# Konzeption und Evaluation eines domänenbasierten Explorers als VS-Code Extension für SAP CAP Anwendungen

**Bachelorarbeit**

aus dem Studiengang Wirtschaftsinformatik Software Engineering

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Max Christian Meinel

30.03.2026


| | |
|---|---|
| **Matrikelnummer, Kurs:** | 7864687, WWI23SEB |
| **Unternehmen:** | SAP SE, 69190, Walldorf |
| **Abteilung:** | CAP Tools & MTX |
| **Leiter des Studiengangs:** | Prof. Dr. Henning Pagnia |
| **Betreuer im Unternehmen:** | Christian Fuchs |
| **Betreuer an der DHBW:** | Ulrich Wolf |

# Selbstständigkeitserklärung

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden- Württemberg vom 29.09.2017. Ich versichere hiermit, dass ich meine Arbeit mit dem Thema:

**Konzeption und Evaluation eines domänenbasierten Explorers als VS-Code Extension für SAP CAP Anwendungen**

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass alle eingereichten Fassungen übereinstimmen.

Walldorf 30.03.2026

Max Christian Meinel

# Abstract

Abstract hier

# Inhaltsverzeichnis

**1 Einleitung** . . . . . . . . **1**
1.1 Motivation . . . . . . . . 1
1.2 Problemstellung . . . . . . . . 1
1.3 Zielsetzung der Arbeit . . . . . . . . 1
1.4 Forschungsfragen . . . . . . . . 1
1.5 Aufbau der Arbeit . . . . . . . . 1

**2 Grundlagen** . . . . . . . . **2**
2.1 SAP Cloud Application Programming Model (CAP) . . . . . . . . 2
2.2 Language Server und Language Server Protocol . . . . . . . . 4
2.3 VS Code Extension Architektur . . . . . . . . 4

**3 Methodik** . . . . . . . . **5**
3.1 Design Science Research . . . . . . . . 6
3.2 Einordnung der Arbeit in den DSR-Prozess . . . . . . . . 6

**4 Problem- und Anforderungsanalyse** . . . . . . . . **7**
4.1 Problemkontext und Analyse von CAP-Repositories . . . . . . . . 7
4.2 Identifizierte Herausforderungen bei der Navigation . . . . . . . . 10
4.3 Ableitung der Designziele . . . . . . . . 13
4.4 Funktionale Anforderungen . . . . . . . . 14
4.5 Abgrenzung der Arbeit . . . . . . . . 15

**5 Integrationsstrategien** . . . . . . . . **17**
5.1 Repository-basierte Integration . . . . . . . . 17
5.2 CSN-basierte Integration . . . . . . . . 18
5.3 Language-Server-basierte Integration . . . . . . . . 19
5.4 Vergleich und Auswahl . . . . . . . . 21
5.5 Auswahl der Integrationsstrategie . . . . . . . . 23

**6 Implementierung des Prototyps** . . . . . . . . **24**
6.1 Technische Architektur der Extension . . . . . . . . 24
6.2 Umsetzung der Integrationsansätze . . . . . . . . 24
6.3 Visualisierung im VS Code Explorer . . . . . . . . 24
6.4 Update- und Synchronisationsstrategie . . . . . . . . 24

**7 Evaluation** . . . . . . . . **25**

7.1 Evaluationskriterien ........ 25
7.2 Vergleich der Integrationsansätze ........ 25
7.3 Analyse der Ergebnisse ........ 25

**8 Diskussion ........ 26**
8.1 Interpretation der Ergebnisse ........ 26
8.2 Limitationen ........ 26

**9 Fazit und Ausblick ........ 27**
9.1 Zusammenfassung der Ergebnisse ........ 27
9.2 Beantwortung der Forschungsfragen ........ 27
9.3 Ausblick auf zukünftige Arbeiten ........ 27

**Literatur ........ a**

# Abbildungsverzeichnis

**Abbildung 1** CAP-Architektur mit CDS als zentralem Baustein [1] . . . . . . . . . . . 2

**Quellcode 2** Einfaches CDL-Datenmodell mit Entities und Association . . . . . 3

**Abbildung 3** CDS-Kompilierungsfluss mit CSN als zentralem Zwischenformat [2] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

**Abbildung 4** Konzeptionelle Fragmentierung eines CAP-Services (vereinfacht). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

**Abbildung 5** Navigationskomplexität typischer Entwicklungsaufgaben im Bookshop-Projekt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**Quellcode 6** CDS-Service-Definition mit Projektionen und Action . . . . . . . . . 17

**Quellcode 7** Ausschnitt eines CSN-Modells mit Service und Entity . . . . . . . . 19

**Quellcode 8** Beispiel einer Workspace-Symbol-Antwort des CAP Language Servers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

**Abbildung 9** Ergebnis der Nutzwertanalyse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

# Tabellenverzeichnis

**Tabelle 1** **Zentrale Artefakte eines CAP-Services (eigene Darstellung basierend auf [3, Sec. Domain Modeling])** . . . . . . . . . . . . . . . . . . . . . . . . . **10**

**Tabelle 2** **Bewertungskriterien und Gewichtung** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **21**

**Tabelle 3** **Nutzwertanalyse der Integrationsstrategien** . . . . . . . . . . . . . . . . . . . . . **22**

**Tabelle 4** **Detaillierte Begründung der Punktvergabe in der Nutzwertanalyse** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **d**

**Tabelle 5** **Szenario „Service verstehen" – 3 Dateien, 10 Navigationsschritte** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **e**

**Tabelle 6** **Szenario „Feld hinzufügen" – 6 Dateien, 6 Navigationsschritte** . . . **e**

**Tabelle 7** **Szenario „Auth anpassen" – 3 Dateien, 4 Navigationsschritte** . . . . **e**

**Tabelle 8** **Szenario „Handler debuggen" – 3 Dateien, 4 Navigationsschritte** . **f**

# Codeverzeichnis

**Abbildung 1** **Exemplarische Struktur eines SAP CAP-Projekts (vereinfacht) [4]. . . . . . 8**

**Abbildung 2** **Verteilung der CatalogService-Artefakte im Bookshop-Projekt . . . . . . 13**

DHBW Duale Hochschule Baden-Württemberg SAP

# 1 Einleitung

## 1.1 Motivation

## 1.2 Problemstellung

## 1.3 Zielsetzung der Arbeit

## 1.4 Forschungsfragen

## 1.5 Aufbau der Arbeit

# 2 Grundlagen

## 2.1 SAP Cloud Application Programming Model (CAP)

Abbildung 1 zeigt die Architektur von CAP im Überblick. CDS bildet den zentralen Baustein, der Services, Datenmodelle und User Interfaces verbindet. CAP unterstützt zwei Laufzeitumgebungen: Node.js mit Express sowie Java mit Spring.

