



UNIVERSITÄT ZU LÜBECK

Abbilden und Nutzen von Versionsübergreifenden Medizinischen Klassifikationen mittels FHIR ConceptMaps

Visualization and Mapping of Medical Classifications across Multiple Versions with FHIR ConceptMaps

Masterarbeit

verfasst am

Institut für Medizinische Informatik

im Rahmen des Studiengangs

Medizinische Informatik

der Universität zu Lübeck

vorgelegt von

Simon Müller

ausgegeben und betreut von

Prof. Dr. Josef Ingenerf

mit Unterstützung von

Tessa Ohlsen, M. Sc.

Lübeck, den 7. November 2024

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Simon Müller

Zusammenfassung

Das Bundesinstitut für Arzneimittel und Medizinprodukte veröffentlicht jedes Jahr neue Versionen der Kodiersysteme ICD-10-GM und OPS. Die Änderungen in den Codes erschweren die versionsübergreifende Analyse klinischer Daten. Um diesem Problem entgegen zu wirken, wurde ein Software-Projekt entwickelt, welches aus drei Komponenten besteht. Erstens: Einem Datenintegrationsprozess, der möglichst automatisiert die Dateien der ICD-10-GM und OPS Versionen einliest. Zweitens: Suchalgorithmen, welche die Überleitungen von Codes über alle Versionen ermitteln. Drittens: Eine Web-Applikation, die zur Visualisierung der Daten und dem Generieren von FHIR ConceptMaps dient. Als Ergebnis wird die interoperable und versionsübergreifende Weiterverwendung der ursprünglich in Tabellen –pro Überleitung zwischen je zwei Versionen– veröffentlichten Daten ermöglicht.

Abstract

The German Federal Institute for Drugs and Medical Devices (BfArM) publishes new versions for the code systems ICD-10-GM and OPS every year. These changes in the codes impede the analysis of clinical data across multiple versions. To remedy this issue, a software project was implemented, consisting of three components. First: A data integration process, which aims to process the ICD-10-GM and OPS files in an automated manner. Second: Search algorithms, which determine the transitions of codes across all versions. Third: A web application, which visualizes the data and generates FHIR ConceptMaps. As a result, this approach facilitates the interoperable usage of BfArM data across all ICD-10-GM and OPS versions.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Grundbegriffe	1
1.2	Verwandte Arbeiten	6
1.3	Aufbau & Beiträge dieser Arbeit	9
2	BfArM-Daten	10
2.1	Kode- und Umsteiger-Dateien	10
2.2	Datenintegrationsprozess	15
2.3	Datenvorverarbeitung	17
3	Umsteiger-Suche	22
3.1	Allgemeine Funktionen	22
3.2	Transitive Hülle	26
3.3	Vergleich der Suchalgorithmen	38
3.4	Schreiben der FHIR ConceptMap	39
4	Web-Applikation	40
4.1	Überblick	40
4.2	Aufbau des bfarmer -Projekts	44
4.3	Backend	46
4.4	Frontend	48
4.5	AJAX	49
5	Zusammenfassung	51
5.1	Ergebnisse	51
5.2	Zusammenfassung	54
5.3	Ausblick	55
	Literatur	57
A	Appendix	63
A.1	Abweichungen zwischen ICD-10-GM und OPS Versionen	63
A.2	Beispielergbnis der Horizontalen Suche	69
A.3	Beispielergbnis der Vertikalen Suche / ConceptMap	71

1

Einleitung

Das Bundesinstitut für Arzneimittel und Medizinprodukte, kurz: BfArM, veröffentlicht jedes Jahr aktualisierte Versionen medizinischer Klassifikationen, wie zum Beispiel der Kodiersysteme ICD-10-GM und OPS. Die jährlichen Veröffentlichungen enthalten Überleitungstabellen von der neuen Version auf die jeweilige Vorgängerversion. Diese Änderungen in den Codes erschweren die versionsübergreifende Analyse klinischer Daten, wie sie beispielsweise vom Deutsches Forschungsdatenportal für Gesundheit (Medizin-informatik-Initiative, o. D.) zur Verfügung gestellt werden. Ein technischer Ansatz das Code-Mapping-Problem zu lösen wäre die Verwendung von *ConceptMaps* aus dem FIHR Standards für den Austausch elektronischer Gesundheitsdaten.

1.1 Grundbegriffe

Klassifikationen ICD-10-GM und OPS

Diese Arbeit bezieht sich auf die Systematischen Verzeichnisse von ICD-10-GM und OPS, das heißt der nach dem Code strukturierten Veröffentlichungen der Kodiersysteme.

ICD-10-GM

Zitiert aus (Gaus, 2005, Seite 97):

Die International Statistical Classification of Diseases (ICD) geht auf das Jahr 1855 zurück [...] und wird heute von der World Health Organisation (WHO) betreut. Die 10. Revision trat am 01.01.1993 in Kraft unter der Bezeichnung [...] ICD-10. Die deutschen Ausgaben werden vom Deutschen Institut für Medizinische Dokumentation und Information (DIMDI) als dem WHO Kooperationszentrum erarbeitet. Die am 01.01.2005 in Kraft getretene Ausgabe heißt [...] ICD-10-GM 2005, wobei GM für German Modification steht.

Seit Mai 2020 ist das DIMDI in das BfArM eingegliedert. (BfArM, o. D.[c])

1 Einleitung

Wiederum (Gaus, 2005, Seite 98ff):

Die ICD-10-GM 2005 ist ein Ordnungssystem für *Krankheiten (Diagnosen)*, das nach dem *Ordnungsprinzip Klassifikation* aufgebaut ist und insgesamt ca. 64 000 Klassen hat. Die *Notation* ist vier- oder fünfstellig. Sie besteht aus einem Buchstaben, einer zweistelligen Zahl, einem Punkt (dient nur der Strukturierung, liefert keine Information und wird deshalb bei der Stellenzahl nicht mitgezählt) und dann noch eine (vierstellige Notation) oder zwei (fünfstellige Notation) einstellige Zahlen. [...]

Die ICD-10-GM 2005 ist hierarchisch geordnet. Bei der Notation bezeichnen die 3 Stellen vor dem Punkt ein hierarchisches Niveau, das Raum für 2 500 gleichgeordnete Klassen bietet und nur durch Überschriften weiter gegliedert ist. Die Stellen nach dem Punkt geben weitere Niveaus an, d.h. die vierstellige Notation beschreibt 2, die fünfstellige Notation 3 hierarchische Niveaus. [...]

Ein allgemeines Problem der systematischen Anordnung von Diagnosen ist, ob eine Diagnose unter dem Ort ihrer Manifestation oder unter dem ihr zugrunde liegenden Krankheitsprozess eingeordnet werden soll. *Beispiel*: Soll die Lungenentzündung unter der Lokalisation Lunge oder unter dem Krankheitsprozess Entzündung eingeordnet werden? Werden alle Krankheiten eines bestimmten Organs nebeneinander gestellt, so folgt die Systematik dem topologisch-organspezifischen Aspekt (*Topologie* = Lehre von der Lage und Anordnung der Dinge im Raum). In einer Systematik können jedoch auch alle Krankheiten mit dem gleichen Krankheitsprozess, z.B. alle Entzündungen, alle Autoimmunkrankheiten oder alle bösartigen Neubildungen, nebeneinander gestellt werden. Diese Einteilung nennt man ätiologisch, pathologisch oder nosologisch (*Ätiologie* = die Krankheit auslösende Ursache, *Pathologie und Nosologie* = Lehre von den Krankheiten). Dieses Problem ist mit dem Ordnungsprinzip Klassifikation nicht lösbar. Die [ICD-10-GM] benutzt im Wesentlichen den ätiologischen Aspekt.

Definition des Begriffs Klassifikation aus (Gaus, 2005, Seite 86f):

Von allen Ordnungsprinzipien ist die Klassifikation das einfachste. Es beruht auf dem Grundsatz: „Jedes Ding (jeder Sachverhalt) an seinen Platz“. Das zu dokumentierende Sachgebiet wird in einzelne *getrennte Sachverhalte* eingeteilt, die man als *Klassen* bezeichnet. [...] Die Klassen sind *disjunkt*, d.h. sie schließen sich gegenseitig aus und überlappen sich nicht. Jede Klasse wird durch einen Deskriptor repräsentiert. Die Klassen einer Klassifikation sind gleichzeitig *Äquivalenzklassen* von Begriffen. Im strengen Fall ist die Zuordnung einer Dokumentationseinheit zu einer Klasse eindeutig, d.h. eine Dokumentationseinheit wird genau einer Klasse zugeteilt. Eine Klassifikation ist einfach und praktisch. Sie ist sozusagen das „*natürliche Ordnungsprinzip*“. [...]

Ein *Klassifikationssystem* – ein Ordnungssystem, das nach dem Ordnungsprinzip Klassifikation aufgebaut ist – muss vollständig sein. Vollständig sein bedeutet, dass die Klassen alle Sachverhalte des dokumentarisch zu bearbeitenden Sachgebiets umfassen. [...] Durch Schaffung einer Klasse „*sonstiges*“ oder besser durch die Schaffung mehrerer Klassen mit dem Zusatz „*sonstiges*“ wird die geforderte Vollständigkeit des Klassifikationssystems [...] formal erreicht. [...] Um auf die einzelnen Klassen bequem und sicher zugreifen zu können, werden sie *hierarchisch* oder anderweitig *systematisch* angeordnet.

Vom BfArM werden Notation und Begriff bezeichnet als Kode und Titel.

OPS

Ebenfalls zitiert aus (Gaus, 2005, Seite 101ff):

In der Medizin werden nicht nur Krankheiten, sondern auch ärztliche Tätigkeiten dokumentarisch erfasst und – ebenso wie die Diagnosen für die Abrechnung und für wissenschaftliche Zwecke verwendet. Die von der WHO erstmals 1978 herausgegebene International Classification of Procedures in Medicine (ICPM) wurde (ebenso wie die ICD-10) vom DIMDI ins Deutsche übertragen und an deutsche Verhältnisse angepasst. Die am 01.01.2005 in Kraft getretene Ausgabe heißt „Operationen- und Prozedurenschlüssel – Internationale Klassifikation der Prozeduren in der Medizin (OPS 2005)“. Der OPS 2005 ist – wie die Bezeichnung „Schlüssel“ ausdrückt – eine Klassifikation mit numerischer Notation. Allerdings haben in einigen Bereichen die 10 gleichgeordneten Klassen nicht ausgereicht, deshalb treten an der 5. und 6. Stelle der Notation gelegentlich Buchstaben auf. Von den insgesamt 14 000 Klassen betreffen etwa 70% Operationen, ca. 15% nichtoperative therapeutische Maßnahmen, 8% diagnostische Maßnahmen, 4% bildgebende Diagnostik und 2% ergänzende Maßnahmen.

Die *Notation* des OPS 2005 ist vier- bis sechsstellig. Die erste Stelle enthält eine 1, 3, 5, 8 oder 9 und bezeichnet einen der eben genannten Bereiche (z.B. 1 = diagnostische Maßnahmen). Es folgt ein Bindestrich und eine dreistellige Zahl. Je nach Detaillierungsgrad ist damit die Notation beendet oder es folgen noch ein Punkt und weitere 1 bis 2 Stellen. [...] In der fünften und sechsten Stelle der Notation (d.h. nach dem Punkt) können Ziffern durch die Buchstaben x oder y ersetzt werden, es bedeutet x = sonstige und y = nicht näher bezeichnet.

FHIR ConceptMap

FHIR steht für „Fast Healthcare Interoperability Resources“.

HL7 FHIR

Zitiert aus (Heckmann, 2022, Seite 309):

„Health Level 7“ wurde 1987 gegründet, um Standards für klinische Informationssysteme zu erarbeiten. [...] HL7 ist vom nationalen amerikanischen Normeninstitut (ANSI) seit 1994 akkreditiert. HL7-verbundene Organisationen existieren inzwischen in über 40 Ländern. Die erste nationale Partnergesellschaft wurde 1993 in Deutschland gegründet. Die Aufgabe dieser sogenannten „Affiliates“ ist es, die Verbreitung und Implementierung der internationalen HL7-Standards in dem jeweiligen Land zu fördern und gegebenenfalls erforderliche Anpassungen und Erweiterungen für deren Nutzung zu entwickeln. FHIR ist die dritte Generation von Interoperabilitätsstandards aus der Feder von HL7.

Die Entwicklung begann im Jahre 2011 als Reaktion auf die Forderungen aus der Industrie nach einer standardisierten Lösung für die Entwicklung webbasierter Applikationen für das Gesundheitswesen. [...] Die vierte, 2018 veröffentlichte Version „R4“ enthält erstmals normative Inhalte.

Ebenfalls aus (Heckmann, 2022, Seite 310):

Im Gegensatz zu älteren HL7-Standards [...] bedient sich FHIR erstmals moderner web-basierter Technologien, die auch jenseits des Gesundheitswesens weit verbreitet und bei Entwicklern bekannt und beliebt sind. FHIR legt den Fokus auf die query-getriebene Art der Kommunikation, die dazu dient, Anwendungen und Anwendern die benötigten Daten zum benötigten Zeitpunkt in adäquatem Umfang und Format zur Verfügung zu stellen. Dazu bedient sich FHIR dem Representational State Transfer (kurz: „REST“), einem Paradigma, das die Struktur und das Verhalten des World Wide Webs abstrahiert und basierend auf denselben Technologien und Grundlagen eine einheitliche Form der Kommunikation von verteilten Systemen und Webservices definiert. Die auszutauschenden Datenobjekte werden als „Ressourcen“ bezeichnet, auf denen jeweils sogenannte „CRUD“-Interaktionen (create, read, update, delete) über die HTTP-Methoden GET, PUT, POST und DELETE ausgeführt werden können.

Die Autorin hebt außerdem folgende Eigenschaften hervor:

- „FHIR als Baukasten“: Standards und Leitfäden in FHIR sind selbst Ressourcen. Sie können erweitert und für verschiedene Zwecke eingesetzt werden.
- „FHIR als Community-Standard“: Die FHIR-Spezifikation ist frei verfügbar und wird gemeinschaftlich entwickelt, statt von einem geschlossenen Gremium.

Die am meisten verbreitetsten Implementationen von FHIR sind der als freie Software lizenzierte API-Server *HAPI*, sowie der proprietäre Terminologie-Server *Ontoserver*, welcher unter anderem im australischen Gesundheitssystem eingesetzt wird. (Braunstein, 2022)

REST

Representational State Transfer ist –wie oben erwähnt– ein Paradigma für die Software-architektur von verteilten System, welches erstmals in (Fielding, 2000) dargelegt wurde. Es besteht aus sechs Bedingungen:

1. Kommunikation per Client-Server-Modell.
2. Zustandslosigkeit: Jede Nachricht zwischen Client und Server muss alle Information enthalten, die zum Verständnis der Nachricht benötigt wird.
3. Erhöhung der Effizienz durch Einsatz von Caching.
4. Einheitliche Schnittstellen:
 - 4.1 Jede Ressource wird durch eine URL identifiziert.
 - 4.2 Ressourcen werden verändert durch Repräsentationen, zum Beispiel strukturierte Dateiformate.
 - 4.3 Die Inhalte der Nachrichten sind selbstbeschreibend.
 - 4.4 Informationen werden vom Anwendungsserver dynamisch bereitgestellt über Hypermedien wie HTTP.

5. Mehrschichtigkeit: Komplexität wird hinter den offenen Endpunkten verborgen.
6. „Code on Demand“: Es soll die Möglichkeit für Clients bestehen, Verfahren zur Verwendung neuer Medientypen vom Server abfragen zu können.

ConceptMap

Die ConceptMap ist eine FHIR Ressource, die zum Mapping zwischen Kodiersystemen vorgesehen ist. Die Definition ist frei verfügbar, zum Beispiel für FHIR Release 4 unter (HL7, 2019, FHIR R4). Die Komponenten, die für ein Mapping zwischen Versionen der ICD-10-GM und des OPS mindestens benötigt werden, sind folgende:

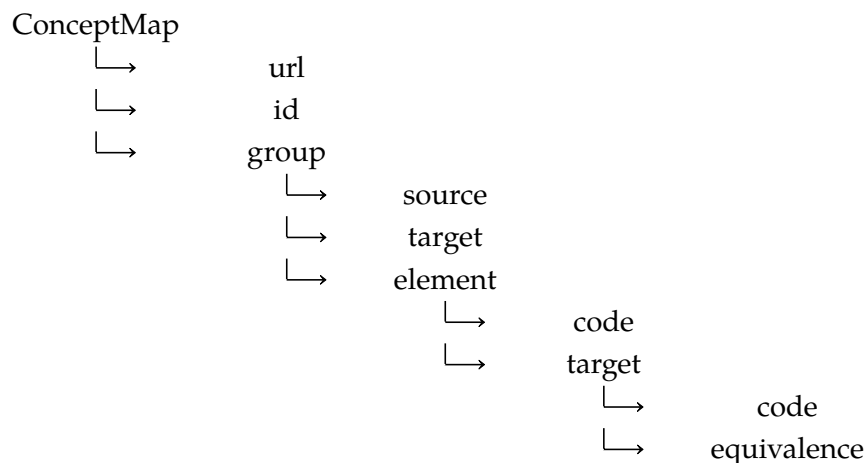


Abbildung 1.1: Hierarchischer Aufbau der FHIR Ressource „ConceptMap“.

- *url*: Eine absolute URI, *Uniform Resource Identifier*, welche einmal pro ConceptMap enthalten ist und diese global eindeutig identifiziert. Idealerweise handelt es sich um eine existierende URL. Laut Spezifikation ist die Angabe eigentlich optional, aber sie wird von HAPI vorausgesetzt.
- *id*: Die logische ID der Ressource; das heißt sie ist innerhalb eines Servers eindeutig. Sie kann nicht verändert werden und ist wie die URL pro ConceptMap einmalig eingetragen, sowie optional. Aber es ist natürlich sinnvoll die ID dafür zu verwenden, Zweck und Umfang der ConceptMap zu kennzeichnen.
- *group*: Eine Gruppe an Mappings, die alle das selbe Start- und Zielsystem haben. Eine ConceptMap kann mehrere Gruppen enthalten.
- *source*: Eine absolute URI, die das Start- beziehungsweise Ursprungssystem der Konzepte für das Mapping identifiziert. Sie kann einmal pro *group* angegeben werden. Bei einer in sich geschlossenen ConceptMap, das heißt ohne Referenz auf eine FHIR „Code System“ Ressource, kann dieser Eintrag auch frei gewählt sein.
- *target*: Wie *source* nur für das Zielsystem.
- *element*: Mappings für ein einzelnes Konzept der Ursprungsversion auf ein oder mehrere Konzepte der Zielversion. Pro *group* sind mehrere *element*-Einträge möglich.

- *code*: Ein Eintrag pro *element*, der dieses identifiziert.
- *target*: Konzept(e) der Zielversion, für das Mapping von *element*.
- *code*: Kode, der das Zielkonzept identifiziert.
- *equivalence*: Die Relation zwischen Konzepten der Ursprungs- und Zielversion.
In FHIR R5 heißt diese Komponente *relationship*.

Wie die Relationen angegeben werden, unterscheidet sich zwischen FHIR Release 4 und 5 (HL7, 2023). Release 6 wird hierbei unverändert zu R5 bleiben. Für die Überleitungen zwischen den Versionen der ICD-10-GM und des OPS gibt es vier Fälle:

1. Es gibt keine Veränderung. R4 und R5: *equivalent*.
2. Der Kode wurde verändert. R4: *relatedto*. R5: *related-to*.
3. Die Überleitung erfolgt auf mehrere mögliche Kodes. R4: *wider*.
R5: *source-is-narrower-than-target*.
4. Der Kode wurde entfernt. R4: *unmatched*.
In R5 ist kein *target*-Eintrag mehr notwendig, sondern es wird stattdessen auf der gleichen Ebene wie der *code* eine Komponente *noMap* mit Wert *true* eingetragen.

Zusätzlich gibt es noch die Komponente *unmapped* pro *group*. Hier kann angegeben werden, wie Konzepte behandelt werden sollen, die zwar in dem unter *source* referenzierten Kodiersystem enthalten sind, für die es allerdings keinen Eintrag im Mapping als *element*→*code* gibt. Da bei den Überleitungen zwischen den Versionen der ICD-10-GM und des OPS meistens die große Mehrheit der Kodes unverändert bleibt, wäre es also sinnvoll, um die Größe der ConceptMap gering zu halten, alle Kodes mit *equivalent*-Relation gar nicht einzutragen und stattdessen den *unmapped*-Modus „provided“ zu verwenden, der diese Kodes einfach wie angegeben als „Ziel“ des Mappings zurückgibt, was ebenfalls der Äquivalenz entspricht. Leider wird aber *unmapped* in der aktuellen HAPI-Version nicht unterstützt und von Ontoserver nur im beschränkten Maße: (Lawley, 2023).

1.2 Verwandte Arbeiten

In (Hund u. a., 2016) wird die Verwendung der ICD-10-GM und des OPS im deutschen Gesundheitssystem diskutiert, sowie der Nutzung die daraus resultierenden Daten in der Forschung. Als eine der Herausforderungen wurde dabei der Vergleich über die verschiedenen, jährlich veröffentlichten Versionen identifiziert. Um diese zu lösen wurde eine Software-Library entwickelt: (Hund, 2018). Darin ist das Ermitteln von Überleitungen ausgehend von jeweils einem Kode in rückwärts chronologischer Reihenfolge implementiert. Allerdings war das letzte Update von 2020 und das Hinzufügen neuer Versionen erfordert die Programmierung mehrerer Klassen pro Version. Außerdem setzt das Kompilieren der Library das Vorhandensein von Dateien in einer Größe von circa 1 GB voraus und das Laden der Daten pro Applikationsstart ist ein relativ langer Prozess.

Im US-amerikanischen Gesundheitssystem erhält die ICD-10-CM, Clinical Modification, ebenfalls jedes Jahr eine neue Version. Zum Beispiel in (Manchikanti u. a., 2015) werden die jährlichen Mehrkosten des Einsatzes im Vergleich zu ICD-9-CM betont. Programmatische Ansätze zur Überleitung zwischen den Versionen innerhalb der ICD-10-CM sind allerdings nicht bekannt. Es gibt jedoch Mappings zwischen ICD-9-CM und ICD-10-CM, wie zum Beispiel in (Wu u. a., 2019).

FHIR Mappings

Seit Veröffentlichung der FHIR Release 4 in 2019 werden bevorzugt ConceptMap Ressourcen für Mappings im klinischen Kontext verwendet. Interessant ist beispielsweise die Verwendung mehrerer *group*-Einträge in (Saripalle, 2019), um UMLS in FHIR ConceptMaps abzubilden. Das *Unified Medical Language System* ist ein Ansatz, mittels eines Kompendiums kontrollierter Vokabulare zwischen medizinischen Terminologien zu übersetzen. Einen ähnlichen Zweck verfolgt das *Observational Medical Outcomes Partnership* (OMOP) *Common Data Model* (CDM) der *Observational Health Data Sciences and Informatics* (OHDSI) Initiative. Ein Beispiel für die Verwendung dessen in einer ConceptMap ist (Essaid u. a., 2024). Eine weitere Vorgehensweise wie in (Ohlsen u. a., 2022) ist wiederum ein Mapping über SNOMED CT, wobei *Systematized Nomenclature of Medicine Clinical Terms* die umfassendste medizinische Terminologie ist.

