TU WIEN Informatics

Integration von Case-Based Reasoning und Content-Management

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Mag. Christoph Kiss, BSc.
Matrikelnummer 01326346

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Inf. Dr.-Ing. Jürgen Dorn

Wien, 1. November 2022

Christoph Kiss	Jürgen Dorn

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Integration of Case-Based Reasoning and Content-Management

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Mag. Christoph Kiss, BSc.
Registration Number 01326346

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Inf. Dr.-Ing. Jürgen Dorn

Vienna, 1st November, 2022

Christoph Kiss	Jürgen Dorn

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Mag. Christoph Kiss, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. November 2022

Christoph Kiss

KURZFASSUNG

Viele Unternehmen und Organisationen speichern und verwalten interne Informationen in Content-Management-Systemen. Dieses gespeicherte Wissen ist oft nur schwer für neue Problemstellungen nutzbar. Daten aus bestehenden Installationen von Content-Management-Systemen zur Generierung neuer Lösungen mit Hilfe von Case-Based Reasoning zu verwenden, ist das Ziel dieser Diplomarbeit. Dabei wird untersucht, wie sich ein System für Case-Based Reasoning aufbauend auf einem Content-Management-System realisieren lässt und welche Vor- und Nachteile sich aus diesem Ansatz ergeben. Zudem wird erforscht, wie dabei auch referenzierte Inhalte zur Ähnlichkeitsberechnung verwendet werden können. Gezeigt wird dies an einem Prototyp, der anhand eines Szenarios evaluiert wird. Der in dieser Diplomarbeit erstellte Code unterliegt der GNU General Public License, Version 2 und ist unter https://www.drupal.org/project/cbr veröffentlicht.

Content-Management-Systeme speichern Daten in Inhaltsstrukturen aus Feldern. Diese Felder können um Funktionen zur Berechnung der Ähnlichkeit erweitert werden und werden dadurch zu Attributen eines Falls. Unterstützt wird dies durch die objektorientierte Umsetzung des Content-Management-Systems. Aufbauend auf diesem Konzept wird ein flexibel einsetzbares System für Case-Based Reasoning als Modul für Drupal 9 umgesetzt. Das so entstehende System für Case-Based Reasoning kann nicht nur die Datenstruktur des Content-Management-Systems nutzen, sondern auch deren Weboberfläche zum Verwalten der Wissensbasis, die Administrationsoberfläche zur Konfiguration des Systems und die Rollen- und Benutzerverwaltung zum Management der Berechtigungen. Umgekehrt erweitert Case-Based Reasoning das Content-Management-System um die Möglichkeit, Inhalte über alle Felder hinweg mit anderen Inhalten zu vergleichen und ähnliche Inhalte zu finden. Dies kann in Onlineshops oder bei der Suchmaschinenoptimierung hilfreich sein.

Ebenfalls wird ein Verfahren vorgestellt, bei welchem Inhalte, auf die referenziert wird, zur Ermittlung der Ähnlichkeit verwendet werden können. Dazu werden die referenzierten Inhalte geladen und zusammengefasst. Im Prototyp kann gezeigt werden, dass dieses System auch bei mehreren tausend Testdaten noch immer rasch ähnliche Fälle ermitteln kann. Möglich wird dies durch Caching, das es erlaubt, Zwischenergebnisse der Ähnlichkeitsberechnung zu speichern und später wiederzuverwenden.

Der erstellte Prototyp wird mit myCBR und JColibri verglichen. Dabei bietet der als Webanwendung umgesetzte Prototyp viele Vorteile bei der Bedienung, der Konfiguration und der Erweiterbarkeit gegenüber den als Desktopapplikationen entwickelten Systemen myCBR und JColibri.

Abstract

Many organizations manage and store their internal information in content management systems. This stored knowledge is often underutilized for solving new problems. The goal of this diploma thesis is to use data from an existing content management system installation to generate solutions using case-based reasoning. It will be examined how a system for case-based reasoning can be developed based on a content management system and what the advantages and disadvantages of this approach are. Furthermore, the inclusion of referenced content in the calculation of the similarity between two cases is researched. The results are demonstrated by implementing a prototype. The prototype is evaluated using a scenario. The source code created in this thesis is licensed under the GNU General Public License, version 2, and is published at https://www.drupal.org/project/cbr.

Content management systems store data in structures consisting of fields. These fields can be extended with functions to calculate the similarity of a case. This is supported by the object-oriented implementation of the content management system. Based on this concept, a fully functional case-based reasoning system is implemented as a module for Drupal 9. The resulting case-based reasoning system uses the data structure of the content management system, but also its web interface to manage the knowledge base and configure the system. The prototype also uses the role and user management features of the content management system. In exchange, case-based reasoning extends the content management system with the possibility of comparing content across all fields. This can be helpful in web shops or for search engine optimization.

In addition, a methodology is presented for how the referenced content can be used to determine similarity. For this purpose, the referenced content is loaded and summarized. The prototype demonstrates that the system is able to quickly identify similar cases, even with thousands of test cases. This is made possible by caching intermediate results and using them to calculate the similarity.

The prototype is compared to myCBR and JColibri. The prototype, implemented as a web application, offers many advantages in terms of usability, configuration, and extensibility compared to myCBR and JColibri, which are developed as desktop applications. This thesis also describes how content management systems can be used as a platform for developing complex applications.

INHALT

ABBILDUNGSVERZEICHNIS

1 EINFÜHRUNG

1.1 PROBLEMSTELLUNG

Content-Management-Systeme werden in Unternehmen und Organisationen nicht nur zum Betrieb von öffentlich erreichbaren Webseiten eingesetzt, sondern dienen auch zur Verwaltung von Daten, welche intern verwendet werden. Das kann beispielsweise eine Anwendung im Intranet sein, in der Mitarbeiter/innen neben ihren Adressdaten, ihre Ausbildung, Fähigkeiten und Erfahrungen speichern. Diese Daten referenzieren oftmals andere Daten, wie etwa in welchen Projekten Mitarbeiter/innen tätig sind. Werden mit Hilfe des Systems Mitarbeiter/innen für beispielsweise ein neues Projekt gesucht, so erfolgt dies meist über eine Volltextsuche oder eine Kombination aus verschiedenen Filtern. Oft bedarf es viel Erfahrung, nach welchen Stichworten gesucht werden muss oder welche Filter gesetzt werden müssen, um das gewünschte Suchergebnis zu erhalten. Werden zu viele Filter gesetzt, werden möglicherweise Mitarbeiter/innen für das Projekt nicht berücksichtigt, da ihre Fähigkeiten zwar ähnlich sind, aber nicht genau den Filterkriterien entsprechen. Werden zu wenige Filter gesetzt, ist aus der Menge der Resultate nur schwer eine Auswahl der am besten geeigneten Mitarbeiter/innen zu treffen. Um dieses Problem zu lösen, sollen Konzepte, Vorgehensweisen und Algorithmen des Fallbasierten Schließens in die Domäne des Content-Managements integriert werden.

Aus gelösten Problemen lernen – was für Menschen ganz natürlich ist, ist für Maschinen nicht trivial. Fallbasiertes Schließen, im Englischen auch Case-Based Reasoning (CBR) genannt, verfolgt genau diesen Ansatz. Aus einer Wissensbasis von gelösten Problemstellungen, auch „Fälle“ oder „Cases“ genannt, soll zu einer neuen Problemstellung ein möglichst ähnlicher Fall ermittelt werden. Dieser soll dem/der Anwender/in bei der Lösung seines/ihres Problems behilflich sein. Ist die Problemstellung gelöst, kann die Problemstellung sowie die Lösung ebenfalls der Wissensbasis hinzugefügt werden.

In dieser Diplomarbeit soll Fallbasiertes Schließen in ein Content-Management-System integriert werden. Das Besondere an diesem System soll sein, dass die Beziehung zwischen Inhalten berücksichtigt wird. Dies leitet sich daraus ab, dass Inhalte in Content-Management-Systemen nicht ausschließlich aus Feldern und Attributen von trivialen Datentypen wie Boolean, Integer oder String bestehen, sondern dass Attribute auch von einem bestimmten Typ sein können, welcher es erlaubt, ein oder mehrere Objekte zu referenzieren. Diese Objekte sind wiederum aus Attributen von trivialen Datentypen aufgebaut. Dabei sind sowohl quantitative Attribute wie Boolean oder Integer, als auch qualitative Attribute wie Strings, zu berücksichtigen. Diese referenzierten Objekte

enthalten für die Suche nach ähnlichen Fällen wichtige Informationen und sollen daher dementsprechend genutzt werden.

Aus dieser Problemstellung leiten sich zwei Forschungsfragen ab:

• Wie kann ein System für Fallbasiertes Schließen in ein Content-Management-System integriert werden, sodass bestehende Datenstrukturen und Daten als Wissensbasis fungieren können? Welche Vor- und Nachteile ergeben sich aus dem gewählten Ansatz?

• Wie lassen sich dabei Attribute, welche auf andere Objekte verweisen, für Fallbasiertes Schließen nutzen? Welche Auswirkungen hat dies auf die Laufzeit bei der Suche nach ähnlichen Fällen?

1.2 Erwartetes Resultat

Es soll untersucht werden, wie mit Hilfe von CBR in Content-Management-Systemen Inhalte verglichen werden können, wie weitere Konzepte aus dem Case-Based Reasoning in ein Content-Management-System integriert werden können und welche Vor- und Nachteile dies im Vergleich mit bestehenden Lösungen hat. Gezeigt werden soll dies anhand eines Prototyps. Dieser Prototyp soll ein Modul für ein Content-Management-System sein, welches alle nötigen Algorithmen mitbringt, damit aus einer bestehenden Installation des CMS ein System für Fallbasiertes Schließen wird. In diesem System sollen neben String, Boolean und Integer auch Attributtypen verwendet werden können, welche auf andere Objekte verweisen. Die Laufzeit bei der Suche nach ähnlichen Fällen mit diesen sogenannten hierarchischen Attributen soll theoretisch geschätzt werden und es soll untersucht werden, wie die Laufzeit möglichst niedrig gehalten werden kann.

1.3 Methodisches Vorgehen

Vor der Entwicklung wird der State of the Art durch eine ausführliche Literaturrecherche und die genauere Untersuchung der beiden CBR-Systeme myCBR und JColibri evaluiert. Hier wird untersucht, ob die bestehenden Lösungen in ein Content-Management-System integriert werden können. Zudem werden mehrere Content-Management-Systeme verglichen und dann eine Entscheidung für ein Content-Management-System getroffen, für welches der Prototyp realisiert wird. Die Definition der Kriterien für diese Entscheidung erfolgt später in dieser Arbeit.

Rapid Prototyping: Nach jedem wichtigen Meilenstein wird ein Prototyp des CBR-Systems erstellt, an welchem dieser Schritt anhand einer kleinen Datenmenge evaluiert wird. Liefert der Prototyp das gewünschte Ergebnis, wird mit dem nächsten Meilenstein begonnen. Falls nicht, so wird der Programmcode des Prototyps überarbeitet und erneut überprüft.

Szenario-Entwicklung: Um das CBR-System in einem praktischen Einsatz zu zeigen und zu testen, wird das Szenario eines CBR-Systems im Softwareprojektmanagement entworfen und das CBR-System entsprechend konfiguriert. Das CBR-System soll ermöglichen, dass zu einem Projekt ähnliche Projekte gefunden werden können. Dabei soll das Szenario möglichst praxisnah sein und zudem alle Features, wie beispielsweise hierarchische Attribute, enthalten.

1.4 Evaluierung

Zur Beantwortung von Forschungsfrage 1 wird eine Architekturanalyse des Prototyps durchgeführt. Zur Beantwortung von Forschungsfrage 2 wird das finale Szenario mit einer großen Zahl von zufällig generierten Testdaten gefüllt und dann hinsichtlich Performance und Bedienbarkeit untersucht. Die Komplexität bei der Suche nach ähnlichen Fällen wird hinsichtlich des Speicherplatzes und der Laufzeit mit Hilfe der O-Notation ermittelt. Das Szenario kann die Forschungsfrage beantworten, da im Bereich des Projektmanagements bei der Modellierung des Falls „Projekt“ Attribute auftreten, welche sich nicht oder nur schwer durch „triviale“ Datentypen abbilden lassen, wie beispielsweise Projektmitarbeiter/innen. Ein Projekt kann 1..n unterschiedliche Projektmitarbeiter/innen haben. Diese haben ihrer/seinerseits wieder Attribute wie beispielsweise Fähigkeiten und Erfahrungen. Projektmanagement als Anwendungsgebiet für CBR ist in der Literatur mehrfach erwähnt (vgl. [1], [2], [3], [4]).

1.5 Veröffentlichung

Der Prototyp wird nach der Evaluation mit den daraus gewonnenen Erkenntnissen verbessert. Das fertige Modul wird anschließend unter eine Open-Source-Lizenz gestellt und veröffentlicht. Damit kann das Modul sowohl von interessierten Anwender/innen verwendet werden, als auch als Ausgangsbasis für weitere wissenschaftliche Arbeiten dienen.

Im nächsten Abschnitt werden die Grundlagen des Case-Based Reasoning erklärt. Dabei wird bewusst Literatur aus den Anfangsjahren des Case-Based Reasonings verwendet, da praktisch alle weiteren Arbeiten auf diesen Werken aufbauen.

2 GRUNDLAGEN

„We retrieve a prior case from memory, attempt to determine its relevance, and decide what to do, based upon what happened in that case. That is what I have called case-based reasoning." [5, S. 11]

2.1 CASE-BASED REASONING

2.1.1 EINFÜHRUNG

In der wissenschaftlichen Literatur wird Case-Based Reasoning ab Mitte der 1980er immer häufiger erwähnt. Die Forschung zu Case-Based Reasoning erfolgte damals einerseits aus dem Blickwinkel der Kognitionswissenschaft mit dem Ziel, menschliches Verhalten zu modellieren, und anderseits aus der Sicht der Forschung zum Thema der Künstlichen Intelligenz, mit dem Ziel, KI-Systeme effizienter zu gestalten [6, S. 4]. Damals wurden Expertensysteme vielfach regelbasiert umgesetzt. Das bedeutet, dass Expert/innen einer bestimmten Domain nach den Regeln befragt werden, mit denen sie Entscheidungen treffen. Diese werden dann im KI-System modelliert. Ein regelbasiertes Expertensystem hat eine Bibliothek von Regeln in der Form „Wenn A, dann B" und nutzt diese, meist in einer bestimmten Abfolge verkettet, zum Lösen von Problemen [5, S. 26]. Dabei ist es jedoch fraglich, ob Expert/innen wirklich ihre Entscheidungen in Form von Regeln treffen. *Riesbeck et al.* weisen auf den Unterschied zwischen Lehrbuchwissen und Wissen aus Erfahrung hin. Dabei ist es meist das Wissen aus den eigenen Erfahrungen, das Expert/innen zu Expert/innen macht [5, S. 10]. Zudem sind die meisten Anwendungsgebiete so komplex, dass es nicht möglich ist, alle Regeln, welche zur Entscheidungsfindung notwendig sind, zu spezifizieren [5, S. 26].

Wenn Menschen ein neues Problem lösen, so tun sie dies selten nach vorgegebenen Regeln. Stattdessen erinnern sie sich an ein ähnliches Problem, das sie entweder selbst gelöst haben oder von einer Lösung gehört haben, und passen die Lösung auf das aktuelle Problem an [5, S. 5]. *Riesbeck et al.* nennen diesen Prozess „Case-Based Reasoning" [5, S. 11]. Sie verweisen darauf, dass auch an Schulen Wissen in Form von Beispielen vermittelt wird. Etwa in der Mathematik werden Sachverhalte an Beispielen dargestellt, anhand deren dann Schüler/innen das Prinzip an ähnlichen Problemen anwenden sollen [5, S. 7]. Von besonderer Bedeutung sind das Extrahieren und Anpassen des „Lösungsweges" aus dem Beispiel. Gebe es etwa in der Wirtschaft ein Beispiel, das ohne Anpassung unweigerlich zum Erfolg führen würde, so würde es jede/r anwenden. Dem ist jedoch offensichtlich nicht so. Stattdessen ändert jeder durchgeführte „Fall" die Welt, sodass eine exakte Wiederholung nicht funktionieren würde. Hingegen kann die angewandte Methode in einer anderen Branche, einer anderen Stadt oder mit einem anderen Produkt erfolgreich sein. [5, S. 6f.]

Case-Based Reasoning bezeichnet also das Lösen von Problemen mit Hilfe von Cases, im Deutschen als Fälle bezeichnet. Laut *Richter 2003* kann man sich zum einfacheren Verständnis dabei einen Fall im juristischen Sinn vorstellen und unter dem Begriff „Schließen" die Vorgehensweise, wie diese gelösten Fälle für zukünftige Fälle nutzbar gemacht werden – wie im angelsächsischen Recht üblich [7, S. 180]. Die grundlegende Funktionsweise basiert auf der Annahme, dass ähnliche Probleme mit ähnlichen Lösungen gelöst werden können [6, S. 3].

In einem System für Case-Based Reasoning wird unter einem Fall eine Problemstellung mit einer Lösung verstanden, formal notiert als Fall = (Problem, Lösung) [7, S. 180]. Ein CBR-System hat in seiner Fallbasis eine Menge von Problemen gespeichert sowie eine Menge von Lösungen, wobei jedem Problem eine Lösung zugeordnet ist (siehe Abbildung 1). Um nun ein Problem zu lösen, wird in der Menge der im System gespeicherten Probleme nach dem ähnlichsten Problem gesucht. Dieses Problem verweist auf eine Lösung, die nun dupliziert und an das Eingabeproblem angepasst wird. [6, S. 9]

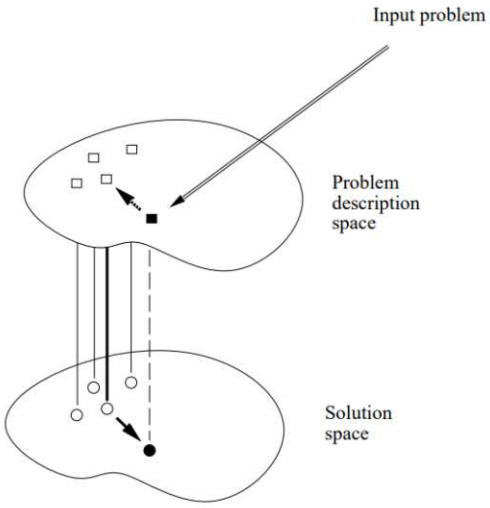

Abbildung 1: Generierung einer neuen Lösung in einem CBR-System. Abbildung übernommen von [1, S. 9].

