



UNIVERSITÄT ZU LÜBECK  
INSTITUT FÜR  
THEORETISCHE INFORMATIK

## **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 1. 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**

## **Abstract**

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Grundbegriffe	1
1.2	Verwandte Arbeiten	3
1.3	Aufbau & Beiträge dieser Arbeit	4
1.4	Quali	4
<b>2</b>	<b>BfArM-Daten</b>	<b>9</b>
2.1	Kode- und Umsteiger-Dateien	9
2.2	Datenintegrationsprozess	14
2.3	Datenvorverarbeitung	16
<b>3</b>	<b>Umsteiger-Suche</b>	<b>21</b>
3.1	Allgemeine Funktionen	21
3.2	Transitive Hülle	25
3.3	Vergleich der Suchalgorithmen	38
3.4	Schreiben der FHIR ConceptMap	38
<b>4</b>	<b>Web-Applikation</b>	<b>40</b>
4.1	Überblick	40
4.2	Aufbau des bfarmer -Projekts	42
4.3	Backend	44
4.4	Frontend	45
4.5	AJAX	45
<b>5</b>	<b>Zusammenfassung</b>	<b>47</b>
5.1	Ergebnisse	47
5.2	Zusammenfassung	48
5.3	Ausblick	48
	<b>Literatur</b>	<b>50</b>
<b>A</b>	<b>Appendix</b>	<b>54</b>
A.1	Abweichungen zwischen ICD-10-GM und OPS Versionen	54
A.2	Beispielergebnis der Horizontalen Suche	60





# 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. Diese Änderungen in den Codes erschweren die versionsübergreifende Analyse klinischer Daten, wie sie beispielsweise vom Deutschen Forschungsdatenportal für Gesundheit (Medizininformatik-Initiative, o. D.) zur Verfügung gestellt werden. Ein technischer Ansatz dieses Code-Mapping-Problem zu lösen wäre die Verwendung von *ConceptMaps* aus dem FHIR Standards für den Austausch elektronischer Gesundheitsdaten.

### 1.1 Grundbegriffe

#### Klassifikationen ICD-10-GM und OPS

(Gaus, 2005)

#### FHIR ConceptMap

«HL7® FHIR® (im Folgenden „FHIR“ genannt) ist der modernste Interoperabilitäts-Standard aus der Produktfamilie von Health Level 7 International (kurz: „HL7“), einer internationalen Standardisierungsorganisation für das Gesundheitswesen, die in der Vergangenheit schon viele erfolgreiche und weit verbreitet genutzte Standards, wie zum Beispiel HL7 Version 2 oder HL7 CDA (Clinical Document Architecture) hervorgebracht hat. [...] HL7 wurde 1987 gegründet, um Standards für klinische Informationssysteme zu erarbeiten. [...] 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.» (Heckmann, 2022)

„Health Level 7 wurde 1987 gegründet, um Standards für klinische Informationssysteme zu erarbeiten. [...] FHIR ist die dritte Generation von Interoperabilitätsstandards aus der Feder von HLR7. 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.“ (Heckmann, 2022, Seite 309)

FHIR (Braunstein, 2022) HAPI, Ontoserver

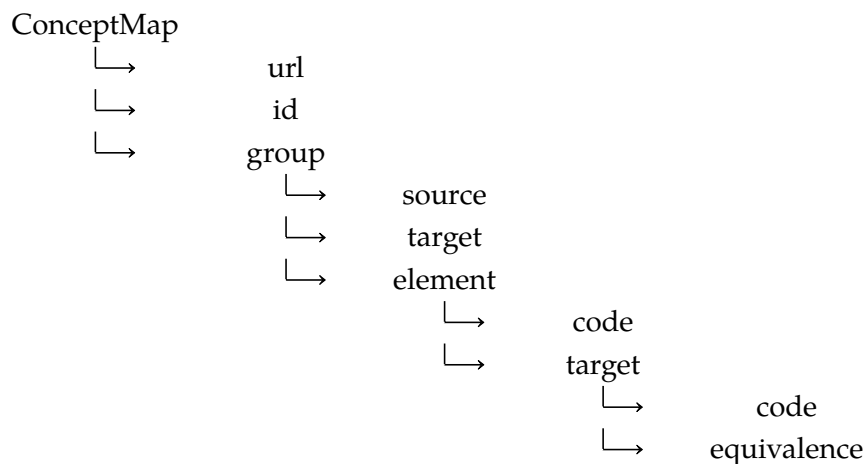
- FHIR als Baukasten
- FHIR als Community-Standard

### REST

REST (Fielding, 2000)

### ConceptMap

<http://hl7.org/fhir/R4/conceptmap.html>



- ConceptMap: A statement of relationships from one set of concepts to one or more other concepts - either concepts in code systems, or data element/data element concepts, or classes in class models.

A map from one set of concepts to one or more other concepts

- url: An absolute URI that is used to identify this concept map when it is referenced in a specification, model, design or an instance; also called its canonical identifier. This SHOULD be globally unique and SHOULD be a literal address at which an authoritative instance of this concept map is (or will be) published. This URL can be the target of a canonical reference. It SHALL remain the same when the concept map is stored on different servers.

Canonical identifier for this concept map, represented as a URI (globally unique)

§ wird von HAPI benötigt

- id: The logical id of the resource, as used in the URL for the resource. Once assigned, this value never changes.

Logical id of this artifact § kann nicht verändert werden

- group: A group of mappings that all have the same source and target system.  
Same source and target systems



- source: An absolute URI that identifies the source system where the concepts to be mapped are defined.  
Source system where concepts to be mapped are defined
- target: An absolute URI that identifies the target system that the concepts will be mapped to.  
Target system that the concepts are to be mapped to
- element: Mappings for an individual concept in the source to one or more concepts in the target.  
Mappings for a concept from the source set
- code: Identity (code or path) or the element/item being mapped.  
Identifies element being mapped
- target: A concept from the target value set that this concept maps to.  
Concept in target system for element
- code: Identity (code or path) or the element/item that the map refers to.  
Code that identifies the target element
- equivalence: The equivalence between the source and target concepts (counting for the dependencies and products). The equivalence is read from target to source (e.g. the target is 'wider' than the source).  
§ Auflisten, alle Versionen
- unmapped: What to do when there is no mapping for the source concept. "Unmapped" does not include codes that are unmatched, and the unmapped element is ignored in a code is specified to have equivalence = unmatched.  
What to do when there is no mapping for the source concept  
§ provided

## 1.2 Verwandte Arbeiten

Mappingqualität

Medicats (ICD-10-GM Versionen)

Beispiele für FHIR-Mappings

UMLS

Oncology -> SNOMED CT

SNOMED als Interlingua

## Normen für Mappings

Health Informatics, Terminology resource map quality measures (MapQual) (ISO 21564, 2019)

Health informatics – Principles of mapping between terminological resources (ISO 12300, 2014)

### Medicats

Medical Classification and Terminology Systems in a Secondary Use Context: Challenges and Perils (Hund u. a., 2016)

Medicats (Hund, o. D.)

### ConceptMap für UMLS

Building Interoperable FHIR-based Vocabulary Mapping Services: A Case Study of OH-DSI Vocabularies and Mappings

using multiple groups (Saripalle, 2019)

instead? – An Interoperable UMLS Terminology Service Using FHIR

### Interlingua SNOMED CT

SNOMED (Philipp u. a., 2022)

Probleme:

- Uneindeutig, med. Fachpersonal benötigt
- Deutsche Version SNOMED CT
- OPS
- Wenn Mapping von ICD-10-GM auf SNOMED CT existiert, kann versionsübergreifendes Mapping für ICD-10-GM immer noch hilfreich sein.

