahl der ignorierten Operationen pro Analysepaket an. Die Rollen stellen die Grundlage für alle weiteren Untersuchungen dar. Sie werden in Abschnitt C.1 des Anhangs mitsamt einer kurzen Beschreibung aufgelistet.[44]

4.6.3 Diskussion

Von den 98 bekannten Rollen konnten 44 im seekda-Korpus beobachtet werden. Die Vermutung hat sich bestätigt, dass thematische Rollen für API-Verträge einen sehr spezifischen Charakter haben. Dies ist beispielsweise anhand der Rollen FORMULA und

[44] Für die von Gelhausen übernommenen thematischen Rollen ist ebenfalls der dortige Beschreibungstext übernommen worden (siehe Gelhausen (2010), Seite 227 - 229).

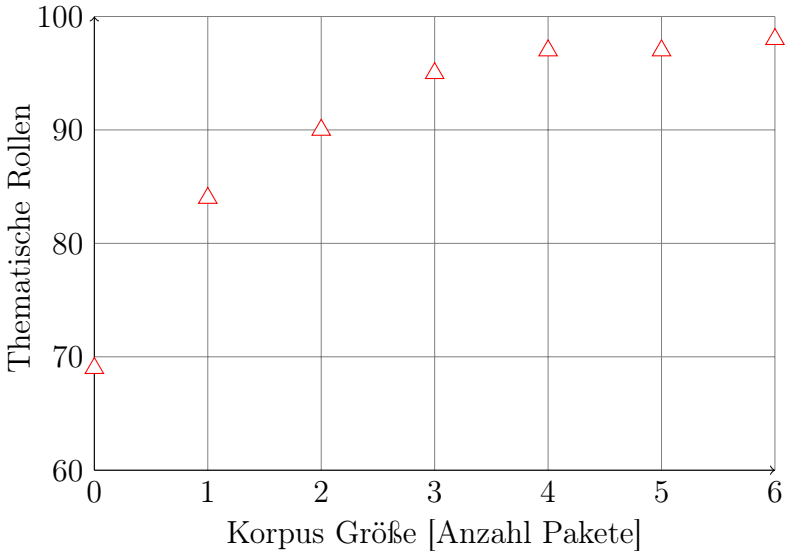


Abbildung 4.7: Identifikation thematischer Rollen mit wachsender Korpusgröße

OPERAND erkennbar. Die Beschreibung mathematischer Terme in narrativen natürlichsprachlichen Sätzen ist eher selten.[45] Zur Spezifikation einer Berechnungsoperation ist eine derartige Angabe allerdings meist unerlässlich. Ähnlich verhält es sich z.B. mit den Rollen USERNAME, PASSWORD, TRANSACTION, CHECKSUM, PRIORITY oder SESSION. Auch sie haben einen sehr software-spezifischen Charakter.

Eine weitere interessante Beobachtung ist, dass die Verwendung bestimmter thematischer Rollen nicht allein aus dem Verb herrührt. Wie in Abschnitt 4.3.1 dargestellt, lässt sich die Verwendung der Rollen ORDERING oder LIMIT aus dem Numerus der Belegung der Rolle OBJECT ableiten. Die Verwendung der Rollen USERNAME, PASSWORD und SESSION ist hingegen in sehr vielen Fällen abhängig vom gewählten Systementwurf. Einige der betrachteten Web Services benötigen eine Verknüpfung des jeweiligen Operationsaufrufes mit einer aktiven Sitzung eines bestimmten Benutzers, z.B. zwecks Authentifizierung. Dies kann entweder mit der wiederholten Übergabe von Benutzernamen und Passwort mit jedem Operationsaufruf geschehen. Alternativ dazu kann auch einmalig eine Operation `logIn` zur Eröffnung der Sitzung aufgerufen werden. Diese liefert eine eindeutige Sitzungskennung zurück. Bei den weiteren Operationsaufrufen muss diese

[45] Selbstverständlich hängt die Häufigkeit der Erwähnung mathematischer Terme neben der Art auch vom Thema des Textes ab. Dennoch sind diese Rollen von keinem der recherchierten Autoren beschrieben worden. Dieser Umstand deutet bereits darauf hin, dass diese thematische Rolle nicht in hoher Anzahl in textuellen Spezifikationen (im Fall von Gelhausen) bzw. Handbüchern zu UNIX-Befehlen (im Fall von Girardi und Ibrahim) vorkommt. Dies wird als Indiz für ihren besonderen Charakter gewertet.

Sitzungskennung dann als Parameter übergeben werden. Je nachdem für welche Variante sich der Architekt entscheidet, ist die entsprechende Verwendung der thematischen Rollen USERNAME, PASSWORD und SESSION zu beobachten.

Ähnlich verhält es sich mit der Rolle TRANSACTION. Eine Belegung dieser Rolle ist z.B. dann erforderlich, wenn eine über mehrere Teilnehmer verteilte Transaktion bearbeitet werden soll. Die Einhaltung bestimmter Konsistenzanforderungen im Rahmen einer Transaktion ist eine Entwurfsentscheidung, die sich aus den Anforderungen des Auftraggebers, aber nicht aus einer linguistischen Eigenschaft eines Bezeichners ableitet. Möglicherweise existieren weitere Wissensbasen, aus denen sich relevante API-Vertragsinhalte ableiten lassen. Hier zeigt sich, dass noch viel Raum für zukünftige Forschungsvorhaben besteht.

4.7 Identifikation thematischer Raster für API-Verträge

Die im vorangegangenen Abschnitt beschriebene Querschnittstudie hat 44 thematische Rollen zur Gestaltung von API-Verträgen ergeben. Zur Beantwortung der Forschungsfrage 3 müssen diese bestimmten thematischen Rastern zugeordnet werden. Zuvor gilt es diese thematischen Raster jedoch zu ermitteln. Dieser Abschnitt beschreibt die Ermittlung der thematischen Raster.

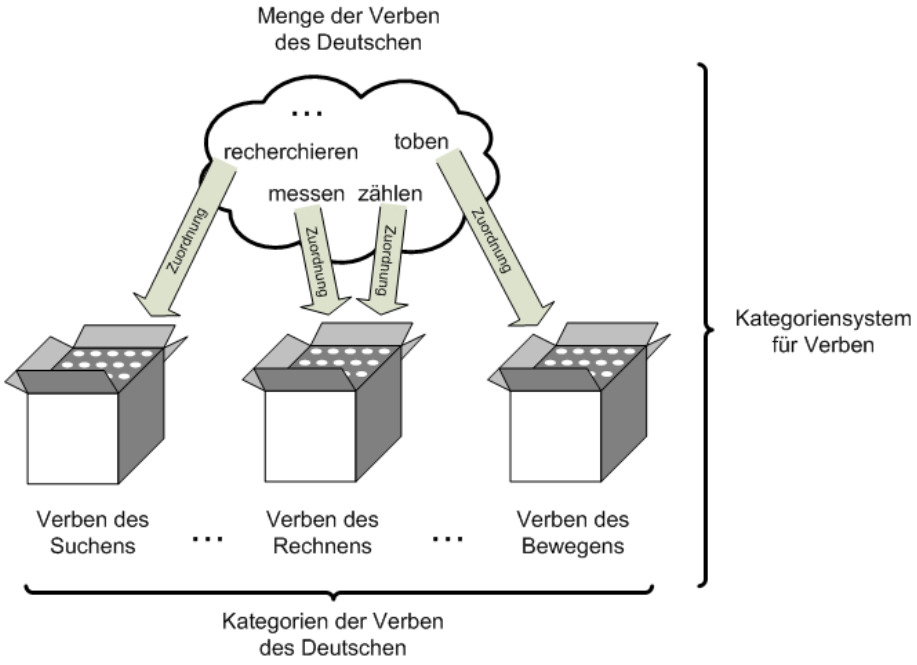


Abbildung 4.8: Prinzip eines Kategoriensystems für Verben

Als Ausgangspunkt für die Definition thematischer Raster soll ein Kategoriensystem für Verben dienen. Das Prinzip eines solchen Kategoriensystems ist in Abbildung 4.8 dargestellt: Es fasst inhaltlich ähnliche Verben zu Gruppen zusammen. Sowohl in der Linguistik als auch in der Organisationslehre sind derartige Kategoriensysteme in den letzten Jahren entwickelt worden. Auf Basis eines solchen Kategoriensystems werden im Folgenden thematische Raster für API-Verträge definiert. Jede Verbkategorie des Systems entspricht dabei einem thematischen Raster oder anders ausgedrückt: einer Operationskategorie. Ein bereits diskutiertes Beispiel für ein thematisches Raster ist die Kategorie der suchenden Operationen. Die thematischen Raster unterscheiden sich untereinander bzgl. der Charakteristika ihrer zugehörigen Operationen. Eine suchende Operation hat anders als eine speichernde Operation (beides Beispiele für thematische Raster) keinen Zielspeicher, der durch die Operation verändert wird (Charakteristikum als Belegung der thematischen Rolle DESTINATION).

Im Folgenden werden zuerst verschiedene derartige Kategoriensysteme aus der Literatur vorgestellt und auf ihre Eignung für die Zwecke dieser Arbeit hin überprüft. Im Anschluss daran wird ermittelt, welches der sich eignenden Kategoriensysteme die meisten der identifizierten Verben aus den Operationsbezeichnern abdeckt.[46] Auf dieser Basis erfolgt anschließend die Definition der thematischen Raster für API-Verträge.

4.7.1 Kategoriensysteme für Verben

Im Folgenden werden drei bestehende Kategoriensysteme für englische Verben kurz vorgestellt. Aus diesen Kandidaten wird anschließend eines auf Basis empirischer Kriterien ausgewählt. Dieses Klassifikationssystem wird die Basis des zu konzipierenden Kataloges thematischer Raster für die Gestaltung von API-Verträgen darstellen.

FrameNet

Das FrameNet-Projekt strebt den Aufbau eines Lexikons an, in dem englische Wörter und insb. deren Valenzen dokumentiert sind. Jeder Eintrag des Lexikons besteht aus drei Teilen: eine Beschreibung der Bedeutung des Wortes, die Argumente des Wortes und Beispiele in Form von annotierten natürlichsprachlichen Sätzen. Die Bedeutungen eines Wortes werden durch sogenannte Frames abgebildet (vgl. Baker et al., 1998, Seite 86

[46] Die in diesem Abschnitt vorgestellten Ergebnisse sind teilweise bereits in dem Konferenzbeitrag von Krause (2010) vorgestellt worden.

f.). Für den Zweck dieser Arbeit ist es hinreichend, das Konzept eines Frame mit dem eines thematischen Rasters gleichzusetzen. In der FrameNet-Version 1.5 konnten 1.019 Verbkategorien (Frames) und 4.670 Verben identifiziert werden.

MIT Process Handbook

Malone, Crowston und Herman stellen das MIT Process Handbook (MIT PH) als Grundlage zur Organisation und Standardisierung von Wissen zu Geschäftsabläufen und Dienstleistungen vor. Das MIT PH ist eine Online-Bibliothek, die über 5.000 Geschäftsaktivitäten in einer Hierarchie organisiert (vgl. Malone und Herman, 2003, Seite 221 f.). Diese Hierarchie bildet Generalisierungen- und Spezialisierungen der Verben untereinander ab. Außerdem sind mögliche Verwendungen eines Verbs in verschiedenen geschäftlichen Aktivitäten dokumentiert. Ferner können Verben auch detailliert werden, so dass erkennbar ist, aus welchen Untertätigkeiten sich eine Tätigkeit zusammensetzt (vgl. Malone et al., 2003b, Seite 15 - 17). Die Wurzel der Verbhierarchie stellt das Verb *to act* dar. Dieses Verb wird auf der folgenden Ebene in die acht weiteren Verben *to create*, *to modify*, *to preserve*, *to destroy*, *to combine*, *to seperate*, *to decide* und *to manage* verfeinert (vgl. Malone und Herman, 2003, Seite 247 - 249). Diese Verben sind aus dem Verbbestand des Lexikons WordNet extrahiert worden (vgl. Malone und Herman, 2003, Seite 249 f.). In der gesamten Hierarchie des MIT Process Handbook konnten insgesamt 502 Verben identifiziert werden, die den acht generischen Verben untergeordnet sind.[47]

VerbNet

Kipper-Schuler schlägt das Lexikon VerbNet für den Einsatz in Software-Anwendungen mit Bezug zur Sprachverarbeitung vor. VerbNet systematisiert die Verben auf Basis der von Levin für das Englische identifizierten Verbkategorien (vgl. Kipper-Schuler, 2005, Seite 28).[48] Anders als FrameNet oder das MIT Process Handbook ist dieses Kategorien-System somit ein Ergebnis linguistischer und nicht empirischer Analysen (vgl. Levin, 1993, Seite 14 - 17). Korhonen und Briscoe erweitern Levins Kategorien-System um 57 neue Kategorien,

[47] Diese 502 Verben sind mit freundlicher Genehmigung maschinell aus der Online-Version des MIT Process Handbook vom 02.04.2010 unter `http://process.mit.edu` extrahiert worden.

[48] Die Verbklassen von Levin sind in Levin (1993) katalogisiert.

welche ebenfalls Eingang in VerbNet finden.[49] So enthält die im Folgenden verwendete Version VerbNet 3.0 insgesamt 270 Hauptkategorien und 5.733 Verben.[50]

4.7.2 Empirische Auswahl eines Kategoriensystems

Aus den drei zuvor beschriebenen Kandidaten gilt es nun das am besten passende Kategoriensystem für Operationen auszuwählen. Die Bewertung der Kandidaten geschieht anhand zweier Kriterien. Zum einen ist die Anzahl der im seekda-Korpus identifizierten Verben von Interesse, die von den Kandidaten klassifiziert werden kann. Klassifizieren bedeutet in diesem Zusammenhang die Zuordnung eines Verbs zu einer Verbkategorie. Ein Kategoriensystem kann ein Verb demnach klassifizieren, wenn es das betreffende Verb in eine seiner Kategorien einordnen kann. Zum anderen ist aufgrund der exponentiellen Verteilung der Verben zu prüfen, wie viele Operationsbezeichner jeweils klassifiziert werden können. Schließlich ist es wenig hilfreich, wenn zwar viele Verben durch die Kategorien klassifiziert werden, aber diese Verben nur sehr selten Verwendung finden. Die Prüfung erfolgt pro Klassifikationssystem wie folgt:

1. Prüfe für jedes der 830 identifizierten Verben, ob es durch das betrachtete Kategoriensystem klassifiziert werden kann.

2. Sollte ein Verb nicht klassifizierbar sein, so prüfe, ob mind. ein in WordNet gespeichertes Synonym klassifiziert werden kann. Falls ja, so wird das ursprüngliche Verb als klassifiziert behandelt.

3. Zähle die klassifizierten Verben.

4. Wiederhole die Prozedur mit den 150.188 betrachteten Operationen anstelle der 830 Verben. Klassifiziere in diesem Fall das Prädikat aus dem Operationsbezeichner. Zähle die klassifizierten Operationen.

Die Resultate der Untersuchung sind in Tabelle 4.9 dargestellt.[51] VerbNet erreicht bei diesem Vergleich das beste Ergebnis. 90% der identifizierten Verben und 98% der

[49] Vgl. Kipper-Schuler (2005), Seite 99 und Korhonnen und Briscoe (2004), Seite 44.

[50] Bei dieser Zählung sind zwei gleich geschriebene Verben aus verschiedenen Kategorien auch als zwei verschiedene Verben gewertet worden. Bei Unterscheidung ausschließlich auf der lexikographischen Ebene sind 3.965 Verben enthalten.

[51] Der in Krause (2010) veröffentlichte Wert für die durchschnittliche Anzahl der Katgeorien pro Verb für das MIT Process Handbook weicht aufgrund eines früheren Messfehlers von dem hier genannten Wert um 0,06 ab.

Kennzahl:	**MIT PH**	**VerbNet**	**FrameNet**
Anzahl klassifizierter Verben	644	744	729
Anzahl klassifizierter Operationen	147.454	147.890	138.729
Anteil klassifizierter Verben	0,78	0,90	0,88
Anteil klassifizierter Operationen	0,98	0,98	0,92
Durchschn. Anzahl Kategorien pro Verb	3,44	1,25	6,13

Tabelle 4.9: Klassifikation der Operationen des seekda-Korpus durch MIT Process Handbook (MIT PH), VerbNet und FrameNet

Operationen insgesamt werden in VerbNet durch mind. ein thematisches Raster abgebildet. Die Anzahl der klassifizierten Operationen ist auch im MIT Process Handbook mit ebenfalls 98% und FrameNet mit 92% verhältnismäßig hoch. Beide Kategoriensysteme schneiden jedoch bzgl. der klassifizierten Verben mit 78% (MIT Process Handbook) bzw. 88% (FrameNet) deutlich schlechter ab als VerbNet. Diese Beobachtung lässt sich mit der exponentiellen Verteilung der Verbhäufigkeiten im seekda-Korpus erklären. Demnach deckt bereits eine geringe Anzahl von Verben eine hohe Anzahl von Operationen ab. Für die 99 häufigsten Verben wurde jeweils eine absolute Häufigkeit von 100 oder mehr Operationen ermittelt. Unter diesen Verben ist die Situation wesentlich ausgeglichener. VerbNet und das MIT Process Handbook klassifizieren jeweils 92 und FrameNet 91 Verben. Auch bzgl. der klassifizierten Operationen liegen die Kandidaten bei ausschließlicher Betrachtung der ersten 99 Verben dicht beieinander. Das MIT Process Handbook ist mit 139.382 klassifizierten Operationen Spitzenreiter, gefolgt von VerbNet mit 139.307 und FrameNet mit 130.427 klassifizierten Operationen. Erst unter Berücksichtigung der verbleibenden 731 Verben setzt sich VerbNet gegenüber den anderen Kandidaten durch. Diese Verben machen ca. 88% aller identifizierten Verben aus, werden aber nur in 6% der Operationsbezeichner des seekda-Korpus verwendet.

Einen weiteren Unterschied von VerbNet zu den anderen Kategoriensystemen stellt die sehr geringe Streuung der klassifizierten Verben über die Kategorien dar. Ein Verb wird in VerbNet durchschnittlich in nur 1,25 Kategorien klassifiziert. Das MIT Process Handbook und FrameNet sind mit durchschnittlich 3,44 bzw. 6,13 Klassifizierungen wesentlich mehrdeutiger. Daher ist VerbNet als Grundlage für die Definition von thematischen Rastern für API-Verträge gegenüber dem MIT Process Handbook und FrameNet zu bevorzugen.

Auch mit dem gewählten Klassifikationssystem VerbNet verbleiben 86 im seekda-Korpus identifizierte Verben, die nicht abgebildet werden können. 45 dieser Verben können durch eines der anderen Kategoriensysteme in eine Kategorie eingeordnet werden. Hier besteht folglich die Möglichkeit gegebenenfalls Teile dieser Kategoriensysteme zur Konstruktion weiterer thematischer Raster zu übernehmen. Sieben der verbleibenden 41 Verben sind wahrscheinlich fehlerhaft als Prädikat in den jeweiligen Bezeichnern erkannt worden (*to eat*, *to enlighten*, *to smile*, *to single*, *to school*, *to anagram* und *to wobble*). Die verbleibenden 34 Verben sind durchaus als Teil eines Operationsbezeichners vorstellbar (beispielsweise *to optimize*, *to mirror*, *to relocate*, *to cancel* oder *to handle*) und sollten gegebenenfalls in den konstruierten thematischen Rastern ergänzt werden.

4.7.3 Empirische Ableitung der thematischen Raster

Wie im vorangegangenen Abschnitt dargelegt worden ist, bildet VerbNet die Grundlage der zu konstruierenden thematischen Raster für API-Verträge. Die Konstruktion erfolgt auf Basis der 300 zwecks Identifikation der thematischen Rollen manuell annotierten Operationen. Wie in Abschnitt 4.6.2 beschrieben, sind 13 Operationen aus dieser Menge ignoriert worden, weil ihre Bedeutung nicht eindeutig aus den Signaturen und Verträgen hervorgegangen ist. Aus diesem Grund sind 287 Operationen im weiteren Verlauf analysiert worden. Das Vorgehen dazu lautet wie folgt:

1. Klassifiziere jede Operation anhand des als ACTION annotierten Verbs aus dem Bezeichner in VerbNet. Mehrere mögliche Klassifikationen sind erlaubt.

2. Ordne die annotierten thematischen Rollen der Operation den jeweiligen Verbkategorien automatisch zu.

3. Prüfe manuell für jede Operation, welche der möglichen Verbkategorien repräsentativ für diese Operation sind. Nicht repräsentative Kategorien sind zu ignorieren. Passende Kategorien werden gegebenenfalls zusammengefasst.

Die 287 betrachteten Operationen lassen sich automatisch 63 Verbkategorien aus VerbNet zuordnen. Die manuelle Revision hat zum Ausschluss von 40 Kategorien geführt, wie z.B. von amuse-31.1 (Verben der Unterhaltung), banish-10.2 (Verben der Ausweisung von Personen) oder consider-29.9 (Verben des Überlegens). Sieben der verbleibenden 23 Kategorien konnten aufgrund starker semantischer Ähnlichkeit mit anderen Kategorien zusammengelegt werden. So hat diese Analyse 16 VerbNet-Kategorien als thematische

Thematisches Raster:	**Namen der urspr. VerbNet-Kategorie(n):**
Prüfende Operation	investigate-35.4, peer-30.3, sight-30.2
Konvertierende Operation	convert-26.6.2, Turn-26.6.1
Traversierende Operation	swarm-47.5.1
Erzeugende Operation	create-26.4, engender-27
Beschreibende Operation	characterize-29.2
Duplizierende Operation	transcribe-25.4
Verbindende Operation	establish-55.5
Lesende Operation	get-13.5.1
Berechnende Operation	multiply-108
Zusammenführende Operation	mix-22.1
Deponierende Operation	put-9.1
Löschende Operation	destroy-44, remove-10.1
Suchende Operation	Search-35.2, obtain-13.5.2
Sendende Operation	send-11.1, tell-37.2
Initiierende Operation	begin-55.1
Substituierende Operation	exchange-13.6

Zusätzlich identifizierte Raster (in Projekten der AKRA GmbH)
Parsende Operation, Rücksetzende Operation, Schlussfolgernde Operation, Protokollierende Operation

Ignorierte VerbNet-Kategorien (siehe Schritt 3):
exchange-13.6, throw-17.1, patent-101, contribute-13.2, bill-54.5, Acquiesce-95, amuse-31.1, appoint-29.1, banish-10.2, base-97.1, braid-41.2.2, bump-18.4, calibratable_cos-45.6, carry-11.4, confine-92, consider-29.9, declare-29.4, discover-84, dress-41.1.1, dub-29.3, exist-47.1, fire-10.10, force-59, grow-26.2, image_impression-25.1, indicate-78, learn-14, meander-47.7, other_cos-45.4, performance-26.7, preparing-26.3, register-54.1, roll-51.3.1, run-51.3.2, say-37.7, scribble-25.2, slide-11.2, sound_existence-47.4, split-23.2, spray-9.7, transfer_mesg-37.1.1

Tabelle 4.10: Ergebnis der Studie zur Ableitung thematischer Raster für API-Verträge

Raster ergeben. Der Anhang C.2 gibt eine detaillierte Darstellung dieser inklusive vier weiterer Raster, die während des Einsatzes des Konzeptes in Projekten der AKRA GmbH identifiziert worden sind. Die daraus resultierenden 20 thematischen Raster sind zur Beantwortung der Forschungsfrage 3 in Tabelle 4.10 dargestellt. In den bisherigen Beispielen sind zwecks Veranschaulichung stets deutschsprachige Namen für die thematischen Rollen gewählt worden. Im Folgenden werden ausschließlich die englischsprachigen Namen aus der soeben ermittelten Menge verwendet.

4.8 Evaluierung der ermittelten thematischen Raster

Nach der Ableitung der thematischen Raster für API-Verträge stellt sich die Frage, wie gut diese skalieren. Sind sie bereits repräsentativ und damit auch praktisch verwendbar? Immerhin basieren die Raster auf einer Analyse von nur 287 Operationen. Dieser Abschnitt untersucht zur Beantwortung der Forschungsfrage 4, inwieweit die ermittelten Raster die Operationen des gesamten seekda-Korpus klassifizieren. Diese Untersuchung folgt bei ihrer Durchführung exakt den Schritten, die auch mit VerbNet, FrameNet und dem MIT Process Handbook vollzogen worden sind (siehe Abschnitt 4.7.2). Die Tabelle 4.11 zeigt in der Spalte “API-Raster” die mit den neuen thematischen Rastern erzielten Werte. Zum besseren Vergleich sind zusätzlich die Werte der anderen Kategoriensysteme angegeben.

Die 20 neu abgeleiteten Raster beinhalten 479 Verben und klassifizieren auf dieser Basis ca. 90% der betrachteten Operationen. Dieser Wert liegt unterhalb der Abdeckungsraten der anderen Kategoriensysteme. Vor dem Hintergrund des Umfang der API-Raster ist er aber dennoch beachtlich. Im Vergleich mit den beiden Spitzenreitern VerbNet und FrameNet ist der Bestand an Verben in den neuen Rastern wesentlich geringer. VerbNet enthält etwa 10,7- und FrameNet 8,7-mal so viele Verben wie die neuen Raster. Einzig das MIT Process Handbook klassifiziert mit 32 weniger Verben knapp 1% mehr Operationen. Allerdings ist ebenso anzumerken, dass das MIT Process Handbook nur über 8 thematische Raster verfügt. Folglich ist die Anzahl der Mehrfachklassifikationen eines Verbs mit ca. 3,44 im Verhältnis zu den neuen Rastern mit 2,31 relativ hoch. Noch deutlicher ist dieser Sachverhalt bei FrameNet mit durchschnittlich 6,13 Klassifikationen pro Verb. Je feiner die Klassifikationen der Verben sind, desto präzisere Hinweise auf Stille auf Basis thematischer Rollen sind möglich. Hier ist die geringe Streuung bei der Verbklassifikation in VerbNet mit durchschnittlich 1,25 Kategorien pro Verb besonders hervorzuheben.

Kennzahl:	**API-Raster**	**MIT PH**	**VerbNet**	**FrameNet**
Anzahl klassifizierter Verben	479	644	744	729
Anzahl klassifizierter Operationen	136.663	147.454	147.890	138.729
Anteil klassifizierter Verben	0,58	0,78	0,90	0,88
Anteil klassifizierter Operationen	0,91	0,98	0,98	0,92
Durchschn. Anzahl Kategorien pro Verb	2,31	3,44	1,25	6,13
Anzahl enthaltener Verben	545	502	5733	4670
Anzahl enthaltener Raster	20	8	270	1019

Tabelle 4.11: Vergleich von VerbNet, FrameNet und dem MIT Process Handbook mit den neu abgeleiteten thematischen Rastern

VerbNet verfügt allerdings auch über den höchsten Verb- und den zweithöchsten Rasterbestand.

Zusammenfassend ist festzustellen, dass ein sehr gutes Skalierungsverhalten der neu abgeleiteten thematischen Raster beobachtet werden kann. Durch die Auswertung von nur 287 annotierten Operationen konnten über 90% der Operationen des seekda-Korpus klassifiziert werden. Dabei erreicht das neue Verfahren mit dem zweitgeringsten Bestand an Verben die zweitgeringste Streuung bei der Klassifikation der Operationen über verschiedene Kategorien (im Durchschnitt 2,3 Kategorien pro Operation). Es stellt damit einen sehr zufriedenstellenden Kompromiss zwischen dem Umfang des Verblexikons und der Abdeckung der Operationen dar.

Abschließend stellt sich die Frage, wie repräsentativ diese Ergebnisse für die Grundgesamtheit aller Operationen in APIs sind. Die Erhebungen sind schließlich nur auf Basis des seekda-Korpus erfolgt. An dieser Stelle sind weitere Studien auf Basis anderer Korpora notwendig. Trotz der hohen Anzahl klassifizierter Operationen sind die abgeleiteten thematischen Raster sehr wahrscheinlich ergänzungsbedürftig. Bei diesen Ergänzungen sind zu einem großen Anteil Raster zu erwarten, die besonders domänenspezifische Operationen klassifizieren, z.B. Operationen zur Bezahlung der Inanspruchnahme eines Dienstes. Des Weiteren ist anzumerken, dass die Identifikation sowohl der thematischen Rollen als auch der Raster ausschließlich durch den Autor dieser Arbeit und damit durch nur eine Person

erfolgt ist. Es kann keine Aussage darüber getroffen werden, inwieweit andere Personen die Rollen und Raster mit sinnvollen Belegungen assoziieren. Ferner ist festzustellen, dass diese Studien lediglich die Klassifikation der Operationen in den neu abgeleiteten Rastern betrachtet haben. Die Auswirkungen auf die Vollständigkeit von Verträgen durch den Einsatz dieser thematischen Raster müssen noch untersucht werden. Hierfür sei auf Kapitel 6 verwiesen.

4.9 iDocIt! - Ein Editor für API-Verträge

Nach der Ableitung spezifischer thematischer Raster für API-Verträge stellt sich die Frage, wie Autoren sie zur Formulierung eines Vertrages einsetzen können. Dieser Abschnitt stellt den zu diesem Zweck entwickelten Editor *iDocIt!* vor.[52] Ziel dieses Abschnitts ist weniger die Darstellung software-technischer Implementierungsdetails, sondern mehr die Veranschaulichung der Arbeit mit thematischen Rastern.[53] *iDocIt!* wird im weiteren Verlauf dieser Arbeit zur Evaluierung der thematischen Raster eingesetzt (siehe Kapitel 6).

Gemäß den Oracle *javadoc*-Konventionen "gehören" die Verträge eines API dessen Programmierern. "Gehören" bedeutet in diesem Zusammenhang, dass die Programmierer für die Erstellung, die Pflege und die Löschung der Verträge zuständig sind (vgl. Oracle Corp., ohne Jahr, Abschnitt "Who Owns and Edits the Doc Comments"). Aus diesem Grund ist *iDocIt!* als Plugin für die Entwicklungsumgebung *Eclipse* konzipiert worden. In der aktuellen Version 0.0.11 unterstützt es die Bearbeitung von *javadoc*- und *WSDL*-Verträgen. Über den Plugin-Mechanismus von *Eclipse* können jedoch weitere Programmier- und Auszeichnungssprachen hinzugefügt werden (vgl. Meier-Eickhoff, 2011, Seite 52 - 61).

Die Funktionsweise von *iDocIt!* aus Benutzersicht soll im Folgenden anhand eines Fallbeispiels veranschaulicht werden. Dieses Fallbeispiel ist in den Kontext eines fiktiven Online-Buchversandhändlers einzuordnen. Zu spezifizieren ist die Operation `findRecommendations` der Schnittstelle `CustomerCareService`, welche eine Liste von Kaufempfehlungen für den übergebenen Kunden berechnet. Die Beschreibung dieser Schnittstelle liegt als Java-Datei in *Eclipse* vor. Nach dem Öffnen dieser Datei präsentiert *iDocIt!* dem Autor den in Abbildung 4.9 dargestellten Dialog. Er teilt sich in drei Bereiche auf. Oben links sind die Operationen der Schnittstelle in einer Baumansicht dargestellt. Der

[52] *iDocIt!* steht auf `http://idocit.googlecode.com` unter der Apache-Lizenz 2.0 zum Download bereit.

[53] Eine sehr gute Darstellung der Architektur und Implementierung von iDocIt! gibt Meier-Eickhoff in Meier-Eickhoff (2011) Diese Bachelor-Thesis ist im Rahmen dieses Dissertationsprojektes betreut worden.

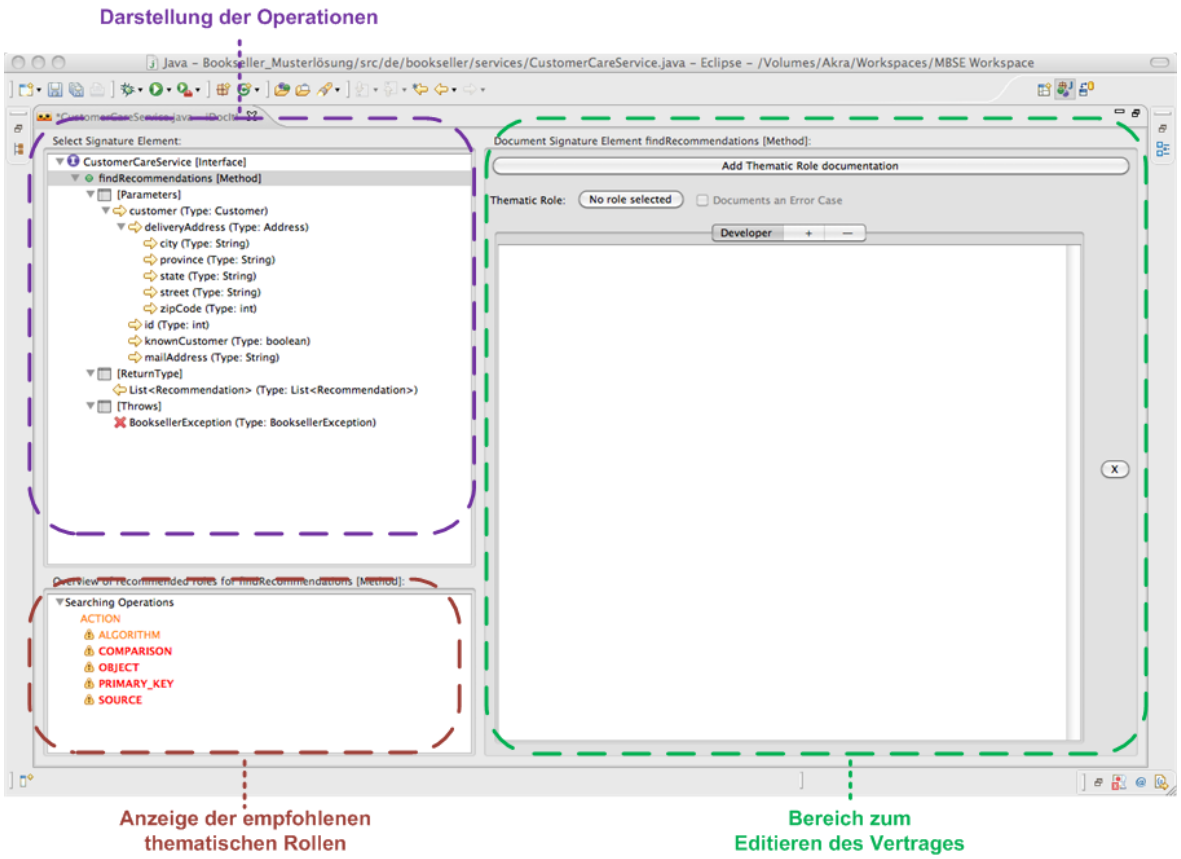


Abbildung 4.9: Initialer Zustand von iDocIt! nach dem Öffnen der Schnittstelle AccountingService

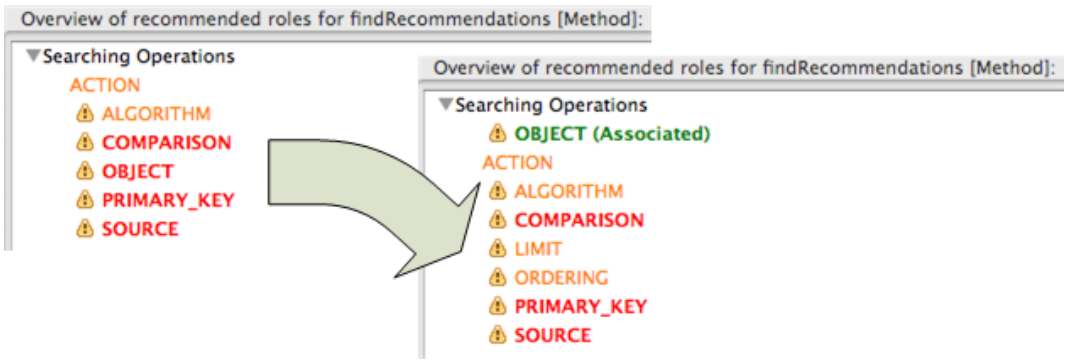


Abbildung 4.10: Aktualisierung der empfohlenen thematischen Rollen nach Spezifikation des OBJECTs

Wurzelknoten repräsentiert die Schnittstelle. Darunter sind auf erster Ebene die Operationen und auf zweiter Ebene deren Parameter, Resultate und Ausnahmen angegeben. Sofern deren Datentypen eine innere Struktur aufweisen, wird auch diese in Form weiterer Kindknoten dargestellt. In dieser Baumansicht kann der Autor per Mausklick dasjenige Signaturelement selektieren, welches er anschließend spezifizieren möchte.

Rechts daneben befindet sich der Editierbereich. Hier erfolgt die Spezifikation der Bedeutung des selektierten Signaturelementes in einem Freitextfeld. Zusätzlich kann der Autor die zugehörige thematische Rolle auswählen. Sofern zum selektierten Signaturelement mehrere verschiedene Rollen erfasst werden sollen, kann der Autor per Mausklick auf den Knopf "Add thematic role documentation" ein weiteres Freitextfeld mit eigener Auswahl einer thematischen Rolle hinzufügen. Unterhalb des Baumes der Signaturelemente zeigt *iDocIt!* die empfohlenen thematischen Raster und Rollen für die selektierte bzw. zur Selektion gehörige Operation in einem Listenfeld an. Im aktuellen Beispiel wird das thematische Raster "Searching Operation" (suchende Operation) aufgrund des Verbs *to find* im Operationsbezeichner angegeben. Die thematischen Rollen werden in verschiedenen Farben und Formatierungen angezeigt:

- **Farbe Orange:** Diese thematische Rolle ist fakultativ und unspezifiziert.
- **Farbe Rot:** Diese thematische Rolle ist obligatorisch und unspezifiziert.
- **Farbe Grün:** Diese thematische Rolle ist belegt und damit spezifiziert.
- **Formatierung Fett:** Diese thematische Rolle ist obligatorisch.
- **Formatierung Normal:** Diese thematische Rolle ist fakultativ.

Sofern das Verb des Operationsbezeichners mehreren thematischen Rastern zugeordnet ist, muss der Autor das passende Raster durch Überlegung auswählen. Das gewählte Raster kann er dann über ein Kontextmenü als *Referenzraster* markieren. Danach werden die Empfehlungen auf Basis der rasterbasierten Regeln des Referenzrasters neu berechnet und angezeigt. Ein Beispiel dafür zeigt die Abbildung 4.10. In diesem Fall ist die zurückgelieferte Liste von Kaufempfehlungen als OBJECT markiert worden (erkennbar an der grünen Schriftfarbe und dem Postfix "Associated"). Die rasterbasierten Regeln des thematischen Rasters *Suchende Operation* definieren, dass die Angabe eines LIMIT und einer ORDERING zu empfehlen ist, sobald das OBJECT den Numerus Plural annehmen kann.

Da dies bei einer Liste der Fall ist, werden die Empfehlungen durch *iDocIt!* entsprechend ergänzt.

Auch das in Abschnitt 4.3.3 beschriebene Verfahren zur Identifikation von möglichen Ausnahmen und Fehlern wird von *iDocIt!* unterstützt. Im Listenfeld der empfohlenen thematischen Rollen wird vor fast allen thematischen Rollen ein ACHTUNG-Symbol angezeigt (siehe Abbildung 4.10). Dieses Symbol weist den Autor darauf hin, dass die Nichtverfügbarkeit der jeweiligen Rollenbelegung noch nicht spezifiziert ist. Was soll beispielsweise passieren, wenn die SOURCE der Kaufempfehlungen nicht verfügbar ist? Um die Spezifikation einer Ausnahme bzw. eines Fehlers kenntlich zu machen, muss das Häkchen *Documents an Error Case* gesetzt werden. Ein mit *iDocIt!* erstelltes, vollständiges Beispiel für diesen Vertrag ist in Listing 4.8 angegeben.

Sofern eine thematische Rolle einem Signaturelement zugeordnet worden ist, für das *javadoc* kein spezifisches Tag definiert, wird ein neues Tag aus dem Bezeichner der thematischen Rolle definiert (z.B. *@algorithm* für die Rolle ALGORITHM). Gibt *javadoc* bereits ein Tag vor, so erscheint die thematische Rolle in eckigen Klammern direkt nach dem *javadoc*-Tag (z.B. in *@return* [OBJECT]). *javadoc* sieht für jedes Signaturelement nur eine Beschreibung vor. Da einem Signaturelement mehrere thematische Rollen zugeordnet werden können, liefert *iDocIt!* zu den Tags *param*, *return* und *throws* drei neue gleichnamige Tags mit dem Präfix *sub* (z.B. *@subparam* für das Tag *@param*). Im Fall eines Parameters können so Attribute seines Datentyps im Vertrag der Operation spezifiziert werden (z.B. das Attribut *id* des Parameters *kunde*).

4.10 Zusammenfassung

Das vorliegende Kapitel beschreibt einen neuen Ansatz zur Identifikation von Stille in API-Verträgen. Die Grundidee dieses Verfahrens besteht darin, Operationen auf Basis des Prädikates ihres Bezeichners zu kategorisieren. Zu jeder dieser Kategorien sind in einem Katalog generische Charakteristika der enthaltenen Operation hinterlegt. Die Tabelle 4.12 gibt die Zusammenfassungen der Antworten auf die aufgeworfenen Forschungsfragen wider. Ein großer Vorteil dieses Verfahrens ist, dass es bereits zu einem sehr frühen Zeitpunkt des Entwurfes bzw. der Implementierung eingesetzt werden kann: sobald die Signatur einer Operation vorliegt. Ferner ist es nicht auf die Analyse von Quelltexten angewiesen. Damit ist es für alle Beteiligten der in Abschnitt 3.1 beschriebenen Szenarien nutzbar. Außerdem ist es auch manuell durchführbar. Letztendlich genügt das Nachschlagen des

```
/**
 * @algorithm Um die Empfehlungen zu erzeugen, wird das
 *  k-Nearest-Neighbour-Verfahren eingesetzt. Der Zentroid-
 *  Vektor leitet sich dabei aus dem Kunden der gegebenen
 *  Bestellung ab.
 * @source Die Alternativvektoren stammen aus dem Data
 *  Warehouse.
 *
 * @param kunde
 * @subparam id [PRIMARY KEY]
 *
 * @return [OBJECT]
 * @subreturn [ORDERING] Die Empfehlungen sind absteigend
 *   nach Relevanz sortiert. Der Eintrag mit dem Index 0 ist
 *   demnach der relevanteste, usw.
 * @subreturn [LIMIT] Es werden max. 5 Empfehlungen zurück-
 *   geliefert.
 *
 * @throws BooksellerException [ERROR PRIMARY KEY]
 *   WENN customer null ist oder customer.id ungültig ist
 *   Fehlercode: 12
 *   Fehlernachricht: Kundenstammdaten liegen nicht vor
 * @subthrows BooksellerException [ERROR SOURCE]
 *   WENN das Data Warehouse nicht verfügbar ist
 *   Fehlercode: 07
 *   Fehlernachricht: Data Warehouse ist nicht verfügbar
 *
 * @thematicgrid Suchende Operation
 */
public List<Recommendation> findRecommendations
 (Customer customer) throws BooksellerException;
```

Listing 4.8: Vertrag der Operation findRecommendations

Prädikates im Lexikon der thematischen Raster. Diese Untersuchung hat gezeigt, dass das Lexikon der erforderlichen thematischen Rollen nach wenigen Erweiterungen einen gewissen Sättigungsgrad erreichen kann. Folglich ist zu erwarten, dass der Bedarf an weiteren thematischen Rollen und Rastern auch in den verschiedenen Anwendungsgebieten mit wenig Aufwand saturiert werden kann.

Damit wird allerdings ein großer Nachteil dieses Ansatzes sichtbar: Es gibt nicht den einen allgemeingültigen Satz an thematischen Rollen und Rastern. Auch an einer einheitlichen Methodik zur Identifikation und Abgrenzung thematischer Rollen mangelt es.[54] Je nach Anwendungsgebiet eines Systems werden spezifische Anpassungen bzw. die Definition gänzlich neuer Raster erforderlich sein. Des Weiteren gibt das Verfahren keinen Aufschluss über die Kardinalitäten der thematischen Rollen: Wie viele SOURCEs nutzt eine bestimmte suchende Operation? Befinden sich die Datensätze z.B. alle in einer oder mehreren verschiedenen Datenbanken? Wie viele OPERANDs sind für eine FORMULA notwendig? Auch diese Frage bietet Raum für spätere Forschungsarbeiten.

Unterm Strich bleibt festzuhalten, dass die Qualität der Empfehlungen durch thematische Raster sehr stark von der Wahl semantisch gehaltvoller Verben in den Operationsbezeichnern abhängt. Wie die Häufigkeitsverteilung der Verben des seekda-Korpus zeigt, scheinen viele Programmierer gerade bei der Wahl eines solchen Verbs Unterstützung zu benötigen. Auf diese Weise könnte die besonders häufige Verwendung von *to get* vermieden werden. Hier besteht folglich noch Raum für zukünftige Forschungsarbeiten. So stellt sich z.B. die Frage, ob und wie Programmierer bei der Formulierung eines Bezeichners unterstützt werden können. Vielleicht durch Assistentensysteme, die semantisch wenig gehaltvoll formulierte Bezeichner erkennen und Verbesserungsvorschläge anbieten können? Eine erste Formulierung derartiger Regeln liefert diese Arbeit in Abschnitt 4.3.4. Des Weiteren wäre es interessant zu untersuchen, ob sich durch eine Erweiterung der Wissensbasis zusätzliche bzw. noch detailliertere Empfehlungen für den betreffenden Vertrag ableiten lassen. Falls ja, wie müsste die Wissensbasis dafür erweitert werden?

[54] Siehe zum Beispiel Dowty (1989), Seite 69 - 71.

Forschungsfrage 1: Können Bezeichner auch bei der Vollständigkeitsprüfung eines API-Vertrages helfen?
Ja. Ein Verb kann durch Forderung von weiteren Argumenten (thematische Rollen) seine Bedeutung spezifizieren (Valenz). Enthält ein Operationsbezeichner ein Verb, so beschreiben die von diesem Verb geforderten thematischen Rollen einzelne Charakteristika der Operation. Die thematischen Rollen eines Verbs werden in einem thematischen Raster katalogisiert. Sofern aus dem Vertrag der Operation keine Belegung für eine der vom Verb geforderten thematischen Rollen hervorgeht, ist dies ein Anzeichen für Stille.
Forschungsfrage 2: Werden verschiedenartige Verben in signifikanter Häufigkeit zur Bildung von Operationsbezeichnern verwendet?
Sowohl als auch: In einem Korpus von 186.580 Web Service-Operationen konnten in 80,9% der Operationsbezeichner 830 verschiedene englische Verben identifiziert werden. Daher ist davon auszugehen, dass englische Verben häufig bei der Formulierung von Operationsbezeichnern verwendet werden. Allerdings genügt die Häufigkeitsverteilung der Verben wahrscheinlich einer Exponentialverteilung. Daher ist davon auszugehen, dass Programmierer im praktischen Einsatz Unterstützung bei der Formulierung semantisch gehaltvoller Bezeichner benötigen.
Rechercherfrage 5: Existieren thematische Rollen bzw. Raster im Kontext softwaretechnischer Arbeiten, die für die Gestaltung von API-Verträgen verwendet werden können?
Ja. Die Arbeiten von Gelhausen sowie von Girardi und Ibrahim liefern einen Umfang von 69 thematischen Rollen für software-technische Kontexte. Aber bereits mit sehr einfachen Beispielen lässt sich zeigen, dass diese Rollen nicht hinreichend für API-Verträge sind. Ein existierender Katalog thematischer Raster konnte bei den Recherchen nicht identifiziert werden.
Forschungsfrage 3: Welche thematischen Rollen und Raster sind für die Gestaltung von API-Verträgen notwendig?
Der Anhang C enthält 44 thematische Rollen und 20 Raster, die empirisch auf Basis des seekda-Korpus ermittelt worden sind. Diese Vorschläge dienen als Ausgangspunkt zur Identifikation von Stille in API-Verträgen.
Forschungsfrage 4: Wie skalieren thematische Rollen und Raster für API-Verträge?
Sehr gut. Auf Basis der Analyse von nur 287 Operationen und des Einsatzes in Projekten der AKRA GmbH konnten 20 thematische Raster ermittelt werden. Diese Raster klassifizieren über 90% der Operationen des seekda-Korpus. Außerdem erreichen sie mit einem relativ geringen Bestand an Verben eine sehr zuverlässige Klassifikation der Operationen.

Tabelle 4.12: Beantwortung der Recherche- und Forschungsfragen des Kapitels 4

Kapitel 5

Messung von Stille mit Testfällen

Um den Beitrag der im vorherigen Kapitel eingeführten thematischen Raster zur Vervollständigung von API-Verträgen quantitativ bewerten zu können, sind entsprechende Kennzahlen erforderlich. Dieses Kapitel gibt einen Überblick über publizierte Kennzahlen zur Messung von Stille und diskutiert diese vor dem Hintergrund der Szenarien aus Abschnitt 3.1. Danach stellt diese Arbeit zwei neue Verfahren zur Quantifizierung von Stille vor. Beide Verfahren wenden systematisch konstruierte Testfälle einer Operation auf deren Vertrag an.

Als Ausgangspunkt für die systematische Konstruktion von Testfällen dient die Spezifikation der Operation. Eine Präzisierung des Begriffes "Testfall" führt in diesen Teil des Kapitels ein. Anschließend erfolgt die Beschreibung der systematischen Ableitung von Testfällen. Die zwei neuen Messverfahren für Stille werden im folgenden Abschnitt detailliert dargestellt und anhand mehrerer Beispiele veranschaulicht. Das Kapitel schließt mit einer Zusammenfassung.

5.1 Wann und warum Stille messen?

In Abschnitt 3.1 sind drei Szenarien vorgestellt worden, in denen die Überprüfung der Vollständigkeit von API-Verträgen eine zentrale Rolle spielte. In Szenario 1 führt der Hersteller einer Komponente eine Selbstkontrolle durch. Auf diese Weise will er seine hohe Reputation unter Kunden weiterhin erhalten. Außerdem schützt er sich durch vollständige Verträge vor möglichen Nacherfüllungsansprüchen der Käufer seiner Komponente. In Szenario 2 prüft der Auftragnehmer die Spezifikation der von ihm zu entwickelnden Komponente auf Vollständigkeit. Erst wenn er sich der Vollständigkeit der Spezifikation

sicher ist, ist ihm eine verlässliche Kalkulation seines Angebotes möglich. In Szenario 3 führt der Käufer einer Komponente eine Vollständigkeitsprüfung der API-Verträge durch. Auf Basis einer vollständigen Spezifikation kann er die Erfüllung seiner Anforderungen durch die Komponente sicherstellen. Außerdem erhält er so einen Einblick in die Sorgfalt des Herstellers. Ein wichtiger Unterschied zwischen den Szenarien besteht in der Verfügbarkeit der Quelltexte. Während der Hersteller in Szenario 1 Zugriff auf die vollständigen Quelltexte der Komponente hat, steht diese Möglichkeit dem Auftragnehmer und dem Käufer aus den Szenarien 2 und 3 nicht zur Verfügung.

Welche Anforderungen an Kennzahlen zur Messung von realer Stille können aus diesen Szenarien abgeleitet werden? Der im ersten Szenario erwähnte Hersteller misst Stille mit dem Ziel, die Verträge seiner Komponente zu vervollständigen. Ihn interessiert nicht, inwieweit Lücken in seinen Verträgen die Arbeit Dritter beeinträchtigen. Er benötigt eine Liste mit offenen Punkten, an denen seine technischen Autoren die Verträge nachbessern müssen. Auf Basis dieser Liste erfolgt eine Aufwandsschätzung für die Nachbesserungsarbeiten. Der Hersteller interessiert sich folglich vorrangig für den Umfang der Stille.

Der Auftragnehmer des zweiten Szenarios benötigt Auskunft darüber, ob er mit der Implementierung der Komponente sofort beginnen kann oder ob noch Nachbesserungen an der Spezifikation erforderlich sind. Gegebenenfalls muss er seinem Auftraggeber ausführlich darlegen, weshalb er diese Nachbesserungen leisten und damit zusätzliche Aufwände tragen muss. Der Auftragnehmer benötigt also eine Auskunft über die Relevanz der Stille.

Der Käufer aus dem dritten Szenario hegt Interesse an zweierlei Fragen. Vor dem Kauf der Komponente interessiert ihn, mit welcher Sorgfalt deren Verträge erstellt worden sind. Daraus erhofft er sich Rückschlüsse auf die Qualität des Produktes und die Arbeitsweise des Lieferanten insgesamt. Des Weiteren möchte er wissen, ob die Komponente seine Anforderungen erfüllt. Sollte es offene Punkte geben, so benötigt er Unterstützung bei der Sammlung von zu klärenden Fragen. Ihn interessiert folglich sowohl der Umfang als auch die Relevanz der Stille.

Anforderung 1 *Die Kennzahlen müssen sowohl den Umfang als auch die Relevanz von Inhalten eines API-Vertrages quantifizieren können. Zwei Vertragsinhalte müssen aufgrund ihrer Relevanz vergleichbar sein.*

Wie die Szenarien zeigen, kann in zwei der drei Szenarien nicht davon ausgegangen werden, dass Zugriff auf den Quelltext der Komponente besteht. Gerade in solchen Fällen muss der Umfang von Stille in den zugehörigen Verträgen trotzdem messbar sein. Daher

darf die Kennzahl sowohl für die Identifikation von Stille als auch für deren Relevanzbewertung nicht auf eine Analyse der Quelltexte angewiesen sein.

Anforderung 2 *Die Kennzahl darf bei der Relevanzbewertung nicht auf Quelltexte von Benutzern oder Anbietern zurückgreifen, sondern muss eine Blackbox-Perspektive auf die Komponente einnehmen.*[1]

Zur Erfüllung der zuvor beschriebenen Anforderungen sind die folgenden zwei Forschungsfragen zu beantworten:

Forschungsfrage 5 *Wie kann der Umfang von Stille in einem Vertrag ohne Analyse des Quelltextes gemessen werden?*

Forschungsfrage 6 *Wie kann die Relevanz der Stille in einem Vertrag ohne Analyse des Quelltextes gemessen werden?*

Die Forschungsfrage 5 zielt darauf ab, die Anzahl von Lücken in einem Vertrag zu zählen. Diese Kennzahl ist wichtig, damit der Autor des Vertrages den Aufwand zur Beseitigung der Stille einschätzen kann. In Bezug auf sie könnten beispielsweise der Hersteller aus dem ersten und der Käufer aus dem dritten Szenario Interesse hegen. Die Forschungsfrage 6 fragt hingegen nach den Auswirkungen der Lücken für die Leser des Vertrages. An dieser Kennzahl dürften der Auftragnehmer und der Käufer aus dem zweiten und dritten Szenario interessiert sein.

5.2 Kennzahlen zur Messung von Stille

Wie in Kapitel 3 gezeigt worden ist, existiert eine große Zahl von Richtlinien für Vertragsinhalte. Erstaunlicherweise bleiben die meisten Beiträge eine Quantifizierung der Vollständigkeit eines Vertrages schuldig. Schreck et al. schlagen drei signaturorientierte Kennzahlen zur Bewertung der Vollständigkeit von *javadoc*-Kommentaren vor (vgl. Schreck et al., 2007, Seite 5). Diese Kennzahlen referenzieren zwar *javadoc*-Kommentare für Java-Programme, sind aber auch auf andere Programmiersprachen übertragbar.

$$ANYJ = \frac{\text{declarations with any } \textit{javadoc} \text{ comment}}{\text{total number of declarations}} \tag{5.1}$$

[1] Mit "Quelltext" ist hier nur die Beschreibung der Implementierung in einer Programmiersprache gemeint. Die formale Signatur der Operation ist in jedem Fall öffentlich und darf somit auch durch die Kennzahl ausgewertet werden.