Damit ist der Prozess jedoch noch nicht zu Ende. Die vom CBR-System vorgeschlagene Lösung wird nun von dem/der Benutzer/in auf das tatsächliche Problem angewandt. Wenn die vorgeschlagene Lösung das Problem erfolgreich gelöst hat, dann wird die Lösung in dieser Form der Fallbasis hinzugefügt. Auf diese Art wird die Wissensbasis des Systems vergrößert, wodurch es eher funktionierende Lösungen vorschlagen kann und das System so seine „Erfahrung" aufbaut. [6, S. 8]

Auch wenn die vorgeschlagene Lösung nicht zum Ziel geführt hat, kann das System daraus lernen. Die gespeicherte Lösung wird so angepasst, dass sie mit der tatsächlichen Lösung übereinstimmt. Dieses Anpassen wird in der Terminologie der CBR-Forschung „Reparieren" genannt. Zudem kann Information über den Misserfolg zur Analyse mit neuen Informationen gespeichert werden, sowie künftig bei ähnlichen Problemen eine Warnung angezeigt werden, welche Lösungen wahrscheinlich nicht zum Erfolg führen. [6, S. 8]

2.1.2 Der CBR Cycle

Der Prozess, der notwendig ist, um von einem neuen Problem zu einem neuen, in der Wissensbasis gespeicherten Fall zu gelangen, wird oft als CBR Cycle beschrieben und dargestellt. Dieser wurde von *Agnar Aamodt* und *Enric Plaza* 1994 vorgestellt. Genauer betrachtet umfasst dieser, die als „four REs" bezeichneten, folgenden Schritte: Retrieve, Reuse, Revise, Retain [8, S. 45].

Retrieve

Der Prozess (siehe Abbildung 2) beginnt damit, dass ein/e Anwender/in eine neue Problembeschreibung hinzufügt, diese wird im System als neuer Fall angelegt [8, S. 46]. Basierend auf dieser Beschreibung wird ein möglichst ähnlicher Fall in der Datenbank der bisher gelösten Fälle gesucht. Um einen möglichst ähnlichen Fall finden zu können, müssen die bisher gespeicherten Fälle mit entsprechenden Informationen versehen sein. Diese Informationen sind in verschiedene Dimensionen gegliedert (zum Beispiel: Zeitpunkt des Problems, beteiligte Personen und so weiter). Die Ähnlichkeit zwischen den Fällen hängt sowohl davon ab, wie stark die einzelnen Informationen innerhalb der Dimensionen übereinstimmen, als auch von der Wichtigkeit der Dimensionen (zum Beispiel: die beteiligten Personen sind wichtiger als der Zeitpunkt des Problems). [5, S. 32f.]

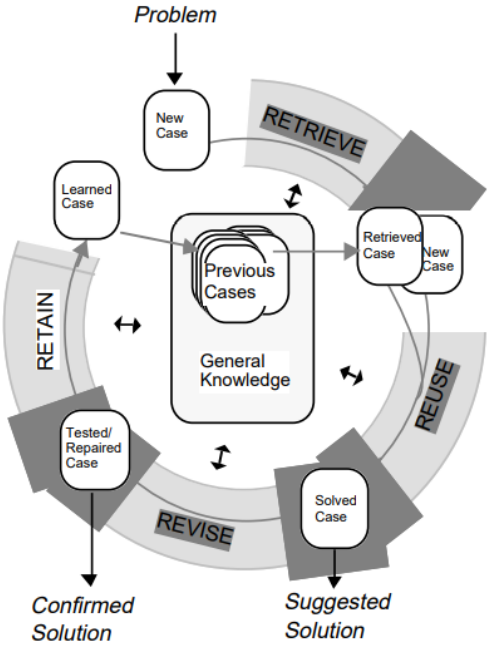

Abbildung 2: CBR Cycle von *Agnar Aamodt* und *Enric Plaza*. Abbildung übernommen von [8, S. 46].

Reuse

Die Lösung des als am ähnlichsten bestimmten Falls wird bei einfachen Systemen unverändert als Lösung des neuen Falls genutzt. Andere Systeme vergleichen hingegen die Problembeschreibung des neuen Falls mit der Problembeschreibung des gefundenen Falls und passen die Lösung dementsprechend an, bevor sie dem neuen Fall hinzugefügt wird. [8, S. 54]

Die Adaptierung eines neuen Falls mit Hilfe des gefundenen Falls durch das CBR-System erfolgt meist regelbasiert. Damit benötigen solche CBR-Systeme Wissen als Regeln modelliert, was durch den Einsatz von CBR-Systemen eigentlich vermieden werden soll [6, S. 22f.]. Wie im vorherigen Abschnitt beschrieben, ist es nicht immer möglich, Wissen in Form von Regeln zu modellieren. Zudem ist eine Adaptierung durch das CBR-System nicht immer möglich, beispielsweise wenn

etwa die Lösung in Form von Videos hinterlegt ist. Daher verzichten viele Systeme auf eine automatische Adaptierung und überlassen das Anpassen der Lösung dem/der Benutzer/in. [6, S. 10f.]

REVISE

In dieser Phase wird die generierte Lösung evaluiert und gegebenenfalls repariert. Die Evaluierung erfolgt außerhalb des CBR-Systems – etwa indem die vorgeschlagene Lösung beim konkreten Problem angewandt wird. Diese Evaluierung kann, je nach Einsatzgebiet des CBR-Systems, von ein paar Stunden bis hin zu mehreren Monaten (zum Beispiel: in der Medizin) dauern. Der Fall bleibt in dieser Zeit in der Fallbasis gespeichert, wird jedoch als nicht-evaluiert gekennzeichnet. [8, S. 55]

Wenn der vorgeschlagene Lösungsweg nicht erfolgreich war, wird versucht zu erklären, warum dies geschehen ist. In manchen Anwendungen wird aus der erfolgten Diagnose der Fall automatisch repariert (ähnlich dem Adaptieren im vorherigen Schritt), in anderen Systemen werden bei der Reparatur alle neu verfügbaren Informationen zum Fall hinzugefügt und geschaut, welcher Fall nun dem Fall am ähnlichsten ist. Ist es erneut der Fall, dessen Lösung schon probiert wurde, dann wird jener Fall genommen, der nach diesem Fall der ähnlichste ist und dessen Lösung angepasst und ausprobiert. Wichtig ist, dass auch die Lösung, die nicht erfolgreich war, im System bleibt und mit der erfolgreichen Lösung verknüpft wird. Wird künftig bei einem neuen Problem die nicht erfolgreiche Lösung vorgeschlagen, so kann das System darauf hinweisen, dass es mit dieser Lösung Schwierigkeiten gab und auf eine funktionierende Lösung verweisen. [5, S. 53f.]

RETAIN

In diesem Schritt wird das durch diesen Fall gelernte Wissen dem CBR-System hinzugefügt und zur künftigen Nutzung aufbereitet. Dazu kann entweder ein bereits gespeicherter Fall mit neuen Informationen erweitert werden oder der neue Fall wird in die Fallbasis aufgenommen. Zudem wird darauf geachtet, dass der Fall um alle Informationen ergänzt wird, die notwendig sind, damit dieser künftig möglichst gut mit neuen Problemstellungen verglichen werden kann. Auch das gewonnene Wissen aus dem Revise-Schritt wird aufbereitet und gespeichert. Dies kann entweder direkt im gelösten Fall gespeichert werden oder es werden separate „Failure Cases“ angelegt. [8, S. 56]

Mittlerweile gibt es viele Abwandlungen des von *Agnar Aamodt* und *Enric Plaza* 1994 vorgestellten CBR Cycles. So besteht beispielsweise der von *Eisenstadt* 2019 vorgeschlagene „FLEA-Cycle” aus den Schritten „Find”, „Learn”, „Explain” und „Adapt”, welche beliebig kombiniert und auf Wunsch wiederholt durchlaufen werden können [9, S. 53].

2.1.3 Die Ähnlichkeit von Fällen ermitteln

Wie gut ein CBR-System funktioniert, hängt maßgeblich davon ab, ob zu einem neuen Problem ein ähnlicher Fall in der Wissensbasis gefunden wird. Damit dies gelingt, wird jeder Fall durch unterschiedliche Dimensionen, auch Attribute genannt, beschrieben. Das kann etwa der akademische Grad eines/einer Mitarbeiter/in [10, S. 69], das Gewicht eines/einer Patienten/Patientin [11, S. 165] oder auch die dominante Farbe eines Kunstwerks [12, S. 54] sein. Soll nun zu einem neuen Fall ein ähnlicher Fall gefunden werden, so wird für alle bereits gespeicherten Fälle jedes Attribut mit dem äquivalenten Attribut des neuen Falls verglichen. Ergebnis dieses Vergleichs ist die „Local Similarity" [10, S. 69]. Je nach Datentyp des Attributs wird hier unterschiedlich vorgegangen. Beim Vergleich von Zahlenwerten wird beim Berechnen der Ähnlichkeit zuerst die Differenz der beiden Werte auf das Intervall [0,1] normalisiert [13, S. 331] und anschließend wird dieser Wert von 1 subtrahiert. Beim Vergleich von Strings kann die Cosinus-Ähnlichkeit verwendet werden [10, S. 69, 14, S. 25]. Alternativ wird zum Vergleichen von Strings auch der Jaccard-Koeffizient verwendet [12, S. 55, 14, S. 25].

Nicht alle Attribute haben dieselbe Bedeutung beim Vergleich von zwei Fällen. Um dies zu berücksichtigen, kann jedem Attribut eine Wichtigkeit, sprich ein Gewicht, zugeordnet werden. Zur Berechnung der Ähnlichkeit zwischen zwei Fällen, der „Global Similarity", wird dann das gewichtete arithmetische Mittel aller „Local Similarities" verwendet [10, S. 70, 15, S. 127, 16, S. 812].

Die „Global Similarity", oft auch nur „Similarity" oder Ähnlichkeit genannt, ist ein Zahlenwert im Intervall von 0 bis 1. Für die Similarity *sim* und die Fälle *x, y* gilt [17, S. 250]:

- Reflexivität: sim(x,x) == 1
- Symmetrie (x, y) == sim(y, x)

Manche Systeme, etwa myCBR, erlauben dem/der Nutzer/in die manuelle Zuordnung der Ähnlichkeit, etwa wenn akademische Grade verglichen werden sollen. Hier kann entweder symmetrisch oder asymmetrisch vorgegangen werden, womit die obige Aussage zur Symmetrie nicht für alle CBR-Systeme zutrifft. [10, S. 69]

Der absolute Wert der Ähnlichkeit hat nur wenig Bedeutung, wichtiger ist das Verhältnis der Fälle zueinander, welches durch die Ähnlichkeit bestimmt wird [17, S. 250]. Der Fall mit der größten Ähnlichkeit wird im zuvor beschriebenen CBR Cycle zur Generierung der Lösung verwendet.

2.1.4 Einsatzgebiete von CBR-Systemen

Schon früh wurde der Einsatz von Case-Based Reasoning im Bereich der Strafverfolgung in den USA erforscht, etwa mit dem „JUDGE"-Projekt im Jahr 1986, bei dem versucht wurde, die Schuld oder Unschuld eines/einer Mandaten/Mandantin anhand zuvor gespeicherter Entscheidungen zu bestimmen [5, S. 27f.]. Ein ähnliches Prinzip verfolgte das ebenfalls 1986 vorgestellte Projekt „HYDRO", das anhand einer Fallbasis bestimmte, ob eine Patentverletzung vorliegt und wie diese verteidigt werden kann [5, S. 28]. Ebenfalls früh zum Einsatz kam CBR im Bereich der Medizin, etwa 1988 im Projekt „CASEY", welches bei der Untersuchung von Herzerkrankungen helfen sollte. Dem System wurden als Input die Symptome des/der Patienten/Patientin gegeben, welches als Antwort die möglichen Ursachen für diese lieferte. [5, S. 30]

Ein weiteres Einsatzgebiet für Case-Based Reasoning sind Empfehlungssysteme, welche etwa im E-Commerce zum Einsatz kommen. Dabei bietet Case-Based Reasoning den Vorteil, dass dem/der Anwender/in nicht nur ein Produkt empfohlen wird, sondern dass das System auch darlegen kann, warum es dieses Produkt empfiehlt. Untersuchungen haben gezeigt, dass Anwender/innen diesen Systemen mehr vertrauen als Systemen, welche ihre Entscheidungen für den/die Benutzer/in intransparent treffen. Stimmen die Wünsche der benutzenden Person nicht vollständig mit den Vorschlägen überein, kann das CBR-System zudem die Abweichungen erklären. Anwendung sind etwa Empfehlungen für Autos oder Pauschalreisen. [18, S. 179ff.]

Case-Based Reasoning kann zudem eingesetzt werden, um Fehler in technischen Komponenten zu finden und dafür Lösungen zu zeigen. Ein solches System wurde 2021 für den Einsatz bei künftigen Missionen ins Weltall vorgestellt [19, S. 1].

2.1.5 Neueste Entwicklungen aus dem Bereich der CBR-Forschung

Ein Großteil der bisher zitierten Literatur stammt aus der frühen Phase des Case-Based Reasoning. Dieser Abschnitt soll deutlich machen, dass zu Case-Based Reasoning weiterhin aktiv geforscht und es für unterschiedlichste Problemstellungen eingesetzt wird. Auf der jährlich stattfindenden „International Conference on Case-Based Reasoning" werden die aktuellen Entwicklungen aus der Forschung diskutiert. Auch 2021 wurden einige interessante Projekte und Forschungsarbeiten vorgestellt, auf welche nun kurz eingegangen wird. So wurde etwa ein CBR-System vorgestellt, bei dem die Adaptierung der gefundenen Lösung mit Hilfe eines neuronalen Netzes erfolgt [20, S. 279]. Zwei Konferenzbeiträge beschäftigen sich mit der Analyse von Zeitreihen mit Hilfe von Case-Based Reasoning. *Delaney et al.* bestimmen mit Hilfe von CBR anhand von Elektrokardiogrammen, ob bei Patient/innen Erkrankungen vorliegen und welche. Der Beitrag von *Dolphin et al.* hingegen stellt einen neuen Ansatz zum Vergleich von Kursverläufen auf den Finanzmärkten mit

Hilfe von CBR vor [21, S. 64ff.]. Case-Based Reasoning kann auch zum Umwandeln von Informationen in eine andere Darstellungsform verwendet werden. So wird ein CBR-System zum Umwandeln von Daten in Text vorgestellt [22, S. 232ff.], ebenso ein CBR-System, welches aus Text SQL-Abfragen generiert [23, S. 294ff.]. Zudem lässt sich CBR verwenden, um aus einer Vielzahl von Optionen die am besten passenden auszuwählen und zu kombinieren. Von *Abbas et al.* wurde ein System zum Generieren von möglichst abwechslungsreichen, aber individuellen Speiseplänen entworfen und untersucht [24, S. 1ff.]. *Flogard et al.* generieren mit Hilfe von CBR Checklisten zur Überprüfung der Arbeitssicherheit [25, S. 94ff.].

2.2 Bestehende CBR-Systeme

In der jüngeren Literatur zur CBR-Forschung werden primär myCBR und JColibri erwähnt [10, 67ff, 15, S. 130, 25, S. 103, 26, 131ff, 27, 3ff]. Beide Systeme sind Open Source und wurden an europäischen Universitäten entwickelt.

myCBR[1] entstand aus einer Kooperation zwischen dem Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) und der School of Computing and Technology der University of West London. Die neueste Version myCBR 3.1 wurde im Mai 2015 veröffentlicht und kann kostenlos von http://myCBR-project.org/download.html heruntergeladen werden.

JColibri wurde an der Universidad Complutense de Madrid entwickelt. Die neueste Version JColibri 3 kann von https://sourceforge.net/projects/JColibri-cbr/ heruntergeladen werden. Sie wurde am 14.01.2019 veröffentlicht. JColibri richtet sich an Systemdesigner/innen, welche damit ein eigenes CBR-System entwickeln können [28, S. 70].

In diesem Abschnitt werden myCBR und JColibri genauer untersucht. Dazu wird ein Szenario in beiden CBR-Systemen umgesetzt. Ausgangslage ist ein Unternehmen, das im Projektgeschäft tätig ist. Mit Hilfe von Case-Based Reasoning sollen Erfahrungen, die bei der erfolgreichen Durchführung von Projekten entstanden sind, für neue Projekte nutzbar gemacht werden. Wenn also ein neues Projekt begonnen wird, dann werden alle bekannten Parameter in das CBR-System eingetragen und das System liefert eine Liste ähnlicher Projekte. Das im CBR-System gespeicherte Wissen in Form von Dokumenten, Zeitplänen und Links zu externen Ressourcen kann dann an die neue Problemstellung angepasst angewendet werden. Dabei wird nicht nur explizites Wissen weitergegeben. Da zu jedem Projekt auch die daran beteiligten Mitarbeiter/innen gespeichert sind, kann ein neues Projektteam auch aus Mitarbeiter/innen ähnlicher Projekte gebildet werden oder von

[1] http://mycbr-project.org/, abgerufen am 02.04.2022

diesen unterstützt werden. Technisch wird dazu im CBR-System die in Abschnitt 2.3.4 genauer vorgestellte Datenstruktur aus Projekten, Mitarbeiter/innen und Artefakten umgesetzt.

2.2.1 myCBR

myCBR besteht aus einem „myCBR Workbench" genannten Tool, welches eine Benutzeroberfläche besitzt und einem Software-Development-Kit, mit dem Softwareentwickler/innen myCBR in ihre Anwendungen integrieren können. Die Benutzeroberfläche von myCBR erlaubt das Modellieren und Verwalten von Fällen. Ebenso kann über die Benutzeroberfläche nach ähnlichen Fällen gesucht werden. Bevor Fälle in myCBR importiert oder angelegt werden können, muss ihre Struktur modelliert werden. Dazu werden sogenannte „Conzepts" in der Benutzeroberfläche angelegt, im Fall des Szenarios sind das „Projekte", „Mitarbeiter/innen" und „Artefakte". Zu jedem „Conzept" können anschließend Attribute hinzugefügt werden. Diese können die Datentypen „Boolean", „Date", „Double", „Float", „Integer", „Interval", „String" oder „Symbol" haben. Für jedes Attribut kann angegeben werden, ob ein Wert oder mehrere Werte gespeichert werden sollen. Zu einem „Conzept" kann auch ein Attribut vom Typ „Conzept" hinzugefügt werden. Damit können Hierarchien modelliert werden, im Fall des Szenarios kann so von „Projekte" auf „Artefakte" und „Mitarbeiter/innen" verwiesen werden. Nachdem die Struktur modelliert wurde, können Fälle aus CSV-Dateien importiert werden.