Kode	Titel
203533003	Peroneal spastic flat foot
90374001	Acquired spastic flat foot
53226007	Talipes planus
44480001	Acquired flexible flat foot
203531001	Hypermobility flat foot
203534009	Acquired pes planus
203532008	Rigid flat foot

## 1.3 Aufbau & Beiträge dieser Arbeit

## 1.4 Quali

WHO-FIC Classifications and Terminology Mapping

### Principles

1. Establish use case(s) before developing the map
2. Clearly define the purpose, scope, and directionality of the map.
3. Maps should be unidirectional and single purposed. Separate maps should be maintained for bidirectional maps (to support both a forward and a backward map table). Such unidirectional maps can be handy to support data continuity for epidemiological and longitudinal studies. Maps should not be reversed.
4. Develop clear and transparent documentation that is freely available to all and describes the purpose, scope, limitations, and methodology of the map.
5. Ideally, the producers of both terminologies in any map participate in the mapping effort to ensure that the result accurately reflects their terminologies' meaning and usage. At a minimum, both terminology producers should define the primary purpose and parameters of the mapping task, review and verify the map, develop the plan for testing and validation, and devise a cost-effective strategy for building, maintaining, and enhancing the map over time.
6. Map developers should agree on team members' competencies, knowledge, and skills at the project's onset. Ideally, users of the map also participate in its design and testing to ensure that it fits its intended purpose.
7. Quality Assurance (QA) and Usage Validation: QA and usage validation is ensuring the reproducibility, traceability, usability and comparability of the maps. Establish the QA and usage validation protocols at the beginning of the project and apply them throughout the mapping process. Factors that may be involved in quality assurance include quality-assurance rules, testing (test protocols, pilot testing), and quality metrics (such as computational metrics or precisely defined cardinality, equivalence, and conditionality). Usage validation of maps is an independent process involving users of the maps (not developer of the maps) in order to determine whether the maps are fit for purpose (e.g. do the end users reach to the correct code in the target terminology when using manual and automated maps etc.). Usage validation is essential to ensure the integrity of the information from source data to the final coding. Key principles for usage validation of maps include:
  - 7.1 use of the ground truth of the original source data1 (e.g. diagnosis as written in the medical record) as the reference point;
  - 7.2 compare the original source data with the end results of the following two processes
    - i. Coding of original source data with a source terminology – map code(s) of source terminology to code(s) of target terminology
    - ii. Coding of original source data with target terminology
  - 7.3 statistically significant sample size that is representative of the target terminology and its prototypical use case settings.

### 7.4 Usage validation of automated maps should always include human (i.e. manual) validation

Clear documentation of the QA process and validation procedures is essential in this step in the mapping process. If conducting a pilot test is feasible, it will improve the QA/validation process. Mapping is an iterative process that will improve overtime as it is used in real settings.

8. Dissemination: Upon publication and release, include information about release mechanisms, release cycle, versioning, source/target information, licence agreement requirements, and a feedback mechanism for users. Dissemination of maps should also include documentation, as stated above, describing the purpose, scope, limitations, and methodology used to create the maps.
9. Maintenance: establish an ongoing maintenance mechanism, release cycle, types and drivers of changes, and versioning of maps. The maintenance phase should include an outline of the overall lifecycle plan for the map, open transparent resolution mechanism for mapping problems, continuous improvement process, and decision process around when an update is required. Whenever maps are updated, the cycle of QA and validation must be repeated.
10. When conducting mapping manually, it is recommended to provide map specialists with the necessary tools and documentation to drive consistency when building the map. These include such items as the tooling environment (workflow details and resources related to both source and target schemes); source and target browsers, if available; technical specifications (use case, scope, definitions); editorial mapping principles or rules to ensure consistency of the maps, particularly where human judgement is required; and implementation guidance. Additionally, it is best practice to provide an environment that supports dual independent authoring of maps as this is thought to reduce bias between human map specialists. Developing a consensus management process to aid in resolving discrepancies and complex issues is also beneficial.
11. In computational mapping, it is advisable to include resources to ensure consistency when building a map using a computational approach, including a description of the tooling environment, when human intervention would occur, documentation (e.g. the rules used in computerized algorithms), and implementation guidance. It is also advisable to always compute the accuracy and error rate of the maps. It is also essential to manually verify and validate the computer-generated mapping lists. Such manual checking is necessary for the quality assurance process, as maps generated automatically will almost always contain errors. Such manually verified maps can also help train the machine- learning model when maps for different sections of terminologies are being generated sequentially.
12. Cardinality is a metric in mapping that must be clearly defined regarding what is being linked between source and target and how the cardinalities are counted. For example, SNOMED CT codes for functional impairments are semantically different from ICF codes. A 1:1 map between the two does not mean semantic equivalence. In

terms of counting, what SNOMED International considers to be a 1:1 map includes what others may consider being a 1:many map.

13. Level of equivalence, such as broader, narrower, or overlap, should be specified.
14. Maps must be machine-readable to optimize their utility<sup>2</sup>.
15. ICD-11: When creating maps using ICD-11 and other WHO systems, mapping into the Foundation Component comes first, then maps to MMS could be created through linearization aggregation<sup>2</sup>

ISO/TR 12300

Prinzipien:

1. Each map should have a (preferably single) declared purpose
2. Scenarios are developed and articulated to define the requirements for the map table
3. The map table should be in a machine processable format
4. Identify each version of each terminological resource as a version of the Map Table
5. Members of the project team should have knowledge of both of the terminological resource and experience in their practical application
6. Establish the extent to which the conventions and rules of each terminological resource will be followed
7. Custodians of terminological resources should be involved in mapping projects
8. The automated and manual methods applied should be transparent and documented
9. Every map should describe the direction of the map
10. Cardinality of each individual map should be clearly specified
11. Any loss or gain of meaning should be made explicit and risk assessed  
All maps should demonstrate the degree of equivalence
12. All mapping projects should make explicit the guidelines and heuristics applied in developing and interpreting the maps when implemented
13. Documentation supporting the map should describe the map data structures, distribution format and licensing arrangements
14. Every mapping project should have a quality assurance plan which includes testing and validation
15. Every mapping project should have a consensus management process
16. Maps should be maintained and routinely updated during their lifespan
17. Every map should have a maintenance and evaluation plan, which includes the mechanisms for version control

18. Maps should have continuing improvement processes
19. Every map should have supporting documentation to assist implementation and use
20. Map development and maintenance is best managed through a team

ISO 21564

Determinants of map quality

1. Common categorical structure (ja)
2. Shared semantic domain (ja)
3. Language and Translation (keine Übersetzung)
4. Equivalence Identification / Publication (ja)
5. Equivalence Assessment (abweichungen zw. source und target)
6. Map Set Outliers
7. Clear documentation of the purpose of the map
8. Currency of the map (zeitliche Nähe)
9. Business Arrangements
10. Methodology Documentation
11. Percentage of map validated
12. Method of validation
13. Decision making (klare Prozesse)
14. Tools used to develop or maintain the map (ja)
15. Workforce
16. Governance (Entscheidungen)
17. Map Maintenance

# 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. CSV steht für „Comma-Separated Values“ und 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 hier: (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 von ICD-10-GM, Version 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. (BfArM, o. D.[c], 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 ICD-10-GM, Version 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 nicht
  4. Wenn A: automatisch überleitbar von 2017 auf 2016, sonst nicht
- 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, Version 2024 Überleitung auf Version 2023:

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

Die zusätzlichen Spalten jeweils nach den Kodes speziell für OPS sagen aus, ob Zusatzkennzeichen notwendig sind, siehe dazu auch: (BfArM, o. D.[d], 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 Absichten hinter einem Prozess. Das heißt eine Änderung der Intention einer Software-Lösung sollte 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 funktionieren, 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: Der Prozess der Datenintegration sollte unabhängig von der Version 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 dann dem Datenintegrationsprozess in einem strukturierten Dateiformat zur Verfügung gestellt werden. [§TODO: Verweis auf konkrete Implementation](#)

### 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“ (Bonnefoy 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, gilt dieser stattdessen.

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`

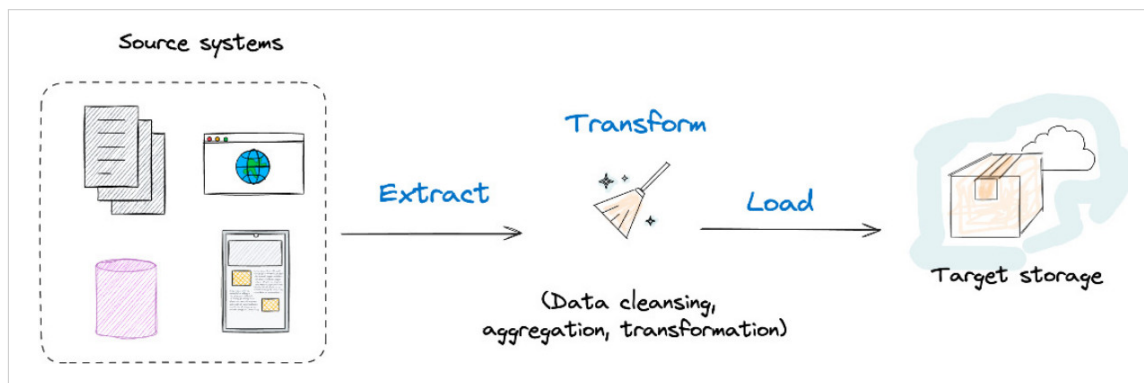
### Sonstige Abweichungen

- Vorab-Version  
Diese Version hat noch keine Seite für die Kode-Suche. [§TODO: Verweis auf Kode-Suche \(Frontend\)](#)
- 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 unter Abschnitt 3.
- Punkt-Strich-Notation, Kreuz-Stern-System  
Die Kodes älterer ICD-10-GM Versionen hatten Sonderzeichen gemäß (BfArM, o. D.[c]).
- 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 ICD-10-GM 2.0 auf 1.3. auch für nicht-endständige Kodes enthalten.
- None statt UNDEF  
Von OPS Version 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 OPS Version 2.1 enthielten zusätzliche Spalten wegen Mehrfachverschlüsselung wie die älteren ICD-10-GM Versionen.
- 3-Spalten-Umsteiger  
OPS Version 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 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)