SNOMED CT als Interlingua

Basierend auf dem letzten Ansatz wäre also eine Möglichkeit das Mapping zwischen Versionen der ICD-10-GM und des OPS über SNOMED CT als Interlingua zu gestalten. Es wurden für ICD-10-GM ↔ SNOMED CT Mappings in beide Richtungen bereits elektronische Ressourcen entwickelt. Zum Beispiel gibt es I-MAGIC von der *National Library of Medicine* (NIH, o. D.) um differenzierte Mappings von SNOMED CT nach ICD-10-CM interaktiv zu erstellen. SNOMED International stellt einen Browser (SNOMED, o. D.[a]) zur Verfügung, der mit Expression Constraint Language Queries für ICD-10 Codes die Abfrage der mehreren zutreffenden SNOMED CT Konzepte ermöglicht. Das Mapping Tool (SNOMED, o. D.[b]) funktioniert je nach Eingabe in beide Richtungen.

Beispielsweise werden für den ICD-10-CM M21.4, „Flat foot [pes planus] (acquired)“ folgende SNOMED CT IDs zurückgegeben:

SCTID	Begriff
203533003	Peroneal spastic flat foot
90374001	Acquired spastic flat foot
53226007	Talipes planus
44480001	Acquired flexible flat foot
203531001	Hypermobile flat foot
203534009	Acquired pes planus
203532008	Rigid flat foot

Für die andere Richtung existiert eine 1:1 Kardinalität. (WHO-FIC, 2021)

Ein Ansatz die für ein Mapping am besten geeignetste SCTID zu ermitteln wird in (Philipp u. a., 2022) vorgestellt. SNOMED CT Konzepte sind hierarchisch in |is-a|-Relationen angeordnet und so wird hier das „passendste“ Konzept dem ranghöchsten gleichgesetzt. Im Beispiel oben ist 90374001–„Acquired spastic flat foot“ ein Kind von 53226007–„Talipes planus“. Letztere und 203534009–„Acquired pes planus“ sind allerdings gleich ranghoch.

Eine andere Idee wäre das Mapping anhand der Ähnlichkeit von Begriff in SNOMED CT und Titel in ICD-10 zu berechnen. Dazu gibt es Algorithmen wie (Winkler, 1990), welcher für das M21.4–„Flat foot [pes planus] (acquired)“–Beispiel „Acquired pes planus“ als beste Übereinstimmung ermittelt und (Gotoh, 1982) „Talipes planus“.

Es wäre also möglich geeignete Konzepte für ein Mapping vorzuschlagen, aber die endgültige Auswahl müsste medizinisches Fachpersonal treffen. In (Vikström u. a., 2007) wurde ein Mapping der schwedischen ICD-10 Version auf SNOMED CT manuell durchgeführt. Die Urteilerübereinstimmung der beiden Kodierer betrug 89%, was als moderates Ergebnis bewertet wird.

Ein weiteres Problem ist, dass der Vergleich der deutschen Titel dadurch erschwert wird, dass die Übersetzung der deutschen SNOMED CT Version noch nicht weit fortgeschritten ist; siehe (Heidel, Hoffmann und Ammon, 2024). Ähnlich ernüchternd sind Ergebnisse von Mappings zwischen OPS und SNOMED CT wie in (Schulz u. a., 2019).

In dieser Arbeit wird also ein Mapping zwischen den verschiedenen Versionen innerhalb der ICD-10-GM und dem OPS auf Basis der vom BfArM veröffentlichten Überleitungstabellen vorgenommen. Sobald ein operativer Reifegrad der Mappings zwischen diesen Kodiersystemen und SNOMED CT erreicht ist, wäre es wahrscheinlich immer noch hilfreich die Versionen innerhalb von ICD-10-GM und OPS auf zum Beispiel die aktuellste Version anzugleichen.

Qualitätsstandards für Mappings

Es folgt eine Zusammenfassung verschiedener Grundprinzipien und Maßstäbe für das Erstellen von Mappings im Gesundheitswesen. In eckigen Klammern sind die Listennummern in den jeweiligen Quellen referenziert:

A „Classifications and Terminology Mapping – Principles and Best Practice“, World Health Organization, *Arbeitsgruppe: Family of International Classifications*. (WHO-FIC, 2021)

B „Principles of Mapping between Terminological Systems“, International Organization for Standardization, *Komitee: Health Informatics*. (ISO, 2014)

C „Terminology Resource Map Quality Measures“, International Organization for Standardization, *Komitee: Health Informatics*. (ISO, 2019)

1. Einsatzszenarien müssen vor Entwicklung des Mappings etabliert sein. Zweck, Umfang und die Richtung müssen klar definiert sein. Mappings sollten jeweils eine Richtung und einen Zweck haben. [A1, A2, A3, B1, B2, B9]

2. Die Urheber des Kodiersystem sollten in der Erstellung und Validierung des Mappings beteiligt sein. Mit dem Kodiersystemen verbundenen Konventionen und Richtlinien sollten eingehalten werden. [A5, B6, B7]
3. Das Team sollte nach den vorhandenen Kompetenzen strukturiert sein. Es ist förderlich, wenn ein Interesse externer Organisationen besteht. [A6, B5, B20, C9, C15]
4. Dokumentationen über Zweck, Umfang, Einschränkungen und Implementierung des Mappings müssen erstellt werden. Bei automatisierten, beziehungsweise maschinell erstellten Mappings müssen Informationen über die verwendeten Technologien und Algorithmen bereitgestellt werden. Es ist wichtig die Ergebnisse manuell zu überprüfen. [A4, A11, B8, B12, B19, C7, C10, C14]
5. Qualitätssicherung und Validierung sind essentiell; Testprotokollen müssen existieren und zukünftige Anwender des Mappings sollten involviert sein. [A7, B4, B14, B15, C11, C12]
6. Die Veröffentlichung des Mappings muss Informationen über Versionen, Update-Zyklen und Lizenzen enthalten. Ein Wartungsprozess muss definiert sein. Die Weiterentwicklung sollte durch Feedback von Anwendern gewährleistet sein und über einen Konsensbildung erfolgen. Mappings sollten möglichst aktuell sein. [A8, A9, B13, B16, B17, B18, C8, C13, C16, C17]
7. Kardinalität, Relationen, sowie Ursprung und Ziel des Mappings müssen klar definiert sein. [A12, A13, B10, B11, C4, C5, C6]
8. Mappings sollten maschinenlesbar sein. [A14, B3]

Für diese Arbeit nicht relevant sind Punkte zu manuell erstellten Mappings [A10], spezifischen Kodiersystemen wie ICD-11 [A15], Eigenschaften von Ursprung- und Zielsystem, falls diese abweichen [C1, C2], Übersetzung natürlicher Sprachen [C3].

1.3 Aufbau & Beiträge dieser Arbeit

Die Arbeit ist in drei Teile untergliedert:

1. Ein Integrationsprozess für die BfArM-Daten, der für ICD-10-GM und OPS möglichst gleich funktioniert und die Aufnahme einer neuen Version möglichst einfach macht.
2. Zwei Suchalgorithmen, um Überleitungen der Codes versionsübergreifend und chronologisch in beide Richtungen zu bestimmen.
3. Eine Web-Applikation, welche die Ergebnisse der vorherigen zwei Punkte anzeigt und FHIR ConceptMaps für das versionsübergreifende Mapping von ICD-10-GM und OPS generiert.

2

BfArM-Daten

Dieses Kapitel behandelt die elektronische Verarbeitung der BfArM-Daten.

2.1 Kode- und Umsteiger-Dateien

Das BfArM stellt für jede neue ICD-10-GM und OPS Version Dateien für die Überleitung auf die Vorgänger-Version zum Download zur Verfügung, gebündelt jeweils in einer Zip-Datei: (BfArM, o. D.[b], Downloads).

Es handelt sich hierbei um CSV-formatierte Text-Dateien. „Comma-Separated Values“ ist ein sehr einfaches Format, um Daten zu strukturieren. Es wird ein Satzzeichen verwendet, um den restlichen Text in Spalten zu trennen – laut dem Namen normalerweise ein Komma, aber für die BfArM-Dateien wurde Strichpunkt als Trennzeichen gewählt, wahrscheinlich weil die Klassentitel auch Kommata enthalten können. Weitere Informationen zum CSV Datei-Format finden sich in (Bonney u. a., 2024, Seite 131f).

Vom BfArM werden Codes als Schlüsselnummern bezeichnet, wenn diese eindeutig sind und einzelne Überleitungen zwischen Codes werden Umsteiger genannt.

Kodes / Schlüsselnummern

Hier beispielhaft die ersten sieben Zeilen der Kode-Datei aus der ICD-10-GM 2024:

```
UNDEF;Undefined
A00;Cholera
A00.0;Cholera durch Vibrio cholerae 0:1, Biovar cholerae
A00.1;Cholera durch Vibrio cholerae 0:1, Biovar eltor
A00.9;Cholera, nicht näher bezeichnet
A01;Typhus abdominalis und Paratyphus
A01.0;Typhus abdominalis
```

Anmerkungen:

- Ein Strichpunkt = zwei Spalten
 1. Kode
 2. Klassentitel
- Für OPS ist das Format der Kode-Datei identisch.
- Mit Ausnahme des UNDEF-Eintrags in der ersten Zeile ist die Datei alphabetisch nach dem Kode sortiert. UNDEF ist kein ICD-10-GM- oder OPS-Kode, sondern wird für Umsteiger verwendet, um entfernte beziehungsweise neu hinzugefügte Kodes zu kennzeichnen.
- Die Datei enthält nicht-endständige Kodes – im Beispiel oben A00 und A01. Ein Kode ist endständig, wenn er keine Subkategorie hat, siehe (BfArM, o.D.[d], Kategorie und Kode in der ICD-10-GM).

Umsteiger / Überleitungen

Im Gegensatz zu den Kodes haben die Umsteiger für ICD-10-GM und OPS unterschiedliche Formate. Hier also zuerst zwei Ausschnitte aus der Umsteiger-Datei für die ICD-10-GM 2017 Überleitung auf Version 2016:

A00.0;A00.0;A;A
A00.1;A00.1;A;A
A00.9;A00.9;A;A
A01.0;A01.0;A;A
A01.1;A01.1;A;A

U06.0;UNDEF;A;
UNDEF;Z99.0;;
Z99.0;Z99.0;A;

Anmerkungen:

- Drei Strichpunkte = vier Spalten
 1. Alter Kode (2016)
 2. Neuer Kode (2017)
 3. Wenn A: automatisch überleitbar von 2016 auf 2017, sonst kein Zeichen
 4. Wenn A: automatisch überleitbar von 2017 auf 2016, sonst kein Zeichen
- Der obere Abschnitt umfasst die fünf ersten Zeilen der Umsteiger-Datei.
- Der untere Abschnitt enthält beispielhaft zwei Umsteiger mit UNDEF. UNDEF als neuer Kode heißt der alte Kode wurde entfernt. UNDEF als alter Kode heißt der neue Kode wurde hinzugefügt. In diesem Beispiel wurde Z99.0 umbenannt.

- Die Datei ist alphabetisch nach dem alten Kode sortiert und falls dieser bei mehreren Einträgen identisch ist, anschließend nach dem neuen Kode.
- Es sind nur endständige Kodes enthalten.

Dazu im Vergleich ein einzelner Umsteiger aus dem OPS 2024, Überleitung auf Version 2023:

1-100;N;1-
100;N;A;A

Die zusätzlichen Spalten jeweils nach den Kodes speziell für den OPS sagen aus, ob Zusatzkennzeichen notwendig sind, siehe dazu auch: (BfArM, o. D.[e], Kategorie und Kode im OPS).

„DRY“-Prinzip

„Don't Repeat Yourself“ ist eines der Kardinalprinzipien in der Software-Entwicklung. Obwohl der Grundsatz, Wiederholungen zu vermeiden, wahrscheinlich schon in der Programmierung angewandt wird seit es diesen Beruf gibt, wurde „DRY“ erstmals 1999 formuliert von (Thomas und Hunt, 2019, Seite 79ff). In der zwanzigjährigen Jubiläumsausgabe verdeutlichen die Autoren, dass es ihnen hierbei nicht nur um das Schreiben von Programmcode geht, sondern vielmehr um die Intention hinter einem Prozess. Das heißt eine Änderung der Funktionalität in einer Software-Lösung sollte möglichst *nicht* mehrere Änderungen an mehreren Stellen nach sich ziehen.

Bei der Integration der BfArM-Daten kann das „DRY“-Prinzip auf zwei Arten angewandt werden.

1. Bezogen auf Kodiersysteme: Alle Funktionen sollten unabhängig davon anwendbar sein, ob es sich um ICD-10-GM- oder OPS-Daten handelt. Auch die Aufnahme eines zusätzlichen Systems, beispielsweise ATC, sollte möglichst nur Anpassungen erfordern, die durch Abweichungen in der Integration der Daten dieses Systems notwendig sind.
2. Bezogen auf Versionen: Unabhängig von der Version sollte der Prozess der Datenintegration gleich ablaufen und das gleiche Ergebnis liefern, bezogen auf die Datenstruktur. Jede Abweichung zwischen Versionen sollte nur eine möglichst einfach zu implementierende Modifikation des Gesamtprozesses darstellen. Konkret heißt das beim Hinzufügen einer neuen Version, dass an nur einer Stelle die Abweichungen von der Standardversion angegeben werden sollten und der Datenintegrationsprozess danach einmal angestoßen wird. Idealerweise ändert sich bei einer neuen Version nur die Versionsnummer und die Download-URL.

Standardverfahren und Abweichungen

Im diesem Abschnitt werden alle Abweichungen der ICD-10-GM- und OPS-Versionen von der als Standard gewählten Version 2024 in Tabellen aufgeführt. Konkret gemeint

sind damit: Version, Download-URL, Pfad der Kode- und Umsteiger-Dateien, Sonstiges. Diese Informationen können dem Datenintegrationsprozess in einem strukturierten Dateiformat zur Verfügung gestellt werden.

Liste aller Versionen

Im Anhang A.1 befinden sich Tabellen, die alle Abweichungen zwischen den Versionen für ICD-10-GM und OPS bezogen auf 2024 enthalten.

Datei-Adressen und -Pfade

Die Download-URL der Zip-Dateien setzt sich wie folgt zusammen:

`https://multimedia.gsb.bund.de/BfArM/downloads/klassifikationen/ ...`
& für ICD-10-GM: `icd-10/ ...`
& für OPS: `ops/ ...`
& einen pro Version unterschiedlichen Teil, siehe URL-Eintrag in den Tabellen.
Die URL dient damit als „Single Source of Truth“ (Bonney u. a., 2024, Seite 257).

Die Kode- und Umsteiger-Dateien sind in einem Verzeichnis enthalten:

Klassifikationsdateien

Wenn die Tabellen einen Verzeichnis-Eintrag enthalten, wird dieser vorangestellt.

Zum Beispiel für ICD-10-GM Version 2021:

`icd10gm2021syst-ueberl-20201111/Klassifikationsdateien`

Der Pfad der Kode-Datei lautet:

Verzeichnis `...`
& für ICD-10-GM: `icd10gm ...`
& für OPS: `ops ...`
& die Version `...`
& `syst.txt`

Also zum Beispiel: `Klassifikationsdateien/icd10gm2024syst.txt`

Wenn die Tabellen einen Codes-Eintrag enthalten, wird dieser stattdessen verwendet.

Die Umsteiger-Datei funktioniert ähnlich, nur dass der Dateiname normalerweise so ist:

für ICD-10-GM: `icd10gm ...`
für OPS: `ops ...`
& Version `...`
& `syst_umsteiger_` `...`
& Vorgänger-Version `...`
& `_` `...`
& Version `...`
& `.txt` Zum Beispiel: `ops2024syst_umsteiger_2023_2024.txt`

Sonstige Abweichungen

- Vorab-Version
Diese Version hat noch keine Seite für die Kode-Suche, siehe Abschnitt 4.5.
- Zip-Unterdatei
Die Zip-Datei der 2022 Versionen enthielt weitere Zip-Dateien. Vorher wurden alle Dateien zu einer Versionen nach Verwendungszweck nur in Unterverzeichnisse gegliedert, weswegen die gebündelte Zip-Datei insgesamt relativ groß wurde. Ab 2023 werden die Zip-Unterdateien separat zum Download angeboten.
- ISO-8859-1
Vor 2009 waren Dateien in ISO-8859-1 kodiert, auch Latin-1 genannt, statt UTF-8. Mehr dazu auf der nächsten Seite.
- Punkt-Strich-Notation, Kreuz-Stern-System
Die Codes der frühesten ICD-10-GM Versionen hatten Sonderzeichen gemäß (BfArM, o. D.[d], Kategorie und Kode in der ICD-10-GM).
- 6-Spalten-Umsteiger
Umsteiger älterer ICD-10-GM Versionen enthielten Informationen zur Mehrfachkodierung.
- Nicht endständige Umsteiger
Im Gegensatz zu allen anderen Überleitungen sind die Umsteiger-Einträge für die ICD-10-GM 2.0 auf 1.3. auch für nicht-endständige Kodes enthalten.
- None statt UNDEF
Von OPS 2009 bis 2004 wurde statt UNDEF der Bezeichner „None“ verwendet.
- KOMBI-Kode
OPS Versionen 2.1 und 2.0 enthalten in der Kodes-Datei einen zusätzlichen Eintrag: KOMBI, „Kombinationsschlüsselnummer erforderlich“.
- 6-Spalten-Umsteiger (altes Format), 5-Spalten-Umsteiger, 4-Spalten-Umsteiger
Die Umsteiger der OPS-Versionen von 2009 bis 2005 waren anders formatiert, weil 2005 die Informationen bezüglich Zusatzkennzeichen hinzukamen und bis 2009 die Spalten unterschiedlich angeordnet waren als in allen neueren OPS Versionen.
- 6-Spalten-Umsteiger (ursprüngliches Format)
Die Umsteiger für den OPS 2.1 enthielten zusätzliche Spalten wegen Mehrfachverschlüsselung wie die älteren ICD-10-GM Versionen.
- 3-Spalten-Umsteiger
OPS 2.0 zeigte mit nur einer Spalte an, ob automatische Überleitungen möglich sind.
- Keine Überleitung
Aus der ältesten Version, die Überleitungen enthält, wird zusätzlich die Kodes-Datei für die Vorgänger-Versionen verarbeitet.

2.2 Datenintegrationsprozess

Wie erwähnt durchlaufen alle Daten unabhängig von Version und Kodiersystem den gleichen Integrationsprozess. Dieser orientiert sich an dem klassischen „Extract-Transform-Load“ Modell, siehe (Bonnefoy u. a., 2024, Seite 247ff).

1. *Extract*: Die Daten werden in einem bestimmten Format aus einem Quell-System extrahiert.
2. *Transform*: In einem oder mehreren Prozessen werden die Daten in ein standardisiertes Format transformiert, was zum Beispiel Bereinigung, Validierung und Imputation (Generieren fehlender Daten) beinhalten kann.
3. *Insert*: Die Daten werden in ein Ziel-System integriert, um dort von weiteren Applikationen verwendet zu werden.

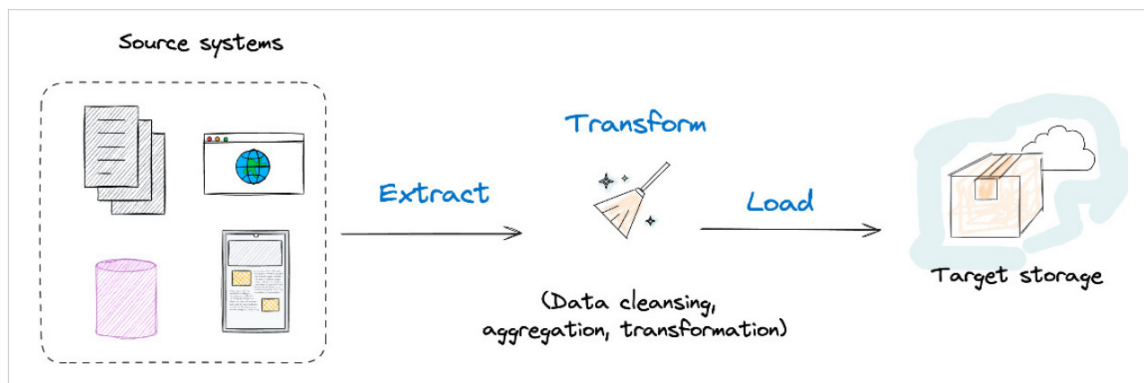


Abbildung 2.1: ETL-Modell nach (Bonnefoy u. a., 2024, Seite 63)

Für die BfArM-Daten sieht der Integrationsprozess konkret so aus:

1. *Download*: Die Zip-Dateien werden heruntergeladen. Alternativ kann geprüft werden, ob die Dateien schon lokal vorhanden sind mit einem bestimmten Pfad, der sich nach Kodiersystem und Versionsnummer immer gleich zusammensetzt, zum Beispiel: (Projektverzeichnis)/files/icd10gm2024.zip. Die Download-Funktion sollte die Zip-Dateien ebenfalls unter diesem Pfad abspeichern, falls so gewünscht.
2. *Unzip*: Die Kodes- und Umsteiger-Dateien werden aus der Zip-Datei extrahiert. Normalerweise muss dafür nicht das ganze Archiv in temporäre Dateien entpackt werden – außer eventuell bei den Versionen 2022, weil das Extrahieren verschachtelter Zip-Dateien eher eine Nischenanwendung ist und nicht unbedingt standardmäßig von Programmiersprachen oder Bibliotheken unterstützt wird.
3. *Convert Encoding*: Die in ISO-8859-1 kodierten Kodes-Dateien müssen in UTF-8 umgewandelt werden. In (Fernández und Manuel, 2022) werden die beiden Zeichenkodierungen genauer erklärt, aber für die BfArM-Daten ist eigentlich nur relevant, dass Umlaute mit unterschiedlichen Werten kodiert sind. Also würde das Einlesen eines in ISO-8859-1 kodierten Umlauts als UTF-8 ein anderes Zeichen als Resultat ergeben.

2 BfArM-Daten

Die Umsteiger-Dateien sind davon nicht betroffen, weil in diesen keine Umlaute enthalten sind.

4. *Parse CSV*: Ein Parser wandelt eine Datei in eine Datenstruktur um; für CSV sollte jede Programmiersprache so eine Funktion standardmäßig zur Verfügung stellen. Für die BfArM-Dateien ist das Ergebnis ein zweidimensionales Array mit zwei Spalten für die Codes, beziehungsweise drei bis sechs Spalten für die Umsteiger je nach Kodiersystem und Version.
5. *Process Data*: Aufgrund der oben erwähnten Abweichungen ist die Vorverarbeitung der Daten der komplexeste Schritt und wird im nächsten Abschnitt genauer erklärt. Außerdem müssen nicht alle Daten gespeichert werden. Vor allem in Bezug auf die Zip-Dateien ergibt das eine Reduktion der Datenmenge um etwa einen Faktor von zehn.
6. *Insert*: Die bearbeiteten Daten werden für die Verwendung durch Applikationen gespeichert. Zum Beispiel für eine relationale Datenbank werden pro Dateityp, Kodiersystem und Version eine Tabelle angelegt und die Daten in diese geschrieben. Konkret für SQL müssen außerdem die Hochkommata in den Codes-Dateien beachtet werden.