Die Kennzahl ANYJ liefert den Anteil mit *javadoc* kommentierter Java-Methoden.[2] Diese Kommentare können allerdings auch unvollständig sein. Dies ist z.B. der Fall, wenn der *javadoc*-Kommentar einen formalen Parameter nicht mittels des Tag *@param* beschreibt. Die Vollständigkeit des *javadoc*-Kommentars bewertet die Kennzahl DIR (documented items ratio):

$$DIR = \frac{\text{documented items}}{\text{documentable items}} \tag{5.2}$$

Gemäß den *javadoc*-Konventionen ist die Beschreibung aller formalen Parameter mittels *@param* für einen vollständigen *javadoc*-Kommentar notwendig. Außerdem müssen das Resultat mittels *@return* und die zu erwartenden Ausnahmen mittels *@throws* beschrieben werden (sofern vorhanden).[3] Schreck et al. lassen bei der Definition dieser Kennzahl aus, dass Methoden-Kommentare gemäß der *javadoc*-Konventionen mit einem zusammenfassenden Satz beginnen müssen.[4] Dieser Satz ist demnach ebenfalls als *documentable item* zu zählen. Schreck et al. weisen darauf hin, dass einige Programmierer anstelle von *javadoc*- Zeilen bzw. Inline-Kommentare über Java-Methoden nutzen (z.B. "// Computes the price.").[5] Auch diese Kommentare können für Programmierer wertvoll sein, da sie z.B. Angaben über den Zweck der Java-Methode enthalten können. Die Kennzahl ANYC schließt diese Kommentare neben *javadoc*-Kommentaren ebenfalls ein:

$$ANYC = \frac{\text{methods with any kind of comment}}{\text{methods}} \tag{5.3}$$

Alle drei Kennzahlen können Werte zwischen 0 und 1 annehmen. Je höher die Werte sind, desto vollständiger sind die Kommentare. Der Vorteil dieser Kennzahlen ist, dass sie automatisiert berechenbar sind. Möglich wird ihre Erhebung durch die strikte Orientierung an der Operationssignatur. Besonders aussagekräftig ist dabei die Kennzahl DIR. Sie prüft, ob jedes Signaturelement über eine korrespondierende *javadoc*-Beschreibung verfügt. Auf diese Weise werden sehr relevante Angaben zur Nutzung der Operation wie Einschränkungen des Wertebereiches von Formalparametern und des Resultates beschrie-

[2] Gemäß der in Kapitel 2 eingeführten Terminologie können Java-Methoden sowohl Operationen (alle Methoden in Java-Schnittstellen und als abstrakt deklarierte Methoden in abstrakten Klassen) sowie Methoden (in allen sonstigen Fällen) sein.

[3] Vgl. Schreck et al. (2007), Seite 5 und Oracle Corp. (ohne Jahr), Abschnitte "Tag Conventions" und "Documenting Exceptions with @throws Tag".

[4] Dies wird gefordert in Oracle Corp. (ohne Jahr), Abschnitt "First Sentence".

[5] Die Begriffe Zeilenkommentar und Inline-Kommentar werden an dieser Stelle als Synonyme verwendet.

ben. Zusätzlich umfasst diese Kennzahl die Beschreibung der Bedingungen, unter denen Ausnahmen von der Operation geworfen werden.

Kritisch an den Kennzahlen ANYJ und ANYC ist, dass sie ausschließlich die Existenz irgendeines Kommentars prüfen. Stille innerhalb der Kommentare wird dadurch nicht abgedeckt. Dadurch sind beide Anforderungen verletzt. Etwas anders ist dies im Fall der Kennzahl DIR. Sie schließt alle dokumentierbaren Signaturelemente ein. Damit erfüllt sie zumindest teilweise Anforderung 1 bzgl. des Umfangs. Die Vollständigkeit des Vertrages wird aber ausschließlich anhand der Beschreibung aller Signaturelemente bemessen. Damit berücksichtigt DIR allenfalls nicht beschriebene Signaturelemente. Benötigte Ressourcen oder fachliche Geschäftsregeln, die nicht Teil der Signatur sind, werden nicht beachtet. Außerdem differenziert die Kennzahl DIR nicht nach der Relevanz der Signaturelemente.

All diese Kritikpunkte führen letztendlich zu dem Schluss, dass die Anforderung 1 durch keine der dargestellten Kennzahlen erfüllt wird. Damit sind sie nicht zur Messung von Stille geeignet. Diese Kritik kann bzgl. der Kennzahl DIR durch den Hinweis entschärft werden, dass der Umfang der gemessenen Stille sich nach der Definition der Menge der "items" richtet. Wenn beispielsweise der Algorithmus einer Operation ebenfalls als "item" definiert wird, erfasst diese Formel auch Stille bzgl. des Algorithmus. In diesem Fall ist das sehr naheliegend, weil jede Operation bzw. Methode einen Algorithmus implementieren muss. Auf die Existenz eines Algorithmus kann demnach auch automatisiert geschlossen werden. Aber welche Vertragsinhalte sind beispielsweise aus den von *javadoc* per Konvention geforderten Prüfpunkten "Security Constraints" oder "OS/Hardware dependencies" abzuleiten?[6] Diese Frage kann nur nach vorheriger (wahrscheinlich manueller) Analyse des Quelltextes beantwortet werden. Damit wird Anforderung 2 verletzt. Folglich werden beide Anforderungen durch keine der vorgestellten Kennzahlen erfüllt.

5.3 Begriff des Testfalls

In dieser Arbeit wird vorgeschlagen, die Stille eines Vertrages auf Basis von Testfällen zu messen. Vor der Beschreibung dieses Ansatzes ist allerdings eine Präzisierung des Begriffes "Testfall" notwendig. Myers et al. definieren das Testen von Software als den Prozess der Ausführung eines Programms mit dem Ziel, Fehler zu finden (vgl. Myers et al., 2012, Seite 6). Eine detaillierte Definition des Begriffes Testfall findet sich bei Pilorget. Nach

[6] Beide Themenbereiche werden in Smith und Kramer (2003), Abschnitt "Method Specification" gefordert.

Pilorget ist ein Testfall der Nachweis einer erfüllten Anforderung. Ein Testfall definiert die Voraussetzungen, die vor seiner Ausführung gelten müssen (z.B. dass ein vollständiger und gültiger Kundendatensatz vorliegen muss). Außerdem beinhaltet ein Testfall eine Liste der auszuführenden Testschritte auf Basis von bestimmten Testdaten. Für jeden Testschritt wird das erwartete Ergebnis vorab festgelegt (vgl. Pilorget, 2012, Seite 14 f.). Dies führt zur folgenden Definition des Begriffes Testfall für diese Arbeit:

Definition 14 (Testfall) *Ein Testfall beschreibt die Ausführung eines Programms mit dem Ziel, die Erfüllung mind. einer Anforderung nachzuweisen. Ein Testfall beinhaltet die Voraussetzungen, die vor seiner Ausführung gelten müssen. Außerdem definiert er die Testdaten und die auszuführenden Testschritte sowie die erwarteten Ergebnisse pro Testschritt.*

Diese Definition des Begriffes Testfall deutet bereits an, dass das Resultat eines Testfalls nicht ausschließlich vom Zustand der getesteten Operation bzw. Methode (also den Speicherbelegungen seiner Variablen) abhängt. Vielmehr beeinflussen die Aktualparameter und Zustände externer Ressourcen (z.B. Inhalt einer Datei oder einer Datenbanktabelle) das Ergebnis des Operationsaufrufes. All diese Einflussgrößen werden unter dem Begriff Programmsituation zusammengefasst:[7]

Definition 15 (Programmsituation) *Eine Programmsituation des Programms p beschreibt die Belegungen der internen, globalen Variablen von p sowie die Belegungen bzw. Inhalte von durch p genutzten externen Ressourcen, Komponenten und Systemen.*

Zur Veranschaulichung dient der in Tabelle 5.1 enthaltene Testfall für die im Listing 5.1 beschriebene Operation `speichereKunde`. Dieser Testfall prüft, ob ein konkreter Datensatz in der Kundendatenbank gespeichert wird. Um zu prüfen, ob diese Operation korrekt funktioniert, müsste sie mit allen denkbaren Kundendatensätzen aufgerufen werden. Aber wie bereits erwähnt, ist es unmöglich, die gelieferten Ergebnisse für alle möglichen Eingaben eines Programms zu überprüfen. Weshalb ist nun der Datensatz von Max Mustermann und nicht ein anderer für den Test ausgewählt worden? Myers et al.

[7] Der Leser mag einwenden, dass hier alternativ auch der gebräuchlichere Begriff des Programmzustands verwendet werden könnte. Dieser Begriff schließt jedoch nicht explizit die Zustände der benötigten Ressourcen ein. Diese Zustände determinieren aber (zusätzlich zum Programmzustand) ebenfalls den weiteren Ablauf des eigenen Programms. Der hier gewählte Begriff der *Programmsituation* schließt auch diese erweiterte Zustandsbetrachtung explizit mit ein.

Testfall 01:	
Beschreibung:	Ein vollständiger Kundendatensatz wird in der Kundendatenbank gespeichert.
Vorausetzungen:	Kundendatenbank ist verfügbar.
Testdaten:	Vollständiger Kundendatensatz: {Vorname: Max, Nachname: Mustermann, Anschrift: {Straße: Musterweg 22, PLZ: 12345, Ort: Musterstadt}}
Erw. Ergebnis:	Kundendatensatz mit zurückgelieferter ID ist in Kundendatenbank enthalten.

Tabelle 5.1: Beispiel für einen Testfall der Operation $O_{speichereKunde}$

```
1  /**
   * Speichert den übergebenen Kundendatensatz in der
3  * Kundendatenbank. Der übergebene Kundendatensatz
   * muss vollständig sein. Die Liste der erforderlichen
5  * Attribute für einen vollständigen Kundendatensatz ist
   * unter http://intranet.bookseller.de/anforderungen zu
7  * finden. Als Resultat wird die ID des gespeicherten
   * Datensatzes geliefert.
9  *
   * @param kunde Zu speichernder Kunde. Darf nicht
11 *  null und muss vollständig sein.
   * @return Primärschlüssel des gespeicherten Kunden
13 * @throws IOException
   *  Fehler 1: Kundendatenbank ist nicht verfügbar
15 * @throws IllegalArgumentException
   *  Fehler 2: Der Kundendatensatz ist null oder
17 *  unvollständig
   */
19 public int speichereKunde(Kunde kunde) throws IOException;
```

Listing 5.1: Beispiel für den Vertrag $V_{speichereKunde}$

empfehlen, Testfälle so zu wählen, dass mit möglichst wenigen Testfällen möglichst viele Programmsituationen abgedeckt werden. Aus diesem Grund werden Programmsituationen in sogenannten Äquivalenzklassen zusammengefasst. Für jede Äquivalenzklasse gilt, dass ein durch eines ihrer Elemente gefundener Fehler auch durch jedes andere Element gefunden werden muss (vgl. Myers et al., 2012, Seite 49 f.). Dies führt zur folgenden Definition (in Anlehnung an Myers et al., 2012, Seite 50):

Definition 16 (Äquivalenzklasse) *Eine Äquivalenzklasse ist eine Menge von Programmsituationen, von denen anzunehmen ist, dass sie bei der Verarbeitung durch die zu testende Operation o dieselben Fehler in o aufdecken.*

Im obigen Beispiel repräsentiert der Datensatz von Max Mustermann als Äquivalenzklasse die Menge der vollständigen Kundendatensätze. Wenn die Methode für diesen vollständigen Datensatz fehlerfrei funktioniert, ist anzunehmen, dass sie auch für jeden anderen vollständigen Kundendatensatz funktioniert.

5.4 Weshalb Testfälle als Bewertungsgrundlage nutzen?

Weshalb sollten Testfälle zur Messung von Stille in Verträgen eingesetzt werden? Per Definitionem erfüllt ein Vertrag die Aufgabe, die Rechte und Pflichten des Benutzers und des Anbieters einer Operation festzuschreiben. Der Vertrag ist als vollständig anzusehen, wenn seine Inhalte das zu erwartende Verhalten der Operation in allen möglichen Programmsituationen eindeutig festlegen. Dazu muss er alle Charakteristika der Operation bzw. Methode eindeutig darlegen. Natürlich ist es unmöglich, alle möglichen Programmsituationen einzeln zu betrachten, in denen eine Operation verwendet wird. Myers et al. bestätigen diese Einschätzung bzgl. der Konzeption von Testfällen. Deshalb empfehlen sie, Testfälle so zu definieren, dass wenige Testfälle ausreichen, um möglichst viele verschiedene Programmsituationen exemplarisch zu prüfen (vgl. Myers et al., 2012, Seite 49 f.). Derartig konstruierte Testfälle bilden aus den folgenden Gründen eine gute Grundlage für die Messung von Stille:

1. **Zur Messung des Umfangs von Stille:** Die in den Testfällen beschriebenen Voraussetzungen, Testdaten und erwarteten Ergebnisse bilden das Wirken der Operation repräsentativer ab als deren Signatur. So können durch die Operation hervorgerufene Zustandsänderungen aufgedeckt werden, z.B. in einer Datenbank. Schließlich

muss mind. ein Testfall des obigen Beispiels prüfen, ob die Kundendaten korrekt gespeichert worden sind.

2. **Zur Messung der Relevanz von Stille:** Die Häufigkeit der Bezüge auf ein bestimmtes Charakteristikum in Testfällen (in Form von Voraussetzungen, Testdaten oder erwarteten Ergebnissen) kann als Indiz für dessen Relevanz interpretiert werden. Auf Basis der Testfälle können diese Häufigkeiten gezählt werden.

Die häufigkeitsbasierte Bewertung der Relevanz von Stille ist ein anfechtbarer Ansatz. Ein hohes Gebot der Software-Entwicklung ist schließlich die Korrektheit des Programms. Selbstverständlich muss der Programmierer dafür jedes Charakteristikum beachten, weil ansonsten eine Fehlbenutzung der Operation und damit ein Fehler im Programm droht. Dieses Argument spricht für eine Gleichgewichtung aller Charakteristika. Dem ist aber entgegenzuhalten, dass z.B. für die Priorisierung der Korrektur von Programmierfehlern in Software-Projekten ähnliche Bewertungsmodelle herangezogen werden. Hier sind die zu erwartende Schadenshöhe, die Anzahl der betroffenen Nutzer oder der Grad der Nutzerbeeinträchtigung ausschlaggebend (so z.B. in Wolf und Leszak, 2007, Seite 11). Wenn ein Charakteristikum in vielen Testfällen eine Rolle spielt, dann hat es für die Operation eine zentrale Bedeutung und betrifft nicht nur einen Sonderfall. Dies kann durchaus als Indikator für eine hohe Relevanz des betreffenden Charakteristikums interpretiert werden.

Ein Beispiel für ein häufig zu beachtendes Charakteristikum ist der zulässige Wertebereich des Parameters *kunde* der Operation `speichereKunde` (siehe Listing 5.1). Dessen Wertebereich umfasst die Menge der speicherbaren Kunden. Eine Verletzung dieses Wertebereiches kann potentiell bei jedem programmierten Aufruf der zugehörigen Operation auftreten. Der Programmierer muss diese Situation daher durch geeignete Bedingungen im Kontrollfluss ausschließen. Da jeder Testfall der Operation `speichereKunde` diese auch aufrufen muss, wird auch der Formalparameter in jedem Testfall mit einem Wert oder der leeren Belegung versehen. Gemäß dem Vertragsmodells prüft die Operation die übergebenen Parameter auf Einhaltung des Vertrages.[8] Daher muss jeder Testfall den Wertebereich beachten und dieses Charakteristikum ist demnach besonders relevant. Der

[8] Der Leser mag an dieser Stelle mit Verweis auf Meyers Darstellung in Meyer (1988), Seite 116 f. einwerfen, dass der Zweck einer im Vertrag festgelegten Vorbedingung gerade die Vermeidung derartiger Prüfungen im Quelltext der zugehörigen Methode ist. Dem sei entgegnet, dass Meyer selbst im gleichen Werk auf Seite 148 f. die Einführung von Ausnahmen für diesen Fall thematisiert. Eine Ausnahme tritt nach Meyer auf, sobald eine Methode erkennt, dass sie ihre Aufgabe nicht wie vertraglich vereinbart erfüllen kann. Dies ist beispielsweise bei Verletzung der Vorbedingung der Fall.

hohe Grad der Relevanz der Formalparameter ist wahrscheinlich auch ein wichtiger Grund für die ausgeprägte Tradition, sie zu dokumentieren.

Anders verhält es sich mit der Verfügbarkeit des verwendeten Speichers, z.B. einer Datenbank. Sofern die Datenbank nicht verfügbar ist, muss eine Fehlerbehandlung stattfinden. Diese Ausnahmebehandlung betrifft weniger Testfälle als die Belegung des Formalparameters: Die Testfälle zur Prüfung der korrekten Parameterbelegung und zur korrekten Speicherung des Kunden sind hier nicht enthalten. Vielmehr geht es hier nur um diejenigen Testfälle, die die Ausgabe einer entsprechenden Fehlermeldung bei der Nichtverfügbarkeit der Datenbank erwarten. Demnach ist diese Ausnahme als Charakteristikum auch weniger relevant als der Wertebereich des Formalparameters. Das ist auch einleuchtend, denn diese Ausnahmebehandlung ist nicht Zweck der Operation. Sie tritt nur in einem Ausnahmefall auf und betrifft die aufrufenden Programme damit nur äußerst selten.

Auch bzgl. der Bewertung des Umfangs von Stille bietet der Einsatz von Testfällen gegenüber der ausschließlichen Orientierung an der Signatur einen großen Vorteil. Bei der Konstruktion der Testfälle sind die von der Operation gelieferten Resultate bzw. Zustandsänderungen zu prüfen. Dafür muss eine bestimmte Ausgangssituation in Form von Voraussetzungen und Testdaten vorliegen. Fehlen diese Angaben im Vertrag, so herrscht Stille. Auf diese Weise können von der Operation erwartete Ressourcen, die nicht an der Signatur sichtbar sind, identifiziert werden. So wird ein Erwartungshorizont konstruiert, vor dessen Hintergrund die Bewertung des Vertrages erfolgt.

Natürlich bedeutet die Konstruktion von Testfällen zusätzlichen Aufwand. Diesem Einwand ist aber zu entgegnen, dass die Testfälle nicht nur zur Überprüfung der Vollständigkeit der Verträge, sondern auch zur Korrektheit der zugehörigen Programme eingesetzt werden können. Der Hersteller aus Szenario 1 und der Auftragnehmer aus Szenario 2 können die Testfälle zur internen Qualitätssicherung der entwickelten Komponente nutzen. Und der Käufer aus Szenario 3 erhält durch die Testfälle eine Art “Eingangskontrolle” bei der Lieferung neuer Versionen zur Einhaltung seiner Anforderungen. Daher ist dieser Einwand damit zu entkräften, dass es sich bei diesen Aufwänden nur in wenigen Fällen um Mehraufwände ausschließlich zur Qualitätssteigerung der API-Vertäge handelt.

5.5 Systematische Erzeugung von Testfällen

Im vorherigen Abschnitt ist die Messung von Umfang und Relevanz von Stille in einem Vertrag auf Basis von Testfällen beschrieben worden. Dabei sind die Testfälle als ge-

geben angenommen worden, was selbstverständlich nicht der Realität entspricht. Auch erscheint es nicht sinnvoll, beliebig geartete Testfälle als Bewertungsgrundlage heranzuziehen. Vielmehr müssen die Testfälle systematisch konstruiert werden. Die Schaffung der Bewertungsgrundlage von Stille durch Testfälle erfolgt in zwei Schritten (siehe Abbildung 5.1).

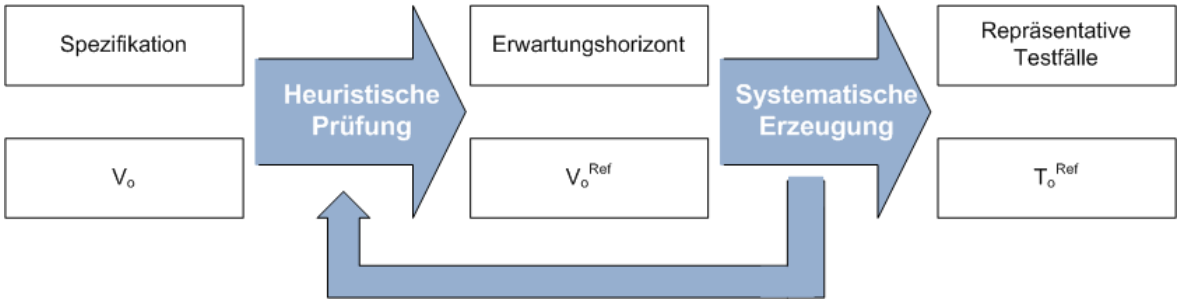


Abbildung 5.1: Prinzip der systematischen Testfallerzeugung

Den Ausgangspunkt stellt der Vertrag V_o der betrachteten Operation dar. Ziel ist es, den Umfang und die Relevanz der in ihm enthaltenen Stille zu quantifizieren. Dazu wird der Vertrag im ersten Schritt heuristisch auf Vollständigkeit und Konsistenz geprüft. Der gegebenenfalls vervollständigte Vertrag V_o^{Ref} stellt dann den Erwartungshorizont dar, von dem ausgehend die Menge der repräsentativen Testfälle T_o^{Ref} erzeugt wird. Auf dieser Basis werden dann Umfang und Relevanz der zuvor identifizierten Stille quantifiziert. Sollte während der Konstruktion der Testfälle eine weitere Lücke im Vertrag entdeckt werden, so wird in die Phase der heuristischen Prüfung zurückgesprungen.

5.5.1 Heuristiken zur Konstruktion des Erwartungshorizontes

Eine wichtige Voraussetzung für die Messung des Umfangs und der Relevanz von Stille ist deren Identifikation im Vertrag. Dafür wird allerdings eine Referenz benötigt, mit welcher der Vertrag abgeglichen werden kann. Im ersten der drei Einsatzszenarien kann der Quelltext der zugehörigen Methoden als Referenz herangezogen werden. Aber in den verbleibenden Szenarien liegt er entweder nicht vor (so in Szenario 3) oder existiert noch gar nicht (so in Szenario 2). Folglich muss die Identifikation im Zweifel ausschließlich auf Basis des Vertrages und der Signatur der Operation stattfinden. Daher werden an dieser Stelle verschiedene Heuristiken vorgestellt, mit denen Inkonsistenzen im Vertrag identifiziert werden können. Diese Heuristiken sind nicht als vollständig oder algorithmisch ausführbare Instruktionen, sondern vielmehr als Leitlinien zur Identifikation von Stille

in Form einer Prüfliste zu verstehen. Ergänzend zu diesen Heuristiken können weitere Prüflisten eingesetzt werden, z.B. die Prüfliste aus Anhang A.

- **Signatur:**
  1. Sind Formalparameter, das Resultat und zu erwartende Ausnahmen (jeweils falls vorhanden) bzgl. ihrer Bedeutung und ihres zulässigen Wertebereiches beschrieben?
  2. Ist auf Basis dieses Vertrages erkennbar, wie die Operation sich bei Belegung der Parameter mit Grenzwerten (z.B. eine leere Sammlung [bei Listen, Mengen, Zuordnungen etc.], die leere Zeichenkette "" etc.) verhalten soll?
  3. Sind `null`-Belegungen für Formalparameter möglich? Falls ja, ist das Verhalten der Operation für diesen Fall definiert?
- **Thematische Raster:**
  1. Kann der Operation ein thematisches Raster zugeordnet werden? Falls nicht: Ist es möglich, ein neues thematisches Raster für die Operation zu definieren, oder ist ihre Bedeutung unklar?
  2. Kann jeder thematischen Rolle eine Belegung aus dem Vertrag zugeordnet werden?
  3. Für jede thematische Rolle:
     - Kann es passieren, dass die Belegung dieser Rolle nicht verfügbar ist (z.B. eine nicht verfügbare Datenbank als Belegung der Rolle SOURCE)?
     - Ist das Verhalten der Operation für diesen Fall definiert?
- **Kontextwissen:**
  1. Benötigt die Operation externe Ressourcen oder Systeme? Falls ja: Ist im Vertrag definiert, wie bzw. wo diese Ressourcen und Systeme durch die Laufzeitumgebung bereitgestellt werden müssen?
  2. Ist das Verhalten der Operation für den Fall beschrieben, dass diese Ressourcen und Systeme nicht verfügbar sind?
  3. Werden gekapselte Ressourcen benötigt? Eine gekapselte Ressource ist eine Ressource aus einer Ressource, z.B. eine bestimmte Datei aus einem Verzeichnis (Verzeichnis ist Ressource, Datei ist gekapselte Ressource des Verzeichnisses).

Ein auf Basis dieser Regeln konstruierter Erwartungshorizont ist der Ausgangspunkt für die im folgenden Abschnitt beschriebene Konstruktion von Testfällen. Während der Testfallkonstruktion besteht ebenfalls die Möglichkeit der Identifikation von Stille. Dies ist beispielsweise der Fall, wenn im Fall der Nichtverfügbarkeit einer Datenbank eine entsprechende Ausnahme geworfen werden soll und diese nicht im Vertrag beschrieben bzw. in der Signatur deklariert ist. In diesem Fall erfolgt ein Rücksprung aus der Phase der Testfallkonstruktion in die Phase der heuristischen Vertragsprüfung. Gegebenenfalls ergeben sich aufgrund der neu identifizierten Stille weitere Lücken.

5.5.2 Verfahren zur systematischen Konstruktion von Testfällen

Um Stille auf Basis von Testfällen messen zu können, muss zuerst die Menge der Testfälle T_o^{Ref} aus V_o^{Ref} erzeugt werden. Myers et al. unterscheiden zwei verschiedene Kategorien von Verfahren zur Testfallerzeugung:

- **Erzeugung von White-Box-Tests:** White-Box-Tests werden mit Kenntnis des Quelltextes des zu testenden Programms entwickelt. Sie zielen darauf ab, möglichst viele Ausführungspfade und Bedingungen im Programm zu prüfen (vgl. Myers et al., 2012, Seite 42).

- **Erzeugung von Black-Box-Tests:** Black-Box-Tests werden mit Kenntnis der Spezifikation des zu testenden Programms entwickelt. Sie zielen darauf ab, Abweichungen des Programms von seiner Spezifikation zu finden (vgl. Myers et al., 2012, Seite 49).

Da die Quelltexte der Methoden nicht in allen Szenarien zur Verfügung stehen bzw. noch gar nicht existieren, werden im weiteren Verlauf dieser Untersuchung ausschließlich Verfahren zur Erzeugung von Black-Box-Tests betrachtet. V_o^{Ref} wird auf Basis eines solchen Verfahrens in die Menge von repräsentativen Testfällen T_o^{Ref} überführt. Repräsentativ bedeutet, dass die Testfälle über alle für die zu testende Operation existierenden Äquivalenzklassen gleichmäßig streuen. Im Idealfall wird jede Äquivalenzklasse durch die minimal notwendige Anzahl von Testfällen abgedeckt.

Die systematische Erzeugung von Black-Box-Testfällen ist seit vielen Jahren ein aktives Forschungsfeld mit mittlerweile vielen verschiedenen Ansätzen. Im Prinzip erzeugen alle Verfahren aus einer Spezifikation eine Menge von Testfällen. Sie unterscheiden sich

hauptsächlich im erwarteten Format der Spezifikation und in ihrem Vorgehen zur Erzeugung von Testschritten und Testdaten. Die vorgestellten Ansätze werden anhand des erwarteten Spezifikationsformates gruppiert aufgelistet.

Zustandsbasierte Verfahren

Zustandsbasierte Verfahren setzen eine Spezifikation voraus, die die möglichen Programmzustände und die entsprechenden Operationen zum Zustandsübergang beinhaltet. Howe et al. schlagen die Anwendung von Methoden zur Lösung von Planungsproblemen aus dem Bereich der künstlichen Intelligenz vor, um Testfälle zu erzeugen. Dabei hinterlegt der Benutzer die Grammatik möglicher Kommandos, den Startzustand, den Zielzustand sowie Beschränkungen von Wertebereichen und der validen Nutzung von Kommandos in einem speziellen Format (vgl. Howe et al., 1997, Seite 90 f.). Zur Testfallerzeugung werden zufällig Kommandos und Werte für deren Formalparameter selektiert. Die Auswahl der Kommandos und Werte kann beschränkt werden (vgl. Howe et al., 1997, Seite 95).

Der Ansatz von Kim et al. basiert auf UML-Zustandsdiagrammen. Nach diesem Ansatz wird das UML-Zustandsdiagramm zuerst in einen endlichen Automaten überführt. Im Anschluss daran berechnet das Verfahren alle kombinatorisch möglichen Pfade durch den Automaten (vgl. Kim et al., 1999, Seite 187). Die Erzeugung von Testdaten wird in dieser Arbeit explizit ausgeklammert (vgl. Kim et al., 1999, Seite 190).

Offutt und Abdurazik erzeugen ebenfalls Testfälle aus UML-Zustandsdiagrammen. Sie verfolgen hingegen den Ansatz einer kombinatorischen Analyse der im Diagramm modellierten Zustandsübergänge (Transitionen). Dabei ist zu gewährleisten, dass alle Transitionen spezielle Prädikate enthalten. Diese Prädikate definieren die Werte für die Variablen, bei denen die Transition feuert. Aus den Definitionen der Prädikate werden anschließend die Testdaten extrahiert (vgl. Offutt und Abdurazik, 1999, Seite 418 - 421).

Gnesi, Latella und Massink geben eine Algebra an, mit der UML-Zustandsdiagramme beschrieben werden können. Auf Basis dieser Algebra generiert ihr Verfahren zufällige valide Ausdrücke (Testschritte) mit zufällig ausgewählten validen Ein- und Ausgaben (Testdaten). Ein Ausdruck wird auf der Spezifikation und dem zu testenden Programm ausgeführt und muss zu demselben Resultat führen (vgl. Gnesi et al., 2004, Seite 81 - 83).

Vertragsbasierte Verfahren

Vertragsbasierte Verfahren gehen von der Spezifikation von Vor- und Nachbedinungen sowie Invarianten einer Operation aus.[9] Dick und Faivre schlagen einen Ansatz zur Erzeugung von Testschritten basierend auf Spezifikationen in VDM-SL (Vienna Development Method - Specification Language) vor (vgl. Dick und Faivre, 1993, Seite 268 - 270). Testdaten wählen sie regelbasiert aus den spezifizierten Wertebereichen aus. Die Regeln beschreiben dabei Grenzfälle des Wertebereiches (kleinster Wert, größter Wert etc.) (vgl. Dick und Faivre, 1993, Seite 273 f.).

Oriat vertritt einen ähnlichen Ansatz basierend auf einer Spezifikation in der Java Modeling Language zur Erzeugung von JUnit-Testfällen. Bei diesem Verfahren werden sowohl die Testschritte als auch die Testdaten grundsätzlich zufallsbasiert ermittelt. Allerdings bietet es die Möglichkeit zur Vergabe von Gewichtungen für einzelne Methoden einer Java-Klasse, um diese bei der Auswahl der Testschritte zu bevorzugen bzw. auszuschließen. Außerdem kann die Anzahl der zu erzeugenden Instanzen der getesteten Klasse festgelegt werden. Bei der Erzeugung von Testdaten für Java-Primitivtypen (int, boolean, byte etc.) orientiert sich der Ansatz ebenfalls an der Regel, zufällig Werte in den Grenzbereichen des jeweiligen Wertebereiches zu selektieren. Die Erzeugung von Testdaten für Objekttypen wird nicht beschrieben (vgl. Oriat, 2005, Seite 250 - 252).

Anwendungsfallbasierte Verfahren

Anwendungsfallbasierte Verfahren arbeiten auf dem tabellarischen Use Case-Schema nach Cockburn (vgl. Cockburn, 1997, Seite 60 f.). Fröhlich und Link setzen wie Howe et al. Planungsmethoden ein. Die Spezifikation des zu testenden Systems wird aus Anwendungsfällen nach dem Schema von Cockburn extrahiert und in ein UML-Zustandsdiagramm überführt (vgl. Fröhlich und Link, 2000, Seite 474 - 479). Auf dieser Basis erzeugt das Verfahren aus dem Zustandsdiagramm mögliche Testschritte (vgl. Fröhlich und Link, 2000, Seite 481 f.). Aus den möglichen Testschritten und den vorab manuell definierten Testdaten wählt es zufällig eine gültige Sequenz aus Testschritten mit Testdaten zur Ausführung aus (vgl. Fröhlich und Link, 2000, Seite 486 f.).

[9] Der Vertragsbegriff umfasst in diesem Zusammenhang die von Meyer geprägte ursprüngliche Bedeutung und nicht die in Abschnitt 3.2.3 beschriebenen Erweiterungen.

Relationenbasierte Verfahren

Relationenbasierte Verfahren erwarten eine Spezifikation, die der Form der relationalen Algebra genügt. Tsai et al. gehen von solch einer Spezifikation aus, die in Form von Anfragen an ein relationales Datenbankschema formuliert ist. Da in der Datenbankschema-Definition die Wertebereiche der Relationen genau beschrieben sind, kann das Verfahren umfangreiche Testdatenbanken mit Zufallswerten erzeugen. Diese sind regelbasiert derart gewählt, dass sie als Grenzfälle entweder "gerade eben" durch die Abfragen abgedeckt werden oder nicht. Die durch die Spezifikation selektierten Testdaten werden abschließend mit den vom Programm selektierten Resultaten verglichen (vgl. Tsai et al., 1990, Seite 319 f.).

Schnittstellenbasierte Verfahren

Schnittstellenbasierte Verfahren gehen ausschließlich von den Operationssignaturen einer Schnittstelle aus und erzeugen Testfälle auf dieser Basis. Ausgangspunkt der Arbeit und von Bai et al. ist die Beschreibung einer Web Service-Schnittstelle in WSDL (Web Service Description Language) und der verwendeten Datentypen repräsentiert als XML-Schema. Das Verfahren analysiert alle in der Schnittstelle enthaltenen Operationen und berechnet die Kompatibilität der Operationen untereinander anhand der Typkompatibilität ihrer Parameter und ihres Resultates. Auf Basis dieser Analyse erzeugt es Testschritte in Form von Operationsaufrufen. Die Erzeugung der Testdaten für Primitivtypen erfolgt zufällig unter Beachtung von unter anderem im XML-Schema definierten Beschränkungen und von in sogenannten Facets definierten Mustern. Im Fall strukturierter Typen befolgt das Verfahren eine Menge an Regeln (vgl. Bai et al., 2005, Seite 210 f.).

Textbasierte Verfahren

Textbasierte Verfahren erzeugen aus einer Spezifkation in Form eines natürlichsprachlichen Textes Testfälle. Myers et al. geben ein Verfahren zur manuellen, systematischen Erzeugung von Testfällen aus einer Spezifikation in Textform an. Dabei extrahiert der Tester alle vom Programm erwarteten Ein- und erzeugten Ausgaben aus dem Text. Anschließend modelliert er diese in einem Graph unter Berücksichtigung von Einschränkungen. Abschließend wird der Graph in eine Entscheidungstabelle überführt. Jede Spalte dieser Tabelle repräsentiert genau einen Testfall (vgl. Myers et al., 2012, Seite 61 f.).

Auswahl eines Verfahrens

Welches der vorgestellten Verfahren soll nun für die Messung der Relevanz von Stille eingesetzt werden? Da in dieser Arbeit vorwiegend natürlichsprachliche Verträge betrachtet werden, lässt sich die Auswahl auf die Ansätze von Myers et al. auf der einen und Bai et al. auf der anderen Seite eingrenzen. Das Verfahren von Bai et al. arbeitet auf Basis von Operationssignaturen. Wie bereits mehrfach dargelegt worden ist, besteht die Gefahr von Stille vor allem für Charakteristika, die nicht Teil der Signatur sind. Daher ist trotz eines vollständigen Vertrages V_o^{Ref} nicht zu erwarten, dass dieses Verfahren eine repräsentative Menge an Testfällen liefert. Anders sieht es mit dem Ansatz von Myers et al. aus. Dieses Verfahren verarbeitet den natürlichsprachlichen Vertrag und kann damit von der Vervollständigung von V_o^{Ref} durch die zuvor dargestellten Heuristiken profitieren. Daher wird das Verfahren von Myers et al. für diese Untersuchung eingesetzt. Im folgenden Abschnitt erfolgt eine detaillierte Beschreibung dieses Verfahrens.

Testfallerzeugung mit dem CEG-Verfahren

Cause-Effect-Graphing ist ein von Myers entwickeltes Verfahren, um aus natürlichsprachlichen Spezifikationen eine Menge an repräsentativen Testfällen zu erzeugen. Es besteht aus den folgenden durchzuführenden Schritten (vgl. Myers et al., 2012, Seite 62):

1. Aufteilung der Spezifkation in zusammengehörige und überschaubare Teilspezifikationen (workable pieces).

2. Identifikation der in jeder Teilspezifikation enthaltenen Ein- (causes) und Ausgaben (effects). Eine Eingabe kann ein einzelner Wert oder eine ganze Äquivalenzklasse sein. Jede Ein- und Ausgabe erhält eine eindeutige Nummer.

3. Überführung der Ein- und Ausgaben in einen Cause-Effect-Graph.

4. Annotation des Graphs mit Beschränkungen, die bestimmte Kombinationen von Ein- und Ausgaben aufgrund von syntaktischen, logischen oder umgebungstechnischen Gründen ausschließen.

5. Überführung des Graphs in eine Entscheidungstabelle. Jede Spalte dieser Tabelle repräsentiert einen Testfall.

6. Überführung der Spalten der Entscheidungstabelle in programmierte Testfälle.

Zur Ausführung des Verfahrens sind präzise Definitionen der Begriffe Eingabe und Ausgabe notwendig. In Anlehnung an Myers et al. gelten für diese Arbeit die folgenden Definitionen (vgl. Myers et al., 2012, Seite 62):

Definition 17 (Eingabe) *Eine Eingabe ist ein einzelner, von der zu testenden Operation gelesener und verarbeiteter Wert bzw. eine Äquivalenzklasse.*

Definition 18 (Ausgabe) *Eine Ausgabe ist ein einzelner, von der zu testenden Operation gelieferter bzw. geschriebener Wert. Als geschriebene Werte zählen auch von der Operation bewirkte Zustandsänderungen im eigenen System oder in Fremdsystemen (z.B. Datenbanken).*

Eingabe	**Wert / Äquivalenzklasse**
E_1	Vollständiger Kundendatensatz
E_2	Verfügbare Kunden-DB
Ausgabe	**Wert / Äquivalenzklasse**
A_1	Primärschlüssel (Resultat)
A_2	Kunden-DB enthält gesp. Datensatz für Primärschlüssel
A_3	Fehler 01: Kunden-DB ist nicht verfügbar
A_4	Fehler 02: Kundendatensatz ist nicht vollständig

Tabelle 5.2: Ein- und Ausgaben von $O_{speichereKunde}$

Bei der Betrachtung eines Vertrages einer Operation bzw. Methode kann der erste Schritt in der Regel entfallen, weil ein solcher Vertrag bereits einen recht überschaubaren Umfang hat. Das Listing 5.2 enthält den bereits bekannten Vertrag $V_{speichereKunde}$ mit farblich markierten Ein- und Ausgaben. Die Kundendatenbank stellt sowohl eine Ein- als auch eine Ausgabe dar. Deshalb ist sie jeweils zur Hälfte mit beiden Farben eingefärbt. Anschließend werden die Ein- und Ausgaben nummeriert und mit einem Wert bzw. einer Äquivalenzklasse belegt (siehe Tabelle 5.2).

```
/**
 * Speichert den übergebenen Kundendatensatz in der
 * Kundendatenbank. Der übergebene Kundendatensatz
 * muss vollständig sein. Die Liste der erforderlichen
 * Attribute für einen vollständigen Kundendatensatz ist
 * unter http://intranet.bookseller.de/anforderungen zu
 * finden. Als Resultat wird die ID des gespeicherten
 * Datensatzes  geliefert.
 *
 * @param kunde Zu speichernder Kunde. Darf nicht
 *  null und muss vollständig sein.
 *
 * @return  Primärschlüssel des gespeicherten Kunden
 *
 * @throws IOException
 *  Fehler 1: Kundendatenbank ist nicht verfügbar
 * @throws IllegalArgumentException
 *  Fehler 2: Der Kundendatensatz ist null oder
 *  unvollständig
 */
public int speichereKunde(Kunde kunde)
    throws IOException;
```

Listing 5.2: Vertrag $V_{speichereKunde}$ mit unterstrichenen Ein- und Ausgaben

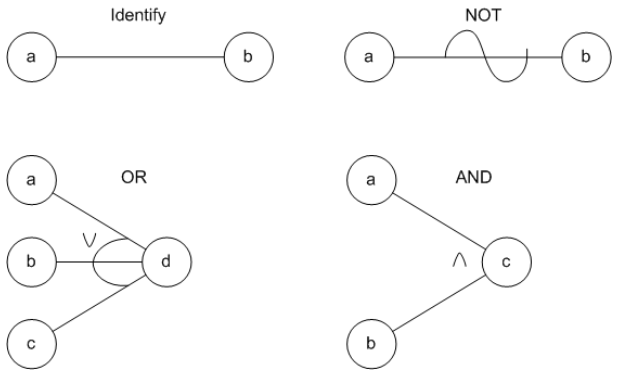


Abbildung 5.2: Symbole des Cause-Effect-Graphs

Die Liste der Ein- und Ausgaben wird nun in den Cause-Effect-Graph (CEG) überführt. Myers weist darauf hin, dass der Graph ein boole'scher Graph ist. Demnach kann jeder Eingabeknoten den Zustand "1" oder "0" annehmen. "1" bedeutet, der vom Eingabeknoten beschriebene Wert bzw. die beschriebene Äquivalenzklasse ist verfügbar. "0" bedeutet entsprechend das Gegenteil (vgl. Myers et al., 2012, Seite 62). Die Belegung der Ausgabeknoten leitet sich aus der im Graph definierten Verknüpfung der Eingabeknoten ab. Die im CEG zulässigen Symbole werden in Abbildung 5.2 dargestellt (entnommen aus Myers et al., 2012, Seite 63). Die dargestellten Funktionen entsprechen den gleichnamigen boole'schen Operatoren (vgl. Myers et al., 2012, Seite 63):

- Identify: wenn a = 1, dann b = 1, sonst b = 0
- NOT: wenn a = 1, dann b = 0, sonst b = 1
- OR: wenn a = 1 oder b = 1 oder c = 1, dann d = 1, sonst d = 0
- AND: wenn a = 1 und b = 1, dann c = 1, sonst c = 0

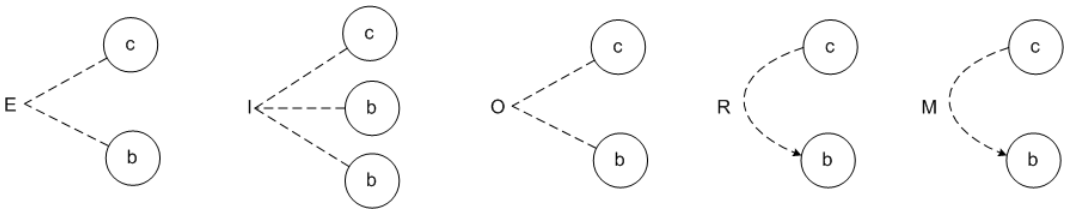


Abbildung 5.3: Symbole zur Beschränkung des Cause-Effect-Graphs

Ein CEG, der ausschließlich aus Knoten dieser Typen besteht, erlaubt, dass beide Fehler aus dem obigen Beispiel zugleich auftreten können. Jedoch ist es sinnvoller, zuerst den

Aktualparameter auf Vollständigkeit zu prüfen und gegebenenfalls die *IllegalArgument-Exception* zu werfen. Schließlich können nur vollständige Datensätze gespeichert werden. Daher gilt die Konvention, dass zuerst die Aktualparameter und dann die benötigten Ressourcen geprüft werden. Um unter anderem derartige Fälle abzubilden, sind nach Myers weitere Symbole für einen CEG notwendig. Myers führt aus diesem Grund die in Abbildung 5.3 dargestellten Beschränkungen ein (vgl. Myers et al., 2012, Seite 64 f.) [10]:

- E-Beschränkung: a und b dürfen nie gleichzeitig 1 sein.
- I-Beschränkung: a, b und c dürfen nie gleichzeitig 0 sein.
- O-Beschränkung: entweder muss a = 1 oder b = 1 sein, aber nie beide gleichzeitig.
- R-Beschränkung: wenn a = 1, dann muss b = 1 gelten.
- M-Beschränkung: wenn a = 1, dann muss b = 0 gelten.

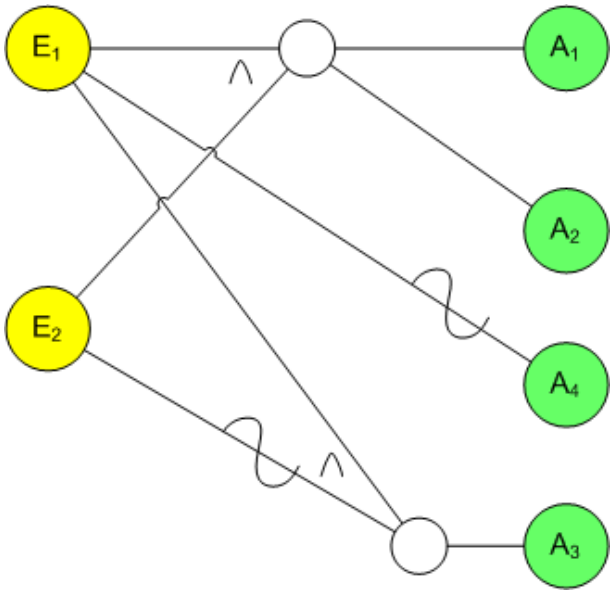


Abbildung 5.4: Cause-Effect-Graph für $O_{speichereKunde}$

Abbildung 5.4 zeigt den CEG für die Operation $O_{speichereKunde}$. Der Übersicht halber sind die Ein- und Ausgabeknoten analog zur Hervorhebung im Vertrag in gelber und grüner Farbe eingefärbt. Außerdem wurden zwei anonyme Knoten eingefügt (weiß eingefärbt),

[10] Die Abbildung 5.3 ist in Anlehnung an die Darstellung in Myers et al. (2012), Seite 64 erstellt worden.

um die Anzahl der Kanten zu reduzieren. Aus diesem Graph lässt sich die Entscheidungstabelle 5.3 ableiten. Die vier Spalten ganz rechts in der Entscheidungstabelle entsprechen den Testfällen (t_1 - t_4).

Eingabe	**Wert / Äquivalenzklasse**	t_1	t_2	t_3	t_4
E_1	Vollständiger Kundendatensatz	1	1	0	0
E_2	Verfügbare Kunden-DB	1	0	1	0
Ausgabe					
A_1	Primärschlüssel (Resultat)	1	0	0	0
A_2	Kunden-DB enthält gespeicherten Datensatz für Primärschlüssel	1	0	0	0
A_3	Fehler 01: Kunden-DB ist nicht verfügbar	0	1	0	0
A_4	Fehler 02: Kundendatensatz ist nicht vollständig	0	0	1	1

Tabelle 5.3: Aus dem CEG erzeugte Entscheidungstabelle für $O_{speichereKunde}$

In diesem Beispiel konnten die Äquivalenzklassen recht einfach bestimmt werden: Für jede Eingabe ist genau eine Äquivalenzklasse formuliert worden. Es ist bei Bedarf aber auch möglich, mehrere Äquivalenzklassen pro Eingabe zu formulieren. Denkbar ist in diesem Beispiel die weitere Äquivalenzklasse "kunde ist null" für den Formalparameter. Um die Entscheidungstabelle konsistent zu halten, müssen die verschiedenen Äquivalenzklassen pro Eingabe mit einer O-Beschränkung versehen werden. So ist gewährleistet, dass immer nur genau eine Äquivalenzklasse erfüllt ist. Die Entscheidungstabelle 5.4 enthält die Testfälle für dieses erweiterte Beispiel. Die Testfälle t_1 und t_2 sind aufgrund der Verletzung der O-Beschränkung weggefallen. Schließlich kann der Kundendatensatz nicht gleichzeitig `null` und vollständig sein.

5.6 Messung des Umfangs von Stille

Die Messung des Umfangs auf Basis von Testfällen folgt dem einfachen Prinzip des Anteilswertes. Die Grundlage bildet die in Abschnitt 5.2 vorgestellte Kennzahl *Documented Items Ratio*. Diese Kennzahl berechnet den Anteil der dokumentierten "items" von den dokumentierbaren "items". Als "item" sind die Signaturelemente der Operation, also deren Formalparameter, ihr Resultat und von ihr geworfene Ausnahmen definiert. Damit

Eingabe	**Wert / Äquivalenzklasse**	~~t_1~~	~~t_2~~	t_3	t_4	t_5	t_6	t_7	t_8
E_1	Vollständiger Kundendatensatz	~~1~~	~~1~~	1	1	0	0	0	0
E_2	Kundendatensatz ist null	~~1~~	~~1~~	0	0	1	1	0	0
E_3	Verfügbare Kunden-DB	~~1~~	~~0~~	1	0	1	0	1	0
Ausgabe									
A_1	Primärschlüssel (Resultat)	~~0~~	~~0~~	1	0	0	0	0	0
A_2	Kunden-DB enthält gesp. Datensatz für Primärschlüssel	~~0~~	~~0~~	1	0	0	0	0	0
A_3	Fehler 01: Kunden-DB ist nicht verfügbar	~~1~~	~~1~~	0	1	0	1	1	1
A_4	Fehler 02: Kundendatensatz ist nicht vollständig	~~0~~	~~1~~	0	0	1	0	0	0

Tabelle 5.4: Aus dem erweiterten CEG erzeugte Entscheidungstabelle für $O_{speichereKunde}$

berücksichtigt die Kennzahl *Documented Items Ratio* ausschließlich die von außen sichtbaren Elemente der Signatur.

Zur Messung von Stille ist diese Einschränkung zu weitreichend. Deshalb wird die Menge der "items" um die Menge aller Ein- und Ausgaben der betreffenden Operation erweitert. Die Konstruktion eines Erwartungshorizontes mit Hilfe des Cause-Effect-Graphing-Verfahrens ermöglicht diese Erweiterung. Repräsentative Testfälle decken schließlich auch solche Ein- und Ausgaben ab, die nicht an der Signatur sichtbar sind. Die Listen der Ein- und Ausgaben werden auf Basis des heuristisch vervollständigten Vertrages V_o^{Ref} erstellt (also auf Basis des Erwartungshorizontes). Anschließend erfolgt der Abgleich beider Listen mit dem ursprünglichen Vertrag V_o. Dabei wird für jede Ein- und Ausgabe geprüft, ob diese auch aus V_o hervorgeht. Tabelle 5.5 enthält die Definition der Funktionen, die diese Prüfungen abbilden. Auf dieser Grundlage werden die Kennzahlen *Grad der Eingabenvollständigkeit*, *Grad der Ausgabenvollständigkeit* oder *Grad der Vollständigkeit* (für Ein- und Ausgaben zusammen) wie in den Formeln 5.4, 5.5 und 5.6 angegeben definiert. Beide Kennzahlen können Werte zwischen 0 und 1 annehmen. Je höher die Werte sind, desto vollständiger sind die Kommentare und desto weniger Stille herrscht vor.

$$GdEV^{Umfang}_{(V_o,T_o^{Ref})} = \frac{|eingaben(V_o, T_o^{Ref})|}{|eingaben(V_o^{Ref}, T_o^{Ref})|} \tag{5.4}$$

Funktion	Bedeutung
$eingaben(v, T)$	Liefert alle Eingaben der Testfälle aus T, die sich auch aus dem Vertrag v ergeben.
$ausgaben(v, T)$	Liefert alle Ausgaben der Testfälle aus T, die sich auch aus dem Vertrag v ergeben.

Tabelle 5.5: Übersicht der in Formeln verwendeten Funktionsbezeichner

$$GdAV^{Umfang}_{(V_o,T_o^{Ref})} = \frac{|ausgaben(V_o, T_o^{Ref})|}{|ausgaben(V_o^{Ref}, T_o^{Ref})|} \tag{5.5}$$

$$GdV^{Umfang}_{(V_o,T_o^{Ref})} = \frac{|eingaben(V_o, T_o^{Ref})| + |ausgaben(V_o, T_o^{Ref})|}{|eingaben(V_o^{Ref}, T_o^{Ref})| + |ausgaben(V_o^{Ref}, T_o^{Ref})|} \tag{5.6}$$

```
/**
 * Speichert den übergebenen Kundendatensatz in der
 * Kundendatenbank. Als Resultat wird die ID des
 * gespeicherten Datensatzes geliefert.
 *
 * @param kunde Zu speichernder Kunde
 * @return Primärschlüssel des gespeicherten Kunden
 * @throws IOException
 *  Fehler 1: Kundendatenbank ist nicht verfügbar
 * @throws IllegalArgumentException
 *  Fehler 2: Der Kundendatensatz ist null
 */
public int speichereKunde_2(Kunde kunde)
    throws IOException;
```

Listing 5.3: Der Vertrag $V_{speichereKunde}$ ohne Erwähnung der Vollständigkeit

Zur weiteren Veranschaulichung folgen abschließend zwei kleine Rechenbeispiele. Angenommen für die Operation `speichereKunde_2` ist im Vertrag nicht festgeschrieben, dass nur vollständige Kundendatensätze gespeichert werden (siehe Listing 5.3). Folglich fehlt in diesem Fall die Eingabe E_1. Damit ergibt sich für den Grad der Eingabenvollständigkeit

$$GdEV^{Umfang}_{(V_{speichereKunde},T^{Ref}_{speichereKunde})} = \frac{|\{E_2\}|}{|\{E_1,E_2\}|} = 0,5$$

und für den Grad der Vollständigkeit bzgl. des Umfangs von Stille

$$GdV^{Umfang}_{(V_{speichereKunde},T^{Ref}_{speichereKunde})} = \frac{|\{E_2\}|+|\{A_1,A_2,A_3,A_4\}|}{|\{E_1,E_2\}|+|\{A_1,A_2,A_3,A_4\}|} = \frac{5}{6} = 0,8\overline{3}$$

Das Listing 5.4 enthält ein Beispiel für eine fehlende Ausgabe. In diesem Fall ist die Ausnahme A_4 nicht erwähnt, welche bei Unvollständigkeit oder `null`-Belegung des Formalparameters *kunde* geworfen wird. So ergibt sich für den Grad der Ausgabenvollständigkeit

$$GdAV^{Umfang}_{(V_{speichereKunde},T^{Ref}_{speichereKunde})} = \frac{|\{A_1,A_2,A_3\}|}{|\{A_1,A_2,A_3,A_4\}|} = 0,75$$

und für den Grad der Vollständigkeit bzgl. des Umfangs von Stille

$$GdV^{Umfang}_{(V_{speichereKunde},T^{Ref}_{speichereKunde})} = \frac{|\{E_1,E_2\}|+|\{A_1,A_2,A_3\}|}{|\{E_1,E_2\}|+\{A_1,A_2,A_3,A_4\}} = \frac{5}{6} = 0,8\overline{3}$$

Es ist erkennbar, dass beide Lücken bzgl. ihres Umfangs gleich bewertet werden. Ausschließlich an der Anzahl der ausgelassenen Ein- und Ausgaben gemessen wären beide Verträge mit $83,\overline{3}$ %, also ziemlich vollständig. Dieses Ergebnis impliziert, dass beide Angaben gleich relevant sind. Da bei dieser Rechnung ein Modell zur Berücksichtigung der Relevanz einer Ein. bzw. Ausgabe fehlt, werden diese Kennzahlen im weiteren Verlauf durch Relevanzangaben modifiziert. Eine erneute Betrachtung der hier berechneten Werte erfolgt nach der Vorstellung der modifizierten Kennzahlen.

```
/**
 * Speichert den übergebenen Kundendatensatz in der
 * Kundendatenbank. Der übergebene Kundendatensatz
 * muss vollständig sein. Die Liste der erforderlichen
 * Attribute für einen vollständigen Kundendatensatz ist
 * unter http://intranet.bookseller.de/anforderungen zu
 * finden. Als Resultat wird die ID des gespeicherten
 * Datensatzes geliefert.
 *
 * @param kunde Zu speichernder Kunde. Darf nicht
 *  null und muss vollständig sein.
 *
 * @return Primärschlüssel des gespeicherten Kunden
 *
 * @throws IOException
 *  Fehler 1: Kundendatenbank ist nicht verfügbar
 */
public int speichereKunde_3(Kunde kunde)
    throws IOException;
```

Listing 5.4: Der Vertrag $V_{speichereKunde}$ ohne Erwähnung der Ausnahme bei Unvollständigkeit des übergebenen Kunden

5.7 Messung der Relevanz von Stille

Wie können Unterschiede der Ein- und Ausgaben bzgl. ihrer Relevanz gemessen werden? In dieser Arbeit wird vorgeschlagen, die Relevanz einer Ein- bzw. Ausgabe danach zu bewerten, wie häufig sie bei der Programmierung beachtet werden muss. Selbstverständlich muss der Programmierer jede Ein- und Ausgabe beachten, weil ansonsten eine Fehlbenutzung der Operation und damit ein Fehler im Programm droht. Aber wenn z.B. eine häufig zu beachtende Eingabe vom Programmierer ignoriert wird, dann droht die Fehlbenutzung in jeder einzelnen Programmsituation, die eigentlich die Beachtung der Eingabe erfordert.