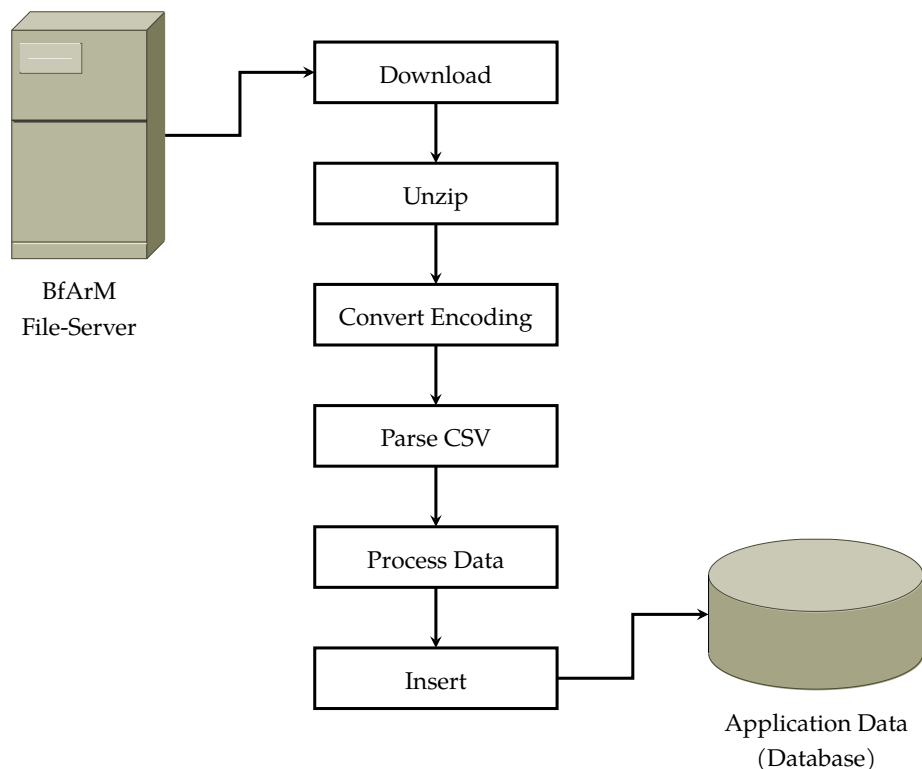
Für die BfArM-Daten sieht der Integrationsprozess konkret so aus:

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: `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 ein Nischenfall 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.

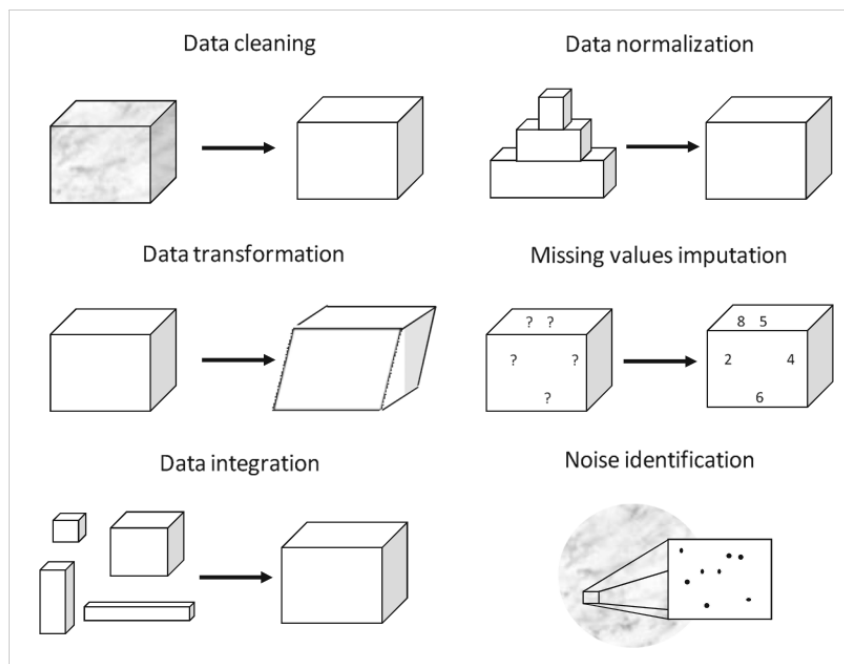


**Abbildung 2.2:** BfArM-Datenintegrationsprozess

## 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 automatische Überleitungen in den gleichen Kode. Diese müssen also gar nicht in eine Applikation aufgenommen werden, unter der Annahme, dass nicht vorhandene Umsteiger gleich automatische Überleitungen sind. In dem Fall 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 Kodes enthält. Durch das Entfernen der Sonderzeichen von Punkt-Strich-Notation und Kreuz-Stern-System gibt es außerdem doppelte Einträge in der er-



sten Spalte, das heißt bei den Kodes der Vorgänger-Version. Folgende Funktion<sup>1</sup> entfernt die überflüssigen Einträge.

Angenommen die Kodes 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.

Anmerkungen:

- 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.

<sup>1</sup>Der Pseudocode ist beschrieben in Struktogrammen nach (Nassi und Shneiderman, 1973). Die später vorkommenden Algorithmen zur Umsteiger-Suche enthalten viele Variablenzuweisungen, die abhängig von einer Bedingung sind und deren parallele Darstellung in Nassi-Shneidermann-Diagrammen übersichtlicher ist. Sie werden also im Sinne der Einheitlichkeit ebenfalls für den Algorithmus in diesem Abschnitt 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	
re-index \$data	

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 funktioniert der oben erklärte Algorithmus ebenfalls.

### Integration zusätzlicher Informationen

Im nächsten Abschnitt wird 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

Dieser Abschnitt behandelt die versionsübergreifende Suche nach Umsteigern. Es geht darum zu bestimmen, ob und wie ein bestimmter Code in Betrachtung auf alle Versionen des Kodiersystems übergeleitet wurde. Die hier aufgeführten Algorithmen sind unabhängig von einer konkreten Implementation verfasst.

§TODO 1: Umsteiger nur Veränderungen

§TODO 2: Datenstrukturen

### 3.1 Allgemeine Funktionen

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

#### 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 Kode-Wert auch alle Daten für die angegebenen Parameter gelesen.

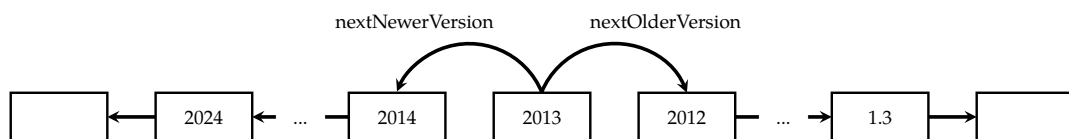
- \$type  
Die Art an Daten, die gelesen werden sollen. Für die Suche der Umsteiger werden folgende benötigt – angegeben ebenfalls wieder als mögliche Konstanten:
  - 'kodes'  
Kodes-Daten: Kode und Titel.
  - 'umsteiger'  
Umsteiger-Daten: alter Kode und neuer Kode, sowie Information über die automatische 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“.

#### Nächste Version

Außerdem wird eine Funktion benötigt, die ausgehend von System und Version die jeweils nächstältere und -neuere Version ermittelt. Wenn diese nicht existiert, bleibt der Wert leer.

Hierfür kann die in [§TODO: Verweis im Kommentar](#) erwähnte strukturierte Datei dienen, welche die Versionen und Abweichungen definiert.