Ist die Struktur der Fälle modelliert und sind entsprechende Testdaten vorhanden, muss jedem Attribut eine Funktion zum Vergleich unterschiedlicher Werte zugeordnet werden. Für numerische Werte und Datumswerte müssen ein Minimum und ein Maximum angegeben werden. Anschließend kann angegeben werden, ob von zwei Werten die Differenz oder der Quotient zur Berechnung der Ähnlichkeit verwendet werden soll. Weiters kann angegeben werden, ob die Ähnlichkeit mit Hilfe eines Schwellenwertes bestimmt werden soll. Dies kann zum Beispiel sinnvoll sein, wenn alle Projekte, deren Budget um bis zu 100.000 € abweicht, als gleich betrachtet werden sollen. Alternativ kann der Exponent einer Polynomfunktion angegeben werden, also ob eine lineare, quadratische, etc. Funktion zur Bestimmung der Ähnlichkeit verwendet werden soll. Die Funktion wird in der Benutzeroberfläche von myCBR grafisch dargestellt.

„Symbole" sind vordefinierte Zeichenketten, wie etwa Programmiersprachen und akademische Titel. Sie können auf drei Arten verglichen werden: Entweder werden die Ähnlichkeiten zwischen den Symbolen in einer Matrix festgelegt (siehe Abbildung 3) oder jedem Symbol wird ein Zahlenwert zugeordnet, der dann wie in Abschnitt 2.1.3 beschrieben weiterverarbeitet wird. Dabei werden alle Symbole in einer Baumstruktur angeordnet (siehe Abbildung 4).

Über die Benutzeroberfläche von myCBR können für Strings keine Ähnlichkeitsfunktionen konfiguriert werden. Ein Blick in den Quellcode[2] zeigt jedoch, dass die Ähnlichkeit von Strings unterschiedlich berechnet werden kann. Beispielsweise können sie als Ganzes verglichen werden, also dass Zeichenketten, die exakt übereinstimmen, eine Ähnlichkeit von eins haben und alle Strings, welche voneinander abweichen, eine Ähnlichkeit von null haben. Weiters kann wahlweise die Levenshtein-Distanz oder der Dice-Koeffizient der aus dem String gebildeten N-Grams ermittelt werden und zur Berechnung der Ähnlichkeit verwendet werden.

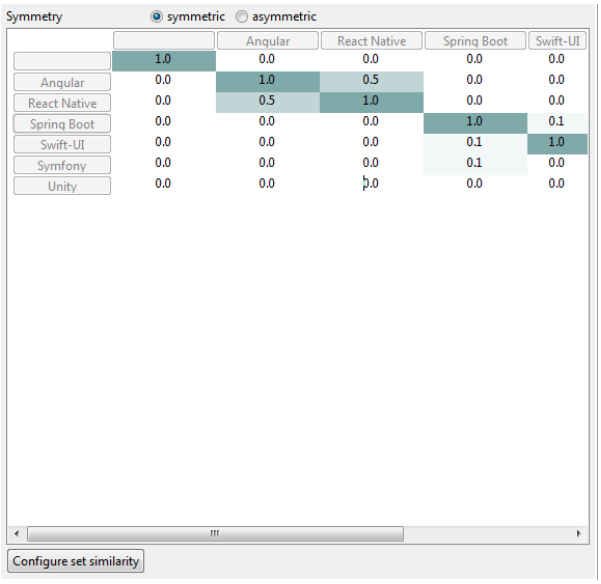

Abbildung 3: Festlegen der Ähnlichkeiten zwischen Symbolen in myCBR mit einer Tabelle. Eigene Aufnahme.

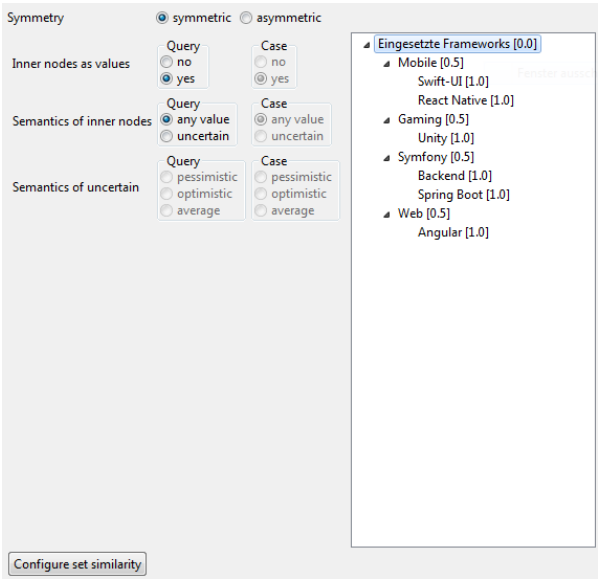

Abbildung 4: Festlegen der Ähnlichkeiten zwischen Symbolen in myCBR mit einer Hierarchie. Eigene Aufnahme.

Auf Wunsch können Attribute mehrere Werte speichern. Wie in Abbildung 5 ersichtlich ist, kann zur Ermittlung der Ähnlichkeit zwischen Attributen mit mehreren Werten entweder jener Wert genommen werden, welcher am meisten mit dem des zu vergleichenden Falls übereinstimmt, jener, der am wenigsten übereinstimmt, oder es wird für jeden Wert des Falls, für den ähnliche Fälle gesucht werden sollen, der am meisten übereinstimmende Wert des zu vergleichenden Falls ermittelt. Von den so gefundenen Wertepaaren kann nun

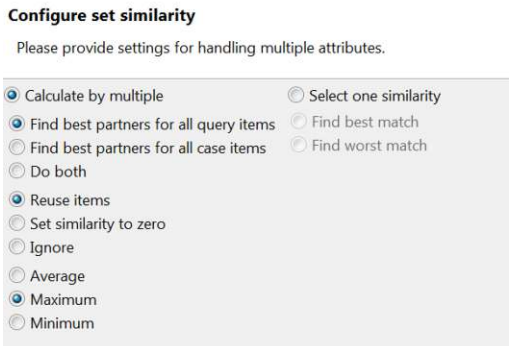

Abbildung 5: Dialog für das Konfigurieren der Ähnlichkeit von Attributen, welche mehrere Werte enthalten können. Eigene Aufnahme.

[2] https://github.com/kerstinbach/mycbr-sdk/blob/master/src/de/dfki/mycbr/core/similarity/StringFct.java, abgerufen am 27.03.2022

der Mittelwert, das Minimum oder das Maximum der Ähnlichkeiten als Ähnlichkeit genommen werden. Beim Case-Based Reasoning wird ein neu angelegter Fall mit den bereits in der Fallbasis gespeicherten Fällen verglichen. In myCBR wird der neue Fall „Query“ genannt und die bereits in der Fallbasis gespeicherten Fälle werden als „Case Items“ bezeichnet. Zu jedem Wert in der „Query“ können die am besten passenden Partner aus den „Case Items“ ermittelt werden und umgekehrt. Zudem kann beides zur Berechnung der Ähnlichkeit ermittelt werden.

Um von den zuvor beschriebenen, lokalen Ähnlichkeiten zur globalen Ähnlichkeit zu gelangen, bietet myCBR mehrere Optionen. So kann jedem Attribut ein Gewicht zugeordnet werden und das gewichtete arithmetische Mittel aller lokalen Ähnlichkeiten ergibt die globale Ähnlichkeit. Alternativ kann auch die euklidische Distanz oder das Minimum / Maximum der lokalen Ähnlichkeiten verwendet werden. Die globale Ähnlichkeit wird bei der Suche nach ähnlichen Fällen dem/der Benutzer/in angezeigt. Eine weitere automatische Anpassung der Fälle erfolgt nicht. Damit unterstützt myCBR nur den ersten Schritt des CBR Cycles, das Retrieve.

Nachteilig erwies sich beim myCBR Workbench der Umstand, dass myCBR nicht mehr mit einer neuen Version von Java gestartet werden kann, sondern nur mit Java 8 ausgeführt werden kann. Da myCBR nicht mehr weiterentwickelt wird, ist hier nicht mit einer Behebung des Problems zu rechnen, auch wenn der Quellcode vorliegt und das Projekt so jederzeit von Dritten wiederbelebt werden könnte. Ein Beispielcode des Software-Development-Kits konnte hingegen mit Java 18 gestartet werden.

2.2.2 JCOLIBRI

Das von der Universidad Complutense de Madrid entwickelte System JColibri besteht, ähnlich wie myCBR, aus zwei Komponenten: Das JColibri Framework[3] stellt für die Entwicklung einer eigenen CBR-Anwendung entsprechende Methoden zur Verfügung, während Colibri Studio[4] eine Entwicklungsumgebung ist, welche auf die Anwendungsentwicklung mit JColibri angepasst wurde. Colibri Studio basiert auf der Entwicklungsumgebung „Eclipse“. Wird eine neue CBR-Anwendung erstellt, so stellt Colibri Studio Vorlagen dafür zur Verfügung. Ein Assistent erlaubt das Anlegen einer Fallstruktur (siehe Abbildung 6), das Konfigurieren einer Datenquelle sowie das Zuweisen von Funktionen zur Berechnung der globalen und lokalen Ähnlichkeit (siehe Abbildung 7). Im Abschluss generiert der Assistent Java-Klassen für die Fallstruktur sowie einen Programmcode, welcher das Ermitteln von ähnlichen Fällen zu einem als Query bezeichneten Fall übernimmt. Ebenso

[3] https://gaia.fdi.ucm.es/research/colibri/jcolibri/, zuletzt abgerufen am 07.04.2022
[4] https://gaia.fdi.ucm.es/research/colibri/colibristudio/, zuletzt abgerufen am 07.04.2022

wird eine XML-Datei, in welcher der Datenimport aus einer CSV-Datei konfiguriert wird, erstellt. Der Aufruf des Datenimports muss jedoch selbst implementiert werden, auch wenn JColibri entsprechende Methoden zur Verfügung stellt. Ebenso muss die Benutzeroberfläche für die Eingabe von neuen Fällen, einer Suchabfrage oder die Ausgabe der Ergebnisse selbst implementiert werden. Der generierte Code unterstützt nur das „Retrieve“ aus dem CBR Cycle, weitere Schritte müssen selbst implementiert werden.

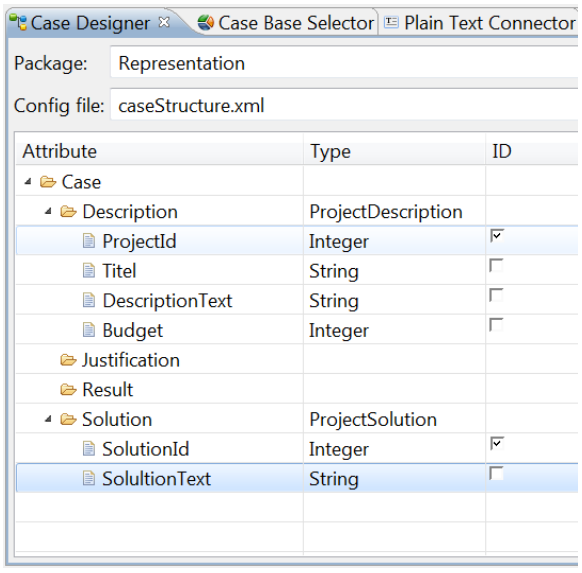

Abbildung 6: Erstellen der Fallstruktur in Colibri-Studio. Eigene Aufnahme.

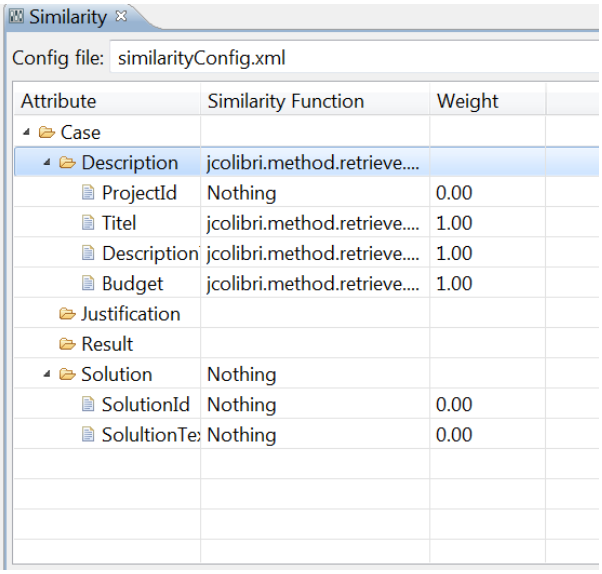

Abbildung 7: Zuordnung einer Funktion zur Berechnung der Ähnlichkeit und eines Gewichts zu einem Attribut. Eigene Aufnahme.

Das zuvor in myCBR umgesetzte Szenario konnte mit JColibri nicht umgesetzt werden, da JColibri zwar sogenannte „Combound Attributes“ erlaubt, welche wiederum aus „Simple Attributes“, also Attributen vom Typ String, Integer oder Boolean bestehen, diese jedoch nur einen Wert haben können. Damit ist es nicht möglich, dass ein Projekt mehrere Mitarbeiter/innen oder Artefakte besitzt. Ebenfalls gibt es Einschränkungen im Vergleich zu myCBR bei der Berechnung der globalen Ähnlichkeit. Diese ist auf den gewichteten Mittelwert beschränkt. Im Gegenzug dazu unterstützt JColibri eine Vielzahl von Methoden zur Berechnung der lokalen Ähnlichkeit. Für Strings sind das unter anderem die Kosinus-Ähnlichkeit, der Dice-Koeffizient, der Jaccard-Koeffizient oder der Szymkiewicz-Simpson-Koeffizient (auch als Overlap coefficient bezeichnet). Für numerische Werte kann beispielsweise die Ähnlichkeit zwischen zwei Zahlenwerten in einem Intervall nach der Formel $1 - (|x - y|/\text{interval})$ berechnet werden oder es wird eine Ähnlichkeit von eins verwendet, wenn die Differenz zwischen Zahlenwerten unter einem von dem/der Entwickler/in definierten Schwellenwert liegt.

Mit JColibri können komplexe CBR-Lösungen entwickelt werden. Auch bei der Verwendung von Colibri Studio muss Programmcode geschrieben werden, womit es für den/die Anwender/in des Systems beispielsweise nicht möglich ist, die Fallstruktur zu ändern. Auch wenn die Konfiguration für die Fallstruktur und die Berechnung der Ähnlichkeit in XML-Dateien gespeichert wird, muss bei einer Änderung Programmcode über Colibri Studio generiert werden.

Beide Systeme sind in Java geschrieben, während Content-Management-Systeme meist in PHP geschrieben sind. Zwar gibt es Möglichkeiten, wie ein Java-Code aus PHP aufgerufen werden kann, dieser Ansatz wird jedoch nicht weiterverfolgt, da er als nicht wartbar und komplex in der Fehlersuche gesehen wird. Stattdessen werden die aus der Analyse der beiden CBR-Systeme gewonnenen Erkenntnisse genutzt, um ein CBR-System als Modul für ein Content-Management-System zu implementieren. Bei der Berechnung der lokalen und globalen Ähnlichkeit wird analog zu den in myCBR und JColibri verwendeten Algorithmen vorgegangen.

2.3 Content-Management-Systeme & Enterprise-Content-Management

Content management is not just about sorting the content, collecting, and publishing but on effective content storage, recovery, and reuse. [29, S. 52]

2.3.1 Content-Management-Systeme

Ein Content-Management-System (CMS) ist eine Software zur Verwaltung von Inhalten jeglicher Art auf einer Webseite [30, S. 182]. Dabei können Inhalte auch ohne technische Kenntnisse erstellt und bearbeitet werden [31, S. 95]. Die Inhalte werden in einem Datenspeicher, der entweder direkt im CMS integriert ist, sich auf demselben Server befindet oder auf einem anderen Server gehostet ist, gespeichert [32, S. 6]. Das Content-Management-System generiert daraus den entsprechenden HTML-Code, welcher dann im Browser anzeigt wird [29, S. 55]. Die Kategorie der Content-Management-Systeme ist sehr umfangreich und umfasst unter anderem statische Seitengeneratoren, Blogs, Onlineshops und Foren [33, S. 89].

Die meisten Content-Management-Systeme bieten in mehr oder weniger ausgeprägter Form folgende Funktionen an [32, S. 10, 34, S. 10, 35, S. 1]:

- Inhalte können einfach erstellt und bearbeitet werden
- Speichern unterschiedlicher Versionen eines Inhalts
- Verwaltung von Menüs und Menüeinträgen
- Suchfunktion für Inhalte
- Inhalte können mit Schlagwörtern versehen werden

- Benutzerverwaltung mit unterschiedlichen Rollen und Berechtigungen (wer darf welche Inhalte sehen und/oder bearbeiten)
- Eine zentrale Administrationsoberfläche für alle Funktionen des Systems
- Erweiterbarkeit durch Plugins
- Design des CMS kann durch Templates geändert werden
- Assistent zur einfachen Installation des Content-Management-Systems
- Funktionen zur Aktualisierung des Content-Management-Systems und der installierten Plugins und Templates

2.3.2 Inhalte von Content-Management-Systemen

Deane Barker definiert Inhalte im Kontext von Content-Management als Informationen, die zur Publikation an Dritte entsprechend aufbereitet werden. Diesen Prozess bezeichnet er als „editorial process" [32, S. 5]. Damit ist beispielsweise die Erstellung von Nachrichtenartikeln gemeint, bei der aus demselben Sachverhalt von zwei unterschiedlichen Teams unterschiedliche Inhalte entstehen können [32, S. 3]. Für *Mauthe et al.* ist Inhalt jede Information, die als Text oder auch in einem beliebigen Ton-, Bild- oder Videoformat vorliegt [36, S. 4]. Zusätzlich können in Content-Management-Systemen Metadaten gespeichert und verwaltet werden. Metadaten sind Daten über Daten. In Content-Management-Systemen werden Metadaten von anderen Attributen nicht explizit unterschieden, sind aber alle Daten, die zur Verwaltung des Inhalts dienen [32, S. 92]. Das kann etwa der Titel eines Inhalts oder der Autor sein [37, S. 3].