Sehr wahrscheinlich ist es unmöglich, alle Programmsituationen zu betrachten, in denen eine Operation verwendet wird. Myers et al. bestätigen diese Einschätzung bzgl. der

Konzeption von Testfällen. Deshalb empfehlen sie, Testfälle so zu definieren, dass wenige Testfälle ausreichen, um möglichst viele verschiedene Programmsituationen exemplarisch zu prüfen (vgl. Myers et al., 2012, Seite 49 f.). Da solche Testfälle per Definitionem möglichst viele Programmsituationen abdecken, stellen sie eine sehr gute Grundlage für die Relevanzbewertung von Ein- und Ausgaben dar.

Dieser Ansatz bewertet die Relevanz einer Ein- bzw. Ausgabe nach der Anzahl der Testfälle, die ohne die Ein- bzw. Ausgabe nicht entscheidbar wären. Ein Testfall t gilt auf Basis eines Vertrages V_o als entscheidbar, wenn alle Ein- und Ausgaben in V_o genannt sind, auf denen t beruht. Die Verwendung des Adjektivs *entscheidbar* wird wie folgt begründet: Für einen Testfall t kann auf Basis des Vertrages V_o entschieden werden, ob die durch t geforderten Ausgaben einer Operation o für die durch t an o übergebenen Eingaben vollständig und korrekt sind. Sobald für den Testfall relevante Ein- bzw. Ausgaben in V_o fehlen, kann keine fundierte Aussage zu dessen Vollständigkeit und Korrektheit getroffen werden. Diese Überlegung führt zur folgenden Definition der Entscheidbarkeit eines Testfalls:

Definition 19 (Entscheidbarkeit eines Testfalls t auf Basis von V_o) *Ein Testfall t einer Operation o gilt als entscheidbar, wenn alle Ein- und Ausgaben, auf denen t beruht, im zugehörigen Vertrag V_o beschrieben sind.*

Die Quantifizierung der Relevanz erfolgt nach den folgenden Schritten. Alle verwendeten Variablen- und Funktionsbezeichner sind der Übersicht halber in Tabelle 5.6 dargestellt.

1. Erstelle heuristisch einen Referenzvertrag V_o^{Ref} auf Basis von V_o, in dem die in V_o aus dem ausgelassenen Charakteristikum resultierenden Ein- und Ausgaben enthalten sind (für die Heuristiken siehe Abschnitt 5.5.1).

2. Entwickle mit dem Cause-Effect-Graphing-Verfahren systematisch eine Menge repräsentativer Testfälle T_o^{Ref} auf Basis von V_o^{Ref} (siehe Abschnitt 5.5.2).

3. Berechne die Menge der auf Basis von V_o entscheidbaren Testfälle (ET) T_o^{ET} aus T_o^{Ref}. Die Menge T_o^{ET} enthält alle Testfälle, die auf den in V_o enthaltenen Ein- und Ausgaben beruhen. Diese Testfälle sind auf Basis von V_o entscheidbar.

4. Berechne den Grad der Vollständigkeit des Vertrages V_o auf Basis der Relevanz nach der Formel 5.7.

Bezeichner	**Beschreibung**
o	Betrachtete Operation.
V_o	Potentiell unvollständiger Vertrag zur Operation o.
V_o^{Ref}	Heuristisch vervollständigter Vertrag zur Operation o.
t	Testfall.
T_o^{Ref}	Menge aller Testfälle der Operation o, die auf Basis des Referenzvertrages V_o^{Ref} erzeugt werden können.
T_o^{ET}	Teilmenge von T_o^{Ref}, die alle Testfälle enthält, die auf Basis von V_o entscheidbar sind.
$testfaelle(V)$	Funktion, die die Menge der Testfälle auf Basis des Vertrages V erzeugt.
$et(T, V)$	Funktion, die die Menge der auf Basis des Vertrages V entscheidbaren Testfälle aus T liefert.

Tabelle 5.6: Übersicht der verwendeten Variablen- und Funktionsbezeichner

$$GdV^{Relevanz}_{(V_o,V_o^{Ref})} = \frac{|et(testfaelle(V_o^{Ref}), V_o)|}{|testfaelle(V_o^{Ref})|} = \frac{|T_o^{ET}|}{|T_o^{Ref}|} \tag{5.7}$$

Auch in diesem Fall gilt, dass die Kennzahl nur Werte zwischen 0 und 1 annehmen kann. Je höher der Wert ist, desto vollständiger sind die Kommentare und desto weniger Stille herrscht vor. Mit Hilfe des im vorherigen Abschnitt vorgestellten Cause-Effect-Graphing-Verfahrens kann eine natürlichsprachliche Spezifikation systematisch in eine Menge von Testfällen überführt werden. Konkret bedeutet das, dass der vervollständigte Referenzvertrag V_o^{Ref} auf diese Weise in die Menge der Referenztestfälle T_o^{Ref} überführt wird. Anschließend kann für jeden Testfall aus T_o^{Ref} geprüft werden, ob er auf Basis von V_o entscheidbar ist. Offen geblieben ist bisher, wie die Entscheidbarkeit eines Testfalls berechnet wird. Dazu ist wie folgt vorzugehen:

1. Berechnung der Mengen $Stille_o^{Ein}$ und $Stille_o^{Aus}$. Diese Mengen enthalten jeweils Ein- bzw. Ausgaben, die in V_o^{Ref}, aber nicht in V_o stehen. Bzgl. dieser Ein- und Ausgaben herrscht in V_o Stille.

2. Berechnung der entscheidbaren Testfälle $T_o^{ET^{Aus}}$ aus T_o^{Ref}, die auf keiner Ausgabe aus $Stille_o^{Aus}$ beruhen.

3. Berechnung der entscheidbaren Testfälle $T_o^{ET^{Ein}}$ aus T_o^{Ref}, die auf keiner Eingabe aus $Stille_o^{Ein}$ beruhen.

Im Folgenden werden die letzten beiden Schritte detailliert beschrieben und veranschaulicht. Die Ableitung der Entscheidbarkeit eines Testfalls von einer Ausgabe ist etwas einfacher nachzuvollziehen. Aus diesem Grund wird mit diesem Fall begonnen.

Entscheidbarkeit bzgl. einer Ausgabe

Ein Testfall beruht auf einer Ausgabe, wenn er diese explizit prüft, z.B. durch eine `assert`-Anweisung. Zur Veranschaulichung wird daher der Testfall t_2 der Operation `speichereKunde` betrachtet (siehe Entscheidungstabelle 5.3). Dieser Testfall prüft, ob eine entsprechende Fehlermeldung bei Nichtverfügbarkeit der Kundendatenbank in Form einer Ausnahme geworfen wird. Für diesen Testfall muss sichergestellt sein, dass die Kundendatenbank abgeschaltet ist. Anschließend erfolgt der Aufruf von `speichereKunde` mit einem vollständigen Kundenobjekt. Der Test besteht in einer expliziten Überprüfung, ob die erwartete Ausnahme von der Operation geworfen worden ist. Derlei explizite Prüfungen werden durch die aus dem Cause-Effect-Graphing resultierende Entscheidungstabelle nicht abgedeckt. Die Prüfung auf den Wert "1" für die jeweilige Ausgabe ist nicht hinreichend. Es könnte ja sein, dass der Test explizit sicherstellt, dass keine Ausnahme geworfen wird. In diesem Fall wäre der betreffenden Ausgabe der Wert "0" zugeordnet, obwohl der Testfall diese Ausgabe prüft. Aus diesem Grund wird die formale Betrachtung eines Testfalls um die Angabe der Existenz einer expliziten Prüfung pro Ausgabe ergänzt (Prüfung $P_1, \ldots, P_m$ und $P \in \{0, 1\}$):

$$t = ([E_1, \ldots, E_n], [A_1, \ldots, A_m], [P_1, \ldots, P_m]) \tag{5.8}$$

Diese Form gibt explizit an, für welche Ausgabe i eine explizite Prüfung im Rahmen des Testfalls vorgesehen ist ($P_i = 1$) und für welche nicht ($P_i = 0$). Auf dieser formalen Grundlage wird die Entscheidbarkeit eines Testfalls bzgl. einer Ausgabe wie folgt definiert:

Definition 20 (Beruhen eines Testfalls auf einer Ausgabe) *Ein Testfall t beruht dann und genau dann auf einer Ausgabe A_i, wenn er A_i explizit prüft. In diesem Fall gilt $P_i = 1$.*

Auch die Relevanzbewertung einer fehlenden Ausgabe wird anhand eines kleinen Rechenbeispiels veranschaulicht. Die Entscheidungstabelle 5.7 beinhaltet die entsprechende

Eingabe	**Wert / Äquivalenzklasse**	t_1	t_2	t_3	t_4
E_1	Vollständiger Kundendatensatz	1	1	0	0
E_2	Verfügbare Kunden-DB	1	0	1	0
Ausgabe					
A_1	Primärschlüssel (Resultat)	1	0	0	0
A_2	Kunden-DB enthält gesp. Datensatz für Primärschlüssel	1	0	0	0
A_3	Fehler 01: Kunden-DB ist nicht verfügbar	0	1	0	0
A_4	Fehler 02: Kundendatensatz ist nicht vollständig	0	0	1	1
Prüfung					
P_1		1	0	0	0
P_2		1	0	0	0
P_3		0	1	0	0
P_4		0	0	1	1

Tabelle 5.7: Um explizite Prüfungen erweiterte Entscheidungstabelle für $O_{speichereKunde}$

formale Erweiterung des Beispiels um explizite Prüfungen. Auch in diesem Fall wird davon ausgegangen, dass die Ausnahme im Fall eines ungültigen Aktualparameters *kunde*, also Ausgabe A_4, im Vertrag fehlt. Da die Testfälle t_3 und t_4 die Ausgabe A_4 explizit prüfen, sind sie nicht entscheidbar. So ergibt sich

$$GdV^{Relevanz}_{(V_{speichereKunde}, V^{Ref}_{speichereKunde})} = \frac{|\{t_1, t_2\}|}{|\{t_1, t_2, t_3, t_4\}|} = \frac{2}{4} = 0,5$$

Verglichen mit der Vollständigkeit nach Umfang in Höhe von $83,\overline{3}$ % ist dies ein deutlich geringerer Wert. Zur besseren Einordnung beider Resultate werden im Folgenden zusätzlich noch die Grade der Vollständigkeit nach Umfang und Relevanz für das Fehlen der Ausgabe A_3 berechnet:

$$
\begin{aligned}
GdV^{Umfang}_{(V_{speichereKunde}, T^{Ref}_{speichereKunde})} &= \frac{|\{E_1, E_2\}| + |\{A_1, A_2, A_4\}|}{|\{E_1, E_2\}| + \{A_1, A_2, A_3, A_4\}} \\
&= \frac{5}{6} \\
&= 0,8\overline{3} \\
GdV^{Relevanz}_{(V_{speichereKunde}, V^{Ref}_{speichereKunde})} &= \frac{|\{t_1, t_3, t_4\}|}{|\{t_1, t_2, t_3, t_4\}|} \\
&= \frac{3}{4} \\
&= 0,75
\end{aligned}
$$

Es ist erkennbar, dass die Vollständigkeitsbewertung nach Umfang beiden Ausgaben dieselbe Relevanz zuordnet: $83,\overline{3}$ %. Die Vollständigkeitsbewertung nach Relevanz liefert zwei deutlich abgestufte Werte: bei Fehlen von A_3 eine Vollständigkeit von 75% und bei Fehlen von A_4 eine Vollständigkeit von nur 50%. Demnach ist A_4 deutlich relevanter als A_3. Aber ist dem wirklich so?

Als Leitlinie für die Relevanzbewertung einer Ein- bzw. Ausgabe ist die Notwendigkeit ihrer Beachtung bei der Programmierung herausgestellt worden. In diesem Fall wäre A_4 demnach tatsächlich relevanter als A_3, weil eine nicht-verfügbare Datenbank in den meisten Fällen weniger aus einem Programmierfehler, sondern vielmehr aus einem Konfigurationsfehler bzw. einem Infrastrukturproblem herrührt. Die Übergabe eines ungültigen Aktualparameters deutet hingegen auf einen Programmierfehler im Kontrollfluss hin. Daher ist letztere Angabe für den Programmierer relevanter als erstere, weil sie bei jedem Operationsaufruf beachtet werden muss. Diese Abstufung wird durch das Verfahren plausibel abgebildet.

Entscheidbarkeit bzgl. einer Eingabe

Im vorangegangenen Abschnitt wurde das Beruhen eines Testfalls auf einer Ausgabe formal definiert. Die Ableitung des Beruhens eines Testfalls auf einer Eingabe erfolgt mit Rückgriff auf diese Definition. Ausgangspunkt ist auch hier der CEG. Aus dem CEG einer Operation kann für jede Ausgabe ein boole'scher Ausdruck abgeleitet werden. Dieser Ausdruck gibt an, unter welcher Eingabekonstellation die Ausgabe auftreten soll. Die boole'schen Ausdrücke für die Operation `speichereKunde` ergeben sich aus dem in Abbildung 5.4 dargestellten CEG wie folgt:

Ausdruck	**Wert**
$\neg null$	$null$
$1 \wedge null$	$null$
$0 \wedge null$	$null$
$1 \vee null$	1
$0 \vee null$	$null$

Tabelle 5.8: Behandlung der leeren Belegung null durch erweiterte, boole'sche Operatoren

$$A_1 = E_1 \wedge E_2$$
$$A_2 = E_1 \wedge E_2$$
$$A_3 = \neg E_1$$
$$A_4 = E_1 \wedge \neg E_2$$

Auf Basis dieser Ausdrücke können die von einer Eingabe abhängigen Ausgaben berechnet werden. Dazu wird die boole'sche Algebra neben den Werten $wahr$ und $falsch$ um die leere Belegung $null$ und um die in Tabelle 5.8 angegebene Behandlung der leeren Belegung durch die boole'schen Operatoren ergänzt. Erläuterungsbedürftig scheint hier die Definition $0 \vee null = null$. In diesem Fall ergibt sich die linke Seite des Ausdrucks mit den Eingabebelegungen des jeweiligen Testfalls zu $falsch$. Daher besteht die Möglichkeit, dass der Testfall die rechte Seite zu $wahr$ auswerten sollte. Gemäß dem Prinzip 'im Zweifel nicht entscheidbar' wird das Resultat daher mit $null$ anstelle von $falsch$ belegt.

Eine Ausgabe A hängt genau dann von einer Eingabe E ab, wenn der zu A gehörige boole'sche Ausdruck auf Basis der erweiterten Semantik der boole'schen Operatoren unter Belegung von $E = null$ zu $null$ ausgewertet wird. Ergibt dieser Ausdruck $wahr$ oder $falsch$, so ist von keiner Abhängigkeit von E auszugehen. Sobald alle von E abhängigen Ausgaben bekannt sind, wird gemäß der bereits bekannten Definition geprüft, ob der Testfall auf mind. einer dieser Ausgaben beruht. Dies führt zur folgenden Definition des Beruhens eines Testfalls auf einer Eingabe:

Definition 21 (Beruhen eines Testfalls auf einer Eingabe) *Ein Testfall t beruht genau dann auf einer Eingabe E, wenn gilt $P_i = 1$ und der boole'sche Ausdruck von A_i im Fall $E = null$ zu null ausgewertet wird.*

Als Beispiel wird analog zum vorherigen Abschnitt die Relevanz der Eingabe E_1 berechnet. Dazu wird die Eingabe E_1 in den obigen boole'schen Ausdrücken mit `null` belegt. Die Eingabe E_2 wird aus Gründen des Lesbarkeit mit 1 belegt, obwohl für den Zweck dieser Auswertung auch eine Belegung mit 0 zulässig wäre.

$$
\begin{aligned}
A_1 =& null \wedge 1 = null \\
A_2 =& null \wedge 1 = null \\
A_3 =& \neg null = null \\
A_4 =& null \wedge \neg 1 = null
\end{aligned}
$$

Bei der Belegung E_1 = `null` wird jeder Ausdruck zu `null` ausgewertet. Daher ist bei Fehlen dieser Eingabe kein Testfall entscheidbar. Damit ergibt sich ein Grad der Vollständigkeit bzgl. der Relevanz von 0 %.

$$
\begin{aligned}
GdV^{Relevanz}_{(V_{speichereKunde}, V^{Ref}_{speichereKunde})} &= \frac{|\{\}|}{|\{t_1, t_2, t_3, t_4\}|} \\
&= 0,0
\end{aligned}
$$

Demnach wäre der Vertrag bei Fehlen der Eingabe E_1 zu 0% vollständig. Ist dieses Ergebnis plausibel interpretierbar? Aufgrund des Datentyps *Kunde* und des Bezeichners des Formalparameters kann jeder Programmierer darauf schließen, dass es sich hierbei um den zu speichernden Kunden handelt. Die grundsätzliche Bedeutung des Parameters ist damit klar. Aber die Operation kann nicht beliebige, sondern nur vollständige Kunden speichern. Unter welchen Bedingungen ein Kunde als vollständig gilt, ist aus der Operationssignatur nicht ersichtlich. Die Programmierer der aufrufenden Programme müssen aber sicherstellen, dass diese Operation nur mit vollständigen Kundenobjekten aufgerufen wird. Andernfalls droht eine Fehlersituation im Programm - mit jedem Aufruf von `speichereKunde`. Aus diesem Grund schweigt der Vertrag ohne die Angabe von E_1 zu einer sehr relevanten Eingabe. Diese Einschätzung wird von dieser Kennzahl im Gegensatz zur Vollständigkeitsbewertung nach Umfang korrekt widergespiegelt. Zum Vergleich: Die Vollständigkeitsbewertung gemäß Umfang schätzt die Vollständigkeit desselben Vertrages auf $83,\overline{3}$ %. Der Vertrag erscheint nach diesem Wert fälschlicherweise von recht hoher Qualität zu sein. Dieser hohen Bewertung zum Trotz kann für keinen einzigen Operationsaufruf die Vollständigkeit des übergebenen Kundenobjektes garantiert werden (weil

der Vertrag diese Einschränkung verschweigt). Insofern droht bei allen Operationsaufrufen eine vom Programmierer unverschuldete Fehlersituation.

5.8 Zusammenfassung

Die Messung von Stille dient der Qualitätskontrolle von API-Verträgen. Auf diese Weise kann das Verständnis der zu implementierenden bzw. zu integrierenden Komponenten geschärft und die Kommunikation von Geschäftspartnern im Software-Projekt unterstützt werden. Deshalb müssen Messverfahren sowohl den Umfang der Stille in einem Vertrag (Anzahl der Lücken) als auch deren Relevanz bewerten können. In der Literatur publizierte Kennzahlen erfüllen diese Anforderungen zurzeit nicht. Daher wird in dieser Arbeit vorgeschlagen, Stille auf Basis von Testfällen der Operation zu identifizieren und ihre Auswirkungen abzuschätzen. Dieses Kapitel beantwortet die zwei in Tabelle 5.9 dargestellten Forschungsfragen.

Forschungsfrage 5: Wie kann der Umfang von Stille in einem Vertrag V ohne Analyse des Quelltextes gemessen werden?
Testfälle helfen dabei, nicht an der Signatur sichtbare Charakteristika zu identifizieren. Alle Charakteristika der Operation werden gezählt. Für jedes vorhandene Charakteristikum wird geprüft, ob es aus dem Vertrag der Operation hervorgeht. Dies ermöglicht die Bestimmung des Umfangs von Stille.
Forschungsfrage 6: Wie kann die Relevanz der Stille in einem Vertrag V ohne Analyse des Quelltextes gemessen werden?
Die Relevanz des Charakteristikums bemisst sich nach der Anzahl der Testfälle, die es betrifft.

Tabelle 5.9: Beantwortung der Forschungsfragen 5 und 6

Die Erstellung von Testfällen lohnt sich nicht nur zur Kontrolle der Vertragsqualität, sondern ist auch für andere Qualitätsaspekte der Software nützlich, z.B. Konformität mit der Spezifikation und Robustheit. Voraussetzung dafür ist die systematische Erzeugung der Testfälle. In dieser Arbeit wird die Anwendung des Cause-Effect-Graphing-Verfahrens zu diesem Zweck vorgeschlagen. Somit steht ein Messverfahren für den Umfang und die Relevanz von Stille zur Verfügung, welches auch in späteren Phasen der (Weiter-)Entwicklung und des Betriebs der Anwendung einen Mehrwert liefert.

Kapitel 6

Evaluierung

Dieses Kapitel beschreibt die Untersuchungen zur empirischen Überprüfung der ermittelten thematischen Raster auf Basis der im vorherigen Kapitel entwickelten Kennzahlen. Dies erfolgt sowohl im Hinblick auf die Abdeckung der aus den Rastern abgeleiteten Empfehlungen als auch unter Berücksichtigung ihrer an Testfällen orientierten Relevanz. Zusätzlich zu einer Analyse des Potentials von thematischen Rastern zur Vermeidung von Stille wird auch die Umsetzung der Empfehlungen durch Probanden geprüft.

Der erste Abschnitt dieses Kapitels formuliert und begründet zwei entsprechende Forschungsfragen. Anschließend erfolgen die Auswahl der angewandten Forschungsmethoden und die detaillierte Darlegung des gewählten Versuchsaufbaus und -ablaufes. Im darauf folgenden Abschnitt werden die erhobenen Ergebnisse im Detail beschrieben und diskutiert. Das Kapitel schließt mit der Zusammenfassung der gewonnenen Erkenntnisse.

6.1 Zielsetzung und Forschungsfragen

In Abschnitt 4.8 ist gezeigt worden, dass die in Anhang C beschriebenen thematischen Raster einen Großteil der im seekda-Corpus vorkommenden Operationen klassifizieren. Offen ist jedoch, ob die Klassifikation einer Operation zur Vermeidung von Stille im zugehörigen Vertrag beiträgt. In diesem Abschnitt wird überprüft, inwieweit sich der niedergeschriebene Anteil relevanter Vertragsbestandteile durch thematische Raster erhöhen lässt. Diese Analyse mündet in zwei Forschungsfragen:

Forschungsfrage 7 *Können thematische Raster signifikant zur Vermeidung von Stille beitragen (konzeptuelle Perspektive)?*

Die erste Forschungsfrage zielt auf einen grundsätzlichen Gewinn an vermiedener Stille durch den Einsatz thematischer Raster ab. Dazu soll das Konzept der Raster möglichst isoliert von anderen Störeinflüssen, wie z.B. Verständnis- oder Bedienungsaspekten bezüglich des eingesetzten Werkzeugs, betrachtet werden. Diese Aspekte werden durch die zweite Forschungsfrage adressiert.

Forschungsfrage 8 *Kann ein Software-Werkzeug auf Basis thematischer Raster Programmierer bei der Vermeidung von Stille unterstützen (integrierte Perspektive)?*

Die zweite Forschungfrage schließt explizit einen Versuchsaufbau unter Berücksichtigung von Software-Werkzeugen ein. Am Beispiel von *iDocIt!* wird untersucht, wie thematische Raster Programmierer bei der täglichen Software-Entwicklung unterstützen können.

6.2 Konzeptuelle Überlegungen

6.2.1 Untersuchungsgegenstand

Beide Forschungsfragen lassen sich sehr gut durch einen Vergleich beantworten. Die erste Forschungsfrage zielt primär auf einen konzeptuellen Vergleich der Hinweise für Vertragsinhalte ab. Die verglichenen Hinweise stammen von den thematischen Rastern und konventionellen Prüflisten auf Basis von *javadoc*. Mit den im vorherigen Kapitel beschriebenen Kennzahlen werden die jeweiligen Hinweise im Bezug auf ihren Umfang und ihre Relevanz vergleichbar bewertet. Diese Vorgehensweise erlaubt eine direkte Einschätzung des jeweiligen Ansatzes zur inhaltlichen Gestaltung von Verträgen. Die zweite Forschungsfrage adressiert hingegen Zusammenhänge eines größeren Kontextes. Sie berücksichtigt, inwieweit die Autoren die Hinweise mit konkreten Vertragsinhalten assoziieren können. Was hilft es schließlich, wenn die Hinweise zwar gut gemeint, aber unverständlich sind?

Daher erscheint es in diesem Zusammenhang sinnvoller, anstelle der Hinweise vollständige Verträge zu vergleichen. Die Formulierung dieser Verträge wird entweder durch thematische Raster oder durch konventionelle Prüflisten, wie z.B. von *javadoc*, angeleitet (siehe Kapitel 3). Sowohl thematische Raster als auch *javadoc* benötigen zur Ableitung ihrer Empfehlung Operationssignaturen. Daher bietet sich erneut die Verwendung eines Korpus von Operationen an. Zu jeder Operation muss dieses Korpus die folgenden Angaben beinhalten:

- **Signatur:** Zur Signatur der Operation zählen der Operationsbezeichner, alle formalen Parameter (Bezeichner und Datentyp), der Datentyp des Resultates und alle zu erwartenden Ausnahmen bzw. Fehler.
- **Vertrag:** Semi-formale Beschreibung des Verhaltens der Operation mit natürlichsprachlichen Anteilen.
- **Ein- und Ausgaben:** Übersicht aller möglichen Ein- und Ausgaben der Operation.

6.2.2 Relevante empirische Forschungsmethoden

Es stellt sich nun die Frage, wie die Operationen des Korpus erhoben werden sollen. Zu diesem Zweck kommen verschiedene empirische Forschungsmethoden in Betracht. Vor allem in den Sozial- und Erziehungswissenschaften ist in der Vergangenheit ein breites Spektrum an verschiedenen Methoden entwickelt worden. An dieser Stelle werden daher nur diejenigen für diese Untersuchungen relevanten Forschungsmethoden kurz skizziert.

Querschnittstudie

Die Querschnittstudie als zeitgleiche Momentaufnahme einer Population in Form mehrerer Stichproben ist bereits in Abschnitt 4.4 vorgestellt worden.

Fallstudie

Yin definiert eine Fallstudie als eine empirische Forschungsmethode, die ein zeitgenössisches Phänomen im Detail und innerhalb seines natürlichen Kontextes untersucht und beschreibt. Diese Methode eignet sich besonders, wenn das Phänomen nicht klar von seinem Kontext abgegrenzt werden kann oder nur wenige Daten vorliegen (vgl. Yin, 2009, Seite 18). Meist werden für Fallstudien Daten aus Beobachtungen oder Inhaltsanalysen von Dokumenten extrahiert (z.B. Protokolle oder Zeitungsartikel) (vgl. Borchardt und Göthlich, 2009, Seite 37 und 42). Fallstudien sind besonders zur Entwicklung von Hypothesen oder Erklärungsmodellen für spätere Untersuchungen komplexer, bisher wenig bekannter Phänomene geeignet (vgl. Borchardt und Göthlich, 2009, Seite 46).

Laborbeobachtung

Die Beobachtung im Allgemeinen ist eine Datenerhebungstechnik, die weitestgehend dem intuitiven Verständnis des Begriffs *Beobachtung* entspricht. Nach Jahoda et al. gilt ei-

ne Beobachtung als wissenschaftlich, wenn sie einem bestimmten Forschungszweck dient. Sie muss ferner systematisch geplant und darf nicht zufällig sein. Sie bedarf einer systematischen und nicht zufälligen Aufzeichnung und sie muss regelmäßig bezüglich ihrer Gültigkeit, Zuverlässigkeit und Genauigkeit kontrolliert werden (vgl. Jahoda et al., 1972, Seite 77). Die Bezeichnung Laborbeobachtung leitet sich aus der Laborumgebung ab, in der diese Form der Beobachtung durchgeführt wird (vgl. Schnell et al., 2011, Seite 382 f.).

Experiment

Das Experiment ist eine empirische Forschungsmethode zur Überprüfung von Hypothesen mittels statistischer Verfahren (vgl. Rack und Christophersen, 2007, Seite 17). Der Forscher erhebt Daten systematisch und unter kontrollierten Bedingungen. Dazu werden Einflussgrößen bei gleichzeitiger Eliminierung oder Neutralisierung von Störfaktoren gezielt variiert (vgl. Christensen, 1980, Seite 34). Gezielt bedeutet in diesem Zusammenhang, dass einzelne Einflussgrößen verändert, während andere konstant gehalten (Neutralisierung) oder ausgeschaltet (Eliminierung) werden (vgl. Sarris, 1990, Seite 129 f.). Drei wesentliche Kriterien kennzeichnen ein Experiment (vgl. Greenwood, 1972, Seite 178):

- Test einer Kausalhypothese durch
- Vergleich einer zufällig zusammengestellten, "behandelten" (Versuchsgruppe) mit einer ebenso zufällig zusammengestellten, "unbehandelten" Probandengruppe (Kontrollgruppe)
- unter kontrollierten Umgebungsbedingungen.

6.2.3 Auswahl der Forschungsmethodik

Ein Korpus für Verträge konventionellen Ursprungs zur Beantwortung der Forschungsfrage 7 kann auf verschiedene Weise gewonnen werden:

1. **Querschnittstudie:** Verwendung der dokumentierten Web Services des seekda-Korpus.
2. **Querschnittstudie:** Ziehen einer neuen Stichprobe aus einem öffentlichen Quelltext-Depot (z.B. gitHub oder Sourceforge).[1]

[1] gitHub und Sourceforge sind Online-Archive für Quelltexte. Die Archive sind unter `http://www.github.com` bzw. `http://sourceforge.net` erreichbar (letzter Abruf jeweils am 13.06.2013).

3. **Fallstudie:** Verwendung öffentlich verfügbarer API-Verträge (z.B. des Google Places API [2] oder des Ebay Trading API [3]).

4. **Fallstudie:** Verwendung nicht-öffentlicher Quelltexte eines oder mehrerer Industrieprojekte.

5. **Laborbeobachtung:** Probanden erstellen unter kontrollierten Bedingungen mit einem konventionellen Verfahren Operationen für eine präzise definierte Aufgabenstellung.

Für die vorgeschlagenen Querschnitt- und Fallstudien spricht ihre Nähe zur Realität. Diesen Analysen liegen "echte" API-Verträge zugrunde, die nach dem aktuellen Stand der Forschung für reale Einsatzszenarien formuliert worden sind. Insbesondere die in den zwei vorgeschlagenen Querschnittstudien erhobenen Daten dürften aufgrund des zu erwartenden Stichprobenumfangs besonders repräsentativ sein.

Im Fall der Verwendung des seekda-Korpus stellt sich allerdings ein praktisches Problem: die Bestimmung aller notwendigen Vertragsinhalte. Für keinen der im seekda-Korpus enthaltenen Web Services liegen die zugrunde liegenden Quelltexte vor. Insofern kann auf die Menge der Ein- und Ausgaben sowie der Konfigurationen nur rückgeschlossen werden. Es ist sehr unwahrscheinlich, dass dieser Rückschluss einen belastbaren Erwartungshorizont für die Inhalte der Verträge liefert.

Dieser Nachteil gilt nicht für die zweite vorgeschlagene Querschnittstudie. Im Fall einer Stichprobe aus einem Quelltext-Depot läge der Quelltext zu jeder Operation vor. Damit könnte die Menge der Ein- und Ausgaben für jede Operation bestimmt werden (mit erheblichem Analyseaufwand). Allerdings ist aus den Quelltext-Depots nicht ersichtlich, unter welchen Bedingungen die Verträge erstellt worden sind. Welcher Zeit- und Personalaufwand ist beispielsweise investiert worden? Da die meisten Depots mit einer Versionskontrolle ausgestattet sind, können zwar Indikatoren für den erbrachten Aufwand erhoben werden (z.B. Anzahl der Autoren oder die Anzahl der Änderungen eines Vertrages). Ein eindeutiger Rückschluss auf beispielsweise die aufgewendete Zeit und beteiligten Personen ist mit diesen Daten jedoch nicht möglich. Beide Faktoren sind allerdings für die Bewertung der Vertragsqualität entscheidend. Es ist sehr wahrscheinlich, dass ein von

[2] Die Google Places API ist verfügbar unter `https://developers.google.com/places/` (letzter Abruf am 13.06.2013).

[3] Die Ebay Trading API ist verfügbar unter `https://go.developer.ebay.com/developers/ebay/products/trading-api` (letzter Abruf am 13.06.2013).

mehreren Personen über einen langen Zeitraum erstellter und geprüfter Vertrag vollständiger ist als ein solcher, der von nur einer Person in sehr kurzer Zeit formuliert worden ist.

Ein weiterer, erheblicher Nachteil dieser Forschungsmethodik ist, dass nicht bestimmbar ist, in welcher Entwicklungsphase des Programms die Verträge erstellt worden sind. Wurde der Vertrag während der Feinentwurfs- oder während der Implementierungsphase des Systems formuliert? Nach Balzert werden während der Feinentwurfsphase Komponenten und Subsysteme einer Softwarearchitektur detailliert entworfen (vgl. Balzert, 2011, Seite 6). In der Implementierungsphase findet hingegen die Programmierung des Software-Entwurfes in Form eines lauffähigen Softwaresystems statt (vgl. Balzert, 2011, Seite 492).

Die Berücksichtigung dieses Zeitpunktes ist erheblich für die Bewertung der Qualität der empfohlenen Vertragsinhalte - insbesondere bezüglich der im Vertrag aufgeführten zu erwartenden Ausnahme- und Fehlerfälle. Im Idealfall sind diese Fälle bereits während der Feinentwurfsphase vorhergesehen und ihre Behandlung ist entsprechend spezifiziert worden. So kann die Reaktion des Gesamtsystems auf die Ausnahme möglichst frühzeitig genau abgestimmt werden. Dies ist im Allgemeinen von Vorteil, da Anpassungen eines Programms in späteren Entwicklungsphasen tendenziell mit höheren Kosten verbunden sind als in früheren Phasen (vgl. Boehm, 1981, Seite 39 f.). Daher ist ein Vergleich von Verträgen aus der Feinentwurfsphase eines Systems besonders interessant.

Vor diesem Hintergrund erscheint die vorgeschlagene Laborbeobachtung als eine geeignete Forschungsmethode zur Erhebung des Korpus an Operationen. Im Labor könnten Probanden zu einer definierten Aufgabenstellung Signaturen und Verträge mit einem konventionellen Werkzeug wie *javadoc* erstellen.[4] Die Signaturen der auf diese Weise erhobenen Operationen können als Grundlage für den konzeptuellen Vergleich zur Beantwortung der Forschungsfrage 7 dienen. Aber auch für die Untersuchung der Forschungsfrage 8 ist die Laborbeobachtung reizvoll. Ein naheliegender Gedanke ist die Einführung einer zweiten Probandengruppe, welche unter gleichen Bedingungen im Labor Verträge auf Basis thematischer Raster zur gleichen Aufgabenstellung erstellt. Im Labor ist es möglich, die Bedingungen der Probanden, wie den Zeitaufwand oder den Personaleinsatz, für die Erstellung der Verträge zu kontrollieren. Des Weiteren kann die Menge der relevanten Bestandteile pro Vertrag für eine Aufgabenstellung im Vorwege bestimmt werden. Dies ermöglicht eine robuste qualitative Bewertung der erstellten Verträge mit den in Kapitel

[4] Die Signatur muss in diesem Fall in Java notiert werden.

5 vorgestellten Kennzahlen. Gleichzeitig ist die Erhebung von repräsentativen Verträgen gewährleistet, da die Probanden für die Laborbeobachtung zufällig bestimmt werden.

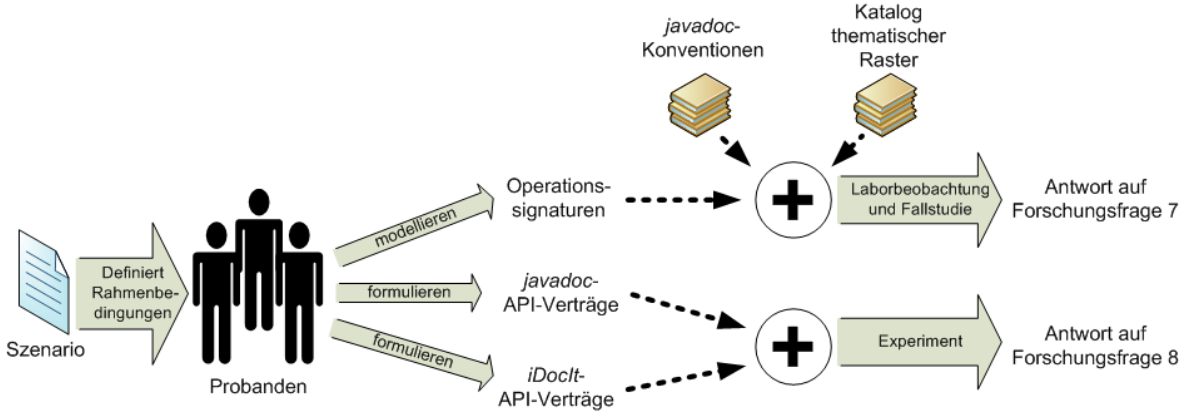


Abbildung 6.1: Darstellung des Versuchsaufbaus, der -teilnehmer und -komponenten in Bezug auf die Forschungsfragen

Aus den genannten Gründen wird zur Beantwortung der Forschungsfrage 7 die Methode der Fallstudie in Kombination mit der Laborbeobachtung eingesetzt. Im Rahmen der Laborbeobachtung werden Signaturen für ein vorgegebenes Szenario erhoben. Die auf Basis dieser Signaturen gegebenen Empfehlungen für Vertragsinhalte von *javadoc* allein und *javadoc* in Kombination mit thematischen Rastern werden einander im Rahmen einer Fallstudie gegenübergestellt. Zur Beantwortung der Forschungsfrage 8 werden die mit *javadoc* und *iDocIt!* erstellten Verträge im Rahmen eines Experimentes miteinander verglichen. Eine zusammenfassende Darstellung dieser Zusammenhänge gibt die Abbildung 6.1.

6.3 Szenario

6.3.1 Vorüberlegungen

Zur Durchführung der Laborbeobachtung wird eine Aufgabenstellung für die Probanden benötigt. Auf Basis dieser Aufgabenstellung erstellen die Probanden die Operationen. Folglich wird eine Reihe von Tätigkeiten benötigt, die in jeweils einer Operation abgebildet und durch den zugehörigen Vertrag zu spezifizieren sind. Für das Verfassen der Spezifikation muss der Proband ein Verständnis der dahinter stehenden Tätigkeit entwickeln. Hier zeigt sich ein grundsätzlicher Konflikt: Einerseits muss die Aufgabenstellung

präzise genug sein, um ein vergleichbares Verständnis der Probanden und damit vergleichbare Operationen bzw. Verträge zu erhalten. Gleichzeitig muss sie aber abstrakt genug sein, um die Gestaltung der Operationen und insbesondere der Verträge nicht zu sehr zu beeinflussen. Wenn die Probanden die Beschreibung einer Tätigkeit z.B. in Textform erhalten, könnten sie einzelne Formulierungen rein mechanisch in den jeweiligen Vertrag übernehmen.

Zuallererst ist zu entscheiden, ob die Tätigkeiten in einem inhaltlichen Zusammenhang stehen sollen oder nicht. Wenn die Tätigkeiten inhaltlich unabhängig voneinander sein dürfen, können Operationen aus öffentlich verfügbaren Quelltexten bzw. Industrieprojekten selektiert werden. So wird ein breites Spektrum realer Fallbeispiele abgedeckt. Allerdings müssen die Probanden bei dieser Variante immer wieder in den jeweiligen Kontext der Operation eingeführt werden. Dieser Nachteil besteht bei inhaltlich zusammenhängenden Operationen, z.B. in Form eines Prozesses, nicht. Hier wäre eine solche Einführung nur einmalig notwendig. Somit können die Probanden den größeren Teil ihrer Aufmerksamkeit zur Formulierung der Verträge und nicht zum Verständnis der Aufgabenstellung aufwenden. Das gilt aber nur, wenn der ausgewählte Prozess nicht zu komplex ist und ohne spezifisches Branchenwissen verstanden werden kann. Aus diesen Gründen werden Tätigkeiten mit inhaltlichem Zusammenhang in Form eines Prozesses für die weitere Untersuchung präferiert.

Leider zeigt die Erfahrung, dass Prozesse in realen Betrieben nicht nur Gesetzmäßigkeiten und Normen einer Branche, sondern auch Besonderheiten der jeweils ausführenden Organisation beinhalten können (“Wir machen es auf diese Weise, weil wir es noch nie anders gemacht haben”). Diese Besonderheiten sind meist nur mit tiefem Hintergrundwissen nachvollziehbar. Deshalb soll dieser Prozess nicht aus einem öffentlich verfügbaren Quelltext extrahiert, sondern für ein fiktives Szenario modelliert werden. Für dieses Szenario wird die Bestellung von Büchern bei einem Buchversandhändler im Internet als Grundlage verwendet. Es wird dabei angenommen, dass eine hohe Zahl von Personen über eine gewisse Affinität zu diesem Kontext und ggf. sogar über eigene Erfahrungen verfügt.

Anschließend gilt es die Form der Tätigkeitsbeschreibung festzulegen. Wie bereits ausgeführt, ist die Verwendung natürlicher Sprache in Textform möglichst zu vermeiden, um der mechanischen Übernahme von Formulierungen in den Vertrag entgegenzuwirken. Zur Beschreibung von Tätigkeiten sind verschiedene alternative Formen zur rein textuellen Beschreibung denkbar:

- **Filmaufnahme:** Darstellung des Ablaufs im Rahmen einer realitätsnahen Video-Aufzeichnung.

- **Tonaufnahme:** Verbale Beschreibung des Ablaufs im Rahmen einer Audio-Aufzeichnung.[5]

- **Prozessdiagramm:** Grafische Darstellung des Ablaufs in standardisierter Notation (z.B. als Ereignisgesteuerte Prozesskette oder als Prozessdiagramm in der Business Process Modeling Notation (BPMN)).

Für die realitätsnahe Filmaufnahme spricht, dass sie der natürlichen Beobachtung des Prozesses durch den Probanden am nächsten kommt. Die Probanden sind so bezüglich der sprachlichen Beschreibung der Tätigkeiten geringstmöglich beeinflusst. Allerdings eignet sich die Filmaufnahme schlecht zur Darstellung von software-technischen Prozessen, wie z.B. dem Speichern eines Datensatzes in einer Datenbank. Bei der Tonaufnahme entfällt dieser Nachteil, weil software-technische Tätigkeiten sprachlich beschrieben werden können. Allerdings besteht ein erhebliches Missverständnispotential bei einer rein sprachlichen Beschreibung des Prozesses, z.B. bei der Beschreibung von Spaltungen und Zusammenführungen des Ablaufs durch Bedingungen. Ein Prozessdiagramm hingegen dient zur übersichtlichen und exakten Darstellung von Abläufen. Damit ist die Gefahr eines Missverständnisses wesentlich geringer als bei einer Tonaufnahme.

Des Weiteren lassen sich durch ein Prozessdiagramm anders als in einer Filmaufnahme sowohl manuelle als auch software-technische und kombinierte Tätigkeiten darstellen. Der Nachteil des Prozessdiagramms ist, dass es zum Teil natürliche Sprache in Form von Beschriftungen grafischer Modellelemente enthält (z.B. die Beschriftung “Lese Datensatz” eines Aktivitätssymbols). Da diese Beschriftungen allerdings sehr elementar sind, ist dieser Nachteil nicht schwer zu gewichten. Eine Ausnahme hiervon bilden komplexere Regeln, wie z.B. Formeln zur Berechnung eines Wertes. Da derlei Inhalte aber in jedem Fall exakt durch die Aufgabenstellung zu beschreiben sind (um die Vergleichbarkeit der von den Probanden erstellten Verträge zu gewährleisten), wird das Risiko der analogen Übernahme in den Vertrag in diesem Fall in Kauf genommen. Daher wird das Prozessdiagramm als Darstellungsform für die zu spezifizierenden Tätigkeiten verwendet.

[5] Die Tonaufnahme hat, wie eine textuelle Spezifikation auch, eine natürlichsprachliche Form. Im Unterschied zum Text können die Probanden aus einer Tonaufnahme aber keine Fragmente per “Kopieren und Einfügen”-Funktion in ihre Spezifikation übernehmen.

Es stellt sich anschließend die Frage nach der Anzahl der zu spezifizierenden Operationen. Zur Beantwortung dieser Frage wird der in Abschnitt 6.4 beschriebenen Versuchsplanung vorgegriffen: für die Bearbeitung des Szenarios stehen ca. 90 Minuten zur Verfügung. Es wird angenommen, dass zwischen zehn und fünfzehn Minuten für die Spezifikation einer Operation benötigt werden. Folglich darf das Prozessmodell maximal sechs bis sieben zu spezifizierende Operationen beinhalten.

6.3.2 Aufgabenstellung

Der Proband ist als Software-Architekt beim fiktiven Buchversandhändler Bookseller AG angestellt. Er erhält vom Stab zur Prozessoptimierung eine verbesserte Version des Bestellabwicklungsprozesses.[6] Dieser soll nun erstmalig vollständig durch Software unterstützt werden. Von dieser Maßnahme verspricht sich die Bookseller AG Kosteneinsparungen in Höhe von ca. 1 Mio. EUR pro Jahr.

Jeder Proband hat die Aufgabe, die Implementierung des beschriebenen Geschäftsprozesses für einen Programmierer exakt zu spezifizieren. Die Spezifikation soll in Form von API-Verträgen für einzelne Operationen erfolgen. Die API-Verträge müssen in englischer Sprache erstellt werden. Um die erstellten Spezifikationen vergleichbar zu halten, ist ihre Form wie folgt vorgegeben:

1. Jede der im Diagramm enthaltenen Abteilungen (repräsentiert durch Lanes des Pools “Bookseller AG”) stellt eine eigene Java-Schnittstelle.

2. Zu spezifizieren sind nur die in Orange gefärbten Aktivitäten. Die Spezifikation der Aktivitäten “Prüfe Bonität anhand der Rechnungsanschrift” und “Erzeuge Packauftrag” sind optional. Diese Aktivitäten sollen nur spezifiziert werden, wenn der Proband über ausreichend Restzeit verfügt. Die grau gefärbten Aktivitäten dienen lediglich dem Verständnis des Gesamtprozesses und sind nicht zu spezifizieren.

3. Die Spezifikation der Dienstleistungen einer Abteilung erfolgt in Form einer mit *javadoc* oder *iDocIt!* kommentierten Java-Operation in der zugehörigen Java-Schnittstelle. Das zu verwendende Werkzeug richtet sich nach der Gruppenzugehörigkeit des Probanden (siehe Abschnitt 6.4).

4. Die im Prozess benötigten Datenstrukturen sind in Form von Java-Klassen als Eclipse-Projekt vorgegeben (siehe Abbildung 6.2).

[6] Das Originaldokument der Aufgabenstellung befindet in Anhang D.1.

Die Musterlösung für die von den Probanden zu formulierenden API-Verträge ist in Anhang E angegeben.

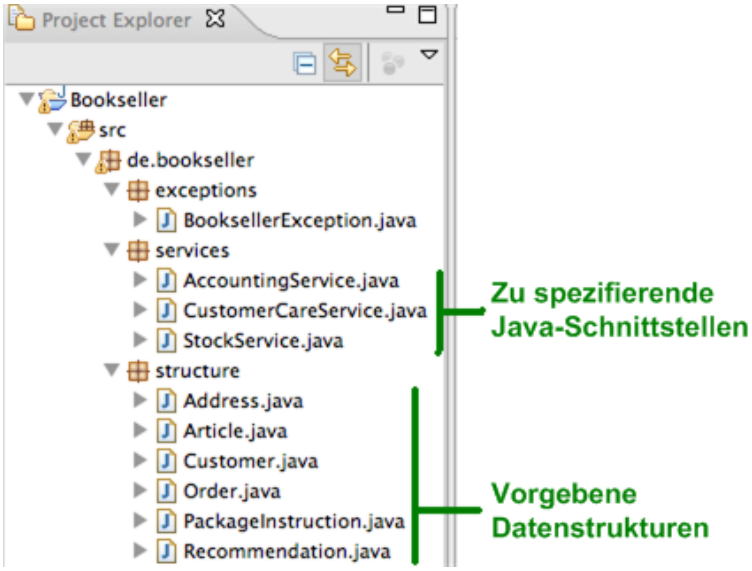


Abbildung 6.2: Vorgegebene Projektstruktur

6.3.3 Anforderungen an die Probanden

Zur Bearbeitung der Aufgabenstellung sind die folgenden Anforderungen an die teilnehmenden Probanden identifiziert worden:

- Grundlegende Programmierkenntnisse in *Java* (Beschreibung von Schnittstellen, Methoden und Ausnahmen).
- Fortgeschrittene Kenntnisse in *javadoc* (Kenntnis der Tags @param, @return, @throws und ihrer Verwendungskontexte).
- Grundlegende Kenntnisse der Notation BPMN 2 für Geschäftsprozessmodelle (Pools, Lanes, Datenobjekte, Datenspeicher, Annotationen, Gateways, Aktivitäten, (Nachrichten- und Zeit-)Ereignissse, Nachrichtenflüsse).
- Grundlegende Kenntnisse in der Arbeit mit der Entwicklungsumgebung *Eclipse* (Java-Perspektive und -Editor, Öffnen und Nutzung von Editoren und Views).

6.4 Aufbau und Durchführung

6.4.1 Rekrutierung der Probanden

Bei Evaluierungen mit menschlichen Probanden stellt sich immer die Frage nach deren Rekrutierung. Wie zuvor beschrieben, müssen die Probanden bestimmte Anforderungen erfüllen. Zum anderen muss für die Probanden grundsätzlich ein Anreiz geschaffen werden, um geeignete Kandidaten zur Teilnahme zu bewegen. Dazu bieten sich verschiedene Möglichkeiten an, beispielsweise:

- Bezahlung der Probanden mit einer Aufwandsentschädigung.
- Angebot eines Erkenntnisgewinns für die Probanden (z.B. im Rahmen einer Schulung).
- Angebot, weitere Menschen mit ähnlichen Aufgaben und Interessen kennenzulernen (z.B. im Rahmen einer Übungsveranstaltung mit anderen Interessierten).
- Kombination der Untersuchung mit einer Lehrveranstaltung für Studenten an einer Universität oder (Fach-)Hochschule.

Idealerweise ist die Stichprobe der rekrutierten Probanden möglichst homogen bezüglich der Erfahrung im Lesen und Verfassen von API-Verträgen (und weiterer Kontrollvariablen). Je weniger die Kontrollvariablen streuen, desto eindeutiger sind qualitative Unterschiede zwischen den erstellten API-Verträgen mit den Veränderungen der unabhängigen Variablen erklärbar. Als Beispiel ist hier ein Programmierer mit 20 Jahren Berufserfahrung vorstellbar. Wenn diese Person einen qualitativ hochwertigen API-Vertrag auf Basis thematischer Raster erstellt, so könnte die hohe Qualität auch auf den Erfahrungsschatz des Programmierers zurückgeführt werden. Bei verhältnismäßig unerfahrenen Programmierern ist die Erklärung der hohen Vertragsqualität mit den Empfehlungen der thematischen Raster hingegen wesentlich wahrscheinlicher. Aus diesem Grund werden die Kombination mit einer Lehrveranstaltung und die Einbindung von Studenten als Probanden gegenüber der Rekrutierung von erfahrenen Programmierern aus Unternehmen favorisiert. Die zuvor aufgeführten Anforderungen an die Probanden sowie der Kontext der API-Vertragsgestaltung bilden sehr große Schnittmengen zu den Inhalten der Vorlesung “Modellbasierte Software-Entwicklung” am Department Informatik der Universität Hamburg. Es bietet sich daher an, Probanden aus dem Hörerkreis dieser Veranstaltung für die weiteren Untersuchungen zu rekrutieren.

6.4.2 Kontroll- und Versuchsgruppe

Die Forschungsfrage 8 soll durch einen Vergleich von mit *javadoc* und *iDocIt!* erstellten Verträgen beantwortet werden. Dies geschieht im Rahmen eines Experiments. Daher ist die Aufteilung der Probanden in zwei Gruppen erforderlich. Die Probanden der ersten Gruppe bearbeiten die Aufgabenstellung mit *iDocIt!* (Versuchsgruppe) und die der zweiten Gruppe mit *javadoc* (Kontrollgruppe). Zu den Anforderungen an ein Experiment gehört die zufällige Zuordnung von Probanden zu einer Gruppe. Aus diesem Grund werden die Probanden auch für diese Untersuchung per Losverfahren verteilt. Als eine sehr wichtige Determinante der späteren Vertragsqualität gilt die Erfahrung des Probanden beim Schreiben von Verträgen. Wenn ein Großteil der erfahrenen Autoren der einen Gruppe und viele unerfahrene Autoren der anderen Gruppe zugelost werden, dann sind Verzerrungen des Versuchsergebnisses zu erwarten. Aus diesem Grund wird das Losverfahren in drei Schritten durchgeführt:

1. Die Probanden schätzen sich zuerst selbst bezüglich ihrer Vorkenntnisse von *javadoc* ein.[7] Die vorgegebenen Möglichkeiten zur Einordnung lauten:

   (a) "Etwas darüber gehört / gelesen" (Ordinalwert: 1).

   (b) "In Lehrveranstaltung behandelt" (Ordinalwert: 2).

   (c) "In Praktikum / Übung behandelt" (Ordinalwert: 3).

   (d) "In Entwicklungsprojekten angewendet" (Ordinalwert: 4).

2. Zunächst werden die erfahrenen Probanden in die Gruppen gelost (Selbsteinschätzung in den Kategorien c) und d)). Dazu wird die Anzahl der erfahrenen Probanden halbiert. Entsprechend viele Lose für beide Gruppen werden in den Lostopf gegeben. Sollte die Anzahl der erfahrenen Probanden ungerade sein, so wird für die Kontrollgruppe ein zusätzliches Los zugeführt. Auf diese Weise ist sichergestellt, dass die mit *iDocIt!* arbeitende Gruppe keinen Vorteil hat.

3. Abschließend erfolgt die Zuordnung der unerfahrenen Probanden analog. Bei einer ungeraden Anzahl an Probanden wird darauf geachtet, dass sich für die Versuchsgruppe ein zusätzliches Los im Topf befindet. So ist sichergestellt, dass die *iDocIt!* nutzenden Probanden tendenziell unerfahrener sind.

[7] Es wird dabei angenommen, dass jemand, der viel Erfahrung mit *javadoc* hat, auch erfahren im Verfassen von API-Verträgen ist.

6.4.3 Erhebung der Variablen

Nach Klärung der Rekrutierung und Grupenzuordnung der Probanden sind die durchzuführenden Messungen zu spezifizieren: Welche abhängigen und unabhängigen Variablen sowie Kontrollvariablen werden erhoben? Mit Hilfe von Veränderungen der unabhängigen Variablen werden Änderungen an den abhängigen Variablen erklärt.

Variable	**Wertebereich**	**Beschreibung**
MISS_INP	$\mathbb{N}$	Anzahl fehlender Eingaben (pro Operation und pro Proband)
MISS_OUTP	$\mathbb{N}$	Anzahl fehlender Ausgaben (pro Operation und pro Proband)
NOT_DEC_TC	$\mathbb{N}$	Anzahl nicht entscheidbarer Testfälle (pro Operation und pro Proband)

Tabelle 6.1: Übersicht der erhobenen abhängigen Variablen

Als abhängige Variablen werden die Bestandteile der in Kapitel 5 vorgestellten Kennzahlen erhoben (siehe Tabelle 6.1). Alle abhängigen Variablen können auf Basis der erstellten API-Verträge durch Zählen erhoben werden[8]. Die einzige unabhängige Variable dieser Untersuchung ist die Variable TOOL. Sie repräsentiert das Werkzeug, mit dem ein Proband seine API-Verträge formuliert. Die Variable TOOL ist nominalskaliert und kann die Werte "idocit" oder "javadoc" annehmen. Ihr Wert für einen Probanden ergibt sich automatisch aus dessen Gruppenzugehörigkeit. Die angenommenen Beziehungen zwischen unabhängigen und abhängigen Variablen sind in Abbildung 6.3 dargestellt.