#### nextNewerVersion / nextOlderVersion



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

#### Beispieldaten mit graphischer Darstellung

Um beispielsweise die Kodes M21.4 und M21.6 aus der ICD-10-GM Version 2016 abzufragen, sehen die Funktionsaufrufe dann so aus:

```
readData ('icd10gm', '2014', 'M21.4', 'kodes')
readData ('icd10gm', '2014', 'M21.6', 'kodes')
```

Und das Ergebnis als Tabelle:

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

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

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')
```

Das Ergebnis sieht wie folgt aus; „Auto“ steht für die automatische Überleitbarkeit in die jeweilige Richtung, das heißt 2014←2013 und 2014→2013.

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

Von Version 2014 auf 2013 ändern sich die angegebenen ICD-10-GM Codes nicht. Wie bereits in der Datenintegration erwähnt, müssen diese Einträge 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 nur die Veränderungen relevant. Anstatt alle nicht relevanten Umsteiger-Einträge abzuspeichern und bei jeder Abfrage auszuschließen, kann alternativ auch angenommen werden, weil eine Suche nach Umsteigern immer von einem vorhanden Code ausgeht, dass bei Nichtvorhandensein eines Umsteigers zwischen zwei Versionen dieser Code einfach gleich bleibt.

Insgesamt treten bei M21.4 über alle Versionen keine Veränderungen in den Umsteigern auf. M21.6 hat allerdings folgende:

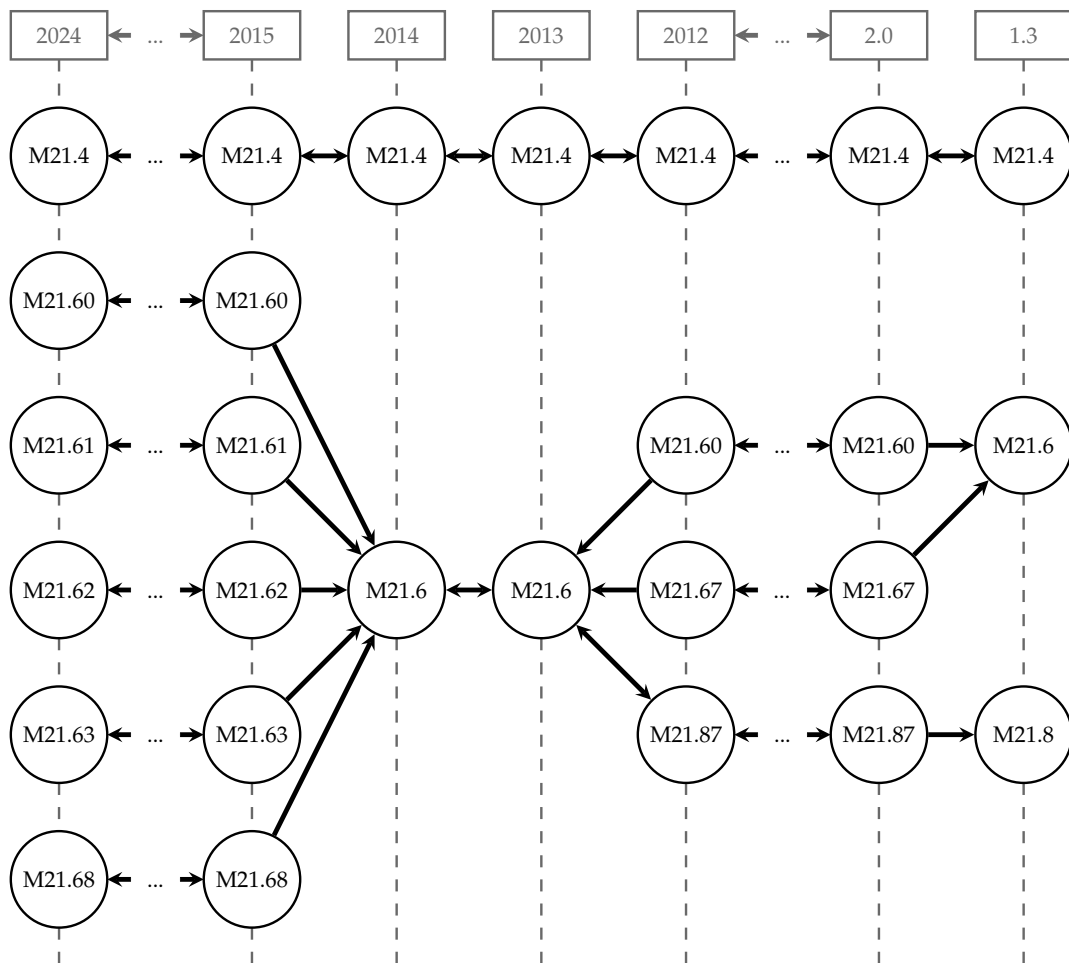
new (Kode Version 2015)	old (Kode Version 2014)	←Auto→	
M21.4	M21.4	A	A
M21.60	M21.6		A
M21.61	M21.6		A
M21.62	M21.6		A
M21.63	M21.6		A
M21.68	M21.6		A

### 3 Umsteiger-Suche

new (Kode Version 2013)	old (Kode Version 2012)	← Auto →	
M21.4	M21.4	A	A
M21.6	M21.60	A	
M21.6	M21.67	A	
M21.6	M21.87	A	A
new (Kode Version 2.0)	old (Kode Version 1.3)	← Auto →	
M21.4	M21.4	A	A
M21.60	M21.6		A
M21.67	M21.6		A
M21.87	M21.8		A

Anmerkung: Von Kode M21.8, Version 1.3. ausgehend gibt es noch wesentlich mehr Umsteiger, aber diese werden hier nicht gelistet, weil sie sind von M21.6, Version 2014 aus nicht erreichbar sind.

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



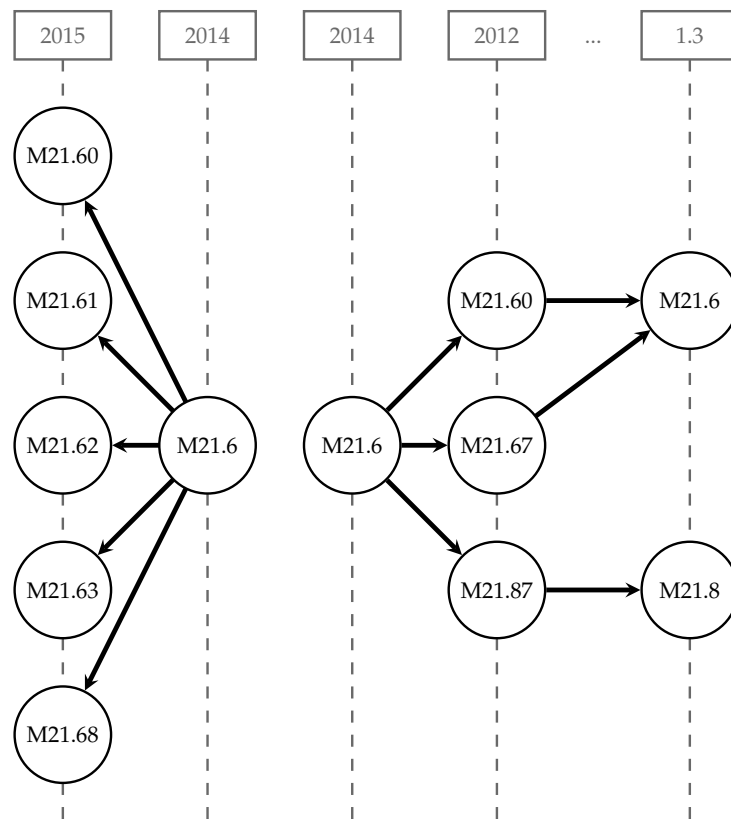
**Abbildung 3.2:** Graphische Repräsentation der Umsteiger ausgehend von den Codes M21.4 und M21.6 der ICD-10-GM Version 2014.

## 3.2 Transitive Hülle

Wenn statt der graphischen Repräsentation oben:

1. Die nicht relevanten Umsteiger ausgeschlossen werden; also wie besprochen die Umsteiger-Einträge ohne Veränderung von Codes.
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. Die Suche nach über Umsteiger erreichbaren Codes entspricht daher der Bestimmung der transitiven Hülle in der Graphentheorie.



**Abbildung 3.3:** Von M21.6, Version 2014 erreichbare Codes als gerichteter Graph.

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.

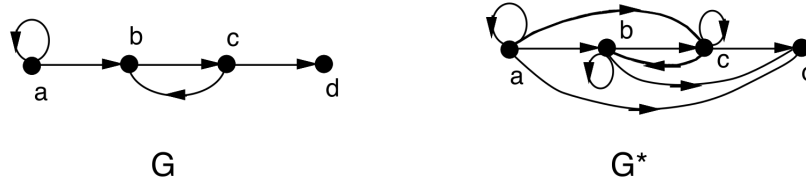
In (Gross, Yellen und Zhang, 2013, Seite 172) ist hierzu folgendes Beispiel dargestellt:



**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 Kodes– und die *vertikale* Suche primär über die Kodes –es werden zuerst alle Kodes 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 vorherigen Abschnitt für den Code M24.6 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.

Ergebnisse für andere Knoten, beziehungsweise Kodes, 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 in Funktion 3.1 readData.

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.

<code>\$data[ 'fwd' ] = searchHorizontalRecursion (\$system, \$version, \$code, true)</code>
<code>\$data[ 'rev' ] = searchHorizontalRecursion (\$system, \$version, \$code, false)</code>
<code>RETURN \$data</code>

Ausgehend von einem gegebenen Kode einer Version eines Kodiersystems 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 der Listen auch die Versionen vorher und nachher. 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 dem einem Key recursion abgelegt und die Suche fortgesetzt. Das Beispielergebnis für M21.4 befindet sich im Anhang A.2.
- \$otherVersion  
Die Version, zu der die Umsteiger ermittelt werden.
- \$otherCode  
Die Kode, zu dem 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/-nachher gespeichert. Dann endet auch hier die Rekursion.

Appendix 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

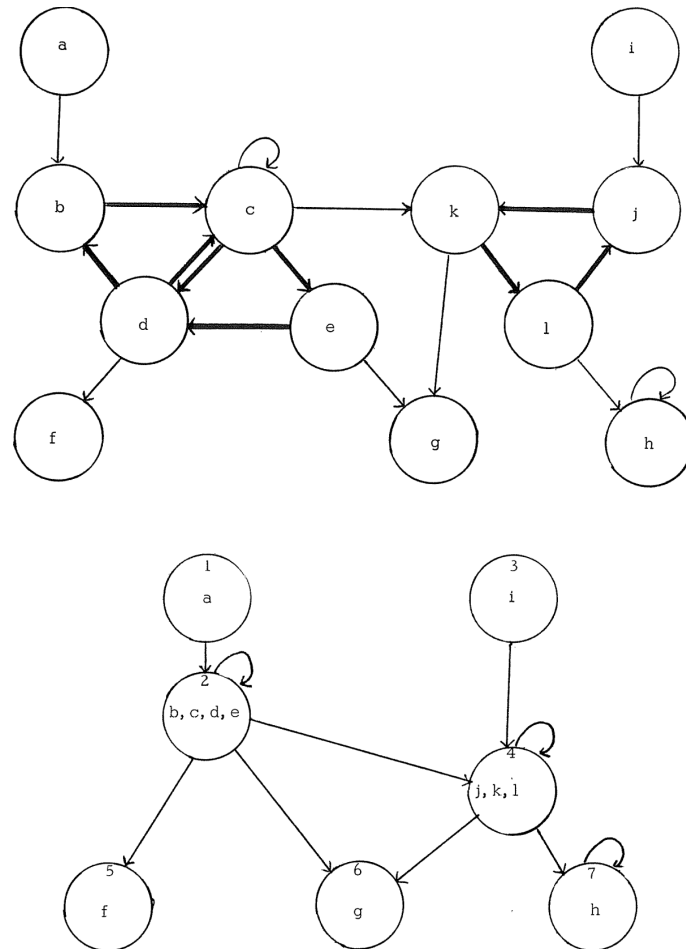
Wie bereits erwähnt startet der Algorithmus der horizontalen Suche für jeden Kode bei Null. Gerade wenn die Umsteiger für alle Kodes einer Version – beziehungsweise eines Kodiersystems– gefunden werden sollen, ist diese Vorgehensweise nicht effizient.