Ein wichtiges Merkmal von Content-Management-Systemen ist die Trennung von Inhalt und Darstellung. Genauer betrachtet unterscheidet man hier zwischen *Content*, *Structure*, *View* und *Presentation*. Strukturen können aus Komponenten und Containern zusammengesetzt werden. Dabei entspricht ein Container einer Sammlung von Komponenten, welche zusammen präsentiert werden. Komponenten hingegen entsprechen den konkreten Datenobjekten, die zusammen den Inhalt einer Webseite bilden. Als Beispiel kann der Lebenslauf einer Person betrachtet werden, der sich aus unterschiedlichen persönlichen Angaben zusammensetzt. Mit diesem Prinzip lassen sich komplexe Strukturen mit beliebiger Tiefe erstellen. *Views* bestimmen, welche Daten der Struktur auf einer Webseite angezeigt werden, die *Presentation* in welcher Form. [38, S. 155f.]

Es kann unterschieden werden zwischen kommerziellen Content-Management-Systemen und solchen, die unter einer freien Lizenz wie der GNU General Public License veröffentlicht werden. Diese erlaubt die uneingeschränkte Benützung der Software sowie auch Änderungen am Quellcode der Software. Werden diese Änderungen jedoch an Dritte weitergegeben, so muss dies ebenfalls

unter der GNU General Public License erfolgen. [39] Damit können sowohl das System als auch meist eine Vielzahl von Erweiterungen und Designs kostenlos verwendet werden. Bei weit verbreiteten Content-Management-Systemen kann man davon ausgehen, dass Sicherheitslücken schnell behoben werden. Bei kommerziellen Systemen besteht im Fall eines Konkurses des Herstellers die Gefahr, dass der Support eingestellt wird und Sicherheitslücken nicht mehr behoben werden. [29, 55f] Als Beispiele für Content-Management-Systeme unter einer freien Lizenz werden in der Literatur WordPress, Joomla und Drupal genannt. [29, S. 56]

2.3.3 Enterprise-Content-Management-Systeme

Enterprise-Content-Management-Systeme sind eine Untergruppe der Content-Management-Systeme, welche nicht auf die öffentliche Verbreitung von Inhalten abzielen [32, S. 7]. Stattdessen werden in Enterprise-Content-Management-Systemen unternehmensinterne Inhalte gespeichert und verwaltet [32, S. 9]. Das können etwa Lebensläufe von Mitarbeiter/innen oder Unfallberichte sein [32, S. 7]. In produzierenden Unternehmen können etwa Produktinformationen gespeichert werden [40, S. 447]. Enterprise-Content-Management-Systeme können zudem auch eingesetzt werden, um Erfahrungswissen zu speichern. Dazu werden zu den Aufgaben von dem/der ausführenden Mitarbeiter/in Informationen wie „Lessons learned", „Dos and Don'ts" oder „Best practices" im System gespeichert [40, S. 449].

Für Enterprise-Content-Management bestimmte Systeme sind nicht scharf von anderen Content-Management-Systemen abgegrenzt. Hinter dem Wort „Enterprise" steht keine allgemeingültige Definition, man kann es jedoch als „für große Organisationen bestimmt" verstehen [32, S. 9]. Enterprise-Content-Management ist ebenfalls nicht eindeutig definiert [41, S. 269]. 2012 schlugen *Knut R. Grahlmann et al.* auf Basis unterschiedlicher Literatur folgende, vereinigte Definition vor: *„Enterprise Content Management comprises the strategies, processes, methods, systems, and technologies that are necessary for capturing, creating, managing, using, publishing, storing, preserving, and disposing content within and between organizations."* [41, S. 272] Üblicherweise sind in Enterprise-Content-Management-Systemen Funktionen zur Zusammenarbeit mehrerer Personen, zur Verwaltung von Inhalten und zur Dateiverwaltung enthalten [32, S. 7].

2.3.4 Auswahl eines Content-Management-Systems

Methodik

Ziel dieser Arbeit ist die Integration eines Systems für Case-Based Reasoning in ein Content-Management-System. Die Auswahl des Content-Management-Systems erfolgt zweistufig. Zuerst werden anhand von Installationsstatistiken die am häufigsten verwendeten Content-Management-Systeme ermittelt. Von diesen werden die drei am weitesten verbreiteten Content-Management-

Systeme, die unter einer freien Lizenz verfügbar sind, genauer untersucht. Dazu wird das System installiert und untersucht, ob und wie die erforderlichen Anwendungsfälle umgesetzt werden können.

Dabei wird von folgendem Szenario ausgegangen: Das CMS wird in einem Unternehmen für Enterprise-Content-Management eingesetzt. Das Unternehmen ist im Softwarebereich tätig und betreut unterschiedliche Projekte. Im Content-Management-System werden Projekte gespeichert und verwaltet. Dazu werden zu den Projekten folgende Informationen gespeichert:

- Projekttitel und Beschreibung
- Budget
- Projektbeginn und geschätzte Projektdauer in Monaten
- Beteiligte Mitarbeiter/innen
- Dokumente wie Projektauftrag, Projektplan, Formulare, Checklisten und Protokolle
- Zu erstellende Artefakte wie Apps, Weboberfläche und Backend
- Links zu externen Tools, wie zum Beispiel zur Ticketverwaltung

Für jede/n Mitarbeiter/in soll Folgendes gespeichert werden:

- Name
- Datum des Eintritts in das Unternehmen
- Höchster akademischer Grad
- Für verschiedene Programmiersprachen kann ausgewählt werden, ob er/sie damit
    - Keine Kenntnisse hat
    - Einsteiger ist (weniger als 1 Jahr Berufserfahrung)
    - Gute Kenntnisse hat (1 bis 3 Jahre Berufserfahrung)
    - Sehr gute Kenntnisse hat (mehr als 3 Jahre Berufserfahrung)
- Ein Textfeld oder Tags für weitere Programmiersprachen
- Ein Textfeld oder Tags für weitere Skills, wie beispielweise Erfahrungen in Projektmanagement.

Zu jedem Artefakt soll gespeichert werden können:

- Titel
- Beschreibung
- Eingesetzte Frameworks
- Eingesetzte Programmiersprachen

Gemäß der Webseite w3techs.com nutzen im Februar 2022 66,4 % aller Webseiten ein CMS (siehe Abbildung 8). Davon wiederum nutzen 65,3 % das freie Content-Management-System WordPress[5], welches damit klarer Marktführer ist. 6,7 % der Webseiten, welche ein CMS einsetzen, nutzen die proprietäre Software Spotify[6]. Das am dritthäufigsten eingesetzte Content-Management-System ist mit einem Marktanteil von 2,9 % der proprietäre Webseitenbaukasten Wix[7]. Darauf folgt mit einem Marktanteil von 2,7 % Squarespace[8], welches ebenfalls ein proprietärer Webseitenbaukasten ist. Danach folgt das unter der GNU General Public License stehende CMS Joomla mit einem Marktanteil von 2,5 %. Das ebenfalls unter der GNU General Public License stehende CMS Drupal erreicht einen Marktanteil von 2,0 % und ist damit das am sechsthäufigsten eingesetzte Content-Management-System. Danach folgen mit Adobe Systems[9], Blogger [10]und Bitrix [11] proprietärere Lösungen. Das zehntmeist genutzte Content-Management-System ist die freie E-Commerce-Plattform OpenCart[12]. [42]

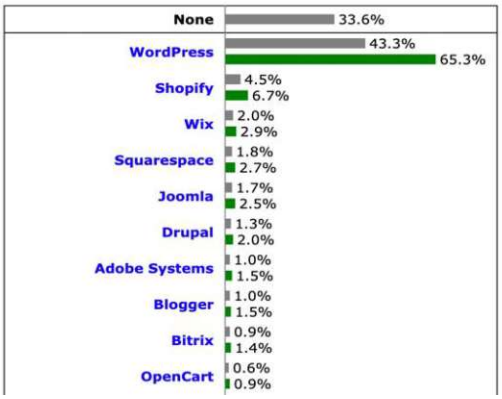

Abbildung 8: Marktanteile der am meisten genutzten Content-Management-Systeme gemäß w3techs.com im Februar 2022. Abbildung übernommen von [42].

Top In CMS Usage Distribution in the Top 1 Million Sites

Technology	Websites	%
WordPress	371,433	37.14
Drupal	34,995	3.5
Google Search Appliance	29,780	2.98
WP Engine	28,746	2.87
Plesk	28,651	2.87
My Salesforce	28,030	2.8
CPanel	23,530	2.35
Atlassian Cloud	18,487	1.85
Joomla!	17,839	1.78
HubSpot CMS Hub	17,119	1.71

Abbildung 9: Marktanteile der am meisten genutzten Content-Management-Systeme gemäß builtwith.com im Februar 2022. Abbildung übernommen von [43].

Die Webseite builtwith.com wertet die Nutzung des eingesetzten Content-Management-Systems von 1 Million der meistbesuchten Webseiten aus (siehe Abbildung 9). Davon verwenden 37,14 %

[5] https://wordpress.org/, abgerufen am 22.02.2022
[6] https://www.shopify.de/preise, abgerufen am 22.02.2022
[7] https://de.wix.com/upgrade/website, abgerufen am 22.02.2022
[8] https://de.squarespace.com/preise, abgerufen am 22.02.2022
[9] https://business.adobe.com/at/products/experience-manager/sites/web-content-management.html, abgerufen am 22.02.2022
[10] https://www.blogger.com, abgerufen am 22.02.2022
[11] https://www.bitrix24.de/prices, abgerufen am 22.02.2022
[12] https://www.opencart.com, abgerufen am 22.02.2022

WordPress, 3,5 % Drupal und 1,78 % Joomla. Zusätzlich nutzen noch 2,87 % eine spezielle Version von WordPress, welche vom Anbieter „WP Engine“ gehostet wird. Alle anderen der laut builtwith.com zehn meistgenutzten Content-Management-Systeme sind keine Open-Source-Lösungen. [43]

Damit sind WordPress, Drupal und Joomla die drei am weitesten verbreiteten Open-Source-Content-Management-Systeme. Sie werden im Folgenden vorgestellt und näher betrachtet.

WordPress

WordPress ist das derzeit am weitesten verbreitete Content-Management-System [42, 43]. Es wurde 2003 von Matt Mullenweg und Mike Little ins Leben gerufen und steht von Beginn an unter der GNU General Public License 2.0. Seit 2010 liegen die Rechte der Marke bei der damals gegründeten WordPress Foundation. Der Einsatzbereich von WordPress ist vielfältig, von Privatpersonen ohne technische Kenntnisse, die damit ihre Webseite betreiben, bis hin zu Unternehmen, welche WordPress beispielsweise als E-Commerce-Lösung einsetzen. WordPress bietet eine Representational State Transfer (REST) API, womit es einfach in andere Systeme integriert werden kann. [33, S. 90]

WordPress ist zum größten Teil in der Programmiersprache PHP geschrieben, als Datenbank wird MySQL verwendet. Rund ein Drittel des Codes ist JavaScript, das im Frontend eine wichtige Rolle spielt. [33, S. 89] Eine große Stärke von WordPress ist die Erweiterbarkeit. WordPress kann durch Plugins um Funktionalität erweitert werden, mit Themes kann das Design der Webseite geändert werden. Aktuell können im offiziellen Verzeichnis auf wordpress.org knapp 60.000 Plugins[13] und fast 10.000 Themes[14] kostenlos heruntergeladen werden. Bei einer neuen Version von WordPress wird darauf geachtet, dass die Schnittstellen zu Modulen und Themes nicht verändert werden und diese somit ohne Anpassungen auch in der neuen Version nutzbar sind. [33, S. 89]

Das am Beginn des Abschnitts beschriebene Szenario kann mit WordPress wie folgt umgesetzt werden: WordPress unterstützt standardmäßig die Inhaltstypen „Posts“, „Pages“, „Media“, „Users“ und „Comments“. Zusätzlich gibt es noch sogenannte Taxonomien, welche standardmäßig „Categories“ und „Tags“ sind und welche den Inhalten flexibel zugeordnet werden können. Mit kostenlosen sowie kostenpflichtigen Plugins können jedoch eigene Inhaltstypen angelegt werden, WordPress bietet dafür alle notwendigen Schnittstellen. Es können auch eigene Inhaltstypen

[13] https://wordpress.org/plugins/, abgerufen am 27.02.2022
[14] https://wordpress.org/themes/, abgerufen am 27.02.2022

per Programmcode hinzugefügt werden. Da in dieser Arbeit jedoch ein System entworfen werden soll, bei dem auch Benutzer/innen ohne Programmierkenntnisse neue Inhaltstypen hinzufügen und konfigurieren können sollen, wird darauf nicht weiter eingegangen.

Für die Umsetzung des Szenarios werden die Plugins „Pods[15]“ und „Advanced Custom Fields[16]“ untersucht. „Advanced Custom Fields“ bietet eine kostenlose Version an, die jedoch einen beschränkten Funktionsumfang aufweist: So können etwa nur zu bestehenden Inhaltstypen Felder hinzugefügt und geändert werden. Zwar können Inhaltstypen über andere kostenlose Plugins hinzugefügt werden und diese dann mit „Advanced Custom Fields“ mit Feldern ergänzt werden, allerdings erhöht dies die Komplexität der Konfiguration. Aus diesem Grund wird das Szenario mit dem Plugin „Pods“ umgesetzt. „Pods“ steht vollständig unter der GNU General Public License, Version 2. Eine Trennung in eine kostenlose und kostenpflichtige Version gibt es nicht, der vollständige Funktionsumfang lässt sich kostenlos nutzen.

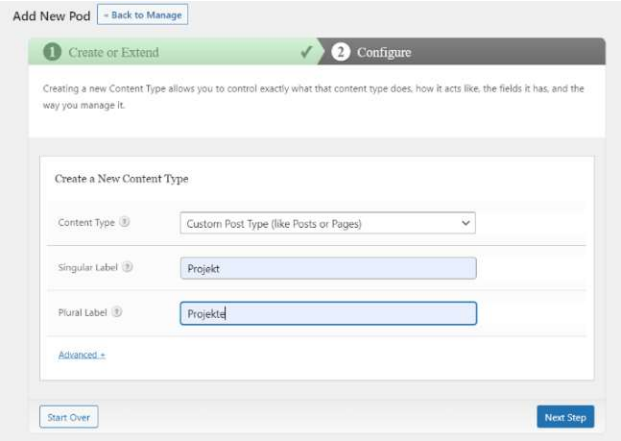

Abbildung 10: Erstellung eines neuen Inhaltstyps mit dem Plugin „Pods“. Eigene Aufnahme.

„Pods“ erlaubt das Anlegen neuer Inhaltstypen sowie das Ergänzen von Inhaltstypen um eigene Felder. Zur Umsetzung des Szenarios werden die Inhaltstypen „Projekte“, „Mitarbeiter/innen“, „Artefakte“ und „Links“ angelegt. Wie in Abbildung 10 zu sehen ist, muss dazu ein Name für den Inhaltstyp im Singular und Plural angegeben werden. Danach können Felder hinzugefügt werden, dazu kann aus verschiedenen Feldtypen gewählt werden, wie etwa „Plain Text“, „Website“, „Phone“, „Date“, „Currency“, „File / Image / Video“. Zusätzlich erlaubt der Feldtyp „Relationship“ den Verweis auf einen oder mehrere Inhalte, Taxonomien oder WordPress-Objekte wie etwa Dateien. „Relationships“ können bidirektional sein. Dies ermöglicht etwa im umgesetzten Anwendungsfall, dass sowohl wenn ein/e Mitarbeiter/in bearbeitet wird, ihm/ihr ein oder mehrere Projekte zugeordnet werden können als auch, dass wenn ein Projekt bearbeitet wird, ein/e oder mehrere Mitarbeiter/innen zugeordnet werden können. Jedes Feld, mit Ausnahme von „Relationships“ und „File / Image / Video“, erlaubt die Speicherung von nur einem Wert. Sollen in einem Feld mehrere

[15] https://pods.io/, abgerufen am 04.03.2022
[16] https://www.advancedcustomfields.com/, abgerufen am 04.03.2022

Werte gespeichert werden, so kann für diese ein eigener Inhaltstyp angelegt werden und auf diesen referenziert werden. Auch das Userinterface unterstützt diesen Anwendungsfall, so können etwa beim Anlegen eines neuen Projekts über den Button „Add New“ neue Inhalte vom Typ „Links“ angelegt werden, ohne dass in ein neues Formular gewechselt werden muss (siehe Abbildung 11). Der neue Inhalt kann in einem Overlay angelegt werden und wird nach dem Speichern automatisch dem Relationship-Feld hinzugefügt.

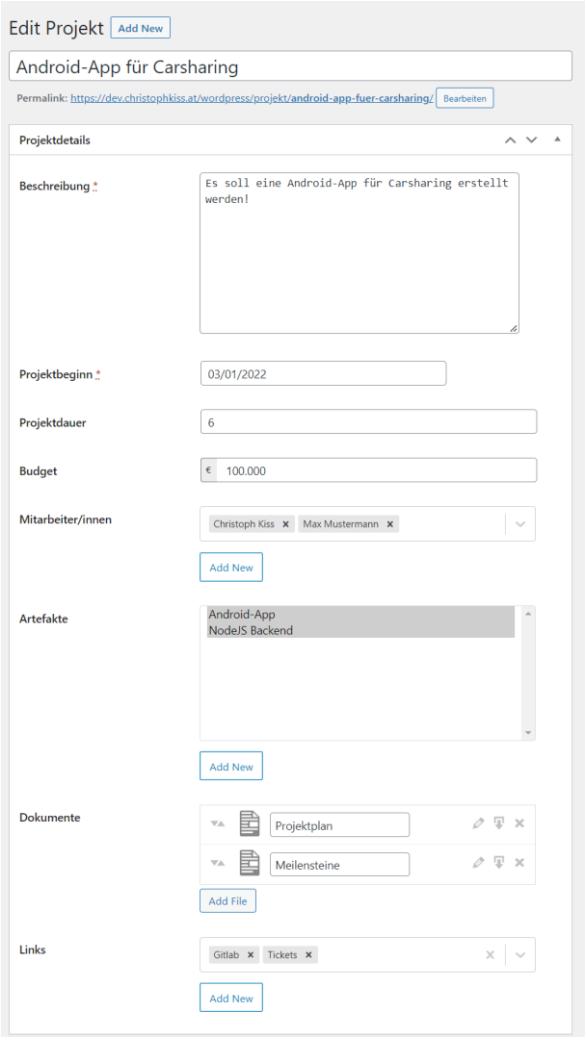

Abbildung 11: Formular zur Erstellung eines neuen Inhalts vom Typ „Projekt“. Über den „Add New“-Button können neue Mitarbeiter/innen oder Links erstellt werden, ohne das Formular zu verlassen. Eigene Aufnahme.