Abschließend erfolgt die Beschreibung der erhobenen Kontrollvariablen (siehe Tabelle 6.2). Alle Kontrollvariablen werden mit einem standardisierten Fragebogen erhoben[9]. Die Variablen API_THEORY, ID_THEORY und JD_THEORY repräsentieren jeweils einen Punktwert, den ein Proband mit seinen Antworten auf Wissensfragen erreicht hat. Jede Wissensfrage besteht aus einer Aussage, z.B. "Der Vertrag einer Operation muss alle von außen sichtbaren Effekte der Operation beschreiben (z.B. das Versenden einer E-Mail)". Der Proband kann diese Aussage bestätigen (Antwort "Trifft zu"), sich enthalten (Antwort

[8] Auf eine detaillierte Beschreibung des Vorgehens beim Zählen wird an dieser Stelle verzichtet und auf das Kapitel 5 verwiesen.

[9] Die Originaldokumente der Fragebögen für beide Gruppen sind im Anhang in den Anhängen D.2 und D.3 enthalten.

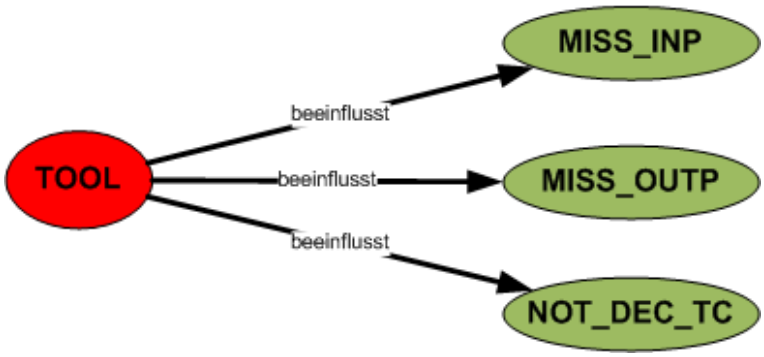


Abbildung 6.3: Angenommene Beziehungen zwischen unabhängigen und abhängigen Variablen

Variable	**Wertebereich**	**Beschreibung**
API_THEORY	$\{-5, -4, ..., 4, 5\}$	Erreichter Punktwert bei Wissensfragen bezüglich API-Verträgen im Allgemeinen.
THEORY	$\{-10, -9, ..., 9, 10\}$	Erreichter Punktwert bei Wissensfragen bezüglich *iDocIt!* oder *javadoc*.
SEMESTER	$\mathbb{N}$	Anzahl der Semester, in denen ein Informatik-Studiengang belegt worden ist.
ECLIPSE_EXP	$\{1, 2, 3, 4\}$	Selbsteinschätzung der Probanden bezüglich ihres Kenntnisstandes im jeweiligen Werkzeug auf der vierstufigen Skala (siehe den vorherigen Abschnitt).
JAVA_EXP		
SVN_EXP		
BPMN_EXP		

Tabelle 6.2: Übersicht der erhobenen Kontrollvariablen

"Weiß nicht") oder ablehnen (Antwort "Trifft nicht zu"). Für jede richtige Antwort erhält der Proband einen Punkt und für jede falsche Antwort wird ihm ein Punkt abgezogen. Die neutrale Antwort "Weiß nicht" wird mit keinem Punkt bewertet. Auf diese Weise wird erreicht, dass niemand bei Unsicherheit bezüglich einer Antwort zum Raten gezwungen wird. Die Unsicherheit wird so explizit sichtbar und kann zur Erklärung für die besonders gute oder besonders geringe Qualität der vom jeweiligen Probanden erstellten Verträge beitragen.

Jeder Proband muss dieselben fünf Wissensfragen zu API-Verträgen im Allgemeinen beantworten. Die Variable API_THEORY repräsentiert die hierbei erreichten Punkte. Außerdem werden den Probanden je nach Gruppenzugehörigkeit zehn Wissensfragen zum

jeweiligen Werkzeug gestellt. Die erreichten Punktwerte pro Proband werden durch die Variablen ID_THEORY bzw. JD_THEORY ausgewiesen. Kein Proband wird zu beiden Werkzeugen befragt.

Die Relevanz der Erfahrung der Probanden als Programmierer für die Qualität der erstellten Verträge ist bereits angesprochen worden. Die Erfahrung wird über die Variablen PROJECT_EXP und SEMESTER mittels der Fragebögen erhoben. Die Variable PROJECT_EXP misst die Anzahl von Monaten, in denen der Proband bereits in einem Software-Entwicklungsprojekt mitgearbeitet hat (z.B. an der Universität, in der Industrie oder im Opensource-Umfeld). Die Variable SEMESTER repräsentiert die Anzahl der Semester, in denen der Proband einen Informatik-Studiengang belegt hat.

Abschließend sind die Kenntnisse der Probanden bezüglich der im Labor eingesetzten Werkzeuge zu kontrollieren. Dazu schätzen die Probanden sich selbst bezüglich der Werkzeuge Eclipse (integrierte Entwicklungsumgebung), Subversion (Versionskontrolle), Java und der Business Process Modeling Notation 2 (BPMN) ein. Zur Skalierung werden die bereits im Abschnitt 6.4.2 genutzten vier Möglichkeiten zur Selbsteinschätzung vorgegeben.

6.4.4 Versuchsaufbau und -bedingungen

Die Laborbeobachtungen sind in zwei Computerräumen des Department Informatik der Universität Hamburg durchgeführt worden. Versuchs- und Kontrollgruppe haben die Aufgabenstellung jeweils in einem eigenen Raum bearbeitet. Auf diese Weise konnte durch den Versuchsleiter sichergestellt werden, dass die Probanden keinen Störeinflüssen wie z.B. hohe Umgebungslautstärke oder Kontaktversuchen ausgesetzt sind, z.B. durch Kommilitonen oder Anrufe auf dem eigenen Mobiltelefon. In einem Gemeinschaftsraum sind immer die Probanden beider Gruppen anwesend gewesen. Dies war z.B. für die gemeinsame Einführung in die Aufgabenstellung erforderlich. In den Gruppenräumen hingegen befanden sich jeweils nur Probanden entweder der Kontroll- oder der Versuchsgruppe. Jeder Proband verfügte über einen eigenen Computerarbeitsplatz.[10]

Alle Arbeitsplätze waren mit dem Internet verbunden und einheitlich mit den Programmen Windows XP, Eclipse 3.7, Subversive[11], Java Development Kit 1.7 Update 9, Adobe Reader und Firefox ausgestattet. Nur auf den Computern der Versuchsgruppe ist zusätz-

[10] Hier sei angemerkt, dass zwei Probanden der Kontrollgruppe aufgrund technischer Probleme an jeweils einem Computerarbeitsplatz im Raum der Versuchsgruppe gearbeitet haben.

[11] Subversive ist ein Eclipse-Plugin für den Zugriff auf das Versionskontrollsystem Subversion.

lich *iDocIt!* installiert worden. Es war den Probanden gestattet, während des Experiments im Internet zu recherchieren. Die Probanden beider Gruppen haben zusätzlich eine sog. Kurzreferenzkarte zur Dokumentation mit thematischen Rastern und *iDocIt!* bzw. mit *javadoc* im DIN-A4-Format erhalten.[12] Auf diesen Karten sind wichtige Konventionen und Bedienungshinweise zu dem jeweiligen Werkzeug aufgeführt. Zu konventionellem *javadoc* existiert im Internet sehr viel Material, zu *iDocIt!* und thematischen Rastern hingegen nicht. Mit den Kurzreferenzkarten soll dieser Nachteil neutralisiert werden.

Die Aufgabenstellung wurde einzeln bearbeitet. Deshalb verfügte jeder Proband ferner über ein eigenes Depot in der bereitgestellten Versionskontrolle.[13] Auf dieses Depot können nur der jeweilige Proband und der Versuchsleiter zugreifen. Die vorgegebene Projektstruktur wurde mitsamt einer elektronischen Version der Aufgabenstellung über die Depots an die Probanden ausgegeben. Nach Durchführung der Untersuchung haben die Probanden die erstellten Verträge wieder in ihr Depot eingecheckt. Auf diese Weise wurde der Abgabezeitpunkt exakt protokolliert.

6.4.5 Versuchsdurchführung

Nachdem der Kontext der Laborbeobachtung beschrieben worden ist, wird an dieser Stelle der genaue Ablauf der Untersuchung dargelegt. Die Gesamtdauer der Veranstaltung betrug 195 Minuten. Sowohl Kontroll- als auch Versuchsgruppe wurden während der Bearbeitung der Aufgabenstellung ständig beaufsichtigt. Die Probanden hatten während dieser Zeit die Möglichkeit, Fragen zur Aufgabenstellung oder zur Bedienung des jeweiligen Werkzeugs zu stellen. Fragen zur inhaltlichen Gestaltung der Verträge wurden nicht beantwortet.

Zu Beginn der Übung erfolgte eine Zuordnung der Probanden nach dem zuvor beschriebenen Verfahren zur Kontroll- bzw. Versuchsgruppe. Im Anschluss daran wechselten die Gruppen in ihre Räume. Dort erhielten die Probanden zeitgleich eine 30-minütige Kurzeinführung in *javadoc* (Kontrollgruppe) bzw. *iDocIt!* und thematische Raster (Versuchsgruppe) in Vortragsform. Während dieser Kurzeinführung bestand die Möglichkeit zur Klärung von Fragen. Nach einer 15-minütigen Pause ist den Probanden beider Gruppen eine gemeinsame 20-minütige Einführung in die Aufgabenstellung gegeben worden. Es bestand ebenfalls die Möglichkeit zur Klärung von Fragen bezüglich der Aufgabenstellung. Anschließend hatten die Probanden 90 Minuten Zeit für die Bearbeitung der Aufgaben-

[12] Die Originaldokumente beider Kurzreferenzkarten sind in den Anhängen D.4 und D.5 enthalten.
[13] Als Versionskontrolle wird das System Subversion eingesetzt.

stellung. Danach wurden die erstellten Verträge über die Versionskontrolle eingesammelt. Die Probanden füllten abschließend für 15 Minuten die Fragebögen aus. Das Ausfüllen der Fragebögen war für das Ende der Veranstaltung vorgesehen, weil so die eigenen Erfahrungen der Probanden mit dem jeweiligen Werkzeug während der Übung mit in die erhobenen Antworten eingingen. Auf diese Weise ergab sich in den erhobenen Daten ein repräsentativeres Bild der Kenntnisstände während der Übung. Dies wäre nicht der Fall, wenn die Fragebögen beispielsweise direkt nach den Kurzeinführungen ausgefüllt worden wären.

6.4.6 Regelbasierte Erzeugung der API-Verträge

Die Durchführung der zuvor detailliert beschriebenen Laborbeobachtung ergibt neben den mit *javadoc* und *iDocIt!* erstellten Verträgen auch eine Fülle von Operationssignaturen. Diese Signaturen werden durch die Probanden modelliert. Wie in Abschnitt 6.2.3 dargelegt, schließt der direkte Vergleich dieser Verträge einige einflussreiche Fehlerquellen ein, z.B. Defizite in der Bedienung und damit der Verständlichkeit des eingesetzten Werkzeugs.

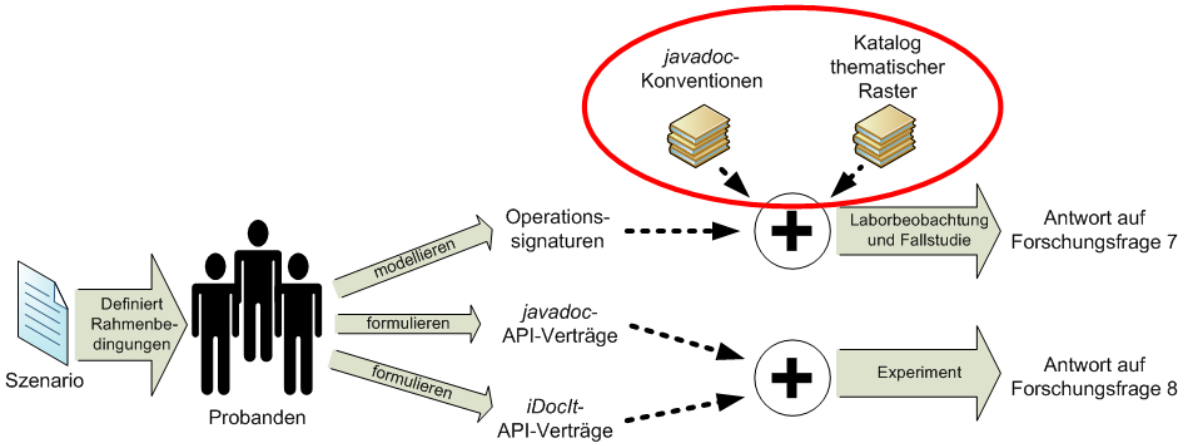


Abbildung 6.4: Einordnung der Regelsätze in den Versuchskontext

Aus diesem Grund werden zur Beantwortung der Forschungsfrage 7 nicht die von den Probanden erstellten Verträge verwendet. Hierfür werden hingegen Verträge herangezogen, die auf Basis der im Labor gewonnenen Signaturen und zweier vorab definierter Regelsätze (Konventionen) generiert worden sind. Dieser Abschnitt stellt diesen Regelsatz in Form von Konventionen für *javadoc* vor. Als Regelsatz für die thematischen Raster wird die folgende, echte Teilmenge des Katalogs aus Anhang C verwendet: löschende Operation, prüfende Operation, erzeugende Operation, konvertierende Operation, berechnende

Operation, suchende Operation, deponierende Operation und sendende Operation. Die Abbildung 6.4 hebt die Einordnung beider Regelsätze (*javadoc*-Konventionen und thematische Raster) im Versuchsaufbau hervor.

Für jede Aktivität existiert eine Menge an von den Probanden modellierten Signaturen. Zuerst ist zu klären, welche dieser Signaturen als Basis für den zu generierenden Vertrag gewählt werden soll. Als Abgrenzungskriterium sind zwei Alternativen denkbar:

- Wähle die Signatur mit der jeweils größten Summe der im Vertrag genannten Ein- und Ausgaben. Sofern die Summe mind. zweier Lösungen gleich ist, so wähle die Lösung gemäß der Reihenfolge der Probanden MBSE 18, MBSE 12, MBSE 1, MBSE 08, MBSE 05 und MBSE 03.[14]

- Wähle die Signatur mit der jeweils kleinsten Anzahl der nicht entscheidbaren Testfälle. Sollten mind. zwei Lösungen bezüglich dieser Anzahl gleich sein, so entscheide nach dem obigen Kriterium.

Für diese Fallstudie werden beide obigen Verfahren kombiniert. Die Ableitung der *javadoc*-Empfehlungen verläuft nach dem folgenden Schema:

1. Wähle aus den Operationen der Laborbeobachtung für die aktuelle Aktivität nach einem der obigen Kriterien die beste aus.

2. Beschreibe alle deklarierten formalen Parameter mit ihrer Bedeutung, den Beschränkungen ihres Wertebereiches und dem Verhalten der Operation bei `null`-Belegung des Parameters (sofern dies möglich ist).[15] Dabei ist zu beachten, dass Attribute von strukturierten Parametern (z.B. Klassenattribute in Java) bei Relevanz ebenfalls beschrieben werden müssen.[16]

3. Definiere eine Ausnahme für Parameter mit ungültigem Format oder unzulässiger `null`-Belegung.[17] Bei der späteren Analyse werden diese Ausnahmen als von *ja-*

[14] Die Abkürzung MBSE steht für den Titel der Lehrveranstaltung *Modellbasierte Software-Entwicklung*. Die Probanden haben jeweils eine entsprechende Kennung erhalten. Die hier angegebene Reihenfolge entspricht der zufälligen Auswertungsreihenfolge der Lösungen.

[15] Die *javadoc*-Konventionen verlangen die Beschreibung des Verhaltens einer Operation bei `null`-Belegung formaler Parameter. Ferner muss der Wertebereich der Formalparameter definiert werden (siehe Smith und Kramer (2003), Abschnitt "Method Specification").

[16] Diese Regel ist in den *javadoc*-Konventionen nicht explizit enthalten. Sie ist aus der allgemeinen Forderung nach Beschreibung eines Parameters gefolgert worden.

[17] Diese Konvention wurde aufgrund der expliziten Forderung der Beschreibung des Verhaltens im Falle einer `null`-Belegung in Smith und Kramer (2003), Abschnitt "Method Specification" gebildet.

vadoc empfohlene Fehlerbehandlung gewertet. In der ursprünglichen Signatur deklarierte Ausnahmen werden beibehalten. Diese Fehlersituationen gelten als von *javadoc* empfohlene Fehlerbehandlungen. Die im Bezeichner der Ausnahme erwähnten Ressourcen werden als benötigte Ressource in den Vertrag übernommen (siehe das dieser Auflistung nachfolgende Beispiel).

4. Beschreibe die Bedeutung und den Wertebereich des Resultates.[18] Sofern das Resultat ein Objekt sein kann, wird ein `null`-Resultat als "Objekt ist nicht vorhanden" interpretiert. Diese Interpretation ist gleichbedeutend einer entsprechenden Fehlermeldung. Wenn ein Objekt von der Operation erstellt und gespeichert wird, dann gilt es bei Rückgabe automatisch als gespeichert.

5. Sofern in der Signatur der Operation bereits Ausnahmen für nicht verfügbare Ressourcen enthalten sind, dann ergänze die jeweilige Ressource in einem Abschnitt "Required Resources" des abgeleiteten Vertrages.

Der Inhalt des einleitenden Satzes ist schwer bis gar nicht durch präzise Anweisungen algorithmisch ableitbar.[19] Aus diesem Grund wird er weggelassen. Für die Ableitung der Empfehlungen der thematischen Raster gilt dasselbe Schema wie für *javadoc*. Es wird abschließend um die folgenden Schritte erweitert:

6. Wähle ein passendes thematisches Raster für die Operation aus dem Katalog.

7. Ergänze den existierenden Vertrag um die thematischen Rollen entsprechend ihrer rasterspezifischen Bedeutung und den Angaben aus dem Prozessmodell. Belege unbelegte thematische Rollen im Vertrag mit "To be defined". Auf diese Weise werden Lücken des Prozessmodells markiert.

8. Ergänze vom Raster identifizierte Fehlersituationen im Vertrag. Auch hier gilt, dass die Nichtverfügbarkeit unbelegter thematischer Rollen mit "To be defined" gekennzeichnet wird.

Die vorgestellten Schemata sollen anhand eines Beispiels veranschaulicht werden. Angenommen, die Aktivität "Erzeuge Auftragsablehnung" aus dem Prozessmodell wird durch

[18] Die Eingrenzung des Wertebereiches inkl. der `null`-Belegung wird in Smith und Kramer (2003), Abschnitt "Method Specification" gefordert.

[19] Für eine Beschreibung der erwarteten Inhalte des einführenden Satzes wird auf Oracle Corp. (ohne Jahr), Abschnitt "Descriptions", "First Sentence" verwiesen.

die Operation `sendOrderRejection` am besten abgebildet (Schritt 1). Der abgeleitete *javadoc*-Vertrag wird in Listing 6.1 angegeben (Schritt 2 - 4).

```
/**
 * @param order The rejected order (not nullable)
 * @param customerId Who placed the given
 *    order. Must be greater than 0.
 *
 * Required resources:
 *  - Customer Relationship Management System
 *
 * @throws BooksellerException
 *    Error 01: if the given order is null
 *    Error 02: if no customer exists for the given
 *      customer-id
 * @throws CRMException
 *    Error 03: if Customer Relationship Management
 *      System is not available
 */
public void sendOrderRejection(Order order, int customerId)
    throws BooksellerException, CRMException;
```

Listing 6.1: Schematisch erzeugtes Beispiel für einen javadoc-Vertrag

In diesem Beispiel wird das *Customer Relationship Management System* als benötigte Ressource aufgeführt, weil es im Bezeichner der Ausnahme `CRMException` genannt wird (Schritt 5). Diese Operation verschickt eine Auftragsablehnung. Das Prozessmodell lässt den Typ der Nachricht offen. Die angegebene Operation ist folglich als sendende Operation zu klassifizieren (Schritt 6). Im Anschluss daran erfolgt die Ergänzung des *javadoc*-Vertrages um die thematischen Rollen und deren Belegung (Schritt 7).

```
/**
 * @param order The rejected order (not nullable)
 * @param customerId [PRIMARY KEY] Must be greater than 0.
 *
 * Required resources:
 *  - Customer Relationship Management System
 *
 * @object The customer who placed the order
 * @destination To be defined
 * @format The content of the sent message is defined in
 *  a template stored in the template management system.
 *  The template has to be defined.
 * @instrument To be defined
 *
 * @throws BooksellerException
 *  Error 01: if the given order is null
 *  Error 02: if no customer exists for the given id (ERROR
 *    OBJECT)
 *  Error 04: if the given id is invalid (ERROR PRIMARY KEY)
 *  Error 05: if the VMS itself or the template is not
 *   available (ERROR FORMAT)
 *  Error 06: To be defined (ERROR INSTRUMENT)
 *  Error 07: To be defined (ERROR DESTINATION)
 * @throws CRMException
 *  Error 03: if Customer Relationship Management
 *    System is not available
 */
public void sendOrderRejection(Order order, int customerId)
    throws BooksellerException, CRMException;
```

Listing 6.2: Schematisch erzeugtes Beispiel für einen Vertrag mit thematischen Rollen

Der Abgleich der katalogisierten thematischen Rollen mit dem obigen *javadoc*-Vertrag führt zu diversen Erweiterungen des Vertrages. Die Beschreibung des Parameters `customerId` ist durch die Erwähnung der Rolle PRIMARY KEY etwas gekürzt worden. Diese thematische Rolle drückt aus, dass über die ID der bestellende Kunde gefunden wird. Die Beschreibung "Who placed the given order" wird an das Tag *@object* verschoben, weil sie der Belegung der Rolle OBJECT entspricht (Schritt 7). In den Zeilen 12 - 19 befinden sich die Behandlungen der vom thematischen Raster empfohlenen Fehlerfälle (Schritt 8). Das Listing 6.2 enthält den um thematische Rollen ergänzten Vertrag.

6.5 Auswertung und Diskussion

Nach der detaillierten Beschreibungen des Szenarios und des Vorgehens bei dieser Untersuchung erfolgen in diesem Abschnitt die Darstellung und Diskussion der erhobenen Daten und gewonnenen Erkenntnisse. Zuvor wird der Umfang der während der Laborbeobachtung erhobenen Daten kurz beschrieben.

6.5.1 Umfang der erhobenen Daten

Insgesamt haben 14 Probanden an der Laborbeobachtung teilgenommen. Folglich umfassten Versuchs- und Kontrollgruppe jeweils sieben Probanden. Die Probanden sollten sieben bestimmte Aktivitäten des Prozessmodells spezifizieren. So ergeben sich pro Gruppe maximal 49 und insgesamt 98 Verträge. Von den 98 möglichen Verträgen sind 76 Verträge durch die Probanden erstellt worden. Davon entfallen 34 auf die Versuchsgruppe (thematische Raster und *iDocIt!*) und 42 auf die Kontrollgruppe (*javadoc*). Alle Probanden haben alle Fragen der standardisierten Fragebögen beantwortet und zurückgegeben. Daher liegen 14 vollständig ausgefüllte Fragebögen vor.

6.5.2 Stille in den Empfehlungen von *javadoc*

Die Forschungsfrage 7 zielt auf den Beitrag thematischer Raster zur Vermeidung von Stille ab. In der Abbildung 6.5 werden die Empfehlungen von *javadoc* für alle von den Probanden der Kontrollgruppe modellierten Signaturen mit dem Erwartungshorizont an Ein- und

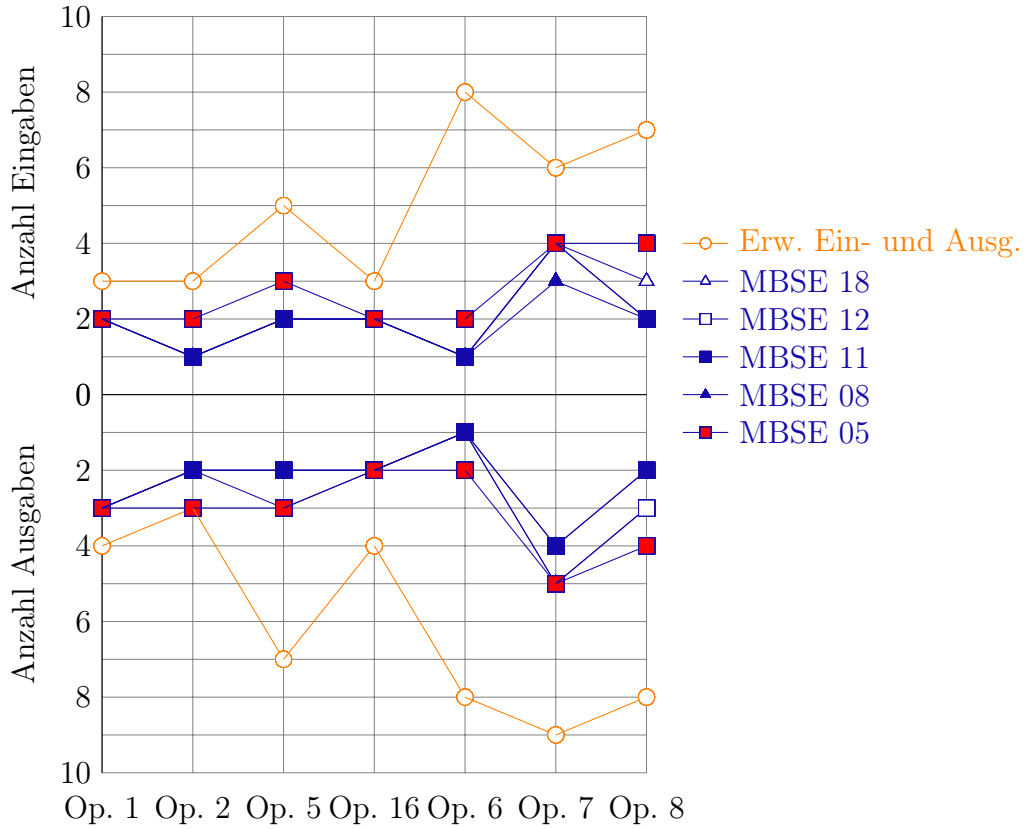


Abbildung 6.5: Abdeckung von Ein- und Ausgaben in den javadoc-Verträgen

Ausgaben verglichen.[20] [21] Der Erwartungshorizont wird durch den orange Graph repräsentiert. Im oberen Abschnitt sind die Eingaben abgetragen und im unteren Abschnitt die Ausgaben. Die Operationen sind auf der Abszisse nach ihrer Reihenfolge im Prozessmodell aufgeführt. Da diese Achse nominalskaliert ist, wäre auch jede andere Reihenfolge denkbar.

Zuerst fällt auf, dass die Anzahl der Empfehlungen von *javadoc* für die unterschiedlichen Signaturen pro Operation ziemlich ähnlich ist. Bei den ersten sieben Operationen unterscheidet sich die Anzahl der abgedeckten Ein- und Ausgaben maximal um eins. Lediglich bei der Operation 8 streuen die Empfehlungen zwischen zwei und vier Ein- und Ausgaben. Als Grundlage für die besten Empfehlungen ist die von Proband MBSE 05

[20] Die Lösungen der Probanden MBSE 09 und MBSE 03 sind hier ausgelassen worden, weil diese Probanden für jeweils vier der insgesamt sieben Operationen aus der Aufgabenstellung keine Lösung abgegeben haben. Damit tragen diese Daten nicht zu einem repräsentativen Gesamtbild bei.

[21] Die grafische Darstellung der Verläufe müsste korrekterweise jeweils ein Balkendiagramm sein, weil lineare Approximationen zwischen den gezählten Ein- und Ausgaben nicht zulässig sind. Da die Verläufe mit den Approximationen aber besser erkennbar sind, wurde hier auf die Stufendarstellung verzichtet.

modellierte Signatur mit jeweils vier abgedeckten Ein- und Ausgaben erkennbar. Dieser Proband hat jeweils eine Ausnahme für die Nichtverfügbarkeit der Systeme zur Vorlagenverwaltung und Finanzbuchhaltung in der Signatur modelliert. Gemäß des im vorangegangenen Abschnitt beschriebenen Konstruktionsschemas für die *javadoc*-Verträge sind damit nicht nur die Fehlermeldungen, sondern beide Systeme automatisch als benötigte Eingaben deklariert. Somit ist dieser Ausreißer nicht auf die Qualität der Empfehlungen von *javadoc*, sondern auf die Modellierungsfähigkeiten des Probanden zurückzuführen. Schließlich hat der Proband und nicht *javadoc* die Abhängigkeit der Operation zu beiden externen Systemen identifiziert und in der Signatur durch die Ausnahmen repräsentiert.

Diese Beobachtungen bestätigen die große Gefahr für Stille in API-Verträgen, die auf Basis aktueller Konventionen verfasst werden. In diesem Fall deckt *javadoc* mit seinen Empfehlungen bestenfalls 41 von 78 erwarteten Ein- und Ausgaben ab. Das entspricht einer Abdeckungsrate von ca. 53%. Diese Beobachtungen verdeutlichen die Notwendigkeit für zusätzliche Unterstützung bei der Formulierung von API-Verträgen.

6.5.3 Vermeidung von Stille durch thematische Raster

Können thematische Raster hier eine Verbesserung bewirken? Aufschluss darüber gibt die Abbildung 6.6.[22] Hier ist die bezüglich der Abdeckung von Ein- und Ausgaben optimale Kombination aller *javadoc*-Lösungen als blauer Graph dargestellt. Der grüne Graph repräsentiert die optimale *javadoc*-Lösung in Kombination mit thematischen Rastern. Im Folgenden wird letztere Variante als *kombinierte Lösung* bezeichnet. Diese ist nicht gleichzusetzen mit *iDocIt!*, weil die Verträge der kombinierten Lösung auf Basis der in Abschnitt 6.4.6 beschriebenen Konventionen erzeugt worden sind. Die *iDocIt!*-Verträge sind hingegen von Probanden formuliert worden.

Es fällt auf, dass die thematischen Raster für alle Operationen eine Verbesserung bringen. Allerdings schwankt das Optimierungspotential sowohl innerhalb als auch zwischen Ein- und Ausgaben. Positiv zu vermerken ist, dass die Abdeckungsrate der Operationen 1, 2, 5, 16 und 8 bei den Eingaben durch thematische Raster auf 100% verbessert wird. Mit *javadoc* allein beträgt die Abdeckungsrate bei den Operationen 1 und 2 ca. 67%, bei Operation 5 genau 60%, bei Operation 16 ca. 67% und bei Operation 8 ca. 57%. Bei der verbleibenden Operation 7 wird mit einer Abdeckungsrate von ca. 83% (vorher ca.

[22] Die grafische Darstellung der Verläufe müsste korrekterweise jeweils ein Balkendiagramm sein, weil lineare Approximationen zwischen den gezählten Ein- und Ausgaben nicht zulässig sind. Da die Verläufe mit den Approximationen aber besser erkennbar sind, wurde hier auf die Stufendarstellung verzichtet.

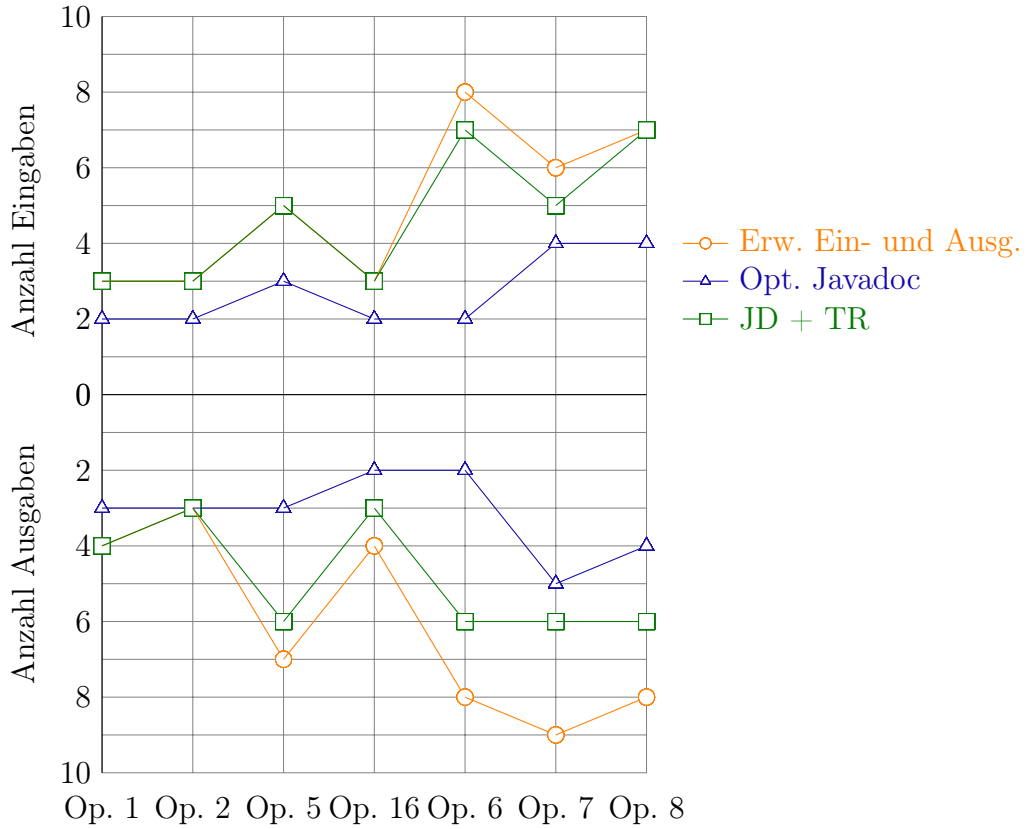


Abbildung 6.6: Abdeckung von Ein- und Ausgaben durch optimales javadoc und durch javadoc kombiniert mit thematischen Rastern

67%) ebenfalls eine deutliche Verbesserung erreicht. Der höchste Zuwachs bezüglich der Abdeckungsrate gelingt bei der Operation 6. Thematische Raster decken in Kombination mit *javadoc* ca. 88% der erwarteten Eingaben ab. *javadoc* allein erreicht nur 25%. Die Abdeckungsraten bei den Ausgaben sind durch die kombinierte Lösung ebenfalls deutlich gestiegen. Für die Operation 1 werden 100% der Ausgaben durch *javadoc* in Kombination mit thematischen Rastern identifiziert (*javadoc* allein: 75%). Bei der Operation 2 ist dies ebenfalls der Fall. Eine deutliche Steigerung ist bei der Operation 5 zu erkennen. Hier identifiziert die kombinierte Lösung ca. 86% der Ausgaben, während *javadoc* allein nur ca. 43% erkennt. Auch bei Operation 16 konnte die Abdeckungsrate von 50% auf 75% verbessert werden. Für die verbleibenden drei Operationen bietet sich ein interessantes Bild: Unabhängig von der Anzahl der erwarteten und der von *javadoc* identifizierten Ausgaben deckt die kombinierte Lösung konstant sechs Ausgaben ab. Dies ist in allen Fällen eine Verbesserung. Der größte Zuwachs wird bei Operation 6 erreicht. Während *javadoc* allein 25% der erwarteten Ausgaben abdeckt, erreicht die kombinierte Lösung 75%.

Bei der Betrachtung aller 35 Eingaben und 43 Ausgaben aller Operationen insgesamt zeigt sich ebenfalls der deutliche Zugewinn durch thematische Raster. Bei den Eingaben identifiziert *javadoc* 19 (ca. 54%) und die kombinierte Lösung 33 (ca. 94%). Bei den Ausgaben deckt *javadoc* 22 (ca. 51%) und die kombinierte Lösung 35 (ca. 81%) ab. Die Anzahl der abgedeckten Ein- und Ausgaben steigt also von 41 auf 67 bzw. die Abdeckungsrate von ca. 53% auf ca. 86%. Im Anschluss stellen sich nun zwei Fragen:

1. Durch Abdeckung welcher Ein- und Ausgaben konnten die thematischen Raster diese hohen Zuwächse erreichen?

2. Welche Ein- und Ausgaben werden nach wie vor nicht abgedeckt?

Um diese Fragen zu beantworten, erscheint eine Kategorisierung der Ein- und Ausgaben sinnvoll. Anhand einer solchen Kategorisierung kann abgelesen werden, welche Arten von Ein- und Ausgaben durch *javadoc* und welche zusätzlich durch die kombinierte Lösung identifiziert werden und welche nicht. Die Eingaben lassen sich in folgende Kategorien einteilen:

- **Wert:** Vor dem Aufruf der Operation existierende Ausprägungen von primitiven (z.B. der `int` "1" oder der `char` "A") oder komplexen Datenstrukturen (z.B. Instanz der Klasse "Kunde" mit belegten Attributen oder Zeile in der Tabelle "Kunde" mit belegten Spalten einer Datenbank).

- Eine **Ressource** ist eine fremde Komponente oder ein externes System bzw. Gerät, welches von der Operation zur Durchführung ihrer Tätigkeit benötigt wird.

- Eine **Regel** beschreibt die Beziehung zwischen den in die Operation eingehenden Werten und Ressourcen sowie den aus der Operation resultierenden Werten, Zustandsänderungen und Fehlermeldungen (siehe die Definitionen der gleichnamigen Kategorien für die Ausgaben weiter unten im Text). Einer Operation können mehrere Regeln zugrunde liegen.

Bei den Ausgaben sind die folgenden Kategorien identifiziert worden:

- **Wert (im Sinne eines Resultates):** Nach dem Aufruf der Operation existierende Ausprägungen von primitiven oder komplexen Datenstrukturen (für Beispiele siehe die Definition der Kategorie "Wert" bei den Eingaben).

- Eine **Fehlermeldung** resultiert aus einer Operation, wenn sie mit einer ungültigen Kombination von Eingaben aufgerufen wird und diese Situation vom Programmierer oder Spezifizierer vorhergesehen worden ist. Diese Fehlermeldungen entsprechen geprüften Ausnahmen (siehe Abschnitt 4.3.3).
- Eine **Zustandsänderung** ist ein außen sichtbarer Effekt, den die Operation durch ihre Tätigkeit in einer Ressource hervorruft.

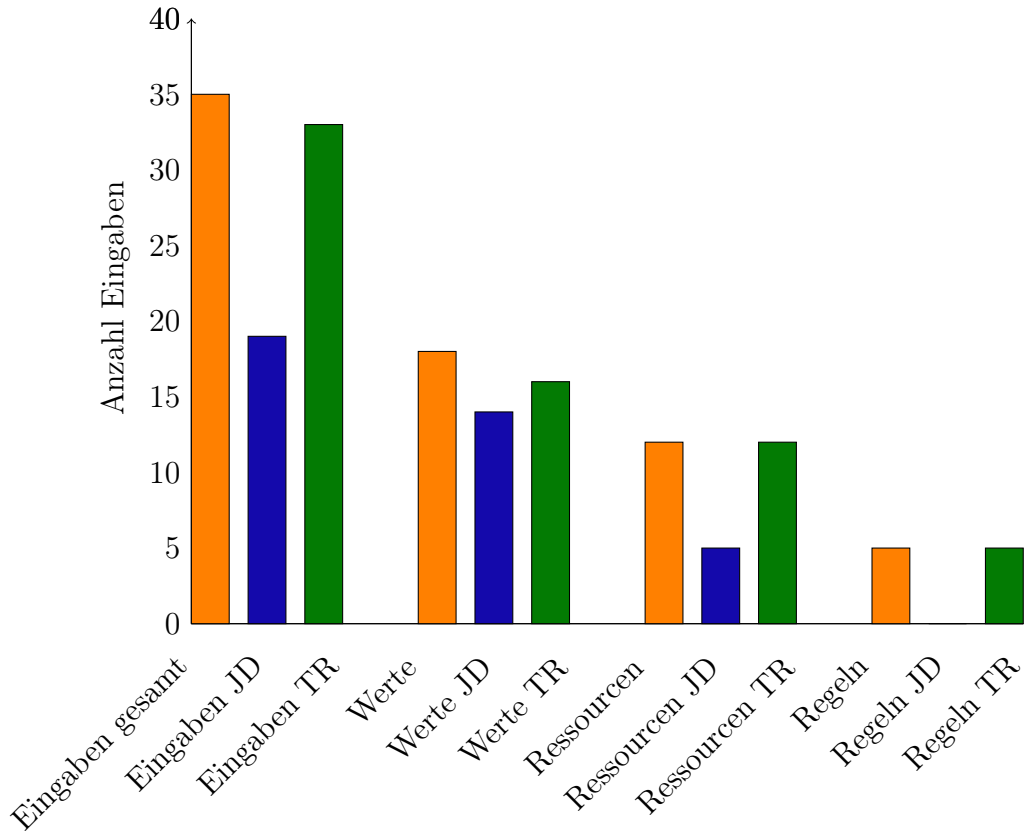


Abbildung 6.7: Abdeckung von Eingaben der verschiedenen Kategorien

Die Abdeckungssituation bei den Eingaben ist in Abbildung 6.7 dargestellt.[23] Sieben der insgesamt zusätzlich abgedeckten 14 Eingaben sind als Ressourcen klassifiziert. Dabei handelt es sich z.B. um das Finanzbuchhaltungssystem zum Speichern der erstellten Rechnung (Aktivität "Erstelle Rechnung" im Prozessmodell), den Drucker[24] zum Ausdrucken der erstellten Rechnung (Aktivität "Drucke und packe Rechnung" im Prozessmodell)

[23] Hinweis zur Interpretation des Diagramms: JD steht für *javadoc* und TR für thematische Raster.

[24] An dieser Stelle mag der Leser einwenden, dass ein Drucker nicht in den Vertrag aufzunehmen sei. Bei seiner Nichtverfügbarkeit sei die korrekte Bearbeitung der Durckaufträge schließlich durch das

oder das Auftragsverwaltungssystem zum Speichern der Bestellung (Aktivität "Speichere Bestellung"). Alle drei Ressourcen werden durch die thematische Rolle DESTINATION abgebildet: im ersten und letzten Fall das Ziel eines Speichervorgangs und im zweiten Fall das Ziel eines Sendungsvorgangs. In zwei weiteren Fällen sind Ausprägungen der thematischen Rolle FORMAT hinzugekommen. Diese Rolle ist mit den verwendeten Vorlagen für das Rechnungsdokument bzw. das Dokument zur schriftlichen Ablehnung einer Bestellung belegt worden (Aktivitäten "Erstelle Rechnung" und "Erzeuge Auftragsablehnung"). Die letzten zwei Ressourcen sind das System zur Kundenstammdatenverwaltung (thematische Rolle SOURCE der Aktivität "Lese Kundendatensatz anhand der Kundennummer") und der Mail-Server der Bookseller AG (thematische Rolle INSTRUMENT der Aktivität "Erzeuge und versende Auftragsablehnung").

Fünf weitere abgedeckte Eingaben sind fachliche Geschäftsregeln. Dabei handelt es sich um vier mögliche Umsatzkategorien, in die Kunden zur Bestimmung eines zu gewährenden Rabattes eingeordnet werden (thematische Rolle RULE der Aktivität "Berechne Rabatt"). Die fünfte Regel definiert die Bonitätsprüfung eines Bestandskunden (ebenfalls thematische Rolle RULE der Aktivität "Prüfe Bonität anhand des Zahlungsverhaltens"). Die verbleibenden zwei zusätzlichen Eingaben sind in die Kategorie der Werte eingeordnet worden. Dabei handelt es sich um die im Finanzbuchhaltungssystem gespeicherte Rechnung (thematische Rolle OBJECT der Aktivität "Drucke und packe Rechnung") und die Umsatzdaten eines Kunden im Finanzbuchhaltungssystem (gefordert durch die thematische Rolle RULE der Aktivität "Berechne Rabatt").

Bei der Abdeckung der erwarteten Ausgaben ist ein ähnliches Bild zu beobachten (siehe Abbildung 6.8). Absolut gesehen sind 13 zusätzliche abgedeckte Ausgaben durch thematische Raster hinzugekommen. Mit acht Ausgaben gehört der Großteil zur Kategorie der Fehlermeldungen. Dahinter verbergen sich ausschließlich Fehlermeldungen für den Fall der Nichtverfügbarkeit einer Ressource, z.B. von Dokumentvorlagen, Druckern oder Fremdsystemen. Die hinzugekommenen Fehlermeldungen korrespondieren nahezu vollständig mit den neun hinzugekommenen Ressourcen bei den Eingaben. Lediglich auf die Nichtverfügbarkeit der Kundenumsätze der letzten zwölf Monate im Finanzbuchhaltungssystem

zugehörige Warteschlangensystem gewährleistet. Dem ist entgegenzuhalten, dass es in jedem Fall eine Abhängigkeit vom Drucksystem gibt. In diesem Szenario werden alle Abhängigkeiten betrachtet (folglich auch der Drucker bzw. das Drucksystem). In einem realen Szenario müssten die Vertragsparteien an dieser Stelle abwägen, ob sie diesen Punkt in den Vertrag aufnehmen wollen oder nicht. Abhängig vom gegenseitigen Vertrauen könnte dieser Punkt dann auch ausgelassen werden.

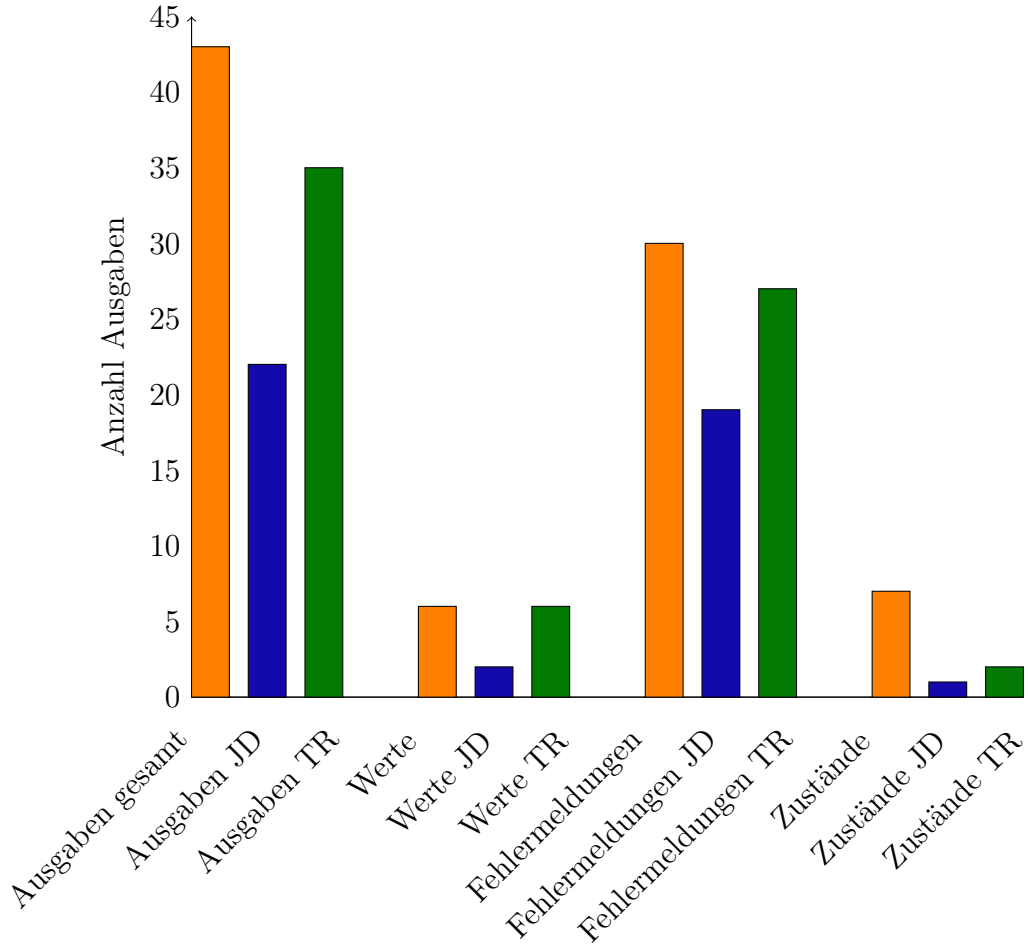


Abbildung 6.8: Abdeckung von Ausgaben der verschiedenen Kategorien

konnte nicht rückgeschlossen werden.[25] Die direkte Beziehung zwischen Ressourcen und Fehlermeldungen ergibt auch Sinn, denn schließlich könnte jede dieser Ressourcen zur Laufzeit der jeweiligen Operation nicht verfügbar sein. Für diese Fälle ist die Betriebsbereitschaft der Operation nicht gewährleistet.

Die vier hinzugekommenen Werte korrespondieren ebenfalls mit vier hinzugekommenen Eingaben. In diesem Fall handelt es sich um die vier Regeln zur Rabattberechnung anhand des Kundenumsatzes in den letzten zwölf Monaten. Aus den vier in den Regeln beschriebenen Intervallen leitet sich jeweils ein Rabattsatz als Ausgabe ab, sofern der Kunde in

[25] Diese Ressource wird bei den Eingaben durch die thematische Rolle RULE abgedeckt. Der Rückschluss auf Fehlermeldungen funktioniert immer nach dem Prinzip "Was passiert, wenn die Belegung der thematischen Rolle nicht verfügbar ist?". Die RULE ist in diesem Fall immer verfügbar (sie ist ja programmiert). Deshalb werden die Umsatzdaten als Bestandteil der RULE nicht eindeutig durch den Rückschluss abgedeckt und entsprechend auch nicht mitgezählt.

das entsprechende Intervall fällt. Daher führen die Empfehlungen der thematischen Raster automatisch zu den vier explizit erwähnten möglichen Rabattsätzen.

Auch die letzte hinzugekommene Ausgabe lässt sich auf eine hinzugekommene Eingabe zurückführen. Es handelt sich um den Inhalt der verschickten Nachricht zur Ablehnung eines Auftrags durch die Aktivität "Erzeuge und versende Auftragsablehnung". Das thematische Raster der sendenden Operationen hat diese Eingabe über die Rolle FORMAT der versendeten Nachricht abgedeckt. Daraus leitet sich ab, dass dieses Format in der verschickten E-Mail auch enthalten sein muss. Diese Ausgabe ist als Zustand (der versendeten Mail) klassifiziert worden.

Zusammenfassend lässt sich sagen, dass der hohe Zuwachs an abgedeckten Eingaben hauptsächlich aus Ressourcen und fachlichen Geschäftsregeln besteht. Während die eingehenden Werte als Parameter Bestandteile der Signatur einer Operation sind, treten diese beiden Arten von Eingaben dort nicht in Erscheinung. Aus diesem Grund sind sie von *javadoc* nicht erfasst worden. Ähnlich verhält es es sich bei den Ausgaben. Erst durch die Nennung einer Ressource im Vertrag kann auf die Ausnahmesituation der Nichtverfügbarkeit dieser Ressource geschlossen werden. Daher können große Teile möglicher Ausnahmesituationen prinzipiell nicht durch *javadoc* behandelt werden, weil die Ressourcen für *javadoc* schlicht "unsichtbar" sind. Eine Ausnahme bilden hier Ausnahmesituationen ausgelöst durch unzulässige Belegungen von Parametern. Hier sind die Konventionen von *javadoc* nahzu algorithmisch für jeden deklarierten Parameter anwendbar (Was passiert bei `null`-Belegung?, Gibt es Wertebereichsbeschränkungen? etc.).

Offen geblieben ist bisher jedoch, welche Ein- und Ausgaben nach wie vor nicht abgedeckt werden. Bei den Eingaben ist die Notwendigkeit der Systemzeit zweimal nicht erkannt worden. Diese dient als Referenzzeit zur Bestimmung der relevanten Kundenumsätze (Aktivität "Berechne Rabatt") und des Rechnungsdatums (Aktivität "Erstelle Rechnung") . In beiden Fällen ist diese Eingabe ein Spezialfall. So kann im allgemeinen Fall einer erstellenden Operation nicht auf die Relevanz des Systemdatums als Attribut des erstellten Objektes geschlossen werden. Denkbar wäre es, das OBJECT (in diesem Fall die Rechnung) mit in die Ableitung der thematischen Rollen einzubeziehen. Eine Rechnung muss in der Bundesrepublik Deutschland aufgrund rechtlicher Vorgaben ein Rechnungsdatum (Ausstellungsdatum) ausweisen (siehe § 14, Abs. 4 UStG).

Dieser Mechanismus greift allerdings nicht im Fall der Eingrenzung der relevanten Kundenumsätze zur Rabattberechnung. Die Entscheidung, den Rabatt auf Basis der Umsätze der vergangenen zwölf Monate zu gewähren, ist von der Bookseller AG getroffen wor-

den und nicht generalisierbar. Hier wäre es allerdings möglich, auf die Notwendigkeit des Datums aufgrund des Inhalts der Regel zur Rabattberechnung rückzuschließen. In dieser Regel wird der Zeitraum von zwölf Monaten erwähnt. Dieser Zeitraum muss zwingend durch ein Anfangs- oder Enddatum bestimmt werden. Nach diesem Datum könnte das thematische Raster fragen. In beiden Fällen ist die Wissensbasis für die empfohlenen Vertragsinhalte allerdings weit über den Bezeichner der Operation ausgedehnt worden. Im ersten Fall wurde ein Gesetzestext und im zweiten Fall wurden Belegungen thematischer Rollen in Kombination mit Weltwissen ("Ein Zeitraum ist bestimmbar durch einen Anfangs- oder Endzeitpunkt") hinzugezogen. Daher gehen diese Optimierungsmöglichkeiten über den Rahmen dieser Arbeit hinaus und bieten weitere Fragestellungen für die zukünftige Forschung.

6.5.4 Relevanz der vermiedenen Stille

Im vorangegangenen Abschnitt ist gezeigt worden, dass thematische Raster in Kombination mit den Empfehlungen von *javadoc* zu großen Zuwächsen bei den identifizierten Ein- und Ausgaben führen können. In erster Linie wird dadurch der Umfang der Verträge erhöht. Daraus resultiert die Frage, ob der Zuwachs an abgedeckten Ein- und Ausgaben denn überhaupt ein Gewinn ist. Zu diesem Zweck ist in Abschnitt 5.7 die Kennzahl *Grad der Vollständigkeit anhand der Relevanz* ($GdV^{Relevanz}$) entwickelt worden. Diese Kennzahl misst den Anteil von Testfällen einer Operation, die auf Basis ihres Vertrages nicht entscheidbar sind. Ein Testfall gilt als nicht entscheidbar, wenn er auf Angaben beruht, zu denen im zugehörigen Vertrag nichts geschrieben steht.

In der Abbildung 6.9 sind die identifizierten Testfälle den auf Basis von *javadoc* bzw. thematischen Rastern entscheidbaren Testfällen gegenübergestellt.[26] [27] Wie auch zuvor gibt der orange Graph die Anzahl der identifizierten Testfälle pro Operation an. Der blaue Graph zeigt, wie viele Testfälle pro Operation auf Basis der *javadoc*-Verträge entschieden werden können. Analog ist der grüne Graph für die Verträge der thematischen Raster in Kombination mit *javadoc* zu interpretieren. Insgesamt sind ca. 62% der Testfälle auf Basis der *javadoc*-Verträge entscheidbar. Für die Verträge auf Basis thematischer Raster sind dies ca. 88%. Die *javadoc*-Verträge decken für die erste Operation keinen der fünf Testfälle

[26] Die Abkürzung ET in der Legende bedeutet "Entscheidbare Testfälle".

[27] Die grafische Darstellung der Verläufe müsste korrekterweise jeweils ein Balkendiagramm sein, weil lineare Approximationen zwischen den gezählten Ein- und Ausgaben nicht zulässig sind. Da die Verläufe mit den Approximationen aber besser erkennbar sind, wurde hier auf die Stufendarstellung verzichtet.