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

1. Ausgehend von einem Graphen  $G$ : Bestimme die stark zusammenhängenden Komponenten (SCC) in  $G$  und vereinige diese in jeweils einen einzelnen Knoten. Das ergibt einen zyklensfreien, verdichteten Graphen  $G_c$ .
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.

Zum Beispiel ist die topologische Sortierung aus Schritt zwei in der Anwendung der Suche nach Umsteigern 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.

§ Gleiche Codes möglich

Aber die vertikale Suche verwendet wesentlich 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:

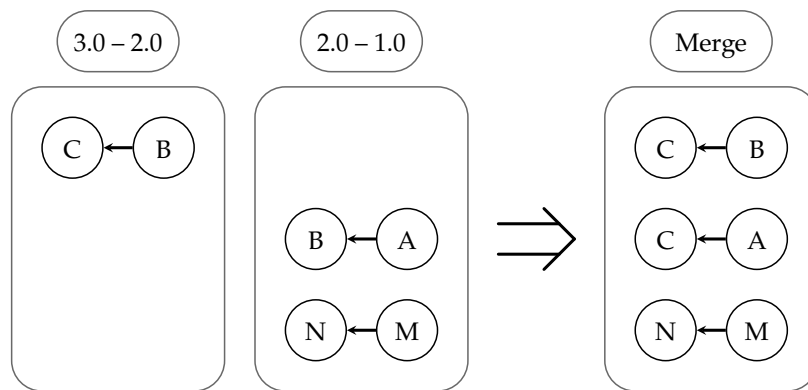
1. Topologisch rückwärts gerichtete Vorgehensweise: So ist zum Beispiel bei der Suche nach den Umsteigern eines ICD-GM-10 Kode der Version 1.3 die Reihenfolge, in der die anderen Versionen abgearbeitet werden: 2024, 2023, 2022 und so weiter. Oder anders ausgedrückt: Wenn die Überleitungen in chronologischer 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 Kode 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.

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 Kode 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.

Die gesamte Algorithmus für die vertikale Suche als Pseudocode:

#### searchVertical

Funktionsparameter:

- \$system  
Das Kodiersystem, beispielsweise als Konstanten definiert 'icd10gm' und 'ops'.
- \$targetVersion  
Die Zielversion, auf die von allen anderen Version die Umsteiger über alle Codes ermittelt werden.
- \$function  
Eine optionale, anonyme Funktion.

Lokale Variable:

- \$data      Rückgabewert, initial: leer.  
Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn \$function nicht gesetzt ist. Die erste Dimension hat als Keys alle Versionen außer der Zielversion. Die zweite Dimension hat als Keys die Codes der Version und als Value die Liste der zugehörigen Umsteiger, inklusive 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:

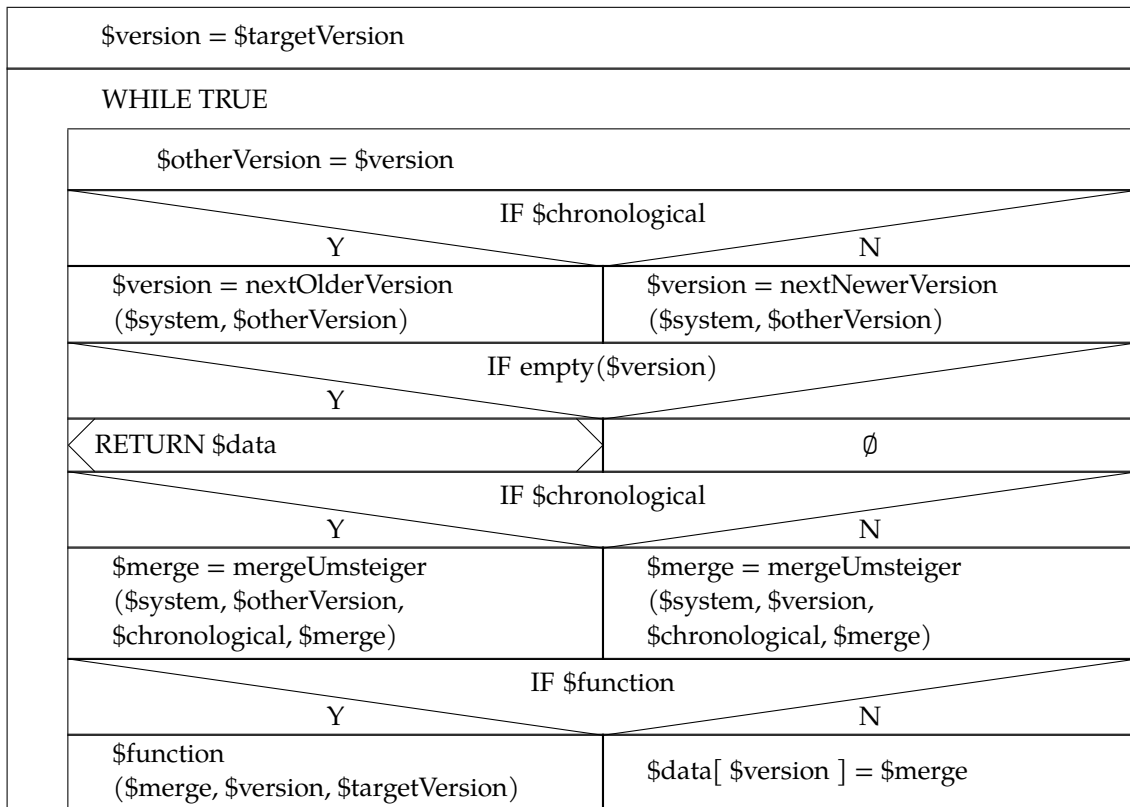
- `$system, $targetVersion, $function`  
Wie `searchVertical`.
- `$chronological`  
Bestimmt die Richtung, in der die Versionen abgearbeitet werden. TRUE: chronologisch vorwärts und FALSE: rückwärts. Die Reihenfolge ist wie erwähnt topologisch rückwärts, das heißt ausgehend von der Zielversion.

Lokale Variablen:

- `$data`            Rückgabewert, initial: leer.  
Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn `$function` nicht gesetzt ist! Die erste Dimension hat als Keys alle Versionen außer der Zielversion. [§TODO: Rest streichen?](#) Die zweite Dimension hat als Keys die Codes aller bisher verarbeiteten Umsteiger und als Values die Listen der Codes, in die übergeleitet wird, inklusive Vereinigungen wie oben beschrieben. Das heißt im Gegensatz zur horizontalen Suche werden die Umsteiger als Key-Value Paare gespeichert: Kode der aktuellen Version  $\Rightarrow$  Kode/s in die Vergleichsversion. Codes ohne Umsteiger werden nicht aufgenommen.
- `$merge`  
Wie Abbildung 3.5 [§TODO: Bei klarem Kopf data und merge neu schreiben.](#)
- `$version, $otherVersion`  
Die aktuell zu verarbeitende Version und die nächstältere beziehungsweise -neuere 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 wie oben beschrieben befüllt. Abhängig davon, ob die anonyme Funktion \$function definiert ist, wird diese entweder ausgeführt [§TODO: Verweis auf ConceptMap-Schreiben?](#) oder die Variable \$data für die Rückgabe befüllt.

#### mergeUmsteiger

Funktionsparameter:

- \$system, \$version, \$chronological  
Wie ?
- \$merge  
Die vereinigten Umsteiger der bisher durchlaufenen Versionen.

[§TODO: ab hier weitermachen](#)

Lokale Variable:

- \$data      Rückgabewert, initial: leer.  
Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn \$function nicht gesetzt ist. Die erste Dimension hat als Keys alle Versionen außer der Zielversion. Die zweite Dimension hat als Keys die Codes der Version und als Value die Liste der zugehörigen Umsteiger, inklusive Vereinigung. Codes ohne Umsteiger werden nicht aufgenommen.

### 3 Umsteiger-Suche

- \$version  
bla, initial = target.

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

Q

#### 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 Kodes der anderen Versionen. Bei der vertikalen Suche werden immer alle Kodes pro Version abgefragt.

### 3 Umsteiger-Suche

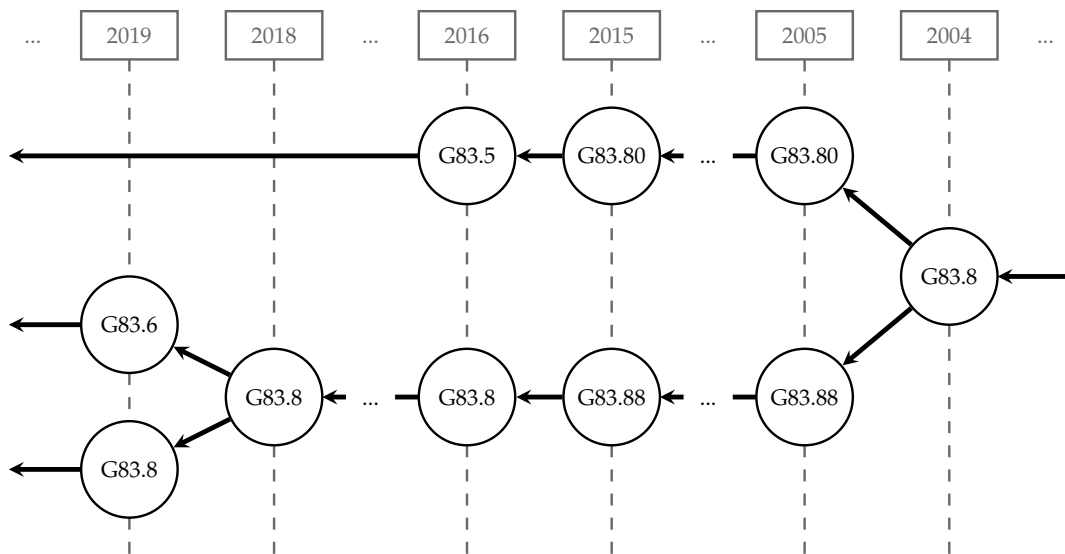


Abbildung 3.6: caption

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 schrittweise so aufgebaut:

```
'2018' => [ 'G83.8' => [ 'G83.6', 'G83.8' ] ]
'2015' => [ 'G83.8' => [ 'G83.6', 'G83.8' ],
            'G83.80' => [ 'G83.5' ],
            'G83.88' => [ '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  $G83.8$  schon gibt.

```
'2004' => [ 'G83.8' => [ 'G83.5', 'G83.6', 'G83.8' ],
            'G83.80' => [ 'G83.5' ],
            'G83.88' => [ '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

Wenn die Daten aus einer Datenbank gelesen werden, dann ist der Menge pro readData Aufruf weniger entscheidend als die Anzahl solcher Aufrufe. Das heißt alle Umsteiger einer Version auf einmal zu lesen ist wesentlich schneller als sie einzeln zu lesen.

Außerdem redundant

In dem Fall ist der

Horizontal:

Schneller bei einem Kode

Mehr Daten hinzufügar. Vertikale Suche mit Titeln würde schnell mehrere hundert MB groß werden, Arbeitsspeicher.

Vertikal:

Schneller bei allen Kodes

Verwendung für ConceptMap

Verwendung für Bestimmen ob ein Kodes Umsteiger haben für alle Kodes und Versionen

### 3.4 Schreiben der FHIR ConceptMap

When to stream? (Braaksma, 2014)

Verweis auf Intro.

Die anonyme Funktion, die in der vertikalen Suche übergeben werden kann, dient zum Schreiben der FHIR 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. Konkret erhält die Funktion (wie in §Verweis) . Zusätzlich wird allerdings noch die Information benötigt, welche Kodes insgesamt in der Startversion vorhanden sind, weil wie in (§Verweis) beschrieben die Liste der vereinigten Umsteiger immer auch Einträge aus vorher verarbeiteten Versionen enthalten durch die rückwärts topologische Vorgehensweise bedingt. Also zum Beispiel Kodes die in neueren Versionen erst hinzukommen. Außerdem wird über die beiden Informationen Umsteiger über die Versionen zwischen Start und Ziel sowie Kodes der Startversion die relationship bestimmt.

Kode in Startversion; Kode in Vereinigte Umsteiger; relationship vorhanden; Liste an Umsteigern; wider vorhanden; ein Umsteiger; related-to vorhanden; => UNDEF; unmapped vorhanden; nicht vorhanden; equivalent

Ablauf

1. Anfang der ConceptMap schreiben (Parent Element, id, url)
2. Vertikale Suche mit Funktion starten; Funktion schreibt jeweils eine Gruppe

#### 3. Ende der ConceptMap schreiben

Außerdem erlaubt die Vorgehensweise mit der anonymen Funktion ein Streaming der Datei. Das heißt muss nicht erst komplett im Arbeitsspeicher erstellt werden. Selbst wenn Umsteiger von allen Versionen als Start auf alle Versionen als Ziel geschrieben werden soll, werden als Daten im Arbeitsspeicher nur maximal die Sammlung der vereinigten Umsteiger von Start zu Ziel (oder andersherum)

Appendix A.3 in XML.

# 4

## Web-Applikation

### 4.1 Überblick

asada bfarmer asd

Version Control: thomas2019pragmatic (data1)

- Daten Integration per Kommandozeilenbefehl
- Darstellung der Daten
- Generieren von ConceptMaps
- Beispiel-Anwendung: Virtuelle Patienten

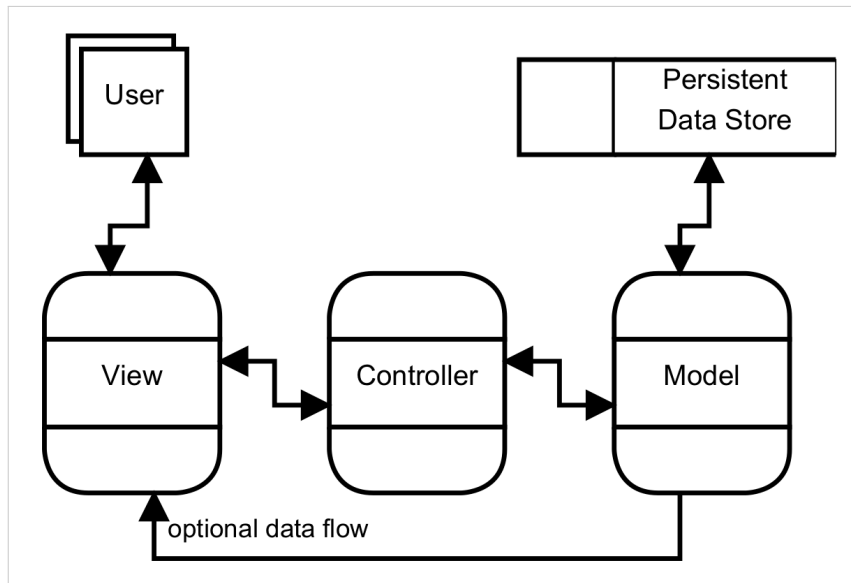
Symfony – Warum? (Potencier, 2022)

Composer (Adermann, Boggiano u. a., o. D.)

PHP Verwendung (W3Techs, o. D.)

Programming Languages Harmful (Janssens, Schultz und Zaytsev, 2017)

Model-View-Controller (Voorhees, 2020, Seite 176f)



**Abbildung 4.1:** (Voorhees, 2020, Seite 177)

Abstraction and Decoupling – Orthogonality: thomas2019pragmatic

Symfony (Potencier, 2022)

Dependency Injection (Seemann und Deursen, 2019)

### Symfony Komponenten

Console (Symfony, o. D.[a], Console Commands)

Http Foundation: Request, Response, StreamedResponse (Symfony, o. D.[b], The HttpFoundation Component)