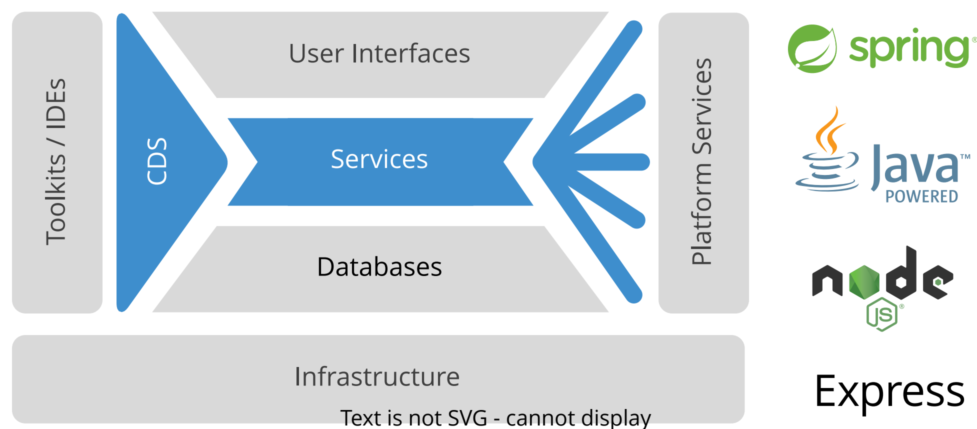


Abbildung 1 — CAP-Architektur mit CDS als zentralem Baustein [1]

- Framework für Enterprise-Cloud-Anwendungen von SAP [5]
- Ziel: Maximale Produktivität durch Best Practices „out of the box“
- Kernprinzipien [5]:
  - Domain-First: Fokus auf Geschäftslogik statt technische Details
  - Separation of Concerns: Trennung von Domäne, Services und UI [6]
  - Service-orientierte Architektur: Funktionalität über definierte APIs [7]
- Technische Features:
  - Multi-Tenancy: Mehrere Mandanten auf einer Anwendungsinstanz, Tenant-spezifische Datenbanken und Erweiterungen
  - Protokolle: OData für Fiori-UIs, OpenAPI für REST-Clients, AsyncAPI für Event-basierte Integration
  - Event-Driven: Asynchrone Kommunikation zwischen Services über Message Broker (SAP Event Mesh, Redis)

  - Datenbank-Abstraktion: Einheitliches API für HANA (Produktion), SQLite (Entwicklung), PostgreSQL (Alternative)
  - SDKs: Node.js/TypeScript für schnelle Entwicklung, Java für Enterprise-Integration

### 2.1.1 CDS und CDL

- CDS (Core Data Services): Rückgrat von CAP für deklarative Modellierung [2]
- CDL (CDS Definition Language): Menschenlesbare Syntax für Modelle [8, Sec. Language Preliminaries]
- Kernkonstrukte [8, Sec. Entities & Type Definitions]:
  - `entity`: Datenstrukturen mit Eigenschaften und Beziehungen
  - `type`: Wiederverwendbare Typdefinitionen (einfach oder strukturiert)
  - `aspect`: Mixins für Querschnittsfunktionalität (z.B. `managed`, `cuid`)
  - `Association`: Beziehungen zwischen Entities (managed oder unmanaged)
- CDS-Toolkit parst Quelldateien und kompiliert in verschiedene Zielformate (Abbildung 3)
- Quellcode 2 zeigt die grundlegende CDL-Syntax für Datenmodellierung

```
entity Books {
  key ID : Integer
  title : String(100);
  author : Association to Authors
}

entity Authors {
  key ID : Integer
  name : String(100);
}
```

Quellcode 2 — Einfaches CDL-Datenmodell mit Entities und Association

### 2.1.2 Core Schema Notation (CSN)

- Kompakte JSON-Repräsentation kompilierter CDS-Modelle [9, Sec. Anatomy]

- Zentrales Zwischenformat: Alle Quellformate werden zu CSN geparst, alle Zielformate aus CSN kompiliert (Abbildung 3)
- Enthält alle Definitionen des Projekts unter vollqualifizierten Namen

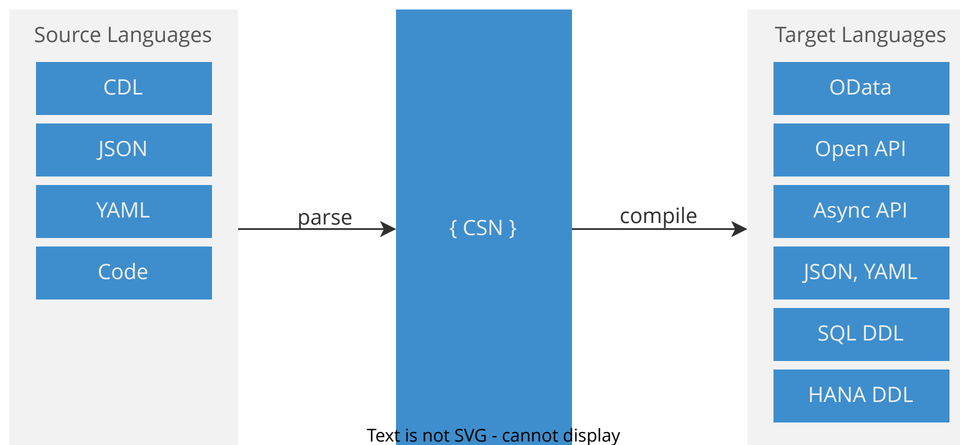


Abbildung 3 — CDS-Kompilierungsfluss mit CSN als zentralem Zwischenformat [2]

## 2.2 Language Server und Language Server Protocol

### 2.2.1 Architektur des Language Servers

### 2.2.2 Relevante LSP-Funktionalitäten

## 2.3 VS Code Extension Architektur

### 2.3.1 Tree View API

### 2.3.2 Integration von Language Servern

# 3 Methodik

**grobe Idee**

Für diese Arbeit wird der Design-Science-Research-Ansatz (DSR) als methodische Grundlage verwendet. Dieser Ansatz eignet sich insbesondere für Arbeiten, bei denen ein konkretes Artefakt entwickelt wird, das ein praktisches Problem adressiert. In dieser Arbeit stellt die entwickelte VS-Code Extension sowie das zugrunde liegende Strukturmodell (domänenbasierter Explorer) dieses Artefakt dar.

Der Design-Science-Research-Ansatz gliedert sich in mehrere Phasen. Zunächst wird das zugrundeliegende Problem präzise analysiert und abgegrenzt. Dabei wird untersucht, welche strukturellen Herausforderungen in größeren SAP CAP Projekten entstehen und weshalb bestehende Werkzeuge nur eingeschränkt eine domänenorientierte Übersicht ermöglichen.

Auf Basis dieser Problemdefinition werden funktionale sowie nicht-funktionale Anforderungen an die Erweiterung definiert. Dazu gehören unter anderem Anforderungen an die Strukturierung der Services, die Filter- und Sortiermöglichkeiten sowie Aspekte der Usability.

Im Anschluss erfolgt die Konzeption des Artefakts. Dabei werden unterschiedliche Modellierungsansätze zur Strukturierung des Repositories betrachtet. Insbesondere werden das Core Schema Notation (CSN) als statische Strukturquelle sowie der Language Server als Quelle für positions- und dateibezogene Informationen analysiert. Ziel ist es, ein geeignetes Modell zu entwerfen, das beide Informationsquellen sinnvoll kombiniert.

Parallel zur konzeptionellen Modellierung werden verschiedene grafische Mockups erstellt. Diese dienen dazu, unterschiedliche Ansätze der Visualisierung und Interaktion zu vergleichen. Für den Vergleich werden zuvor definierte Kriterien

herangezogen, beispielsweise Übersichtlichkeit, Nachvollziehbarkeit der Domänenstruktur und Erweiterbarkeit.