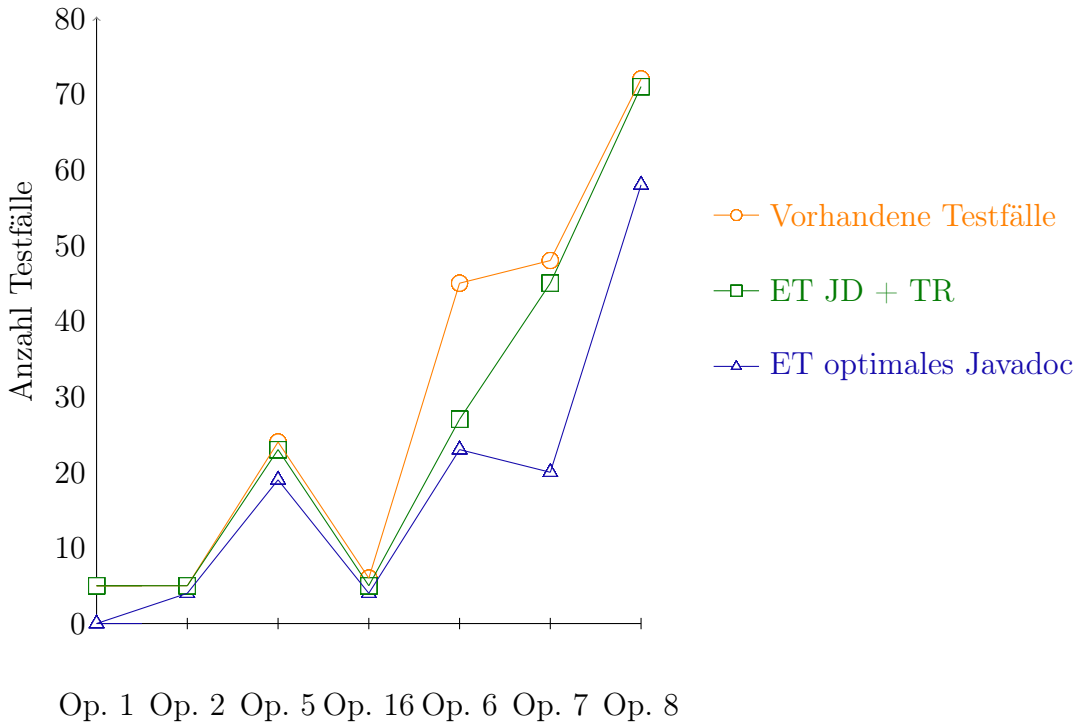


Abbildung 6.9: Entscheidbarkeit von Testfällen

ab. Für die Operationen 2, 5 und 16 liegen die Abdeckungsraten dagegen mit 80%, ca. 79% und ca. 67% deutlich höher. Aber für all diese Operationen weisen die Verträge der kombinierten Lösung (*javadoc* und thematische Raster) eine höhere Abdeckungsrate auf. Für die ersten beiden Operationen beträgt diese sogar jeweils 100%. Der Anteil entscheidbarer Testfälle für die Operationen 5 und 16 beträgt ca. 96% bzw. 83%. Bis einschließlich Operation 6 liegen die Abdeckungsraten dicht beieinander. Für die Operationen 7 und 8 grenzen sich die Verläufe der Graphen zugunsten der kombinierten Lösung deutlich ab. Deren Verträge erreichen Abdeckungsraten von 60% (Operation 6), ca. 94% (Operation 7) und ca. 99% (Operation 8), während die *javadoc*-Verträge nur ca. 51% (Operation 6), ca. 42% (Operation 7) und ca. 82% (Operation 8) der Testfälle entscheiden können.

Diese Beobachtungen scheinen zu bestätigen, dass die durch die thematischen Raster zusätzlich identifizierten Vertragsbestandteile eine hohe Relevanz aufweisen. Um diese Vermutung zu prüfen, werden im Folgenden auch bezüglich der Testfälle die folgenden zwei Fragen erörtert:

1. Durch Abdeckung welcher Testfälle konnten die thematischen Raster diese Zuwächse erreichen?

2. Welche Testfälle können nach wie vor nicht entschieden werden?

Wie zuvor bei den Ein- und Ausgaben erscheint auch zur weiteren Klärung dieser zwei Fragen eine Kategorisierung der Testfälle notwendig. Dabei sind im Rahmen der abgeleiteten Testfälle sechs Kategorien identifiziert worden (siehe Abbildung 6.10). Die erste Ebene unterscheidet zwischen Positiv- und Negativtests. Nach Pilorget wird bei einem Positivtest geprüft, ob das getestete Programm bei zulässigen Eingaben wie spezifiziert funktioniert und korrekte Resultate liefert. Der Negativtest ist eine Erweiterung des Positivtests. Hier wird geprüft, ob ein Programm bei Eingabe von laut Spezifikation nicht zulässigen Daten eine angemessene Fehlerbehandlung durchführt (z.B. durch Werfen einer bestimmten Ausnahme) (vgl. Pilorget, 2012, Seite 69).

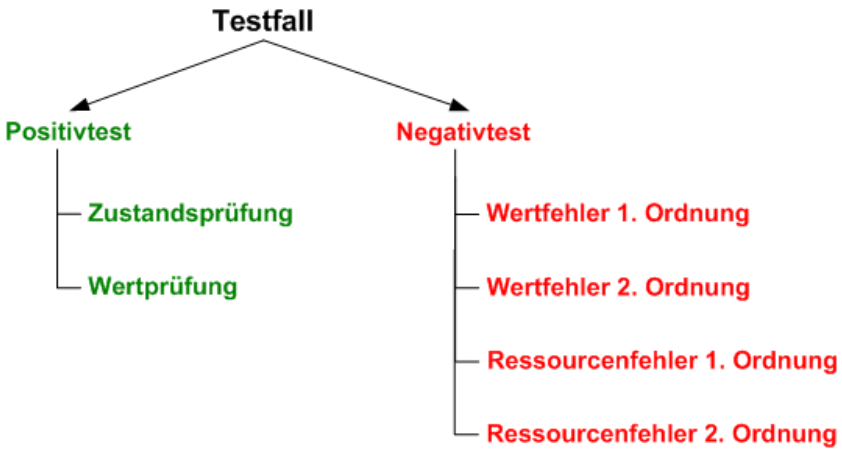


Abbildung 6.10: Identifizierte Kategorien in der Menge der betrachteten Testfälle

- **Wertprüfung (Positivtest):** Testfall, der das Resultat der getesteten Operation mit einem Referenzresultat vergleicht.
- **Zustandsprüfung (Positivtest):** Testfall, der den Zustand einer von der getesteten Operation genutzten Ressource mit einem Referenzzustand vergleicht.
- **Wertfehlerprüfung 1. Ordnung (Negativtest):** Testfall, der eine Fehlermeldung nach dem Aufruf der getesteten Operation in einer Konstellation nicht verfügbarer, aber von der Operation benötigte Werte erwartet.
- **Wertfehlerprüfung 2. Ordnung (Negativtest):** Testfall, der eine Fehlermeldung nach dem Aufruf der getesteten Operation in einer Konstellation nicht verfügbarer, aber von der Operation benötigter Werte aus Ressourcen erwartet. Ein

solcher Wert ist ein Datum, das von einer Ressource bereitgestellt wird (z.B. ein Datensatz aus einer Datenbank).

- **Ressourcenfehlerprüfung 1. Ordnung (Negativtest):** Testfall, der eine Fehlermeldung nach dem Aufruf der getesteten Operation in einer Konstellation nicht verfügbarer, aber von der Operation benötigter Ressourcen erwartet.
- **Ressourcenfehlerprüfung 2. Ordnung (Negativtest):** Testfall, der eine Fehlermeldung nach dem Aufruf der getesteten Operation in einer Konstellation nicht verfügbarer, aber von der Operation benötigter innerer Ressourcen erwartet. Eine innere Ressource ist ein Datum, das von einer Ressource bereitgestellt wird (z.B. ein Datensatz aus einer Datenbank).

Wertfehler- und Ressourcenfehlerprüfungen haben einen sehr ähnlichen Charakter. Beides sind Negativtests, die eine bestimmte geprüfte Ausnahme (Checked Exception) erwarten. Der Unterschied besteht in der Ursache der geprüften Ausnahmen: Wertfehlerprüfungen erwarten Ausnahmen aufgrund fehlender bzw. ungültiger Werte. Die von Ressourcenfehlerprüfungen erwarteten Ausnahmen resultieren hingegen aus nicht verfügbaren Ressourcen. Diese Unterscheidung ist zur Erkennung des Mehrwertes thematischer Raster sehr wichtig. Wertfehler rühren häufig aus ungültigen Aktualparametern her (Wertfehlerprüfung 1. Ordnung). Damit werden sie zu einem Großteil bereits von *javadoc* über das Tag *@param* abgedeckt. Bei Ressourcen ist dies nicht der Fall.

Auch die Unterscheidung zwischen erster und zweiter Ordnung dient dazu, Unterschiede zwischen *javadoc* und thematischen Rastern sichtbar zu machen. Als Fehler erster Ordnung wird die Nichtverfügbarkeit eines Wertes oder einer Ressource betrachtet, die direkt verwendet werden (z.B. ein SMTP-Server zum Versenden von Mails). Ein Fehler zweiter Ordnung beschreibt die Nichtverfügbarkeit einer gekapselten Ressource, z.B. einer Datei aus einem Verzeichnis. Zum Lesen dieser Datei muss auch das Verzeichnis für das Programm lesbar sein. Sollte die Datei nicht verfügbar sein, wird eine geprüfte Ausnahme (bzw. ein Fehler) zweiter Ordnung geworfen. Die Identifikation geprüfter Ausnahmen zweiter Ordnung stellt eine besondere Herausforderung dar, weil die Bezugswerte bzw. -ressourcen gekapselt sind.

Die zusätzlich durch thematische Raster entscheidbaren 53 Testfälle sind in Abbildung 6.11 auf die verschiedenen Kategorien verteilt worden. Den größten Zuwachs hat es mit 26 Testfällen durch die Behandlung von Wertfehlerprüfungen 1. Ordnung gegeben. Dahinter verbergen sich 24 Testfälle der Operation 7 (“Erstelle Rechnung”), die den Umgang

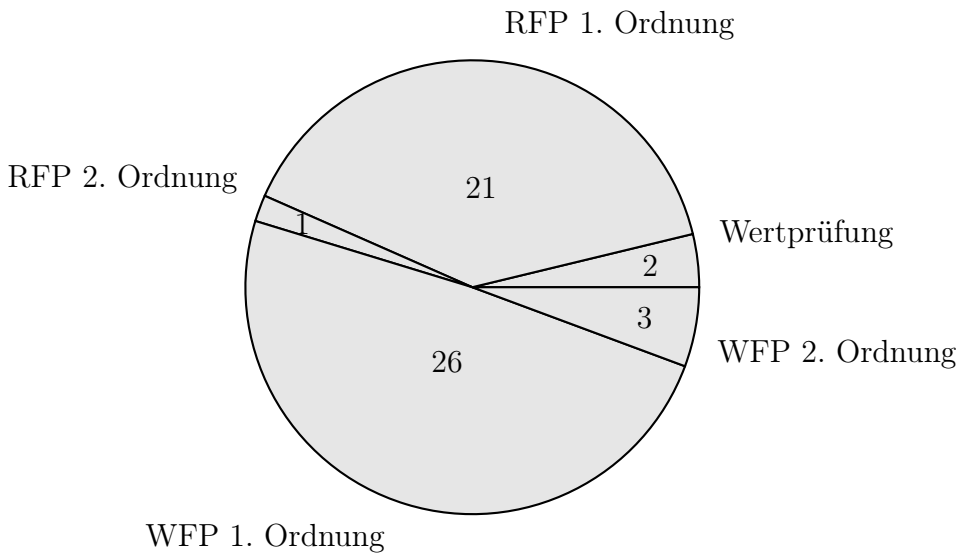


Abbildung 6.11: Verteilung der durch thematische Raster zusätzlich entscheidbaren Testfälle (RFP = Ressourcenfehlerprüfung, WFP = Wertfehlerprüfung)

der Operation mit nicht verfügbaren Kundenstammdaten prüfen. Im thematischen Raster *Erzeugende Operation* für eine Rechnung belegen die Kundenstammdaten die Rolle ATTRIBUTE. Die verbleibenden zwei Testfälle erwarten eine Fehlermeldung, sobald die Kundennummer nicht dem vorgegebenen Format entspricht. Diese Testfälle betreffen die Operation 1 ("Lese Kundenaccount anhand der Kundennummer"), die durch das Raster *Lesende Operation* abgebildet wird. Der beschriebene Fehlerfall wird durch die Nichtverfügbarkeit der thematischen Rolle PRIMARY KEY abgedeckt.

Die zweitgrößten Zuwächse hat mit 21 weiteren, zusätzlich abgedeckten Testfällen die Kategorie der Ressourcenfehlerprüfung 1. Ordnung zu verzeichnen. Dabei handelt es sich um die Fehlerbehandlung des nicht verfügbaren Druckers (10x), des nicht erreichbaren Finanzbuchhaltungssystems (6x), des nicht erreichbaren SMTP-Servers (3x) und um die nicht erreichbare Kundenstammdaten- und Auftragsverwaltung (jeweils 1x). In all diesen Fällen benannte der optimale *javadoc*-Vertrag die verwendeten Systeme nicht als benötigte Ressourcen. Diese Ressourcen werden hier über die thematischen Rollen SOURCE, DESTINATION oder INSTRUMENT abgebildet.

Die restlichen zusätzlich entschiedenen Testfälle verteilen sich auf die Kategorien Wertfehlerprüfungen 2. Ordnung (3x), Wertprüfung (2x) und Ressourcenfehlerprüfung 2. Ordnung (1x). Hinter den drei Wertfehlerprüfungen 2. Ordnung verbergen sich zwei Tests des Umgangs mit nicht verfügbaren Rechnungen (Operation 8: "Drucke und packe Rechnung")

und ein Test des Umgangs mit einem nicht verfügbaren Kundendatensatz (Operation 1: "Lese Kundenaccount anhand der Kundennummer"). Die Entscheidbarkeit dieser Testfälle durch Verträge auf Basis thematischer Raster ist auch hier durch die explizite Benennung der Ressourcen Kundenstammdatenverwaltung und Finanzbuchhaltungssystem zu erklären. Da diese Tests die genannten Ressourcen voraussetzen, müssen sie zwecks Entscheidbarkeit auch im zugehörigen Vertrag erwähnt werden. Da dies nicht der Fall ist, können alle drei Testfälle auf Basis der *javadoc*-Verträge nicht entschieden werden.

Ähnlich verhält es sich mit der verbleibenden Ressourcenfehlerprüfung 2. Ordnung. In diesem Testfall wird eine entsprechende Fehlermeldung erwartet, sofern die Vorlage für die zu erzeugende E-Mail nicht vorliegt. Da der *javadoc*-Vertrag diese Vorlage nicht erwähnt, kann der Testfall auch nicht entschieden werden. Bei Einsatz des thematischen Rasters *Sendende Operation* wird die Rolle FORMAT durch diese Ressource belegt. Daher kann in diesem Fall auch eine Aussage zur Entscheidbarkeit des Testfalls getroffen werden.

Diese Messwerte decken sich mit den Ergebnissen der ersten Analyse. Vor allem der große Zuwachs an genannten Ressourcen und den damit verbundenen Fehlermeldungen hat sich in der Anzahl der entscheidbaren Testfälle niedergeschlagen. So sind 47 der zusätzlich entscheidbaren Testfälle Negativtests, die die korrekte Behandlung der Nichtverfügbarkeit von Ressourcen bzw. Werten prüfen. Ist dieser Zuwachs ein inhaltlicher Gewinn für die Verträge? Ja, denn durch die thematischen Raster sind in erster Linie Ressourcen in die Verträge aufgenommen worden. Sie stellen Abhängigkeiten von der Laufzeitumgebung der Operationen bzw. Methoden dar, die erfüllt werden müssen. Durch ihre explizite Nennung gilt sowohl für den Anbieter bei der Implementierung als auch für den Benutzer während des Betriebs größtmögliche Transparenz. Allerdings bleiben 24 Testfälle auch bei Verwendung der thematischen Raster nicht entscheidbar. Es stellt sich die Frage, welche Testfälle das sind und weshalb sie nach wie vor nicht abgedeckt werden.

Die Verteilung der weiterhin nicht entscheidbaren Testfälle auf die genannten Kategorien ist in der Abbildung 6.12 dargestellt. Den mit Abstand größten Anteil machen die 13 Tests auf Wertfehlerprüfungen 1. Ordnung aus. In all diesen Tests werden die Operationen 6 und 7 auf den Umgang mit dem nicht verfügbaren Systemdatum geprüft. Da das Systemdatum in keinen Vertrag Eingang gefunden hat, ist folglich auch der Fehler der Nichtverfügbarkeit unbehandelt. Diese Angabe betrifft so viele Testfälle, weil das Systemdatum für diese Auswertung als Wert gezählt worden ist (vergleichbar mit einem Parameter). Es stellt damit eine elementare Eingabe dar, die zur Entscheidbarkeit vieler Testfälle die Voraussetzung bildet.

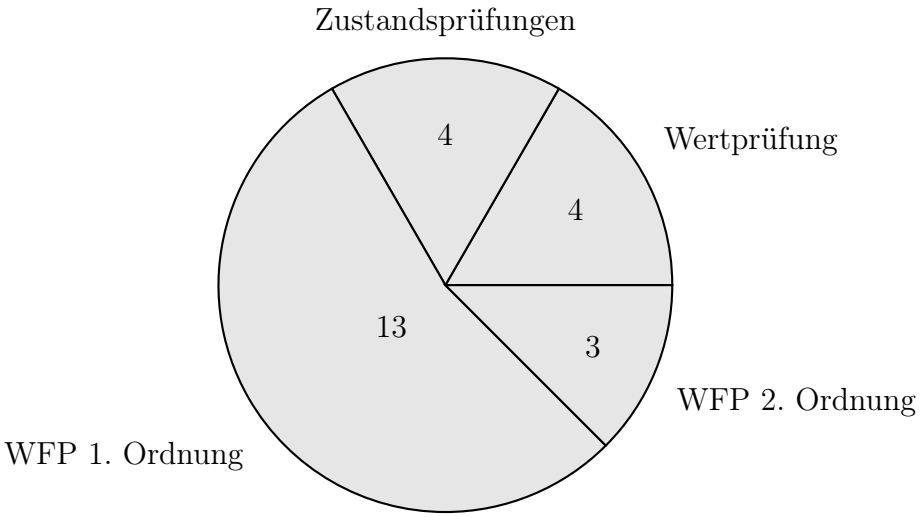


Abbildung 6.12: Verteilung der weiterhin nicht entscheidbaren Testfälle auf die verschiedenen Kategorien

Vier weiterhin nicht entscheidbare Wertprüfungen sind auf dieselbe Ursache zurückzuführen. Auch hier fehlt das Systemdatum zur Prüfung der Rabattberechnung der Operation 6. Bei den drei verbleibenden Wertfehlerprüfungen 2. Ordnung ist der ausgewertete, aber nicht im Vertrag genannte Kundenumsatz der vergangenen zwölf Monate die Ursache (Operation 6 "Berechne Rabatt"). Zusätzlich bleiben vier Testfälle der Kategorie Zustandsprüfung nicht entscheidbar. Drei davon sind auf nicht an den Aufrufer gelieferte Erfolgsmeldungen zurückzuführen (Operation 5 "Erzeuge Auftragsablehnung": Nachricht mit Ablehnung erfolgreich verschickt; Operation 8 "Drucke und packe Rechnung": Rechnung erfolgreich gedruckt; Operation 16 "Speichere Bestellung": Bestellung erfolgreich gespeichert). Ein einziger Testfall der Operation 7 ("Erstelle Rechnung") lässt sich erneut mit dem fehlenden Systemdatum als erforderliche Eingabe erklären.

Wie sind die weiterhin nicht entscheidbaren Testfälle zu bewerten? Bezüglich der Wertfehlerprüfungen gilt: Das Systemdatum bewirkt einen nach außen sichtbaren Effekt. Entweder grenzt es die Menge der betrachteten Kundenumsätze ein und liefert so die Grundlage für den gewährten Rabattsatz oder es erscheint als Datum der Rechnungsstellung auf einem Dokument, welches an Kunden geschickt wird und damit juristische Konsequenzen haben kann. Streng genommen gehört es folglich im allgemeinen Fall unbedingt in den Vertrag. Nun kann diese Position relativiert werden, wenn der Vertrag nur der Erläuterung der Operation für Konsumenten dient. Da diese das Datum nicht als Parameter

übergeben können, haben sie auch keinerlei Einfluss auf dessen Belegung und könnten es folglich vernachlässigen.

Ganz anders verhält es sich, wenn der Vertrag als Spezifikation für einen Programmierer dient. Äußert sich der Vertrag in diesem Fall nicht zu diesem Datum, besteht die Möglichkeit, dass der Programmierer über die Belegung des Datums selbst entscheidet. In diesem Fall besteht ein gewisses Restrisiko einer Fehlentscheidung. Letztendlich kann nur ein Gespräch mit der Fachabteilung dieses Risiko ausräumen. Hier wird erkennbar, dass sich die Frage nach der Relevanz einer Ein- oder Ausgabe schwer pauschal beantworten lässt. Sie ist zum Teil abhängig vom Interesse des Lesers eines Vertrages. Daher ist es sicher in jedem Fall sinnvoll, dass ein Autorenwerkzeug auf die Lücke im Vertrag hinweist. Die Entscheidung über ihre Relevanz kann der Autor dann immer noch treffen. Insofern stellt die Erkennung von derlei Werten als Eingabe einer Operation ein wichtiges Forschungsfeld für weitere zukünftige Arbeiten dar. Bezüglich der ausgelassenen Erfolgsmeldungen könnte man argumentieren, dass ein Aufruf einer Operation ohne erhaltene Fehlermeldungen durchaus als Erfolgsmeldung zu werten sei. Dieser Interpretation ist bei dieser Fallstudie nicht gefolgt worden, weil nur explizit durch das jeweilige Empfehlungsraster abgedeckte Aspekte gezählt worden sind. In jedem Fall könnte dieses Problem durch die einfache Konvention gelöst werden, bei sendenden oder speichernden Operationen immer einen Wahrheitswert als Ergebnis zu liefern. Dieser repräsentiert den Erfolg der Operation.[28] Insofern können die betreffenden drei Testfälle weitgehend vernachlässigt werden.

Ausgangspunkt dieser Fallstudie war die Forschungsfrage 7. Sie untersucht, ob der Beitrag thematischer Raster zur Vermeidung von Stille signifikant ist. In der Fallstudie konnte eine Steigerung der identifizierten Eingaben um ca. 74% und bei den identifizierten Ausgaben um 59% nachgewiesen werden. Auch bezüglich der entscheidbaren Testfälle ist eine Verbesserung zu beobachten. Die Anzahl der entscheidbaren Testfälle konnte durch thematische Raster von ca. 62% auf ca. 81% gesteigert werden. Angesichts dieser Ergebnisse ist diese Forschungsfrage für das Fallbeispiel zu bejahen. Allerdings muss betont werden, dass diesen Zahlen algorithmisch abgeleitete Verträge zugrunde liegen. Außerdem sind diese für ein Fallbeispiel bestehend aus nur sieben Operationen erstellt worden. Daher

[28] Alternativ wäre es ebenfalls denkbar, das gespeicherte Objekt bzw. den gespeicherten Datensatz zurück zu liefern. Die Konvention lautet in diesem Fall, dass dieses Resultat leer bzw. `null` (oder dergleichen) ist, sobald der Datensatz nicht gespeichert werden konnte. Außerdem kann auch der Operationsaufruf ohne abgefangene Ausnahme als Erfolgsfall angesehen werden. Unabhängig von der gewählten Konvention bleibt die Essenz gleich: Die Erfolgsmeldungen müssen nicht zwingend im Vertrag erwähnt werden.

handelt es sich hier auch um Laborwerte. Insofern sind Zweifel an der Generalisierbarkeit der ermittelten Zahlenwerte berechtigt. In diesem Zusammenhang muss allerdings darauf hingewiesen werden, dass die Operationen häufig in der Software-Entwicklung vorkommende Anwendungsfälle repräsentieren, wie z.B. das Speichern, Suchen, Prüfen oder Drucken von Datensätzen. Insofern konnte selbst bei vermeintlich einfachen Beispielen ein erhebliches Verbesserungspotential gegenüber konventionellen Ansätzen zur Empfehlung von Vertragsinhalten demonstriert werden. Dabei sind neben technischen auch Hinweise auf fachliche Geschäftsregeln aufgezeigt worden. Weitere Assoziationen menschlicher Autoren aufgrund bereits notierter Vertragsinhalte oder Erfahrungen aus vergangenen Projekten sind hier natürlich nicht berücksichtigt worden. Daher wäre es interessant zu untersuchen, ob und welche Verbesserungspotentiale durch Verträge sich bei den studentischen Probanden erschließen.

6.5.5 Vermeidung von Stille durch iDocIt!

Die Forschungsfrage 8 fragt, ob ein Software-Assistent mittels thematischer Raster bei der Vermeidung von Stille helfen kann. Zur Klärung dieser Forschungsfrage ist in Abschnitt 6.2 als Forschungsmethode das Experiment ausgewählt worden.

Dieser Teil der Untersuchung betrachtet also die von den Probanden erstellten Verträge. Die mit *javadoc* arbeitenden Probanden stellen die Kontrollgruppe ("unbehandelt") und die mit *iDocIt!* arbeitenden Probanden die Versuchsgruppe ("behandelt") dar. Die Umgebungsbedingungen werden wie in Abschnitt 6.4 beschrieben kontrolliert. Die Kausalhypothesen dieses Experimentes wenden die in Kapitel 5 vorgestellten Kennzahlen *Grad der Vollständigkeit anhand des Umfangs* ($GdV^{Umfangs}$) und *Grad der Vollständigkeit anhand der Relevanz* ($GdV^{Relevanz}$).[29] V_V sei die Menge der erstellten Verträge der Versuchsgruppe. V_K sei analog die Menge der erstellten Verträge der Kontrollgruppe und T sei die Menge der mit dem Cause-Effect-Graphing-Verfahren für die Operationen der Aufgabenstellung identifizierten Testfälle.

Allen Hypothesen liegt dieselbe Idee zugrunde: Bezüglich der Anzahl der Ein- und Ausgaben wird erwartet, dass die mit *iDocIt!* erstellten Verträge höhere Vollständigkeitsraten

[29] H_0 sei die Nullhypothese und H_1 die Alternativhypothese.

aufweisen. Dadurch, so wird erwartet, sinkt automatisch die Anzahl der nicht entscheidbaren Testfälle.

$$H_0^1 : GdV^{Umfang}(V_V, T) < GdV^{Umfang}(V_K, T) \tag{6.1}$$

$$H_1^1 : GdV^{Umfang}(V_V, T) \geq GdV^{Umfang}(V_K, T) \tag{6.2}$$

$$H_0^2 : GdV^{Relevanz}(V_V, T) < GdV^{Relevanz}(V_K, T) \tag{6.3}$$

$$H_1^2 : GdV^{Relevanz}(V_V, T) \geq GdV^{Relevanz}(V_K, T) \tag{6.4}$$

Die erhobenen Werte der Kennzahlen sind in der Tabelle 6.3 angegeben. Die im Vergleich mit der Fallstudie hohen absoluten Zahlen resultieren aus einer Normierung. In der Fallstudie sind nur zwei Lösungen miteinander verglichen worden. Im Falle des Probandenexperimentes liegen jeweils maximal sieben Lösungen pro Operation vor. Daher ist die Anzahl der erwarteten Ein- und Ausgaben jeder Operation mit den pro Operation betrachteten abgegebenen Lösungen multipliziert worden. Sofern für eine Operation unterschiedlich viele Lösungen pro Gruppe abgegeben worden sind, sind die Lösungen mit der niedrigsten Summe von Ein- und Ausgaben aus der jeweils anderen Gruppe entfernt worden. Auf diese Weise ist sichergestellt, dass für jede Operation dieselbe Anzahl von Lösungen aus den beiden Gruppen vorliegt. Diese so ermittelten vergleichbaren Werte werden als "bereinigt" bezeichnet.

Variable	***javadoc*-Gruppe**	***iDocIt!*-Gruppe**
GdV^{Umfang}	$\frac{124+78}{357} =\sim 0,566$	$\frac{94+61}{357} =\sim 0,434$
$GdV^{Relevanz}$	$\frac{44}{870} =\sim 0,051$	$\frac{55}{870} =\sim 0,063$

Tabelle 6.3: Übersicht der erhobenen "bereinigten" Kennzahlen

Als Erstes ist festzustellen, dass in den Verträgen beider Gruppen sehr viel Stille herrscht. Beide Gruppen haben nur knapp die Hälfte der erwarteten Ein- und Ausgaben in ihren Verträgen beschrieben (± 5%). Ein sehr großer Anteil der für die Testfälle relevanten Angaben fehlt in beiden Gruppen ebenfalls in sehr ähnlichem Umfang (ca. 95%). Die mit *javadoc* arbeitenden Probanden haben 17 Ein- und 30 Ausgaben mehr in ihren Verträgen erwähnt als die mit *iDocIt!* arbeitende Gruppe. Angesichts der in der

Fallstudie zuvor nachgewiesenen hohen Zuwächse dieser Kennzahlen durch thematische Raster überrascht dieses Ergebnis. Wie ist diese Beobachtung zu erklären?

Variable	**javadoc-Gruppe**		**iDocIt!-Gruppe**	
	μ	σ	μ	σ
PROJECT_EXP	9,14	6,42	17,14	15,96
SEMESTER	9,00	1,93	9,43	2,87
ECLIPSE_EXP	3,29	0,45	0,57	0,49
JAVA_EXP	3,29	0,45	3,57	0,49
SVN_EXP	3,14	0,64	3,43	0,49
BPMN_EXP	2,43	1,05	2,29	0,7
API_THEORY	2,29	1,67	2,14	1,55
THEORY	5,43	1,5	4,43	1,59

Tabelle 6.4: Übersicht von Mittelwert μ und Standardabweichung σ der erhobenen Kontrollvariablen

Zur Beantwortung dieser Frage empfiehlt es sich, zunächst die erhobenen Kontrollvariablen beider Gruppen zu prüfen (siehe Tabelle 6.4). Mit zwei Ausnahmen sind beide Gruppen bezüglich der Kontrollvariablen ziemlich ausgeglichen. Die erste Ausnahme betrifft die Projekterfahrung. Hier ist die *iDocIt!*-Gruppe deutlich besser besetzt. Dies liegt an einem Probanden, der 50 Monate als bisherige Projektarbeitszeit angegeben hat. Dies erklärt aber nicht das schlechte Abschneiden der beiden Gruppen. Interessant hierfür ist die Ausprägung der Variablen THEORY. Die *javadoc*-Probanden konnten im Durchschnitt exakt einen Punkt mehr bei den theoretischen Fragen erzielen.

Hieraus kann gefolgert werden, dass die *javadoc*-Probanden in der Anwendung der Konzepte und Konventionen hinter javadoc sicherer sind als die *iDocIt!*-Probanden bezüglich der thematischen Raster. Diese Interpretation wird durch die Anzahl der mit "Weiß ich nicht" angekreuzten Fragen untermauert. So kreuzten *iDocIt!*-Probanden im Mittel 4,14-mal bei zehn Fragen insgesamt diese Antwort an (Standardabweichung: 2,36). In der Gruppe der *javadoc*-Probanden kommt dies im Mittel nur 1,14-mal vor (Standardabweichung: 1,12).

Man kann also sagen, dass die *javadoc*-Probanden die richtigen Antworten zu kennen glaubten (und damit häufig falsch lagen). Die *iDocIt!*-Probanden hingegen haben nur geantwortet, wenn sie sich sicher waren (und damit richtig lagen). Aufgrund dieser Beobach-

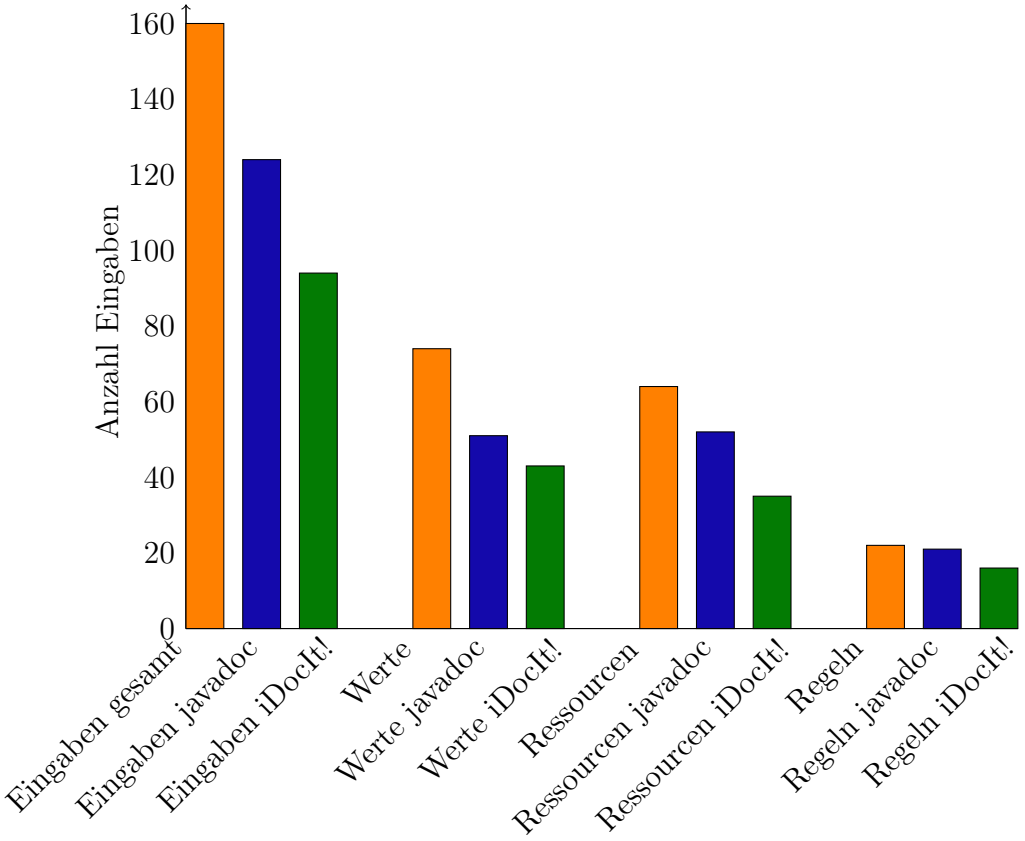


Abbildung 6.13: Abdeckung von Eingaben der verschiedenen Kategorien

tung ist zu vermuten, dass *javadoc* sehr intuitiv und ohne eine ausreichende theoretische Basis verwendet wird. Die Anwendung thematischer Raster hingegen scheint nicht intuitiv und erfordert wohl eine gewisse Einarbeitungszeit. Diese stand den Probanden offenbar nicht ausreichend zur Verfügung.

Zur weiteren Klärung des schlechten Abschneidens beider Gruppen erscheint es sinnvoll, die nicht identifizierten Ein- und Ausgaben dieser Probanden näher zu betrachten. Die Abdeckung der Eingaben ist in der Abbildung 6.13 dargestellt. Hier ist erkennbar, dass die *iDocIt!*-Probanden lediglich etwas mehr als die Hälfte der erwarteten Eingaben (94 von 160 oder ca. 59%) identifiziert haben. Die nicht identifizierten 66 Eingaben sind in der Abbildung 6.14 auf die im Rahmen der Fallstudie identifizierten Kategorien verteilt dargestellt.

Mit 35 Eingaben fällt der größte Teil in die Kategorie der “Werte”. Dies ist überraschend, da *iDocIt!* in diesem Bereich keine Neuerungen bietet und mit 8 identifizierten Werten deutlich schlechter abschneidet als *javadoc*. Es wäre zu erwarten gewesen, dass beide Werkzeuge bezüglich dieser Kategorie ungefähr gleichauf liegen. Aus diesem Grund

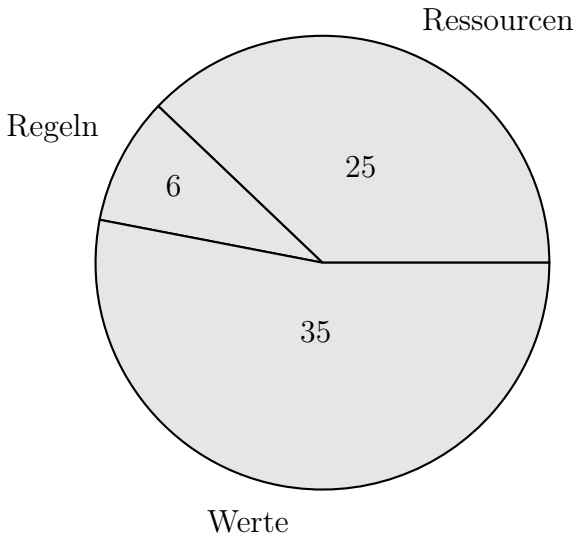


Abbildung 6.14: Verteilung der von den iDocIt!-Probanden nicht identifizierten Eingaben

sind die 35 von *iDocIt!*-Probanden nicht identifizierten Werte differenziert analysiert worden (siehe Abbildung 6.15). Folgende Kategorien lassen sich unterscheiden:

- **Nicht beschriebene Werte** sind Parameter, die zwar deklariert worden sind, aber keine Beschreibung haben.
- Unter **inneren Parametern** sind Attribute der als Parametertypen deklarierten Klassen zu verstehen, die von der Operation benötigt, aber im Vertrag nicht beschrieben werden.
- **Nicht modellierte Werte** sind benötigte Eingaben, die weder in der Signatur noch im Vertrag erwähnt werden.
- **Nicht genannte Werte** sind Eingaben, die in der Signatur nicht unbedingt, aber auf jeden Fall im Vertrag erwähnt werden müssen.

Als Beispiel für nicht genannte Werte ist der Rechnungsdatensatz zu nennen, der für das Drucken der Rechnung benötigt wird (siehe Operation 8: “Drucke und packe Rechnung”). Dieser Datensatz könnte der Operation als Parameter übergeben werden. Alternativ könnte die Operation diesen Datensatz aber auch selbst aus der Datenbank lesen, denn die zugehörige Bestellung liegt in Form eines Parameters vor. Unabhängig davon, welche Variante gewählt wird, muss der Datensatz als benötigter Wert im Vertrag erwähnt werden.

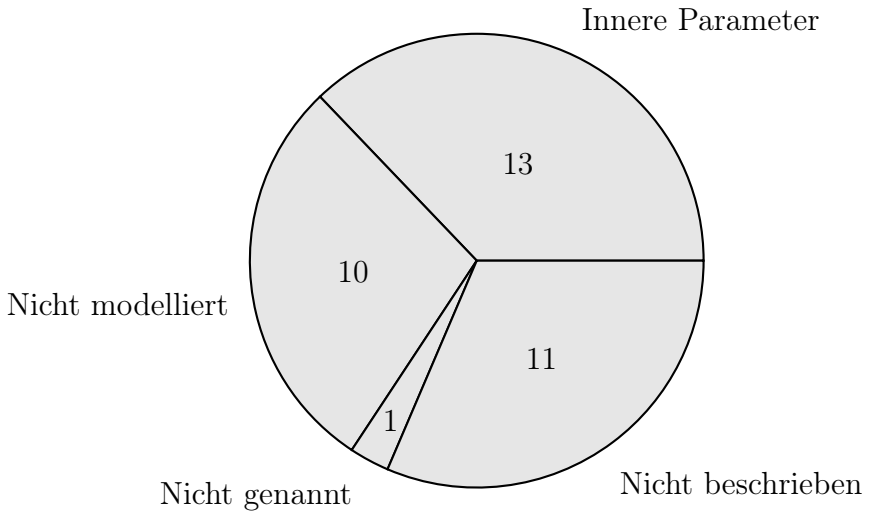


Abbildung 6.15: Verteilung der von den iDocIt!-Probanden nicht identifizierten Werte

Wie in Abbildung 6.15 dargestellt, fallen elf Werte in die Kategorie der nicht beschriebenen Parameter. Diese sind zwar in der Signatur enthalten, aber nicht kommentiert worden. Dies ist eine Verletzung der bereits seit Langem existierenden Konvention, dass Parameter zu beschreiben sind. *iDocIt!* gibt diesbezüglich keine Hinweise. Da die *javadoc*-Probanden die Parameter größtenteils beschrieben haben, ist davon auszugehen, dass die *iDocIt!*-Probanden dies in derselben Arbeitsumgebung auch getan hätten. Die Arbeitsumgebung der *javadoc*-Probanden ist der Java-Editor von Eclipse. Dieser stellt den Quelltext und den Vertrag gemeinsam in einem Textfeld dar und ermöglicht das Editieren beider Operationsbestandteile. *iDocIt!* hingegen stellt die Signatur der Operation als Baum abstrahiert in einer grafischen Oberfläche dar. Daher ist zu vermuten, dass die *iDocIt!*-Probanden die nicht beschriebenen Parameter bei der Betrachtung der Operation im Quelltexteditor ebenfalls bemerkt hätten.

Selbiges gilt für die zehn nicht modellierten Parameter. Die Entdeckungswahrscheinlichkeit dieser Parameter im Quelltext ist scheinbar höher als in der abstrakten grafischen Oberfläche von *iDocIt!*. Was bedeuten diese Erkenntnisse für den Entwurf eines Autorenwerkzeugs für Verträge? Im besten Fall besteht bei der Integration von *iDocIt!* in den Quelltexteditor die Chance, dass max. 21 von den 35 nicht identifizierten Werten durch die Probanden entdeckt werden. Im Fall von als Parameter übergebenen Objekten muss dabei auch auf Attribute dieser Objekte eingegangen werden. Kein Proband hat in seinen Verträgen definiert, dass eine Bestellung auch Artikel beinhalten muss. Demnach wäre es vertragskonform möglich, eine Bestellung mit leerer Artikelliste an die jeweiligen

Operationen zu übergeben. Fälle dieser Art machen weitere 13 der fehlenden Eingaben aus.

Des Weiteren fällt auf, dass in den Verträgen der *iDocIt!*-Gruppe nur ca. 55% der erwarteten Ressourcen beschrieben werden. In der Fallstudie ist hingegen zu sehen, dass die thematischen Raster in Kombination mit *javadoc* 100% der Ressourcen identifizieren können. Diese Beobachtung deckt sich mit der sehr geringen Anzahl der identifizierten Fehlermeldungen der *iDocIt!*-Gruppe. Es sind lediglich ca. 15% der möglichen Fehler beschrieben worden (siehe Abbildung 6.16). In der Fallstudie ist gezeigt worden, dass die thematischen Raster in Kombination mit *javadoc* 90% der erwarteten Fehlermeldungen abdecken.

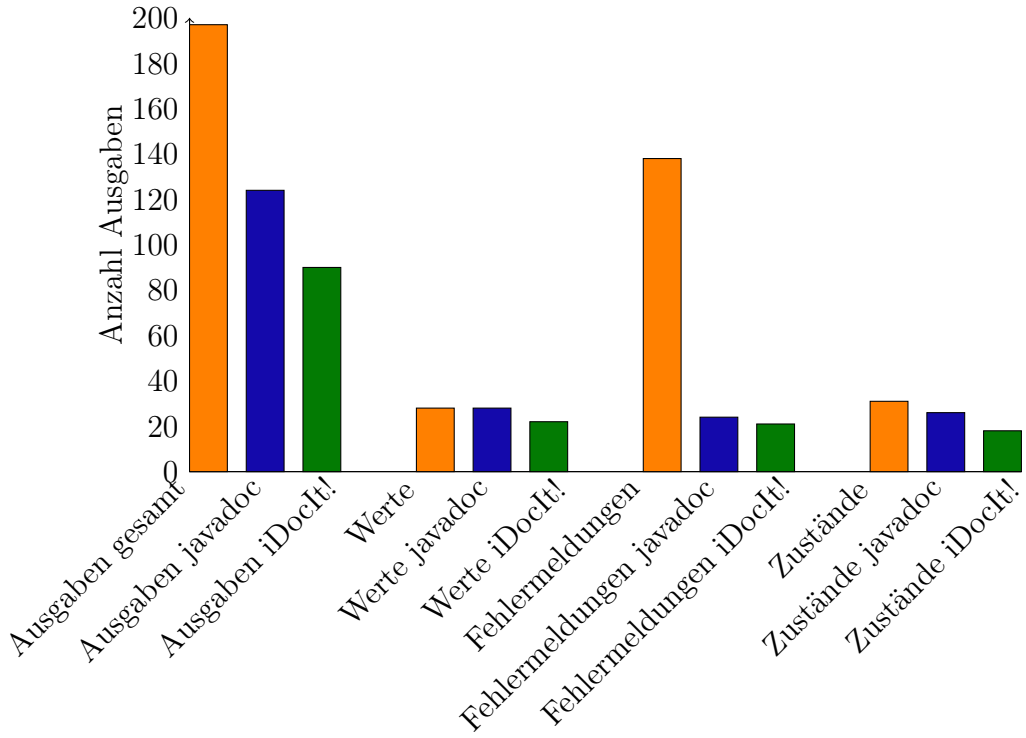


Abbildung 6.16: Abdeckung von Ausgaben der verschiedenen Kategorien

Hier ist zu vermuten, dass die Art der Darstellung in *iDocIt!* für die Probanden missverständlich war. In der aktuellen Implementierung zeigt *iDocIt!* lediglich die Namen der erwarteten thematischen Rollen pro Raster an, z.B. SOURCE. Es ist an dem Benutzer, mit der SOURCE einer suchenden Operation den Suchraum, z.B. eine Datenbank, zu assoziieren. Vermutlich sind die Hinweise von *iDocIt!* aufgrund dieser Abstraktion für die Probanden nicht ausreichend.

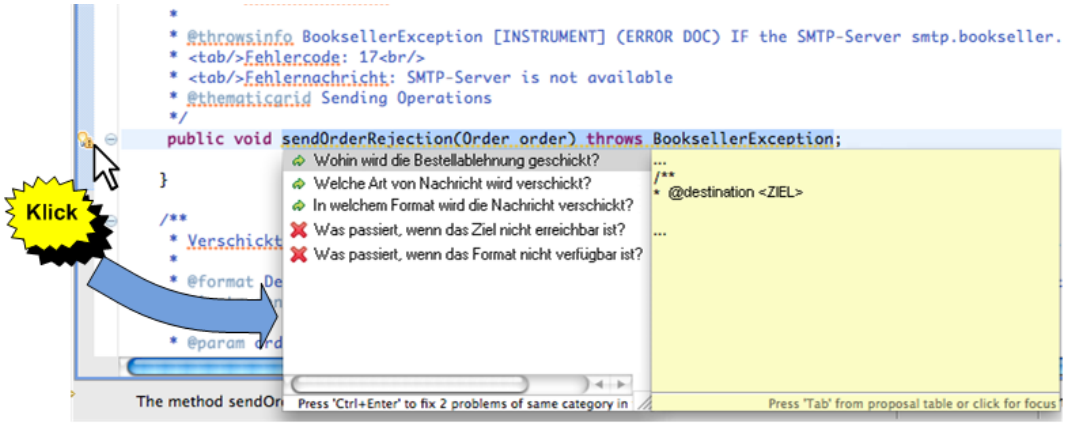


Abbildung 6.17: Integration von iDocIt! in den Quelltexteditor am Beispiel von Eclipse

Alternativ wäre es möglich, den Aufwand zur Interpretation dieser Anzeige auf ein Minimum zu begrenzen. Dies könnte durch Darstellung der Empfehlungen in Form einer Frage erfolgen. Für das obige Beispiel der SOURCE könnte eine solche Frage für die Operation 1 (“Lese Kundendatensatz anhand der Kundennummer”) lauten: “Wo sucht diese Operation nach dem Kunden?”. Ein Beispiel für eine derartige Integration von *iDocIt!* in den Quelltexteditor gibt die Abbildung 6.17.

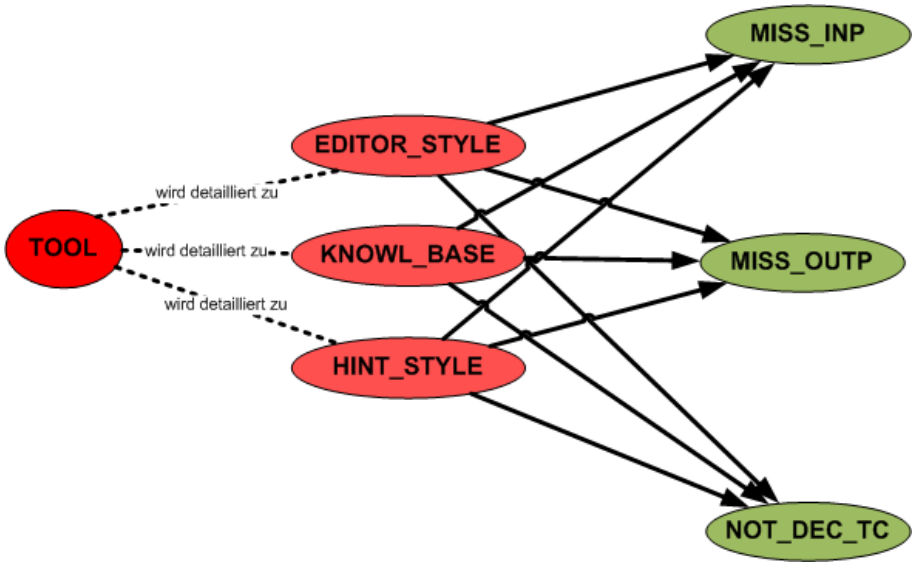


Abbildung 6.18: Angenommene Variablenbeziehungen zwischen unabhängigen und abhängigen Variablen nach Durchführung des Experimentes

Zusammenfassend ist das zuvor entwickelte Modell der Beziehungen zwischen abhängigen und unabhängigen Variablen zu verfeinern. Damit kann die Forschungsfrage 8 noch

nicht bejaht werden. Hier besteht noch viel Raum für zukünftige Forschungsvorhaben. Das an dieser Stelle vorgeschlagene korrigierte Modell wird in Abbildung 6.18 dargestellt. Die Variable TOOL ist hier in drei unabhängige Variablen aufgespalten worden:

- Die Variable **EDITOR_STYLE** ist nominalskaliert mit den Ausprägungen "isoliert" und "integriert". Unter einem isolierten Editorstil wird ein eigener Bereich in der Entwicklungsumgebung des Programmierers verstanden, welcher die Empfehlungen des thematischen Rasters visualisiert und den Benutzer durch die Vertragserstellung führt. Integrierter Editorstil hingegen bedeutet, dass die Empfehlungen direkt im Quelltexteditor dargestellt werden und nicht als eigenständiges Modul der Entwicklungsumgebung erkennbar sind (siehe Abbildung 6.17).

- Die Variable **KNOWL_BASE** ist ebenfalls nominalskaliert mit den Ausprägungen "javadoc" und "javadoc+tr". Sie misst den Umfang der Konventionen, auf denen die Empfehlungen des Werkzeugs basieren.

- Auch die Variable **HINT_STYLE** ist nominalskaliert mit den Ausprägungen "aussage", "rollenname" und "frage". Diese Variable misst die textuelle Form der Empfehlung. Ihre Bedeutung kann exemplarisch durch die Belegung der Rolle SOURCE einer suchenden Operation veranschaulicht werden, die Kundendatensätze für die gegebene Kundennummer liefert:

  - Aussage: "Die Quelle der durchsuchten Kunden ist noch nicht beschrieben worden. / Die Menge der durchsuchten Kunden ist noch nicht abgegrenzt worden."
  - Rollenname: "SOURCE"
  - Frage: "Woher stammen die durchsuchten Kundendatensätze? / Welche Kunden sind in der Menge der durchsuchten Kunden enthalten?"

6.6 Zusammenfassung

Im Rahmen dieser Evaluierung ist gezeigt worden, dass thematische Raster in Kombination mit einem konventionellen Verfahren wie *javadoc* zu hohen Zuwachsraten bezüglich der notwendigen und im Vertrag genannten Ein- und Ausgaben führen können. Zwei zusammengefasste Antworten auf die behandelten Forschungsfragen sind in Tabelle 6.5 angegeben.

Forschungsfrage 7: Können thematische Raster signifikant zur Vermeidung von Stille beitragen (konzeptuelle Perspektive)?
Ja, sie helfen vorrangig bei der Identifikation benötigter Ressourcen sowie der damit verbundenen Fehlerfälle. Außerdem unterstützen sie bei der Identifikation benötigter Regeln und Algorithmen. So konnte der Anteil identifizierter Eingaben durch thematische Raster in dieser Studie von ca. 54% auf ca. 94% und identifizierter Ausgaben von ca. 51% auf ca. 81% gesteigert werden. Die Anzahl entscheidbarer Testfälle konnte durch thematische Raster von ca. 62% auf ca. 88% gesteigert werden.
Forschungsfrage 8: Kann ein Software-Werkzeug auf Basis thematischer Raster Programmierer bei der Vermeidung von Stille unterstützen (integrierte Perspektive)?
Vermutlich, sofern die Empfehlungen der thematischen Raster den Autoren in gewohnter Umgebung und sprechender Weise präsentiert werden. Die genaue Umsetzung dieser Erkenntnisse in Werkzeuge obliegt zukünftigen Forschungsvorhaben.

Tabelle 6.5: Beantwortung der Forschungsfragen des Kapitels 6

In der beschriebenen Fallstudie ist eine Steigerung der im Vertrag genannten Ein- und Ausgaben von insgesamt ca. 66% nachgewiesen worden. Dieser hohe Zuwachs ist vorwiegend durch die Identifikation benötigter Ressourcen und damit verbundener Fehlersituationen zu erklären. Ein weiterer gewichtiger Faktor ist die Identifikation von fachlichen Geschäftsregeln und Formeln zur Berechnung der Operationsresultate. Thematische Raster haben damit das Potential, einen nicht zu achtenden Beitrag zur Senkung der Entwicklungskosten von Software-Systemen zu leisten. Sie helfen Software-Architekten dabei, in der Phase des Feinentwurfs Abhängigkeiten von Ressourcen und daraus resultierende Ausnahmesituationen zu erkennen und zu berücksichtigen. So kann der Architekt die systemweite Behandlung von derlei Ausnahmen gewährleisten. Auf diese Weise werden die Anzahl nachträglicher Systemänderungen zur Ausnahmebehandlung reduziert und somit Entwicklungskosten eingespart. Außerdem senkt der Architekt durch Einsatz der thematischen Raster während der Formulierung die Anzahl potentieller Lücken im Vertrag. Dies vermindert das Fehlerrisiko bei der Aufwandsschätzung z.B. durch die nicht berücksichtigte Integration von Fremdsystemen.

Auch während der Implementierungs- und Testphase tragen thematische Raster zur Aufwandsreduktion bei. Die Programmierer erhalten eine vollständigere, detailliertere Spezifikation. Vor allem bei über mehrere Standorte oder Länder verteilten Entwicklungsteams trägt dies zu einem gemeinsamen und möglichst identischen Verständnis der umzusetzenden Aufgaben bei. Ferner können die Regeln und Formeln sowie die identifizierten Ressourcen und die damit verbundenen Fehlermeldungen beim Entwurf der Komponententests und -integrationstests berücksichtigt werden. Dies führt zu einer erhöhten Testabdeckung und Robustheit des Systems bei geringeren Aufwänden. Zudem erreicht der Architekt Rechtssicherheit für seine Komponente durch explizit spezifizierte Ressourcen, die in der Laufzeitumgebung erwartet werden.

Die Anwendung thematischer Raster ist aber auch mit der Beachtung einer großen Menge an Regeln verbunden. Neben der Zuordnung von Verben und thematischen Rollen zu den Rastern fallen auch die raster- und rollenbasierten Regeln in diese Menge. Es ist nicht unwahrscheinlich, dass der Aufwand zur Beherrschung dieser Menge an Regeln den Gewinn an Vollständigkeit kompensieren kann. Schließlich ist die Vermeidung von Stille nur bei Anwendung aller Regeln zu gewährleisten. Daher hängt der Erfolg thematischer Raster zu einem Großteil von der Integration dieses Konzeptes in die genutzten Werkzeuge ab. Diese Evaluierung konnte zusätzlich auch einige wichtige Erkenntnisse bezüglich der Entwicklung von derlei Werkzeugen liefern:

- Die Empfehlungen der thematischen Raster sollten in bereits eingesetzte Entwicklungsumgebungen integriert werden.
- Die optischen Veränderungen der grafischen Benutzeroberfläche sollten sich dabei auf ein Minimum beschränken. Für Programmierer hieße das, dass die Empfehlungen direkt im Quelltext-Editor angezeigt werden.
- Die angezeigten Hinweise müssen sich dabei möglichst spezifisch an der jeweiligen Problemstellung des Nutzers orientieren.
- Die Wissensbasis zur Generierung der Hinweise muss neben den thematischen Rastern zusätzlich bestehende Konventionen, wie z.B. die Beschreibung des Wertebereiches von Parametern, abdecken.