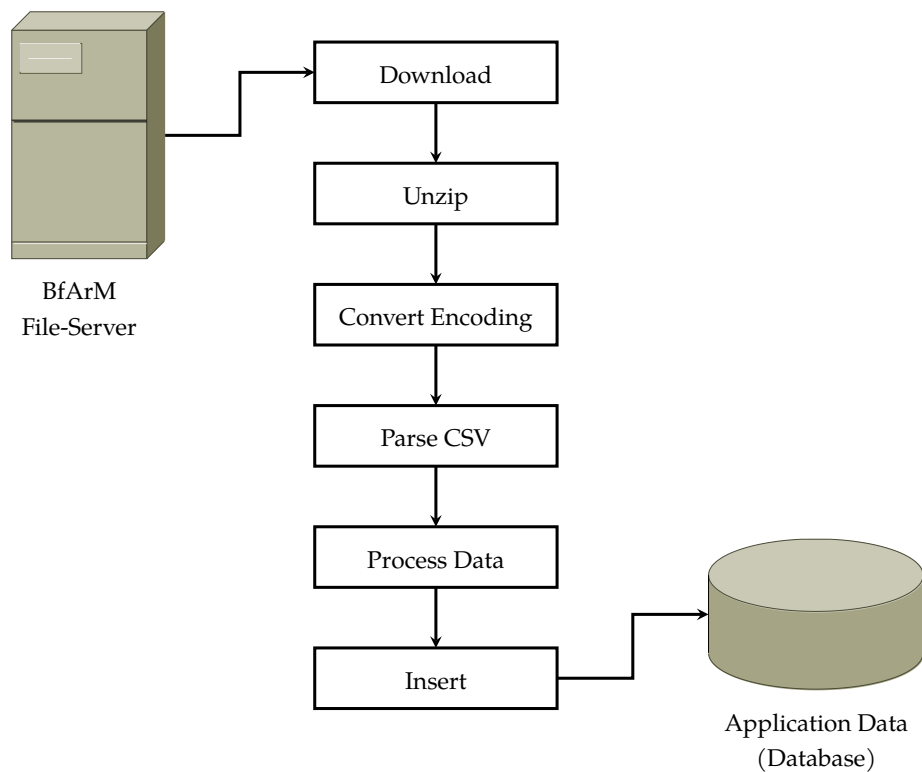


Abbildung 2.2: BfArM-Datenintegrationsprozess

2.3 Datenvorverarbeitung

„Data Preprocessing“ ist ein wichtiger Schritt in Feldern der Informatik wie *Machine Learning* und *Big Data*. In (García u. a., 2016) werden mehrere Methoden vorgestellt, wovon folgende in der Verarbeitung der BfArM-Daten zur Verwendung kommen:

1. *Data Cleaning*: Daten werden bereinigt, was sowohl das Korrigieren einzelner Werte, als auch das Entfernen überflüssiger Datensätze beinhaltet. Letzteres wird *Instance Reduction* genannt.
2. *Data Normalization*: Umwandlung der Datensätze auf ein bestimmtes Format.
3. *Data Integration*: Ein Datensatz wird durch zusätzliche Informationen bereichert, beziehungsweise mehrere Informationen werden zu einem Datensatz kombiniert.
4. *Missing values imputation*: Falls Informationen fehlen, müssen die betroffenen Datensätze mit einer bestimmten Logik behandelt werden oder alternativ können Daten durch eine Zufallsfunktion simuliert werden.

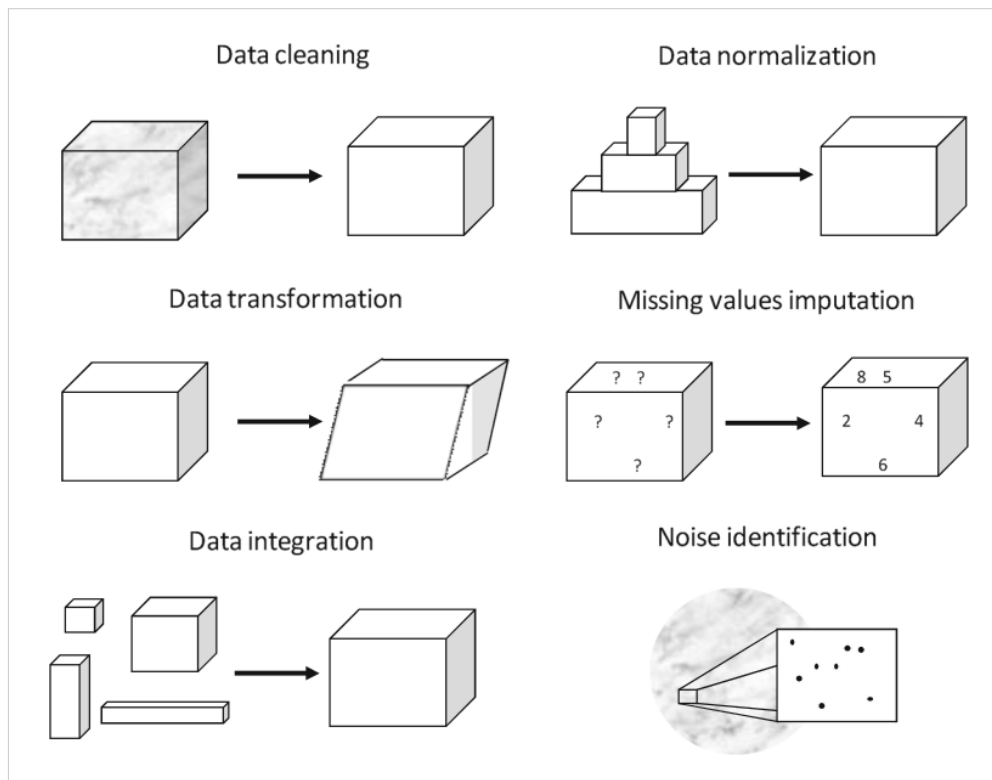


Abbildung 2.3: „Preprocessing Tasks“ aus (García u. a., 2016, Seite 4)

Die folgenden Unterabschnitte erklären die Vorverarbeitungsschritte für die BfArM-Daten und beziehen sich damit auf die in 2.1 genannten Abweichungen. Die Schritte erfolgen in der gelisteten Reihenfolge. Obwohl die Daten nach dem CSV-Parsing schon in einer von der Programmiersprache abhängigen Struktur vorliegen, wird zur Erklärung trotzdem noch die Datei-Struktur verwendet.

Datennormalisierung

6-Spalten-Umsteiger

Sowohl die ICD-10-GM Versionen 2004 und 2.0, als auch die OPS Version 2.1 beinhalteten Umsteiger in folgendem Format:

A00.0;A00.0;A;A;O;UNDEF
1-202;1-202;A;A;;

Um diese an das ICD-10-GM Format von 2024 anzupassen, werden die letzten beiden Spalten entfernt.

OPS Umsteiger

Für OPS Versionen ab 2010 sehen die Umsteiger-Einträge so aus:

1-100;N;1-100;N;A;A

Durch Entfernung der zweiten und vierten Spalte stimmen diese mit den ICD-10-GM Umsteiger Format von 2024 überein.

OPS 6-Spalten-Umsteiger, altes Format

Die OPS Versionen 2009 bis 2006 hatten ebenfalls sechs Spalten für die Umsteiger, aber in einer anderen Reihenfolge:

1-100;1-100;N;N;A;A

Hier müssen also die dritte und vierte Spalte entfernt werden.

OPS 5-Spalten-Umsteiger

Die Umsteiger von OPS Version 2005 waren in einem ganz eigenen Format geschrieben:

5-062.0;5-062.0;N;A;A
5-062.1;5-062.1;N;A;A
5-062.2;5-062.8;J;E;E
5-062.3;5-062.8;J;B;B

Hier wird dritte Spalte entfernt und außerdem werden die Sonderformen für automatische Überleitbarkeit von *B* und *E* nach *A* umbenannt.

Imputation

Für Umsteiger der OPS Version 2.0 gibt es nur drei Spalten:

1-208.0;A;1-209.0
1-208.x;;1-209.4

Die zweite Spalte zeigt allein die Überleitbarkeit an. Für die Angleichung an das ICD-10-GM Format von 2024 wird also die zweite Spalte entfernt und gedoppelt angehängt. Aus den beiden Beispielzeilen wird damit:

1-208.0;1-209.0;A;A
1-208.x;1-209.4;;

Datenbereinigung

KOMBI-Kode Aus OPS Versionen 2.1 und 2.0 wird die erste Zeile der Kode-Datei entfernt, welche den KOMBI-Eintrag enthält.

None statt UNDEF Für OPS Versionen 2009 bis 2004 wird der Kode-Wert None durch UNDEF ersetzt, sowohl in den Kodes-, als auch in den Umsteiger-Dateien.

Kreuz-Stern-System Für die ICD-10-GM Versionen 2.0 und 1.3 werden die Zeichen +, * und ! aus den Kode-Werten entfernt – sowohl in der Kodes-, als auch der Umsteiger-Datei.

Punkt-Strich-Notation Für die ICD-10-GM Versionen 2004, 2.0 und 1.3 wird zuerst die Zeichenfolge . – aus den Kode-Werten in beiden Dateitypen entfernt. Danach wird nochmals – entfernt. Die Reihenfolge ist wichtig, weil in Version 2004 zum Beispiel Kodes A00. – und G82.1 – vorkommen.

Instanzreduktion

Umsteiger

Für viele Versionen sind über 90% der Umsteiger-Einträge beidseitig automatische Überleitungen in den gleichen Kode. Diese müssen also gar nicht in eine Applikation aufgenommen werden, unter der Annahme, dass nicht vorhandene Umsteiger damit automatischen Überleitungen entsprechen. Dadurch können alle Umsteiger ausgeschlossen werden, bei denen der neue Kode gleich dem alten Kode ist und die automatische Überleitbarkeit in beide Richtungen gegeben ist.

Nicht-endständige Umsteiger

Die Überleitung von ICD-10-GM Version 2.0 auf 1.3 ist der einzige Fall, in dem die Umsteiger-Datei nicht-endständige Codes enthält. Durch das Entfernen der Sonderzeichen von Punkt-Strich-Notation und Kreuz-Stern-System gibt es außerdem doppelte Einträge in der ersten Spalte, also bei den Codes, die sich auf die Vorgänger-Version beziehen.

Folgende Funktion¹ entfernt die überflüssigen Einträge.

Angenommen die Codes befinden sich in einem Array mit Index von 0 bis (n-1), zum Beispiel für die ersten sechs Zeilen der Umsteiger-Datei aus ICD-10-GM, Überleitung 2.0 nach 1.3:

Index	old (Alter Kode)	new (Neuer Kode)
0	A00	A00.0
1	A00	A00.1
2	A00	A00.9
3	A00.0	A00.0
4	A00.1	A00.1
5	A00.9	A00.9

removeNonTerminal

Funktionsparameter:

- \$data
Umsteiger-Einträge

Lokale Variablen:

- \$index
Der Umsteiger-Eintrag, der aktuell verarbeitet wird.
Die Funktion läuft vom letztem Eintrag zum ersten.
- \$current
Der alte Kode des aktuellen Umsteiger-Eintrags.
Hierbei bedeutet \$data[\$index]['old'] Zugriff auf Zeile mit Index = Wert von \$index und Spalte 'Alter Kode'. Zum Beispiel: data[4]['old'] = A00.1.
- \$prev
Der alte Kode des zuletzt verarbeiteten Umsteiger-Eintrags.
Die Variable wird auf \$current gesetzt, wenn der Eintrag *nicht* entfernt wird.

¹Der Pseudocode ist beschrieben in Struktogrammen nach (Nassi und Shneiderman, 1973). Die später vorkommenden Algorithmen zur Umsteiger-Suche enthalten viele Variablenzuweisungen, die abhängig von der Suchrichtung sind und deren parallele Darstellung in Nassi-Shneidermann-Diagrammen übersichtlicher ist. Sie werden also im Sinne der Einheitlichkeit ebenfalls für den Algorithmus in diesem Kapitel verwendet.

2 BfArM-Daten

\$index = count(\$data) - 1	
\$current = \$data[\$index]['old']	
WHILE \$index > 0	
\$prev = \$data[\$index-1]['old']	
IF \$current contains \$prev AND length(\$current) > length(\$prev)	
Y	N
remove(\$data[\$index-1])	\$current = \$prev
\$index = \$index - 1	
reindex(\$data)	

Die IF-Bedingung bezieht sich auf String-basierte Operationen; also contains: „enthält“ als Sub-String und length: Anzahl der Zeichen im String. remove entfernt eine ganze Zeile, beziehungsweise einen ganzen Umsteiger-Eintrag.

Beispiel-Array nach Durchlaufen des Algorithmus:

Index	old (Alter Kode)	new (Neuer Kode)
0	A00.0	A00.0
1	A00.1	A00.1
2	A00.9	A00.9

Nicht-endständige Kodes

Für die meisten Anwendungen sind eigentlich nur die endständigen Kodes relevant. Statt diese bei jeder Operation herauszufiltern, können beim einmaligen Einlesen der Daten auch einfach die nicht-endständigen Kodes ausgeschlossen werden. Zu diesem Zweck kann der oben erklärte Algorithmus ebenfalls eingesetzt werden.

Integration zusätzlicher Informationen

Das nächst Kapitel erklärt, wie ermittelt wird, ob es zu einem Kode einer bestimmten Version Umsteiger in einer älteren oder neueren Version gibt. Um diese zusätzliche Information speichern zu können, werden die Kodes um eine Spalte erweitert.

3

Umsteiger-Suche

Dieses Kapitel behandelt die versionsübergreifende Suche nach Umsteigern. Es geht darum zu bestimmen, ob und wie ein bestimmter Kode in Betrachtung auf alle Versionen des Kodiersystems übergeleitet wurde. Die hier aufgeführten Algorithmen sind unabhängig von einer konkreten Implementation verfasst – als Datenstrukturen werden nur assoziative Arrays, also Maps mit Key⇒Value Paaren, sowie gewöhnliche Arrays mit Index verwendet.

3.1 Allgemeine Funktionen

Grundlegend ist davon auszugehen, dass Funktionen zum Lesen der Daten vorliegen – nach erfolgreicher Integration aus dem vorherigen Kapitel.

Daten Lesen

Konkret könnten zum Beispiel die Inhalte der Kodes- und Umsteiger-Dateien mit folgenden Parametern ausgelesen werden.

readData

- `$system`
Das Kodiersystem, beispielsweise als Konstanten definiert: 'icd10gm' und 'ops'.
- `$version`
Die Version, beziehungsweise Jahreszahl.
- `$code`
Ein Kode, nach dem gesucht wird. In Bezug auf die Notation von ICD-10-GM und OPS ist es sinnvoll, automatisch an rechter Stelle einen Wildcard zu implizieren – das heißt einen Platzhalter für beliebige, weitere Zeichen. Dann werden mit einem leeren Wert alle Daten für die anderen Parameter gelesen ohne Einschränkung auf den Kode.

- \$type

Die Art der Daten, die gelesen werden sollen. Möglich ebenfalls als Konstanten:

- 'kodes'

Kodes-Daten: Kode und Titel.

- 'umsteiger'

Umsteiger-Daten: alter Kode und neuer Kode, sowie Information über die automatische Überleitbarkeit in beide Richtungen. Es werden die Umsteiger von der angegebenen auf die nächstältere Version abgefragt. Suche über \$code = neuer Kode.

- 'umsteiger_join'

Wie Umsteiger-Daten, aber zusätzlich mit Titel für jeweils den alten und neuen Kode. „Join“ bezieht sich auf die entsprechende Datenbankoperation.

- 'umsteiger_join_alt'

Wie oben, außer dass die Suche über \$code = alter Kode und in die andere Richtung erfolgt. Das heißt es wird die angegebene Version mit der nächstneueren Version verglichen. „Alt“ für „alternate direction“.

Nächste Version

Außerdem wird eine Funktion benötigt, die ausgehend von System und Version die jeweils nächstältere oder -neuere Version ermittelt. Hierfür kann die in Unterabschnitt 2.1 erwähnte strukturierte Datei dienen, welche alle Versionen und Abweichungen definiert.

Wenn keine neuere oder ältere Version existiert, bleibt der Rückgabewert leer.

nextNewerVersion / nextOlderVersion

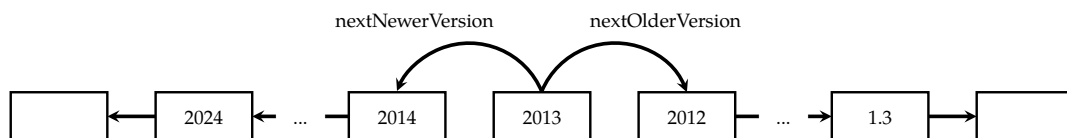


Abbildung 3.1: Ermitteln der nächstliegenden Versionen am Beispiel von ICD-10-GM.

Beispieldaten mit graphischer Darstellung

Um beispielsweise die Codes M21.4 und M21.6 aus der ICD-10-GM Version 2016 mit der Funktion readData abzufragen, sehen die Aufrufe so aus:

```
readData ('icd10gm', '2014', 'M21.4', 'kodes')
```

```
readData ('icd10gm', '2014', 'M21.6', 'kodes')
```

Die Ergebnisse in Tabellenform:

code (Kode)	name (Titel)
M21.4	Plattfuß [Pes planus] (erworben)
M21.6	Sonstige erworbene Deformitäten des Knöchels und des Fußes

Für das restliche Kapitel wird angenommen, dass die Ergebnisse der readData-Funktion in einer zweidimensionalen Datenstruktur gespeichert werden, die einer Tabelle ähnelt – das heißt eine Array-Liste mit einem Eintrag pro Zeile beginnend mit Index = 0, sowie einem assoziativen Array (Map oder alternativ eine Klassenstruktur) mit Key = Spaltenüberschrift. Der Zugriff auf den Titel „Plattfuß“ in deren oberen Tabelle erfolgt dann über `$data[0]['name']`.

(Mit dem \$-Zeichen werden Variablennamen hervorgehoben.)

Analog zu den Codes können die Umsteiger wie folgt abgefragt werden:

```
lies_daten ('icd10gm', '2014', 'M21.4', 'umsteiger')
```

```
lies_daten ('icd10gm', '2014', 'M21.6', 'umsteiger')
```

Resultierend in:

new (Kode Version 2014)	old (Kode Version 2013)	←Auto→	
M21.4	M21.4	A	A
M21.6	M21.6	A	A

“Auto” steht für die automatische Überleitbarkeit in die jeweilige Richtung, das heißt 2014←2013 und 2014→2013.

Von Version 2014 auf 2013 ändern sich die angegebenen ICD-10-GM Codes nicht. Sie sind auch in beide Richtungen automatisch überleitbar. Wie bereits in der Datenintegration erwähnt, müssen diese Einträge eigentlich gar nicht aufgenommen werden. Sie sind hier nur zur Vereinfachung aufgelistet, weil die Einträge auch so in den BfArM-Dateien enthalten sind. Bei der versionsübergreifenden Suche nach Umsteigern sind allerdings nur die Veränderungen relevant. Anstatt alle nicht relevanten Umsteiger-Einträge abzuspeichern und bei jeder Abfrage auszuschließen, kann alternativ auch angenommen werden, dass bei Nichtvorhandensein eines Umsteigers zwischen zwei Versionen dieser Code einfach gleich bleibt. Die Suche nach Umsteigern geht immer von einem vorhandenem Code aus.

Wichtig: Für den Rest des Kapitels bedeutet Umsteiger, dass bei der Überleitung eines Codes von der alten auf die neue Version wirklich eine Veränderung vorliegt.

3 Umsteiger-Suche

M21.4 hat also keine Umsteiger. M21.6 hat folgende:

new (Kode Version 2015)	old (Kode Version 2014)	← Auto →	
M21.60	M21.6	A	
M21.61	M21.6	A	
M21.62	M21.6	A	
M21.63	M21.6	A	
M21.68	M21.6	A	
new (Kode Version 2013)	old (Kode Version 2012)	← Auto →	
M21.6	M21.60	A	
M21.6	M21.67	A	
M21.6	M21.87	A	A

Die Suche wird ab Version 2012 mit M21.60, M21.67 und M21.87 fortgesetzt:

new (Kode Version 2.0)	old (Kode Version 1.3)	← Auto →	
M21.60	M21.6	A	
M21.67	M21.6	A	
M21.80	M21.8	A	
M21.81	M21.8	A	
M21.82	M21.8	A	
M21.83	M21.8	A	
M21.84	M21.8	A	
M21.85	M21.8	A	
M21.86	M21.8	A	
M21.87	M21.8	A	
M21.88	M21.8	A	
M21.89	M21.8	A	A

Anmerkung: Die Umsteiger für Kode M21.8, Version 1.3. sind nur zur Vollständigkeit gelistet. Von M21.6, Version 2014 ausgehend ist nur M21.87, Version 2.0 erreichbar.

Die Ergebnisse können als Graph dargestellt werden; siehe nächste Seite. Die Versionen sind horizontal angeordnet und die Codes pro Version vertikal darunter als Knoten. Ausgangspunkt sind wie erwähnt die Codes M21.4 und M21.6 der Version 2014. Die Knoten der anderen Versionen entsprechen den Codes, die über Umsteiger erreichbar sind. Die Pfeilspitzen geben die Richtung der automatischen Überleitbarkeit an.