Auf Grundlage des ausgewählten Konzepts wird anschließend ein prototypischer Explorer als VS-Code Extension implementiert. Die Implementierung dient dabei nicht primär als Produktivlösung, sondern als Demonstrator des entworfenen Modells.

Abschließend erfolgt eine Evaluation des entwickelten Artefakts. Dabei werden sowohl die strukturelle Modellierung (Kombination von CSN und Language Server) als auch die gewählte grafische Darstellung anhand der zuvor definierten Kriterien bewertet. Ziel der Evaluation ist es, die Eignung des entwickelten Konzepts zur Unterstützung von Entwicklern in größeren CAP-Projekten zu überprüfen.

## 3.1 Design Science Research

## 3.2 Einordnung der Arbeit in den DSR-Prozess

### 3.2.1 Problemidentifikation

### 3.2.2 Zieldefinition

### 3.2.3 Artefaktentwurf

### 3.2.4 Demonstration

### 3.2.5 Evaluation

# 4 Problem- und Anforderungsanalyse

Dieses Kapitel konkretisiert die im Methodikteil beschriebene Problemidentifikation und Zieldefinition im Kontext von SAP CAP-Projekten. Zunächst wird die strukturelle Organisation typischer CAP-Repositories analysiert und die Verteilung servicebezogener Artefakte untersucht. Darauf aufbauend werden navigationsbezogene Herausforderungen abgeleitet, aus denen Designziele sowie funktionale und nicht-funktionale Anforderungen an die zu entwickelnde VS-Code Extension formuliert werden. Abschließend wird die Arbeit inhaltlich abgegrenzt.

## 4.1 Problemkontext und Analyse von CAP-Repositories

### 4.1.1 Typische Struktur von CAP-Projekten

SAP CAP gibt eine strenge Projektstruktur vor, die das Prinzip der Separation of Concerns umsetzt und damit die Wartbarkeit skalierender Projekte unterstützt [10, Sec. Separation of Concerns]. Ein typisches CAP-Projekt folgt der in Abbildung 1 dargestellten Struktur.

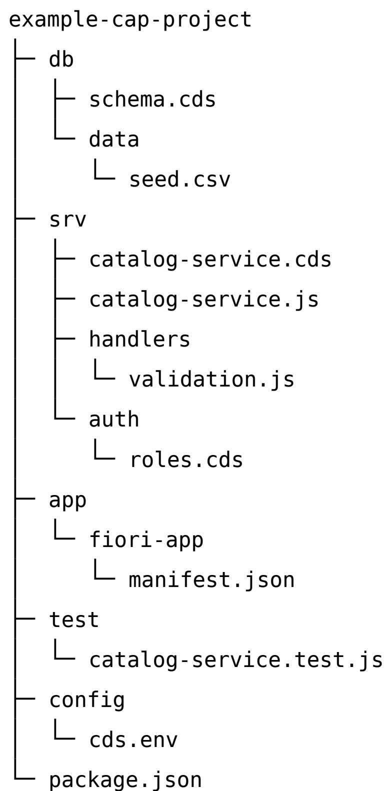


Abbildung 1 — Exemplarische Struktur eines SAP CAP-Projekts (vereinfacht) [4].

**example-cap-project/**
Das Rootverzeichnis bildet die Basis des Projekts und bündelt alle fachlichen, technischen und konfigurativen Artefakte nach dem Prinzip der Separation of Concerns.

**db/**
Dieses Verzeichnis enthält das fachliche Domänenmodell. Hier werden Entities und ihre Beziehungen in CDS definiert, die die Persistenz- und Datenstrukturebene repräsentieren.

**srv/**
Dieses Verzeichnis beinhaltet Service-Definitionen und Geschäftslogik. Der Service exponiert ausgewählte Teile des Domänenmodells und verknüpft deklarative Modellierung mit imperativer Implementierung.

**app/**
Dieses Verzeichnis enthält optionale Frontend-Anwendungen, die die definierten Services konsumieren und die Präsentationsebene repräsentieren.

**test/**
Dieses Verzeichnis beinhaltet Testfälle zur Validierung der Service-Logik und unterstützt damit Qualitätssicherung und Wartbarkeit.

**config/**
Dieses Verzeichnis enthält umgebungsspezifische Konfigurationen zur Steuerung von Laufzeit- und Deployment-Parametern.

**package.json**
Diese Datei definiert Projektmetadaten, Abhängigkeiten sowie Skripte für Build, Start und Tests.

### 4.1.2 Verteilung servicebezogener Artefakte

Ein CAP-Service besteht nicht aus einem einzelnen Artefakt, sondern setzt sich aus mehreren funktionalen Bereichen zusammen, die unterschiedlichen Verantwortlichkeiten zugeordnet sind. Diese Fragmentierung erfolgt entlang technischer und fachlicher Concerns, wobei die Zuordnung zwischen den Artefakten über das Projekt verteilt ist.

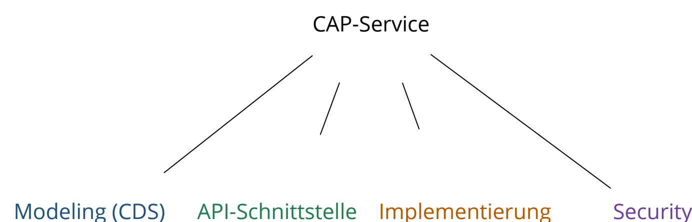


Abbildung 4 — Konzeptionelle Fragmentierung eines CAP-Services (vereinfacht).

Abbildung 4 verdeutlicht, dass ein CAP-Service nicht als monolithisches Artefakt vorliegt, sondern sich über mehrere konzeptionelle Ebenen erstreckt. Modell- und Service-Definition sind konzeptionell getrennt, wobei die Service-Schnittstelle nicht zwingend das vollständige Domänenmodell abbildet. Das Laufzeitverhalten ist nicht im Modell selbst definiert, sondern in separaten Implementierungen ausgelagert. Sicherheitsrelevante Regeln werden als Annotationen außerhalb der

Kernlogik verortet. Dadurch sind fachliche und technische Aspekte strukturell entkoppelt [11, Sec. Focus on Domain].

| Kind | Ebene | Beispiele |
|---|---|---|
| Entity | Modeling (CDS) | Books, Authors |
| Type | Modeling (CDS) | UUID, String, Decimal |
| Association | Modeling (CDS) | to Author, to Items |
| Aspect | Modeling (CDS) | Reusable model extensions |
| Service | API-Schnittstelle | CatalogService |
| Projection | API-Schnittstelle | Books as projection on … |
| Action / Function | API-Schnittstelle | submitOrder(), topSelling() |
| Event Handler | Implementierung | before / on / after |
| Authorization Annotations | Security | @restrict, @requires |

Tabelle 1 — Zentrale Artefakte eines CAP-Services (eigene Darstellung basierend auf [3, Sec. Domain Modeling])

Tabelle 1 konkretisiert die in der Abbildung 4 dargestellten Ebenen durch die zentralen Artefakt-Kinds des CAP-Modells. Ein vollständiges Serviceverständnis erfordert die Betrachtung mehrerer Artefakt-Kategorien, deren Beziehungen über Referenzen hergestellt werden. Eine zentrale, aggregierte Repräsentation eines Services existiert nicht auf Artefaktebene. Änderungen an einem Service können daher mehrere Ebenen gleichzeitig betreffen, da sich die Service-Identität erst aus dem Zusammenspiel aller Ebenen ergibt. Diese Architektur folgt den Prinzipien von Domain-Driven Design und Separation of Concerns [3, Sec. Separation of Concerns], [11, Sec. Focus on Domain].