Die Umsetzung dieser Anforderungen in Werkzeuge für verschiedene Anwendungsfälle wie die Software-Spezifikation, Prozessmodellierung oder Testfallkonstruktion bietet viel Raum für zukünftige Forschungsvorhaben.

Kapitel 7

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit besteht in der Entwicklung einer Methode, um Stille in API-Verträgen von Operationen identifizieren zu können. Der hier entwickelte Ansatz schließt aufgrund linguistischer Analysen des Operationsbezeichners auf notwendige Vertragsinhalte. Dazu werden die betrachteten Operationen aufgrund des Prädikates in ihrem Bezeichner in verschiedene Kategorien eingeteilt (thematische Raster). Jeder Kategorie sind in einem Lexikon definierte notwendige Vertragsinhalte zugeordnet. Diese Vertragsinhalte betreffen der Operation zugrunde liegende Regeln, Algorithmen, Ressourcen und auch mögliche Fehlerfälle.

Um den Mehrwert der neu entwickelten Methode zu demonstrieren, ist zusätzlich ein quantitatives Messverfahren für Stille entwickelt worden. Dieses Verfahren bewertet sowohl den Umfang als auch die Relevanz auf Basis von systematisch erzeugten Testfällen für die betreffende Operation. Eine detailliertere inhaltliche Zusammenfassung dieser Untersuchung geben die Tabellen 7.1, 7.2 und 7.3. Angesichts der beschriebenen Ergebnisse ist insgesamt festzustellen, dass mit thematischen Rastern ein transparenter und umfassender Ansatz zur Erreichung des ausgegebenen Ziels entwickelt worden ist (weitere Betrachtungen folgen nach den folgenden Tabellen).

Recherchefrage 1: Welche Inhalte gehören in einen API-Vertrag?
Die Verträge sind so knapp wie möglich zu halten. Jeder Vertragsinhalt muss zur Spezifikation der zu erbringenden Leistung unerlässlich sein (Geheimnisprinzip). Die Leistung wird durch Vorbedingungen und Nachbedingungen spezifiziert. Bei Realisierungsverträgen kann das Geheimnisprinzip abgeschwächt werden (siehe Antwort auf Recherchefrage 2). In diesem Fall ist die zulässige Verwendung der zusätzlich preisgegebenen Angaben explizit zu reglementieren. Verhandlungsgegenstände sind konkret z.B. die Bereitstellung von Ressourcen oder die Einhaltung von Wertebereichsbeschränkungen.
Recherchefrage 2: Gibt es verschiedene Arten von API-Verträgen? Falls ja, welche Arten sind das?
Unterschieden werden Nutzungs- und Realisierungsverträge. Ein Nutzungsvertrag sichert dem Benutzer Leistungen nach einem abstrakten Konzept zu. Mittels eines Realisierungsvertrages garantiert ein Anbieter eine konkrete Implementierung eines Nutzungsvertrages. Dabei können auch wichtige Details der Implementierung beschrieben werden, welche sie besonders kennzeichnen (Abschwächung des Geheimnisprinzips).
Recherchefrage 3: In welcher Form werden an welchem Ort welche Inhalte der API-Verträge hinterlegt?
API-Verträge bestehen meist aus einer Mischung von formalen, natürlichsprachlichen oder grafischen Inhalten. Ihre Bestandteile stehen direkt im Quelltext und können programmatisch zu einem Dokument zusammengesetzt werden. Die Tabelle 3.1 gibt eine Übersicht über konkrete Vorschläge für Vertragsinhalte.
Recherchefrage 4: Welche Ansätze zur Vermeidung von Stille existieren derzeit?
Zu unterscheiden sind prüflistenbasierte und quelltextbasierte Verfahren. Erstere sind sehr universell einsetzbar und haben daher einen sehr allgemeinen Charakter. Sie müssen aber manuell angewendet werden und bieten ein hohes Risiko von Inkonsistenzen im API-Vertrag. Letztere sind sehr zuverlässig, aber gleichzeitig hochgradig spezialisiert. Sie benötigen zudem Zugriff auf den Quelltext.

Tabelle 7.1: Zusammenfassung der gegebenen Antworten auf Recherchefragen dieser Dissertation

Forschungsfrage 1: Können Bezeichner auch bei der Vollständigkeitsprüfung eines API-Vertrages helfen?
Ja. Ein Verb kann durch Forderung von weiteren Argumenten (thematische Rollen) seine Bedeutung spezifizieren (Valenz). Enthält ein Operationsbezeichner ein Verb, so beschreiben die von diesem Verb geforderten thematischen Rollen einzelne Charakteristika der Operation. Die thematischen Rollen eines Verbs werden in einem thematischen Raster katalogisiert. Sofern aus dem Vertrag der Operation keine Belegung für eine der vom Verb geforderten thematischen Rollen hervorgeht, ist dies ein Anzeichen für Stille.
Forschungsfrage 2: Werden verschiedenartige Verben in signifikanter Häufigkeit zur Bildung von Operationsbezeichnern verwendet?
Sowohl als auch: In einem Korpus von 186.580 Web Service-Operationen konnten in 80,9% der Operationsbezeichner 830 verschiedene englische Verben identifiziert werden. Daher ist davon auszugehen, dass englische Verben häufig bei der Formulierung von Operationsbezeichnern verwendet werden. Allerdings genügt die Häufigkeitsverteilung der Verben wahrscheinlich einer Exponentialverteilung. Daher ist davon auszugehen, dass Programmierer im praktischen Einsatz Unterstützung bei der Formulierung semantisch gehaltvoller Bezeichner benötigen.
Rechercherfrage 5: Existieren thematische Rollen bzw. Raster im Kontext softwaretechnischer Arbeiten, die für die Gestaltung von API-Verträgen verwendet werden können?
Ja. Die Arbeiten von Gelhausen sowie von Girardi und Ibrahim liefern einen Umfang von 69 thematischen Rollen für software-technische Kontexte. Aber bereits mit sehr einfachen Beispielen lässt sich zeigen, dass diese Rollen nicht hinreichend für API-Verträge sind. Ein existierender Katalog thematischer Raster konnte bei den Recherchen nicht identifiziert werden.
Forschungsfrage 3: Welche thematischen Rollen und Raster sind für die Gestaltung von API-Verträgen notwendig?
Der Anhang C enthält 44 thematische Rollen und 20 Raster, die empirisch auf Basis des seekda-Korpus ermittelt worden sind. Diese Vorschläge dienen als Ausgangspunkt zur Identifikation von Stille in API-Verträgen.

Tabelle 7.2: Zusammenfassung der gegebenen Antworten auf Forschungs- und Rechercherfragen dieser Dissertation

Forschungsfrage 4: Wie skalieren thematische Rollen und Raster für API-Verträge?
Sehr gut. Auf Basis der Analyse von nur 287 Operationen und des Einsatzes in Projekten der AKRA GmbH konnten 20 thematische Raster ermittelt werden. Diese Raster klassifizieren über 90% der Operationen des seekda-Korpus. Außerdem erreichen sie mit einem relativ geringen Bestand an Verben eine sehr zuverlässige Klassifikation der Operationen.
Forschungsfrage 5: Wie kann der Umfang von Stille in einem Vertrag V ohne Analyse des Quelltextes gemessen werden?
Testfälle helfen dabei, nicht an der Signatur sichtbare Charakteristika zu identifizieren. Alle Charakteristika der Operation werden gezählt. Für jedes vorhandene Charakteristikum wird geprüft, ob es aus dem Vertrag der Operation hervorgeht. Dies ermöglicht die Bestimmung des Umfangs von Stille.
Forschungsfrage 6: Wie kann die Relevanz der Stille in einem Vertrag V ohne Analyse des Quelltextes gemessen werden?
Die Relevanz des Charakteristikums bemisst sich nach der Anzahl der Testfälle, die es betrifft.
Forschungsfrage 7: Können thematische Raster überhaupt signifikant zur Vermeidung von Stille beitragen (konzeptuelle Perspektive)?
Ja, sie helfen vorrangig bei der Identifikation benötigter Ressourcen sowie der damit verbundenen Fehlerfälle. Außerdem unterstützen sie bei der Identifikation benötigter Regeln und Algorithmen. So konnte der Anteil identifizierter Eingaben durch thematische Raster in dieser Studie von ca. 54% auf ca. 94% und identifizierter Ausgaben von ca. 51% auf ca. 81% gesteigert werden. Die Anzahl entscheidbarer Testfälle konnte durch thematische Raster von ca. 62% auf ca. 88% gesteigert werden.
Forschungsfrage 8: Kann ein Software-Werkzeug auf Basis thematischer Raster Programmierer bei der Vermeidung von Stille unterstützen (integrierte Perspektive)?
Vermutlich, sofern die Empfehlungen der thematischen Raster den Autoren in gewohnter Umgebung und sprechender Weise präsentiert werden. Die genaue Umsetzung dieser Erkenntnisse in Werkzeuge obliegt zukünftigen Forschungsvorhaben.

Tabelle 7.3: Zusammenfassung der gegebenen Antworten auf Forschungsfragen dieser Dissertation

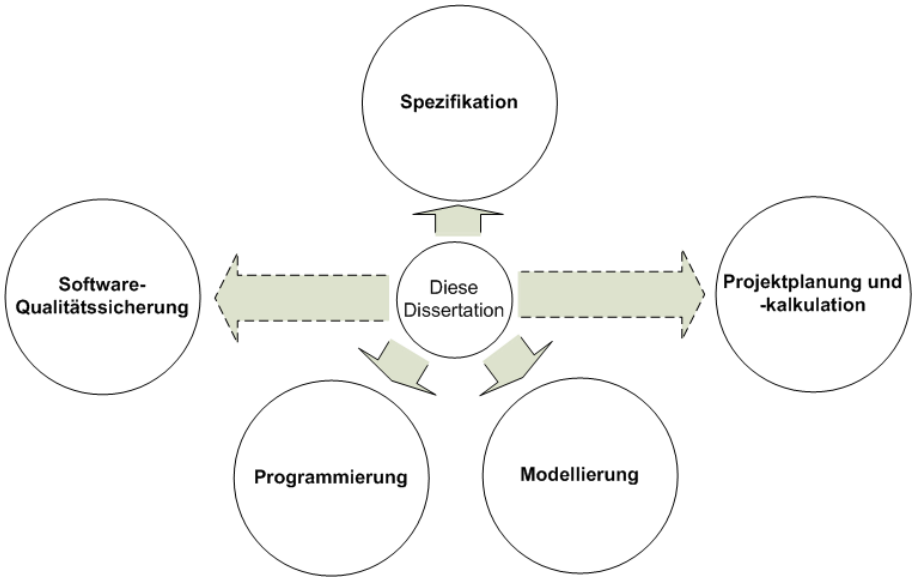


Abbildung 7.1: Übersicht der von dieser Dissertation berührten Gebiete der Informatik

Die Abbildung 7.1 gibt einen Überblick über die von den Ergebnissen dieser Dissertation berührten Gebiete der Informatik. Sehr starke Berührungspunkte bestehen mit den Gebieten der *Spezifikation*, der *Modellierung* und der *Programmierung*. Schwächere Berührungen bestehen mit der *Software-Qualitätssicherung* und der *Projektplanung und -kalkulation*.

Als Erstes wird das Gebiet der *Spezifikation* betrachtet. Diese Arbeit hat gezeigt, dass sich aufgrund der linguistischen Analyse ihres Bezeichners spezifische Erweiterungen für Prüflisten zur Kontrolle des Operationsvertrages erzeugen lassen. Andere Ansätze zur Vertragsgestaltung beinhalten diesen Aspekt nicht, sondern geben für jede Operation immer dieselben Prüfpunkte vor. Das hat zur Folge, dass Prüflisten sehr lang werden und einzelne dieser vielen Prüfpunkte nicht für jede Operation relevant sind. Sie müssen daher ausgelassen werden.

Diese Erkenntnis wirft die Frage auf, ob und welche weiteren Wissensquellen neben dem Bezeichner für die Erzeugung noch spezifischerer Prüflisten herangezogen werden können. Als Beispiel sei hier auf das gesetzlich geforderte Datum in der Rechnung aus dem Szenario der Evaluierung verwiesen, welches nicht Bestandteil der jeweiligen Signaturen war. Ebenso wäre es möglich, mit thematischen Rastern operations- und branchenspezifische (und damit fachliche!) Standards zur Spezifikation und Dokumentation von APIs zu formulieren. Sowohl Auftraggeber als auch -nehmer profitieren von der höheren Transparenz aufgrund der expliziten Erwähnung benötigter Ressourcen, etc. Ein auf

Basis thematischer Raster entworfener Standard zur Spezifikation von APIs könnte beispielsweise zur Klärung von Rechtsstreitigkeiten eingesetzt werden, wenn eine gelieferte Software-Komponente nicht den tatsächlichen Anforderungen genügt.

Des Weiteren besteht die Frage der Verwendbarkeit dieses Ansatzes in anderen Spezifikationsformen, z.B. einem UML-Klassen - oder -Zustandsdiagramm. Diese Überlegung führt zum zweiten berührten Gebiet: der *Modellierung*. Durch Berücksichtigung der Semantik natürlicher Sprache (z.B. in Form thematischer Rollen und Raster) könnte eine Vielzahl von Modellen auf Lücken geprüft werden, in denen Verben in Form von natürlichsprachlichen Beschriftungen bestimmter Modellelemente vorkommen. Die Arbeiten von Mendling et al. oder Körner zeigen, dass gerade in der jüngeren Vergangenheit verschiedene Untersuchungen vermehrt die Schlüsselrolle von Verben in Aktivitätsbeschriftungen von Geschäftsprozessmodellen herausarbeiten.[1] In diesem Zusammenhang können thematische Raster Hinweise auf fehlende Modellbestandteile liefern, die in einer Beziehung zu den analysierten Aktiväten stehen (z.B. die nicht modellierte Kundenkartei, welche von einer Aktivität "Suche Kunden anhand des Nachnamens" verwendet wird).

Besonders reizvoll ist in diesem Zusammenhang der Gedanke eines einheitlichen Lexikons thematischer Raster - sowohl auf Seiten der Spezifikations- als auch der Entwicklungsabteilung. Ein solches Lexikon würde vermutlich einen signifikanten Beitrag zum gegenseitigen Bewusstsein der verschiedenen Begriffswelten in den Abteilungen leisten. Auf diese Weise könnten aufgrund des transparenten Vokabulars Missverständnisse vermieden werden.

In vielen der angesprochenen Entwicklungsabteilungen ist das dritte berührte Gebiet angesiedelt: die *Programmierung*. Diese Dissertation diskutiert das Problem nicht oder wenig aussagekräftiger Bezeichner. Sie liefert neben sprachlichen Konstruktionshinweisen auch einen Katalog von Verben für Operationsbezeichner. Außerdem können durch die thematischen Raster Abhängigkeiten von bestimmten Ressourcen oder die systemweite Behandlung von Ausnahmefällen bereits zu einem sehr frühen Zeitpunkt des Entwicklungsprozesses bzw. der Anforderungsaufnahme berücksichtigt und somit Entwicklungszeit und -kosten für die sonst sehr viel später als notwendig erkannten Anpassungen gespart werden.

Die vorgeschlagenen Konventionen für die Wahl von Verben in Bezeichnern erleichtern Programmierern das Verständnis von und die Orientierung insbesondere in fremden

[1] Siehe die Arbeiten von Mendling et al. in (Mendling und Recker (2008) und Mendling und Reijers (2008)) sowie die Arbeit von Körner (Körner (2009)).

Programmen. Somit wird die Frage nach firmenübergreifenden oder sogar internationalen Standards zur Benennung von Software-Operationen aufgeworfen. Durch derartige Standards könnten z.B. Einarbeitungs- und Orientierungsaufwände von Programmierern in fremde Software-Systeme erheblich gesenkt werden. Gleichzeitig erhielten Programmierer und Software-Architekten durch die thematischen Raster ein gemeinsames Vokabular, mittels dessen Kommunikationsaufwände und Missverständnisse weiter reduzieren ließen.

Ein weiteres zukünftiges Forschungsfeld in der Programmierung hat die empirische Analyse der Häufigkeitsverteilung englischer Verben in Operationsbezeichnern aufgezeigt: Eine sehr kleine Menge von Verben wird sehr häufig verwendet. Dies lässt auf semantisch wenig gehaltvolle Bezeichner schließen. Insofern ist es an späteren Forschungsvorhaben, zu untersuchen, nach welchen weiteren Regeln sich präzise Bezeichner zu einer Operation formulieren lassen. An dieser Stelle wäre ebenfalls eine Überprüfung denkbar, inwieweit der Programmierer bzw. Software-Architekt bei der Formulierung des Bezeichners durch einen Software-Assistenten unterstützt werden kann.

Die Gebiete der *Software-Qualitätssicherung* und der *Projektplanung und -kalkulation* profitieren indirekt von der in dieser Arbeit entwickelten Methode zur Vervollständigung von API-Verträgen. Auf Basis vervollständigter Verträge lassen sich verlässlichere Aufwandsschätzungen für Implementierungsarbeiten anfertigen. So wird eine im Gegensatz zu lückenhaften API-Verträgen realistischere Kostenkalkulation ermöglicht. Ferner kann sich ein Hersteller durch Selbstkontrolle der API-Verträge seiner Komponente und die Vermeidung von Lücken darin vor Nacherfüllungsforderungen seiner Kunden schützen. Ebenso hat der Auftraggeber bzw. Käufer einer Komponente nur auf Basis vollständiger API-Verträge die Möglichkeit zum Abgleich der Zusicherungen und Anforderungen der Komponente mit den Gegebenheiten seiner Betriebsumgebung.

Unabhängig von all den gewonnenen Erkenntnissen ergibt sich zum Abschluss dieser Arbeit auch noch eine weitere Einsicht: Auch mit thematischen Rastern kann kein API-Vertrag automatisiert erzeugt oder inhaltlich ausgewertet werden. Es bedarf auch weiterhin verantwortungsbewusster und erfahrener menschlicher Autoren zur Formulierung und Interpretation der Verträge. An dieser Stelle kann nur darüber gemutmaßt werden, ob sich das jemals ändern wird - sehr wahrscheinlich nicht. So zeigt sich am Ende auch dieser Arbeit einmal mehr die Begrenztheit maschineller Verfahren und die Einzigartigkeit menschlicher Intelligenz.

Anhang A

Prüfliste für Schnittstellenverträge

Bei der nachstehenden Prüfliste handelt es sich um eine vom Autoren dieser Arbeit angefertigte Übersetzung des Schemas zur Schnittstellen-Dokumentation von Clements et al. (vgl. Clements et al., 2003, Seite 228 - 233). Das Schema ist in die Struktur einer Prüfliste nach Grochla überführt worden (vgl. Grochla et al., 1986, Seite 29). Ziel ist es, die Verwendung des Schemas in Form einer Prüfliste zu veranschaulichen. Clements et al. fassen bei der Formulierung des Schemas die in dieser Arbeit voneinander abgegrenzten Begriffe API-Nutzungsvertrag, API-Realisierungsvertrag und API-Dokumentation unter dem Begriff Dokumentation zusammen.

Prüfliste: Dokumentation einer Schnittstelle					
Nr.	Prüffrage	Ja	Teilweise	Nein	Bemerkungen
Prüfbereich 1: Identität Schnittstelle					
01	Hat die Schnittstelle einen Namen?	☐	☐	☐	
02	Hat die Schnittstelle eine Versionsnummer?	☐	☐	☐	
Prüfbereich 2: Angebotene Operationen **Prüfaspekt a:** Syntax der Operationen					
03	Ist die Syntax eines gültigen Aufrufes der Operation beschrieben (falls notwendig)?	☐	☐	☐	
04	Sind die Bedeutungen der Formalparameter beschrieben?	☐	☐	☐	
05	Sind die zulässigen Wertebereiche aller Formalparameter beschrieben?	☐	☐	☐	
Prüfaspekt b: Bedeutung der Operationen					
06	Ist beschrieben, wie alle Resultate der Operation gelesen werden können?	☐	☐	☐	
07	Sind die von der Operation bewirkten Zustandsänderungen der Implementierung beschrieben?	☐	☐	☐	
08	Falls die Operation unterbrochen werden kann: sind alle möglichen Seiteneffekte beschrieben?	☐	☐	☐	

Nr.	Prüffrage	Ja	Teilweise	Nein	Bemerkungen
09	Sind die von der Operation ausgelösten Ereignisse oder versendete Nachrichten beschrieben?	☐	☐	☐	
10	Sind die von der Operation bewirkten Verhaltensänderungen anderer Operationen der Schnittstelle beschrieben?	☐	☐	☐	
11	Sind die von der Operation bewirkten, für menschliche Benutzer sichtbaren, Effekte beschrieben (z.B. Anzeigen auf Displays bei eingebetetten Systemen)?	☐	☐	☐	
12	Ist beschrieben, ob die Operation unterbrechbar oder atomar ist?	☐	☐	☐	
Prüfaspekt c: Nutzungsbeschränkungen der Operationen					
13	Ist beschrieben, unter welchen Umständen die Operation aufgerufen werden darf?	☐	☐	☐	
Prüfbereich 3: Lokale Datentypen					
14	Sind die Bedeutungen der von der Schnittstelle / Klasse genutzten Datentypen beschrieben?	☐	☐	☐	

Nr.	Prüffrage	Ja	Teilweise	Nein	Bemerkungen
15	Ist beschrieben, wie Variablen und Konstanten der Datentypen deklariert werden?	☐	☐	☐	
16	Ist beschrieben, wie Literale der Datentypen erzeugt werden?	☐	☐	☐	
17	Sind die verfügbaren Operationen und Vergleichsoperatoren der Datentypen beschrieben?	☐	☐	☐	
18	Sind die möglichen Konversionen von Werten der Datentypen in andere Datentypen beschrieben?	☐	☐	☐	
Prüfbereich 4: Fehlerbehandlung					
19	Sind die zu erwartenden Fehlermeldungen mitsamt der jeweils ursächlichen Umstände beschrieben?	☐	☐	☐	
Prüfbereich 5: Konfigurationsoptionen					
20	Sind die möglichen Konfigurationsoptionen mitsamt zulässiger Werte und Bedeutung beschrieben?	☐	☐	☐	

Prüfbereich 6: Nicht-funktionale Zusicherungen					
21	Sind die nicht-funktionalen Zusicherungen beschrieben (z.B. bzgl. Performance oder Zuverlässigkeit)?	☐	☐	☐	
Prüfbereich 7: Abhängigkeiten					
21	Sind die von der Schnittstelle benötigten anderen Schnittstellen oder Ressourcen beschrieben?	☐	☐	☐	
Prüfbereich 8: Entwurf der Schnittstelle / Klasse					
22	Ist der Entwurf der Schnittstelle begründet worden (inkl. begründeter Ablehnung von alternativen Entwürfen)?	☐	☐	☐	
23	Sind bereits bekannte zukünftig anstehende Änderungen der Schnittstelle beschrieben worden?	☐	☐	☐	
Prüfbereich 9: Leitfaden zur Nutzung der Schnittstelle					
24	Ist beschrieben, wie diese Programmierer die Schnittstelle benutzen sollten?	☐	☐	☐	

Anhang B

Kategorisierungen innerhalb des seekda-Korpus

Kategorie	**Anzahl Bezeichner:**
Ergebnisse des 1. Durchlaufes	
Analysierte Bezeichner	186.580
Nicht klassifizierte Bezeichner	59.038
Falsch klassifizierte Bezeichner	510
Korrekt ignorierte Bezeichner	15.544
Bezeichner mit Verb	111.488
Ergebnisse des 2. Durchlaufes	
Analysierte Bezeichner	59.038
Nicht klassifizierte Bezeichner	13.180
Falsch klassifizierte Bezeichner	7.158
Korrekt ignorierte Bezeichner	0
Bezeichner mit Verb	38.700
Gesamtergebnis	
Analysierte Bezeichner	186.580
Nicht klassifizierte Bezeichner	13.180
Falsch klassifizierte Bezeichner	7.668
Korrekt ignorierte Bezeichner	15.544
Bezeichner mit Verb	150.188

Tabelle B.1: Klassifikationen der Bezeichner des seekda-Korpus

Anhang C

Thematische Raster für API-Verträge

Dieses Kapitel enthält den auf Basis des seekda-Korpus ermittelten Katalog thematischer Rollen und Raster. Zuerst erfolgt eine Auflistung der thematischen Rollen und deren zugehöriger Beschreibung. Es schließt mit einer detaillierten Darstellung der thematischen Raster.

C.1 Katalogisierte thematische Rollen

Die Annotierung der zufällig ausgewählten 287 Operationen des seekda-Korpus ergab die folgenden thematischen Rollen:

01.	DESTINATION FORMAT	Eine Struktur in die ein OBJECT durch die Handlung überführt wird (neu)
02.	SOURCE FORMAT	Die Struktur über die ein OBJECT vor der Handlung verfügt (neu)
03.	ACTION	Die zusammenfassende Beschreibung der Handlung (Girardi und Ibrahim, Gelhausen)
04.	AGENT	Der Handelnde (Girardi und Ibrahim, Gelhausen)
05.	COMPARISON	Ein Kriterium anhand dessen ein OBJECT identifiziert werden kann (Girardi und Ibrahim, Gelhausen)
06.	PURPOSE	Ein Zweck (oder ein Vorteil) (Girardi und Ibrahim, Gelhausen)

07.	INSTRUMENT	Ein Hilfsmittel bei einer Tätigkeit (Girardi und Ibrahim, Gelhausen)
08.	DESTINATION	Der Ort zu dem etwas hin geht oder bis zu dem es reicht (Girardi und Ibrahim, Gelhausen)
09.	SOURCE	Der Ort von dem etwas herkommt oder an dem es anfängt (Girardi und Ibrahim, Gelhausen)
10.	MANNER	Die Art wie jemand etwas tut (Girardi und Ibrahim, Gelhausen)
11.	ATTRIBUTE	Eine Eigenschaft eines OBJECT (Gelhausen)
12.	OBJECT	Dasjenige auf das eine Handlung einwirkt (Girardi und Ibrahim, Gelhausen)
13.	OWNER	Der Eigentümer oder Besitzer einer Sache, der Halter, der "Haber" (Gelhausen)
14.	STATUS	Ein Zustand (Girardi und Ibrahim, Gelhausen)
15.	CONDITION	Eine Bedingung, eine Voraussetzung (Girardi und Ibrahim, Gelhausen)
16.	DURATION	Eine Zeitspanne, ein Zeitraum (Girardi und Ibrahim, Gelhausen)
17.	TIME	Ein Zeitpunkt (Girardi und Ibrahim, Gelhausen)
18.	ALGORITHM	Die Liste der während der Handlung auszuführenden Instruktionen (neu)
19.	FORMULA	Eine FORMULA nach der etwas berechnet wird (neu)
20.	OPERAND	Ein Argument einer mathematischen Berechnung (neu)
21.	CONFIGURATION	Eine bestimmte Justierung globaler Parameter mit der die Handlung vollzogen wird (z.B. über eine Konfigurationsdatei) (neu)
22.	PLATFORM	Die Grundlage (Hardware oder Software) auf deren Basis die Handlung vollzogen wird(neu)
23.	FORMAT	Eine Struktur über die ein OBJECT verfügt (neu)

24.	LIMIT	Eine Beschränkung (z.B. die Anzahl der zurückgelieferten Datensätze) (neu)
25.	ORDERING	Eine Sortierung (neu)
26.	REPORT	Ein detaillierter Bericht mit Angaben zur Handlung (neu)
27.	RULE	Eine Regel nach der gehandelt wird (neu)
28.	TIME TO LIVE	Ein vorgegebener Zeitpunkt bis zu dem die Handlung abgeschlossen sein muss (neu)
29.	PRIORITY	Die Priorität der Handlung gegenüber anderen Handlungen (neu)
30.	PROVIDER	Jemand, der eine Handlung anbietet (neu)
31.	NAME	Ein nicht-numerischer Bezeichner (neu)
32.	PASSWORD	Ein Passwort (neu)
33.	PRIMARY KEY	Ein eindeutiges Merkmal eines OBJECT (neu)
34.	USERNAME	Ein Benutzername (neu)
35.	CHECKSUM	Eine Prüfsumme (neu)
36.	DEPENDENCY	Eine Abhängigkeit (neu)
37.	IGNORABLE	Ein Signaturelement das während der Ausführung der ACTION nicht beachtet wird (neu)
38.	MESSAGE	Eine Rückmeldung zur Handlung in natürlicher Sprache (neu)
39.	METADATA	Eine Angabe zur Handlung, die nicht direkt erfragt worden ist (neu)
40.	SESSION	Eine Sitzung (neu)
41.	TRANSACTION	Die nicht unterbrechbare Verkettung (auch mehrerer) ACTION (neu)
42.	LANGUAGE	Eine Sprache, in der die Handlung beschrieben wird (neu)
43.	ACCESS	Auf welche Weise auf ein OBJECT zugegriffen werden kann (neu)
44.	ACCESS KEY	Eine Kennung über ein OBJECT zugegriffen werden kann (neu)

Während der Definition der thematischen Raster dieses Katalogs sind die folgenden thematischen Rollen ergänzt worden:

45.	DESTINATION OBJECT	Ein spezielles OBJECT an einem Zielort (neu)
46.	SOURCE OBJECT	Ein spezielles OBJECT an einem Ursprungsort (neu)
47.	DESTINATION PRIMARY KEY	Identifiziert ein OBJECT an einem Zielort eindeutig (neu)
48.	SOURCE PRIMARY KEY	Identifiziert ein OBJECT an einem Ursprungsort eindeutig (neu)
49.	DESTINATION COMPARISON	Identifiziert mehrere OBJECTs an einem Zielort (neu)
50.	SOURCE COMPARISON	Identifiziert mehrere OBJECTs an einem Ursprungsort (neu)

Wie in Abschnitt 4.3.3 dargelegt worden ist, kann die Belegung einer thematische Rolle während des Betriebs der Operation bzw. Methode nicht verfügbar sein. Als Beispiel wäre hier eine Datenbank zu nennen, welche beispielsweise die Rolle SOURCE belegt. Der Ansatz thematischer Raster unterstützt ebenfalls die Ableitung und Beschreibung des Verhaltens der Operation in einem solchen Fall. Zu diesem Zweck wird jeweils der Name der betreffenden thematischen Rolle mit dem Präfix ERROR versehen, z.B. ERROR SOURCE. Die Belegung dieser neuen thematischen Rolle beschreibt dann die entsprechende Ausgabe der Operation, z.B. eine Ausnahme namens `NoDatabaseConnectionException`. Die Liste der auf diese Weise erzeugten Rollen ist im Folgenden angegeben:

51.	ERROR DESTINATION FORMAT	Belegung des DESTINATION FORMAT ist nicht verfügbar
52.	ERROR SOURCE FORMAT	Belegung des SOURCE FORMAT ist nicht verfügbar
53.	ERROR ACTION	Belegung der ACTION ist nicht verfügbar
54.	ERROR AGENT	Belegung des AGENT ist nicht verfügbar
55.	ERROR COMPARISON	Belegung des COMPARISON ist nicht verfügbar
56.	ERROR PURPOSE	Belegung des PURPOSE ist nicht verfügbar
57.	ERROR INSTRUMENT	Belegung des INSTRUMENT ist nicht verfügbar
58.	ERROR DESTINATION	Belegung der DESTINATION ist nicht verfügbar
59.	ERROR SOURCE	Belegung der SOURCE ist nicht verfügbar
60.	ERROR MANNER	Belegung der MANNER ist nicht verfügbar
61.	ERROR ATTRIBUTE	Belegung des ATTRIBUTE ist nicht verfügbar
62.	ERROR OBJECT	Belegung des OBJECT ist nicht verfügbar
63.	ERROR OWNER	Belegung des OWNER ist nicht verfügbar
64.	ERROR STATUS	Belegung des STATUS ist nicht verfügbar
65.	ERROR CONDITION	Belegung der CONDITION ist nicht verfügbar
66.	ERROR DURATION	Belegung der DURATION ist nicht verfügbar
67.	ERROR TIME	Belegung der TIME ist nicht verfügbar
68.	ERROR ALGORITHM	Belegung der ALGORITHM ist nicht verfügbar
69.	ERROR FORMULA	Belegung der FORMULA ist nicht verfügbar
70.	ERROR OPERAND	Belegung des OPERAND ist nicht verfügbar
71.	ERROR CONFIGURATION	Belegung der CONFIGURATION ist nicht verfügbar
72.	ERROR PLATFORM	Belegung der PLATFORM ist nicht verfügbar
73.	ERROR FORMAT	Belegung des FORMAT ist nicht verfügbar

74.	ERROR LIMIT	Belegung des LIMIT ist nicht verfügbar
75.	ERROR ORDERING	Belegung des ORDERING ist nicht verfügbar
76.	ERROR REPORT	Belegung des REPORT ist nicht verfügbar
77.	ERROR RULE	Belegung der RULE ist nicht verfügbar
78.	ERROR TIME TO LIVE	Belegung der TIME TO LIVE ist nicht verfügbar
79.	ERROR PRIORITY	Belegung der PRIORITY ist nicht verfügbar
80.	ERROR PROVIDER	Belegung des PROVIDER ist nicht verfügbar
81.	ERROR NAME	Belegung des NAME ist nicht verfügbar
82.	ERROR PASSWORD	Belegung des PASSWORD ist nicht verfügbar
83.	ERROR PRIMARY KEY	Belegung des PRIMARY KEY ist nicht verfügbar
84.	ERROR USERNAME	Belegung des USERNAME ist nicht verfügbar
85.	ERROR CHECKSUM	Belegung der CHECKSUM ist nicht verfügbar
86.	ERROR DEPENDENCY	Belegung der DEPENDENCY ist nicht verfügbar
87.	ERROR IGNORABLE	Belegung des IGNORABLE ist nicht verfügbar
88.	ERROR MESSAGE	Belegung der MESSAGE ist nicht verfügbar
89.	ERROR METADATA	Belegung des METADATA ist nicht verfügbar
90.	ERROR SESSION	Belegung der SESSION ist nicht verfügbar
91.	ERROR TRANSACTION	Belegung der TRANSACTION ist nicht verfügbar
92.	ERROR LANGUAGE	Belegung der LANGUAGE ist nicht verfügbar
93.	ERROR ACCESS	Belegung des ACCESS ist nicht verfügbar
94.	ERROR ACCESS KEY	Belegung des ACCESS KEY ist nicht verfügbar

95.	ERROR DESTINATION OBJECT	Belegung des DESTINATION OBJECT ist nicht verfügbar
96.	ERROR SOURCE OBJECT	Belegung des SOURCE OBJECT ist nicht verfügbar
97.	ERROR DESTINATION PRIMARY KEY	Belegung des DESTINATION PRIMARY KEY ist nicht verfügbar
98.	ERROR SOURCE PRIMARY KEY	Belegung des SOURCE PRIMARY KEY ist nicht verfügbar
99.	ERROR DESTINATION COMPARISON	Belegung des DESTINATION COMPARISON ist nicht verfügbar
100.	ERROR SOURCE COMPARISON	Belegung des SOURCE COMPARISON ist nicht verfügbar

C.2 Katalogisierte thematische Raster

C.2.1 Prüfende Operation

Beschreibung: Eine prüfende Operation prüft, ob ein oder mehrere OBJECTs einen Test bestehen. Dieser Test wird von mind. einem ALGORITHM oder mind. einer RULE beschrieben. Die geprüften OBJECTs stammen aus einer SOURCE. Alle getesteten OBJECTs werden aus der SOURCE über einen PRIMARY KEY oder mind. ein COMPARISON ausgewählt. Das Ergebnis des Tests ist ein REPORT. Dieses Raster basiert auf den Verbnet-Klassen sight-30.2, peer-30.3 und investigate-35.4.
Verben: attend, canvass, check, contains, descry, discover, eavesdrop, espy, examine, experience, explore, eye, frisk, gape, gawk, gaze, glance, glare, glimpse, goggle, has, inspect, investigate, is, leer, listen, look, make out, monitor, note, observe, ogle, overhear, peek, peep, peer, perceive, peruse, quiz, raid, ransack, recognize, regard, riffle, savor, scan, scent, scrutinize, sight, sniff, snoop, spot, spy, squint, stare, study, survey, tap, test, validate, view, watch, witness
Verwandte Bezeichnungen: Validator

Thematische Rollen:

Rolle	**Beschreibung**	**Pflicht**
ALGORITHM	Definiert den Test.	J
PRIMARY KEY	Identifiziert ein zu prüfendes OBJECT in der SOURCE eindeutig.	N
REPORT	Enthält das Testergebnis.	J
COMPARISON	Identifiziert ein zu prüfendes OBJECT in der SOURCE.	N
ACTION	Beschreibt die Operation.	N
OBJECT	Wird dem Test unterzogen.	J
SOURCE	Enthält die zu prüfenden OBJECT	N
RULE	Definiert den Test.	J

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
ALGORITHM	! existiert (“RULE”)
PRIMARY KEY	! existiert (“COMPARISON”)
REPORT	immer
COMPARISON	! existiert (“PRIMARY KEY”)
ACTION	immer
OBJECT	immer
SOURCE	immer
RULE	! existiert (“ALGORITHM”)

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Platform, Standard Edition 7
Programm:	javax.xml.validation.Validator#validate
Version:	1.7
Quelle:	http://docs.oracle.com/javase/7/docs/api/ (Letzter Abruf: 08.06.2013)
Beschreibung:	Standard-Klassenbibliothek für Java

Rolle:	**Belegung:**
ALGORITHM	Prüft ob ein XML-Dokument konform zu einem vorgegebenen XML-Schema (Grammatik) ist.
PRIMARY KEY	Wird nicht benötigt
REPORT	Das Dokument entspricht dem Schema, wenn die Methode ohne Ausnahme terminiert. Andernfalls wird eine `SAXException` geworfen.
COMPARISON	Die Referenzgrammatik ist abhängig von der Implementierung des Validators.
ACTION	Konformitätsprüfung
OBJECT	Das XML-Dokument
SOURCE	Das Dokument wird per Parameter `source` übergeben.
ERROR OBJECT	Sofern das XML-Dokument nicht gelesen werden kann, wird eine `IOException` geworfen.
ERROR SOURCE	Wenn der Parameter `source null` ist, wird eine `NullPointerException` geworfen.

Die Angabe der thematischen Rollen ERROR ACTION, ERROR COMPARISON, ERROR REPORT, ERROR PRIMARY KEY und ERROR ALGORITHM ist ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.2 Schlussfolgernde Operation

Beschreibung: Eine schlussfolgernde Operation leitet ein OBJECT oder einen REPORT aus einer Menge von ATTRIBUTEs ab. Die Ableitung des OBJECT oder REPORT wird durch eine RULE oder einen ALGORITHM beschrieben. Sobald sich die Schlussfolgerung nach einer mathematischen FORMULA richtet, handelt es sich bei der betrachteten Operation um eine berechnende Operation.
Verben: conclude, derive
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
ALGORITHM	Definiert die Vorgehensweise bei der Schlussfolgerung.	J
OBJECT	Repräsentiert die Schlussfolgerung (z.B. in Form eines Wertes).	J
REPORT	Repräsentiert die Schlussfolgerung (z.B. in Form einer Menge von Werten).	J
ATTRIBUTE	Wert, aus dem das OBJECT oder der REPORT abgeleitet wird.	J
ACTION	Beschreibt die Operation.	N
RULE	Definiert die Vorgehensweise bei der Schlussfolgerung.	J

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
REPORT	! existiert (“OBJECT”)
OBJECT	! existiert (“REPORT”)
ALGORITHM	! existiert (“RULE”)
RULE	! existiert (“ALGORITHM”)
ATTRIBUTE	immer
ACTION	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Header-Datei hukdf.h (Key Derivation Functions (KDFs))
Programm:	Operation: hu_KDFDerive()
Version:	Native SDK for BlackBarry 10
Quelle:	http://developer.blackberry.com/native/reference/core /crypto_libref/topic/hu_kdfderive.html (Letzter Abruf: 31.08.2013)
Beschreibung:	API-Schnittstelle für BlackBerry 10-Smartphones

Rolle:	**Belegung:**
ALGORITHM	Nutzt die per Parameter `algid` spezifizierte Hash-Funktion aus der Menge der unterstützten Algorithmen.
REPORT	Das Resultat.
OBJECT	Der Schlüssel zum Ver- und Entschlüsseln (wird in den Speicher des Parameters `keyValue` geschrieben).
ACTION	Erzeugt einen Schlüssel zur Ver- und Entschlüsselung des geheimen, per Parameter `sharedSecret` übergebenen Datums.
RULE	Wird nicht benötigt.
ATTRIBUTE	Der Parameter `sharedSecret`.
ERROR ALGORITHM	Das Resultat ist `SB_ERR_KDF_BAD_ALGORITHM` (ALGORITHM ist unbekannt).
ERROR ATTRIBUTE	Das Resultat ist `SB_ERR_NULL_INPUT_BUF` (ATTRIBUTE ist `null`) oder `SB_ERR_BAD_INPUT_BUF_LEN` (ATTRIBUTE hat ungültige Länge).
ERROR OBJECT	Das Resultat ist `SB_ERR_NULL_OUTPUT_BUF` (Speicher des OBJECT ist `null`) oder `SB_ERR_BAD_OUTPUT_BUF_LEN` (Speicher des OBJECT hat ungültige Länge).

Die Angabe der thematischen Rollen ERROR ACTION, ERROR REPORT und ERROR RULE ist ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.3 Konvertierende Operation

Beschreibung: Eine konvertierende Operation wandelt ein OBJECT aus einem SOURCE FORMAT in ein DESTINATION FORMAT um. Für diese Umwandlung wird ein INSTRUMENT (externe Ressource oder fremdes Programm) verwendet. Alternativ kann die Umwandlung durch die Operation selbst nach Vorgabe eines ALGORITHM oder einer RULE erfolgen. Dieses Raster basiert auf den VerbNet-Klassen convert-26.6.2 und Turn-26.6.1.
Verben: alter, change, change over, come around, convert, deform, fall, get, get down, go back, metamorphose, move over, mutate, resort, return, revert, settle down, shift, switch, switch over, take, transform, translate, transmute, turn, get around
Verwandte Bezeichnungen: Converter

Thematische Rollen:

Rolle	Beschreibung	Pflicht
ALGORITHM	Definiert die Vorgehensweise bei der Umwandlung.	N
OBJECT	Wird umgewandelt.	J
SOURCE FORMAT	Struktur, über die das OBJECT vor der Umwandlung verfügt.	J
DESTINATION FORMAT	Struktur, in die das OBJECT überführt werden soll.	J
ACTION	Beschreibt die Operation.	N
RULE	Definiert die Vorgehensweise bei der Umwandlung.	N
INSTRUMENT	Externe Ressource oder fremdes Programm, das die Umwandlung durchführt.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
RULE	! existiert (“ALGORITHM”) && ! existiert (“INSTRUMENT”)
ALGORITHM	! existiert (“RULE”) && ! existiert (“INSTRUMENT”)
INSTRUMENT	! existiert (“ALGORITHM”) && ! existiert (“RULE”)
DESTINATION FORMAT	immer
SOURCE FORMAT	immer
OBJECT	immer
ACTION	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Server Faces
Programm:	javax.faces.convert.DateTimeConverter# getAsObject
Version:	2.0
Quelle:	http://docs.oracle.com/javaee/7/api/javax/faces/convert/DateTimeConverter.html (Letzter Abruf: 31.08.2013)
Beschreibung:	Rahmenwerk für Web-Anwendungen in Java
Rolle:	**Belegung:**
RULE	Wird nicht benötigt.
ALGORITHM	Siehe kopierten Vertragsausschnitt in Abbildung C.1.
INSTRUMENT	Das gesetzte Attribut *locale* oder das Attribut *locale* der übergebenen Instanz von UIViewRoot.
DESTINATION FORMAT	java.util.Date.
SOURCE FORMAT	java.lang.String. Das spezifizierte Datumsformat muss den für java.text.SimpleDateFormat spezifizierten Regeln genügen.
OBJECT	Der Parameter `value`.
ACTION	Die Konversion eines Strings in ein Datumsobjekt.
ERROR OBJECT	Sobald der Parameter `value` `null` ist, liefert die Operation auch `null` zurück.

Die Angabe der thematischen Rollen ERROR RULE, ERROR ALGORITHM, ERROR

The `getAsObject()` method parses a String into a `java.util.Date`, according to the following algorithm:

- If the specified String is null, return a `null`. Otherwise, trim leading and trailing whitespace before proceeding.
- If the specified String - after trimming - has a zero length, return `null`.
- If the `locale` property is not null, use that `Locale` for managing parsing. Otherwise, use the `Locale` from the `UIViewRoot`.
- If a `pattern` has been specified, its syntax must conform the rules specified by `java.text.SimpleDateFormat`. Such a pattern will be used to parse, and the `type`, `dateStyle`, and `timeStyle` properties will be ignored.
- If a `pattern` has not been specified, parsing will be based on the `type` property, which expects a date value, a time value, or both. Any date and time values included will be parsed in accordance to the styles specified by `dateStyle` and `timeStyle`, respectively.
- If a `timezone` has been specified, it must be passed to the underlying `DateFormat` instance. Otherwise the "GMT" timezone is used.
- In all cases, parsing must be non-lenient; the given string must strictly adhere to the parsing format.

Abbildung C.1: ALGORITHM der Operation getAsObject()

INSTRUMENT, ERROR DESTINATION FORMAT, ERROR SOURCE FORMAT und ERROR ACTION entfällt, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.4 Erzeugende Operation

Beschreibung: Eine erzeugende Operation erzeugt ein neues OBJECT und initialisiert ggf. einzelne seiner ATTRIBUTEs. Das neu erzeugte OBJECT ist über einen PRIMARY KEY oder seinen NAME identifizierbar. Es existiert an einer DESTINATION und gehört einem OWNER. Im Unterschied zu einer deponierenden Operation wird in diesem Fall ein neues OBJECT erstellt. Eine deponierende Operation hingegen legt ein bereits existierendes OBJECT an einer DESTINATION ab. Dieses Raster basiert auf den VerbNet-Klassen create-26.4 und engender-27.
Verben: beget, bring about, cause, coin, compose, concoct, conjure, construct, contrive, create, derive, design, dig, engender, fabricate, form, formulate, generate, improvise, incite, instigate, invent, kindle, lay, manufacture, mint, model, organize, produce, provoke, publish, rearrange, rebuild, reconstitute, recreate, reorganize, schedule, set off, shape, sire, spark, spawn, stage, store, style, synthesize
Verwandte Bezeichnungen: Konstruktor, Fabrikmethode

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird erzeugt.	J
ATTRIBUTE	Der Wert, der zur Erzeugung des OBJECT genutzt wird (z.B. ein Attribut des OBJECT).	N
PRIMARY KEY	Die eindeutige Kennung des OBJECT.	N
NAME	Der alphanumerischer Bezeichner, über den das OBJECT identifiziert werden kann.	N
ACTION	Beschreibt die Operation.	N
DESTINATION	Der Ort, an dem das OBJECT nach seiner Erzeugung abgelegt wird.	J
OWNER	Die Entität, die dem OBJECT übergeordnet ist.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
DESTINATION	immer
OBJECT	immer
PRIMARY KEY	!existiert ("NAME")
ACTION	immer
NAME	!existiert ("PRIMARY KEY")
OWNER	immer
ATTRIBUTE	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	JFreeChart
Programm:	org.jfree.chart.ChartFactory# createBarChar
Version:	1.0.14
Quelle:	http://www.jfree.org/jfreechart/api/javadoc/
	(Letzter Abruf: 09.06.2013)
Beschreibung:	Java-Bibliothek zur Erzeugung von Diagrammen

Rolle:	**Belegung:**
DESTINATION	Wird nicht benötigt.
OBJECT	Säulendiagramm-Objekt.
PRIMARY KEY	Wird nicht benötigt.
ACTION	Erzeugung eines Säulendiagramms.
NAME	Der Wert des Parameters `title`.
OWNER	Wird nicht benötigt.
ATTRIBUTE	Alle Parameter.

Die Angabe der thematischen Rollen ERROR DESTINATION, ERROR OBJECT, ERROR PRIMARY KEY, ERROR ACTION, ERROR NAME und ERROR OWNER entfällt, weil diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können. Für den Parameter `orientation` ist die Belegung mit `null` nicht erlaubt. Die thematische Rolle ERROR ATTRIBUTE könnte mit dem Resultat bzw. der Ausnahme belegt werden, das bzw. die in diesem Fall erfolgt. Sie ist in diesem Beispiel trotzdem ausgelassen worden, weil der Vertrag der Methode bezüglich des Verhaltens der Methode für diesen Fall schweigt.

C.2.5 Beschreibende Operation

Beschreibung: Eine beschreibende Operation charakterisiert ein oder mehrere OBJECTs auf bestimmte Weise. Diese Charakterisierung kann aufgrund eines oder mehrerer verschiedener ATTRIBUTEs des OBJECT erfolgen. Die charakterisierten OBJECTs werden anhand eines COMPARISON oder PRIMARY KEY ausgewählt. Das Ergebnis einer beschreibenden Operation ist ein REPORT. Sofern das OBJECT aus einer bekannten SOURCE stammt, kann sein dortiges FORMAT beschrieben werden. Dieses Raster basiert auf der VerbNet-Klasse characterize-29.2.
Verben: accept, adopt, bill, cast, certify, characterize, class, classify, conceive, count, define, depict, describe, detail, diagnose, envisage, envision, hail, identify, imagine, interpret, judge, know, paint, peg, perceive, picture, pigeonhole, portray, praise, rank, recast, recollect, redraw, regard, remember, report, represent, reveal, see, select, specify, stereotype, take, treat, typecast, view, visualize
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird beschrieben.	J
ATTRIBUTE	Zur Beschreibung ausgewertete Eigenschaft des OBJECT.	J
COMPARISON	Identifiziert das zu beschreibende OBJECT in der SOURCE.	N
PRIMARY KEY	Identifiziert das zu beschreibende OBJECT in der SOURCE eindeutig.	N
ACTION	Beschreibt die Operation.	N
REPORT	Ist das Resultat der Beschreibung.	J
SOURCE	Der Ort, von dem das OBJECT stammt.	N
FORMAT	Die Struktur des OBJECT in der SOURCE.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
OBJECT	immer
ATTRIBUTE	hatAttribute (“OBJECT”)
COMPARISON	istPlural (“OBJECT”) && ! existiert (“PRIMARY KEY”)
PRIMARY KEY	istPlural (“OBJECT”) && ! existiert (“COMPARISON”)
ACTION	immer
REPORT	immer
SOURCE	immer
FORMAT	existiert (“SOURCE”)

Beispiel:

Referenz:	**Belegung:**
Komponente:	MAchine Learning for LanguagE Toolkit (Mallet)
Programm:	cc.mallet.classify.NaiveBayes# classify
Version:	2.0.7
Quelle:	http://mallet.cs.umass.edu/api/ (Letzter Abruf: 09.06.2013)
Beschreibung:	Java-Bibliothek zur Klassifikation Texten
Rolle:	**Belegung:**
OBJECT	Der Parameter `instance`.
ATTRIBUTE	Die Komponenten des `FeatureVector`.
COMPARISON	Wird nicht benötigt.
PRIMARY KEY	Wird nicht benötigt.
ACTION	Die Klassifikation des übergebenen Vektors in eine Kategeorie nach dem naiven Bayes-Klassifikator.
REPORT	Die Kategorie, in die der Vektor klassifiziert worden ist.
SOURCE	Wird nicht benötigt.
FORMAT	Der Parameter `instance` (muss ein `FeatureVector` sein).

Die Angaben zu den thematischen Rollen ERROR OBJECT, ERROR ATTRIBUTE, ERROR COMPARISON, ERROR PRIMARY KEY, ERROR ACTION, ERROR REPORT, ERROR SOURCE und ERROR FORMAT sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.6 Duplizierende Operation

Beschreibung: Eine duplizierende Operation erstellt eine Kopie eines OBJECT. Diese Kopie ist dem Original gleich. Die Kopie kann von einer SOURCE gelesen und an einer DESTINATION abgelegt werden. Dort liegt sie ggf. jeweils in einem bestimmten SOURCE FORMAT bzw. DESTINATION FORMAT vor. Dieses Raster basiert auf der VerbNet-Klasse transcribe-25.4.
Verben: chronicle, clone, copy, document, duplicate, film, forge, imitate, microfilm, photocopy, photograph, record, tally, tally up, tape, televise, transcribe, type
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird kopiert.	J
SOURCE	Der Ort, an dem das Original liegt.	N
DESTINATION	Der Ort, an dem die Kopie liegt.	N
ACTION	Beschreibt die Operation.	N
DESTINATION FORMAT	Die Struktur der Kopie an der DESTINATION.	N
SOURCE FORMAT	Die Struktur der Kopie in der SOURCE.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
OBJECT	immer
SOURCE	immer
DESTINATION	immer
SOURCE FORMAT	existiert (“SOURCE”)
DESTINATION FORMAT	existiert (“DESTINATION”)
ACTION	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Commons IO
Programm:	org.apache.commons.io.FileUtils # copyFile
Version:	2.4
Quelle:	http://commons.apache.org/proper/commons-io/javadocs/api-2.4/ (Letzter Abruf: 09.06.2013)
Beschreibung:	Java-Bibliothek zur Verarbeitung von Dateien
Rolle:	**Belegung:**
OBJECT	Der Inhalt der zu kopierenden Datei.
SOURCE	Der Parameter `srcFile`.
DESTINATION	Der Parameter `destFile`.
SOURCE FORMAT	Binär.
DESTINATION FORMAT	Binär.
ACTION	Kopiert die Inhalte der Quell- in die Zieldatei. Sofern die Zieldatei bereits existiert, wird sie überschrieben.
ERROR SOURCE	Die `NullPointerException`, wenn der Parameter `srcFile` `null` bzw. `IOException`, wenn die Datei `srcFile` ungültig ist.
ERROR DESTINATION	Die `NullPointerException`, wenn der Parameter `destFile` `null` bzw. `IOException`, wenn die Datei `srcFile` ungültig ist.

Die Angaben der thematischen Rollen ERROR OBJECT, ERROR ACTION, ERROR SOURCE FORMAT und ERROR DESTINATION FORMAT sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.7 Verbindende Operation

Beschreibung: Etabliert eine Verbindung von einer SOURCE zu einer DESTINATION. Die Verbindung wird von mind. zwei PARTICIPANTs genutzt. Die Kommunikation der PARTICIPANTs über die Verbindung geschieht in einem bestimmten FORMAT (Protokoll). Die PARTICIPANTs können über ein COMPARISON oder einen PRIMARY KEY adressiert werden. Die Verbindung wird nach Ablauf einer TIME TO LIVE geschlossen. Dieses Raster basiert auf der VerbNet-Klasse establish-55.5.
Verben: arrange, bring about, connect, constitute, devise, establish, fake, feign, format, found, implement, initiate, innovate, instigate, institute, introduce, launch, machinate, mount, open, open up, organize, originate, pioneer, plant, prepare, simulate, stage, strike up, synthesise
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
SOURCE	Der Ausgangspunkt des Verbindungsaufbaus.	N
DESTINATION	Die Gegenstelle der SOURCE beim Verbindungsaufbau.	J
ACTION	Beschreibt die Operation.	N
PARTICIPANT	Die Organisation, Person, System oder Ressource, mit welchem über die Verbindung kommuniziert wird.	N
FORMAT	Definiert das Kommunikationsprotokoll und ggf. Erweiterungen desselben.	N
COMPARISON	Identifiziert einen oder mehrere PARTICIPANTs anhand einer oder mehrerer Eigenschaften.	N
PRIMARY KEY	Identifiziert einen oder mehrere PARTICIPANTs eindeutig.	N
TIME TO LIVE	Definiert das maximale Alter eines Verbindungsversuches oder einer aufgebauten Verbindung.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
SOURCE	immer
DESTINATION	immer
ACTION	immer
PARTICIPANT	existiert (“DESTINATION”)
FORMAT	immer
COMPARISON	existiert (“PARTICIPANT”;) && ! existiert (“NAME”) && ! existiert (“PRIMARY KEY”)
PRIMARY KEY	existiert (“PARTICIPANT”;) && ! existiert (“COMPARISON”) && ! existiert (“NAME”)
TIME TO LIVE	immer

Beispiel:

Referenz:	**Belegung:**
Komponente: Programm: Version: Quelle: Beschreibung:	The Python Standard Library socket # createConnection 2.7 http://docs.python.org/2/library/ (Letzter Abruf: 09.06.2013) Python-Standardklassenbibliothek
Rolle:	**Belegung:**
SOURCE	Der optionale Parameter `source_address` (ein Tupel aus Absender-IP-Adresse und -Port).
DESTINATION	Der Parameter `address` (ein Tupel aus Absender-IP-Adresse und -Port)
ACTION	Baut eine IPv4- bzw. IPv6-Verbindung zum angegeben Ziel auf.
PARTICIPANT	Die Belegungen der Parameter `source_address` und `address`.
FORMAT	Transmission Control Protocol (TCP).
COMPARISON	Wird nicht benötigt.
PRIMARY KEY	Die Belegungen der Parameter `source_address` und `address`.
TIME TO LIVE	Der optionale Parameter `timeout` in Sekunden.