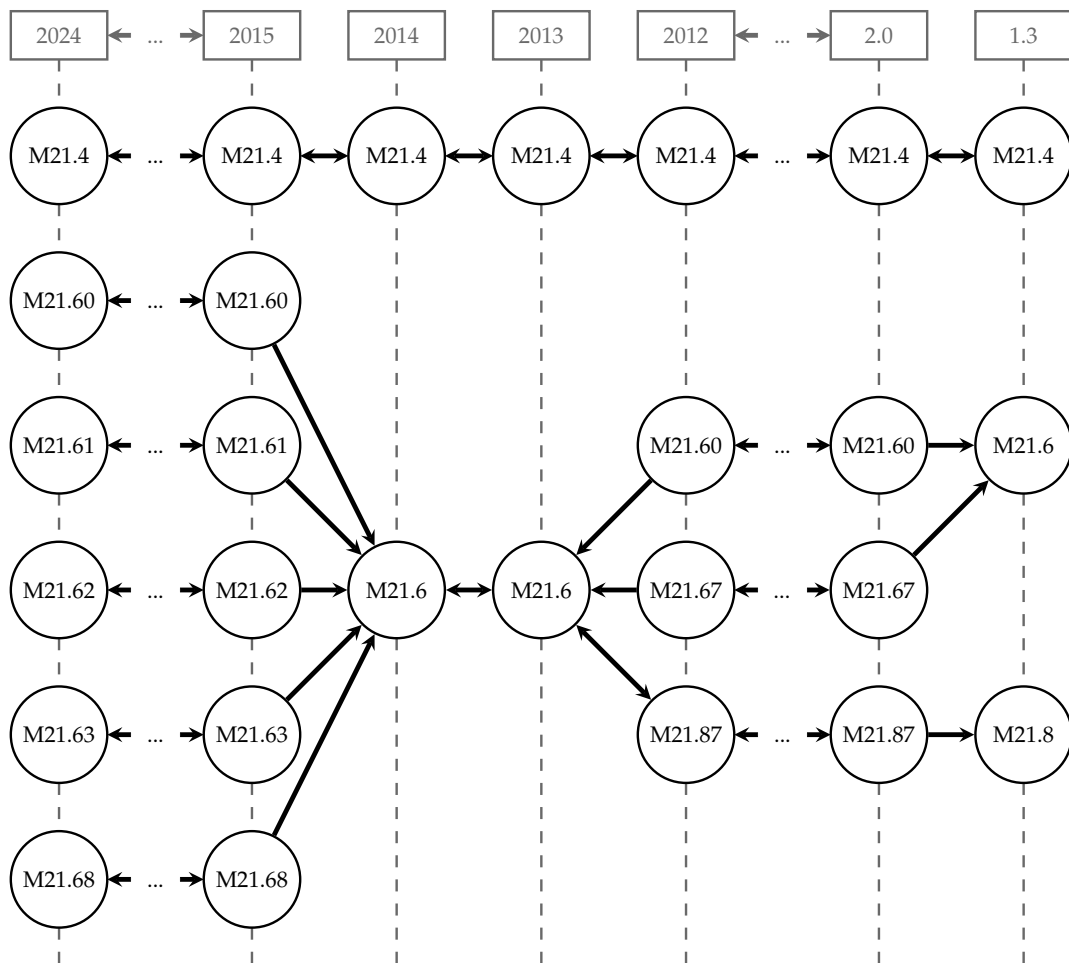


Abbildung 3.2: Graphische Repräsentation der Überleitungen ausgehend von den Codes M21.4 und M21.6 der ICD-10-GM Version 2014.

3.2 Transitive Hülle

Wenn statt der graphischen Repräsentation oben:

1. Die nicht relevanten Überleitungen ausgeschlossen werden; also wie besprochen nur Umsteiger-Einträge mit Veränderung zählen.
2. Die Kanten im Graph die Suchrichtung anzeigen statt der automatischen Überleitbarkeit.

Dann ergibt sich ein gerichteter Graph – das heißt ein Graph, der nur einfach gerichtete Kanten enthält und in dem jeder Knoten mit mindestens einem anderen Knoten über eine gerichtete Kante –auch Bogen genannt– verbunden ist. Unter diesen Voraussetzungen entspricht die versionsübergreifende Suche nach Umsteigern der Bestimmung der transitiven Hülle in der Graphentheorie.

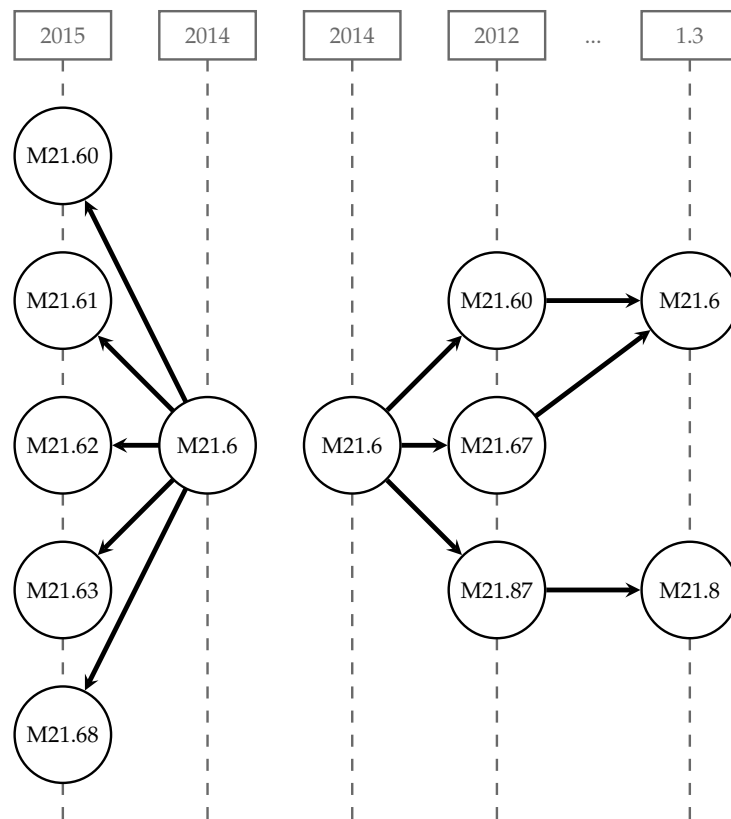


Abbildung 3.3: Von M21.6, Version 2014 über Umsteiger in chronologische und rückwärts chronologische Richtung erreichbare Codes als zwei gerichtete Graphen.

Die transitive Hülle wird in (Gross, Yellen und Zhang, 2013, Seite 172) wie folgt definiert:

D26: The *transitive closure* R^* of a binary relation R is the relation R^* defined by $(x, y) \in R^*$ if and only if there exists a sequence $x = v_0, v_1, v_2, \dots, v_k = y$ such that $k \geq 1$ and $(v_i, v_{i+1}) \in R$, for $i = 0, 1, \dots, k - 1$. Equivalently, the transitive closure R^* of the relation R is the smallest transitive relation that contains R .

D27: Let G be the digraph representing a relation R . Then the digraph G^* representing the transitive closure R^* of R is called the *transitive closure of the digraph* G . Thus, an arc (x, y) , $x \neq y$, is in the transitive closure G^* if and only if there is a directed x - y path in G . Similarly, there is a self-loop in digraph D^* at vertex x if and only if there is a directed cycle in digraph G that contains x .

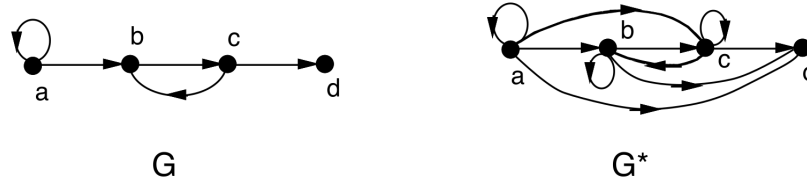
Oder kurz zusammengefasst: die transitive Hülle eines gerichteten Graphens G ist wiederum ein gerichteter Graph G^* , der von jedem Knoten einen Bogen zu allen von diesem Knoten aus in G erreichbaren Nachbarn besitzt.

In (Gross, Yellen und Zhang, 2013, Seite 172) ist hierzu folgendes Beispiel dargestellt:

E7: Suppose a relation R on the set $S = \{a, b, c, d\}$ is given by

$$\{(a, a), (a, b), (b, c), (c, b), (c, d)\}$$

Then the digraph G representing the relation R and the transitive closure G^* are as shown in Figure 3.1.7.



Es gibt mehrere Verfahren, um die transitive Hülle zu bestimmen. Es werden nun zwei Algorithmen vorgestellt, die anhand ihrer primären Suchrichtung benannt sind.

Wie in den Graphen dargestellt, läuft die *horizontale* Suche primär über die Versionen – mit jeweils einzelnen Codes. Die *vertikale* Suche läuft primär über die Codes – das heißt es werden zuerst alle Codes einer Version gelesen.

Horizontale Suche

Ein intuitives Verfahren die transitive Hülle zu bestimmen wird in (Jakobsson, 1991, Seite 200) so beschrieben:

2.2 Search

A second approach for computing the transitive closure is to search the graph n times, each time starting from a different node, thereby determining what can be reached from that node. In this approach, we completely disregard any parts of the TC -relation that have already been computed for other starting nodes. Instead, we search the entire part of the graph that is reachable from the starting node by following every edge we can get to.

Ähnlich wie im vorherigen Abschnitt für den Code M21.6 dargestellt, werden also einfach ausgehend von einem Knoten so viele Suchen gestartet bis keine benachbarten Knoten mehr gefunden werden – oder im Anwendungsfall der Umsteiger bis alle Versionen verarbeitet wurden.

Vorher erzielte Ergebnisse für andere Knoten, beziehungsweise Codes, werden ignoriert. Das hieße im Beispiel oben, dass eine rückwärts chronologische Suche nach Umsteigern von M21.60 und M21.61 der ICD-10-GM Version ab dem Vorgänger M21.6 der Version 2014 zweimal exakt gleich ablaufen würde.

Der Pseudocode für diese Vorgehensweise:

searchHorizontal

Funktionsparameter:

- \$system, \$version, \$code
Wie Funktion readData, siehe Unterabschnitt 3.1.

Lokale Variable:

- \$data Rückgabewert, initial: leer.
Eine assoziative Datenstruktur mit Key fwd \Rightarrow Umsteiger, die chronologisch vorwärts von dem Suchkode erreichbar sind und rev \Rightarrow chronologisch rückwärts.

$\$data[\text{'fwd'}] = \text{searchHorizontalRecursion} (\$system, \$version, \$code, \text{true})$
$\$data[\text{'rev'}] = \text{searchHorizontalRecursion} (\$system, \$version, \$code, \text{false})$
RETURN \$data

Ausgehend von einem gegebenen Kodiersystem, Version, Kode wird chronologisch vorwärts und rückwärts eine rekursive Suche über alle Versionen gestartet.

searchHorizontalRecursion

Funktionsparameter:

- \$system, \$version, \$code
Wie searchHorizontal.
- \$chronological
Bestimmt die Suchrichtung. TRUE: chronologisch vorwärts und FALSE: rückwärts.

Lokale Variablen:

- \$data Rückgabewert, initial: leer.
Eine mehrdimensionale, assoziative Datenstruktur, die die Ergebnisse der rekursiven Suche enthält. Gefundene Umsteiger werden als Liste mit dem Key umsteiger gespeichert; zusätzlich zu den Listen wird auch die Versionen vorher und nachher notiert. Jeder einzelne Umsteiger-Eintrag enthält wiederum die Informationen zur automatischen Überleitbarkeit, sowie die Titel der Kodes vorher und nachher. Falls ein Umsteiger weitere Umsteiger hat, werden diese unter einem Key recursion abgelegt und die Suche fortgesetzt. Das heißt unter recursion ist das gleiche Datenkonstrukt nochmal enthalten. Ein Beispielergebnis für M21.6 befindet sich im Anhang A.2.
- \$otherVersion
Die Version, zu der die Umsteiger ermittelt werden; \$version \rightarrow \$otherVersion.
- \$otherCode
Die Spalte im Umsteiger-Eintrag, zu der der aktuelle Kode übergeleitet wird.

3 Umsteiger-Suche

- \$readType
Auf welche Art die Daten gelesen werden; siehe readData.
- \$umsteiger, \$entry
Liste der Umsteiger, Schleifenvariable: Umsteiger-Eintrag.
- \$recursion
Ergebnis des rekursiven Funktionsaufrufs bei gefundenen Umsteigern.

IF \$chronological	
Y	N
\$otherVersion = nextNewerVersion(\$system, \$version)	\$otherVersion = nextOlderVersion(\$system, \$version)
\$readType = 'umsteiger_join_alt'	\$readType = 'umsteiger_join'
\$otherCode = 'new'	\$otherCode = 'old'

Lokale Variablen werden anhand der Suchrichtung unterschiedlich belegt.

IF empty(\$version) OR empty(\$otherVersion)	
Y	
RETURN \$data	∅

Falls es in der Suchrichtung keine weitere Version mehr gibt, endet die Rekursion.

\$umsteiger = readData(\$system, \$version, \$code, \$readType)	
IF empty(\$umsteiger)	
Y	
RETURN searchHorizontalRecursion(\$type, \$otherVersion, \$code, \$chronological)	∅

Die Umsteiger zwischen den Versionen im aktuellen Rekursionsschritt werden ermittelt. Falls es keine gibt – also keine, die eine Veränderung ausdrücken – dann wird die Rekursion mit der nächsten Version fortgesetzt.

FOREACH \$entry IN \$umsteiger		
IF \$entry[\$otherCode] IS NOT 'UNDEF'		
Y		
\$recursion = searchHorizontalRecursion (\$system, \$version, \$entry[\$otherCode], \$chronological)		∅
IF NOT empty(\$recursion)		
Y		
\$entry['recursion'] = \$recursion	∅	
\$data['umsteiger'][] = \$entry		

Für jeden Umsteiger wird die Rekursion mit dem veränderten Code fortgesetzt. Allerdings nur falls dieser nicht UNDEF ist, weil es sich dann um einen entfer-

ten oder neu hinzugefügten Kode handelt. Also wenn wie im Beispiel der M21.6 der ICD-10-GM Version 2013 die Umsteiger [M21.60, M21.67, M21.87] in der Version 2012 hat, dann wird die Suche in die rückwärts chronologische Richtung mit diesen drei Kodes fortgesetzt statt mit M21.6. Die Ergebnisse dieser Verzweigung, sofern vorhanden, werden abgespeichert mit dem Key `recursion` zusätzlich zu jedem Umsteiger-Eintrag. Die Liste aller Umsteiger-Einträge, inklusive der UNDEF-Umsteiger, wird in das Ergebnis mit dem Key `umsteiger` aufgenommen.

<code>\$data['version'] = \$version</code>
<code>\$data['other'] = \$otherVersion</code>
<code>RETURN \$data</code>

Falls es Umsteiger für einen Kode gibt, werden zusätzlich die Versionen vorher und nachher gespeichert. Dann endet auch hier die Rekursion.

Anhang A.2 enthält das Ergebnis im JSON-Format für den Beispielaufwurf der horizontale Suche für M21.6, Version 2014 entsprechend der Abbildung 3.3.

Vertikale Suche

Wie bereits erwähnt beginnt der Algorithmus der horizontalen Suche für jeden Kode ohne Vorkenntnisse über schon getätigte Aufrufe. Wenn die Umsteiger für alle Kodes einer Version –beziehungsweise eines ganzen Kodiersystems– gefunden werden sollen, ist diese Vorgehensweise nicht effizient.

Besonders für gerichtete Graphen ist allerdings Purdoms Algorithmus gut geeignet, um die transitive Hülle zu bestimmen, wie in (Dar, 1993, Seite 77) beschrieben. Folgende kurze Zusammenfassung des Algorithmus basiert auf dieser Beschreibung, sowie auf (Purdom, 1970) selbst:

1. Ausgehend von einem Graphen G : Bestimme die stark zusammenhängenden Komponenten (SCC) in G und vereinige diese in jeweils einen einzelnen Knoten. Das Ergebnis ist ein zyklenerfreier, verdichteter Graph G_c .
2. Sortiere G_c topologisch.
3. Ermittle die transitive Hülle von G_c über dessen Adjazenzlisten in rückwärts topologischer Reihenfolge.
4. Bestimme die transitive Hülle des ursprünglichen Graphens G über die Nachbarschaftslisten der stark zusammenhängenden Komponenten in G_c : ein Knoten y ist ein erreichbarer Nachbar von Knoten x , falls $SCC(y) = SCC(x)$ oder falls $SCC(y)$ ein erreichbarer Nachbar von $SCC(x)$ ist.

Die Vereinigung der stark zusammenhängenden Komponenten aus Schritt eins wird in (Purdom, 1970) wie folgt dargestellt:

3 Umsteiger-Suche

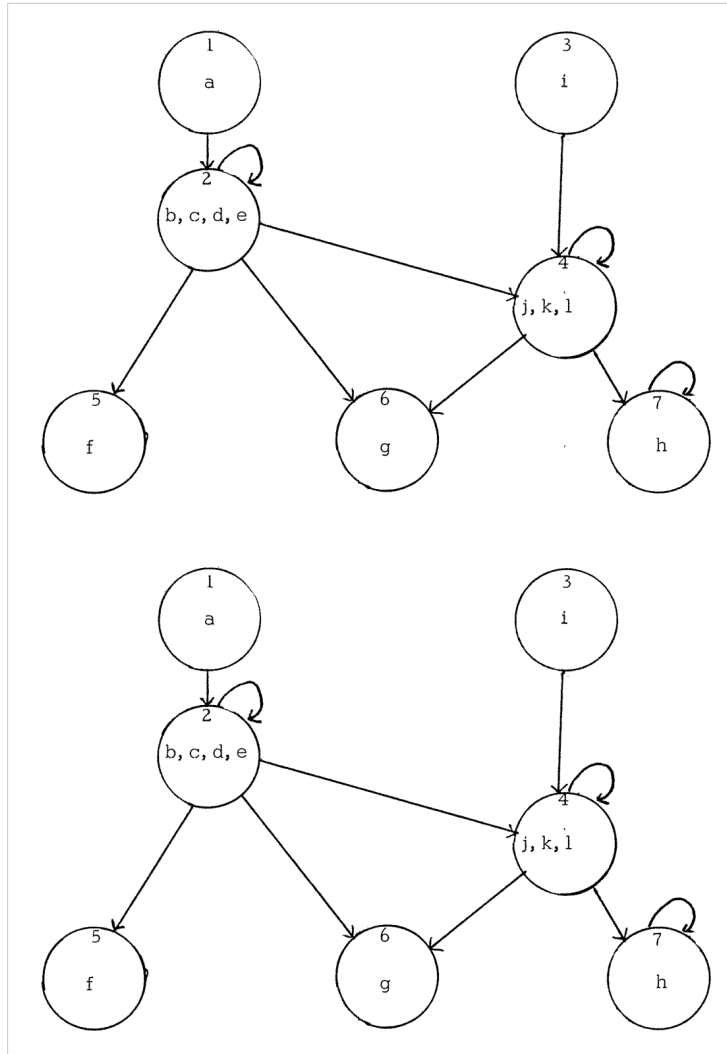


Abbildung 3.4: Beispielgraph G und Vereinigung der stark zusammenhängenden Komponenten aus (Purdom, 1970, Seite 78).

Die hier vorgestellte vertikale Suche nach Umsteigern entspricht nicht ganz Purdoms Algorithmus. Sie weicht in mehreren Punkten ab:

- Die topologische Sortierung aus Schritt zwei in der Anwendung der Suche nach Umsteigern ist nicht notwendig, weil die Versionen immer in einer bestimmte Richtung verglichen werden und die Codes sich nur zwischen zwei Versionen ändern können. Die Daten sind also schon implizit topologisch sortiert.
- Ebenfalls ist nicht notwendig, die komplette transitiven Hülle zu ermitteln, sondern nur wie Codes der Startversion in die Zielversion übergeleitet werden. Die Zwischenergebnisse werden trotzdem gespeichert.
- Auch bei der Suche nach Umsteigern nur mit Veränderungen ist es möglich, anders als bei den Knoten in einem Graphen, dass Codes mehrmals vorkommen. Zum Beispiel gibt es G83.3 in ICD-10-GM nicht zwischen Version 2017 und 2005, aber schon davor und danach.

Trotzdem basiert die vertikale Suche wesentlich auf zwei Besonderheiten von Purdoms Algorithmus, die in (Dar, 1993, Seite 76f) hervorgehoben werden:

5.3.1 The Purdom Algorithm

Purdom, in [Pur70], made two key observations:

1. During the computation of transitive closure of a directed acyclic graph, if node $s \prec_t t$, then additions to the successor list of node s cannot affect the successor list of node t . One should therefore compute the successor list of t first and then that of s . By processing nodes in reverse topological order, one need add to a node only the successor lists of its immediate successors since the latter would already have been fully expanded. We call this idea the *immediate successor optimization* [AgJ90].
2. All nodes within a strongly connected component (SCC) in a graph have identical reachability properties, and the condensation graph obtained by collapsing all the nodes in each strongly connected component into a single node is acyclic.

Konkret bedeuten diese Besonderheiten für die Umsteiger-Suche:

1. Topologisch rückwärts gerichtete Vorgehensweise:
So ist zum Beispiel bei der Suche nach den Umsteigern eines ICD-GM-10 Kodes der Version 1.3 die Reihenfolge, in der die anderen Versionen abgearbeitet werden: 2024, 2023, 2022 und so weiter. Oder anders ausgedrückt: Wenn die Überleitungen in chronologischer Reihenfolge bestimmt werden sollen, dann läuft die vertikale Suche umgekehrt, also chronologisch rückwärts.
2. Vereinigung der stark zusammenhängenden Komponenten:
Nach jedem Schritt über eine Version, wird geprüft, ob ein neu gefundener Umsteiger eine Überleitung in einen Code enthält, der bereits in den davor abgearbeiteten Versionen als Umsteiger vorkommt. Falls ja, dann werden diese Umsteiger vereinigt, das heißt die Zwischenschritte werden zu einem Schritt zusammengefügt.

Abbildung 3.5 auf der nächsten Seite zeigt ein einfaches Beispiel für die Vereinigung:

Angenommen ein abstraktes System hat Umsteiger von der Version 1.0 auf 2.0: $A \rightarrow B$ und $M \rightarrow N$, sowie von Version 2.0 auf 3.0: $B \rightarrow C$. Wenn nun in chronologischer Reihenfolge $1.0 \rightarrow 2.0 \rightarrow 3.0$ Umsteiger ermittelt werden sollen, erfolgt die Suche sowie Vereinigung gefundener Umsteiger in umgekehrter Reihenfolge. Das heißt wenn im Schritt $3.0 \leftarrow 2.0$ der Umsteiger $C \leftarrow B$ gefunden wird, dann wird danach bei der Vereinigung mit Version 1.0 der Umsteiger $B \leftarrow A$ in $C \leftarrow A$ umgewandelt. A in Version 1.0 entspricht C in Version 3.0.

Falls es Verzweigungen gibt, also es mehrere Umsteiger für einen Code gibt, dann werden entsprechend Listen vereinigt. Ein reales Beispiel dazu folgt später.

3 Umsteiger-Suche

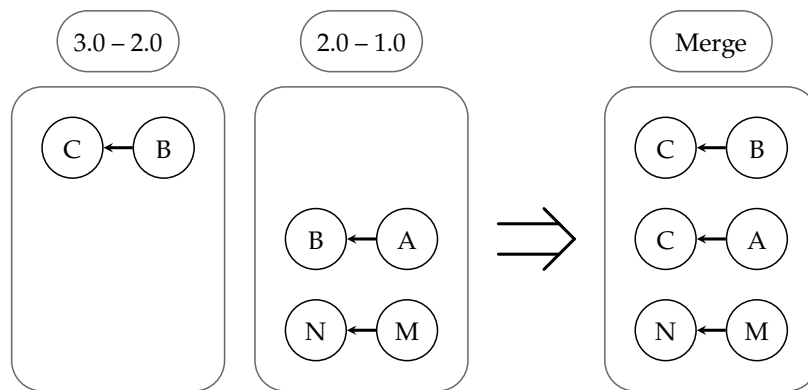


Abbildung 3.5: Graphische Veranschaulichung der zwei Besonderheiten von Purdoms Algorithmus, die in der vertikalen Suche zur Anwendung kommen.