## 4.2 Identifizierte Herausforderungen bei der Navigation

### 4.2.1 Verteilte Artefakte und implizite Abhängigkeiten

Ein CAP-Service erstreckt sich über mehrere konzeptionelle Ebenen, wobei Modell, Schnittstelle, Implementierung und Sicherheitsregeln physisch getrennt sind. Die Beziehungen zwischen diesen Artefakten sind nicht visuell auf einen Blick erkennbar. Projektionen referenzieren Entities, Event Handler referenzieren Service-Ope-

rationen, und Sicherheitsannotationen beeinflussen das Verhalten ohne zentrale Sichtbarkeit. Kein einzelnes Artefakt enthält die vollständige Service-Definition.

Diese Problematik entspricht der von Tarr et al. beschriebenen „Tyrannei der dominanten Dekomposition“ [12, p. 809]: Software wird typischerweise entlang einer primären Dimension strukturiert. Im Fall von CAP-Projekten nach Ordnern und Dateien entsprechend technischer Verantwortlichkeiten. Andere wichtige Dimensionen, wie die servicezentrierte fachliche Sicht, werden dadurch über das Projekt verteilt und sind nicht mehr unmittelbar sichtbar [3].

### 4.2.2 Erhöhter Navigationsaufwand und fehlende Service-Übersicht

Mit zunehmender Weiterentwicklung eines Softwaresystems steigt dessen strukturelle Komplexität. Lehman beschreibt dieses Phänomen im Gesetz der zunehmenden Komplexität, wonach die Komplexität eines Systems mit fortlaufenden Änderungen wächst, sofern keine Maßnahmen zur Reduktion dieser Komplexität ergriffen werden [13, p. 1068]. Zur Analyse eines Services sind mehrere Dateiwechsel erforderlich, etwa zwischen Datenbankmodell, Implementierung und Sicherheitsregeln. Da keine aggregierte Sicht auf alle servicebezogenen Artefakte existiert, muss die Struktur eines Services rekonstruiert werden. Diese Kontextwechsel erhöhen die kognitive Belastung, da Menschen nur über eine begrenzte kognitive Verarbeitungskapazität verfügen und mehrere Informationen gleichzeitig berücksichtigen müssen [14, p. 261].

### 4.2.3 Quantifizierung anhand konkreter Szenarien

Die beschriebenen Navigationsprobleme lassen sich anhand typischer Entwicklungsaufgaben im Bookshop-Beispielprojekt [4] konkretisieren. Abbildung 5 zeigt die Navigationskomplexität für vier repräsentative Szenarien, ermittelt durch manuelle Analyse des Projekts. Die detaillierten Navigationspfade mit konkreten Dateien und Zeilennummern sind im Anhang dokumentiert.

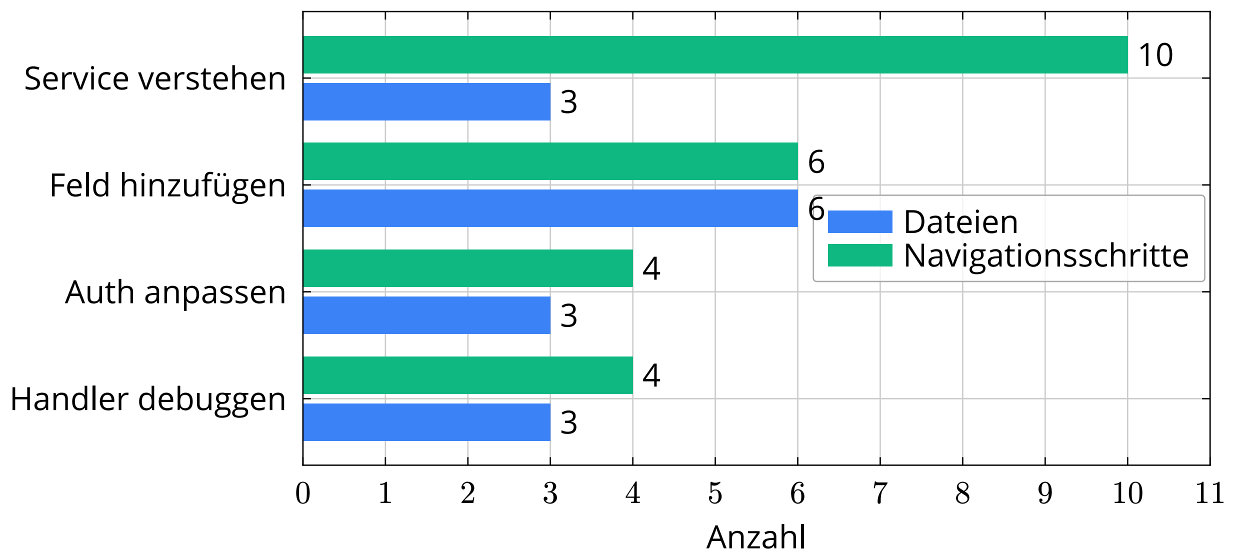


Abbildung 5 — Navigationskomplexität typischer Entwicklungsaufgaben im Bookshop-Projekt

Das Szenario „Handler debuggen" zeigt exemplarisch einen typischen Debugging-Workflow: Ein Entwickler untersucht das Verhalten der `submitOrder`-Action im CatalogService. Er öffnet zunächst `srv/cat-service.js` (Zeile 14), analysiert die Handler-Logik, navigiert zur Action-Definition in `srv/cat-service.cds` (Zeile 19) und prüft das referenzierte Entity `Books` in `db/schema.cds`. Dieser Workflow erfordert 3 Dateien und 4 Navigationsschritte.

Das komplexeste Szenario „Service verstehen" erfordert das Nachvollziehen aller Aspekte des CatalogService: Service-Definition mit allen Projektionen und Actions (`srv/cat-service.cds`), das vollständige Datenmodell mit allen referenzierten Entities (`db/schema.cds`) sowie die Handler-Implementierung (`srv/cat-service.js`). Obwohl nur 3 Dateien beteiligt sind, erfordert das Verständnis 10 Navigationsschritte zwischen verschiedenen Artefakten.

Abbildung 2 visualisiert diese Fragmentierung am Beispiel des CatalogService.

└─ **CatalogService** (fachliche Einheit)

→`db/schema.cds` — Domänenmodell (Books, Authors, Genres, Price)

→`srv/cat-service.cds` — Service-Definition, Projektionen, Action

└─ `srv/cat-service.js` — Event Handler (after, submitOrder)

**3 Dateien** in **2 Verzeichnissen** — keine aggregierte Sicht verfügbar

Abbildung 2 — Verteilung der CatalogService-Artefakte im Bookshop-Projekt

#### 4.2.4 Bestehende Ansätze zur Navigation in Entwicklungsumgebungen

Visual Studio Code bietet grundlegende Navigationsfunktionen wie die Outline-Ansicht [15, Sec. Outline View] und Go-to-Definition [16, Sec. Go to Definition]. Die SAP CDS Language Support Extension [17] erweitert dies um CDS-spezifische Features wie Syntax-Highlighting und Autocomplete. Forschung zu Informationsbedürfnissen in Softwareteams zeigt, dass Entwickler häufig Schwierigkeiten haben, relevante Informationen über Code, Design und Programmausführung aus verschiedenen Artefakten zusammenzuführen [18, p. 347]. Diese Werkzeuge operieren überwiegend datei- und symbolzentriert, sodass eine servicezentrierte Aggregation über mehrere Dateien und Ebenen hinweg nicht unterstützt wird.