HttpClientInterface – Download (Symfony, o. D.[c], HTTP Client)

File System (Symfony, o. D.[d], The Filesystem Component)

UUID (Symfony, o. D.[e], The UID Component)

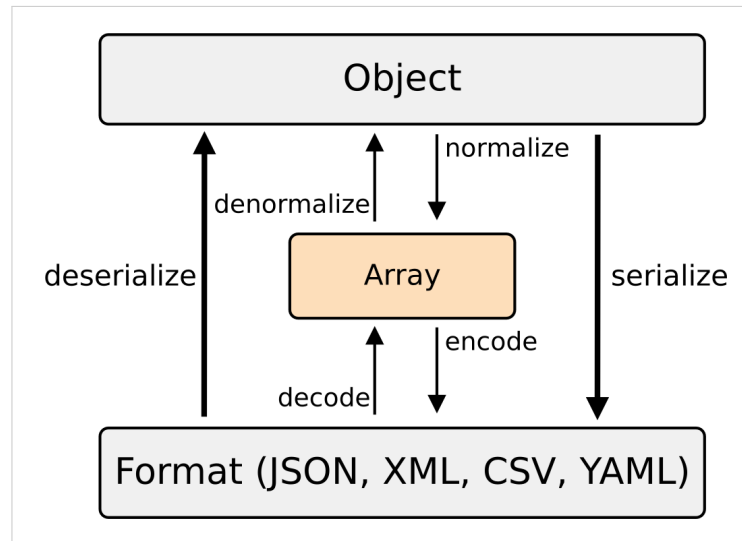
Controller – Routing (Symfony, o. D.[f], Controller)

Serializer: XML, CSV, JSON, SQL-Insert (Symfony, o. D.[g], The Serializer Component)

Database (Symfony, o. D.[h], Databases and the Doctrine ORM)

Templates (Symfony, o. D.[i], Creating and Using Templates)

Twig (Twig, o. D.)



## 4.2 Aufbau des bfarmer -Projekts

### Views

Templates für HTML-Code.

- base  
Grundvorlage für alle Seiten, also zum Beispiel <head> mit Imports für Stylesheets, Javascript und <body>.
- index  
b
- codes  
c
- conceptmap  
d
- modal  
e
- patients  
f
- search\_code  
g
- umsteiger  
h
- umsteiger\_icons  
i



- umsteiger\_search  
j
- umsteiger\_search\_recursion  
k
- umsteiger\_search\_result  
l

### Utility

- Constants  
Enthält globale Konstanten und statische Funktionen, die keine Instanziierung einer Klasse benötigen, um zum Beispiel eindeutige Namen für die SQL-Tabellen zu generieren.
- SqlInsertEncoder  
b
- TwigExtension  
Erweiterung der Twig-Template Engine, um externe Kode-Links und

### Repositories

- BfarmRepository  
a
- ConfigRepository  
b
- DatabaseRepository  
c
- PatientsRepository  
d

### Controllers

- AdminController  
a
- CodesController  
b
- ConceptMapController  
c
- DimdiController  
d

- IndexController  
e
- PatientsController  
f
- UmsteigerController  
g
- UmsteigerSearchController  
h

### Services

- ClientService  
a
- ConceptMapService  
b
- DataService  
c
- PatientsService  
d
- SetupService  
e
- UmsteigerService  
f

### Commands

- BfarmerCommand  
a
- TestCommand  
b

## 4.3 Backend

### Setup-XML

### Konsole

### Datenbank / MySQL / Doctrine

Database must have collation utf8mb4\_unicode\_ci

## 4.4 Frontend

HTML (WHATWG, o. D.), CSS (W3C, o. D.), JavaScript (Ecma International, o. D.)

### Webpack / Node.js

Webpack (Koppers u. a., o. D.), NPM (Schlueter, o. D.), Node.js (Dahl, o. D.)

webpack is a static module bundler for modern JavaScript applications

### Bootstrap

Bootstrap

Layout, Nav-Tabs, Modal, Icons

### SCSS

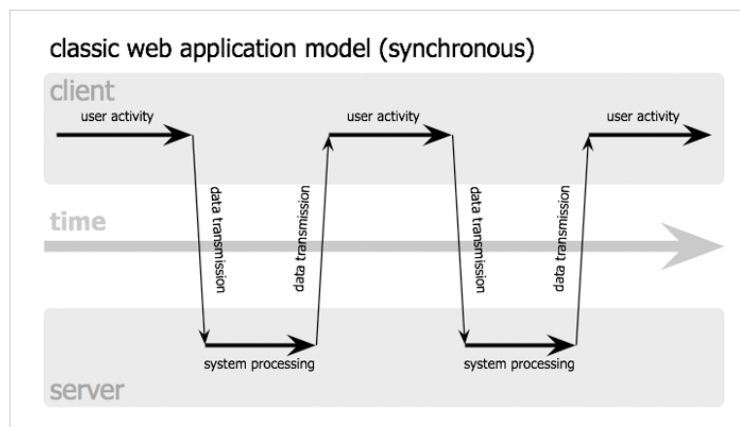
SASS (Catlin, Weizenbaum und Eppstein, o. D.): Einfachere Syntax, Variablen, Funktionen

### Tooltips

Popper (Popper JS, o. D.)

## 4.5 AJAX

AJAX (Garrett, 2005)



## 4 Web-Applikation

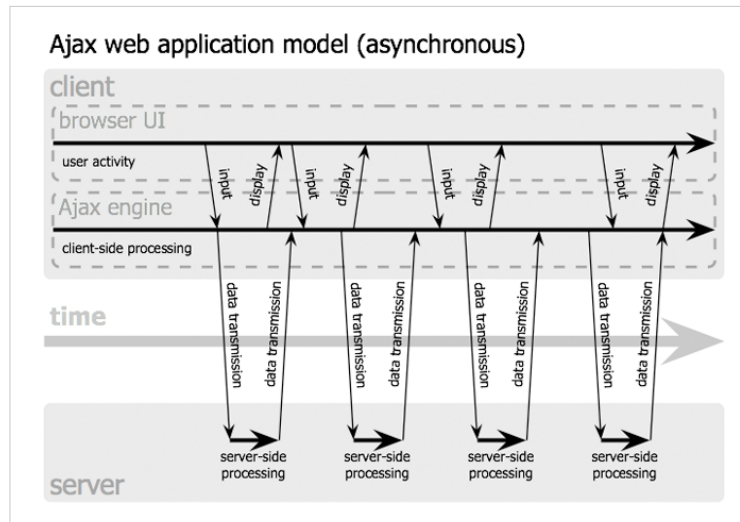


Abbildung 4.2: (Garrett, 2005)

Modal

BfArM/Dimdi Links – Neuer Tab muss geöffnet werden bevor Skripte ausgeführt werden.

CSS Flexbox (Coyier, o. D.)

### Anonyme Funktionen

Callback

Zum Schreiben der ConceptMap

Data-Preprocessing

InsertEncoder

<https://dev.mysql.com/doc/refman/en/insert-optimization.html>

# 5

## Zusammenfassung

### 5.1 Ergebnisse

#### Gegenseitige Validierung / Intra-DB Validierung

(Thurin u. a., 2021)

#### Patienten-DB

bfarmer

Test-Datenbank ICD-10-GM

Patienten insgesamt: 20000

Code:  Titel: Lähmungssyndrome  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, Epicondylus, 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: Patienten

## 5 Zusammenfassung

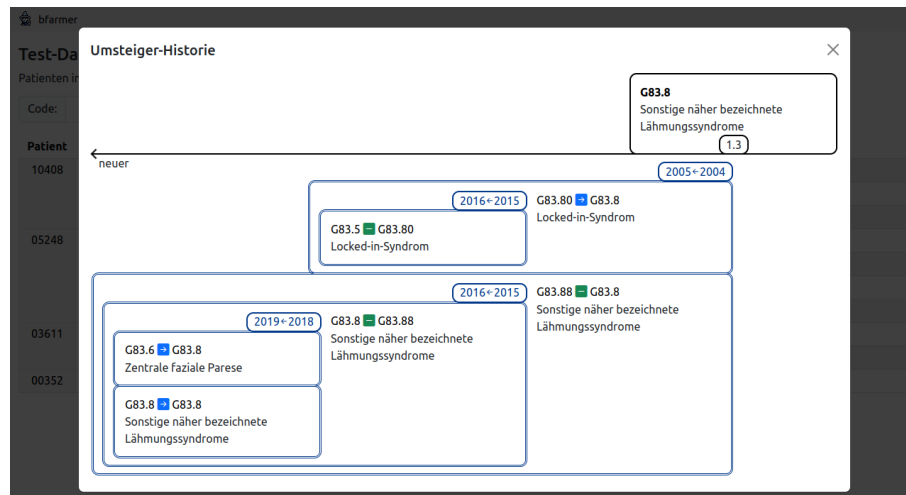


Abbildung 5.2: Umsteiger

Integration:

1. Bootstrap
2. CSS-Datei
3. On-Click Event
4. Ajax-Funktion

Mapping Quality

## 5.2 Zusammenfassung

## 5.3 Ausblick

RESTful (nach außen und innen / mehr AJAX)

ATC

ATC (BfArM, o. D.[a], ATC-Klassifikation)

PDF (Poppler, o. D.)

Sessions

Caching

---