Der gesamte Algorithmus für die vertikale Suche als Pseudocode:

searchVertical

Funktionsparameter:

- \$system
Das Kodiersystem.
- \$targetVersion
Die Zielversion, auf die von allen anderen Version die Umsteiger über alle Codes ermittelt werden sollen.
- \$function
Eine optionale, anonyme Funktion.

Lokale Variable:

- \$data Rückgabewert, initial: leer.
Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn \$function nicht gesetzt ist. Nach Durchlauf des Algorithmus enthält die erste Dimension als Keys alle Versionen außer der Zielversion. Die zweite Dimension hat als Key jeweils einen Code und als Value die Liste der zugehörigen Umsteiger nach erfolgter Vereinigung. Codes ohne Umsteiger werden nicht aufgenommen.

\$data += searchVerticalSubroutine(\$system, \$targetVersion, false, \$function)
\$data += searchVerticalSubroutine(\$system, \$targetVersion, true, \$function)
RETURN \$data

Basierend auf der Zielversionen wird in beide Richtungen die vertikale Suche als Unterfunktion gestartet.

\$data+= bedeutet, dass die assoziativen Datenstrukturen zusammengefasst werden. Wenn ein Key auf beiden Seiten der Addition existiert, wird der von der linken Seite für die Summe übernommen. Das kann allerdings hier bei den Versionen als Keys nicht passieren, weil jede Version nur einmal bearbeitet wird.

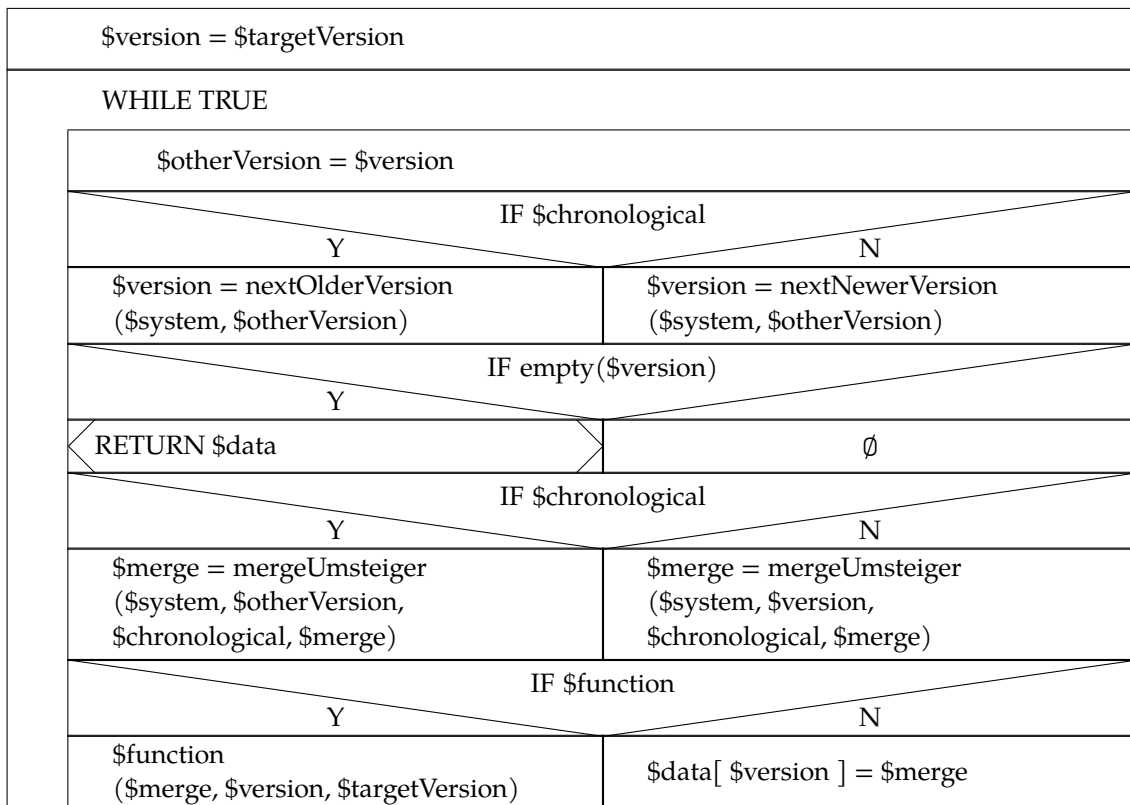
searchVerticalSubroutine

Funktionsparameter:

- \$system, \$targetVersion, \$function
Wie searchVertical.
- \$chronological
Bestimmt die Richtung, in der die Versionen abgearbeitet werden. TRUE: chronologisch vorwärts und FALSE: rückwärts. Die Reihenfolge ist wie erwähnt topologisch umgekehrt, das heißt ausgehend von der Zielversion.

Lokale Variablen:

- \$data Rückgabewert, initial: leer.
Wie searchVertical.
- \$merge
Key-Value-Paare mit Kode \Rightarrow [übergeleitete Kode(s)] nach Vereinigung.
- \$version, \$otherVersion
Die aktuell zu verarbeitende Version und die nächste Version.



3 Umsteiger-Suche

Die Schleife wird solange durchlaufen bis es keine weitere Version in der vorgegebenen Suchrichtung gibt. Pro Durchlauf wird die Variable `$merge` erweitert wie in Abbildung 3.5 dargestellt; sie enthält die vereinigten Umsteiger aller bisher verarbeiteten Versionen. Abhängig davon, ob die anonyme Funktion `$function` definiert ist, wird diese entweder ausgeführt oder die Variable `$data` für die Rückgabe befüllt.

mergeUmsteiger

Funktionsparameter:

- `$system, $version, $chronological, $merge`
Wie `searchVerticalSubroutine`.

Lokale Variablen:

- `$umsteiger`
Liste aller Umsteiger der Version; initial leer.
- `$item`
Schleifenvariable der `readData`-Funktion.
- `$current, $other, $currentCode, $otherCode`
Abhängig von der Richtung werden pro Umsteiger-Eintrag entweder der alte oder neue Code genommen, um die Umsteiger auf die nächste Version zu ermitteln. Im Gegensatz zur horizontalen Suche werden nur die Codes abgefragt – also ohne zusätzliche Informationen. Deswegen ist der `$readType` der `readData`-Funktion hier einfach `'umsteiger'`. Da alle Umsteiger aller Versionen gesammelt werden, könnte die `$merge`-Variable sonst relativ groß werden.

IF \$chronological	
Y	N
<code>\$current = 'old'</code>	<code>\$current = 'new'</code>
<code>\$other = 'new'</code>	<code>\$other = 'old'</code>
FOREACH \$item IN readData(\$system, \$version, "", 'umsteiger')	
<code>\$currentCode = \$item[\$current]</code>	
<code>\$otherCode = \$item[\$other]</code>	
IF exists(\$merge[\$otherCode]) AND NOT (\$currentCode IS 'UNDEF' OR \$otherCode IS 'UNDEF')	
Y	N
<code>\$umsteiger[\$currentCode] = concatenate(\$merge[\$otherCode], \$umsteiger[\$currentCode])</code>	<code>\$umsteiger[\$currentCode][] = \$otherCode</code>
RETURN \$umsteiger + \$merge	

Es werden Umsteiger der aktuellen Versionen gelesen; wie erwähnt nur Überleitungen mit Veränderungen. Wenn der Zielcode einer Änderung bereits Umsteiger in \$merge enthält, der vereinigten Liste vorher verarbeiteter Versionen, dann werden diese genommen und mit den Umsteigern der aktuellen Version vereinigt. concatenate heißt, dass zwei Listen einfach zusammengefügt werden, ohne dass Indizes überschrieben werden. Ansonsten –oder falls Umsteiger UNDEF enthalten, also entfernt oder neu hinzugefügt werden– werden die Umsteiger der aktuellen Version übernommen wie sie sind.

Rückgabewert ist ein assoziatives Array der aktuellen Umsteiger als Key-Value-Paare mit $\text{Code} \Rightarrow [\text{übergeleitete Code(s)}] + \$merge$, wobei + hier bedeutet, dass die Einträge von der linken Seite der Summe genommen werden, falls dieselben Keys auf beiden Seiten vorhanden sind.

Konkretes Beispiel

Angenommen es soll ermittelt werden, wie der ICD-10-GM Kode G83.8 der Version 1.3. in die neueren Versionen übergeleitet wurde.

Der folgende Graph zeigt nur die hierfür relevanten Codes der anderen Versionen. Bei der vertikalen Suche werden immer alle Codes pro Version abgefragt.

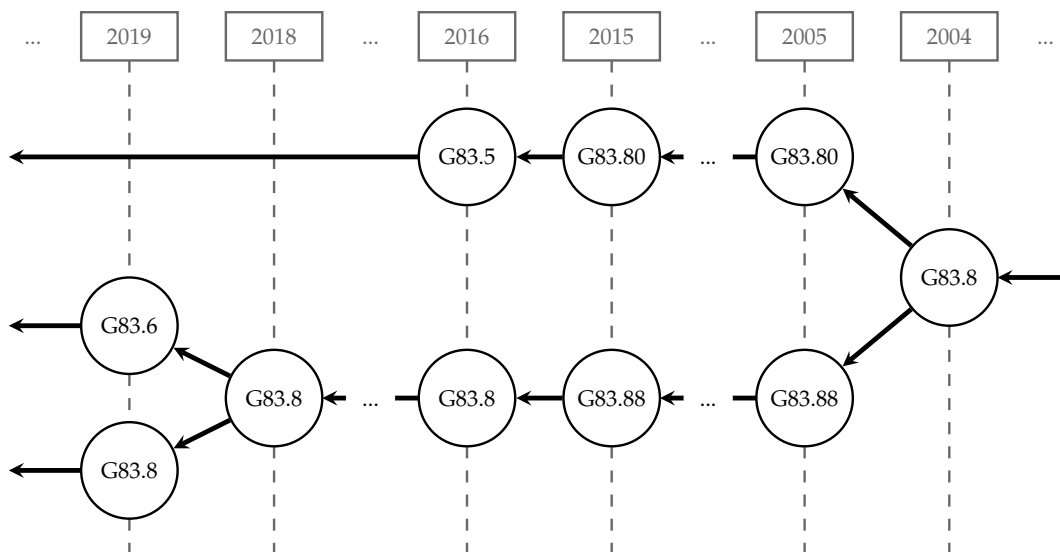


Abbildung 3.6: Graphisches Beispiel für die vertikale Suche nach Umsteigern ausgehend von ICD-10-GM Kode G83.8, Version 1.3 in chronologischer Richtung.

Die Suche erfolgt topologisch rückwärts, das heißt wenn die Suchrichtung chronologisch ist wie im aktuellen Beispiel, dann ist die Bearbeitungsreihenfolge chronologisch rückwärts.

Die Datenstruktur, welche die vereinigten Umsteiger enthält, wird also schrittweise aufgebaut von der neuesten Version ausgehend:

'2018' \Rightarrow ['G83.8' \Rightarrow ['G83.6', 'G83.8']]

'2015' \Rightarrow ['G83.8' \Rightarrow ['G83.6', 'G83.8']],
 'G83.80' \Rightarrow ['G83.5'],
 'G83.88' \Rightarrow ['G83.6', 'G83.8']]

Der erste Eintrag wird übernommen, der zweite kommt neu hinzu und für den dritten Eintrag wird G83.88 \Rightarrow G83.8 entsprechend umgewandelt, weil es Umsteiger für G83.8 schon gibt.

'2004' \Rightarrow ['G83.8' \Rightarrow ['G83.5', 'G83.6', 'G83.8']],
 'G83.80' \Rightarrow ['G83.5'],
 'G83.88' \Rightarrow ['G83.6', 'G83.8']]

Ähnlich wie oben wird hier G83.8 \Rightarrow [G83.80, G83.88] umgewandelt und dann aber der vorhandene Eintrag für G83.8 ersetzt. Das Übernehmen der unveränderten Einträge wird noch bis zur Version 1.3 fortgesetzt, so dass auf der fertigen Datenstruktur gilt:

`$data['1.3']['G83.8'] = ['G83.5', 'G83.6', 'G83.8']`

3.3 Vergleich der Suchalgorithmen

Wie bereits erwähnt erfolgt die horizontale Suche pro Kode über alle Versionen ohne Vorkenntnisse vorheriger Suchergebnisse. Im Gegensatz dazu verarbeitet die vertikale Suche pro Version alle Codes nur einmal, ist also allein deshalb schon effizienter, wenn Umsteiger für mehr als einen Kode ermittelt werden sollen.

Wenn die Daten zusätzlich aus einer Datenbank gelesen werden, ist die Anzahl an Aufrufen der readData-Funktion wesentlich entscheidender als die Menge an Ergebnissen pro Aufruf. Denn das Verbinden mit der Datenbank und Auslösen eines Suchbefehls dauert eine gewisse Zeit, was sich bei sehr vielen Aufrufen schnell aufaddiert. Dementsprechend ist es wesentlich schneller, alle Umsteiger einer Version auf einmal zu lesen, statt einzeln.

Andererseits würde wiederum das Abfragen zusätzlicher Informationen, wie beispielsweise der Titel der Codes eines Umsteigers vorher und nachher, bei der vertikalen Suche schnell die Größe der Daten im Arbeitsspeicher ansteigen lassen, weil pro Aufruf immer alle Umsteiger pro Version zwischengespeichert werden.

Daher werden die beiden Algorithmen für unterschiedliche Zwecke eingesetzt.

Horizontale Suche:

- Darstellung der Umsteiger eines einzelnen Kodes mit Zusatzinformationen

Vertikale Suche:

- Schreiben einer ConceptMap
- Für alle Codes einer Version bestimmen, ob diese Umsteiger haben (ja/nein)

3.4 Schreiben der FHIR ConceptMap

Dieser Abschnitt bezieht sich auf die in Kapitel 1.1 beschriebene ConceptMap-Struktur.

Die anonyme Funktion, die der vertikalen Suche übergeben werden kann, dient zum Schreiben der ConceptMap. Jeder Aufruf der Funktion innerhalb des Algorithmus verfügt über die Daten, die zum Schreiben einer *group* an Umsteigern mit Start- und Zielversion benötigt werden. Ein Durchlauf der vertikalen Suche kann dann mehrere Gruppen generieren, die ein Mapping auf eine Zielversionen ausgehend von allen anderen Versionen des Kodiersystems ermöglichen. Ebenfalls kann die Kombination mehrerer vertikalen Suche für jede Version als Zielversion aufgerufen werden, um so eine ConceptMap von allen Versionen auf alle Versionen zu schreiben.

Eine alle Versionen umfassende ConceptMap wird jedoch sehr groß. Zum Beispiel für die ICD-10-GM mit allen Codes aus allen Versionen im XML-Format ist die Datei circa 1 GB groß. Wie in (Braaksma, 2014) beschrieben macht es Sinn solche Dateien zu streamen, denn andernfalls müsste die komplette Datei im Arbeitsspeicher des Servers geschrieben werden. Die Vorgehensweise über die anonymen Funktionen ermöglicht genau das Streaming, weil nach jedem Aufruf der Ausgabepuffer des Servers geleert werden kann. Ein weiterer Vorteil ist, dass selbst wenn ein User sehr große ConceptMaps generieren lässt, der Download-Fortschritt über das Streaming angezeigt wird.

Konkret enthält jede Iteration der anonymen Funktion die Informationen über Zielversion, aktuelle verarbeitete Version, sowie die vereinigten Umsteiger über alle vorher verarbeiteten Versionen. Zusätzlich muss allerdings noch bekannt sein, welche Codes insgesamt in der Startversion vorhanden sind; zum einen, um Codes ohne Umsteiger abzudecken und zum anderen, um keine Umsteiger aus vorher verarbeiteten Version aufzunehmen, deren Ursprungskode in der aktuellen Version nicht existiert. Hierfür ist also ein weiterer `readData`-Aufruf notwendig. Aus dem Vergleich dieser Codes mit den Einträgen in der Merge-Datenstruktur können die Relationen für das Mapping bestimmt werden:

Kode in Merge-Daten enthalten?	Relation (R4)
Nein	<i>equivalent</i>
Ja, übergeleitet auf UNDEF	<i>unmatched</i>
Ja, übergeleitet auf einen Kode	<i>related-to</i>
Ja, überleitetet auf mehrere Kodes	<i>wider</i>

Ablauf des Schreibens einer ConceptMap mit einer einzelnen Zielversion:

1. Anfang der ConceptMap schreiben: *ConceptMap, url, id*.
2. Vertikale Suche mit anonymer Funktion starten; jeder Aufruf generiert eine *group*. Für jeden Kode wird ein *element→code, element→target→code* Paar geschrieben – oder bei Relation *wider* entsprechend mehrere *target→code* Einträge.
3. Ende der ConceptMap schreiben.

Eine Beispiel-ConceptMap im XML-Format befindet sich im Appendix A.3.

4

Web-Applikation

Dieses Kapitel behandelt die konkrete Implementierung einer Web-Applikation, welche die BfArM-Daten integriert und Ergebnisse der versionsübergreifenden Umsteiger-Suche zur Verfügung stellt.

4.1 Überblick

Die Web-Applikation heißt `bfarmer`, kurz für BfArM Electronic Repository und besteht aus folgenden Komponenten:

- XML-Dateien zur Konfiguration der ICD-10-GM und OPS Versionen.
- Ein Kommandozeilenbefehl, der:
 - Die BfArM-Daten herunterlädt und in einer Datenbank speichert.
 - Für Kodes ermittelt, ob sie Umsteiger besitzen, also Überleitungen auf geänderte Kodes in einer neueren oder älteren Version.
- Webseiten für ICD-10-GM und OPS zur Darstellung von:
 - Kodes für alle Versionen mit Titel.
 - Umsteigern zwischen jeweils zwei Versionen mit Informationen zu Titel und Überleitbarkeit.
 - Umsteiger-Suche über alle Versionen mit Anzeige der Veränderungen.
 - Formulare zum Generieren von ConceptMaps.
- Eine virtuelle Patientendatenbank mit zufällig generierten Daten, um Kode-Änderungen in der ICD-10-GM exemplarisch darzustellen.

Version Control

Zitat aus (Thomas und Hunt, 2019, Seite 163): „Always Use Version Control!“

Das `bfarmer`-Projekt steht hier zur Verfügung: <https://github.com/jacuke/ma>

Orthogonalität

Übersetzt aus (Thomas und Hunt, 2019, Seite 92): „Orthogonalität ist ein entscheidendes Konzept in der Programmierung von Systemen, die einfach zu entwerfen, bauen, testen und erweitern sind. Allerdings wird das Konzept der Orthogonalität selten direkt so genannt. Oft ist es eine implizite Eigenschaft verschiedener Methoden und Techniken.“

Beispielsweise wird Orthogonalität impliziert durch Begriffe wie Modularität, Abstraktion, Entkopplung und so weiter. Zwei Komponenten sind zueinander orthogonal, wenn Änderungen in der einen die andere nicht beeinflussen.

Symfony ist ein PHP-Framework, welches bewusst nach dem Konzept der Orthogonalität entworfen wurde, siehe (Potencier, 2022). Zum einen ist Symfony selbst in Komponenten zerteilt, die einzeln über den Paket-Manager Composer (Adermann, Boggiano u. a., o. D.) zu einem Projekt hinzugefügt werden können. Zum anderen implementiert es Software Design Patterns wie Model-View-Controller und Dependency Injection.

Model-View-Controller

MVC, auf deutsch Modell-Ansicht-Steuerung, ist ein Entwurfsmuster, welches vor allem für den Betrieb von Servern geeignet ist, um die Orthogonalität zu erhöhen durch Unterteilung der Architektur in drei voneinander unabhängige Komponenten:

Modell verwaltet die Daten; Ansicht stellt diese dar. Steuerung nimmt User-Interaktionen entgegen und modifiziert entsprechend Ansicht sowie Daten im Modell.

Zusätzlich ist oft noch eine Datenbank über das Modell angeschaltet.

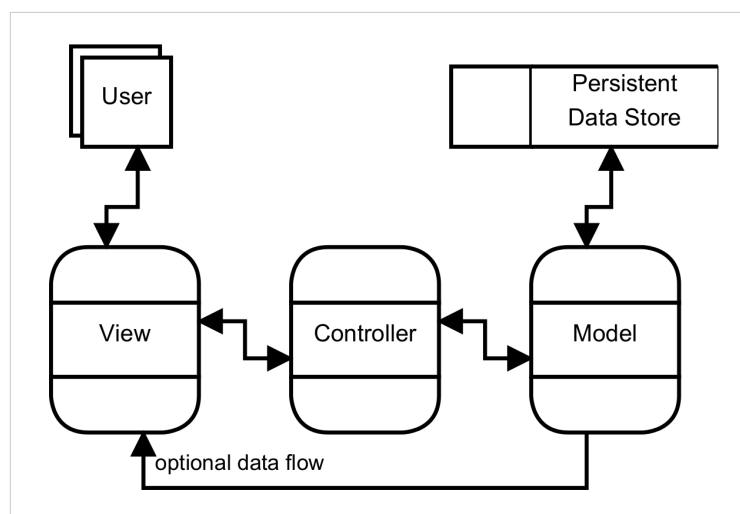


Abbildung 4.1: Model-View-Controller aus (Voorhees, 2020, Seite 177)

In Symfony heißen die Komponenten entsprechend *View*, *Controller* und *Modell* teilt sich auf in *Repository* für Datenbankzugriffe sowie *Service* für die Bereitstellung einer gewissen Funktionalität. *Utility* ist für sonstige Klassen vorgesehen und *Command* definiert Kommandozeilenbefehle außerhalb der MVC-Architektur.

Dependency Injection

Das Ziel des Entwurfsmusters Dependency Injection ist es die Instanziierung von Klassen einer zentralen Komponente zu überlassen, damit diese nicht von Änderungen untereinander abhängig sind und somit die Orthogonalität des Softwaresystem erhöht wird. Für eine ausführlichere Erklärung siehe (Seemann und Deursen, 2019). In Symfony heißt die zentrale Komponente Container. Sie erkennt automatisch Klassen und stellt diese ohne notwendige Instanziierung in den Konstruktoren anderer Klassen zur Verfügung.

Das ermöglicht die Verwendung eines Services durch einen anderen zum Beispiel sehr einfach so:

```
class UmsteigerService {  
  
    public function __construct(  
        private readonly DataService $dataService  
    ) {}  
  
    public function searchAllUmsteigerVertical (  
        string $system, $function = null): array {  
  
        foreach ($this->dataService->getVersions($system) as $version) {  
            // ...  
        }  
    }  
}
```

Symfony-Komponenten

Das `bfarmer`-Projekt verwendet folgende Komponenten des Symfony-Frameworks:

- Console Commands (Symfony, 2023a)
Zum Erstellen von Kommandozeilenbefehlen.
- HttpFoundation Component (Symfony, 2023b)
Für HTTP Request, Response, sowie Streaming.
- HTTP Client (Symfony, 2023c)
Um Downloads zu starten.
- Filesystem Component (Symfony, 2023d)
Für Dateioperationen.
- UID Component (Symfony, 2023e)
Zum Generieren von UUIDs = URIs in den ConceptMaps.
- Controller (Symfony, 2023f)
Für Routing.