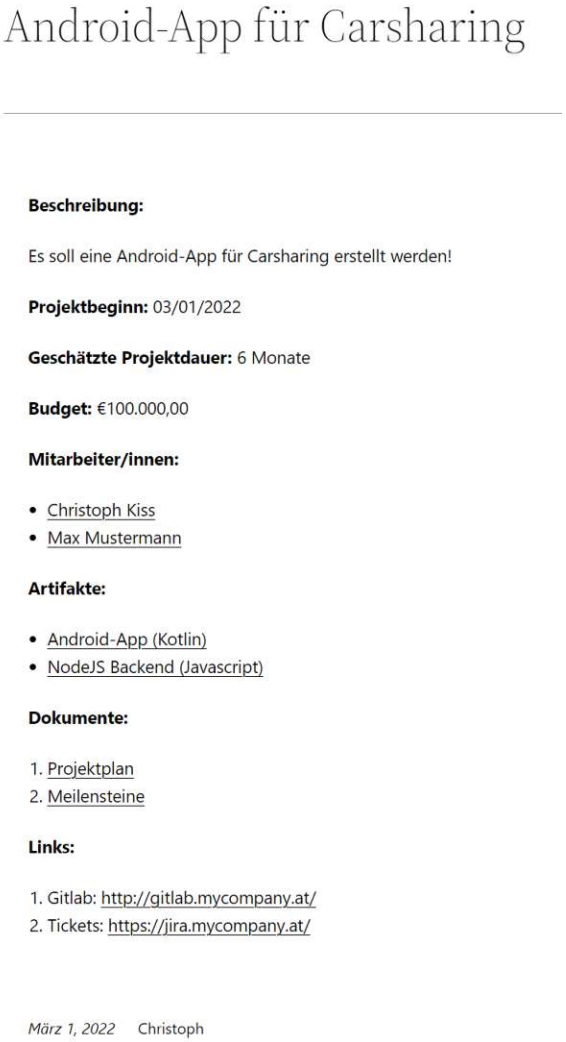

Abbildung 12: Ausgabe eines Inhalts vom Typ „Projekt“. Zur Formatierung wird der in Listing 1 gezeigte Template-Code verwendet. Eigene Aufnahme.

Während das Formular zur Erstellung neuer Inhalte von über „Pods“ konfigurierten Inhaltstypen automatisch erstellt wird (siehe Abbildung 11), wird nach dem Speichern eines neuen Inhalts von diesem zunächst nur das „Titel“-Feld und weitere, von WordPress vordefinierte Felder, wie

Erstellungsdatum und Autor, angezeigt. Felder, welche über Pods erstellt werden, werden hingegen nicht angezeigt. Um den Inhalt dieser Felder anzeigen zu können, muss in „Pods“ ein neues Template erstellt und mit dem Inhaltstyp verknüpft werden. Dabei können zwei Templates per Inhaltstyp verknüpft werden. Ein Template dient zur Anzeige des Inhalts, wenn dieser über die eigene URL aufgerufen wird. Das zweite Template wird verwendet, wenn der Inhalt in Listen angezeigt wird, wie zum Beispiel auf der Kategorieseite oder in den Suchergebnissen. Templates ermöglichen mit Hilfe von HTML und einer eigenen, einfachen Syntax die Ausgabe und Formatierung der in Feldern und referenzierten Feldern gespeicherten Werte (siehe Listing 1 sowie Abbildung 12).

```
<p><b>Beschreibung:</b> {@beschreibung}</p>
<p><b>Projektbeginn:</b> {@projektbeginn}</p>
<p><b>Geschätzte Projektdauer:</b> {@projektdauer} Monate</p>
<p><b>Budget:</b> {@budget}</p>
<p><b>Mitarbeiter/innen:</b></p>
<ul>
[each mitarbeiter_innen]
    <li><a href="{@permalink,esc_url}">{@vorname} {@nachname}</a></li>
[/each]
</ul>
<p><b>Artifakte:</b></p>
<ul>
[each artefakte]
    <li><a href="{@permalink,esc_url}">{@post_title} ({@eingesetzte_programmierspra-
chen})</a></li>
[/each]
</ul>
<p><b>Dokumente:</b></p>
<ol>
[each dokumente]
    <li><a href="{@_src}">{@post_title}</a></li>
[/each]
</ol>
<p><b>Links:</b></p>
<ol>
[each links]
    <li>{@post_title}: <a href="{@url}">{@url}</a></li>
[/each]
</ol>
```

Listing 1: Formatierung des Inhaltstyps „Projekt“ mit HTML und den „Templates“ des WordPress-Plugins „Pods“.

DRUPAL

Drupal ist, je nach Quelle, das aktuell zweithäufigste oder dritthäufigste eingesetzte Content-Management-System [42, 43]. Drupal wurde 2001 von Dries Buytaert veröffentlicht und ist damit ein wenig älter als WordPress (2003) und Joomla (2005) [34, S. 11]. Die aktuelle Version ist Drupal 9.3.7[17]. Drupal verwendet die Lizenz „GNU General Public License, Version 2“. Auch von Dritten geschriebene Module und Themes müssen mit einer General Public License veröffentlicht werden, wenn sich der/die Autor/in zur Veröffentlichung entscheidet. Es besteht keine Pflicht zur

[17] https://www.drupal.org/project/drupal/releases/9.3.7, abgerufen am 12.03.2022

Veröffentlichung des Codes von Modulen und Themes, wenn diese nur in eigenen Projekten eingesetzt werden. [44]

Drupal besteht aus einem „Core“, welcher Basisfunktionen zur Verfügung stellt. Dazu zählen Funktionen zum Erstellen und Bearbeiten von Inhalten, Dateien, Menüs, Benutzer/innen und Taxonomien. [45, S. 2] Drupal kann durch Module und Themes von Dritten erweitert werden. Aktuell können 7.925 Module[18] und 337 Themes[19] für Drupal 9 von drupal.org heruntergeladen werden.

Jede mit Drupal umgesetzte Webseite baut auf den folgenden Komponenten auf: Entities, Taxonomien und Views. Entities sind die Basis für alle Inhalte, Taxonomien und Benutzer/innen. In Drupal können beliebige Inhaltstypen, auch „Content Entities“ genannt, angelegt werden. Das können zum Beispiel Blogposts, Produkte oder Pressemitteilungen sein. Diese können aus verschiedenen Feldern bestehen, wie etwa dem Body, einem Vorschaubild, dem Datum der Veröffentlichung und dem/der Autor/in. [45, S. 36ff.]

Neben den Inhaltstypen spielen Taxonomien, auch „Taxonomy Entities“ genannt, eine zentrale Rolle. Taxonomien können zur Kategorisierung von Inhalten verwendet werden. Das heißt, es kann etwa eine Taxonomie „Artikeltyp“ angelegt werden, um „News“ von anderen Artikeln unterscheiden zu können. [45, S. 38ff.] Taxonomien können auch strukturiert werden. So kann etwa eine Taxonomie „Sportteams“ erstellt werden und auf der ersten Ebene können Sportarten wie Fußball, Basketball und so weiter angelegt werden. Jeder Sportart können dann auf der zweiten Ebene Teams zugeordnet werden. [46, S. 36] Es können beliebig viele verschiedene Taxonomien angelegt werden. Auch Taxonomien können, wie Inhaltstypen, durch das Hinzufügen von Feldern beliebig angepasst werden. [45, S. 38f.]

Zur Anzeige von Entities kommen in Drupal „Views“ zum Einsatz. Nach dem Erstellen einer „View“ kann über die Administrationsoberfläche ausgewählt werden, welche Entities angezeigt werden sollen und welche Elemente einer jeden Entity angezeigt werden sollen. Somit können selbst komplexe Abfragen, ohne eine einzige Zeile Code zu schreiben, umgesetzt werden. Views können entweder über eine eigene URL aufgerufen werden oder in andere Seiten integriert werden. [45, S. 40]

Eine Besonderheit von Drupal war lange, dass mit jedem Update der Version (zum Beispiel von Drupal 7 auf Drupal 8) die Schnittstellen für Module geändert wurden [33, S. 90]. Dadurch mussten

[18] https://www.drupal.org/project/project_module, abgerufen am 12.03.2022
[19] https://www.drupal.org/project/project_theme, abgerufen am 12.03.2022

bei einer neuen Version stets alle Module angepasst oder teilweise sogar neu geschrieben werden. Seit der Version 8 ist dies bei neuen Major-Releases nicht mehr der Fall. Daher können alle Module und Themes von Drupal 8, welche keine als „deprecated“ gekennzeichneten Schnittstellen verwenden, mit Drupal 9 verwendet werden. Änderungen an den Schnittstellen zwischen Drupal und Modulen und Themes werden wie folgt vorgenommen. Zunächst wird in einer Minor-Version (zum Beispiel Drupal 9.1) die neue API parallel zur bestehenden API eingeführt. Die bestehende API wird als „deprecated“ gekennzeichnet und beim Wechsel auf die neue Major-Version (zum Beispiel Drupal 10) entfernt. [47]

Ab Version 8 hat sich der Fokus von Drupal vom Einsatz in kleinen und mittleren Organisationen zu großen Unternehmen und Organisationen verschoben [35, S. 2]. Auch in technischer Hinsicht brachte der Wechsel von Drupal 7 zu Drupal 8 große Veränderungen mit sich. So wurde Drupal 8 auf Basis des Symphony Webframeworks objektorientiert neu entwickelt. Im Vergleich zu der bis Drupal 7 verwendeten prozeduralen Programmierung soll dadurch eine modularere und damit besser wartbare Codebasis für Drupal geschrieben werden. Weitere wichtige Änderungen waren die Einführung der Trennung zwischen Inhalt und Konfiguration sowie die vollständige Einführung der Entity-API. Zudem wurde das „Views“-Modul Teil des Drupal-Cores. [48, 49]

Das am Beginn des Abschnitts beschriebene Szenario kann in Drupal 9 wie folgt umgesetzt werden: Unter „Struktur“ werden die Inhaltstypen „Projekt“, „Mitarbeiter/in“ und „Artefakt“ angelegt. Im Anschluss werden den Inhaltstypen die entsprechenden Attribute als Felder hinzugefügt. Dabei kann aus verschiedenen Datentypen wie „Boolean“, „Datum“, „E-Mail“ sowie unterschiedlichen Zahlen- und Textformen gewählt werden. Zusätzlich können Felder angelegt werden, die Referenzen auf andere Inhalte speichern. Bidirektionale Referenzen sind in Drupal 9 mit den standardmäßig vorhandenen Modulen nicht möglich. Bei jedem Feld kann dabei ausgewählt werden, ob eine begrenzte Anzahl an Werten (zum Beispiel ein Wert) oder eine unbegrenzte Anzahl an Werten gespeichert wird. Im Szenario lassen sich so beliebig viele Links und Dateien direkt zu einem Projekt hinzufügen.

Drupal erstellt sowohl das Formular zum Erstellen von Inhalten (siehe Abbildung 14) als auch ein Template zum Anzeigen der Inhalte (siehe Abbildung 13). Im Vergleich zu WordPress ist damit kein Schreiben eines Codes zur Anzeige nötig.

Abbildung 13: Anzeige eines Inhalts vom Typ „Projekt“ ohne angepasstes Template. Eigene Aufnahme.

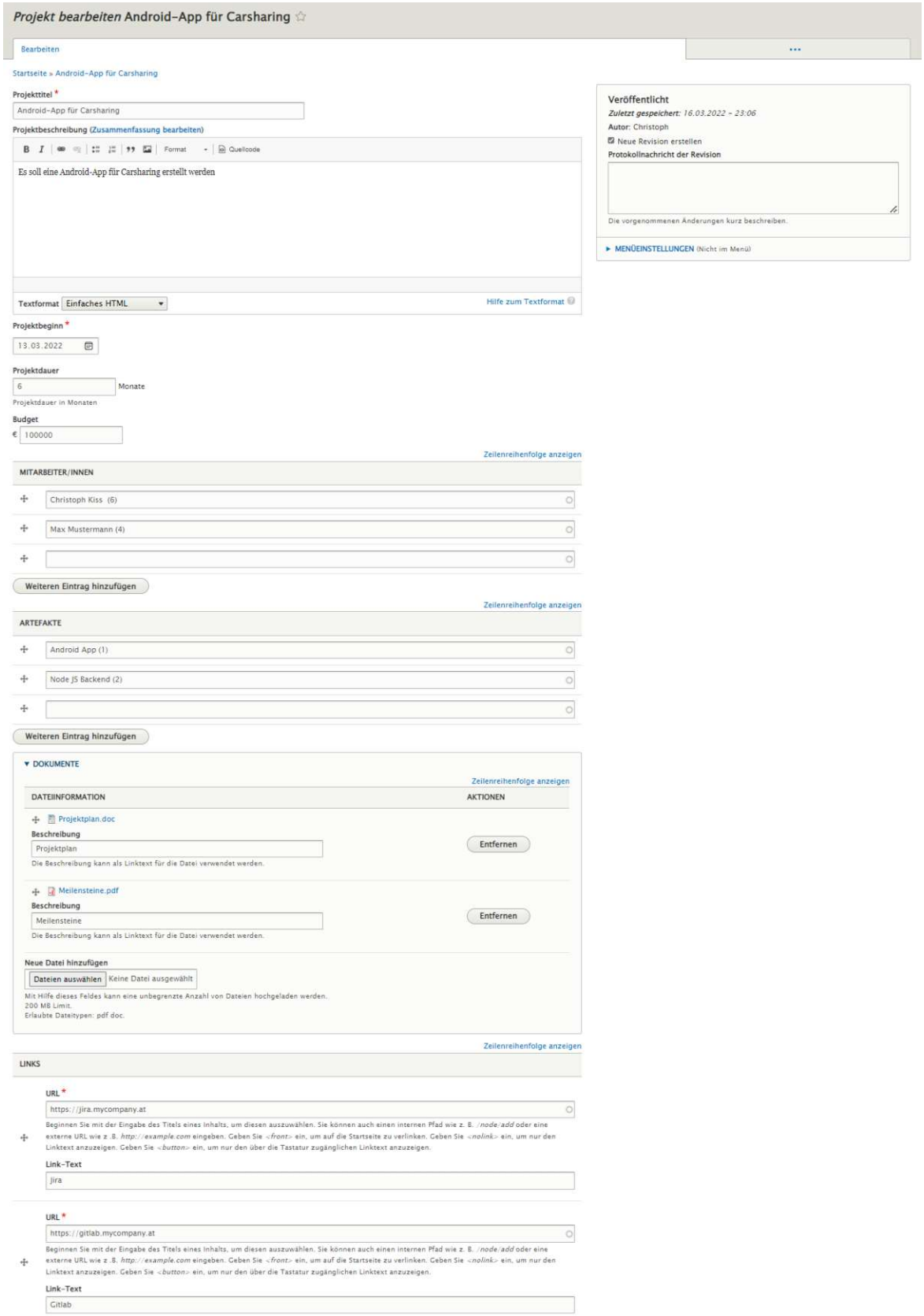

Abbildung 14: Formular zum Erstellen eines Inhalts vom Typ "Projekt" in Drupal 9. Eigene Aufnahme.

Allerdings sind sowohl das Erstellungs- und Bearbeitungsformular als auch die Darstellung der Inhalte optisch nicht so ansprechend gestaltet wie bei WordPress. Ein Überschreiben des Templates kann über Themes umgesetzt werden, wird aber in diesem Anwendungsfall nicht weiterverfolgt. Somit kann das Szenario mit ausschließlich im Drupal-Core enthaltenen Modulen umgesetzt werden. Außerdem muss kein Code geschrieben werden. Dies erlaubt es auch Anwender/innen ohne Programmierkenntnisse, Inhaltstypen hinzuzufügen oder zu bearbeiten.

JOOMLA

Joomla ist laut w3techs.com das am zweithäufigsten eingesetzte Open-Source-Content-Management-System [42]. In der Statistik von builtwith.com liegt es hinter Drupal auf dem dritten Platz [43]. Aus diesem Grund wird Joomla nun genauer untersucht und das schon in WordPress und Drupal umgesetzte Szenario mit Joomla realisiert.

Joomla entstand 2005 durch einen Fork des Open-Source-Content-Management-Systems Mambo [34, S. 11]. Analog zu WordPress und Drupal ist Joomla in PHP programmiert [32, S. 22]. Joomla ist vollständig objektorientiert umgesetzt [50, S. 5] und verwendet das Model-View-Controller Softwaredesignpattern [51]. Der Quellcode unterliegt der GNU General Public License Version 2[20]. Die neueste Version von Joomla ist 4.1.0[21].

Joomla lässt sich mit „Extensions“ erweitern. Dabei wird zwischen Modulen, Komponenten und Plugins unterschieden. Module werden zum Rendern einer Seite eingesetzt, sie können Daten von Komponenten anzeigen. Module müssen jedoch nicht zwingend an Komponenten gekoppelt sein, sondern können auch Text oder statisches HTML anzeigen. [52] Komponenten sind komplexer als Module und bestehen meist aus einem Administratoren- und einem Seitenbestandteil. Beispiele sind etwa die mit Joomla ausgelieferten Komponenten „Inhaltsverwaltung“ und „Kontaktformulare“. [51] Plugins enthalten einen Code, welcher bei bestimmten Events durch Joomla ausgeführt wird. Templates verändern das Erscheinungsbild einer mit Joomla umgesetzten Webseite. [52] Aktuell stehen auf joomla.org 5.924[22] Extensions zum Download zur Verfügung. Auf joomla.org können keine Themes heruntergeladen werden, es gibt jedoch verschiedene Anbieter für kostenlose sowie kostenpflichtige Joomla Themes.