Die Angaben der thematischen Rollen ERROR SOURCE, ERROR DESTINATION, ERROR ACTION, ERROR PARTICIPANT, ERROR FORMAT, ERROR COMPARISON, ERROR PRIMARY KEY und ERROR TIME TO LIVE sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.8 Lesende Operation

Beschreibung: Eine lesende Operation liest ein oder mehrere OBJECTs aus einer SOURCE. Die Position der OBJECTs in der SOURCE ist vor Aufruf der Operation durch einen PRIMARY KEY (z.B. Adressen, Referenzen, Indices, etc.) bereits bestimmt. Dieses Raster basiert auf der VerbNet-Klasse get-13.5.1. Sollte die Position der OBJECT vor Aufruf der Operation unbekannt sein und müssten diese anhand bekannter COMPARISON bestimmt werden, so handelt es sich um eine suchende und keine lesende Operation.
Verben: attain, book, buy, call, cash, catch, charter, choose, conserve, earn, fetch, gain, gather, get, hire, lease, order, phone, pick, pluck, procure, pull, reach, read, rent, reserve, score, secure, shoot, slaughter, steal, vote, win
Verwandte Bezeichnungen: Akzessor, Getter

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird gelesen.	J
PRIMARY KEY	Identifiziert das OBJECT eindeutig in der SOURCE.	J
SOURCE	Der Speicher, aus dem dem das OBJECT gelesen wird.	J
ACTION	Beschreibt die Operation.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
OBJECT	immer
SOURCE	immer
PRIMARY KEY	immer
ACTION	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Commons IO
Programm:	org.apache.commons.io.FileUtils # readFileToString (File, String)
Version:	2.4
Quelle:	http://commons.apache.org/proper/commons-io/javadocs/api-2.4/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Java-Bibliothek zur Verarbeitung von Dateien
Rolle:	**Belegung:**
OBJECT	Die Datei, die über den Parameter `file` adressiert wird.
SOURCE	Das lokale Dateisystem.
ACTION	Öffnet die adressierte Datei, liest sie komplett in einen String und schließt die Datei danach.
PRIMARY KEY	Der Parameter `file` (Pfad zur Datei).

Die Angaben der thematischen Rollen ERROR OBJECT, ERROR SOURCE, ERROR ACTION und ERROR PRIMARY KEY sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.9 Protokollierende Operation

Beschreibung: Eine protokollierende Operation speichert eine MESSAGE an einer DESTINATION zum Zweck der späteren Revision eines Vorgangs. Die MESSAGE liegt in einem FORMAT vor und verfügt über eine PRIORITY.
Verben: log
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	**Beschreibung**	**Pflicht**
MESSAGE	Der erzeugte Protokolleintrag.	J
DESTINATION	Der Speicher, in dem die MESSAGE abgelegt wird.	J
ACTION	Beschreibt die Operation.	N
FORMAT	Der Aufbau der MESSAGE.	N
PRIORITY	Die Relevanz des Inhalts der MESSAGE.	J

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
MESSAGE	immer
DESTINATION	immer
ACTION	immer
FORMAT	immer
PRIORITY	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Platform, Standard Edition 7
Programm:	java.util.logging.Logger #log
Version:	1.7
Quelle:	http://docs.oracle.com/javase/7/docs/api/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Standard-Klassenbibliothek für Java
Rolle:	**Belegung:**
MESSAGE	Der Parameter `msg`.
DESTINATION	Die Log-Datei (abhängig von der Konfiguration in der Datei logging.properties).
ACTION	Schreibt die übergebene Nachricht mit der übergebenen Priorität in das konfigurierte Ziel.
FORMAT	Der Aufbau der MESSAGE (abhängig von der Konfiguration in der Datei logging.properties)
PRIORITY	Der Parameter `level`.

Die Angaben der thematischen Rollen ERROR MESSAGE, ERROR DESTINATION, ERROR ACTION und ERROR FORMAT, ERROR PRIORITY und ERROR CODE sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.10 Berechnende Operation

Beschreibung: Eine berechnende Operation ermittelt das OBJECT durch Anwendung einer mathematischen FORMULA. In die Berechnung geht mind. ein OPERAND ein. Sofern mehrere OBJECTs berechnet worden sind, sind diese gemäß eines ORDERING sortiert. Dieses Raster basiert auf der VerbNet-Klasse multiply-108. Sofern die Berechnung sich nach einer RULE oder einem ALGORITHM definiert, handelt es sich um eine schlussfolgernde Operation.
Verben: add, average, calculate, compute, count, deduct, divide, extrapolate, factor out, interpolate, multiply, subtract, sum, tally
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Die Bezeichnung des oder der berechneten Werte.	J
FORMULA	Die mathematische Vorschrift, nach der das OBJECT berechnet wird.	J
ACTION	Beschreibt die Operation.	N
OPERAND	Der gemäß der FORMULA verrechnete Wert.	J
ORDERING	Die Reihenfolge der OBJECTs (sofern mehrere OBJECTs berechnet worden sind).	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
OBJECT	immer
FORMULA	immer
ACTION	immer
OPERAND	immer
ORDERING	istPlural (“OBJECT”)

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Platform, Standard Edition 7
Programm:	java.math.BigDecimal #devide
Version:	1.7
Quelle:	http://docs.oracle.com/javase/7/docs/api/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Standard-Klassenbibliothek für Java
Rolle:	**Belegung:**
OBJECT	Der Wert des Quotienten.
FORMULA	`this` / Parameter divisor. Für Rundungen wird die durch den Parameter `roundingMode` spezifizierte Rundungsregel angewendet.
ACTION	Die Berechnung des Quotienten.
OPERAND	`this` und Parameter divisor.
ORDERING	Wird nicht benötigt.

Die Angaben der thematischen Rollen ERROR FORMULA, ERROR ACTION, ERROR OBJECT, ERROR OPERAND und ERROR ORDERING sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.11 Zusammenführende Operation

Beschreibung: Eine zusammenführende Operation kombiniert mind. zwei OBJECTs oder einzelne ihrer ATTRIBUTEs. Die OBJECTs stammen aus einer SOURCE. Die Kombination wird durch eine RULE oder einen ALGORITHM beschrieben. Die Kombination kann an einer DESTINATION abgelegt werden. Sofern die verarbeiteten OBJECTs in der SOURCE eingegrenzt werden können, geschieht dies über die Angabe eines LIMITS, eines COMPARISON oder eines PRIMARY KEY. Dieses Raster basiert auf der VerbNet-Klasse mix-22.1.
Verben: add, admix, amalgamate, blend, coalesce, combine, commingle, compound, connect, consolidate, cream, fuse, intermix, join, link, meld, merge, mingle, mix, network, pair, pool, scramble, tie, tumble, unite
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	**Beschreibung**	**Pflicht**
OBJECT	Die Dinge, die zusammengeführt werden.	J
ATTRIBUT	Ein bestimmter Teil der OBJECTs, der zusammengeführt wird.	N
SOURCE	Der Ort, an dem sich die OBJECTs vor ihrer Zusammenführung befinden.	J
RULE	Beschreibt die Zusammenführung der OBJECTs.	J
ALGORITHM	Mehrere RULEs, nach denen die OBJECTs zusammengeführt werden.	J
ACTION	Beschreibt die Operation.	N
DESTINATION	Der Ort, an dem die zusammengeführten OBJECTs abgelegt werden.	N
LIMIT	Definiert eine Minimum- oder Maximumanzahl der betroffenen OBJECTs.	N
COMPARISON	Die Eigenschaft, die die zusammenzuführenden OBJECTs in der SOURCE kennzeichnet.	N
PRIMARY KEY	Die Eigenschaft, die die zusammenzuführenden OBJECTs in der SOURCE eindeutig kennzeichnet.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
OBJECT	immer
SOURCE	immer
ATTRIBUT	hatAttribute (“OBJECT”)
RULE	! existiert (“ALGORITHM”)
ALGORITHM	! existiert (“RULE”)
ACTION	immer
DESTINATION	immer
LIMIT	! existiert (“COMPARISON”) && ! existiert (“PRIMARY KEY”)
COMPARISON	! existiert (“LIMIT”) && ! existiert (“PRIMARY KEY”)
PRIMARY KEY	! existiert (“COMPARISON”) && ! existiert (“LIMIT”)

Beispiel:

Referenz:	**Belegung:**
Komponente:	Ruby Language
Programm:	Array # join
Version:	2.0.0
Quelle:	http://www.ruby-doc.org/core-2.0.0/
	(Letzter Abruf: 01.09.2013)
Beschreibung:	Standard-Klassenbibliothek für Ruby
Rolle:	**Belegung:**
OBJECT	Die Elemente des Arrays.
SOURCE	Das Array, auf dem die Operation aufgerufen wird.
RULE	Wird nicht benötigt.
ALGORITHM	Konvertiere jedes Array-Element in einen String (beginnend mit dem Index 0) und konkateniere diese Strings. Sofern der optionale Parameter `seperator` gesetzt ist, füge diesen String zwischen den Element-Strings ein.
ACTION	Konkateniere die Elemente eines Arrays zu einem String.
DESTINATION	Das Resultat.
LIMIT	Wird nicht benötigt (verarbeitet werden alle Elemente des Arrays).
COMPARISON	Alle Elemente des Arrays (`this`).
PRIMARY KEY	Wird nicht benötigt.

Die Angaben der thematischen Rollen ERROR OBJECT, ERROR SOURCE, ERROR RULE, ERROR ALGORITHM, ERROR ACTION, ERROR DESTINATION, ERROR LIMIT, ERROR COMPARISON und ERROR PRIMARY KEY sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.12 Parsende Operation

Beschreibung: Eine parsende Operation liest ATTRIBUTEs aus einer SOURCE um daraus ein OBJECT zu erstellen. Das OBJECT wird mit den ATTRIBUTEs initialisiert. Die erwartete Struktur der SOURCE wird durch ein FORMAT definiert. Das Vorgehen beim Lesen der SOURCE und beim Zusammensetzen des OBJECT wird durch einen ALGORITHM beschrieben. Eine parsende Operation kann Teilaufgaben an ein INSTRUMENT delegieren.
Verben: parse, read
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
ATTRIBUT	Der aus der SOURCE gelesene Wert.	J
SOURCE	Der Ort, von dem ATTRIBUTEs gelesen werden.	N
OBJECT	Die Struktur, die aus den gelesenen ATTRIBUTEN zusammengesetzt wird.	J
FORMAT	Die Struktur der SOURCE.	J
ALGORITHM	Definiert das Vorgehen beim Einlesen der SOURCE und Zusammensetzen des OBJECT.	N
INSTRUMENT	Externe Komponente, an die Aufgaben von der Operation delegiert werden können.	J
ACTION	Beschreibt die Operation.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
ATTRIBUT	immer
SOURCE	immer
OBJECT	immer
FORMAT	immer
ALGORITHM	immer
INSTRUMENT	immer
ACTION	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Ruby Language
Programm:	Date # parse
Version:	2.0.0
Quelle:	http://www.ruby-doc.org/core-2.0.0/ (Letzter Abruf: 01.09.2013)
Beschreibung:	Standard-Klassenbibliothek für Ruby
Rolle:	**Belegung:**
ATTRIBUTE	Die Tages-, Monats- und Jahresangabe des opt. Parameters `string`.
SOURCE	Der optionale Parameter `string`.
OBJECT	Das in `string` kodierte Datum.
FORMAT	Nicht spezifiziert (implizit und partiell enthalten in Quelltext der referenzierten Methode `lib/date/format.rb # _parse()`. Der Parameter `comp` definiert, ob eine zweistellige Jahresangabe XX als 19XX (`True`) oder 20XX (`False`, default) interpretiert werden soll.
ALGORITHM	Nicht spezifiziert.
INSTRUMENT	Die Methode `lib/date/format.rb # _parse()`.
ACTION	Extrahiere Datumsangaben aus dem übergebenen String und liefere das daraus erzeugte Datumsobjekt.

Die Angaben der thematischen Rollen ERROR ATTRIBUTE, ERROR SOURCE, ERROR OBJECT, ERROR FORMAT, ERROR ALGORITHM, ERROR INSTRUMENT und ERROR ACTION sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.13 Deponierende Operation

Beschreibung: Eine deponierende Operation legt ein vor dem Aufruf existierendes OBJECT an einer DESTINATION ab, z.B. in einer Sammlung oder einem Speicher. Das OBJECT kann dort durch einen PRIMARY KEY oder ein COMPARISON identifiziert werden. Dieses Raster basiert auf der VerbNet-Klasse put-9.1. Im Gegensatz zu einer erzeugenden Operation erstellt eine deponierende Operation das OBJECT nicht selbst, sondern legt es nur ab.
Verben: add, append, arrange, bury, deposit, embed, immerse, implant, insert, install, lodge, mount, move, park, persist, place, plant, position, put, save, set, situate, sling, stash, station, stick, store, stow, superimpose, write
Verwandte Bezeichnungen: Setter

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird deponiert.	N
DESTINATION	Der Ort, an dem das OBJECT deponiert wird.	J
PRIMARY KEY	Identifiziert das OBJECT in der DESTINATION eindeutig.	J
COMPARISON	Identifiziert das OBJECT in der DESTINATION.	J
ACTION	Beschreibt die Operation.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
OBJECT	immer
DESTINATION	immer
PRIMARY KEY	! existiert (“COMPARISON”)
COMPARISON	! existiert (“PRIMARY KEY”)
ACTION	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Platform, Standard Edition 7
Programm:	java.util.Map # put
Version:	1.7
Quelle:	http://docs.oracle.com/javase/7/docs/api/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Standard-Klassenbibliothek für Java
Rolle:	**Belegung:**
OBJECT	Parameter `value`
DESTINATION	Map (`this`)
PRIMARY KEY	Parameter `key`
COMPARISON	Wird nicht benötigt
ACTION	Fügt ein Schlüssel-Wert-Paar in eine Map (`this`) ein
ERROR OBJECT	`null` kann als Wert in einer |Map gespeichert werden.
ERROR PRIMARY KEY	`NullPointerException`, wenn die Implementierung der `Map` keine `null`-Schlüssel unterstützt oder `ClassCastException`, wenn der Typ des Schlüssels oder des Wertes nicht in der Implementierung der `Map` gespeichert werden kann.
ERROR ACTION	`UnsupportedOperationException`, wenn die Implementierung der Map diese Operation nicht unterstützt.

Die Angaben der thematischen Rolle ERROR DESTINATION sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.14 Löschende Operation

Beschreibung: Eine löschende Operation löscht ein oder mehrere OBJECTs permanent aus einer SOURCE. Die zu entfernenden OBJECTs werden anhand eines PRIMARY KEY, eines COMPARISON oder einer RULE bestimmt. Die Anzahl der zu entfernenden OBJECTs kann durch ein LIMIT begrenzt werden. Dieses Raster basiert auf den VerbNet-Klassen remove-10.1 und destroy-44.
Verben: abstract, blitz, cull, decimate, deduct, delete, demolish, depose, destroy, disengage, disfigure, disgorge, dismiss, eject, eradicate, excise, expel, extrude, maim, omit, ostracize, oust, pry, reap, remove, retract, separate, sever, subtract, uproot, withdraw, wreck
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird gelöscht.	J
SOURCE	Der Ort, an dem das OBJECT gelöscht wird.	J
PRIMARY KEY	Identifiziert das OBJECT in der SOURCE eindeutig.	J
COMPARISON	Identifiziert das OBJECT in der SOURCE.	J
RULE	Definiert die zu löschenden OBJECTs.	J
ACTION	Beschreibt die Operation.	N
LIMIT	Die Unter- oder Obergrenze der zu löschenden OBJECTs.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
OBJECT	immer
SOURCE	immer
PRIMARY KEY	! existiert (“COMPARISON”) && ! existiert (“RULE”)
COMPARISON	! existiert (“PRIMARY KEY”) && ! existiert (“RULE”)
RULE	! existiert (“PRIMARY KEY”) && ! existiert (“COMPARISON”)
ACTION	immer
LIMIT	! existiert (“PRIMARY KEY”)

Beispiel:

Referenz:	**Belegung:**
Komponente:	jQuery
Programm:	remove
Version:	1.9
Quelle:	http://api.jquery.com/ (Letzter Abruf: 10.06.2013)
Beschreibung:	JavaScript-Rahmenwerk zur Web-Entwicklung
Rolle:	**Belegung:**
OBJECT	Die DOM-Elemente.
SOURCE	Der DOM-Baum, auf dem die Operation aufgerufen wurde.
PRIMARY KEY	Der Parameter `selector` im Format eines jQuery ID-Selectors.
COMPARISON	Der Parameter `selector`.
RULE	Die DOM-Elemente, auf die die im Parameter `selector` formulierte Bedingung zutrifft.
ACTION	Löscht diejenigen Elemente aus dem DOM-Baum, auf welche die im Parameter `selector` formulierte Bedingung zutrifft. DOM steht für Document Object Model und ist ein standardisierte Schnittstelle zum Zugriff auf Dokumente (für weitere Angaben zum Document Object Model wird auf die Internetseite http://www.w3.org/DOM/ des World Wide Web Consortiums verwiesen (letzter Abruf 01.09.2013)).

Die Angaben der thematischen Rolle ERROR OBJECT, ERROR SOURCE, ERROR PRIMARY KEY, ERROR COMPARISON, ERROR RULE und ERROR ACTION worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.15 Rücksetzende Operation

Beschreibung: Ein rücksetzende Operation verändert bestimmte ATTRIBUTEs eines OBJECT oder das OBJECT selbst in der Weise, dass es einem früheren Zustand entspricht. Dieser alte Zustand stammt aus einer SOURCE.
Verben: reset, revert
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
OBJECT	Wird zurückgesetzt oder kapselt die zurückgesetzten ATTRIBUTEs.	J
ATTRIBUTE	Wird zurückgesetzt.	N
ACTION	Beschreibt die Operation.	N
SOURCE	Enthält die ursprünglichen Belegungen der ATTRIBUTEs.	J

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
OBJECT	immer
ATTRIBUTE	hatAttribute (“OBJECT”)
ACTION	immer
SOURCE	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Platform, Standard Edition 7
Programm:	java.io.InputStream # reset
Version:	1.7
Quelle:	http://docs.oracle.com/javase/7/docs/api/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Standard-Klassenbibliothek für Java
Rolle:	**Belegung:**
OBJECT	Der `InputStream`, auf dem die Operation aufgerufen worden ist.
ATTRIBUTE	Die Position im Byte-Strom, ab der gelesen wird.
ACTION	Setzt den `InputStream` derartig zurück, dass die ab der markierten Position enthaltenen Bytes erneut gelesen werden können.
SOURCE	Die Position, bei der zuletzt die Operation `mark` aufgerufen worden ist.
ERROR ACTION	`IOException`, wenn die Implementierung des `InputStream` keine Markierungen unterstützt.

Die Angaben der thematischen Rolle ERROR OBJECT, ERROR ATTRIBUTE und ERROR SOURCE sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.16 Suchende Operation

Beschreibung: Eine suchende Operation sucht nach einem oder mehreren OBJECTs in einer SOURCE. Die OBJECTs werden anhand eines PRIMARY KEY oder eines COMPARISON identifiziert. Die Anzahl der zu findenden OBJECTs kann durch Angabe einer Unter- oder Obergrenze in Form eines LIMITS beschränkt werden. Sollten potentiell mehrere OBJECTs gefunden werden, so können diese gemäß eines ORDERING angeordnet sein. Ein ALGORITHM kann das Vorgehen bei der Suche spezifizieren. Dieses Raster basiert auf den VerbNet-Klassen Search-35.2 und obtain-13.5.2.
Verben: accept, accrue, accumulate, acquire, advertise, appropriate, borrow, cadge, collect, comb, dive, drag, dredge, exact, excavate, extract, find, get, grab, inherit, obtain, patrol, plumb, probe, prospect, prowl, purchase, quarry, rake, receive, recover, regain, retrieve, rifle, scavenge, scour, scout, search, seize, select, shop, sift, snatch, trawl, troll, watch
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
ACTION	Beschreibt die Operation.	N
OBJECT	Wird gesucht.	J
SOURCE	Wird durchsucht.	J
PRIMARY KEY	Identifiziert das gesuchte OBJECT eindeutig.	J
COMPARISON	Identifiziert das gesuchte OBJECT.	J
LIMIT	Die Ober- oder Untergrenze der Anzahl der gelieferten OBJECTs.	N
ORDERING	Definiert die Reihenfolge der gelieferten OBJECTs.	N
ALGORITHM	Definiert die Vorgehensweise bei der Suche.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
ACTION	immer
OBJECT	immer
SOURCE	immer
PRIMARY KEY	! existiert (“COMPARISON”)
COMPARISON	! existiert (“PRIMARY KEY”)
LIMIT	istPlural (“OBJECT”)
ORDERING	istPlural (“OBJECT”)
ALGORITHM	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java Platform, Standard Edition 7
Programm:	javax.activation.MimetypesFileTypeMap # getContentType
Version:	1.7
Quelle:	http://docs.oracle.com/javase/7/docs/api/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Standard-Klassenbibliothek für Java
Rolle:	**Belegung:**
OBJECT	Der MIME-Typ für die per Parameter `f` spezifizierte Dateiendung (Standard: “application/octet-stream”).
ACTION	Sucht den MIME-Typ für die durch den Parameter `f` spezifizierte Dateiendung.
SOURCE	Siehe aus dem Vertrag kopierte Abbildung C.2 für die durchsuchten Dateien.
PRIMARY KEY	Wird nicht benötigt.
COMPARISON	Die Endung der Datei.
LIMIT	1
ORDERING	Wird nicht benötigt.
ALGORITHM	Nicht spezifiziert.

Die Angaben der thematischen Rolle ERROR OBJECT, ERROR ACTION , ERROR SOURCE, ERROR PRIMARY KEY, ERROR COMPARISON, ERROR LIMIT, ERROR

MIME types file search order:

The MimetypesFileTypeMap looks in various places in the user's system for MIME types file entries. When requests are made to search for MIME types in the MimetypesFileTypeMap, it searches MIME types files in the following order:

1. Programmatically added entries to the MimetypesFileTypeMap instance.
2. The file `.mime.types` in the user's home directory.
3. The file <*java.home*>`/lib/mime.types`.
4. The file or resources named `META-INF/mime.types`.
5. The file or resource named `META-INF/mimetypes.default` (usually found only in the `activation.jar` file).

Abbildung C.2: SOURCE der Operation getContentType()

ORDERING und ERROR ALGORITHM sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.17 Sendende Operationen

Beschreibung: Eine sendende Operation informiert ein oder mehrere PARTICIPANTs an einer oder mehreren DESTINATIONs über ein Ereignis. Die zu informierenden PARTICIPANTs werden aufgrund eines COMPARISON oder eines PRIMARY KEY identifiziert. Die Benachrichtigung geschieht über eine MESSAGE eines bestimmten Typs (E-Mail, Brief, etc.) in einem bestimmten FORMAT. Die MESSAGE wird mit einem INSTRUMENT verschickt bzw. transportiert. Ein REPORT gibt an, ob die MESSAGE erfolgreich abgeschickt werden konnte oder nicht. Dieses Raster basiert auf den VerbNet-Klassen tell-37.2 und send-11.1.
Verben: acquaint, advise, airmail, apprise, convey, deliver, dispatch, express, fedex, forward, hand, inform, mail, notify, pass, pass on, port, post, print, remind, return, send, shift, ship, shunt, slip, smuggle, sneak, tell, transfer, transmit, transport, update, ups, wire
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	**Beschreibung**	**Pflicht**
ACTION	Beschreibt die Operation.	N
OBJECT	Wird informiert.	J
DESTINATION	Wo das OBJECT die MESSAGE empfängt.	J
PRIMARY KEY	Identifiziert das adressierte OBJECT eindeutig.	J
COMPARISON	Identifiziert das adressierte OBJECT.	J
MESSAGE	Wird verschickt.	J
FORMAT	Der Inhalt bzw. die Struktur der MESSAGE.	J
INSTRUMENT	Verschickt die MESSAGE.	J
REPORT	Enthält den Versandstatus der MESSAGE.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
ACTION	immer
PARTICIPANT	immer
DESTINATION	immer
PRIMARY KEY	! existiert (“COMPARISON”)
COMPARISON	! existiert (“PRIMARY KEY”)
MESSAGE	immer
FORMAT	immer
INSTRUMENT	immer
REPORT	immer

Beispiel:

Referenz:	**Belegung:**
Komponente:	Java EE 7 Specification APIs
Programm:	javax.mail.Transport # send (Message)
Version:	7
Quelle:	http://docs.oracle.com/javaee/7/api/ (Letzter Abruf: 10.06.2013)
Beschreibung:	Erw. Klassenbibliothek für Java
Rolle:	**Belegung:**
ACTION	Verschickt die per Parameter `msg` Nachricht als E-Mail.
PARTICIPANT	Der über die Methode `addRecipient` gesetzte Empfänger der E-Mail (Parameter `msg`).
DESTINATION	Das E-Mail-Postfach des über die Methode `addRecipient` gesetzten Empfängers.
PRIMARY KEY	Wird nicht benötigt.
COMPARISON	Wird nicht benötigt.
MESSAGE	Der Parameter `msg`.
FORMAT	E-Mail / MIME.
INSTRUMENT	Abhängig von der Implementierung.
REPORT	Wenn keine Ausnahme geworfen wird, konnte die E-Mail erfolgreich versendet werden.

Die Angaben der thematischen Rolle ERROR ACTION, ERROR PARTICIPANT , ERROR DESTINATION, ERROR PRIMARY KEY, ERROR COMPARISON, ERROR MESSAGE, ERROR FORMAT, ERROR INSTRUMENT und ERROR REPORT sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.18 Initiierende Operation

Beschreibung: Startet eine oder mehrere weitere ACTIONs. Die Dauer der gestarteten ACTIONs kann mit einer TIME TO LIVE beschränkt werden. Dieses Raster basiert auf der VerbNet-Klasse begin-55.1.
Verben: begin, commence, go on, notify, pledge, proceed, recommence, resume, start, start off, trigger, undertake, update
Verwandte Bezeichnungen: Trigger

Thematische Rollen:

Rolle	**Beschreibung**	**Pflicht**
ACTION	Was gestartet wird.	N
TIME TO LIVE	Maximale Zeitdauer der gestarteten ACTIONs.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
ACTION	immer
TIME TO LIVE	immer

Beispiel:

Referenz:	**Belegung:**
Komponente: Programm: Version: Quelle: Beschreibung:	jQuery trigger 1.9 http://api.jquery.com/ (Letzter Abruf: 10.06.2013) JavaScript-Rahmenwerk zur Web-Entwicklung
Rolle:	**Belegung:**
ACTION	Startet die Behandlung des per Parameter `eventType` übergebenen Ereignisses durch die am jeweiligen DOM-Element angemeldeten Event-Handler.
TIME TO LIVE	Nicht spezifiziert.

Die Angaben der thematischen Rolle ERROR ACTION und ERROR TIME TO LIVE sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.19 Substituierende Operation

Beschreibung: Eine substituierende Operation ersetzt ein oder mehrere DESTINATION OBJECTs an einer DESTINATION durch andere SOURCE OBJECTs aus einer SOURCE. Die zu ersetzenden DESTINATION OBJECTs werden anhand eines DESTINATION COMPARISON oder eines DESTINATION PRIMARY KEY identifiziert. Die Identifikation der entsprechenden SOURCE OBJECTs erfolgt analog durch ein SOURCE COMPARISON oder einen SOURCE PRIMARY KEY. Dieses Raster basiert auf der VerbNet-Klasse exchange-13.6.
Verben: barter, change, exchange, replace, substitute, swap, switch, trade
Verwandte Bezeichnungen: Keine

Thematische Rollen:

Rolle	Beschreibung	Pflicht
ACTION	Beschreibt die Operation.	N
DESTINATION OBJECT	Wird ersetzt durch das SOURCE OBJECT.	J
SOURCE OBJECT	Ersetzt das DESTINATION OBJECT.	J
DESTINATION	Der Ort, an dem das DESTINATION OBJECT ersetzt wird.	J
SOURCE	Der Ort, an dem sich das SORUCE OBJECT vor der Ersetzung befindet.	J
DESTINATION COMPARISON	Identifiziert mehrere DESTINATION OBJECTs.	N
DESTINATION PRIMARY KEY	Identifiziert ein DESTINATION OBJECT eindeutig.	N
SOURCE COMPARISON	Identifiziert mehrere SOURCE OBJECTs.	N
SOURCE PRIMARY KEY	Identifiziert ein SOURCE OBJECT eindeutig.	N

Rasterbasierte Regeln:

Rolle	**Gefordert wenn ...**
ACTION	immer
DESTINATION OBJECT	immer
SOURCE OBJECT	immer
DESTINATION	immer
SOURCE	immer
DESTINATION COMPARISON	! existiert (“DESTINATION PRIMARY KEY”)
DESTINATION PRIMARY KEY	! existiert (“DESTINATION COMPARISON”)
SOURCE COMPARISON	! existiert (“SOURCE PRIMARY KEY”)
SOURCE PRIMARY KEY	! existiert (“SOURCE COMPARISON”)

Beispiel:

Referenz:	**Belegung:**
Komponente: Programm: Version: Quelle: Beschreibung:	.NET Framework String # replace (String, String) 4.5 http://msdn.microsoft.com/de-de/library/ (Letzter Abruf: 10.06.2013) Rahmenwerk zur Entwicklung für die Windows-Plattform
Rolle:	**Belegung:**
ACTION	Ersetzt alle Vorkommen des Parameters `oldValue` im aktuellen String durch den Wert des Parameters `newValue`.
DESTINATION	Das Resultat.
SOURCE	Der aktuelle String.
DESTINATION OBJECT	Alle Vorkommen des Parameters `newValue` im Resultat.
SOURCE OBJECT	Alle Vorkommen des Parameters `oldValue` im aktuellen String.
DESTINATION COMPARISON	Der Parameter `newValue`.
DESTINATION PRIMARY KEY	Nicht spezifiziert.
SOURCE COMPARISON	Der Parameter `oldValue`.
SOURCE PRIMARY KEY	Nicht spezifiziert.
ERROR SOURCE COMPARISON	Die Ausnahmen `ArgumentNullException` bzw. `ArgumentException`
ERROR DESTINATION COMPARISON	Wenn der Parameter `newValue` mit `null` belegt ist, dann werden alle Vorkommen von `oldValue` aus dem aktuellen String entfernt.

Die Angaben der thematischen Rolle ERROR ACTION, ERROR DESTINATION, ERROR SOURCE, ERROR DESTINATION OBJECT, ERROR SOURCE OBJECT, ERROR DESTINATION PRIMARY KEY und ERROR SOURCE PRIMARY KEY sind

ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

C.2.20 Traversierende Operation

Beschreibung: Eine traversierende Operation besucht ein oder mehrere OBJECTs an einer DESTINATION. Die besuchten OBJECTs werden anhand eines COMPARISON oder PRIMARY KEY ausgewählt. Auf jedes besuchte OBJECT wird ein ALGORITHM angewendet. Das Resultat der Besuche kann in einem REPORT zusammengefasst werden. Dieses Raster basiert auf der VerbNet-Kategorie swarm-47.5.1.
Verben: abound, bustle, crawl, creep, hop, iterate, run, swarm, swim, teem, throng, traverse, visit
Verwandte Bezeichnungen: Besucher-Entwurfsmuster

Thematische Rollen:

Rolle	Beschreibung	Pflicht
ACTION	Beschreibt die Operation selbst.	N
OBJECT	Wird besucht.	J
DESTINATION	Der Ort, an dem sich das OBJECT befindet.	J
COMPARISON	Identifiziert das OBJECT.	N
PRIMARY KEY	Identifiziert das OBJECT eindeutig.	N
ALGORITHM	Beschreibt die Verarbeitung jedes besuchten OBJECTs.	J
REPORT	Die Zusammenfassung der Resultate der Besuche.	N

Rasterbasierte Regeln:

Rolle	Gefordert wenn ...
ACTION	immer
OBJECT	immer
DESTINATION	immer
COMPARISON	! existiert (“PRIMARY KEY”)
PRIMARY KEY	! existiert (“COMPARISON”)
ALGORITHM	immer
REPORT	immer

Beispiel:

Referenz:	**Belegung:**
Komponente: Programm: Version: Quelle: Beschreibung:	.JBoss ModeShape Public API org.modeshape.graph.query.model.Visitors # visitAll 4.5 http://docs.jboss.org/modeshape/2.1.0.Final/api/ (letzter Abruf: 11.06.2013) Archivierungssystem für Dateien, welches dem Java-Standard JSR-283 entspricht.
Rolle:	**Belegung:**
ACTION	Besucht den übergebenen Wurzelknoten (Parameter `visitable`), sowie alle dessen Kindknoten.
OBJECT	Der übergebene Wurzelknoten (Parameter `visitable`), sowie alle dessen Kindknoten.
DESTINATION	Der Parameter `visitable`.
COMPARISON	Alle erreichbaren Knoten werden besucht.
PRIMARY KEY	Wird nicht benötigt.
ALGORITHM	Der Parameter `strategyVisitor`.
REPORT	Das Resultat (Abruf der vom `strategyVisitor` gesammelten Daten über von der jeweiligen Implementierung des Besuchers abhängigen Methoden).

Die Angaben der thematischen Rolle ERROR ACTION, ERROR OBJECT, ERROR DESTINATION, ERROR COMPARISON, ERROR PRIMARY KEY, ERROR ALGORITHM und ERROR REPORT sind ausgelassen worden, da diese Fehlerfälle nicht spezifiziert sind oder nicht eintreten können.

Anhang D

Anhang zur Evaluierung

D.1 Aufgabenstellung zur Evaluierung

Aufgabenstellung zur Übung MBSE

1 Szenario

Sie sind beim Buchversandhändler Bookseller AG in Hamburg als Software-Architekt angestellt und erhalten vom Stab zur Prozessoptimierung eine verbesserte Version des Bestellabwicklungsprozesses. Dieser soll nun erstmalig zu einem Großteil durch Software unterstützt werden. Von dieser Maßnahme verspricht sich die Bookseller AG Kosteneinsparungen in Höhe von ca. 1 Mio. EUR jährlich.

2 Aufgabenstellung

Mit dieser Aufgabenstellung erhalten Sie ein BPMN-Diagramm des Bestellprozesses der Bookseller AG. Sie haben die Aufgabe, die Implementierung des beschriebenen Geschäftsprozesses für einen Software-Entwickler exakt zu spezifizieren. Die Form der zu erstellenden Spezifikation ist wie folgt vorgegeben:

1. Jede der im Diagramm enthaltenen Abteilungen (repräsentiert durch Lanes des Pools "Bookseller AG") stellt ein eigenes Java-Interface.

2. Die Spezifikation der von jeder Abteilung angebotenen Dienstleistungen erfolgt in Form von Java-Methoden im zugehörigen Java-Interface. **Bitte spezifizieren Sie nur (!) die in Orange gefärbten Aktivitäten.** Die Aktivitäten "Prüfe Bonität anhand der Rechnungsanschrift" und "Erzeuge Packauftrag" sind optional. Bitte spezifizieren Sie diese Aktivitäten nur, wenn Sie über ausreichend Restzeit verfügen.

3. Die Spezifikation der Dienstleistungen einer Abteilung erfolgt als kommentierte Java-Methode.

4. Die im Prozess benötigten Datenstrukturen sind ebenfalls in Form von Java-Klassen vorgegeben.

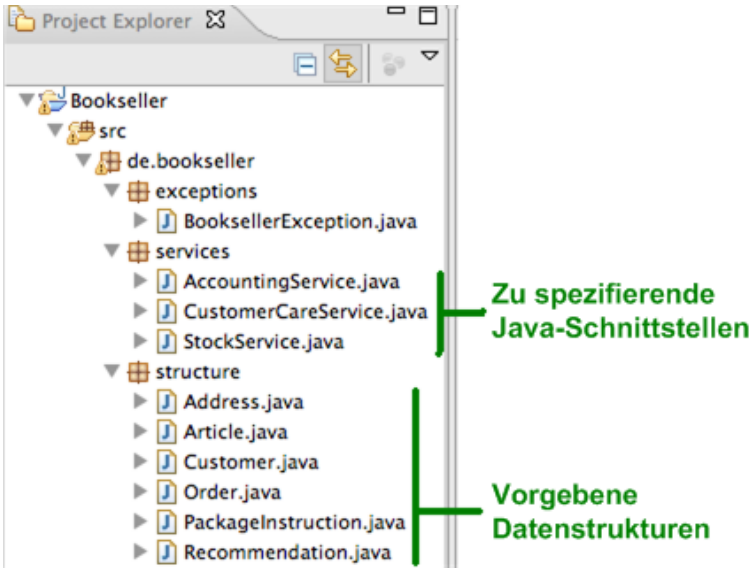


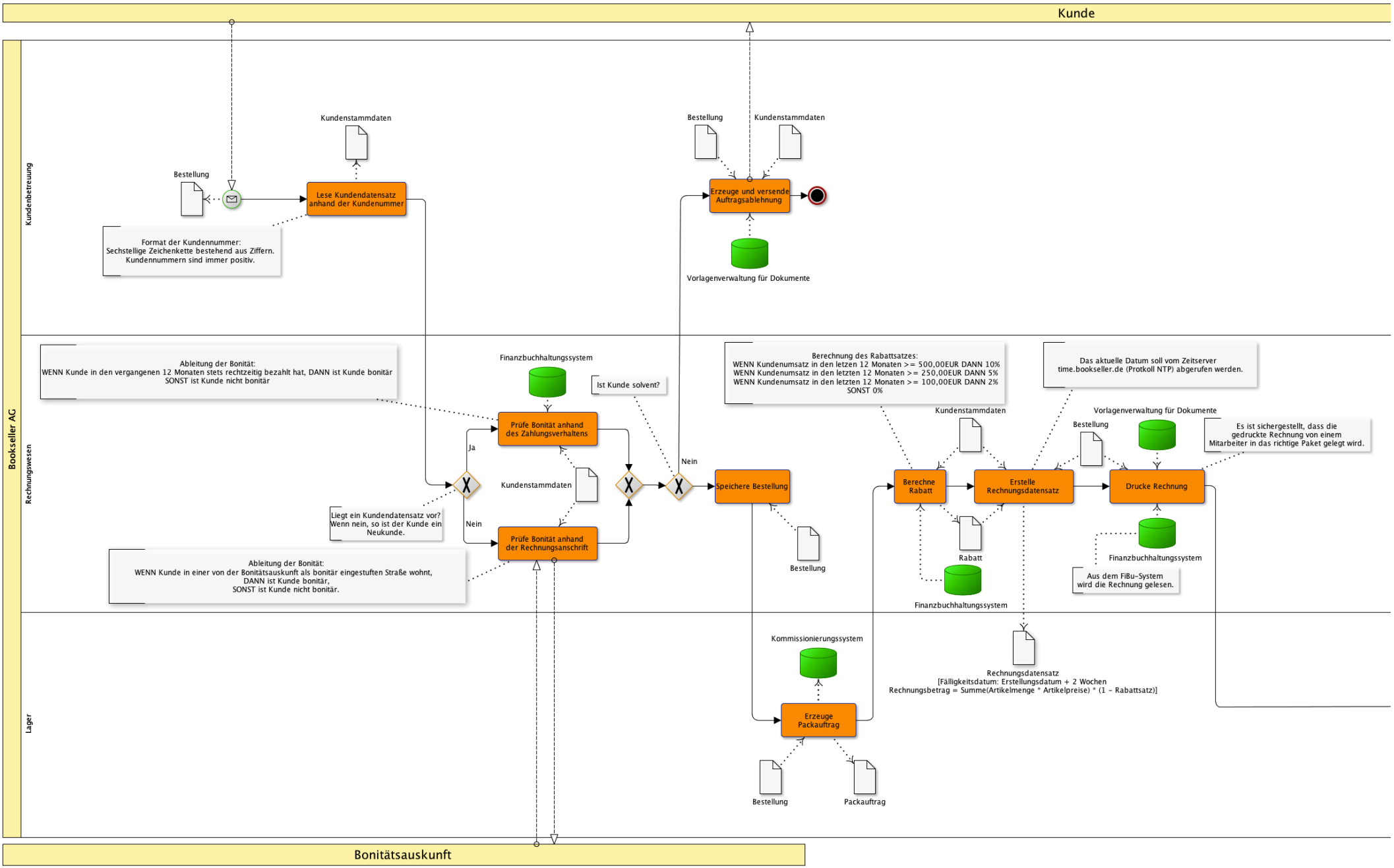
Abbildung 1: Vorgegebene Projektstruktur

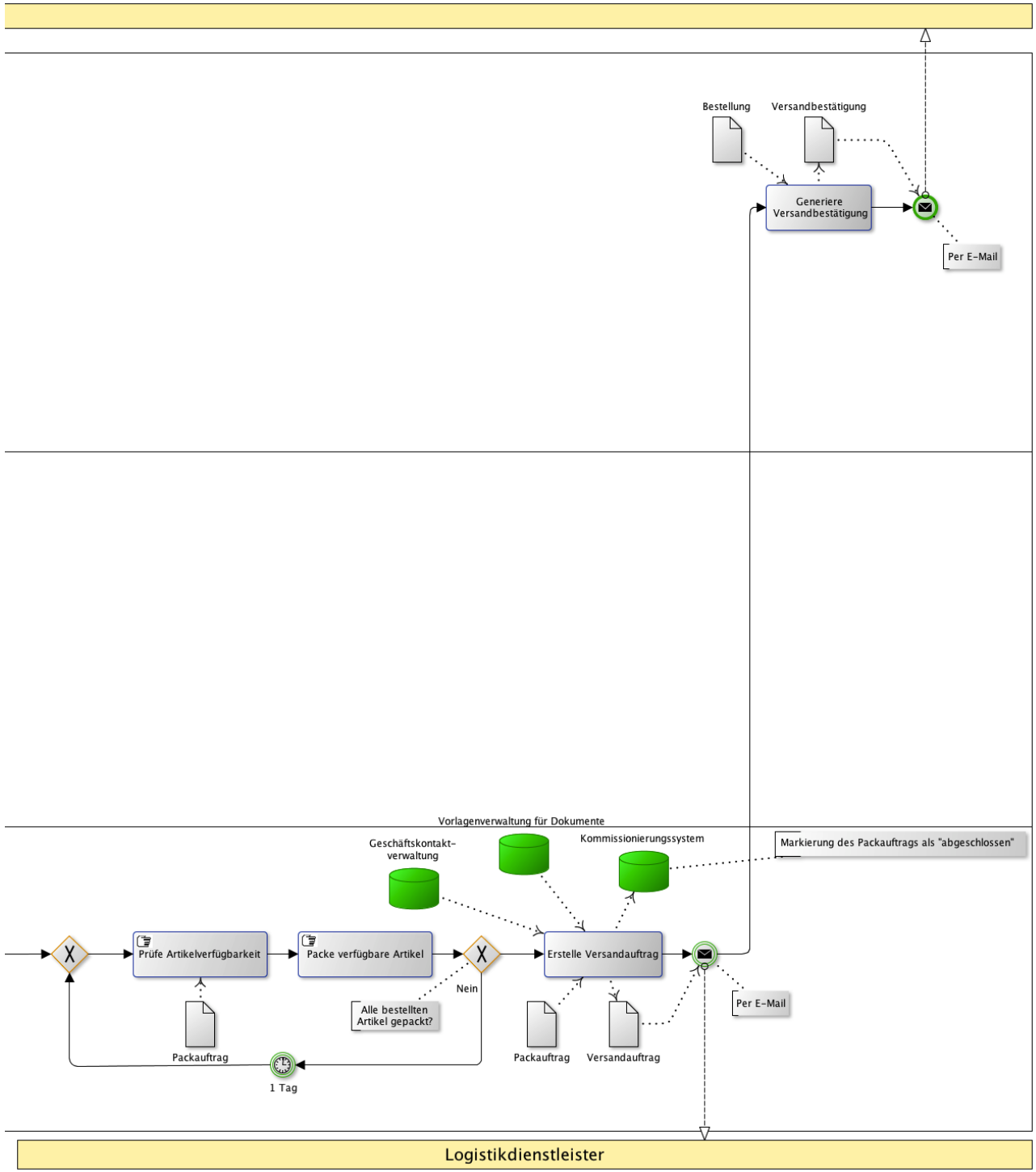
3 Ergänzende Hinweise zum BPMN-Modell

- E-Mails werden bei der Bookseller AG über den SMTP-Server mit dem Namen smtp.bookseller.de verschickt.

- Druckaufträge werden bei der Bookseller AG über zentralen Druck-Server mit dem Namen print.bookseller.de im Postscript-Format abgewickelt.

- Vorlagen für alle Arten von Dokumenten werden bei der Bookseller AG im System "Vorlagenverwaltung für Dokumente" (abgekürzt "VoDo") gespeichert.


Kunde
Bookseller AG
Kundenbetreuung
Rechnungswesen
Lager
Bestellung
Kundenstammdaten
Lese Kundendatensatz anhand der Kundenummer
Format der Kundennummer:
Sechstellige Zeichenkette bestehend aus Ziffern.
Kundennummern sind immer positiv.
Bestellung
Kundenstammdaten
Erzeuge und versende Auftragsablehnung
Vorlagenverwaltung für Dokumente
Ableitung der Bonität:
WENN Kunde in den vergangenen 12 Monaten stets rechtzeitig bezahlt hat, DANN ist Kunde bonitär
SONST ist Kunde nicht bonitär
Finanzbuchhaltungssystem
Ist Kunde solvent?
Prüfe Bonität anhand des Zahlungsverhaltens
Ja
Nein
Kundenstammdaten
Liegt ein Kundendatensatz vor?
Wenn nein, so ist der Kunde ein Neukunde.
Nein
Prüfe Bonität anhand der Rechnungsanschrift
Ableitung der Bonität:
WENN Kunde in einer von der Bonitätsauskunft als bonitär eingestuften Straße wohnt,
DANN ist Kunde bonitär,
SONST ist Kunde nicht bonitär.
Nein
Speichere Bestellung
Bestellung
Berechnung des Rabattsatzes:
WENN Kundenumsatz in den letzen 12 Monaten >= 500,00EUR DANN 10%
WENN Kundenumsatz in den letzten 12 Monaten >= 250,00EUR DANN 5%
WENN Kundenumsatz in den letzten 12 Monaten >= 100,00EUR DANN 2%
SONST 0%
Kundenstammdaten
Berechne Rabatt
Rabatt
Finanzbuchhaltungssystem
Das aktuelle Datum soll vom Zeitserver time.bookseller.de (Protkoll NTP) abgerufen werden.
Vorlagenverwaltung für Dokumente
Bestellung
Erstelle Rechnungsdatensatz
Drucke Rechnung
Es ist sichergestellt, dass die gedruckte Rechnung von einem Mitarbeiter in das richtige Paket gelegt wird.
Finanzbuchhaltungssystem
Aus dem FiBu-System wird die Rechnung gelesen.
Rechnungsdatensatz
[Fälligkeitsdatum: Erstellungsdatum + 2 Wochen
Rechnungsbetrag = Summe(Artikelmenge * Artikelpreise) * (1 - Rabattsatz)]
Kommissionierungssystem
Erzeuge Packauftrag
Bestellung
Packauftrag
Bonitätsauskunft


Bestellung
Versandbestätigung
Generiere Versandbestätigung
Per E-Mail
Vorlagenverwaltung für Dokumente
Geschäftskontakt-verwaltung
Kommissionierungssystem
Markierung des Packauftrags als "abgeschlossen"
Prüfe Artikelverfügbarkeit
Packe verfügbare Artikel
Erstelle Versandauftrag
Nein
Alle bestellten Artikel gepackt?
Per E-Mail
Packauftrag
Packauftrag
Versandauftrag
1 Tag
Logistikdienstleister

4 Sonstige Hinweise zur Bearbeitung

1. Aus dem Prozessmodell ersichtliche Fehlerfälle (bestehend aus Fehlernummer und -nachricht) der Methoden müssen über Java-Exceptions abgebildet werden. Die Basisklasse hierfür ist die vorgegebene Klasse `de.bookseller.exceptions.BooksellerException`. Sie können bei Bedarf weitere Spezialisierungen dieser Exception-Klasse definieren.

2. Die Bezeichner der Methoden und Parameter, sowie die javadoc-Kommentare sind in englischer Sprache zu formulieren.

3. Im Rahmen dieser Übung werden keine Paket- oder Klassenkommentare betrachtet oder erstellt.

4. Bitte sehen Sie von HTML-Kosmetik in den Kommentaren ab. Die Kommentare müssen im Java-Code-Editor von Eclipse lesbar sein.

5. Sollten aus Ihrer Sicht für die Spezifikation relevante Aspekte im BPMN-Modell nicht enthalten sein, so erwähnen Sie diese Aspekte bitte in Ihren Kommentaren mit dem Vermerk TBD ("to be defined").

6. Bitte gehen Sie bei Ihrer Spezifikation nicht auf Hardware- oder Betriebssystemaspekte ein!

5 Glossar

Bei der Bookseller AG werden die im Folgenden aufgeführten deutschen Begriffe wie angegeben ins Englische übersetzt:

- Bestellung: Order
- Kunde: Customer
- Kundenstammdaten: Customer Record
- Bonität: Solvency
- Bonitätsauskunft: Solvency Agency
- Auftragsablehnung: Order Rejection
- Rabatt: Discount
- Finanzbuchhaltungssystem: Accounting System
- Rechnungswesen: Accounting
- Lager: Stock
- Kundenbetreuung: Customer Care
- Rechnung: Invoice
- Vorlagenverwaltung für Dokumente: VoDo (Systemname, wird nicht übersetzt)
- Kommissionierungssystem: KoSy (Systemname, wird nicht übersetzt)
- Finanzbuchhaltungssystem: FiBu (Systemname, wird nicht übersetzt)

D.2 Stand. Fragebogen (Versuchsgruppe)

Fragebogen zur Übung MBSE
Version V

1. Wie lautet Ihre SVN-Repository-Kennung?

mbse ☐☐

---

2. Für wieviele Semester haben Sie einen Informatik-Studiengang belegt ?

Semester ☐☐

---

3. Wieviele Monate haben Sie bis heute in Software-Entwicklungprojekten mitgearbeitet (z.B. an der Universität, in der Industrie oder im Open Source-Umfeld)?

Monate ☐☐

---

4. Ist Ihnen das Konzept der thematischen Raster im Kontext der API-Spezifikation bekannt?

☐ Ja
☐ Nein

---

5. Bitte schätzen Sie Ihre Vorkenntnisse ein:

	Etwas darüber gehört / gelesen	In Lehrveranstaltung behandelt	In Praktikum / Übung behandelt	In Entwicklungsprojekt angewendet
Java	☐	☐	☐	☐
Eclipse	☐	☐	☐	☐
Subversion	☐	☐	☐	☐
Business Process Modelling Notation (BPMN)	☐	☐	☐	☐

---

6. Bitte bewerten Sie, ob die folgenden Aussagen bzgl. des Themengebietes "API-Spezifikation" zutreffend oder unzutreffend sind.

	trifft zu	trifft nicht zu	weiß nicht
Der Vertrag einer Operation muss alle "von außen" sichtbaren Effekte der Operation beschreiben (z.B. das Versenden einer E-Mail).	☐	☐	☐
Der Vertrag einer Operation kann in seltenen Fällen von der Operation genutzte Ressourcen auslassen.	☐	☐	☐
Der Vertrag einer Operation muss alle möglicherweise auftretenden Fehlersituationen /-ausgaben der Operation beschreiben.	☐	☐	☐
Der Vertrag einer Operation aus einer Schnittstelle muss möglichst detailliert auf die möglichen Implementierungen eingehen.	☐	☐	☐
Der Vertrag einer Operation muss sich zu Wertebereichen und insbesondere deren Einschränkung von Parametern äußern.	☐	☐	☐

7. Bitte bewerten Sie, ob die folgenden Aussagen bzgl. iDocIt! zutreffend oder unzutreffend sind.

	trifft zu	trifft nicht zu	weiß nicht
Die von iDocIt! zur Spezifikation empfohlenen Inhalte leiten sich primär aus dem Verb (Prädikat) im Bezeichner der Operation ab.	☐	☐	☐
iDocIt! kategorisiert Operationen anhand der Datentypen der formalen Parameter.	☐	☐	☐
Die Empfehlungen von iDocIt! beziehen sich auch auf nicht an der Signatur der Operation sichtbare Aspekte, wie z.B. eine genutzte DB als Ressource.	☐	☐	☐
iDocIt! kann die Anzahl der von der Operation genutzten Ressourcen exakt ermitteln.	☐	☐	☐
Sobald Ressourcen für eine Operation nicht verfügbar sind, müssen diese den vorliegenden Fehler behandeln. iDocIt! weist auf die fehlende Spezifikation solcher Fehlerbehandlungen hin.	☐	☐	☐
Die Empfehlungen von iDocIt! sind statisch festgelegt und können nicht verändert werden.	☐	☐	☐
iDocIt! nutzt bereits die erstellte Spezifikation einer Operation für neue Vorschläge.	☐	☐	☐
Die thematische Rolle SOURCE einer *suchenden Operation* fragt nach dem verwendeten Suchkriterium.	☐	☐	☐
Die thematische Rolle FORMAT einer *sendenden Operation* fragt nach der Struktur für die gesendete Nachricht.	☐	☐	☐
Die thematische Rolle FORMULA einer *berechnenden Operation* fragt nach der zugrunde liegenden Formel.	☐	☐	☐

---

8. Bitte positionieren Sie sich bzgl. der folgenden Aussagen:

"Die vollständige Kommentierung jeder Operation im eigenen Programm ist grundsätzlich und in jedem Fall sinnvoll."

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich **stimme nicht zu**									Ich **stimme zu**

“Die vollständige Kommentierung jeder public-Operation im eigenen Programm ist grundsätzlich und in jedem Fall sinnvoll.”

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich	**stimme**								Ich **stimme**
nicht zu									**zu**

“Die vollständige Kommentierung jeder Operation einer API des eigenen Programms ist grundsätzlich und in jedem Fall sinnvoll.”

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich	**stimme**								Ich **stimme**
nicht zu									**zu**

“Kommentare in Programmen sind grundsätzlich überflüssig.”

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich	**stimme**								Ich **stimme**
nicht zu									**zu**

D.3 Stand. Fragebogen (Kontrollgruppe)

Fragebogen zur Übung MBSE
Version K

1. Wie lautet Ihre SVN-Repository-Kennung?

mbse ☐☐

---

2. Für wieviele Semester haben Sie einen Informatik-Studiengang belegt?

Semester ☐☐

---

3. Wieviele Monate haben Sie bis heute in Software-Entwicklungprojekten mitgearbeitet (z.B. an der Universität, in der Industrie oder im Open Source-Umfeld)?