### 4.3 Ableitung der Designziele

Aus den identifizierten Navigationsproblemen werden nun zentrale Designziele abgeleitet. Diese beschreiben lösungsneutral, welche Eigenschaften eine geeignete Lösung aufweisen muss. Die Designziele bilden die Grundlage für die anschließende Definition funktionaler und nicht-funktionaler Anforderungen.

#### Aggregierte Service-Sicht

Alle Artefakte eines Services müssen gebündelt dargestellt werden. Die Lösung muss eine einheitliche Einstiegsperspektive auf einen Service bieten, in der Modell, Schnittstelle, Implementierung und Sicherheitsaspekte gemeinsam sichtbar sind. Die Service-Identität muss ohne manuelle Rekonstruktion erfassbar sein.

#### Reduktion von Navigations- und Kontextwechseln

Die Anzahl notwendiger Dateiwechsel muss minimiert werden. Deklarative und imperative Artefakte müssen kontextnah zugänglich sein. Relevante Artefakte müssen schnell auffindbar sein. Kognitive Kontextwechsel zwischen den Ebenen müssen reduziert werden.

#### Explizite Darstellung von Abhängigkeiten

Beziehungen zwischen allen Arten von Entitäten müssen sichtbar gemacht werden. Verknüpfungen zwischen Service-Definition und Implementierung müssen nachvollziehbar sein. Sicherheitsrelevante Annotationen müssen eindeutig zuordenbar sein. Referenzen müssen transparent werden.

#### Servicezentrierte Navigation

Die Navigation muss sich an den fachlichen Services orientieren. Die technische Ordnerstruktur darf nicht primärer Navigationsanker sein. Der Service muss als logische Einheit im Mittelpunkt stehen.

## 4.4 Funktionale Anforderungen

Aus den zuvor formulierten Designzielen lassen sich konkrete funktionale Anforderungen an die zu entwickelnde VS-Code Extension ableiten. Während die Designziele die gewünschten Eigenschaften der Lösung auf konzeptioneller Ebene beschreiben, spezifizieren die folgenden Anforderungen die Funktionen, die das Artefakt zur Unterstützung der Navigation in CAP-Projekten bereitstellen muss.

#### FA1 Identifikation von CAP-Services

Die Extension muss CAP-Services innerhalb eines Projekts automatisch identifizieren und alle Services anzeigen.

#### FA2 Aggregierte Serviceübersicht

Die Extension muss eine aggregierte Darstellung aller zu einem Service gehörenden Artefakte bereitstellen.

#### FA3 Navigation zu Artefakten

Die Extension muss eine direkte Navigation von der Serviceübersicht zu den entsprechenden Artefakten im Quellcode ermöglichen.

**FA4 Darstellung von Artefaktbeziehungen**
Die Extension muss Beziehungen und Hierarchien zwischen servicebezogenen Artefakten sichtbar machen.

### 4.4.1 Nicht-funktionale Anforderungen

Neben den funktionalen Anforderungen ergeben sich aus dem Nutzungskontext innerhalb der Entwicklungsumgebung zusätzliche nicht-funktionale Anforderungen. Diese beschreiben qualitative Eigenschaften und Rahmenbedingungen, die bei der Umsetzung der Extension berücksichtigt werden müssen.

**NFA1 Integration in Visual Studio Code**
Die Lösung muss als Erweiterung für die Entwicklungsumgebung Visual Studio Code implementiert werden und sich in deren Erweiterungsmechanismus [19] integrieren.

**NFA2 Kompatibilität mit CAP-Projektstrukturen**
Die Extension muss mit der typischen Struktur von SAP CAP-Projekten kompatibel sein und relevante Artefakte in den vorgesehenen Verzeichnissen erkennen können.

**NFA3 Performante Analyse von Projekten**
Die Analyse eines CAP-Repositories soll ohne merkliche Verzögerung erfolgen, sodass die Nutzung der Extension den Entwicklungsworkflow nicht beeinträchtigt.

**NFA4 Aktualität der Artefaktübersicht**
Änderungen an relevanten Projektartefakten müssen automatisch erkannt werden, sodass die dargestellte Serviceübersicht den aktuellen Projektzustand widerspiegelt.

## 4.5 Abgrenzung der Arbeit

Die vorliegende Arbeit fokussiert auf die Unterstützung der Navigation in CAP-Projekten durch eine VS-Code Extension. Daraus ergeben sich bewusste Einschränkungen des Untersuchungs- und Lösungsumfangs.

Ziel der Extension ist die Verbesserung der Navigation zwischen servicebezogenen Artefakten. Funktionen zur automatischen Codegenerierung oder Refaktorierung sind explizit nicht Bestandteil der Lösung.

Die Evaluation erfolgt anhand ausgewählter CAP-Projekte. Eine umfassende Nutzerstudie mit CAP-Entwicklern ist nicht Bestandteil der Arbeit.

# 5 Integrationsstrategien

Dieses Kapitel stellt verschiedene Integrationsstrategien für die Implementierung der Service-Struktur-Analyse vor. Es entspricht der DSR-Phase „Artefaktentwurf“. Das Ziel ist die systematische Auswahl einer geeigneten Strategie basierend auf den Anforderungen aus Kapitel 4.

## 5.1 Repository-basierte Integration

Die repository-basierte Integration rekonstruiert die Service-Struktur direkt aus den Dateien des Projektrepositories. Die Analyse erfolgt durch Parsing der `.cds`-Dateien, wobei Entities, Services, Projektionen und Operationen syntaktisch extrahiert werden. Die Implementierungslogik wird durch Analyse von `.js`- und `.ts`-Dateien ermittelt. Beziehungen zwischen Artefakten müssen aus Referenzen im Code rekonstruiert werden. Quellcode 6 zeigt einen Ausschnitt einer typischen CDS-Service-Definition.

```
using { sap.capire.bookshop as my } from ‚../db/schema‘

service CatalogService {
  @readonly entity Books as projection on my.Books {
    *, author.name as author, genre.name as genre
  } excluding { createdBy, modifiedBy };

  @requires: ‚authenticated-user‘
  action submitOrder ( book: Books:ID, quantity: Integer );
}
```

Quellcode 6 — CDS-Service-Definition mit Projektionen und Action

Die technische Umsetzung erfordert einen eigenen Parser für die CDS-Sprache. CDS ist eine domänenspezifische Sprache mit eigener Grammatik, die Konstrukte wie `entity`, `service`, `projection`, `aspect` und `type` unterstützt. Wie Quellcode 6 zeigt, enthält bereits eine einfache Service-Definition mehrere Parsing-Herausforderun-

gen: den `using`-Import mit Alias, verschachtelte Projektionen mit Feld-Aliasing (`author.name as author`), Annotationen (`@readonly`, `@requires`) sowie Actions mit Parametern. Ein Parser müsste all diese Konstrukte erkennen, verschachtelte Strukturen auflösen und Referenzen zwischen Dateien verfolgen. Zusätzlich müssten `.js`- und `.ts`-Dateien analysiert werden, um Event Handler den entsprechenden Services zuzuordnen.