Joomla unterstützt in der Standardinstallation zwar das Anlegen von Feldern, allerdings können keine Inhaltstypen angelegt werden. Um das zu Beginn des Abschnitts beschriebene Szenario

[20] https://tm.joomla.org/joomla-license-faq.html, abgerufen am 16.03.2022
[21] https://downloads.joomla.org/, abgerufen am 16.03.2022
[22] https://extensions.joomla.org/, abgerufen am 16.03.2022

umzusetzen, wurde daher im „Joomla! Extensions Directory“ nach dem Tag „Content Construction[23]“ gefiltert. Die an erster Stelle gelistete Extension „SP Page Builder[24]“ wurde testweise installiert, kann aber für die Umsetzung des Szenarios nicht jedoch verwendet werden, da mit ihr zwar einzelne Seiten des Content-Management-Systems gestaltet, jedoch keine Inhaltstypen angelegt werden können. Die an zweiter Stelle gelistete Extension „FLEXIcontent[25]“ wird ebenfalls evaluiert. Mit dieser Extension wird das Szenario wie folgt umgesetzt:

FLEXIcontent ermöglicht das Anlegen von Inhaltstypen im Administrationsbereich von Joomla. Dort werden die Inhaltstypen „Projekt“, „Mitarbeiter/in“ und „Artefakt“ angelegt. Für jeden Inhaltstyp können für sogenannte „Core fields“, also Felder, die Joomla zur Verfügung stellt, eine Bezeichnung und weitere Einstellungen gesetzt werden. So lässt sich etwa der Projekttitel mit dem „Core field“ Titel umsetzen und die Projektbeschreibung mit dem „Core field“ Beschreibung. Zusätzlich können Felder angelegt werden, die dann einem oder mehreren Inhaltstypen zugeordnet werden. Damit geht FLEXIcontent / Joomla den umgekehrten Weg von Pods / WordPress und Drupal, bei denen den Inhaltstypen Felder zugeordnet werden.

FLEXIcontent / Joomla erlaubt das Anlegen von Feldern unterschiedlichster Typen, darunter Textfelder, Checkboxen und Dropdown-Listenfelder. Die Daten eines Textfeldes können weiter über eine sogenannte „Value Validation Mask“ eingeschränkt werden. Diese erlaubt das Validieren des Inhalts mit unterschiedlichen Masken, darunter „URL“, „Email“, „Currency“, „Integer“, „Date“. Darüber hinaus kann eine serverseitige Validierung eingestellt werden, mit der beispielsweise HTML-Code auf bestimmte HTML-Tags beschränkt werden kann. Ähnlich wie bei Drupal kann für jedes Feld angegeben werden, ob ein Wert, eine bestimmte Anzahl von Werten oder eine beliebige Anzahl von Werten gespeichert werden soll.

Auch Relationen zu anderen Inhalten lassen sich erstellen. Dabei lässt sich einschränken, auf welche Inhaltstypen die Referenz erfolgen darf. Jedes Referenzfeld kann Referenzen zu beliebig vielen Inhalten speichern. Ebenso lässt sich eine „Reverse-Referenz“ einrichten, diese ist allerdings nicht bearbeitbar. Jedoch kann beispielsweise sehr einfach zu jedem/jeder Mitarbeiter/in angezeigt werden, in welchem Projekt er oder sie tätig ist.

---

[23] https://extensions.joomla.org/category/authoring-a-content/content-construction/, abgerufen am 16.03.2022
[24] https://extensions.joomla.org/extension/authoring-a-content/content-construction/sp-page-builder/, abgerufen am 16.03.2022
[25] https://extensions.joomla.org/extension/authoring-a-content/content-construction/flexicontent/, abgerufen am 16.03.2022

Die Benutzeroberfläche von FLEXIcontent erlaubt viel umfangreicher als jene von Drupal, das Aussehen eines benutzerdefinierten Inhalts zu konfigurieren. So können Felder unterschiedlichen Regionen/Blöcken zugeordnet werden, womit optisch sehr ansprechende Darstellungen von benutzerdefinierten Inhaltstypen ohne das Schreiben von Codes erstellt werden können (siehe Abbildung 15).

Abbildung 15: Inhaltsanzeige in Joomla. Felder können Blöcken und Tabs zugeordnet werden. Eigene Aufnahme.

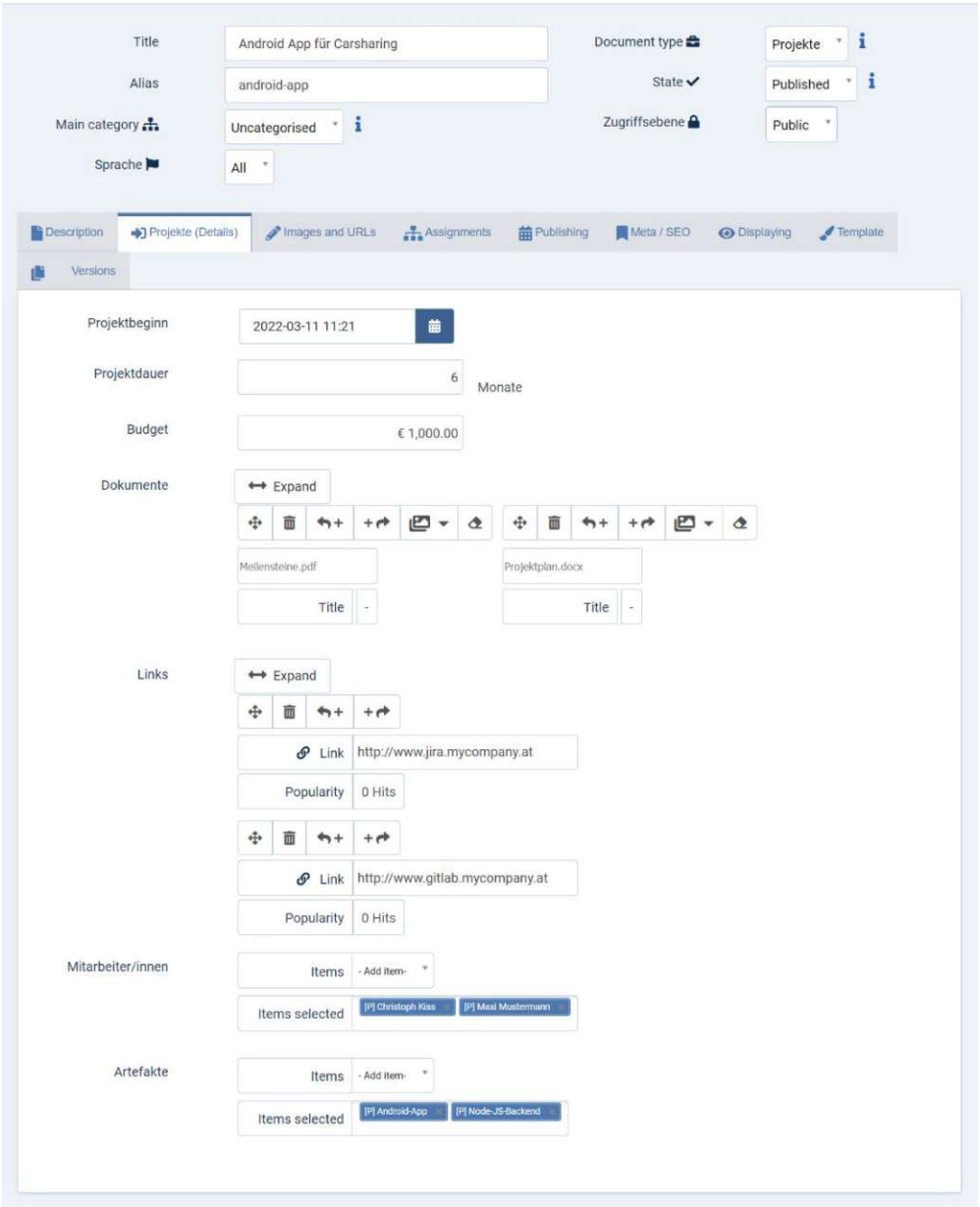

Abbildung 16: Formular zum Anlegen eines neuen Projektes in Joomla. Eigene Aufnahme.

Erstellen Benutzer/innen einen Inhalt eines über FLEXIcontent hinzugefügten Typs, so werden alle Felder, welche nicht „Core fields“ sind, in einem eigenen Reiter angezeigt (siehe Abbildung 16). Felder mit Referenzen erlauben kein Hinzufügen neuer Inhalte im Formular des referenzierten Inhaltstyps. Damit müssen im umgesetzten Szenario erst alle Mitarbeiter/innen und Artefakte angelegt werden, bevor diese einem Projekt hinzugefügt werden können.

Fazit Auswahl eines Content-Management-Systems

Für das zu Beginn des Abschnitts beschriebene Szenario einer internen Projekt- und Mitarbeiterverwaltung wird in Tabelle 1 für unterschiedliche Kriterien dargestellt, ob sich diese mit WordPress, Drupal und Joomla umsetzen lassen. Grundsätzlich lässt sich das Szenario mit allen drei Content-Management-Systemen umsetzen. Bei WordPress und Joomla sind dafür jedoch externe Module notwendig. Ein Nachteil von WordPress ist, dass benutzerdefinierte Inhalte nicht ohne das Schreiben von Template-Code angezeigt werden können. Bei Joomla hingegen können referenzierte Inhalte nicht direkt im Formular des Inhalts erstellt werden, von dem aus referenziert werden soll. Bei Drupal sind keine bidirektionalen Referenzen möglich. Dennoch bietet Drupal von allen verglichenen Content-Management-Systemen die meisten Vorteile. Natürlich muss für jeden Anwendungsfall individuell evaluiert werden, welches Content-Management-System das passendste ist. Drupal wird jedoch mit seiner Ausrichtung auf Enterprise-Content-Management in ähnlichen Evaluierungen oft am geeignetsten sein. Auch in der Literatur ist angemerkt, dass Drupal zwar als Web-Content-Management-System bekannt geworden ist, allerdings von Organisationen auch zur Verwaltung von internen Inhalten verwendet wird [32, S. 7]. Daher wird in dieser Arbeit die Integration von Case-Based Reasoning in das Content-Management-System Drupal umgesetzt.

	WordPress	**Drupal**	**Joomla**
Szenario umsetzbar	Ja	Ja	Ja
Szenario ohne externe Module umsetzbar	Nein	Ja	Nein
Inhalte können andere Inhalte referenzieren	Ja	Ja	Ja
Referenzierte Inhalte können direkt im Formular erstellt werden	Ja	Neutral (können angelegt werden, aber nur der Titel kann gesetzt werden)	Nein
Bidirektionale Referenzen sind möglich	Ja	Nein	Ja
Ein Feld kann mehrere Werte speichern	Nein	Ja	Ja
Benutzerdefinierte Inhalte können ohne Codeanpassungen angezeigt werden	Nein	Ja	Ja
Auflistungen benutzerdefinierter Inhalte lassen sich ohne Codeanpassungen erstellen	Nein	Ja	Nein

Tabelle 1: Auswahl des am besten geeigneten Content-Management-Systems.

3 UMSETZUNG

3.1 MOTIVATION

Damit Case-Based Reasoning in einem Unternehmen erfolgreich eingesetzt werden kann, ist es essenziell, dass die Wissensbasis stetig gepflegt und erweitert wird. Damit dies gelingt, soll es Benutzer/innen möglichst einfach gemacht werden, neue Fälle anzulegen, ähnliche Fälle zu finden und bestehende Lösungen zu ergänzen. Im besten Fall können schon im Unternehmen genutzte und gepflegte Datensätze verwendet werden. Mindestens genauso wichtig wie die Pflege der in den Fällen gespeicherten Daten ist die Modellierung der Struktur der Wissensbasis. Damit das Erstellen der Struktur der Wissensbasis, bestehend aus Inhaltstypen (in myCBR „Conzepts" genannt, in JColibri in Form von annotierten Java-Klassen umgesetzt) und deren Attributen auch durch Nicht-Programmierer/innen beziehungsweise technisch nicht versierte Benutzer/innen möglich ist, soll dies nicht nur vollständig über eine grafische Benutzeroberfläche erfolgen, sondern auch auf einem dem/der Benutzer/in schon bekannten Workflow aufbauen. Damit kann sowohl die Verwendung, als auch die Anpassung des Case-Based-Reasoning-Systems, von Benutzer/innen durchgeführt werden, die keinen technischen Hintergrund haben und über kein Detailwissen zu Case-Based Reasoning verfügen.

Bei der Analyse von JColibri hat sich gezeigt, dass hier davon ausgegangen wird, dass alle Anforderungen und Attribute für das CBR-System in der Entwurfsphase schon bekannt sind. Ein/e Programmierer/in setzt das System anschließend mit Hilfe der Entwicklungswerkzeuge, die in JColibri enthalten sind, um. Diese/r Entwickler/in programmiert, wie Benutzer/innen später Daten in das System eintragen und ändern sowie Ergebnisse des CBR-Prozesses abrufen können. Ist eine Änderung an der Struktur der Fallbasis nötig, kann dies zwar umgesetzt werden, benötigt jedoch erneut eine/n Programmierer/in, welche eine neue Version des CBR-Systems erstellen und deployen muss. Dies ist aufwendig und unflexibel. Da der Nutzen von CBR-Systemen erst mit einer großen Fallbasis und damit längerem Einsatz zum Tragen kommt, ist es wahrscheinlich, dass sich die Anforderungen an die Attribute und deren Gewichtung in diesem Zeitraum ändern, was wiederum entweder eine/n interne/n oder eine/n externe/n Programmierer/in erfordert. Diese/r sollte zudem Erfahrung mit CBR haben, was teuer und in vielen Unternehmen nicht vorhanden ist.

Der myCBR Workbench hingegen erlaubt es auch Benutzer/innen ohne Programmiererfahrung, die Struktur der Fallbasis zu ändern. Hier besteht jedoch das Problem, dass die Pflege der Fallbasis beziehungsweise das Hinzufügen neuer Fälle sowie das Finden ähnlicher Fälle mit dem myCBR

Workbench komplex ist, da das Userinterface zur Nutzung des CBR-Systems nicht klar von jener zur Konfiguration des Systems abgetrennt ist. Natürlich können Programmierer/innen mit Hilfe des Software-Development-Kits eine eigene Benutzeroberfläche für das Erstellen und Verwalten von Fällen schreiben. Allerdings geht dann der Vorteil verloren, dass die Struktur der Falldatenbank ohne Programmierkenntnisse geändert werden kann, womit die für JColibri beschriebene Problematik eintritt.

Daher soll ein Modul für ein Content-Management-System entwickelt werden, das es Benutzer/innen erlaubt, in ihrem bestehenden Workflow zur Erstellung und Bearbeitung von Inhalten Fälle anzulegen und zu bearbeiten. Ebenso soll es Benutzer/innen mit höheren Rechten möglich sein, die Struktur der Fallbasis zu erstellen und zu ändern. Damit soll die Trennung von Anwendung und Konfiguration des Systems für Case-Based Reasoning einerseits aufgehoben werden, andererseits soll durch ein Rechtesystem die Konfiguration nicht jedem/jeder Anwender/in zugänglich sein.

3.2 Anforderungen

Der Prototyp soll folgende Anforderungen erfüllen:

A1: Es soll möglich sein, vorhandene Inhaltstypen als Speicher für die Fallbasis und zur Ermittlung ähnlicher Fälle mittels Case-Based Reasoning zu verwenden.

A2: Inhaltstypen, die für Case-Based Reasoning genutzt werden, können eine beliebige Anzahl von Attributen besitzen. Nicht jedes Attribut wird dabei für die Berechnung der Ähnlichkeit verwendet. Diese Attribute, die etwa die Lösung beschreiben, können einen beliebigen Datentyp haben. Attribute, die zur Berechnung der Ähnlichkeit verwendet werden, müssen vom Datentyp „Text“, „Zahl“, „Datum“, „Taxonomie“ oder „Referenz“ sein.

A3: Benutzer/innen können ohne Programmierkenntnisse neue Attribute zu Fällen hinzufügen oder Attribute entfernen. Dies funktioniert auch, wenn bereits Fälle im System gespeichert sind.

A4: Verweist ein Fall mittels Referenz auf einen oder mehrere andere Inhalte, so können diese ebenfalls zur Berechnung der Ähnlichkeit genutzt werden.

A5: Für jedes Attribut, das für die Berechnung der Ähnlichkeit verwendet wird, kann ein Gewicht angegeben werden. Das Gewicht wird bei der Berechnung der globalen Ähnlichkeit berücksichtigt.

A6: Zu jedem Fall soll gespeichert werden können, ob dieser neu angelegt wurde, ob die Lösung aktuell überprüft wird oder ob der Fall abgeschlossen ist.

A7: Die fünf ähnlichsten Fälle sollen übersichtlich dargestellt werden. Wenn gewünscht, sollen aus den angezeigten Fällen Lösungen für den aktuellen Fall übernommen werden können.

A8: Die Ansicht der ähnlichen Fälle soll ohne Programmierkenntnisse angepasst werden können. Sie soll um Spalten ergänzt werden können und die Anzahl der angezeigten ähnlichen Fälle soll konfigurierbar sein.

A9: Der/die Benutzer/in kann aus der Liste der ähnlichsten Fälle einen Fall auswählen, aus welchem die Lösung für den aktuellen Fall übernommen werden soll. Dabei kann der/die Benutzer/in für jedes Feld auswählen, ob die Lösung übernommen werden soll oder nicht. Die Anpassung der Lösung an den aktuellen Fall kann anschließend manuell vorgenommen werden.

A10: Wird ein Fall als Lösung ausgewählt, so wird dessen Status von „New" auf „Testing solution" geändert. Zudem wird eine Referenz auf den Fall, dessen Lösung ausgewählt wurde, hinzugefügt.

A11: Die Konfiguration des Case-Based-Reasoning-Systems soll unabhängig vom Inhalt der Fallbasis exportiert werden können. Ebenso soll es möglich sein, diese Konfiguration wieder zu importieren, sodass die Struktur der Fallbasis mit allen konfigurierten Attributen, Gewichten und anderen Einstellungen wie exportiert zur Verfügung steht.

A12: Nur bestimmte Benutzer/innen sollen die Struktur der Fallbasis sowie die Konfiguration der Attribute ändern können. Ebenso sollen nur bestimmte Benutzer/innen neue Fälle anlegen und ändern können.

Weiters sollen folgende nichtfunktionale Anforderungen umgesetzt werden:

N1 Usability: Das Modul soll für den/die Anwender/in einfach zu bedienen sein, indem es auf bestehenden Workflows aufbaut. Inhaltstypen, bei denen Case-Based Reasoning nicht aktiviert ist, sollen wie gewohnt funktionieren und keine überflüssigen Buttons oder Hinweise anzeigen.