- Databases and the Doctrine ORM (Symfony, 2023g)
Um Datenbankoperationen auszuführen.
- Templates (Symfony, 2023h) und Twig Template Engine (Twig, 2024)
Für das Rendern von Webseiten.
- Serializer Component (Symfony, 2023i)
Zur Konvertierung von Daten.

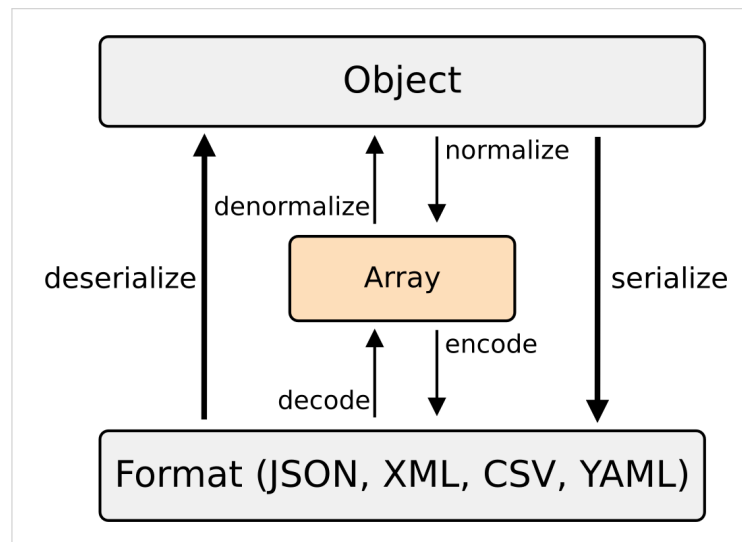


Abbildung 4.2: Schematische Darstellung der Serializer-Komponente (Symfony, 2023i)

Serializer ist die wichtigste Symfony-Komponente im `bfarmer`-Projekt. Sie dient zum:

- Lesen der CSV-Dateien, in der die BfArM-Daten strukturiert sind.
- Schreiben der ConceptMaps in XML oder JSON. Für eine nähere Erklärung der Dateiformate siehe (Bonney u. a., 2024, Seite 133ff).
- Konfigurieren der ICD-10-GM und OPS Versionen in XML-Dateien.
- Generieren von MySQL Insert-Statements mit einem eigenem Encoder.

Anmerkung zu PHP

Die Programmiersprache PHP hat unter Software-Entwicklern einen schlechten Ruf, obwohl –oder vielleicht auch weil– sie von circa 75% aller Webseiten verwendet wird laut (W3Techs, 2024). In (Janssens, Schultz und Zaytsev, 2017) wird allerdings diskutiert, dass nicht das Verwenden einer bestimmter Programmiersprache gute Entwickler kennzeichnet, sondern das Verständnis über Programmierparadigmen, die leichter zu erlernen sind, je mehr Erfahrung mit verschiedenen Programmiersprachen gewonnen wurde. Das `bfarmer`-Projekt ist bewusst möglichst modular, sowie unter Verwendung von möglichst vielen, weit verbreiteten Tools, Frameworks und Standardverfahren entwickelt, um auch die Übersetzung in eine andere Programmiersprache einfacher zu gestalten.

4.2 Aufbau des bfarmer -Projekts

Views

- base
Grundvorlage für alle Seiten, also zum Beispiel <head> mit Imports für Stylesheets, Javascript und <body> mit den Inhalten.
- index
Die Hauptseite der bfarmer -Applikation.
- codes
Seite, welche alle Codes einer Version anzeigt, inklusive Suchfunktion.
- conceptmap
Formular, um versionsübergreifende ConceptMaps zu generieren. Es kann eingestellt werden: Datei-Format, FHIR Release, Zielversion, ob Äquivalenzen ignoriert werden sollen.
- modal
Format für das Modal, in dem zum Beispiel Ergebnisse der Umsteiger-Suche ausgehend von einem Code angezeigt werden. Ein Modal ist wie ein Popup – außer dass kein neues Fenster geöffnet wird.
- umsteiger
Übersicht über alle Umsteiger-Einträge; angezeigt jeweils zwischen zwei Versionen. Außerdem kann eine ConceptMap zwischen den beiden Versionen generiert werden.
- umsteiger_icons
API-Schnittstelle, um eine Legende der Überleitung-Icons zu laden.
- umsteiger_search
Seite, die es erlaubt nach Umsteigern für einen Code zu suchen.
- umsteiger_search_recursion, umsteiger_search_result, search_code
Dienen zur rekursiven Anzeige der Umsteiger-Suchergebnisse.

Utility

- Constants
Enthält globale Konstanten und statische Funktionen, um zum Beispiel eindeutige Namen für die SQL-Tabellen zu generieren.
- SqlInsertEncoder
Fügt mehrere Datensätze zu einem INSERT-Statements zusammen für die Integration. Dadurch werden die Daten wesentlich schneller in die Datenbank eingelesen, siehe (Oracle, o. D.[a]).
- TwigExtension
Erweiterung der Twig Template Engine.

Repositories

- `BfarmRepository`
Dient zum Lesen und Schreiben der BfArM-Daten.
- `ConfigRepository`
Speichert Informationen für den `bfarmer` -Kommandozeilenbefehl.
- `DatabaseRepository`
Parent-Klasse der Repositories, um die Verbindung zur Datenbank aufzubauen. Stellt außerdem einen Wrapper für Operationen zur Verfügung, der SQL-Fehler loggt.

Controllers

- `AdminController`
Lädt die Datenbank-GUI von (Vrána, 2021).
- `CodesController`, `ConceptMapController`, `IndexController`,
`UmsteigerController`, `UmsteigerSearchController`
Reagieren auf Requests an die zugehörigen Views / API-Schnittstellen und generieren entsprechend Responses. Die Routes sind über Annotationen definiert.
- `DimdiController`
Stellt eine API-Schnittstelle für die externen Kode-Links der älteren ICD-10-GM und OPS Versionen zur Verfügung, siehe Abschnitt 4.5.

Services

- `ClientService`
Lädt Dateien und Inhalte von URLs herunter.
- `ConceptMapService`
Enthält die Funktionen zum Schreiben der ConceptMap, siehe Abschnitt 3.4.
- `DataService`
Liest Informationen über die ICD-10-GM / OPS Versionen aus den XML-Dateien.
- `SetupService`
Implementation des Datenintegrationsprozess.
- `UmsteigerService`
Implementation der horizontalen und vertikalen Umsteiger-Suche.

Commands

- `BfarmerCommand`
Kommandozeilenbefehle für die Datenintegration, siehe Abschnitt 4.3.
- `TestCommand`
Startet Tests, wie zum Beispiel den Vergleich zwischen den Ergebnissen aus der horizontalen und vertikalen Umsteiger-Suche.

Beispielanwendung

Es gibt außerdem jeweils View, Repository, Controller, Service und Command für die Beispielanwendung virtuelle Patientendatenbank, siehe Abschnitt 5.1 im nächsten Kapitel.

4.3 Backend

Setup-XML

Die Versionen von ICD-10-GM und OPS, sowie die Abweichungen zwischen den Versionen sind in den XML-Dateien `files/icd10gm.xml` und `files/ops.xml` konfiguriert. Hier ein Auszug aus letzterer:

```
<?xml version="1.0" encoding="UTF-8"?>
<ops_setup>
  <ops>
    <year>2024</year>
    <url>https://multimedia.gsb.bund.de/BfArM/downloads/klassifikationen/ops/
      version2024/ops2024syst-ueberl.zip</url>
  </ops>
  <ops>
    <year>2023</year>
    <url>https://multimedia.gsb.bund.de/BfArM/downloads/klassifikationen/ops/
      version2023/ops2023syst-ueberl.zip</url>
  </ops>
<!-- ... -->
```

Im Idealfall muss nur die Version und der Download-Link angegeben werden.

Kommandozeilenbefehle

Die Verarbeitung der Einträge in der XML-Konfiguration wird mit einem Kommandozeilenbefehl gestartet, der im Hauptverzeichnis des Projekts ausgeführt wird:

```
bin/console bfarmer --setup
```

Dadurch wird pro Version der Datenintegrationsprozesses aus Abbildung 2.2 angestoßen. Für erfolgreiche integrierte Versionen wird ein Eintrag in der CONFIG-Tabelle erstellt, damit diese beim nächsten Aufruf übersprungen werden. Es gibt Optionen:

```
--icd10gm -i | Nur für ICD-10-GM ausführen.
--ops      -o | Nur für OPS ausführen.
--keep     -k | Die heruntergeladene Datei nicht löschen.
```

Folgender Befehl speichert zu jedem Code, ob er Umsteiger hat: `true/false`. Ebenfalls mit Eintrag in der CONFIG-Tabelle, sowie den Optionen `-i/-o`.

```
bin/console bfarmer --umsteiger
```

Nach Erweiterung der XML-Datei um eine neue Version müssen also nur die zwei Befehle ausgeführt werden, um diese in die `bfarmer`-Applikation aufzunehmen.

Datenbank

Für die Datenbank wird MySQL (Oracle, o. D.[b]) als DB Management System verwendet und für den Verbindungsaufbau (Doctrine, o. D.). Nach aktuellen Entwicklungsstand laufen alle DB-Operationen über SQL-Befehle in MySQL, was zwar die Abstraktion der Datenbank verhindert, aber die Portabilität des `bfarmer`-Projekt erhöht.

Das BfArM schlägt folgende Tabellen in Microsoft-Access-Syntax vor, um die Kodes- und Umsteiger-Dateien zu importieren, zum Beispiel für ICD-10-GM 2024:

```
CREATE TABLE ICD10V2024 (
    Code2024    Text(7)    CONSTRAINT Code2024X PRIMARY KEY,
    Titel2024   Text(255)
);

CREATE TABLE UMSTEIGER (
    Code2023     Text(7) CONSTRAINT UICD2023 REFERENCES ICD10V2023,
    Code2024     Text(7) CONSTRAINT UICD2024 REFERENCES ICD10V2024,
    Auto2023_2024 Text(1),
    Auto2024_2023 Text(1)
);
```

Stattdessen werden die Tabellen in `bfarmer` so angelegt:

```
CREATE TABLE `ICD10GM_2024` (
    `code` varchar(7) NOT NULL,
    `name` varchar(255) DEFAULT NULL,
    `umst` tinyint(1) DEFAULT NULL,
    PRIMARY KEY (`code`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE `UMSTEIGER_ICD10GM_2024_2023` (
    `old` varchar(7) NOT NULL,
    `new` varchar(7) NOT NULL,
    `auto` varchar(1) DEFAULT NULL,
    `auto_r` varchar(1) DEFAULT NULL,
    PRIMARY KEY (`new`,`old`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Die Spaltennamen sind pro Tabellentyp gleich, was das Lesen vereinfacht. Hingegen sind die Tabellennamen eindeutig pro Kodiersystem und Version. Der Key für die Umsteiger ist umgedreht, weil normalerweise die neueren Versionen links angezeigt werden. Außerdem gibt es Gründe Foreign Key Constraints zu vermeiden, siehe (PlanetScape, 2023).

4.4 Frontend

Das Frontend basiert auf den Standardtechnologien HTML (WHATWG, o. D.), zur Strukturierung der Inhalte auf den Seiten, CSS (W3C, o. D.), für Layout und Styling der Inhalte und JavaScript (Ecma International, o. D.) für die Client-seitige, interaktive Veränderung von Inhalten. Beispielsweise wird das Ergebnis der horizontale Suche für M21.6 der ICD-10-GM 2013 so dargestellt:

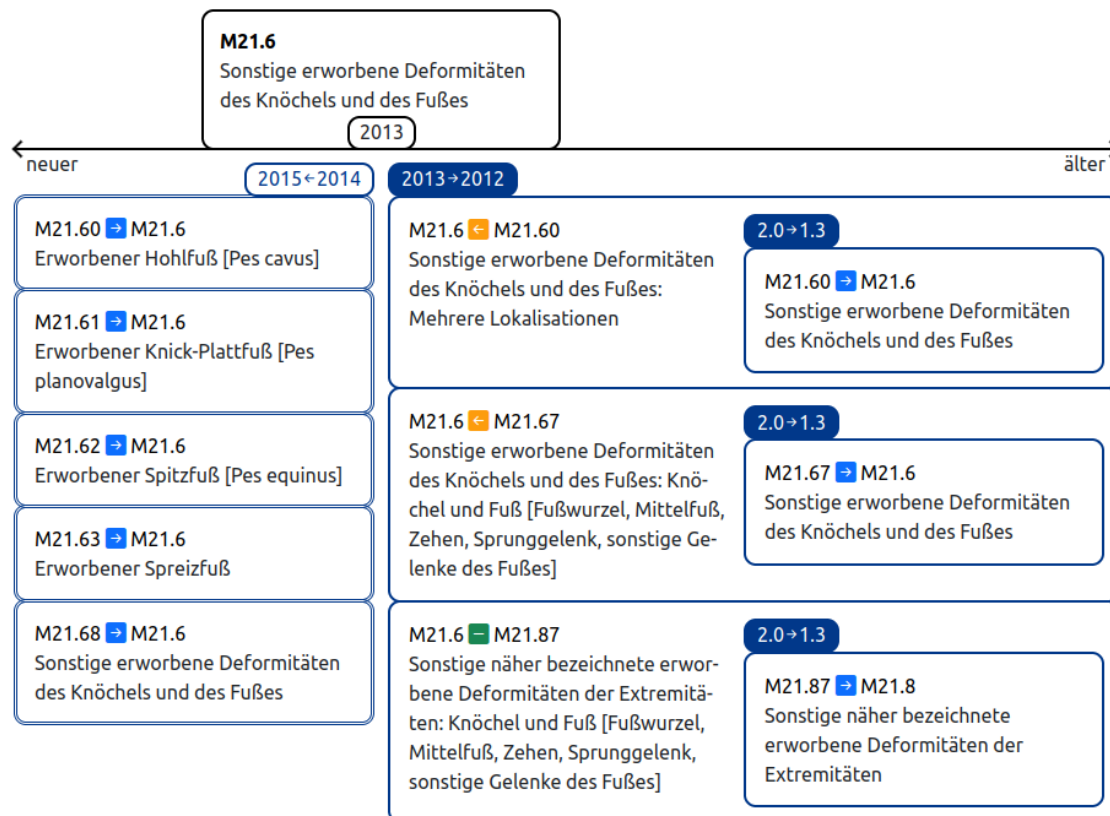


Abbildung 4.3: Verwendung von HTML und CSS

Da das Ergebnis der horizontale Suche rekursiv aufgebaut ist, werden die Daten auch rekursiv gerendert. Die Variablen resultieren in verschiedene <div>-Blöcke, die jeweils mit einer eigenen CSS-Klasse gekennzeichnet sind. Eine Gruppe an Umsteigern wird also vertikal angeordnet unterhalb der Versionsnummern. Jeder Umsteiger-Eintrag erhält einen Rahmen und jeder Rekursionsschritt schiebt die anderen Ergebnisse nach außen. Dieses Layout basiert hauptsächlich auf Flexbox, siehe (Coyier, o. D.).

Außerdem verwendet werden:

- SASS/SCSS (Catlin, Weizenbaum und Eppstein, o. D.), um das Schreiben von CSS zu vereinfachen. Aus der Skriptsprache SCSS wird CSS kompiliert. Merkmale sind beispielsweise eine geschachtelte Syntax, welche die Verwendung von CSS-Selektoren

vereinfacht, sowie Variablen und Funktionen, zum Beispiel für die Berechnung von Pixelbreiten.

- Popper.js (Popper JS, o. D.) für Tooltips.
- Bootstrap (Otto, Thornton u. a., o. D.) für das Basis-Layout, Header-Banner, Navigation-Tabs, Modals und Icons.
- jQuery (Resig u. a., 2023), um beispielsweise Modal-Content per AJAX zu füllen, siehe den nächsten Abschnitt 4.5.

Eingebunden werden diese Tools über Webpack (Koppers u. a., o. D.), einem JavaScript-Modul-Packer, der wiederum auf der JavaScript-Laufzeitumgebung Node.js (Dahl, o. D.) und dem zugehörigen JavaScript-Paketmanager NPM (Schlueter, o. D.) aufbaut.

4.5 AJAX

Der Begriff AJAX, *Asynchronous JavaScript and XML*, wurde erstmals in (Garrett, 2005) geprägt, beschreibt aber damals schon verbreitete Ansätze zur Interaktion mit Webseiten.

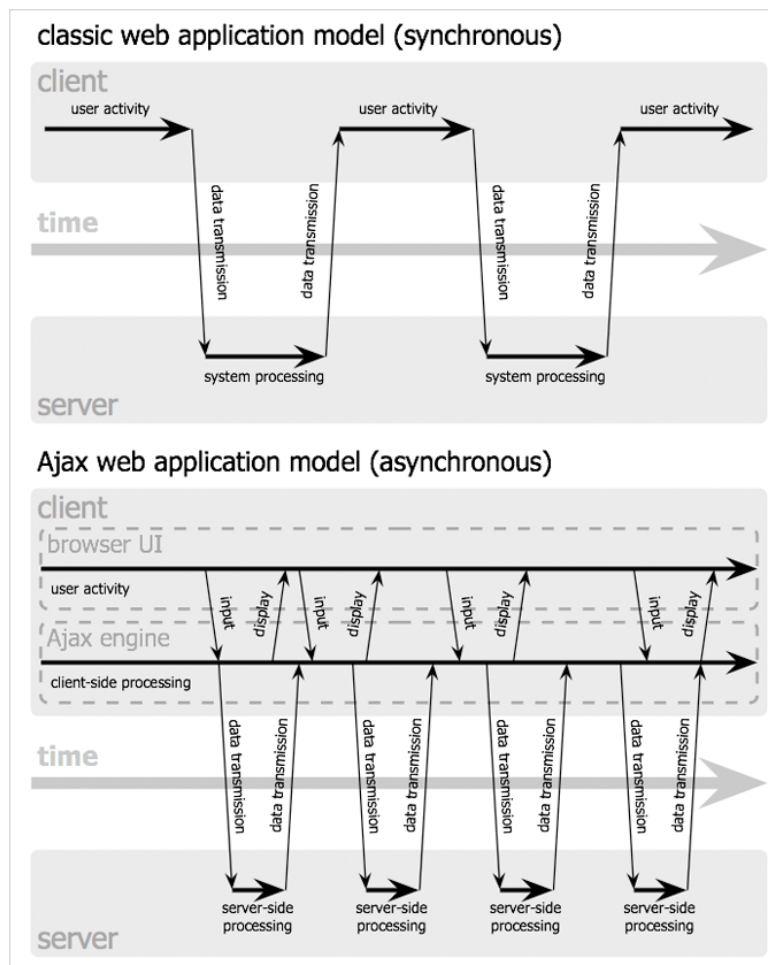


Abbildung 4.4: Klassisches und AJAX Web-Applikations-Modell nach (Garrett, 2005).

Statt dass wie gewöhnlich eine Interaktion das Laden einer ganzen Seite auslöst, erlaubt die AJAX-Engine durch Versenden von Requests, was als JavaScript Client-seitig passiert, das Verändern einzelner Teile einer Seite. Anders als der Name andeutet muss es sich dabei aber nicht um XML-Daten handeln, sondern AJAX funktioniert über jede Art von HTTP-Request.

In bfarmer wird AJAX verwendet, um Bootstrap-Modals nach Klick auf ein Umsteiger-Icon mit Inhalt zu füllen. Dazu mehr im Abschnitt 5.1.

Externe Links zu den BfArM/DIMDI Kode-Seiten

Außerdem dient AJAX dazu, externe Links auf die BfArM- beziehungsweise DIMDI-Seiten zu öffnen, welche mehr Information für Kodes anzeigen. Diese Links werden per Klick auf jeden Kode auf den bfarmer -Seiten bereitgestellt.

Dabei besteht die Schwierigkeit besteht darin, dass die URLs für die Kode-Informationen auf den BfArM- und DIMDI-Seiten nicht einfach nur durch Kodiersystem, Version und Kode besteht. Sondern es muss zusätzlich noch die Kode-Gruppe ermittelt werden. Das ist nicht trivial, weil die Gruppe nicht über den Kode bestimmt werden kann.

Beispiel: <https://www.dimdi.de/static/de/klassifikationen/icd/icd-10-gm/kode-suche/htmlgm2013/block-m20-m25.htm#M21> – Die Gruppe ist m20-m25. Manchmal enthält die Gruppe aber gar nicht die Zeichen des Kodes.

Die BfArM- und DIMDI-Server stellen zwar JavaScript-Dateien zur Verfügung, die eine Funktion zum Bestimmen der Gruppe für einen gegebenen Kode enthält, allerdings heißt diese Funktion immer gleich und jede Datei enthält immer nur die Gruppen für eine Version. Das heißt es ist nicht möglich, wenn auf den bfarmer -Seiten zum Beispiel im Ergebnis der Umsteiger-Suche mehrere Kodes von mehreren Versionen angezeigt werden, alle diese Dateien zu laden, um die Gruppen zu bestimmen, weil die Funktionen in den Dateien sich gegenseitig überschreiben und so über diese Vorgehensweise immer nur pro Seite die Gruppen *einer* Version ermittelt werden könnten.

Stattdessen muss beim Klick auf einen Kode die jeweils passende Datei der Version des Kodes per AJAX geladen und ausgeführt werden. Die Dateien befinden sich ab 2015 auf der BfArM-Seite und vorher auf dem DIMDI-Seite, das heißt hier muss außerdem noch eine Differenzierung passieren. Weiterhin kommt erschwerend hinzu, dass für ICD-10-GM Versionen älter als 2008 und OPS Versionen älter als 2009 die DIMDI-Seiten die JavaScript-Funktionen gar nicht als einzelne Dateien anbieten, sondern diese ist innerhalb von HTML-Seiten eingebettet. Um wiederum dieses Problem zu lösen, gibt es eine zusätzliche API-Schnittstelle im bfarmer -Projekt, welche die HTML-Seiten dieser älteren Versionen lädt, die JavaScript-Funktionalität extrahiert und zurückgibt. Der AJAX-Aufruf geht dann also erst an die eigene Schnittstelle.

Weil das Öffnen von Links nach einem AJAX-Aufruf von Browsern als Popup wahrgenommen und meist blockiert wird, muss das Fenster oder der Tab in dem die BfArM- oder DIMDI-Seite geladen werden soll, schon vor Ausführen des AJAX-Codes geöffnet werden.

5


Zusammenfassung

5.1 Ergebnisse

Beispielanwendung: Virtuelle Patientendatenbank

Um die Veränderungen von Codes über die Versionen der ICD-10-GM beispielhaft darzustellen, wurde ein Prozess implementiert, der zufällige Patientendaten generiert. Hierzu werden Codes der verschiedenen Versionen ausgewählt und zu mehreren virtuellen Patienten zusammengefügt. Abgespeichert werden jeweils: Code und Titel, sowie die Angabe, ob der Code zu anderen Versionen Umsteiger hat. Das passiert im JSON-Format, damit auf der Patienten-Seite auch nach Code und Titel gesucht werden kann. Abhängig von der Information, ob Umsteiger vorhanden sind, erscheint neben dem Code eine Lupe.

Hier beispielhaft die mit 20.000 Patienten gefüllte Datenbank. Gesucht wird nach allen Codes, die den Titel „Lähmungssyndrome“ enthalten. H40.5 hat keine Umsteiger und daher keine Lupe.

 bfarmer

Test-Datenbank ICD-10-GM
Patienten insgesamt: 20000

Code: Titel: 4 Treffer

Patient	Jahr	Code		Titel
10408	2023	G83.8	🔍	Sonstige näher bezeichnete Lähmungssyndrome
		K58.8	🔍	Sonstiges und nicht näher bezeichnetes Reizdarmsyndrom
		S42.44	🔍	Fraktur des distalen Endes des Humerus: Epicondylus, Epicondyli, nicht näher bezeichnet
05248	2020	G83.8	🔍	Sonstige näher bezeichnete Lähmungssyndrome
		K41.30	🔍	Hernia femoralis, einseitig oder ohne Seitenangabe, mit Einklemmung, ohne Gangrän: Nicht als Rezidivhernie bezeichnet
		M12.56	🔍	Traumatische Arthropathie: Unterschenkel [Fibula, Tibia, Kniegelenk]
		S83.43	🔍	Verstauchung und Zerrung des Kniegelenkes: Riss des fibularen Seitenbandes [Außenband]
03611	2013	G83.88	🔍	Sonstige näher bezeichnete Lähmungssyndrome
		H40.5		Glaukom (sekundär) nach sonstigen Affektionen des Auges
00352	1.3	G83.8	🔍	Sonstige näher bezeichnete Lähmungssyndrome

Abbildung 5.1: Virtuelle Patientendatenbank

Alle anderen Codes haben Umsteiger und diese können per Klick auf die Lupe angezeigt werden. Hier zum Beispiel der G83.8 der Version 1.3. Das Modal wird per AJAX geladen.

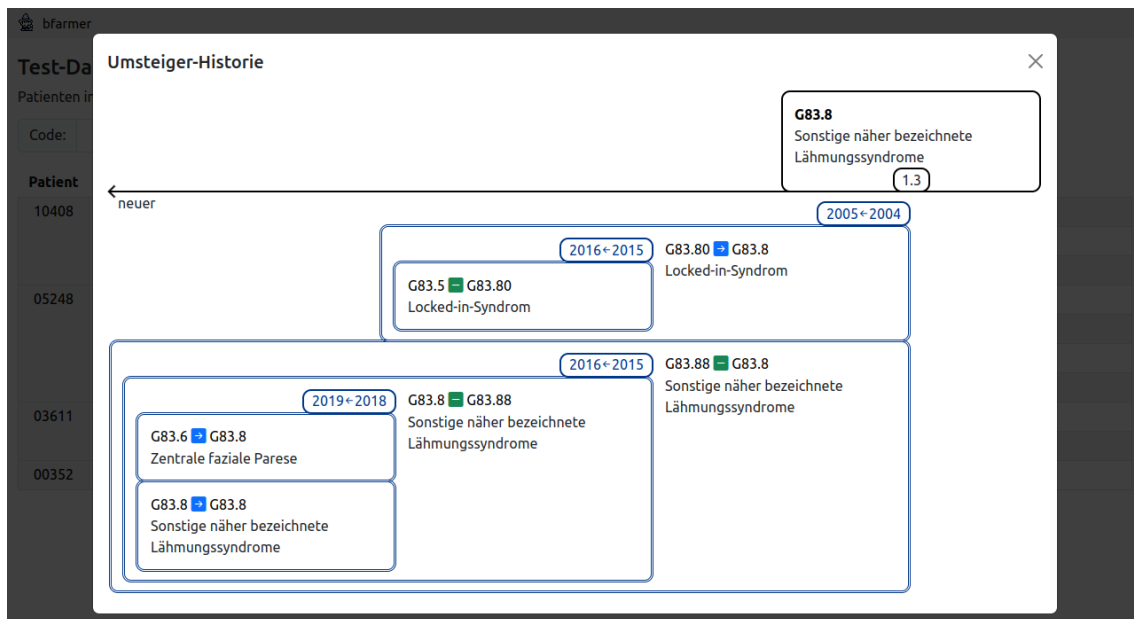


Abbildung 5.2: Umsteiger von G83.8 Version 1.3

Diese Funktionalität kann interoperabel verwendet werden. Vorausgesetzt ist, dass der Server „Cross-Origin Resource Sharing“ auf der Umsteiger-API erlaubt. Dafür muss ein Response-Header gesetzt sein: Access-Control-Allow-Origin:.*.

Wenn das der Fall ist, kann die Umsteiger-Suche mit folgenden Schritten auch auf einer anderen Seite angezeigt werden:

1. bfarmer -Stylesheet im <head> laden:

```
<link rel="stylesheet" href="http://127.0.0.1:8000/build/app.css">
```

2. Stylesheets für Bootstrap und Bootstrap-Icons im <head> laden:

```
<link
  rel="stylesheet" crossorigin="anonymous"
  href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
  integrity=
"sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNllyT2bRjXh0JMhY6hW+ALEwIH"
>
<link
  rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css"
>
```

3. Bootstrap-JavaScript im <head> laden:

```
<script crossorigin="anonymous"
  src=
  "https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
  integrity=
  "sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDz0xhy9GkcIdslK1eN7N6jIeHz">
</script>
```

4. jQuery-Library im <head> laden:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js">
</script>
```

5. AJAX-Funktion im <head> bereitstellen:

```
<script>
  window.ajaxUmsteigerSearchHistory = function(system, version, code) {
    $.get('http://127.0.0.1:8000/umsteiger-suche-api?s=' + system + '&v=' +
      version + '&c=' + code, function(data) {
        $('#edit-modal .modal-content').html(data);
        $('#edit-modal .modal-title').html('Umsteiger-Historie');
      });
  }
</script>
```

6. Leeres Modal im <body> platzieren:

```
<div class="modal fade" id="edit-modal" tabindex="-1" aria-hidden="true">
  <div class="modal-dialog"><div class="modal-content"></div></div>
</div>
```

7. AJAX-Funktion beim Kode verlinken:

```
<a class="umsteiger-search-link" href="" data-bs-toggle="modal"
  data-bs-target="#edit-modal"
  onclick="ajaxUmsteigerSearchHistory('icd10gm','1.3','G83.8')">
  Test G83.8 1.3
</a>
```

Gegenseitige Validierung der Suchalgorithmen

Die beiden Suchalgorithmen können verwendet werden, um deren Ergebnisse gegenseitig zu validieren. Ein ähnliches Verfahren wird beispielsweise in (Thurin u. a., 2021) verwendet, um bekannte Diagnosen in der gleichen Datenbank über die Auswertung anderer Daten zu bestätigen.

Das Ergebnis der horizontalen Suche für ICD-10-GM Kode G83.8, ausgehend von Version 1.3 ist in Abbildung 5.2 dargestellt. Analog kann das Ergebnis der vertikalen Suche über die ConceptMap für ICD-10-GM mit Zielversion 2024 bestimmt werden. Zum Beispiel unter Verwendung von cURL (Stenberg u. a., o. D.) und HAPI:

```
curl -X 'POST' \  
  'http://localhost:8080/fhir/ConceptMap' \  
  -H 'accept: application/fhir+json' \  
  -H 'Content-Type: application/fhir+xml' \  
  -d "@/home/simon/Downloads/conceptmap_r4_icd10gm_2024.xml"
```

```
curl -X 'GET' \  
  'http://localhost:8080/fhir/ConceptMap/$translate?code=G83.8&system=1.3' \  
  -H 'accept: application/fhir+json'
```

Das Ergebnis ist ebenfalls: [G83.6, G83.8, G83.5]. Natürlich kann der Vergleich der beiden Suchalgorithmen auch programmiert werden.

Qualität der Mappings

Die durch diese Arbeit generierten ConceptMaps können anhand der Qualitätsstandards aus Abschnitt 1.2 beurteilt werden:

1. Einsatzzweck und Umfang der ConceptMaps sind klar definiert sein. Mappings erfolgen zwar chronologisch in zwei Richtungen, aber mit dem gleichen Algorithmus.
2. Die ConceptMaps richten sich nach den vom BfArM erstellten Überleitungstabellen.
3. Als Masterarbeit sind die Anforderung an das Team nicht erfüllt. Vertreter des BfArMs waren nicht beteiligt.
4. Diese Arbeit dokumentiert das Mapping und der Programmcode ist frei verfügbar.
5. Getestet wurde automatisiert durch Abgleich der zwei Suchalgorithmen, aber manuell nur stichprobenartig.
6. Der Datenintegrationsprozess wurde bewusst so entworfen, dass ein Update auf neue Versionen möglichst einfach ist. Änderungen am Programmcode sind nur notwendig, wenn sich die BfArM-Daten oder deren Veröffentlichung essentiell ändern.
7. FHIR ConceptMaps erfordern die Definition von Kardinalität, Relationen, sowie Ursprung- und Zielversion des Mappings.
8. FHIR ConceptMaps sind maschinell lesbar.

5.2 Zusammenfassung

Es wurden zwei Suchalgorithmen zur Bestimmung von Umsteigern über alle Versionen der ICD-10-GM und des OPS vorgestellt. Einer dient zum Abbilden vieler Informationen zu einzelnen Codes und der andere zur effizienten Erstellung von ConceptMaps. Beide Varianten können interoperabel eingesetzt werden. Die Aufnahme von neuen Versionen sollte durch den Datenintegrationsprozess gewährleistet sein. Wichtig ist noch das manuelle Testen der Mappings.

5.3 Ausblick

Folgende Weiterentwicklungen sind sinnvoll.

RESTful

Die Web-Applikation sollte nicht nur teilweise, sondern komplett dem REST-Paradigma entsprechen, wie in Abschnitt 1.1 aufgeführt. Das bedeutet alle Informationen des Servers durch Schnittstellen bereitzustellen:

- Kodes pro System und Version als FHIR „Code System“ Ressourcen zum Download.
- Umsteiger pro Version im JSON-Format per API-Schnittstelle. Die Übersichtsseite für Umsteiger lädt aktuell schon relativ lange aufgrund der Menge an Daten und das könnte per AJAX beschleunigt werden, das bedeutet Laden der Umsteiger jeweils nur für die angezeigte Version.
- Ergebnis der horizontalen Suche im JSON-Format per API-Schnittstelle. Ähnlich wie der Modal-Content, aber ohne Formatierung für die Anzeige.
- ConceptMaps per API-Schnittstelle ohne Formular, welches nur manuell ausgefüllt werden kann.

Außerdem sollten die Schnittstellen dokumentiert sein über Swagger UI (Weaver, 2023).

Bei einem wirklich produktiv eingesetzten System wäre es sinnvoll Daten zu cachieren:

- Umsteiger-Ergebnisse der horizontalen Suche.
- ConceptMap-Dateien, damit diese nicht immer neu generiert werden müssen.

Das würde allerdings sehr viel mehr an Speicherplatz belegen.

ATC

Das BfArM stellt neben ICD-10-GM und OPS auch die deutsche Version der ATC Klassifikation, *Anatomical Therapeutic Chemical*, zur Verfügung unter (BfArM, o. D.[a]). Diese wird im Auftrag durch die WIdO erstellt, dem Wissenschaftliche Institut der Allgemeinen Ortskrankenkassen. Die Klassifikations- und Änderungsdateien gibt es nur als PDF. Die Aufnahme von ATC in den bfarmer -Datenintegrationsprozess würde also einen PDF-Parser erfordern.

Mit (Poppler, o. D.) können PDF-Dateien in XML-Dateien umgewandelt werden:

```
pdftohtml atc-vergleichsdatei-amtlich-2024-2023.pdf test.xml -nodrm -i -xml
```

Das Parsen wird dadurch vereinfacht, dass alle Tabellen in den ATC PDF-Dateien mit „Tabelle“ + Zahl beginnen und mit „Quelle: GKV-Arzneimittelindex im Wissenschaftlichen Institut der AOK (WIdO)“ enden. Zum Beispiel:

3 Änderungen der ATC-Bedeutung

Tabelle 3: Änderungen der ATC-Bedeutung, sortiert nach ATC-Code

ATC-Code	ATC-Bedeutung Amtlich 2023	ATC-Bedeutung amtlich 2024
J07BX01	Pocken-Impfstoffe	Pocken und Affenpocken-Impfstoffe

Quelle: GKV-Arzneimittelindex im Wissenschaftlichen Institut der AOK (WIdO)

© WIdO 2023

Abbildung 5.3: Beispiel Überleitungstabelle in einer ATC PDF-Datei.

Die konvertierte XML-Datei enthält Text immer in `<text>`-Tags und durch Auslesen des Inneren dieser Tags ergibt sich:

```
<b>Tabelle 3: Änderungen der ATC-Bedeutung, sortiert nach ATC-Code </b>
<b>ATC-Code </b>
<b>ATC-Bedeutung </b>
<b>Amtlich 2023 </b>
<b>ATC-Bedeutung </b>
<b>amtlich 2024 </b>
J07BX01
Pocken-Impfstoffe
Pocken und Affenpocken-Impfstoffe
<i>Quelle: GKV-Arzneimittelindex im Wissenschaftlichen Institut der AOK (WIdO)
  </i>
<b>© WIdO 2023 </b>
```

Die Tabellen können eine unterschiedliche Anzahl an Spalten haben, aber die Anzahl sowie die Überschrift bleibt pro Tabellentyp gleich. Die Zeilen beginnen immer mit einem Code und dieser könnte per regulärem Ausdruck identifiziert werden. Es müssten anhand der unterschiedlichen Tabellen Umsteiger-Informationen analog zu ICD-10-GM und OPS generiert werden. Außerdem müsste die Information über DDD, die definierte Tagesdosis, als zusätzliche Tabelle gespeichert und extra für ATC angezeigt werden.

Sessions

Ein Session-Management würde es ermöglichen, dass nicht nur die Zielversion, sondern auch die Ausgangsversionen für ConceptMaps wählbar sind. Aktuell sind immer alle Versionen ausgewählt, aber das könnte zum Beispiel durch Checkboxes einstellbar gemacht werden. Diese Einstellungen müssen allerdings als Session-Informationen gespeichert werden. Das ist Client- und Server-seitig möglich. Client-seitig würde den Einbau eines Cookie-Banners notwendig machen. Server-seitig ist es komplizierter umzusetzen und erfordert zusätzlichen Speicherplatz in der Datenbank.

Literatur

Adermann, N., Boggiano, J. u. a. (o. D.). *Composer. A Dependency Manager for PHP*. URL: <https://getcomposer.org/doc/> (besucht am 07. 11. 2024).

BfArM (o. D.[a]). *BfArM – ATC. ATC-Klassifikation*. URL: https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/ATC/_node.html (besucht am 23. 10. 2024).

BfArM (o. D.[b]). *BfArM – Downloads*. URL: https://www.bfarm.de/DE/Kodiersysteme/Services/Downloads/_node.html (besucht am 07. 11. 2024).

BfArM (o. D.[c]). *BfArM – Geschichte – Geschichte des ehemaligen DIMDI. Geschichte des ehemaligen DIMDI*. URL: <https://www.bfarm.de/DE/Das-BfArM/Organisation/Geschichte/geschichte-dimdi.html> (besucht am 07. 11. 2024).

BfArM (o. D.[d]). *BfArM – Systematisches Verzeichnis – Kategorie und Kode. Kategorie und Kode in der ICD-10-GM*. URL: <https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/ICD/ICD-10-GM/Systematik/kodestruktur.html> (besucht am 07. 11. 2024).

BfArM (o. D.[e]). *BfArM – Systematisches Verzeichnis – Kategorie und Kode. Kategorie und Kode im OPS*. URL: <https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/OPS-ICHI/OPS/Systematik/kategorie-und-kode.html> (besucht am 07. 11. 2024).

Bonnefoy, P., Chaize, E., Mansuy, R., Tazi, M. und Heckel, S. (2024). *The Definitive Guide to Data Integration: Unlock the Power of Data Integration to Efficiently Manage, Transform, and Analyze Data*. Packt Publishing. ISBN: 9781837634774.

Braaksma, A. (2014). *Streaming Design Patterns or: How I Learned to Stop Worrying and Love the Stream*. XML LONDON 2014, 24–52. URL: <http://xmllondon.com/2014/xmllondon-2014-proceedings.pdf> (besucht am 07. 11. 2024).

Braunstein, M. (2022). *Health Informatics on FHIR: How HL7's API is Transforming Healthcare*. Health Informatics. Springer International Publishing. ISBN: 9783030915636.

Catlin, H., Weizenbaum, N. und Eppstein, C. (o. D.). *SASS*. URL: <https://sass-lang.com/documentation/> (besucht am 07. 11. 2024).

Coyier, C. (o. D.). *CSS Flexbox Layout Guide*. URL: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> (besucht am 07. 11. 2024).

Dahl, R. (o. D.). *Node.js*. URL: <https://nodejs.org/docs/latest/api/> (besucht am 07.11.2024).

Dar, S. (1993). *Augmenting Databases with Generalized Transitive Closure*. Computer Sciences Technical Report. University of Wisconsin–Madison.

Doctrine (o. D.). *Doctrine: PHP Open Source Project*. URL: <https://www.doctrine-project.org/> (besucht am 07.11.2024).

Ecma International (o. D.). *ECMAScript. JavaScript*. URL: <https://ecma-international.org/publications-and-standards/standards/ecma-262/> (besucht am 07.11.2024).

Essaid, S., Andre, J., Brooks, I.M., Hohman, K.H., Hull, M., Jackson, S.L., Kahn, M.G., Kraus, E.M., Mandadi, N., Martinez, A.K. u. a. (2024). MENDS-on-FHIR: Leveraging the OMOP Common Data Model and FHIR Standards for National Chronic Disease Surveillance. *JAMIA Open* 7, ooae045. doi: 10.1093/jamiaopen/ooae045. URL: <https://academic.oup.com/jamiaopen/article-pdf/7/2/ooae045/58000493/ooae045.pdf> (besucht am 07.11.2024).

Fernández, I. und Manuel, J. (2022). *UTF-8 & Latex Encodings of ISO-8859 (Latin-1) Character Set*. doi: 10.13140/RG.2.2.18402.61121. URL: https://www.researchgate.net/publication/359509972_UTF-8_Latex_Encodings_of_ISO-8859_Latin-1_Character_Set/link/6241ab3b5e2f8c7a03452ac9/download (besucht am 07.11.2024).

Fielding, R.T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral Dissertation. University of California, Irvine. URL: <https://ics.uci.edu/~fielding/pubs/dissertation/top.htm> (besucht am 07.11.2024).

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J.M. und Herrera, F. (2016). Big Data Preprocessing: Methods and Prospects. *Big Data Analytics* 1, 1–22. doi: 10.1186/s41044-016-0014-0. URL: <https://bdataanalytics.biomedcentral.com/articles/10.1186/s41044-016-0014-0> (besucht am 07.11.2024).

Garrett, J.J. (2005). *Ajax: A New Approach to Web Applications*. URL: <https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php> (besucht am 07.11.2024).

Gaus, W. (2005). *Dokumentations- und Ordnungslehre*. Springer. ISBN: 9783540275183.

Gotoh, O. (1982). An Improved Algorithm for Matching Biological Sequences. *Journal of Molecular Biology* 162, 705–708. URL: <https://pubmed.ncbi.nlm.nih.gov/7166760/> (besucht am 07.11.2024).

Gross, J., Yellen, J. und Zhang, P. (2013). *Handbook of Graph Theory, Second Edition. Discrete Mathematics and Its Applications*. Taylor & Francis. ISBN: 9781439880180.

Heckmann, S. (2022). *Digitalstrategie im Krankenhaus: Einführung und Umsetzung von Datenkompetenz und Compliance*. Springer Fachmedien Wiesbaden. Kap. HL7® FHIR®

als Grundlage für moderne Digitalstrategien, S. 307–331. ISBN: 9783658362263. DOI: 10.1007/978-3-658-36226-3_21.

Heidel, A., Hoffmann, M. und Ammon, D. (2024). *Translation of ICD-10-GM codes to SNO-MED CT*. URL: <https://www.egms.de/static/en/meetings/gmds2024/24gmds168.shtml> (besucht am 07. 11. 2024).

HL7 (2019). *ConceptMap of FHIR Specification R4. Resource ConceptMap*. URL: <http://hl7.org/fhir/R4/conceptmap.html> (besucht am 07. 11. 2024).

HL7 (2023). *ConceptMap of FHIR Specification R5. Resource ConceptMap*. URL: <http://hl7.org/fhir/R5/conceptmap.html> (besucht am 07. 11. 2024).

Hund, H. (2018). *medicats. Medical classification and terminology systems library for Java 1.8*. URL: <https://github.com/hhund/medicats> (besucht am 07. 11. 2024).

Hund, H., Gerth, S., Katus, H. und Fegeler, C. (2016). Medical Classification and Terminology Systems in a Secondary Use Context: Challenges and Perils. *Studies in Health Technology and Informatics* 228, 394–398.

ISO (2014). *ISO/TR 12300:2014 – Principles of Mapping between Terminological Systems*. International Organization for Standardization, Technical Committee: Health Informatics. URL: <https://www.iso.org/standard/51344.html> (besucht am 07. 11. 2024).

ISO (2019). *ISO/TS 21564:2019 – Terminology Resource Map Quality Measures (MapQual)*. International Organization for Standardization, Technical Committee: Health Informatics. URL: <https://www.iso.org/standard/71088.html> (besucht am 07. 11. 2024).

Jakobsson, H. (1991). Mixed-Approach Algorithms for Transitive Closure. In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, S. 199–205. URL: <https://dl.acm.org/doi/pdf/10.1145/113413.113431> (besucht am 07. 11. 2024).

Janssens, S., Schultz, U.P. und Zaytsev, V. (2017). *Can Some Programming Languages Be Considered Harmful?* URL: <https://grammarware.net/text/2017/harmful.pdf> (besucht am 07. 11. 2024).

Koppers, T. u. a. (o. D.). *Webpack*. URL: <https://webpack.js.org/concepts/> (besucht am 07. 11. 2024).

Lawley, M. (2023). *Terminology: ConceptMap “dependsOn”, “product”, and “unmapped” Relationships*. URL: <https://chat.fhir.org/#narrow/stream/179202-terminology/topic/conceptmap.20dependson.20product.20and.20unmapped.20relationships/near/326784399> (besucht am 07. 11. 2024).

Manchikanti, L., Kaye, A.D., Singh, V. und Boswell, M.V. (2015). The Tragedy of the Implementation of ICD-10-CM as ICD-10. Is the cart before the horse or is there a tragic paradox of misinformation and ignorance? *Pain Physician* 18, E485–E495.

Medizininformatik-Initiative (o.D.). *Forschungsdatenportal für Gesundheit*. URL: <https://forschen-fuer-gesundheit.de> (besucht am 07. 11. 2024).