Dieser Ansatz bringt einen hohen Implementierungsaufwand für Parsing und Referenzauflösung mit sich. Zudem besteht das Risiko semantischer Fehlinterpretationen der CDS-Sprache, da ein eigener Parser die Sprachsemantik nicht vollständig abbilden kann. Darüber hinaus dupliziert dieser Ansatz Analysefunktionen, die bereits durch das CAP-Tooling bereitgestellt werden. Der Vorteil dieses Ansatzes liegt in seiner hohen Flexibilität. Ein eigener Parser ermöglicht die vollständige Kontrolle über die extrahierten Informationen und ist unabhängig von externen Tools.

## 5.2 CSN-basierte Integration

Die CSN-basierte Integration nutzt das von CAP generierte Core Schema Notation (CSN) Modell als Informationsquelle. CSN ist eine kompilierte JSON-Repräsentation aller CDS-Artefakte. Der CAP-Compiler erzeugt CSN aus den `.cds`-Quelldateien und löst dabei alle Referenzen, Imports und Vererbungen auf.

Im CSN-Modell sind Entities, Services, Projektionen und Operationen strukturiert enthalten. Jedes Artefakt ist unter seinem vollqualifizierten Namen als JSON-Objekt abgelegt. Das Feld `kind` gibt den Artefakttyp an (z.B. `entity`, `service`, `function`). Beziehungen zwischen Artefakten sind bereits explizit aufgelöst, sodass keine eigene Referenzauflösung erforderlich ist. Quellcode 7 zeigt einen Ausschnitt eines CSN-Modells.

```
{
  „definitions“: {
    „TravelService“: {
      „kind“: „service“,
      „$location“: { „file“: „srv/travel-service.cds“, „line“: 5 }
    },
    „TravelService.Travels“: {
      „kind“: „entity“,
      „projection“: {
        „from“: { „ref“: [„sap.capire.travels.Travels“] }
      },
      „elements“: {
        „ID“: { „key“: true, „type“: „cds.Integer“ },
        „Description“: { „type“: „cds.String“, „length“: 1024 },
        „BeginDate“: { „type“: „cds.Date“ }
      }
    }
  }
}
```

Quellcode 7 — Ausschnitt eines CSN-Modells mit Service und Entity

Die semantisch korrekte Interpretation ist durch die Nutzung des CAP-Compilers garantiert. Das JSON-Format ist direkt parsebar und erfordert keinen eigenen CDS-Parser. CSN bietet optional Positionsinformationen über das Feld `$location`, die durch einen speziellen Kompilierungsmodus aktiviert werden können. Diese Positionsinformationen sind jedoch unvollständig: Nicht alle Artefakte und Elemente erhalten Location-Daten. Für eine zuverlässige Navigation zu Quellcode-Positionen (FA3) ist diese Abdeckung nicht ausreichend. Zusätzlich ist CSN integraler Bestandteil des CAP-Compilers. Erweiterungen am CSN-Format würden den Build-Prozess verlangsamen und sind daher nicht praktikabel.

## 5.3 Language-Server-basierte Integration

Die Language-Server-basierte Integration nutzt den CAP Language Server als zentrale Analysequelle. Der Language Server implementiert das Language Server

Protocol (LSP) und stellt semantische Analysen der CDS-Artefakte bereit. Er läuft als separater Prozess und kommuniziert mit VS Code über JSON-RPC.

Für die Service-Struktur-Analyse ist der LSP-Request `workspace/symbol` relevant. Dieser Request liefert alle Symbole eines Workspace in einem strukturierten JSON. Jedes Symbol enthält den Namen, den Container (übergeordnetes Element), die Datei-URI, die exakte Position (range) im Quellcode sowie den Symboltyp (kind). Quellcode 8 zeigt einen Ausschnitt der Antwort des CAP Language Servers.

```
[
  {
    „name“: „Bookings“,
    „containerName“: „TravelService“,
    „location“: {
      „uri“: „file:///srv/travel-service.cds“,
      „range“: {
        „start“: { „line“: 4, „character“: 0 },
        „end“: { „line“: 21, „character“: 1 }
      }
    },
    „kind“: 5
  },
  {
    „name“: „BookingDate“,
    „containerName“: „TravelService.Bookings“,
    „location“: { … },
    „kind“: 8
  }
]
```

Quellcode 8 — Beispiel einer Workspace-Symbol-Antwort des CAP Language Servers

Das Feld `containerName` kodiert die Hierarchie der Artefakte. Ein Wert wie `TravelService.Bookings` bedeutet, dass das Symbol `BookingDate` innerhalb von `Bookings` liegt, welches wiederum zum `TravelService` gehört. Das Feld `kind` gibt den Symboltyp (kind) an: 5 steht für eine Entity, 8 für ein Feld, 12 für eine Funktion und

19 für einen Service. Diese Information ermöglicht die Rekonstruktion der Service-Hierarchie.

Der Language Server bietet eine native Integration in VS Code über das Language Server Protocol. Bei Bedarf ist der Language Server erweiterbar, da diese Arbeit in der Abteilung entsteht, die den Language Server entwickelt. Die Strategie setzt jedoch voraus, dass der Language Server zur Laufzeit verfügbar ist.

## 5.4 Vergleich und Auswahl

Im Rahmen dieser Bachelorarbeit wird nur eine Strategie implementiert. Zur Auswahl wurde sich für eine Nutzwertanalyse entschieden. Die Strategie mit dem höchsten Nutzwert wird in Abschnitt 6 umgesetzt.

### 5.4.1 Nutzwertanalyse

Für die Nutzwertanalyse werden Bewertungskriterien herangezogen. Diese leiten sich von den Anforderungen ab oder werden als sinnvoll erachtet. Tabelle 2 zeigt diese Bewertungskriterien mit ihrer Gewichtung und Herleitung aus den Anforderungen. In den Abschnitten darunter werden die Kriterien näher erläutert.

| Kriterium | Gewicht | Herleitung |
|---|---|---|
| K1: Semantische Genauigkeit | 25% | FA1, FA2 |
| K2: Positionsinformationen | 25% | FA3 |
| K3: VS Code Integration | 20% | NFA1 |
| K4: Implementierungsaufwand | 15% | – |
| K5: Wartbarkeit | 10% | NFA4 |
| K6: Unabhängigkeit | 5% | – |

Tabelle 2 — Bewertungskriterien und Gewichtung

**K1: Semantische Genauigkeit (25%)** beschreibt die Fähigkeit, CDS-Artefakte korrekt zu identifizieren und zu interpretieren. Das Kriterium leitet sich aus FA1 und FA2 ab, da Services korrekt erkannt werden müssen. Es erhält die höchste Gewichtung, da ohne korrekte Erkennung die Extension nicht nutzbar ist.

**K2: Positionsinformationen (25%)** bezeichnet die Verfügbarkeit von Quellcode-Positionen (Zeile/Spalte) für Artefakte. Das Kriterium leitet sich aus FA3 ab, da

Navigation zu Definitionen möglich sein muss. Es erhält ebenfalls die höchste Gewichtung, da Navigation das zentrale Feature der Extension ist.

**K3: VS Code Integration (20%)** bewertet die Kompatibilität mit VS Code APIs und der Erweiterungsarchitektur. Das Kriterium leitet sich aus NFA1 ab und fordert eine nahtlose Einbindung in die IDE. Die hohe Gewichtung ergibt sich daraus, dass schlechte Integration die Akzeptanz mindert.