N2 Security: Es soll nicht möglich sein, dass über das CBR-Modul unberechtigte Personen Inhalte einsehen oder ändern können.

N3 Performance: Es soll möglich sein, dass auch mehr als 1.000 Fälle mit jeweils 10 Attributen vom System verarbeitet werden können.

N4 Erweiterbarkeit: Programmierer/innen sollen mit eigenen Modulen das CBR-Modul erweitern können. Dadurch sollen weitere Feldtypen sowie Ähnlichkeitsfunktionen hinzugefügt werden können.

3.3 Abgrenzung

Da für die Umsetzung der Diplomarbeit nur ein begrenzter Zeitraum zur Verfügung steht, werden folgende Punkte nicht umgesetzt:

- Analog zu myCBR und vielen Beispielen aus der Literatur wird die Lösung nicht vom System angepasst, sondern der/die Nutzer/in kann die Lösung aus ähnlichen Fällen direkt übernehmen und selbst anpassen.
- Verweist ein Fall auf einen anderen Inhalt, der wiederum auf einen anderen Inhalt verweist, so wird dies nicht bei der Berechnung der Ähnlichkeit berücksichtigt.
- Es ist nicht möglich, die Ähnlichkeit von Strings wie in myCBR bei „Symbolen" direkt festzulegen. Stattdessen wird diese immer errechnet.
- Es ist nicht möglich, aus mehreren Methoden zur Berechnung der globalen Ähnlichkeit zu wählen. Stattdessen wird immer der gewichtete Mittelwert der lokalen Ähnlichkeiten verwendet.

3.4 Implementierung des Prototyps

Damit eine hohe Qualität sichergestellt werden kann, werden die Anforderungen in fünf Arbeitspakete aufgeteilt. Jeder Abschluss eines Arbeitspakets stellt einen erreichten Meilenstein dar. Jeder Meilenstein wird anhand eines Throwaway-Prototyps überprüft.

In Arbeitspaket 1 werden die Anforderungen A1, A2, A5 umgesetzt. Damit ist es Benutzer/innen möglich, zu beliebigen Inhalten in Drupal bestimmte Felder hinzuzufügen, deren Inhalt mit anderen Feldern verglichen wird. Damit erhält der/die Nutzer/in die Möglichkeit, besonders ähnliche Inhalte zu einem Inhalt anzuzeigen. Dies kann beispielsweise in einem Webshop nützlich sein, wenn Kund/innen zu einem Produkt ähnliche Produkte angezeigt werden sollen. Ebenso kann es in einem Content-Management-System hilfreich sein, besonders ähnliche Artikel anzuzeigen, um etwa Duplikate zu finden und zu entfernen.

Dazu werden in einem neuen Drupal-Modul die Feldtypen „CBR Number (dezimal)", „CBR Number (integer)", „CBR Text" und „CBR Date" implementiert. Um einen neuen Feldtyp zu einem Inhaltstyp hinzuzufügen, muss im Modul eine Datei im Ordner `src/Plugin/Field/FieldType` angelegt werden. Wird hier nun eine neue Klasse angelegt und entsprechend annotiert, so erscheint der neue Feldtyp unter „Verwaltung > Struktur > Inhaltstypen" in der Administrationsoberfläche von Drupal und Benutzer/innen können ein neues Feld dieses Typs zu einem Inhaltstyp hinzufügen. Jeder neue Feldtyp basiert auf einem von Drupal vordefinierten Feldtyp, beispielsweise „Number

(integer)“, der in der Klasse `Drupal\Core\Field\Plugin\Field\FieldType\IntegerItem` implementiert ist. Die objektorientierte Implementierung von Drupal 9 ermöglicht es, dass die neuen Felder alle Funktionen zum Speichern, Bearbeiten und Anzeigen von den mit Drupal ausgelieferten Feldern übernehmen, indem die Klasse des neuen Feldtyps von `IntegerItem` und so weiter erbt. Um für jedes Feld ein Gewicht für die Berechnung der globalen Ähnlichkeit speichern zu können, wird die Funktion „fieldSettingsForm“ überschrieben. Nach dem Aufruf von `parent::fieldSettingsForm`, welcher alle standardmäßigen Formularelemente des Feldes erstellt, wird ein Eingabeelement zur Angabe eines Gewichts erstellt. Der Wert dieses Eingabeelementes wird in den „thirdPartySettings“ des Feldes gespeichert. Da dieses Eingabeelement in allen CBR-Feldern benötigt wird, ist es in einer Hilfsklasse implementiert und kann so von verschiedenen Klassen aufgerufen werden.

Damit das Feld später zur Berechnung der Ähnlichkeit verwendet werden kann, implementiert es ein neues Interface „`CBRFieldInterface`“ (siehe Listing 2). Das Interface deklariert die Methoden `calculateSimilarity`, `getValueForSimilarityCalculation` und `summerize`. Das Berechnen der lokalen Ähnlichkeit eines Feldes beziehungsweise eines Attributes erfolgt in der Methode `calculateSimilarity`. Dieser Methode werden zwei Werte übergeben, die aus zwei Fällen stammen, die miteinander verglichen werden. In manchen Feldtypen kann der gespeicherte Wert nicht direkt vom CBR-Modul zur Berechnung der Ähnlichkeit verwendet werden. In diesem Fall ermöglicht `getValueForSimilarityCalculation` eine individuelle Berechnung des Wertes. Summerize wird in Arbeitspaket 2 beziehungsweise Anforderung A4 benötigt.

```
1.  interface CBRFieldInterface
2.  {
3.    /**
4.     * Calculates the similarity between two values.
5.     * @param $value1 The first value.
6.     * @param $value2 The second value.
7.     * @param FieldConfig $field_config The field config.
8.     * @return float The similarity [0-1] between the two values.
9.     */
10.   function calculateSimilarity(mixed $value1, mixed $value2, FieldConfig $field_config):
    float;
11.
12.   /**
13.    * Get the value of the field prepared for use in the similarity calculation.
14.    * @param FieldConfig $field_config The field config.
15.    * @return mixed The value of the field, prepared for use in the similarity calculation.
16.    */
17.   function getValueForSimilarityCalculation(FieldConfig $field_config): mixed;
18.
19.   /**
20.    * Summerize multiple values into a single value.
21.    * @param array $values The values to summerize.
22.    * @return mixed The summerized value.
23.    */
24.   function summerize(array $values): mixed;
25. }
```

Listing 2: Jeder Feldtyp, der für CBR zur Verfügung stehen soll, muss CBRFieldInterface implementieren.

Um nun die lokale Ähnlichkeit zu berechnen, muss jeder Feldtyp abhängig vom Datentyp einen eigenen Algorithmus für `calculateSimilarity` implementieren. Der Algorithmus für numerische Werte wird in den Feldern „CBR Number (dezimal)“, „CBR Number (integer)“ und „CBR Date“ verwendet und daher in eine Hilfsklasse ausgelagert. Zur Berechnung der lokalen Ähnlichkeit zwischen zwei Zahlenwerten wird abhängig davon, ob ein Minimum und ein Maximum für das Feld definiert wurden, unterschiedlich vorgegangen.

Wenn ein Minimum und ein Maximum definiert sind, wird die lokale Ähnlichkeit zwischen *a* und *b* wie folgt berechnet:

$$similarity(a,b) = 1 - \frac{abs(a-b)}{\max - min}$$

Die Ähnlichkeitsberechnung zwischen zwei Zahlenwerten wird in dieser Form auch in der Literatur beschrieben [13, S. 331, 17, S. 250] und auch von myCBR und JColibri umgesetzt.

Wenn kein Minimum und kein Maximum definiert sind, dann wird die lokale Ähnlichkeit zwischen *a* und *b* wie folgt berechnet:

$$similarity(a,b) = 1 - \frac{abs\,(a-b)}{abs(a-b)+1} \text{ beziehungsweise vereinfacht } \frac{1}{abs(a-b)+1}$$

Diese Formel zur Berechnung der Ähnlichkeit zwischen zwei Zahlenwerten ohne Minimum und Maximum wird auch von *Richter* [17, S. 250] verwendet.

Bei Datumsangaben können Minimum und Maximum nicht definiert werden, daher wird immer die zweite Formel verwendet. Da bei dieser Formel bei einer großen Differenz zwischen *a* und *b* die Ähnlichkeit schnell sehr klein wird und damit eventuell andere Attribute unverhältnismäßig stark gewichtet werden, kann der/die Ersteller/in des CBR-Systems für jedes Datumsfeld wählen, ob die Differenz zwischen den beiden Datumsangaben in Tagen, Monaten oder Jahren verglichen werden soll. Diese Entscheidung sollte auf der Grundlage der erwarteten Daten getroffen werden, kann jedoch jederzeit geändert werden.

Drupal bietet verschiedene Feldtypen für Texte, wie etwa „Text (formatted)“ und „Text (plain, long)“. Für alle Textfeldtypen wird ein entsprechender Typ, der CBR unterstützt, angelegt. Auch hier erbt die Klasse, die den Feldtyp implementiert, von der in Drupal enthaltenen Klasse. Für alle Textfelder kann über das Konfigurationsmenü des Textfeldes ausgewählt werden, ob die Ähnlichkeit über den Jaccard-Koeffizienten oder über die Kosinus-Ähnlichkeit berechnet werden soll. Bevor die Ähnlichkeit berechnet wird, werden HTML-Tags entfernt und der Text in Kleinbuchstaben umgewandelt. Danach werden alle Zeichen, welche keine Buchstaben oder Ziffern sind, entfernt

und alle Whitespaces durch Blanks ersetzt. Dabei werden mehrfach hintereinander auftretende Whitespaces durch ein einzelnes Blank ersetzt. Anschließend wird daraus ein Array gebildet, wobei jedes Element ein Wort enthält.

Der Jaccard-Koeffizient zwischen zwei Texten wird damit wie folgt berechnet: Zuerst werden aus den zuvor gebildeten Arrays alle Duplikate entfernt, wodurch zwei Mengen A und B entstehen. Zur Berechnung der Ähnlichkeit wird folgende Formel genutzt:

$$similarity(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Die Ähnlichkeit zwischen den Mengen A und B wird also dadurch berechnet, dass die Anzahl der Elemente der Schnittmenge von A und B durch die Anzahl der Elemente der Vereinigungsmenge von A und B dividiert wird. Da der Jaccard-Koeffizient nicht berücksichtigt, ob ein Wort einmal oder mehrmals vorkommt, eignet er sich besonders für den Vergleich von Aufzählungen. Dies können etwa Eigenschaften eines Objektes oder die Kenntnisse einer Person sein.

Soll jedoch auch die Häufigkeit der Wörter berücksichtigt werden, so kann die Kosinus-Ähnlichkeit verwendet werden. Hierbei wird wie zuvor je ein Array von Wörtern aus den beiden Texten, welche miteinander verglichen werden sollen, gebildet. Danach wird aus dem Array ein Vektor gebildet. Dieser besteht aus den Häufigkeiten der Wörter im Text. Kommt ein Wort in Text A nicht vor, in Text B jedoch schon, so wird für dieses Wort 0 dem Vektor hinzugefügt und umgekehrt. Aus den beiden Vektoren wird anschließend der Kosinus (= die Kosinus-Ähnlichkeit) wie folgt berechnet:

$$similarity(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

In Drupal können sogenannte Taxonomien erstellt werden. Diese können als Kategorien oder Tags für Inhalte verwendet werden. Taxonomien können eine beliebige Anzahl von Elementen enthalten. Diese Elemente können hierarchisch angeordnet werden. Damit ist die Modellierung komplexer Ontologien möglich. Da mit Hilfe von Taxonomien Inhalte in beliebigen Dimensionen kategorisiert werden können, sind sie auch für die Ermittlung der Ähnlichkeit sehr wertvoll. Daher wird der Feldtyp „CBR Taxonomy Reference“ hinzugefügt, der dies ermöglicht. Im Prototyp werden zwei Algorithmen zur Ermittlung der Ähnlichkeit implementiert, welche von dem/der Benutzer/in in den Einstellungen des Feldes ausgewählt werden können. Für den Einsatz von Taxonomien, bei denen die Elemente in keiner Beziehung zueinanderstehen, wie etwa Tags, kann der Jaccard-Koeffizient verwendet werden. Die Ähnlichkeit zwischen den Inhalten A und B wird mit der

zuvor beschriebenen Formel für den Jaccard-Koeffizienten berechnet, wobei jedes Element der Taxonomie einem Wort entspricht.

Werden die Elemente einer Taxonomie hierarchisch angeordnet, so bilden sie einen Baum, dessen Wurzelknoten die Taxonomie selbst ist. Die Elemente von Inhalt *A* und Inhalt *B* werden jeweils paarweise miteinander verglichen, wobei jedes Element von *A* mit jedem Element von *B* verglichen wird. Dabei gilt für jedes Paar: Je näher die beiden Elemente im Baum beieinander liegen, desto ähnlicher sind sie. Dazu wird die Anzahl der Kanten der Verbindungen zwischen den beiden Elementen ermittelt. Aus der Anzahl der Kanten zwischen *A* und *B* wird die Ähnlichkeit wie folgt berechnet:

$$similarity(A, B) = \frac{1}{\overline{AB} + 1}$$

Gibt es mehr als ein Paar zwischen *A* und *B*, so ist die resultierende Ähnlichkeit der Mittelwert der Ähnlichkeiten aller Paare.

Für alle Attribute wird eine lokale Ähnlichkeit ermittelt. Jedem Feld (= Attribut) muss ein Gewicht zugewiesen werden. Standardmäßig ist das 1. Aus den lokalen Ähnlichkeiten und dem Gewicht kann die globale Ähnlichkeit wie folgt errechnet werden:

$$globalSimilarity(A, B) = \frac{\sum_{i=1}^{n} w_i \, similarity(A_i, B_i)}{\sum_{i=1}^{n} w_i}$$

Dabei sind *A* und *B* Fälle, *n* ist die Anzahl der Attribute, für die Case-Based Reasoning aktiviert ist, $similarity(A_i\ B_i)$ ist die Ähnlichkeit des Attributes *i* von *A* beziehungsweise *B* und W_i ist das Gewicht des Attributes *i*.

Diese Formel zur Berechnung der globalen Ähnlichkeit zwischen zwei Fällen ist sowohl in myCBR als auch in JColibri implementiert und wird in der Literatur häufig genannt [vgl. 10, S. 70, 15, S. 127, 16, S. 812].

Die Ähnlichkeit wird beim Einfügen oder Ändern eines Inhalts berechnet und in der Datenbank des Content-Management-Systems gespeichert. Dazu wird im CBR-Modul ein neues Service „`SimilarityCalculator`“ angelegt. Dieses hat die Methode „`calculate`“ (siehe Listing 3), welche mit je zwei Fällen, die verglichen werden sollen, aufgerufen wird. Dabei können nur Fälle des gleichen Inhaltstyps verglichen werden. Dazu werden alle Felder der Fälle durchlaufen. Für jedes Feld, welches das `CBRFieldInterface` implementiert, wird aus dessen Konfiguration das Gewicht des Attributs geladen. Anschließend wird von beiden Fällen für das aktuelle Feld der Wert zur Berechnung

der Ähnlichkeit geladen. Zur Berechnung der lokalen Ähnlichkeit wird die Methode `calculateSimilarity` des Feldes aufgerufen. Die errechnete Ähnlichkeit wird anschließend mit dem Gewicht des Feldes multipliziert. Zur Berechnung der globalen Ähnlichkeit werden die gewichteten lokalen Ähnlichkeiten summiert. Abschließend wird die globale Ähnlichkeit auf den Bereich 0-1 normalisiert, indem sie durch die Summe der Gewichte aller beteiligten Felder dividiert wird. Nach dem Speichern des Inhalts werden die fünf ähnlichsten Fälle unter dem Reiter „Similar Cases“ aufgelistet. Diese Ansicht wird in einem späteren Arbeitspaket noch detaillierter umgesetzt.

```php
/**
 * Calculate similarity between two cases.
 * @param $entity1 The first case.
 * @param $entity2 The second case.
 * @return float The similarity [0-1] between the two cases.
 */
function calculate(EntityInterface $entity1, EntityInterface $entity2): float {
    $similarity = 0;
    $sum_weight = 0;
    /* @var FieldItemListInterface $field_item */
    foreach ($entity1->getFields(false) as $field_item) {
        if ($field_item->first() !== null
            && $field_item->first() instanceof CBRFieldInterface) {

            /* @var FieldConfig $field_config */
            $field_config = $field_item->getFieldDefinition()->
            getConfig($entity1->bundle());
            $weight = $field_config->getThirdPartySetting('cbr', 'weight', 1);

            /* @var CBRFieldInterface $field1 */
            $field1 = $field_item->first();
            $value1 = $field1->getValueForSimilarityCalculation($field_config);

            /* @var CBRFieldInterface $field1 */
            $field2 = $entity2->get($field_item->getName())->first();
            if ($field2) {
                $value2 = $field2->getValueForSimilarityCalculation($field_config);
                $similarity += $field1->
                    calculateSimilarity($value1, $value2, $field_config) * $weight;
            }
            $sum_weight += $weight;
        }
    }
    //Normalize similarity to [0-1]
    if ($sum_weight != 0) {
        $similarity = $similarity / $sum_weight;
    } else {
        $similarity = 0;
    }
    return $similarity;
}
```

Listing 3: Berechnung der globalen Ähnlichkeit zwischen zwei Fällen.

In Arbeitspaket 2 wird die Anforderung A4 umgesetzt. In Drupal können Inhalte auf einen oder mehrere andere Inhalte verweisen. Damit ist es etwa möglich, einem Projekt Mitarbeiter/innen zuzuordnen. Für die Berechnung der Ähnlichkeit zwischen zwei Projekten ist es nun sinnvoll, auch Attribute der Mitarbeiter/innen, wie etwa deren Kenntnisse und Ausbildungen, zu vergleichen. Dazu werden alle Attribute des referenzierten Inhalts zusammengefasst. Arbeiten in einem Projekt

beispielsweise drei Mitarbeiter/innen, die über drei, fünf und sieben Jahre Erfahrung mit Softwareentwicklung haben, so wird die Erfahrung auf „im Durchschnitt“ fünf Jahre zusammengefasst und anschließend mit den anderen, ebenfalls zusammengefassten Erfahrungen anderer Projekte verglichen.