Nassi, I. und Shneiderman, B. (Aug. 1973). Flowchart Techniques for Structured Programming. *ACM SIGPLAN Notices* 8, 12–26. doi: 10.1145/953349.953350. URL: https://www.researchgate.net/publication/234805404_Flowchart_techniques_for_structured_programming (besucht am 07. 11. 2024).

NIH (o.D.). *SNOMED CT to ICD-10-CM Map. I-MAGIC – Interactive Map-Assisted Generation of ICD Codes*. URL: https://www.nlm.nih.gov/research/umls/mapping_projects/snomedct_to_icd10cm.html (besucht am 07. 11. 2024).

Ohlsen, T., Kruse, V., Krupar, R., Banach, A., Ingenerf, J. und Drenkhahn, C. (2022). Mapping of ICD-O Tuples to OncoTree Codes Using SNOMED CT Post-Coordination. Bd. 294. ISBN: 9781643682846. doi: 10.3233/SHTI220464. URL: <https://pubmed.ncbi.nlm.nih.gov/35612082/> (besucht am 07. 11. 2024).

Oracle (o.D.[a]). *MySQL. Optimizing INSERT Statements*. URL: <https://dev.mysql.com/doc/refman/en/insert-optimization.html> (besucht am 07. 11. 2024).

Oracle (o.D.[b]). *MySQL. The world's most popular open source database*. URL: <https://www.mysql.com/> (besucht am 07. 11. 2024).

Otto, M., Thornton, J. u. a. (o.D.). *Bootstrap*. URL: <https://getbootstrap.com/docs/5.3/getting-started/introduction/> (besucht am 07. 11. 2024).

Philipp, P., Oliveira, J. Veloso de, Appenzeller, A., Hartz, T. und Beyerer, J. (Dez. 2022). Evaluation of an Automated Mapping from ICD-10 to SNOMED CT. In 2022 International Conference on Computational Science and Computational Intelligence, S. 1604–1609. doi: 10.1109/CSCI58124.2022.00287.

PlanetScale (2023). *Operating without foreign key constraints*. URL: <https://planetscale.com/docs/learn/operating-without-foreign-key-constraints> (besucht am 07. 11. 2024).

Popper JS (o.D.). *Tooltip & Popover Positioning Engine*. URL: <https://popper.js.org/docs/v2/> (besucht am 07. 11. 2024).

Poppler (o.D.). *Poppler is a PDF rendering library based on the xpdf-3.0 code base*. URL: <https://poppler.freedesktop.org/> (besucht am 12. 08. 2024).

Potencier, F. (2022). *Symfony 6: The Fast Track*. Symfony SAS. URL: <https://symfony.com/doc/6.4/the-fast-track/en/index.html> (besucht am 07. 11. 2024).

Purdom, P. (1970). A Transitive Closure Algorithm. *BIT Numerical Mathematics* 10, 76–94.

Resig, J. u. a. (2023). *jQuery. write less, do more*. URL: <https://api.jquery.com/> (besucht am 07. 11. 2024).

Saripalle, R. (2019). Representing UMLS knowledge using FHIR terminological resources. In 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, S. 1109–1112. doi: 10.1109/BIBM47256.2019.8983305.

Schlueter, I.Z. (o. D.). *npm*. URL: <https://docs.npmjs.com/> (besucht am 07. 11. 2024).

Schulz, S., Steffel, J., Polster, P., Palchuk, M.B. und Daumke, P. (2019). Aligning an Administrative Procedure Coding System with SNOMED CT. In JOWO, URL: <https://open.trinetx.com/wp-content/uploads/sites/2/2020/06/OPS-SNOMED-FINAL1.pdf> (besucht am 07. 11. 2024).

Seemann, M. und Deursen, S. van (2019). *Dependency Injection Principles, Practices, and Patterns*. Manning. ISBN: 9781638357100.

SNOMED (o. D.[a]). *SNOMED CT Browser*. URL: <https://browser.ihtsdotools.org/> (besucht am 07. 11. 2024).

SNOMED (o. D.[b]). *SNOMED International Mapping Tool*. URL: <https://prod-mapping.ihtsdotools.org/> (besucht am 07. 11. 2024).

Stenberg, D. u. a. (o. D.). *cURL*. URL: <https://curl.se/docs/> (besucht am 04. 10. 2024).

Symfony (2023a). *Symfony 6.4. Console Commands*. URL: <https://symfony.com/doc/6.4/console.html> (besucht am 07. 11. 2024).

Symfony (2023b). *Symfony 6.4. The HttpFoundation Component*. URL: https://symfony.com/doc/6.4/components/http_foundation.html (besucht am 07. 11. 2024).

Symfony (2023c). *Symfony 6.4. HTTP Client*. URL: https://symfony.com/doc/6.4/http_client.html (besucht am 07. 11. 2024).

Symfony (2023d). *Symfony 6.4. The Filesystem Component*. URL: <https://symfony.com/doc/6.4/components/filesystem.html> (besucht am 07. 11. 2024).

Symfony (2023e). *Symfony 6.4. The UID Component*. URL: <https://symfony.com/doc/6.4/components/uid.html> (besucht am 07. 11. 2024).

Symfony (2023f). *Symfony 6.4. Controller*. URL: <https://symfony.com/doc/6.4/controller.html> (besucht am 07. 11. 2024).

Symfony (2023g). *Symfony 6.4. Databases and the Doctrine ORM*. URL: <https://symfony.com/doc/6.4/doctrine.html> (besucht am 07. 11. 2024).

Symfony (2023h). *Symfony 6.4. Creating and Using Templates*. URL: <https://symfony.com/doc/6.4/templates.html> (besucht am 07. 11. 2024).

Symfony (2023i). *Symfony 6.4. The Serializer Component*. URL: <https://symfony.com/doc/6.4/components/serializer.html> (besucht am 07. 11. 2024).

Thomas, D. und Hunt, A. (2019). *The Pragmatic Programmer: Your Journey to Mastery*. Addison Wesley. ISBN: 9780135957059.

Thurin, N., Bosco, P., Blin, P., Rouyer, M., Jové, J., Lamarque, S., Lignot, S., Lassalle, R., Abouelfath, A., Bignon, E. u. a. (2021). Intra-Database Validation of Case-Identifying Algorithms using Reconstituted Electronic Health Records from Healthcare Claims Data. *BMC Medical Research Methodology* 21. doi: 10.1186/s12874-021-01285-y.

Twig (2024). *Twig 3.x. The flexible, fast, and secure template engine for PHP*. URL: <https://twig.symfony.com/doc/3.x/> (besucht am 07. 11. 2024).

Vikström, A., Skånér, Y., Strender, L.-E. und Nilsson, G. (2007). Mapping the Categories of the Swedish Primary Health Care Version of ICD-10 to SNOMED CT Concepts: Rule Development and Intercoder Reliability in a Mapping Trial. *BMC Medical Informatics and Decision Making* 7, 9. doi: 10.1186/1472-6947-7-9. URL: <https://link.springer.com/article/10.1186/1472-6947-7-9> (besucht am 07. 11. 2024).

Voorhees, D. (2020). *Guide to Efficient Software Design: An MVC Approach to Concepts, Structures, and Models*. Texts in Computer Science. Springer International Publishing. ISBN: 9783030285012.

Vrána, J. (2021). *Adminer. Database Management in a Single PHP File*. URL: <https://www.adminer.org/> (besucht am 07. 11. 2024).

W3C (o. D.). *CSS*. URL: <https://www.w3.org/TR/CSS/> (besucht am 07. 11. 2024).

W3Techs (2024). *Usage Statistics of PHP for Websites*. URL: <https://w3techs.com/technologies/details/pl-php> (besucht am 07. 11. 2024).

Weaver, R. (2023). *Swagger UI: Interactive Docs*. URL: <https://symfonycasts.com/screencast/api-platform/swagger> (besucht am 25. 10. 2024).

WHATWG (o. D.). *HTML*. URL: <https://html.spec.whatwg.org/> (besucht am 07. 11. 2024).

WHO-FIC (2021). *Classifications and Terminology Mapping – Principles and Best Practice*. World Health Organization, Group: Family of International Classifications. URL: <https://www.who.int/publications/m/item/who-fic-classifications-andterminology-mapping> (besucht am 21. 10. 2024).

Winkler, W. (1990). String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. *Proceedings of the Section on Survey Research Methods*. URL: <https://eric.ed.gov/?id=ED325505> (besucht am 07. 11. 2024).

Wu, P., Gifford, A., Meng, X., Li, X., Campbell, H., Varley, T., Zhao, J., Carroll, R., Bastarache, L., Denny, J.C. u. a. (2019). Mapping ICD-10 and ICD-10-CM Codes to Phecodes: Workflow Development and Initial Evaluation. *JMIR Med Inform* 7, e14325. doi: 10.2196/14325. URL: <http://medinform.jmir.org/2019/4/e14325/> (besucht am 07. 11. 2024).



Appendix

A.1 Abweichungen zwischen ICD-10-GM und OPS Versionen

ICD-10-GM

Version	Abweichungen zwischen den Versionen	
2025	URL	version2025-vorab/icd10gm2025syst-ueberl-vorab.zip
	Kodes	Klassifikationsdateien/icd10gm2025syst_vorab.txt
	Umsteiger	Klassifikationsdateien/ icd10gm2025syst_umsteiger_2024_2025_vorab.txt
	Sonstiges	· Vorab-Version
2024	URL	version2024/icd10gm2024syst-ueberl.zip
	Umsteiger	Klassifikationsdateien/ icd10gm2024syst_umsteiger_2023_20221206_2024.txt
2023	URL	version2023/icd10gm2023syst-ueberl_20221206.zip
	Kodes	Klassifikationsdateien/ icd10gm2023syst_20221206.txt
	Umsteiger	Klassifikationsdateien/ icd10gm2023syst_umsteiger_2022_2023_20221206.txt
2022	URL	vorgaenger/icd10gm2022.zip
	Sonstiges	· Zip-Unterdatei: icd10gm2022syst-ueberl.zip
2021	URL	vorgaenger/icd10gm2021.zip
	Verzeichnis	icd10gm2021syst-ueberl-20201111
2020	URL	vorgaenger/icd10gm2020.zip
	Verzeichnis	icd10gm2020syst-ueberl
2019	URL	vorgaenger/icd10gm2019.zip
	Verzeichnis	icd10gm2019syst-ueberl

A Appendix

2018	URL	vorgaenger/icd10gm2018.zip
	Verzeichnis	x1gut2018
2017	URL	vorgaenger/icd10gm2017.zip
	Verzeichnis	x1gut2017
2016	URL	vorgaenger/icd10gm2016.zip
	Verzeichnis	x1gut2016
2015	URL	vorgaenger/icd10gm2015.zip
	Verzeichnis	x1gut2015
2014	URL	vorgaenger/icd10gm2014.zip
	Verzeichnis	x1gua2014
2013	URL	vorgaenger/icd10gm2013.zip
	Verzeichnis	x1gua2013
2012	URL	vorgaenger/icd10gm2012.zip
	Kodes	x1ueb2011_2012/Klassifikationsdateien/ icd10gmsyst2012.txt
	Umsteiger	x1ueb2011_2012/Klassifikationsdateien/ umsteiger_icd10gmsyst2011_icd10gmsyst2012.txt
2011	URL	vorgaenger/icd10gm2011.zip
	Kodes	x1ueb2010_2011/Klassifikationsdateien/ icd10gmsyst2011.txt
	Umsteiger	x1ueb2010_2011/Klassifikationsdateien/ umsteiger_icd10gmsyst2010_icd10gmsyst2011.txt
2010	URL	vorgaenger/icd10gm2010.zip
	Kodes	x1ueb2009_2010/Klassifikationsdateien/ icd10gmsyst2010.txt
	Umsteiger	x1ueb2009_2010/Klassifikationsdateien/ umsteiger_icd10gmsyst2009_icd10gmsyst2010.txt
2009	URL	vorgaenger/icd10gm2009.zip
	Kodes	x1ueb2008_2009/Klassifikationsdateien/ icd10gmsyst2009.txt
	Umsteiger	x1ueb2008_2009/Klassifikationsdateien/ umsteiger_icd10gmsyst2008_icd10gmsyst2009.txt
2008	URL	vorgaenger/icd10gm2008.zip
	Kodes	x1ueb2007_2008/Klassifikationsdateien/ icd10v2008.txt
	Umsteiger	x1ueb2007_2008/Klassifikationsdateien/ umsteiger20072008.txt
	Sonstiges	· ISO-8859-1

A Appendix

2007	URL	vorgaenger/icd10gm2007.zip
	Kodes	x1ueb2006_2007/Klassifikationsdateien/ ICD10V2007.txt
	Umsteiger	x1ueb2006_2007/Klassifikationsdateien/ Umsteiger.txt
	Sonstiges	· ISO-8859-1
2006	URL	vorgaenger/icd10gm2006.zip
	Kodes	x1ueb2005_2006/ICD10V2006.txt
	Umsteiger	x1ueb2005_2006/umsteiger.txt
	Sonstiges	· ISO-8859-1
2005	URL	vorgaenger/icd10gm2005.zip
	Kodes	x1ueb2004_2005/ICD10V2005.txt
	Umsteiger	x1ueb2004_2005/umsteiger.txt
	Sonstiges	· ISO-8859-1
2004	URL	vorgaenger/icd10gm2004.zip
	Kodes	x1ueb20_2004/icd10v2004.txt
	Umsteiger	x1ueb20_2004/Umsteiger.txt
	Sonstiges	· ISO-8859-1 · Punkt-Strich-Notation · 6-Spalten-Umsteiger
2.0	URL	vorgaenger/icd10gm20.zip
	Kodes	x1ueb13_20_v11/icd10v20.txt
	Umsteiger	x1ueb13_20_v11/Umsteiger.txt
	Sonstiges	· ISO-8859-1 · Punkt-Strich-Notation · Kreuz-Stern-System · 6-Spalten-Umsteiger · Nicht endständige Umsteiger
1.3	Kodes	x1ueb13_20_v11/icd10v13.txt
	Sonstiges	· ISO-8859-1 · Punkt-Strich-Notation · Kreuz-Stern-System · Keine Überleitung

OPS

Version	Abweichungen zwischen den Versionen	
2025	URL	version2025-vorab/ops2025syst-ueberl-vorab.zip
	Kodes	Klassifikationsdateien/ops2025syst_vorab.txt
	Umsteiger	Klassifikationsdateien/ ops2025syst_umsteiger_2024_2025_vorab.txt
	Sonstiges	· Vorab-Version
2024	URL	version2024/ops2024syst-ueberl.zip
2023	URL	version2023/ops2023syst-ueberl.zip
2022	URL	vorgaenger/ops2022.zip
	Sonstiges	· Zip-Unterdatei: ops2022syst-ueberl.zip
2021	URL	vorgaenger/ops2021.zip
	Verzeichnis	ops2021syst-ueberl
2020	URL	vorgaenger/ops2020.zip
	Verzeichnis	ops2020syst-ueberl
2019	URL	vorgaenger/ops2019.zip
	Verzeichnis	ops2019syst-ueberl
2018	URL	vorgaenger/ops2018.zip
	Verzeichnis	p1sut2018
2017	URL	vorgaenger/ops2017.zip
	Verzeichnis	p1sut2017
2016	URL	vorgaenger/ops2016.zip
	Verzeichnis	p1sut2016
2015	URL	vorgaenger/ops2015.zip
	Verzeichnis	p1sut2015
2014	URL	vorgaenger/ops2014.zip
	Kodes	p1sua2014-20131104/Klassifikationsdateien/ ops2014syst_20131104.txt
	Umsteiger	p1sua2014-20131104/Klassifikationsdateien/ ops2014syst_umsteiger_2013_2014_20131104.txt
2013	URL	vorgaenger/ops2013.zip
	Kodes	p1sua2013/Klassifikationsdateien/ ops2013syst_20121113.txt
	Umsteiger	p1sua2013/Klassifikationsdateien/ ops2013syst_umsteiger_2012_2013_20121109.txt

A Appendix

2012	URL	vorgaenger/ops2012.zip
	Kodes	p1ueb2011_2012/Klassifikationsdateien/ opssyst2012.txt
	Umsteiger	p1ueb2011_2012/Klassifikationsdateien/ umsteiger_opssyst2011_opssyst2012.txt
2011	URL	vorgaenger/ops2011.zip
	Kodes	p1ueb2010_2011/Klassifikationsdateien/ opssyst2011.txt
	Umsteiger	p1ueb2010_2011/Klassifikationsdateien/ umsteiger_opssyst2010_opssyst2011.txt
2010	URL	vorgaenger/ops2010.zip
	Kodes	p1ueb2009_2010/Klassifikationsdateien/ opssyst2010.txt
	Umsteiger	p1ueb2009_2010/Klassifikationsdateien/ umsteiger_opssyst2009_opssyst2010.txt
2009	URL	vorgaenger/ops2009.zip
	Kodes	p1ueb2008_2009/Klassifikationsdateien/ opsamtl2009.txt
	Umsteiger	p1ueb2008_2009/Klassifikationsdateien/ umsteigeramtl20082009.txt
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)
2008	URL	vorgaenger/ops2008.zip
	Kodes	ops2008amtl/p1ueb2007_2008/ Klassifikationsdateien/opsamtl2008.txt
	Umsteiger	ops2008amtl/p1ueb2007_2008/ Klassifikationsdateien/umsteigeramtl20072008.txt
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)
2007	URL	vorgaenger/ops2007.zip
	Kodes	ops2007amtl/p1ueb2006_2007/ Klassifikationsdateien/opsamtl2007.txt
	Umsteiger	ops2007amtl/p1ueb2006_2007/ Klassifikationsdateien/UmsteigerAmtlich.txt
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)

A Appendix

2006	URL	vorgaenger/ops2006.zip
	Kodes	ops2006amtl/p1ueb2005_2006/opsv2006.txt
	Umsteiger	ops2006amtl/p1ueb2005_2006/umsteiger.txt
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)
2005	URL	vorgaenger/ops2005.zip
	Kodes	ops2005amtl/p1ueb2004_2005_v10/OPS2005.txt
	Umsteiger	ops2005amtl/p1ueb2004_2005_v10/umsteiger.txt
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 5-Spalten-Umsteiger
2004	URL	vorgaenger/ops2004.zip
	Kodes	ops2004amtl/p1ueb21_2004_v10/opsv2004.txt
	Umsteiger	ops2004amtl/p1ueb21_2004_v10/Umsteiger.txt
	Sonstiges	· ISO-8859-1 · 4-Spalten-Umsteiger
2.1	URL	vorgaenger/ops21.zip
	Kodes	ops21amtl/p1ueb20_21_v10/opsv21.txt
	Umsteiger	ops21amtl/p1ueb20_21_v10/Umsteiger.txt
	Sonstiges	· ISO-8859-1 · KOMBI-Kode · 6-Spalten-Umsteiger (ursprüngliches Format)
2.0	URL	vorgaenger/ops20.zip
	Kodes	p1ueb11_20_v11/0psv20.txt
	Umsteiger	p1ueb11_20_v11/Umsteiger.txt
	Sonstiges	· ISO-8859-1 · KOMBI-Kode · 3-Spalten-Umsteiger
1.1	Kodes	p1ueb11_20_v11/0psv11.txt
	Sonstiges	· ISO-8859-1 · Keine Überleitung

A.2 Beispielergebnis der Horizontalen Suche

Ergebnis im JSON-Format für den Aufruf:

```
searchHorizontal('icd10gm', '2014', 'M21.6')
```

Um Platz zu sparen enthält nur der erste Umsteiger die zusätzlichen Informationen, die pro Eintrag ermittelt werden, also die automatische Überleitbarkeit sowie die Titel der Kodes. Außerdem sind für die Lesbarkeit die Sonderzeichen nicht kodiert.

Die Seite zeigt die Richtung 2024←2014 und die nächste Seite 2014→2004, 2.0, 1.3.

```
{
  "fwd": {
    "year": "2014",
    "other": "2015",
    "umsteiger": [
      {
        "old": "M21.6",
        "new": "M21.60",
        "auto": "",
        "auto_r": "A",
        "old_name": "Sonstige erworbene Deformitäten des Knöchels und des Fußes",
        "new_name": "Erworbener Hohlfuß [Pes cavus]"
      },
      {
        "old": "M21.6",
        "new": "M21.61",
      },
      {
        "old": "M21.6",
        "new": "M21.62",
      },
      {
        "old": "M21.6",
        "new": "M21.63",
      },
      {
        "old": "M21.6",
        "new": "M21.68",
      }
    ]
  },
}
```

```

"rev": {
  "year": "2013",
  "other": "2012",
  "umsteiger": [
    {
      "old": "M21.60",
      "new": "M21.6",
      "recursion": {
        "year": "2.0",
        "other": "1.3",
        "umsteiger": [
          {
            "old": "M21.6",
            "new": "M21.60",
          }
        ]
      }
    }
  ],
},
{
  "old": "M21.67",
  "new": "M21.6",
  "recursion": {
    "year": "2.0",
    "other": "1.3",
    "umsteiger": [
      {
        "old": "M21.6",
        "new": "M21.67",
      }
    ]
  }
},
{
  "old": "M21.87",
  "new": "M21.6",
  "recursion": {
    "year": "2.0",
    "other": "1.3",
    "umsteiger": [
      {
        "old": "M21.8",
        "new": "M21.87",
      }
    ]
  }
}
]
}
}

```

A.3 Beispielergebnis der Vertikalen Suche / ConceptMap

FHIR ConceptMap im XML-Format für das Beispiel in Abbildung 3.6. Die komplette ConceptMap enthält noch viel mehr Gruppen und Codes, aber abgedruckt sind nur die relevanten Teile wie beschrieben in Unterabschnitt 3.2 – „Konkretes Beispiel“.

```

<?xml version="1.0"?>
<ConceptMap xmlns="http://hl7.org/fhir">
  <id value="icd10gm_from:2024_to:1.3_target:2024"/>
  <url value="urn:uuid:0192e098-0871-79d5-a4e6-039398aa7803"/>

  <group>
    <source value="2018"/>
    <target value="2024"/>
    <element>
      <code value="G83.8"/>
      <target>
        <code value="G83.6"/>
        <equivalence value="wider"/>
      </target>
      <target>
        <code value="G83.8"/>
        <equivalence value="wider"/>
      </target>
    </element>
  </group>

  <group>
    <source value="2015"/>
    <target value="2024"/>
    <element>
      <code value="G83.80"/>
      <target>
        <code value="G83.5"/>
        <equivalence value="relatedto"/>
      </target>
    </element>
    <element>
      <code value="G83.88"/>
      <target>
        <code value="G83.6"/>
        <equivalence value="wider"/>
      </target>
      <target>
        <code value="G83.8"/>
        <equivalence value="wider"/>
      </target>
    </element>
  </group>

```

```

<group>
  <source value="2004"/>
  <target value="2024"/>
  <element>
    <code value="G83.8"/>
    <target>
      <code value="G83.6"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.8"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.5"/>
      <equivalence value="wider"/>
    </target>
  </element>
</group>

<group>
  <source value="1.3"/>
  <target value="2024"/>
  <element>
    <code value="G83.8"/>
    <target>
      <code value="G83.6"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.8"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.5"/>
      <equivalence value="wider"/>
    </target>
  </element>
</group>
</ConceptMap>

```