**K4: Implementierungsaufwand (15%)** erfasst Komplexität und Zeitbedarf für die Umsetzung. Es gibt keine direkte Anforderung, jedoch besteht eine praktische Ressourcenbeschränkung im Rahmen dieser Bachelorarbeit. Die mittlere Gewichtung spiegelt den realistischen Projektrahmen wider.

**K5: Wartbarkeit (10%)** bewertet die langfristige Anpassbarkeit bei Änderungen am CAP-Framework. Das Kriterium leitet sich aus NFA4 ab, da die Extension auch zukünftig funktionieren soll. Die geringere Gewichtung ergibt sich daraus, dass zunächst die Funktionalität wichtiger ist.

**K6: Unabhängigkeit (5%)** bezeichnet die Freiheit von externen Laufzeitabhängigkeiten. Es gibt keine direkte Anforderung. Abhängigkeiten sind akzeptabel, wenn die Funktionalität dadurch besser wird. Die niedrigste Gewichtung ist ein bewusster Trade-off.

Tabelle 3 zeigt die Bewertung der drei Strategien anhand der definierten Kriterien.

| **Kriterium** | **Gewicht** | **Repository** | **CSN** | **Language Server** |
|---|---|---|---|---|
| K1: Semantische Genauigkeit | 25% | 2 | 5 | 5 |
| K2: Positionsinformationen | 25% | 3 | 1 | 5 |
| K3: VS Code Integration | 20% | 2 | 3 | 5 |
| K4: Implementierungsaufwand | 15% | 2 | 4 | 3 |
| K5: Wartbarkeit | 10% | 2 | 3 | 4 |
| K6: Unabhängigkeit | 5% | 5 | 2 | 2 |
| **Gewichtete Summe** | 100% | **2.40** | **3.10** | **4.45** |

Tabelle 3 — Nutzwertanalyse der Integrationsstrategien

**Punkteskala:** 1 = unzureichend | 2 = ausreichend | 3 = befriedigend | 4 = gut | 5 = sehr gut

Die detaillierte Begründung der Punktvergabe für jedes Kriterium und jede Strategie ist in Tabelle 4 im Anhang dokumentiert. Die wesentlichen Unterschiede lassen sich wie folgt zusammenfassen: Bei K1 (Semantische Genauigkeit) erzielen CSN und Language Server Höchstwerte, da beide den CAP-Compiler nutzen. Der entscheidende Differenzierungsfaktor ist K2 (Positionsinformationen): CSN liefert diese nur unvollständig, während der Language Server über die Location-API präzise Quellcode-Positionen bereitstellt. Bei K3 (VS Code Integration) profitiert der Language Server von der nativen LSP-Unterstützung, während Repository und CSN zusätzliche Adapter erfordern. Der Implementierungsaufwand (K4) ist bei CSN am geringsten, da das JSON-Format direkt parsebar ist.

## 5.5 Auswahl der Integrationsstrategie

Die Nutzwertanalyse ergibt den höchsten Wert für die Language-Server-basierte Integration mit 4.45 Punkten (Abbildung 9). Der Language Server bietet die beste VS Code Integration, da er das Language Server Protocol nativ implementiert. Er liefert vollständige Positionsinformationen, während CSN diese nur unvollständig bereitstellt. Ein eigener Parser wäre aufwendig und fehleranfällig. Zusätzlich kann der Language Server bei Bedarf erweitert werden, da diese Arbeit in der Abteilung entsteht, die ihn entwickelt.

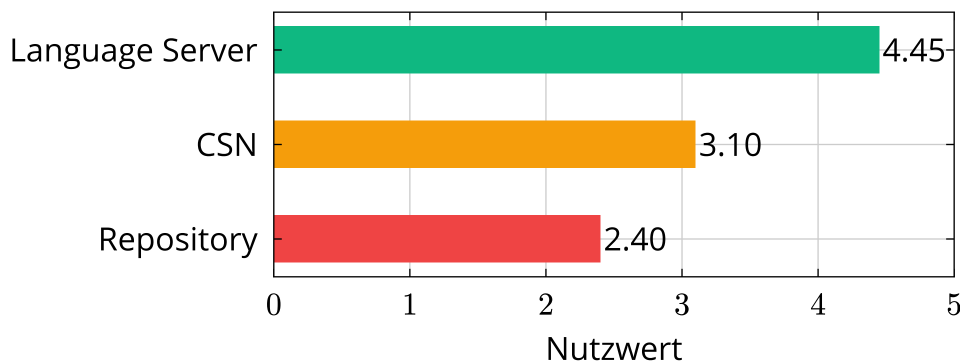


Abbildung 9 — Ergebnis der Nutzwertanalyse

Die Wahl des Language Servers bringt eine Abhängigkeit zur Laufzeit mit sich. Dieser Trade-off wird zugunsten der höheren Funktionalität akzeptiert. Das folgende Kapitel beschreibt die Implementierung der Extension auf Basis der gewählten Language-Server-Integration.

# 6 Implementierung des Prototyps

## 6.1 Technische Architektur der Extension

## 6.2 Umsetzung der Integrationsansätze

## 6.3 Visualisierung im VS Code Explorer

## 6.4 Update- und Synchronisationsstrategie

# 7 Evaluation

## 7.1 Evaluationskriterien

## 7.2 Vergleich der Integrationsansätze

## 7.3 Analyse der Ergebnisse

# 8 Diskussion

## 8.1 Interpretation der Ergebnisse

## 8.2 Limitationen

# 9 Fazit und Ausblick

## 9.1 Zusammenfassung der Ergebnisse

## 9.2 Beantwortung der Forschungsfragen

## 9.3 Ausblick auf zukünftige Arbeiten

# Literatur

[1] SAP, „Concepts Overview | capire“. Zugegriffen: 30. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/get-started/concepts

[2] SAP, „Core Data Services (CDS) | capire“. Zugegriffen: 30. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/cds/

[3] SAP, „Domain Modeling | capire“. Zugegriffen: 10. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/guides/domain

[4] capire, „GitHub – capire/bookshop: Our primer sample to get started in a nutshell“. Zugegriffen: 25. Februar 2026. [Online]. Verfügbar unter: https://github.com/capire/bookshop

[5] SAP, „About CAP | capire“. Zugegriffen: 30. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/

[6] O. Nierstrasz und F. Achermann, „Separation of Concerns through Unification of Concepts“, in *ECOOP 2000 Workshop on Aspects & Dimensions of Concerns*, Bern, Switzerland, 2000.

[7] M. P. Papazoglou, „Service-Oriented Computing: Concepts, Characteristics and Directions“, in *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE)*, IEEE, 2003, S. 3–12. doi: 10.1109/WISE.2003.1254461.

[8] SAP, „Definition Language (CDL) | capire“. Zugegriffen: 30. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/cds/cdl

[9] SAP, „Core Schema Notation (CSN) | capire“. Zugegriffen: 30. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/cds/csn

[10] SAP, „The Bookshop Sample – Separation of Concerns | capire“. Zugegriffen: 10. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/get-started/bookshop#separation-of-concerns

[11] SAP, „The Bookshop Sample – Focus on Domain | capire“. Zugegriffen: 10. März 2026. [Online]. Verfügbar unter: https://cap.cloud.sap/docs/get-started/bookshop#focus-on-domain

[12] P. Tarr, W. Harrison, H. Ossher, A. Finkelstein, B. Nuseibeh, und D. Perry, „Workshop on Multi-Dimensional Separation of Concerns in Software Engineering“, in *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, Limerick, Ireland: ACM, 2000, S. 809–810.