Monate ☐☐

---

4. Ist Ihnen das Konzept der thematischen Raster im Kontext der API-Spezifikation bekannt?

☐ Ja
☐ Nein

---

5. Bitte schätzen Sie Ihre Vorkenntnisse ein:

	Etwas darüber gehört / gelesen	In Lehrveranstaltung behandelt	In Praktikum / Übung behandelt	In Entwicklungsprojekt angewendet
Java	☐	☐	☐	☐
Eclipse	☐	☐	☐	☐
Subversion	☐	☐	☐	☐
Business Process Modelling Notation (BPMN)	☐	☐	☐	☐

6. Bitte bewerten Sie, ob die folgenden Aussagen bzgl. des Themengebietes "API-Spezfikation" zutreffend oder unzutreffend sind.

	trifft zu	trifft nicht zu	weiß nicht
Der Vertrag einer Operation muss alle "von außen" sichtbaren Effekte der Operation beschreiben (z.B. das Versenden einer E-Mail).	☐	☐	☐
Der Vertrag einer Operation kann in seltenen Fällen von der Operation genutzte Ressourcen auslassen.	☐	☐	☐
Der Vertrag einer Operation muss alle möglicherweise auftretenden Fehlersituationen /-ausgaben der Operation beschreiben.	☐	☐	☐
Der Vertrag einer Operation aus einer Schnittstelle muss möglichst detailliert auf die möglichen Implementierungen eingehen.	☐	☐	☐
Der Vertrag einer Operation muss sich zu Wertebereichen und insbesondere deren Einschränkung von Parametern äußern.	☐	☐	☐

7. Bitte bewerten Sie, ob die folgenden Aussagen bzgl. des Themengebietes "javadoc" zutreffend oder unzutreffend sind.

	trifft zu	trifft nicht zu	weiß nicht
Selbsterklärende Parameter bedürfen in javadoc keiner weiteren, expliziten Kommentare.	☐	☐	☐
Bei Vorgabe eines Algorithmus muss dieser mit Namen über das Tag "@algorithm" im Vertrag angegeben werden.	☐	☐	☐
Alle möglicherweise auftretenden Ausnahmen (Exceptions) sollten durch eine möglichst allgemeine und generische Beschreibung zusammengefasst werden (nach "@throws Exception").	☐	☐	☐
javadoc verlangt die Angabe des Namens eines Autoren mittels des Tags "@author" zwingend für jede Operation.	☐	☐	☐
Zur Kommentierung eines Parameters ist in javadoc nach Angabe des Tags "@param" die Nennung des Datentypbezeichners (z.B. Klassenname) erforderlich.	☐	☐	☐
Zu Beginn eines javadoc-Kommentars einer Operation ist in wenigen Sätzen zu beschreiben, was die Operation macht.	☐	☐	☐
Sobald der Wert eines Parameters ein Java-Objekt sein kann, muss das Verhalten der Operation im Fall einer "null"-Belegung dieses Parameters angegeben werden.	☐	☐	☐
Der Vertrag einer Operation eines Java-Interfaces oder einer abstrakten Klasse muss spezifikationskonforme Unterschiede zwischen Implementierungen beschreiben.	☐	☐	☐
Aus dem javadoc-Kommentar zu einer Operation darf nicht hervorgehen, ob und welche Zustandsänderungen in Instanzen der implementierenden Klassen zu erwarten sind.	☐	☐	☐
Der javadoc-Kommentar einer Operation darf Vorgaben bzgl. der in Implementierungen zu verwendenden Algorithmen machen.	☐	☐	☐

8. Bitte positionieren Sie sich bzgl. der folgenden Aussagen:

"Die vollständige Kommentierung jeder Operation im eigenen Programm ist grundsätzlich und in jedem Fall sinnvoll."

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich **stimme nicht zu**									Ich **stimme zu**

"Die vollständige Kommentierung jeder public-Operation im eigenen Programm ist grundsätzlich und in jedem Fall sinnvoll."

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich **stimme nicht zu**									Ich **stimme zu**

"Die vollständige Kommentierung jeder Operation einer API des eigenen Programms ist grundsätzlich und in jedem Fall sinnvoll."

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich **stimme nicht zu**									Ich **stimme zu**

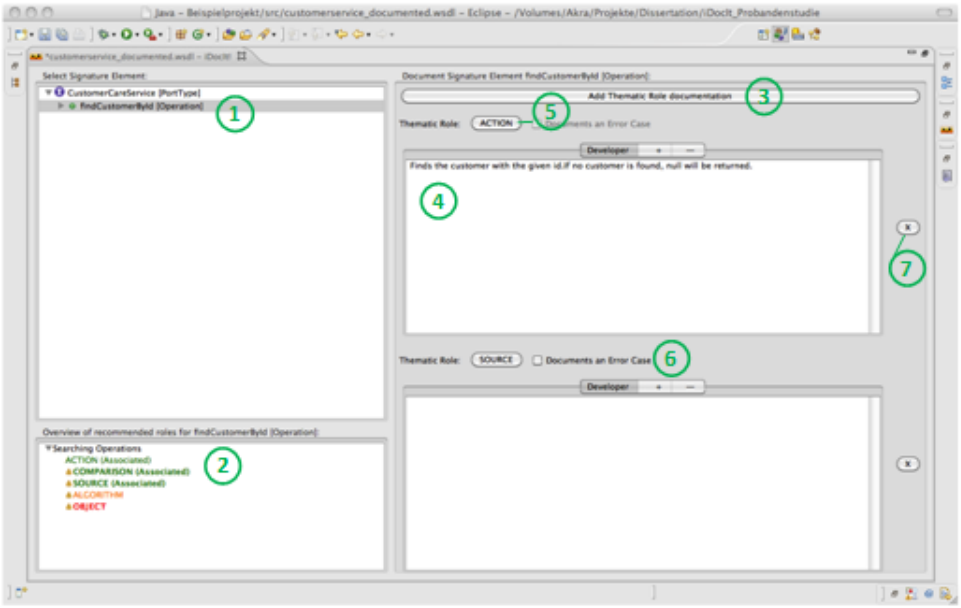
"Kommentare in Programmen sind grundsätzlich überflüssig."

	1	2	3	4	5	6	7	8	
	☐	☐	☐	☐	☐	☐	☐	☐	
Ich **stimme nicht zu**									Ich **stimme zu**

D.4 Kurzreferenzkarte zu *iDocIt!*

Quick Reference Card für iDocIt!

Grafische Oberfläche:



1. Darstellung der Schnittstelle als Baum. Hier können Methoden, Parameter, etc. zur weiteren Beschreibung selektiert werden.
2. Vorschläge für zu spezifizierende Inhalte.
3. Neue Beschreibung an links im Baum selektiertes Element anfügen.
4. Freitextfeld zur Beschreibung.
5. Auswahl der them. Rolle.
6. Wenn diese Checkbox gesetzt ist beschreibt der Text in 4) einen Fehlerfall.
7. Löscht die zugehörige Beschreibung.

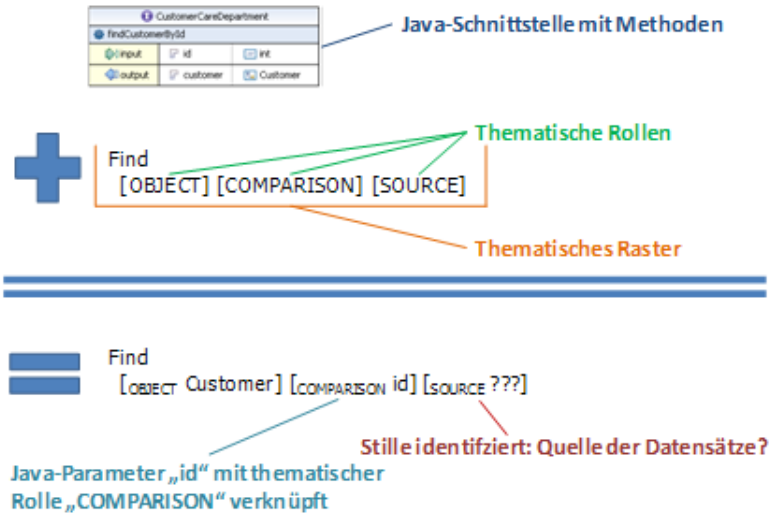
Farblegende zu (2):

Optionale Empfehlung | Bereits beschriebene Empfehlung | Nicht beschriebene, optionale Empfehlung
Pflichtempfehlung | Nicht beschriebene Pflichtempfehlung | Fehlerfall nicht dokumentiert

Bedeutung „Fehlerfall“:

Was passiert, wenn die Belegung der thematischen Rolle nicht verfügbar ist?

Prinzip:



Wichtige them. Raster (mit Verb):

- Löschende Operation (delete)
- Prüfende Operation (check)
- Erzeugende Operation (create)
- Konvertierende Operation (convert)
- Berechnende Operation (compute)
- Suchende Operation (find)
- Deponierende Operation (move, save)
- Sendende Operation (send, print)

Beispiele für die Belegung thematischer Rollen:

RULE: Geschäftsregel in WENN … DANN … (SONST …)-Form.
COMPARISON: Suchkriterium, z.B. Nachname.
SOURCE: Exportierendes System, Datenbank oder Datei.
DESTINATION: Analog zu SOURCE (im Sinne eines importierenden Systems).
PRIMARY_KEY: Eindeutiger fachlicher oder technischer Schlüssel (z.B. Fahrgestellnummer eines Autos).
IGNORABLE: Nicht betrachtetes Attribut.
FORMAT: Struktur einer Datei oder einer Nachricht (definiert durch eine Vorlage oder einen Standard).
INSTRUMENT: Programm oder Dienst, das oder der benutzt wird.
ORDERING: Sortierung von einer Datenliste.
LIMIT: Unter- oder Obergrenze der Anzahl von Elementen einer Datenliste.
ACTION: Zusammenfassende Beschreibung der Tätigkeit der Methode.

D.5 Kurzreferenzkarte zu *javadoc*

Quick Reference Card für javadoc

1 Beispiel Format eines Methoden-Kommentars

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name);
```

2 Verfügbare javadoc-Tags für Methodenkommentare (inkl. Reihenfolge)

Nr.	Tag	Beschreibung
1	@param *parametername*	Beschreibung des entsprechenden Parameters. Gemäß Konvention muss jeder Parameter beschrieben werden.
2	@return	Beschreibung des Methoden-Resultates. Gemäß Konvention muss das Resultat beschrieben werden, sofern vorhanden.
3	@throws *exception name*	Beschreibung der Ausnahme. Per Konvention müssen alle Ausnahmen beschrieben werden, die durch unerwartete Bedingungen außerhalb des Programms verursacht werden (ungültige Benutzereingaben, fehlende Ressourcen etc.).
4	@see	Verweis auf weiterführende Dokumente.
5	@since	Nennt die Version, seit der eine Methode oder ein Feature im Code existiert.
6	@deprecated	Markiert eine Methode als “überholt”. Diese Methode wird in einer zukünftigen Version des Programms möglicherweise entfernt und ist derzeit nur noch aus Kompatibilitätsgründen vorhanden.

3 Wichtige Konventionen für Methodenkommentare

- Methodenbeschreibungen starten mit einer Verbalphrase, z.B. “Gets the label of this button.” statt “This method gets the label of this button.”.
- Der erste Satz dient als Kurzzusammenfassung der Methodenbeschreibung. Danach folgt eine natürlichsprachliche Beschreibung der Methode.
- API-Spezfikationen sollten nach Möglichkeit von möglichen Implementierungen abstrahiert beschrieben werden.
- Wenn sinnvoll, sind gewollte Lücken der Spezifikation explizit zu benennen.

Anhang E

Musterlösung zur Evaluierung

Dieses Kapitel enthält die Musterlösungen der API-Verträge zur Aufgabenstellung aus der Evaluierung. Diese API-Verträge sind allesamt mit *iDocIt!* auf Basis von Java erstellt worden und beinhalten daher syntaktische Erweiterungen von *javadoc*.

Hinweise:

- Die thematische Rolle IGNORABLE drückt aus, dass ein übergebener Parameter von der jeweiligen Operation nicht verarbeitet wird.
- In den folgenden Verträgen werden die in der Aufgabenstellung eingeführten Kurzbezeichnungen für die beteiligten Systeme verwendet (siehe Anhangskapitel D.1).

E.1 Schnittstelle Stockservice (Abt. Lager) - Aktivität "Erzeuge Packauftrag"

```
1 /**
  * Erzeugt einen offiziellen Auftrag zum Packen der Bestellung
3 * im Lager.
  *
5 * @destination Der Packauftrag wird im Kommissionierungssystem
  * gespeichert und verwaltet.
7 *
  * @param order [ATTRIBUTE] Darf nicht null sein.
```

```
9   * @subparam articles [ATTRIBUTE] Darf nicht null sein.
    * @subparam customer [IGNORABLE]
11  * @subparam customer.deliveryAddress [IGNORABLE]
    * @subparam customer.id [IGNORABLE]
13  * @subparam customer.mailAddress [IGNORABLE]
    * @subparam id [PRIMARY KEY] Der Packauftrag bezieht sich auf
15  *  die Bestellung mit dieser Nummer.
    *
17  * @return [OBJECT]
    * @subreturn order [ATTRIBUTE]
19  * @subreturn order.articles [ATTRIBUTE]
    * @subreturn order.customer [ATTRIBUTE]
21  * @subreturn order.customer.deliveryAddress [ATTRIBUTE]
    * @subreturn order.customer.id [ATTRIBUTE]
23  * @subreturn order.customer.mailAddress [ATTRIBUTE]
    * @subreturn order.id [ATTRIBUTE]
25  *
    * @throws BooksellerException [ERROR ATTRIBUTE]
27  *  WENN die Bestellung null ist
    *  DANN Fehlercode: 01; Fehlernachricht: Bestellung ist
29  *      nicht vorhanden.
    * @throwsinfo BooksellerException [ERROR ATTRIBUTE]
31  *  WENN die Liste der Artikel null oder leer ist
    *  DANN Fehlercode: 22; Fehlernachricht: Bestellung enthält
33  *      keine Artikel.
    * @throwsinfo BooksellerException [ERROR DESTINATION]
35  *  WENN das Kommissionierungssystem nicht verfügbar ist
    *  DANN Fehlercode: 08; Fehlernachricht: Kommissionierungs-
37  *      system ist nicht verfügbar.
    * @thematicgrid Erzeugende Operation
39  */
  public PackageInstruction createPackagingInstructions(Order order)
41    throws BooksellerException;
```

E.2 Schnittstelle CustomerCareService (Abt. Kundenbetr.)

E.2.1 Aktivität "Lese Kundendatensatz anhand der Kundennummer"

```
1  /**
   * @source Durchsucht alle Kundendatensätze, die in der
3  *  Kundenstammdatenverwaltung gespeichert sind.
   *
5  * @param id [PRIMARY KEY] Muss > 0 und sechsstellig sein.
   *
7  * @return [OBJECT] Kundendatensatz für die übergebene
   *   Kundennummer.
9  *
   * @throws BooksellerException [ERROR OBJECT]
11 *  WENN kein Kundendatensatz für die übergebene Kunden-
   *      nummer vorhanden ist
13 *  DANN Fehlercode: 25; Fehlernachricht: Kein Datensatz zur
   *      Kundennummer vorhanden.
15 * @throwsinfo BooksellerException [ERROR SOURCE]
   *  WENN Kundenstammdatenverwaltung nicht verfügbar ist
17 *  DANN Fehlercode 03; Fehlernachricht: Kundenstammdaten-
   *      verwaltung ist nicht verfügbar oder antwortet nicht.
19 * @throwsinfo BooksellerException [ERROR PRIMARY KEY]
   *  WENN Kundennummer <= 0 ODER Kundennummer ist
21 *      nicht sechsstellig
   *  DANN Fehlercode: 23; Fehlernummer: Kundennummer ist
23 *      nicht im Format "######" (sechstellige Zeichenkette
   *      bestehend aus Ziffern).
25 * @thematicgrid Suchende Operation
   */
27 public Customer findCustomerById(int id)
       throws BooksellerException;
```

E.2.2 Aktivität "Erzeuge und versende Auftragsablehnung"

```
/**
 * Informiert den Kunden per E-Mail über die Ablehnung seiner
 * Bestellung.
 *
 * @format Der Inhalt der versendeten Mail ist im Vorlagen-
 *  verwaltungssystem für Dokumente unter dem Namen
 *   "CUSTOMER_ORDER_REJECTION_TMPL" gespeichert.
 * @instrument Die Mail wird über den SMTP-Server
 *  smtp.bookseller.de  versendet.
 * @message E-Mail zur Ablehnung der übergebenen Bestellung.
 *
 * @param order [ATTRIBUTE] Darf nicht null sein.
 * @subparam articles [IGNORABLE]
 * @subparam customer [ATTRIBUTE] Darf nicht null sein.
 * @subparam customer.deliveryAddress [ATTRIBUTE]
 *  Wird als Kundenanschrift im versendeten Dokument
 *  eingetragen.
 * @subparam customer.id [IGNORABLE]
 * @subparam customer.mailAddress [DESTINATION] Darf
 *  nicht null und muss eine gültige E-Mail-Adresse sein.
 *
 * @throws BooksellerException [ERROR ATTRIBUTE]
 *  WENN die übergebene Bestellung null ist
 *  Fehlercode: 01; Fehlernachricht: Bestellung nicht vorhanden.
 * @throwsinfo BooksellerException [ERROR DESTINATION]
 *  WENN der übergebene Kunde null oder dessen E-Mail-Adresse
 *      ungültig ist
 *  DANN Fehlercode: 12; Fehlernachricht: Kundenstammdaten
 *      nicht vorhanden.
 * @throwsinfo BooksellerException [ERROR FORMAT]
 *  WENN die Vorlage "CUSTOMER_ORDER_REJECTION_TMPL"
 *      nicht verfügbar ist
 *  DANN Fehlercode: 18; Fehlernachricht: Vorlage
```

```
34   *       "CUSTOMER_ORDER_REJECTION_TMPL" ist nicht
     *        verfügbar.
36   *   WENN das Vorlagenverwaltungssystem nicht verfügbar ist
     *   DANN Fehlercode: 02; Fehlernachricht: VoDo ist nicht
38   *       verfügbar.
     * @throwsinfo BooksellerException [ERROR INSTRUMENT]
40   *   WENN der SMTP-Server smtp.bookseller.de nicht verfügbar ist
     *   DANN Fehlercode: 17; Fehlernachricht: SMTP-Server ist nicht
42   *       verfügbar.
     *
44   * @thematicgrid Sendende Operation
     */
46 public void sendOrderRejection(Order order)
       throws BooksellerException;
```

E.3 Schnittstelle AccountingService (Abt. Rechnungswesen)

E.3.1 Aktivität “Prüfe Bonität anhand der Rechnungsanschrift”

```
1  /**
    * Rule: WENN der Kunde in einer "im Allgemeinen" als bonitär
3   *   eingestuften Straße wohnt, DANN ist der Kunde bonitär
    * SONST ist er nicht bonitär.
5   *
    * Mögliche Klassifikationen des externen Dienstleisters für
7   * eine Straße (nach Bonitätswahrscheinlichkeit aufsteigend
    *   sortiert):
9   * - "Sehr wahrscheinlich nicht bonitär"
    * - "Wahrscheinlich nicht bonitär"
11  * - "Im Allgemeinen bonitär"
    * - "Wahrscheinlich bonitär"
13  * - "Sehr wahrscheinlich bonitär"
    *
```

```
15  * @source Bonitätsklassifikationen für einzelne Straßen durch
    *  Bonitätsdienstleister.
17  *
    * @param customer [OBJECT]
19  * @subparam deliveryAddress [COMPARISON] Die externe
    *  Bonitätsauskunft beurteilt die Bonität des Kunden anhand
21  *   seiner Lieferanschrift.
    * @subparam id [PRIMARY KEY]
23  *
    * @return [REPORT] TRUE: Kunde ist bonitär.
25  * FALSE: Kunde ist nicht bonitär.
    *
27  * @throws SolvencyCheckException [ERROR SOURCE]
    *  WENN die Bonitätsauskunft nicht verfügbar ist
29  *  DANN Fehlercode: 06; Fehlernachricht: Bonitätsauskunft ist nicht
    *      verfügbar.
31  * @throwsinfo SolvencyCheckException [ERROR OBJECT]
    *  WENN die übergebenen Kundenstammdaten null sind
33  *  DANN Fehlercode: 12; Fehlernachricht: Kundenstammdaten
    *      liegen nicht vor.
35  * @throwsinfo SolvencyCheckException [ERROR PRIMARY KEY]
    *  WENN die übergebene Kundennummer nicht existiert
37  *  DANN Fehlercode: 12; Fehlernachricht: Kundenstammdaten
    *      liegen nicht vor.
39  *
    * @thematicgrid Prüfende Operation
41  */
public boolean checkSolvencyByAddress(Customer customer)
43     throws SolvencyCheckException;
```

E.3.2 Aktivität “Prüfe Bonität anhand des Zahlungsverhaltens”

```
1  /**
    * Rule: WENN die letzten 3 Rechnungen pünktlich bezahlt
3   * worden sind, DANN ist der Kunde bonitär,
```

```
     * SONST ist der Kunde nicht bonitär.
     *
     * @source Die letzten drei Rechnungen mit Fälligkeitsdatum
     *  und das Eingangsdatum der Zahlung sind im
     *  Finanzbuchhaltungssystem abgelegt.
     *
     * @param customer [OBJECT]
     * @subparam id [PRIMARY KEY]
     *
     * @return [REPORT]  TRUE: Kunde ist bonitär.
     *  FALSE: Kunde ist nicht bonitär.
     *
     * @throws SolvencyCheckException [ERROR SOURCE]
     *  WENN das FiBu-System nicht verfügbar ist
     *  DANN Fehlercode 11; Fehlernachricht: FiBu-System ist nicht
     *      verfügbar.
     * @throwsinfo SolvencyCheckException [ERROR OBJECT]
     *  WENN die übergebenen Kundenstammdaten null sind
     *  DANN Fehlercode 12; Fehlernachricht: Kundenstammdaten
     *      liegen nicht vor.
     * @throwsinfo SolvencyCheckException [ERROR PRIMARY KEY]
     *  WENN die übergebene Kundennummer nicht existiert
     *  DANN Fehlercode: 12; Fehlernachricht: Kundenstammdaten
     *      liegen nicht vor.
     * @thematicgrid Prüfende Operation
     */
    public boolean checkSolvencyByPayments(Customer customer)
        throws SolvencyCheckException;
```

E.3.3 Aktivität “Erstelle Rechnungsdatensatz”

```
/**
 * Erstellt einen neuen Rechnungsdatensatz für die gegebene
 * Bestellung gegen den gegebenen Kunden im Finanzbuch-
 * haltungssystem. Des Weiteren wird eine neue Forderung erfasst.
```

```
5   * Das Fälligkeitsdatum der Rechnung liegt 14 Tage nach dem
    * Erstellungsdatum. Das aktuelle Datum soll vom dem NTP-
7   * Zeitserver time.bookseller.de abgerufen werden.
    *
9   * @FORMULA Berechnung des Rechnungsbetrages:
    * Rechnungsbetrag =
11  *  SUMME(Artikelmenge * Artikelpreis) * (1 - Rabattsatz)
    *
13  * @object Rechnungsdatensatz und Forderung
    * @destination Finanzbuchhaltungssystem
15  *
    * @param customer
17  * @subparam id [ATTRIBUTE] Adressat der Rechnung
    * @subparam mailAddress [IGNORABLE]
19  * @param order [ATTRIBUTE] Die erzeugte Rechnung enthält
    *   die Positionen dieser Bestellung.
21  * @paraminfo order [PRIMARY KEY] Zwischen Rechnung
    *   und Bestellung besteht eine 1:1-Beziehung, daher kann die
23  *   erstellte Rechnung über die Bestellnummer aufgerufen werden.
    * @param discount [ATTRIBUTE] Gewährter Rabattsatz für den
25  *  übergebenen Kunden in Prozent (Wert liegt zwischen 0.00
    *  und 1.00)
27  *
    * @throws BooksellerException [ERROR DESTINATION]
29  *  Fehlercode 11; Fehlernachricht: FiBu-System ist
    *  nicht verfügbar.
31  * @throwsinfo BooksellerException [ERROR ATTRIBUTE]
    *  WENN customer null oder customer.id ungültig ist
33  *  DANN Fehler 12; Fehlernachricht: Kundenstammdaten liegen
    *      nicht vor.
35  *  WENN order null ist
    *  DANN Fehlercode: 01; Fehlernachricht: Bestellung ist
37  *      nicht vorhanden.
    *  WENN order.orderedArticles null oder leer ist
```

```
39   *  DANN Fehlercode: 22; Fehlernachricht: Bestellung enthält
     *      keine Artikel.
41   *  WENN das Systemdatum nicht verfügbar ist
     *  DANN Fehlercode 26; Fehlernachricht: Der Zeitserver ist
43   *      nicht verfügbar.
     *  WENN der Rabattsatz ungültig ist
45   *  DANN Fehlercode 27; Fehlernachricht: Rabattsatz liegt
     *      nicht vor.
47   * @throwsinfo BooksellerException [ERROR PRIMARY KEY]
     *  WENN die übergebene Bestellung null ist
49   *  DANN Fehlercode: 01; Fehlernachricht: Bestellung ist nicht
     *      vorhanden.
51   *
     * @thematicgrid Erzeugende Operation
53   */
  public void createInvoice(Customer customer, Order order,
55     double discount) throws BooksellerException;
```

E.3.4 Aktivität “Drucke Rechnung”

```
1  /**
     *  Druckt die übergebene Bestellung auf dem Drucker
3    *  print.bookseller.de.
     *
5    * @destination Drucker "print.bookseller.de".
     * @message Das gedruckte Dokument ist die Rechnung
7    *  zur übergebenen Bestellung. Die Rechnungsdaten
     *  müssen im FiBu-System gespeichert sein.
9    * @format Das Layout der Rechnung wird durch die Vorlage
     *  "INVOICE_TMPL" definiert, welche im VoDo-System
11   *  gespeichert ist. Die Druckanweisungen erfolgen im
     *  PostScript-Format.
13   * @instrument Bookseller Postscript Drucker Treiber.
     *
15   * @param order [ATTRIBUTE] Darf nicht null sein.
```

```
 * @subparam articles [ATTRIBUTE] Die in Rechnung gestellten
17  *  Artikel.
 * @subparam customer [ATTRIBUTE] Rechnungsempfänger
19  * @subparam customer.deliveryAddress [ATTRIBUTE] Wohin
 *  die Rechnung geschickt wird.
21  * @subparam customer.id [IGNORABLE]
 * @subparam customer.mailAddress [IGNORABLE]
23  * @subparam id [COMPARISON]
 *
25  * @throws BooksellerException [ERROR DESTINATION]
 *  WENN der Drucker "print.bookseller.de" nicht erreichbar ist
27  *  DANN Fehlercde: 21; Fehlernachricht: Drucker
 *  "print.bookseller.de" ist nicht verfügbar.
29  * @throwsinfo BooksellerException [ERROR INSTRUMENT]
 *  WENN Bookseller Postscript Drucker Treiber nicht verfügbar ist
31  *  DANN Fehlercode: 30; Fehlernachricht: Bookseller Postscript
 *  Drucker Treiber ist nicht verfügbar.
33  * @throwsinfo BooksellerException [ERROR FORMAT]
 *  WENN die Vorlage "INVOICE_TMP" nicht verfügbar ist
35  *  DANN Fehlercode: 09; Fehlernachricht: Vorlage
 *  "INVOICE_TMPL" ist nicht verfügbar.
37  * @throwsinfo BooksellerException [ERROR MESSAGE]
 *  WENN FiBu-System nicht verfügbar ist
39  *  DANN Fehlercode: 11; Fehlernachricht: FiBu-System ist nicht
 *  verfügbar.
41  *  WENN die Rechnungsdaten nicht im FiBu-System vorhanden
 *      sind
43  *  DANN Fehlercode: 20; Fehlernachricht: Rechnungsdaten sind
 *      nicht im FiBu-System gespeichert.
45  * @throwsinfo BooksellerException [ERROR COMPARISON]
 *  WENN order.id nicht im FiBu-System gefunden werden kann
47  *  DANN Fehlercode: 01; Fehlernachricht: Bestellung existert
 *      nicht
49  * @throwsinfo BooksellerException [ERROR ATTRIBUTE]
```

```
     *  WENN die Liste der bestellten Artikel null oder leer ist
51   *  DANN Fehlercode: 22; Fehlernachricht: Bestellung ist leer.
     *
53   * @thematicgrid Sendende Operation
     */
55 public void printInvoice(Order order)
       throws BooksellerException;
```

E.3.5 Aktivität "Speichere Bestellung"

```
   /**
2   * Speichert die übergebene Bestellung in der Auftragsverwaltung.
     *
4   * @destination Auftragsverwaltung
     *
6   * @param order [OBJECT] Darf nicht null sein.
     * @subparam articles [ATTRIBUTE] Muss mind. einen Artikel
8   *  beinhalten.
     *
10  * @return [REPORT] TRUE: Bestellung ist gespeichert worden.
     *  FALSE: Bestellung ist nicht gespeichert worden.
12  *
     * @throws BooksellerException [ERROR OBJECT]
14  *  WENN die übergebene Bestellung null ist
     *  DANN Fehlercode: 01, Fehlernachricht: Bestellung ist nicht
16  *      vorhanden.
     *  WENN die übergebene Bestellung keine Artikel enthält:
18  *  DANN Fehlercode: 22; Fehlernachricht: Bestellung
     *      enthält keine Artikel.
20  * @throwsinfo BooksellerException [ERROR DESTINATION]
     *  WENN die Auftragsverwaltung nicht verfügbar ist:
22  *  DANN Fehlercode: 29; Fehlernachricht: Auftragsverwaltung
     *      ist nicht verfügbar.
24  *
     * @thematicgrid Deponierende Operation
```

```
26   */
  public boolean saveOrder(Order order)
28     throws BooksellerException;
  }
```

E.3.6 Aktivität "Berechne Rabatt"

```
1  /**
    * @rule Berechnung des Rabattsatzes:
3   * WENN Kundenumsatz in den letzen 12 Monaten >= 500,00EUR DANN 10%
    * WENN Kundenumsatz in den letzten 12 Monaten >= 250,00EUR DANN 5%
5   * WENN Kundenumsatz in den letzten 12 Monaten >= 100,00EUR DANN 2%
    * SONST 0%
7   *
    * @operand Rechnungsdatensätze der vergangenen 12 Monate.
9   *
    * @param customer [OPERAND] Darf nicht null sein.
11  * @subparam deliveryAddress [IGNORABLE]
    * @subparam id [OPERAND] Der Umsatz des vergangenen Jahres wird
13  *   anhand dieser Kundennummer bestimmt.
    * @subparam mailAddress [IGNORABLE]
15  *
    * @return [OBJECT] Rabattsatz für den Kunden in % (zwischen 0.00
17  *   und 1.00).
    *
19  * @throws BooksellerException [ERROR OPERAND]
    *  WENN customer null ist
21  *  DANN Fehlercode: 12; Fehlernachricht: Kundenstammdaten liegen
    *       nicht vor.
23  *  WENN customer.id ungütig ist
    *  DANN Fehlercode: 12; Fehlernachricht: Kundenstammdaten liegen
25  *       nicht vor.
    *  WENN Finanzbuchhaltungssystem nicht verfügbar ist
27  *       DANN Fehlercode: 11; Fehlernachricht: FiBu-System
    *       ist nicht verfügbar.
```

```
29   *  WENN Rechnungsdatensätze des Kunden nicht verfügbar
     *  DANN Fehlercode: 25; Fehlernachricht: Kundenumsatz
31   *       der letzten 12 Monate ist nicht zugreifbar.
     *
33   * @thematicgrid Berechnende Operation
     */
35 public double computeDiscount(Customer customer)
       throws BooksellerException;
```

Literaturverzeichnis

Xiaoying Bai, Wenli Dong, Wei-Tek Tek Tsai und Chen Yinong. Wsdl-based automatic test case generation for web services testing. In: (ohne Herausgeber) *Proceedings - IEEE International Workshop on Service-Oriented System Engineering - SOSE 2005*, Seiten 207 – 212, Los Alamitos, Washington, Tokyo, Oktober 2005. IEEE Computer Society Press.

Collin F. Baker, Charles J. Fillmore und John B. Lowe. The Berkeley FrameNet Project. In: Christian Boitet und Pete Whitelock (Hrsg.), *COLING '98: Proceedings of the 17th international conference on Computational linguistics*, Band 1 aus *ACL '98*, Seiten 86 – 90. Association for Computational Linguistics, 1998.

Paul Baltes. *Längsschnitt- und Querschnittsequenzen zur Erfassung von Alters- und Generationseffekten.* Dissertation, Universität des Saarlandes, 1967.

Helmut Balzert. *Lehrbuch der Softwaretechnik - Entwurf, Implementierung, Installation und Betrieb.* Spektrum Akademischer Verlag, Heidelberg, 3. Auflage, 2011.

James Bean. *SOA and Web Services Interface Design - Principles, Techniques and Standards.* Morgan Kaufmann, Amsterdam u.a., 2010.

Joshua Bloch. *Effective Java.* Addison-Wesley Verlag, Upper Saddle River u.a., 2. Auflage, 2008.

Barry W. Boehm. *Software Engineering Economics.* Prentice Hall, Eaglewood Cliffs, 1981.

Andreas Borchardt und Stephan E. Göthlich. Erkenntnisgewinnung durch Fallstudien. In: Stefan Kühl, Petra Strodtholz und Andreas Taffertshofer (Hrsg.), *Handbuch Methoden der Organisationsforschung*, Seiten 33 – 48. Springer, 2009.

Jürgen Bortz und Nicola Döring. *Forschungsmethoden und Evaluation: für Human- und Sozialwissenschaftler*. Springer Medizin Verlag, Berlin, Heidelberg, 4. überarbeitete Auflage, 2006.

Elizabeth Bowman. *The Minor and Fragmentary Sentences of a Corpus of Spoken English*, Band 32 aus *International Journal of American Linguistics - Part II (Ausgabe 3)*. Mouton & Co., Den Haag, 1966.

Ruven Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, Volume 18(Number 6): Seiten 543 – 554, 1983.

Frederick Phillips Brooks Jr. *The Mythical Man-Month*. Addison-Wesley Verlag, Reading u.a., 1995.

Oliver Burn, Lars Kühne, Rick Giles, Oleg Sukhodolsky, Michael Studman und Travis Schneeberger. Checkstyle, (ohne Jahr). Dieses Programm ist unter `http://checkstyle.sourceforge.net` verfügbar (letzter Abruf am 20.04.2013).

Raymond P.L. Buse und Westley R. Weimer. Automatic documentation inference for exceptions. In: (ohne Herausgeber) *ISSTA '08: Proceedings of the 2008 International Symposium on Software Testing and Analysis*, Seiten 273 – 281, New York, 2008. Association for Computing Machinery.

Andrew Carnie. *Syntax: A Generative Introduction*. John Wiley & Sons Inc., Malden u.a., 3. Auflage, 2013.

John Cheesman und John Daniels. *UML Components - A Simple Process for Specifying Component-Based Software*. Addison-Wesley Verlag, Bonn u.a., 2001.

Noam Chomsky. *Lectures on Government and Binding*. Foris Publications, Dordrecht, 4. Auflage, 1986.

Larry B. Christensen. *Experimental Methodology*. Allyn and Bacon, Boston u.a., 2. Auflage, 1980.

Aaron Clauset, Cosma Rohilla Shalizi und M.E.J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4): Seiten 661 – 703, 2009. Dieser Artikel ist verfügbar unter `http://epubs.siam.org/doi/abs/10.1137/070710111` (letzter Abruf am 05.06.2009).

Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord und Judith Stafford. *Documenting Software Architectures - Views and Beyond*. Addison-Wesley Verlag, Boston u.a., 2003.

CloudFare Inc. CloudFlare Hosting Provider API Version 1.1.1, 2012. Diese Spezifikation ist unter `http://www.cloudflare.com/docs/host-api.html` verfügbar (letzter Abruf am 15.04.2013).

Alistair Cockburn. Using goal-based use cases. *Journal of Object-Oriented Programming*, Band 10(Ausgabe 7): Seiten 56 – 62, 1997.

Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON), 2006. Dieser Request for Comments (RFC) ist unter der Nummer 4627 verfügbar unter `http://tools.ietf.org/html/rfc4627` (letzter Abruf am 22.04.2013).

María del Rosario Girardi Gutiérrez und Bertrand Ibrahim. Using english to retrieve software. *Journal of Systems and Software*, Band 30(Ausgabe 2): Seiten 249 – 270, September 1995.

Jeremy Dick und Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In: James C. P. Woodcock und Peter G. Larsen (Hrsg.), *FME '93: Industrial-Strength Formal Methods*, Band 670 aus *Lecture Notes in Computer Science*, Seiten 268 – 284. Springer Verlag, 1993.

David R. Dowty. On the semantic content of the notion of 'thematic role'. In: Gennaro Chierchia, Barbara H. Partee und Raymond Turner (Hrsg.), *Properties, Types and Meaning - Volume II: Semantic Issues*, Seiten 69 – 129. Kluwer Acadamic Publishers, Dordrecht, 1989.

Die Dudenredaktion (Hrsg.). *Deutsches Universalwörterbuch Duden*. Bibliographisches Insitut und F.A. Brockhaus, Mannheim, 6. Auflage, 2009.

Charles J. Fillmore. The case for case. In: Emmon Bach und Robert T. Harms (Hrsg.), *Universals in Linguistic Theory*, Seiten 1 – 88. Holt, Rinehart and Winston, New York u.a., 1968.

Andrew Forward und Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In: (ohne Herausgeber) *DocEng '02: Proceedings of*

the 2002 ACM symposium on Document engineering, Seiten 26 – 33, New York, 2002. Association for Computing Machinery.

Kilian Foth. Eine umfassende Constraint-Dependenz-Grammatik des Deutschen, 2004. Dieses Handbuch ist verfügbar unter `http://nats-www.informatik.uni-hamburg.de/pub/Papa/CdgManuals/deutsch.pdf` (letzter Abruf am 29.08.2013).

Michel Foucault. *Die Ordnung der Dinge (dt. Übersetzung)*. Suhrkamp, Frankfurt am Main, 1971.

Lisa Friendly. The design of distributed hyperlinked programming documentation. In: (ohne Herausgeber) *Hypermedia Design - Proceedings of the International Workshop on Hypermedia Design (IWHD'95), Montpellier, France, 1–2 June 1995*, Seiten 151 – 173, London, 1995. Springer Verlag.

Peter Fröhlich und Johannes Link. Automated test case generation from dynamic models. In: (ohne Herausgeber) *ECOOP '00: Proceedings of the 14th European Conference on Object-Oriented Programming*, Seiten 472 – 492, London, 2000. Springer Verlag.

Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software (dt. Übersetzung)*. Addison-Wesley Verlag, München u.a., 2004.

Tom Gelhausen. *Modellextraktion aus natürlichen Sprachen - Eine Methode zur systematischen Erstellung von Domänenmodellen*. Dissertation, Karlsruher Institut für Technologie, Karlsruhe, 2010. Diese Dissertation ist unter `http://www.ipd.uka.de/Tichy/publications.php?id=244` verfügbar (letzter Abruf am 23.05.2013).

Edward M. Gellenbeck und Curtis R. Cook. An investigation of procedure and variable names as beacons during program comprehension. In: Jürgen Koenemann-Belliveau, Thomas G. Moher und Scott P. Robertson (Hrsg.), *Empirical Studies of Programmers: Fourth Workshop*, Seiten 65 – 81. Ablex Publishing Corporation, Norwood, 1991.

S. Gnesi, D. Latella und M. Massink. Formal test-case generation for uml statecharts. In: (ohne Herausgeber) *Proceedings of the 9th IEEE International Conference on Engineering Complex Computer Systems*, Seiten 75 – 84. IEEE Computer Society, 2004.

Adele Goldberg und David Robson. *Smalltalk-80 - The Language and its Implementation.* Addison-Wesley Verlag, Reading u.a., 1983.

Ernest Greenwood. Das Experiment in der Soziologie. In: König (1972), Seiten 171 – 220.

Erwin Grochla, Heiko Lippold und Jörg Breithardt. *Prüflisten zur Schwachstellenermittlung in Büro und Verwaltung.* FBO Maschinenbau Verlag, Baden-Baden, Frankfurt am Main, 1986.

Jeffrey Steven Gruber. *Studies in Lexical Relations.* Dissertation, Massachusetts Institute of Technology, 1962. Diese Dissertation ist unter `http://dspace.mit.edu/handle/1721.1/13010` verfügbar (letzter Abruf am 20.08.2013).

Richard Helm, Ian M. Holland und Dipayan Gangopadhyay. Contracts: Specifying behavioral compositions in object-oriented systems. In: (ohne Herausgeber) *OOPSLA/E-COOP '90: Proceedings of the European Conference on Object-Oriented Programming Systems, Languages and Applications*, Seiten 169–180, New York, 1990. Association for Computing Machinery.

Gregor Hohpe und Bobby Woolf. *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Verlag, Boston u.a., 2008.

Adele E. Howe, Anneliese von Mayrhauser und Richard T. Mraz. Test Case Generation as an AI Planning Problem. *Automated Software Engineering*, Volume 4(Number 1): Seiten 77 –106, 1997.

Andrew Hunt und Dave Thomas. *Der pragmatische Programmierer (dt. Übersetzung).* Carl Hanser Verlag, München, Wien, 2003.

IBM Corp. und andere. Eclipse Documentation - Plug-in Development Environment Guide, ohne Jahr. Diese Internetseite ist unter `http://help.eclipse.org/juno/index.jsp` > Plug-in Development Environment Guide verfügbar (letzter Abruf am 15.04.2013).

Marie Jahoda, Morton Deutsch und Stuart W. Cook. Beobachtungsverfahren. In: König (1972), Seiten 77 – 96.

G. Keller, M. Nüttgens und A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). In: A.-W. Scheer (Hrsg.), *Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes*,

Band 89, Seiten 1 – 30. Universität des Saarlandes, Saarbrücken, 1992. Diese Publikation ist unter http://www.uni-saarland.de/fileadmin/user_upload/Fachrichtungen/fr13_BWL/professuren/PDF/heft89.pdf verfügbar (letzter Abruf: 21.08.2013).

Fred N. Kerlinger. *Grundlagen der Sozialwissenschaften (Band 2)*. Beltz Verlag, Weinheim und Basel, 1979.

Y.G. Kim, H.S. Hong, D.-H. Bae und S.D. Cha. Test cases generation from uml state diagrams. *IEE Proceedings Software*, Band 146(Ausgabe 4): Seiten 187 – 192, 1999.

Karin Kipper-Schuler. *VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon*. Dissertation, University of Pennsylvania, 2005. Diese Dissertation ist unter http://verbs.colorado.edu/~kipper/Papers/dissertation.pdf verfügbar (letzter Abruf am 23.05.2013).

Donald Ervin Knuth. *Literate Programming*. Band 27 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, 1992.

René König (Hrsg.). *Beobachtung und Experiment in der Forschung*, Köln, 1972. Kiepenheuer & Witsch.

Anna Korhonnen und Ted Briscoe. Extended lexical-semantic classification of english verbs. In: Dan Moldovan und Roxana Girju (Hrsg.), *Proceedings of the HLT-NAACL Workshop on Computational Lexical Semantics*, CLS '04, Seiten 38 – 45, Stroudsburg, 2004. Association for Computational Linguistics.

Bernd Körner. Prozessverben analysieren: Hin zu vollständigen Anforderungen und Modellen. *Objektspektrum*, Jahrgang 2009(Ausgabe 6): Seiten 60 – 67, 2009.

Douglas Kramer. API Documentation from Source Code Comments: a Case Study of Javadoc. In: (ohne Herausgeber) *SIGDOC '99: Proceedings of the 17th annual international conference on Computer documentation*, Seiten 147 – 153, New York, 1999. Association for Computing Machinery.

Jan Christian Krause. Using Thematic Grids to Document Web Service Operations. In: M.A. Karim Sadiq, Dan Tamir und Peraphon Sophatsathit (Hrsg.), *International Conference on Software Engineering Theory and Practice*, Seiten 163 – 170, Red Hook, 2010. International Society for Research in Sciency and Technology (ISRST).

Jan Christian Krause. Stille und Rauschen in Dokumentation - Ein Werkzeug zur Vervollständigung natürlichsprachlicher API-Dokumentation. *Objektspektrum*, Jahrgang 2011 (Ausgabe 3): Seiten 14 – 19, 2011.

Wilfried Kürschner. *Grammatisches Kompendium.* Narr Francke Attempto Verlag, Tübingen, 6. Auflage, 2008.

Wilfried Laatz. *Empirische Methoden - Ein Lehrbuch für Sozialwissenschaftler.* Verlag Harri Deutsch, Thun und Frankfurt am Main, 1993.

Beth Levin. *English Verb Classes and Alternations - A Preliminary Investigation.* The University of Chicago Press, Chicago und London, 1993.

Barbara H. Liskov und Jeanette M. Wing. A Behavioral Notion of Subtyping. *ACM Transaction on Programming Languages and Systems*, Band 16(Ausgabe 6): Seiten 1811 – 1841, 1994.

Sebastian Löbner. *Understanding Semantics.* Arnold, London, 2002.

Kenneth C. Louden. *Programming Languages - Principles and Practice.* Thomson Brooks / Cole, Pacific Grove u.a., 2. Auflage, 2003.

Thomas W. Malone und George A. Herman. What Is in the Process Handbook. In: Malone et al. (2003a), Seiten 221 – 259.

Thomas W. Malone, Kevin Crowston und George A. Herman (Hrsg.). *Organizing Business Knowledge*, Cambridge und London, 2003a. MIT Press.

Thomas W. Malone, Kevin Crowston, Jintae Lee, Brian T. Pentland, Chrysantos Dellarocas, Gerge M. Wyner, John Quimby, Abraham Bernstein, George A. Herman, Mark Klein, Charles S. Osborn und Elisa O'Donnell. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. In: Malone et al. (2003a), Seiten 13 – 37.

Mitchell P. Marcus, Beatrice Santorini und Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, Jahrgang 19(Ausgabe 2): Seiten 313 – 330, 1993.

Robert Cecil Martin. *Agile Software Development - Principles, Patterns and Practices.* Prentice Hall, Upper Saddle River, 2003.

Dirk Meier-Eickhoff. iDocIt!: Werkzeug zur einheitlichen Dokumentation für verschiedene Schnittstellenspezifikationen. Bachelor-Thesis, Fachhochschule Wedel, 2011. Diese Bachelor-Thesis ist unter `https://code.google.com/a/eclipselabs.org/p/idocit/downloads/list` verfügbar (letzter Abruf am 01.06.2013). Sie ist ferner vom Autoren dieser Dissertation betreut worden.

Jan Mendling und Jan Recker. Towards Systematic Usage of Labels and Icons in Business Process Models. In: Terry Halpin, Erik Proper, John Krogstie, Xavier Franch, Ela Hunt und Remi Coletta (Hrsg.), *Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'08)*, Band 337 aus *CEUR Workshop Proceedings*, Seiten 1 – 13. CEUR-WS.org, 2008. Diese Publikation ist unter `http://ceur-ws.org/Vol-337/` verfügbar (letzter Abruf am 20.07.2013).

Jan Mendling und Hajo Reijers. The Impact of Activity Labeling Styles on Process Model Quality. In: Wolfgang Hesse und Andreas Oberweis (Hrsg.), *SIGSAND-EUROPE 2008: Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems*, Band P-129 aus *Lecture Notes in Computer Science - Proceedings*, Seiten 117 – 127, Bonn, 2008. Gesellschaft für Informatik.

Bertrand Meyer. On Formalism in Specifications. *IEEE Software*, Band 2(Ausgabe 1): Seiten 6 – 26, 1985.

Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, New York u.a., 1988.

Bertrand Meyer. Design by Contract. In: Dino Mandrioli und Bertrand Meyer (Hrsg.), *Advances in Object-Oriented Software Engineering*, Seiten 1 – 50. Prentice Hall, New York u.a., 1992.

Microsoft Corp. Design Guidelines for Class Library Developers, ohne Jahr. Diese Programmierrichtlinien sind unter `http://msdn.microsoft.com/en-us/library/czefa0ke(v=vs.71).aspx` verfügbar (letzter Abruf 02.09.2013).

George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, Band 38(Ausgabe 11): Seiten 39 – 41, 1995.

Stefan Müller. *Head-Driven Phrase Structure Grammar: Eine Einführung*. Band 17 in Stauffenburg Einführung. Stauffenburg Verlag, Tübingen, 2. Auflage, 2008.

Glenford J. Myers, Tom Badgett und Corey Sandler. *The Art of Software Testing.* John Wiley & Sons Inc., Hoboken, 3. Auflage, 2012.

E. Nurvitadhi, Wing Wah Leung und C. Cook. Do Class Comments Aid Java Program Understanding? In: (ohne Herausgeber) *Frontiers in Education, 2003. FIE 2003 33rd Annual*, Band 1, Seiten T3C–13 – T3C–17. IEEE Computer Society, 2003.

Jeff Offutt und Aynur Abdurazik. Generating Tests from UML Specifications. In: Robert France und Bernhard Rumpe (Hrsg.), *UML '99 – The Unified Modeling Language*, Band 1723 aus *Lecture Notes in Computer Science*, Seiten 416 – 429. Springer Verlag, Berlin, Heidelberg, 1999.

Ásgeir Ólafsson und Doug Bryan. On the Need for "Required Interfaces" of Components. In: Max Mühlhäuser (Hrsg.), *Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP'96*, Seiten 159 – 165, Heidelberg, 1996. dpunkt Verlag.

Oracle Corp. Code Conventions for the Java Programming Language, 1999. Diese Programmierrichtlinien sind unter `http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html` verfügbar (letzter Abruf 02.09.2013).

Oracle Corp. Java Platform, Standard Edition 7 - API Specification, 2013a. Diese Spezifikation ist unter `http://docs.oracle.com/javase/7/docs/api/` verfügbar (letzter Abruf am 13.03.2013).

Oracle Corp. Java Mail API Documentation, 2013b. Diese Spezifikation ist unter `http://javamail.java.net/nonav/docs/api/` verfügbar (letzter Abruf am 11.09.2013).

Oracle Corp. How to Write Doc Comments for the Javadoc Tool, ohne Jahr. Dieser Artikel ist unter `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html` verfügbar (letzter Abruf am 15.03.2013).

Catherine Oriat. Jartege: A Tool for Random Generation of Unit Tests for Java Classes. In: Ralf Reussner, Johannes Mayer, Judith Stafford, Sven Overhage, Steffen Becker und Patrick J. Schroeder (Hrsg.), *Quality of Software Architectures and Software Quality*, Band 3712 aus *Lecture Notes in Computer Science*, Seiten 242 – 256. Springer Verlag, Berlin, Heidelberg, 2005.

Tim Ottinger. Meaningful Names. In: Robert Cecil Martin (Hrsg.), *Clean Code - A Handbook of Agile Software-Craftsmanship*, Seiten 17 – 30. Prentice Hall, Upper Saddle River, 2009.

David Lorge Parnas. Information distribution aspects of design methodology. In: C.V. Freiman, J.E. Griffith und J.L. Rosenfeld (Hrsg.), *Information Processing 71 - Proceedings of the IFIP Congress 71*, Band 1, Seiten 339 – 344, Amsterdam, London, 1971. North Holland Publishing Company.

David Lorge Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, Band 15(Ausgabe 12): Seiten 1053 – 1058, 1972.

Lionel Pilorget. *Testen von Informationssystemen: Integriertes und prozessorientiertes Testen.* Vieweg + Teubner Verlag, Wiesbaden, 2012.

Ariel Rabkin und Randy Katz. Static Extraction of Program Configuration Options. In: (ohne Herausgeber) *ICSE'11: Proceedings of the 33rd International Conference on Software Engineering (2011)*, Seiten 131 – 140, New York, 2011. Association for Computing Machinery.

Oliver Rack und Timo Christophersen. Experimente. In: Sönke Albers, Daniel Klapper, Udo Konradt, Achim Walter und Joachim Wolf (Hrsg.), *Methodik der empirischen Forschung*, Seiten 17 – 32. Gabler Verlag, 2. Auflage, 2007.

Martin P. Robillard. What makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, Band 26(Ausgabe 6): Seiten 27 – 34, 2009.

Colette Rolland und Camille Ben Achour. Guiding the construction of textual use case specifications. *Data & Knowledge Engineering*, Band 25(Ausgabe 1 - 2): Seiten 125 – 160, 1998.

Viktor Sarris. *Methodologische Grundlagen der Experimentalpsychologie - Band 1: Erkenntnisgewinnung und Methodik.* Ernst Reinhardt Verlag, München, 1990.

August-Willhelm Scheer. *Architektur integrierter Informationssysteme.* Springer Verlag, Berlin u.a., 1991.

Rainer Schnell, Paul Bernhard Hill und Elke Esser. *Methoden der empirischen Sozialforschung.* Oldenbourg Verlag, 9. aktualisierte Auflage, 2011.

Daniel Schreck, Valentin Dallmeier und Thomas Zimmermann. How Documentation Evolves Over Time. In: (ohne Herausgeber) *IWPSE '07: 9th international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*, Seiten 4 – 10, New York, 2007. Association for Computing Machinery.

Kevin A. Smith und Douglas Kramer. Requirements for Writing Java API Specifications, 2003. Dieses Dokument ist verfügbar unter `http://www.oracle.com/technetwork/java/javase/documentation/index-142372.html` (letzter Abruf am: 12.09.2013).

Jesse Snedeker und Lila Gleitmann. Why Is It So Hard to Label Our Concepts? In: D. Geoffrey Hall und Sandra R.Waxman (Hrsg.), *Weaving a Lexicon*, Seiten 257 – 293. MIT Press, Cambridge, 2004.

Alan Snyder. Modeling the C++ object model. In: Pierre America (Hrsg.), *ECOOP'91 European Conference on Object-Oriented Programming*, Band 512 aus *Lecture Notes in Computer Science*, Seiten 1–20. Springer Verlag, Berlin, Heidelberg, 1991.

Giriprasad Sridhara, Lori Pollock und K. Vijay-Shanker. Automatically Detecting and Describing High Level Actions within Methods. In: Richard Newton Taylor, Harald Gall und Nenad Medvidovic (Hrsg.), *ICSE'11: Proceedings of the 33rd International Conference on Software Engineering (2011)*, Seiten 101 – 110, New York, 2011. Association for Computing Machinery.

Thomas Stolz, Cornelia Stroh und Aina Urdze. *On Comitatives and Related Categories*. Mouton de Gruyter, Berlin und New York, 2006.

Clemens Szyperski und Cuno Pfister. Component-Oriented Programming: WCOP'96 Workshop Report. In: Max Mühlhäuser (Hrsg.), *Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP'96*, Seiten 127 – 130, Heidelberg, 1996. dpunkt Verlag.

Clemens Szyperski, Dominik Gruntz und Stephan Murer. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley Verlag, London u.a., 2. Auflage, 2002.

Lucien Tesnière. *Grundzüge der strukturalen Syntax (dt. Übersetzung)*. Klett-Cotta, Stuttgart, 1980.

The Object Management Group Inc. Business Process Model and Notation (BPMN) Version 2.0, 2011a. OMG Dokumentnummer: formal/2011-01-03. Diese Spezifikation ist unter http://www.omg.org/spec/BPMN/2.0/ verfügbar (letzter Abruf am 20.02.2013).

The Object Management Group Inc. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.4.1. Online, 2011b. OMG Dokumentnummer: formal/2011-08-06. Diese Spezifikation ist unter http://www.omg.org/spec/UML/2.4.1/Superstructure/ verfügbar (letzter Abruf am 20.02.2013).

The Object Management Group Inc. OMG Object Constraint Language (OCL), Version 2.3.1 without change bars, 2012. OMG Dokumentnummer: formal/2012-01-01. Diese Spezifikation ist unter http://www.omg.org/spec/OCL/ verfügbar (letzter Abruf am 17.04.2013).

Kristina Toutanova, Dan Klein, Christopher D. Manning und Yoram Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: (ohne Herausgeber) *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, Seiten 173 – 180, Stroudsburg, 2003. Association for Computational Linguistics.

Wei-Tek Tek Tsai, Dimitri Volovik und Thomas F. Keefe. Automated test case generation for programs specified by relational algebra queries. *IEEE Transactions on Software Engineering*, Band 16(Ausgabe 3): Seiten 316 – 324, 1990.

Terry Winograd. *Language as a Cognitive Process - Vol. 1: Syntax.* Addison-Wesley Verlag, Reading u.a., 1983.

Gerhard Wolf und Marek Leszak. *Fehlerklassifikation für Software.* BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., Berlin, 2007. Diese Publikation ist unter https://www.bitkom.org/files/documents/Fehlerklassifikation_fuer_Software_haftung.pdf verfügbar (letzter Abruf am 03.04.2013).

Robert K. Yin. *Case Study Research - Design and Methods.* Sage, Thousand Oaks u.a., 4. Auflage, 2009.

George Kinsley Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley Verlag, Cambridge, 1949.

Eidesstattliche Erklärungen

Ich erkläre hiermit an Eides statt, dass ...

1. ... ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

2. ... ich an keinem anderen Fachbereich einen Antrag auf Eröffnung eines Promotionsprüfungsverfahrens gestellt habe.

Buxtehude, 01.11.2013 *Jan Christian Krause*