```
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'
```

---

# Literatur

Adermann, N., Boggiano, J. u. a. (o. D.). *Composer. A Dependency Manager for PHP*. URL: <https://getcomposer.org/doc/> (besucht am 12. 08. 2024).

BfArM (o. D.[a]). *BfArM – ATC. ATC-Klassifikation*. URL: [https://www.bfarm.de/DE/Kodiersysteme/Klassifikationen/ATC/\\_node.html](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](https://www.bfarm.de/DE/Kodiersysteme/Services/Downloads/_node.html) (besucht am 09. 09. 2024).

BfArM (o. D.[c]). *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 09. 09. 2024).

BfArM (o. D.[d]). *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 09. 09. 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 22. 09. 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 12. 08. 2024).

Coyier, C. (o. D.). *CSS Flexbox Layout Guide*. URL: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> (besucht am 09. 09. 2024).

Dahl, R. (o. D.). *Node.js*. URL: <https://nodejs.org/docs/latest/api/> (besucht am 12. 08. 2024).



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

Ecma International (o. D.). *ECMAScript. JavaScript*. URL: <https://ecma-international.org/publications-and-standards/standards/ecma-262/> (besucht am 12.08.2024).

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](https://www.researchgate.net/publication/359509972_UTF-8_Latex_Encodings_of_ISO-8859_Latin-1_Character_Set/link/6241ab3b5e2f8c7a03452ac9/download) (besucht am 06.09.2024).

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 04.10.2024).

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>.

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 14.10.2024).

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

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.

Hund, H. (o. D.). *medicats. Medical classification and terminology systems library for Java 1.8*. URL: <https://github.com/hhund/medicats> (besucht am 15.05.2024).

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 12300 (2014). *Health Informatics – Principles of Mapping between Terminological Systems*. International Organization for Standardization. URL: <https://www.iso.org/standard/51344.html> (besucht am 21.10.2024).

ISO 21564 (2019). *Health Informatics – Terminology Resource Map Quality Measures (Map-Qual)*. International Organization for Standardization. URL: <https://www.iso.org/standard/71088.html> (besucht am 21.10.2024).

- 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>.
- 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 10. 10. 2024).
- Koppers, T. u. a. (o. D.). *Webpack*. URL: <https://webpack.js.org/concepts/> (besucht am 12. 08. 2024).
- Medizininformatik-Initiative (o. D.). *Forschungsdatenportal für Gesundheit*. URL: <https://forschen-fuer-gesundheit.de> (besucht am 02. 07. 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](https://www.researchgate.net/publication/234805404_Flowchart_techniques_for_structured_programming).
- 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.
- Popper JS (o. D.). *Tooltip & Popover Positioning Engine*. URL: <https://popper.js.org/docs/v2/> (besucht am 12. 08. 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 22. 09. 2024).
- Purdom, P. (1970). A Transitive Closure Algorithm. BIT Numerical Mathematics 10, 76–94.
- 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 12. 08. 2024).
- Seemann, M. und Deursen, S. van (2019). Dependency Injection Principles, Practices, and Patterns. Manning. ISBN: 9781638357100.
- Symfony (o. D.[a]). *Symfony 6.4. Console Commands*. URL: <https://symfony.com/doc/6.4/console.html> (besucht am 12. 08. 2024).

Symfony (o. D.[b]). *Symfony 6.4. The HttpFoundation Component*. URL: [https://symfony.com/doc/6.4/components/http\\_foundation.html](https://symfony.com/doc/6.4/components/http_foundation.html) (besucht am 12. 08. 2024).

Symfony (o. D.[c]). *Symfony 6.4. HTTP Client*. URL: [https://symfony.com/doc/6.4/http\\_client.html](https://symfony.com/doc/6.4/http_client.html) (besucht am 12. 08. 2024).

Symfony (o. D.[d]). *Symfony 6.4. The Filesystem Component*. URL: <https://symfony.com/doc/6.4/components/filesystem.html> (besucht am 12. 08. 2024).

Symfony (o. D.[e]). *Symfony 6.4. The UID Component*. URL: <https://symfony.com/doc/6.4/components/uid.html> (besucht am 12. 08. 2024).

Symfony (o. D.[f]). *Symfony 6.4. Controller*. URL: <https://symfony.com/doc/6.4/controller.html> (besucht am 12. 08. 2024).

Symfony (o. D.[g]). *Symfony 6.4. The Serializer Component*. URL: <https://symfony.com/doc/6.4/components/serializer.html> (besucht am 12. 08. 2024).

Symfony (o. D.[h]). *Symfony 6.4. Databases and the Doctrine ORM*. URL: <https://symfony.com/doc/6.4/doctrine.html> (besucht am 12. 08. 2024).

Symfony (o. D.[i]). *Symfony 6.4. Creating and Using Templates*. URL: <https://symfony.com/doc/6.4/templates.html> (besucht am 12. 08. 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 (o. D.). *Twig 3.x. The flexible, fast, and secure template engine for PHP*. URL: <https://twig.symfony.com/doc/3.x/> (besucht am 12. 08. 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.

W3C (o. D.). *CSS*. URL: <https://www.w3.org/TR/CSS/> (besucht am 12. 08. 2024).

W3Techs (o. D.). *Usage Statistics of PHP for Websites*. URL: <https://w3techs.com/technologies/details/pl-php> (besucht am 25. 10. 2024).

WHATWG (o. D.). *HTML*. URL: <https://html.spec.whatwg.org/> (besucht am 12. 08. 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 Beispiel 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>

```

---