[13] M. M. Lehman, „Programs, Life Cycles, and Laws of Software Evolution“, *Proceedings of the IEEE*, Bd. 68, Nr. 9, S. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.

[14] J. Sweller, „Cognitive Load During Problem Solving: Effects on Learning“, *Cognitive Science*, Bd. 12, Nr. 2, S. 257–285, 1988, doi: 10.1207/s15516709cog1202_4.

[15] Microsoft, „User Interface | Visual Studio Code“. Zugegriffen: 12. März 2026. [Online]. Verfügbar unter: https://code.visualstudio.com/docs/getstarted/userinterface#_outline-view

[16] Microsoft, „Code Navigation | Visual Studio Code“. Zugegriffen: 12. März 2026. [Online]. Verfügbar unter: https://code.visualstudio.com/docs/editing/editingevolved#_go-to-definition

[17] SAP, „cap/cds-lsp: Language Server for CDS“. Zugegriffen: 12. März 2026. [Online]. Verfügbar unter: https://github.tools.sap/cap/cds-lsp

[18] A. J. Ko, R. DeLine, und G. Venolia, „Information Needs in Collocated Software Development Teams“, in *Proceedings of the 29th International Conference on*

*Software Engineering (ICSE '07)*, IEEE Computer Society, Mai 2007, S. 344–353. doi: 10.1109/ICSE.2007.45.

[19] Microsoft, „Extension API | Visual Studio Code“. Zugegriffen: 10. März 2026. [Online]. Verfügbar unter: https://code.visualstudio.com/api

# Anhang

| Kriterium | Strategie | Punkte | Begründung |
|---|---|---|---|
| K1: Semantische Genauigkeit | Repository | 2 | Eigenes Parsing ohne Nutzung des CAP-Compilers; Risiko semantischer Fehlinterpretationen der CDS-Sprache |
| | CSN | 5 | Nutzt kompiliertes CAP-Modell; semantisch korrekte Interpretation durch den CAP-Compiler garantiert |
| | Language Server | 5 | Nutzt CAP-Compiler; semantische Analyse durch SAP-Tooling bereitgestellt |
| K2: Positionsinformationen | Repository | 3 | Positionen aus eigenem Parsing grundsätzlich verfügbar, aber aufwändige Implementierung erforderlich |
| | CSN | 1 | CSN enthält konstruktionsbedingt keine Quellcode-Positionen (Zeile/Spalte) |
| | Language Server | 5 | Location-API liefert präzise Positionen für Definitionen und Referenzen |
| K3: VS Code Integration | Repository | 2 | Keine native Integration; eigene Adapter erforderlich |
| | CSN | 3 | JSON-Daten müssen in VS-Code-Strukturen transformiert werden |
| | Language Server | 5 | Native LSP-Schnittstellen (DocumentSymbol, Location) direkt nutzbar |
| K4: Implementierungsaufwand | Repository | 2 | Hoher Aufwand: eigener Parser für CDS und JS/TS erforderlich |
| | CSN | 4 | Geringer Aufwand: JSON-Parsing und Traversierung des Modells |
| | Language Server | 3 | Mittlerer Aufwand: LSP-Client-Integration und Request-Handling |
| K5: Wartbarkeit | Repository | 2 | Eigener Parser muss bei CDS-Sprachänderungen angepasst werden |
| | CSN | 3 | CSN-Schema kann sich bei CAP-Updates ändern |
| | Language Server | 4 | LSP ist ein stabiler Standard; Änderungen werden vom LS abstrahiert |
| K6: Unabhängigkeit | Repository | 5 | Keine externen Abhängigkeiten zur Laufzeit |
| | CSN | 2 | Abhängigkeit vom CAP-Compiler zur Modellgenerierung |
| | Language Server | 2 | Abhängigkeit vom laufenden CAP Language Server |

Tabelle 4 — Detaillierte Begründung der Punktvergabe in der Nutzwertanalyse

| # | Datei | Zeile | Aktion |
|---|---|---|---|
| 1 | `srv/cat-service.cds` | 1 | Import-Statement und Namespace identifizieren |
| 2 | `srv/cat-service.cds` | 3 | Service-Definition `CatalogService` analysieren |
| 3 | `srv/cat-service.cds` | 6–9 | Projection `ListOfBooks` verstehen |
| 4 | `srv/cat-service.cds` | 12–16 | Projection `Books` verstehen |
| 5 | `srv/cat-service.cds` | 19 | Action `submitOrder` identifizieren |
| 6 | `db/schema.cds` | 4–13 | Entity `Books` mit Associations verstehen |
| 7 | `db/schema.cds` | 15–23 | Referenzierte Entity `Authors` analysieren |
| 8 | `db/schema.cds` | 26–29 | Referenzierte Entity `Genres` analysieren |
| 9 | `srv/cat-service.js` | 3–6 | Handler-Klasse und Entity-Referenzen |
| 10 | `srv/cat-service.js` | 9–23 | Event Handler (`after`, `on`) analysieren |

Tabelle 5 — Szenario „Service verstehen“ – 3 Dateien, 10 Navigationsschritte

| # | Datei | Zeile | Aktion |
|---|---|---|---|
| 1 | `db/schema.cds` | 4–13 | Entity `Books` finden, Feld hinzufügen |
| 2 | `srv/cat-service.cds` | 6–16 | Projektionen `ListOfBooks` und `Books` anpassen |
| 3 | `srv/admin-service.cds` | 5 | AdminService Projection prüfen |
| 4 | `srv/admin-constraints.cds` | 4–18 | Validierungs-Constraints prüfen |
| 5 | `srv/cat-service.js` | 9–23 | CatalogService Handler prüfen |
| 6 | `srv/admin-service.js` | 4–5 | AdminService Handler prüfen |

Tabelle 6 — Szenario „Feld hinzufügen“ – 6 Dateien, 6 Navigationsschritte

| # | Datei | Zeile | Aktion |
|---|---|---|---|
| 1 | `srv/cat-service.cds` | 3 | Service-Definition identifizieren |
| 2 | `srv/cat-service.cds` | 18–19 | `@requires`-Annotation und betroffene Action |
| 3 | `db/schema.cds` | 4–29 | Exponierte Daten auf Sensibilität prüfen |
| 4 | `srv/cat-service.js` | 14–23 | Handler auf Auth-abhängige Logik prüfen |

Tabelle 7 — Szenario „Auth anpassen“ – 3 Dateien, 4 Navigationsschritte

| # | Datei | Zeile | Aktion |
|---|---|---|---|
| 1 | `srv/cat-service.js` | 14 | `submitOrder`-Handler finden |
| 2 | `srv/cat-service.js` | 15–22 | Handler-Logik analysieren |
| 3 | `srv/cat-service.cds` | 19 | Action-Definition und Parameter prüfen |
| 4 | `db/schema.cds` | 4–13 | Entity `Books` mit `stock`-Feld verstehen |

Tabelle 8 — Szenario „Handler debuggen“ – 3 Dateien, 4 Navigationsschritte