Im CBR-Modul wird dazu das Interface, welches alle CBR-Felder implementieren, um den Methodenkopf „`summerize`“ erweitert. Dieser bekommt als Parameter ein Array verschiedener Werte und liefert als Ergebnis einen einzelnen, zusammengefassten Wert. Je nach Feldtyp ist die Methode „`summerize`“ unterschiedlich implementiert:

- Numerische Felder und Datumsfelder geben den Median aller Werte zurück.
- Textfelder geben alle Werte mit einem Leerzeichen verknüpft zurück.
- Felder, welche auf Taxonomien verweisen, bekommen als Input mehrere Sets unterschiedlicher Taxonomieterme und fassen diese zu einem Set zusammen. Dabei werden doppelte Einträge entfernt.

Wird nun die Ähnlichkeit zwischen zwei Feldern *a* und *b*, welche auf andere Inhalte verweisen, berechnet, dann werden zuerst im „`CBRReferenceField`“ alle Inhalte, auf welche verwiesen wird, geladen. Für jedes Attribut des referenzierten Inhaltstyps wird aus den geladenen Inhalten mittels „`summerize`“ ein Wert berechnet und temporär gespeichert. Anschließend wird für jedes Attribut aus den zusammengefassten Werten mit „`calculateSimilarity`“ die Ähnlichkeit ermittelt. Die Methode „`calculateSimilarity`“ wurde bereits in Arbeitspaket 1 implementiert. Alle Ähnlichkeiten werden, analog zur Berechnung der globalen Ähnlichkeit mit dem konfigurierten Gewicht multipliziert, aufsummiert und anschließend durch die Summe aller Gewichte dividiert. Die so erhaltene Ähnlichkeit zwischen zwei Feldern, die auf andere Inhalte verweisen, wird anschließend zusammen mit den Ähnlichkeiten der anderen Felder zur Berechnung der globalen Ähnlichkeit verwendet.

In den beiden vorangegangenen Arbeitspaketen wurde das „Retrieve“ des CBR Cycles von *Aamodt und Plaza* umgesetzt. In Arbeitspaket 3 werden die Anforderungen A6, A7, A8, A9 und A10 implementiert. Damit wird das „Reuse“, „Revise“ und „Retain“ des CBR Cycles umgesetzt.

Damit der Status eines Falls gespeichert werden kann, wird der Feldtyp „CBR Status“ implementiert. Dabei handelt es sich um ein Auswahlfeld mit den Optionen „New“, „Testing solution“, „Solution accepted“ und „Solution rejected“. Wenn von dem/der Benutzer/in ein Inhaltstyp für Case-Based Reasoning verwendet werden soll, dann muss diesem ein Feld vom Typ „CBR Status“ hinzugefügt werden. Anschließend kann für jeden Fall der Status gesetzt werden. Dabei kann ein Fall auch direkt mit „Solution accepted“ angelegt werden, um etwa die Wissensbasis initial zu

befüllen. Ebenso kann ein Fall später noch von „Solution accepted“ auf „Solution rejected“ geändert werden, etwa wenn sich herausstellt, dass die Lösung fälschlicherweise akzeptiert wurde.

Ebenso wurde der Feldtyp „CBR Reference to solution“ implementiert. Wie schon in Abschnitt 2.1.2 erwähnt, ist es wichtig, von einem Fall auf die initiale Lösung zugreifen zu können. Dies wird durch diesen Feldtyp umgesetzt.

Um zu einem Fall die fünf ähnlichsten Fälle anzeigen zu können, wird eine neue View hinzugefügt. Wenn in Drupal ein Inhalt eines Typs geöffnet wird, auf dem CBR aktiv ist, so wird der Tab „Similar Cases“ angezeigt. Dieser Tab zeigt eine Tabelle mit den fünf ähnlichsten Fällen. Die Tabelle hat die Spalten „Title“, „Case Status“, „Similarity“ und „Action“. In der Spalte „Action“ wird ein Button „Select as Solution“ angezeigt. Damit kann der angezeigte Fall als Lösung ausgewählt werden. Die View kann von dem/der Anwender/in jederzeit geändert werden. Es können beispielsweise weitere Spalten hinzugefügt werden und etwa mehr Fälle angezeigt werden, wenn dies gewünscht wird.

Wird ein Fall als Lösung ausgewählt, dann werden dem/der Benutzer/in der neue Fall und der Fall, aus dem die Lösung übernommen werden soll, in zwei Spalten dargestellt. In jeder Zeile wird ein Feld des Falls dargestellt. Für jedes Feld kann ausgewählt werden, ob der Inhalt des Feldes in den neuen Fall übernommen werden soll. Dazu wurde in der routing.`yml` des CBR-Moduls eine Route hinzugefügt und unter `src\Form\ReuseForm.php` das Formular entsprechend implementiert. Die Darstellung der Felder erfolgt über `\Drupal::service('renderer')`. Damit können Felder von beliebigen Modulen dargestellt werden, sodass die Lösung mit beliebigen Feldtypen in Drupal realisiert werden kann und trotzdem vom CBR-Modul verarbeitet werden kann. Wird das Formular abgeschickt, so werden die Inhalte der ausgewählten Felder in den neuen Fall kopiert, der Status des Falls wird auf „Testing Solution“ gesetzt und dem Feld vom Typ „CBR Reference to solution“ wird ein Verweis auf den gewählten Fall hinzugefügt. Anschließend kann der/die Benutzer/in die vorgeschlagene Lösung noch bearbeiten und anschließend „im Feld“ testen. Wie manche andere CBR-Systeme verzichtet auch das im Prototyp umgesetzte CBR-System auf eine regelbasierte Anpassung der Lösung und überlässt eine Anpassung dem/der Benutzer/in.

In Arbeitspaket 4 werden die Anforderungen A11 und A12 umgesetzt. Anforderung A11 wird dadurch umgesetzt, dass alle Einstellungen des CBR-Moduls über die Drupal-API gespeichert werden. Somit können die über das CBR-Modul vorgenommenen Einstellungen in Drupal über die „Configuration synchronization“ im Administrationsbereich exportiert werden. Das Ergebnis ist eine YAML-Datei, die in eine andere Drupal-Installation importiert werden kann. Zusammen mit

dem CBR-Modul kann so auf sehr einfache Weise ein vollständig konfiguriertes CBR-System erstellt und verteilt werden. Für Anforderung A12 werden die Rechte „cbr access similar cases“ und „cbr reuse“ eingeführt. Nur Nutzer/innen, welche sich in Gruppen mit der entsprechenden Rolle befinden, können daher ähnliche Fälle sehen, beziehungsweise das Formular „Reuse“ aufrufen. Dazu wird dem CBR-Modul die Datei „`cbr.permissions.yml`“ hinzugefügt, in welcher die beiden Rollen mit einem Titel und einer Beschreibung eingetragen werden. In der Datei „`cbr.permissions.yml`“ wird bei der Route zum Reuse-Formular unter „`requirements`“ die Zeile „`_permission: cbr access similar cases`“ eingetragen, womit das Formular nur Nutzer/innen mit den entsprechenden Rechten angezeigt wird. Damit der Tab „Similar Cases“ nur bei Inhaltstypen angezeigt wird, die Case-Based Reasoning unterstützen, wird in der Datei „`cbr.services.yml`“ ein „Route Subscriber“ und ein „Access Check“ konfiguriert. Dort wird zunächst geprüft, ob der aufgerufene Inhalt von einem Typ ist, der mindestens ein Feld enthält, das `CBRFieldInterface` implementiert. Anschließend wird überprüft, ob der/die aktuelle Benutzer/in das Recht „cbr access similar cases“ hat (siehe Listing 4).

```
1.  /**
2.   * Checks access for displaying the similar cases tab.
3.   */
4.  class SimilarCasesAccessCheck implements AccessInterface
5.  {
6.      //Set via Constructor Dependency Injection
7.      protected RouteMatchInterface $routeMatch;
8.      protected EntityTypeManagerInterface $entityTypeManager;
9.      protected LoggerChannelInterface $logger;
10.     /**
11.      * Check access for displaying the similar cases tab.
12.      *
13.      * @param \Drupal\Core\Session\AccountInterface $account
14.      * Run access checks for this account.
15.      *
16.      * @return \Drupal\Core\Access\AccessResultInterface
17.      * The access result.
18.      */
19.     public function access(AccountInterface $account)
20.     {
21.         $parameters = $this->routeMatch->getParameters();
22.         $node = $parameters->get('node');
23.         if (is_numeric($node)) {
24.             $node = $this->entityTypeManager->getStorage('node')->load($node);
25.         }
26.         if ($node instanceof Node && cbr_is_enabled_on_entity($node)) {
27.             $accessResult = AccessResult::allowedIfHasPermission($account, 'cbr access simi-
    lar cases');
28.             if (!$accessResult->isAllowed()) {
29.                 $this->logger->info('Access denied to similar cases tab for user %user.',
    ['%user' => $account->getAccountName()]);
30.             }
31.             return $accessResult;
32.         }
33.         return AccessResult::forbidden();
34.     }
```

Listing 4: Implementierungen eines „Access Checks“ für den Tab „Similar Cases“. Für die Constructor Dependency Injection siehe Listing 7.

Im letzten Arbeitspaket wird die Anforderung A4 umgesetzt. Damit soll es möglich werden, bestehende Inhaltsstrukturen um CBR-Felder zu erweitern und damit CBR auch für bestehende Daten und Datenstrukturen zu aktivieren. Die durch das CBR-Modul hinzugefügten Feldtypen lassen sich allen beliebigen Inhaltstypen hinzufügen, unabhängig davon, ob diese neu erstellt wurden oder bereits Daten enthalten. Damit kann etwa eine bestehende Sammlung von Dokumenten um ein Feld vom Typ „CBR Taxonomy Reference“ erweitert werden, mit dem die Dokumente mit Schlagworten versehen werden und das damit zur Ermittlung der Ähnlichkeit dient. Die gewählte Umsetzung mit eigenen Feldtypen für Daten, die anschließend zur Berechnung der Ähnlichkeit verwendet werden sollen, hat jedoch einen Nachteil: Was ist, wenn die gewünschten Daten schon in einem Feldtyp ohne CBR enthalten sind? Dann würde ein Anlegen eines neuen Feldtyps mit CBR bedeuten, dass Daten in bestehenden Feldern nicht zur Berechnung der Ähnlichkeit genutzt werden können.

Drupal erlaubt keine Änderung des Feldtyps. Ist ein Feld erstellt, kann der Typ von diesem nicht mehr geändert werden. Um dennoch bestehende Daten mit CBR nutzbar zu machen, wurde ein eigenes Modul mit dem Namen „Field content copy helper“ erstellt. Dieses fügt der Inhaltsverwaltung im Administrationsbereich ein neues Formular (siehe Abbildung 17) hinzu, bei dem der/die Benutzer/in zuerst einen Inhaltstyp und anschließend zwei Felder dieses Inhaltstyps auswählen kann. Wird das Formular abgeschickt, so werden für alle Inhalte des ausgewählten Typs die Daten des gewählten Feldes in das andere Feld kopiert. Die durch das CBR-Modul hinzugefügten Feldtypen basieren auf den mit Drupal ausgelieferten Datentypen, womit eine Umwandlung leicht möglich ist, da sie mit derselben Struktur in der Datenbank gespeichert werden. Damit ist eine Umwandlung zwischen den Feldern der folgenden Typen möglich:

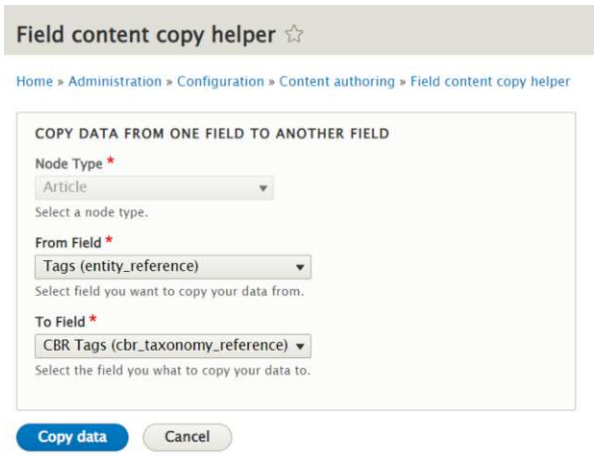

Abbildung 17: Formular des Moduls „Field content copy helper“. Eigene Aufnahme.

- „CBR DateTime“ und „Date“
- „CBR Number“ und „Number“
- „CBR Text“ und „Text“
- „CBR Node Reference“ und „Content“ aus dem Abschnitt „Reference“
- „CBR Taxonomy Reference“ und „Taxonomy Term“ aus dem Abschnitt „Reference“

Das Modul „Field content copy helper" kann auch verwendet werden, wenn das CBR-Modul entfernt werden soll, die erstellten Inhalte jedoch im System verbleiben sollen. Dazu muss für jedes CBR-Feld das entsprechende Feld aus der Standardinstallation von Drupal hinzugefügt werden und anschließend müssen die Daten mit dem „Field content copy helper" kopiert werden. Drupal erlaubt das Entfernen des CBR-Moduls erst, nachdem alle Feldtypen, die durch das Modul implementiert wurden, aus allen Inhaltstypen entfernt wurden. Beim Entfernen eines Feldes gehen auch alle Daten verloren, weshalb diese vorher in ein anderes Feld kopiert werden sollten.

3.5 Veröffentlichung auf drupal.org

Der Prototyp wird als Drupal-Modul auf drupal.org veröffentlicht[26]. Dazu wird ein neues Projekt auf drupal.org angelegt. Dieses dient einerseits dazu, das Modul öffentlich bekannt zu machen, andererseits können darüber auch Releases verwaltet, Issues getrackt, Co-Maintainer ernannt werden und vieles mehr. Für alle Module auf drupal.org wird automatisch ein Repository auf git.drupalcode.org angelegt. Damit kann nicht nur der Code des Moduls verwaltet werden, sondern es können auch Änderungen von Dritten über Merge-Requests eingepflegt werden. Um eine neue Version zu veröffentlichen, wird zunächst ein neuer Git-Tag erstellt. Auf diesen wird dann in der Benutzeroberfläche von drupal.org bei der Erstellung eines neuen Releases verwiesen. Der Paketmanager Composer[27] ermöglicht eine einfache Installation aller auf drupal.org veröffentlichten Module. So kann das in dieser Diplomarbeit erstellte Modul mit dem Befehl „`composer require drupal/cbr`" in jeder Drupal 9 Instanz installiert werden. Dabei wird auch das vom CBR-Modul benötigte Drupal-Modul „views_field_permissions" automatisch installiert.

[26] https://www.drupal.org/project/cbr, abgerufen am 30.10.2022

[27] https://getcomposer.org/, abgerufen am 30.10.2022

4 Evaluierung

4.1 Umsetzung des Szenarios

Um den entwickelten Prototyp zu evaluieren, wird das Szenario aus Abschnitt 2.3.4 erweitert. Einsatzbereich des Szenarios ist ein Unternehmen, welches Softwareprojekte für Kund/innen aus unterschiedlichen Branchen durchführt. Das Unternehmen führt rund 50 Projekte pro Jahr durch und hat zirka 1.000 Mitarbeiter/innen. Um die Planung nicht bei jedem neuen Projekt von null beginnen zu müssen und Knowhow aus einem abgeschlossenen Projekt auf neue Projekte übertragen zu können, entscheidet sich das Unternehmen für den Einsatz von Case-Based Reasoning. Schon bisher wurden Projekte zur internen Verwaltung in einem internen Content-Management-System gespeichert. Alle Mitarbeiter/innen füllen nach dem Eintritt in das Unternehmen ein Formular im CMS aus. Dabei geben sie an, welche Erfahrungen sie mit welchen Programmiersprachen haben und welche weiteren Fähigkeiten und Kompetenzen sie haben. Diese Daten sollen nun für Case-Based Reasoning verwendet werden.

Um nun mit Hilfe des CBR-Moduls aus der bestehenden Drupal-Installation ein funktionsfähiges System für Case-Based Reasoning zu machen, wird wie folgt vorgegangen: Das Modul wird über die Oberfläche von Drupal installiert. Dabei werden die zuvor beschriebenen Feldtypen dem System hinzugefügt. Zusätzlich wird die Ansicht für ähnliche Fälle und das Formular für den „Reuse"-Schritt des CBR Cycle automatisch angelegt. Da schon Daten vorhanden sind, wird zudem das Modul „field content copy helfer" installiert. Die Modellierung der Wissensbasis erfolgt über die Verwaltung der Inhaltstypen im Administrationsbereich von Drupal. Zu diesem Zweck wird der bestehende Inhaltstyp „Projekte" aufgerufen und alle Felder, die für die Ähnlichkeitsberechnung verwendet werden sollen, werden durch Felder mit CBR-Funktionalität ersetzt. Dazu werden neue Felder mit Feldtypen aus dem CBR-Modul angelegt (siehe Abbildung 18). Jedes Feld kann in Drupal konfiguriert werden, beispielsweise kann für numerische Typen ein Minimum und ein Maximum angegeben werden. Für die Berechnung der Ähnlichkeit zwischen numerischen Typen werden Minimum und Maximum aus dieser Konfiguration verwendet. Andere Optionen müssen hingegen explizit konfiguriert werden. Für jedes Feld kann ein Gewicht angegeben werden, das bei der Berechnung der globalen Ähnlichkeit verwendet wird. Für Textfelder und Taxonomien kann zudem die Funktion zur Berechnung

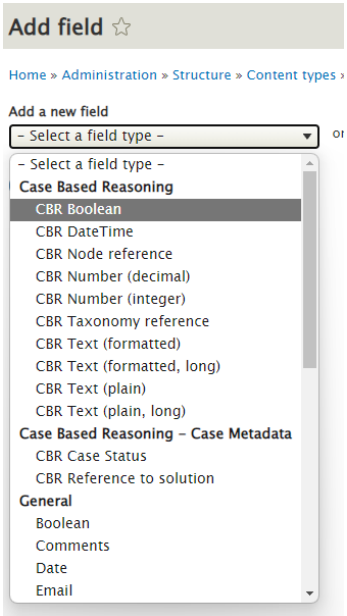

Abbildung 18: Hinzufügen eines neuen Feldes aus dem CBR-Modul zu einem Inhaltstyp. Eigene Aufnahme.

der lokalen Ähnlichkeit konfiguriert werden (siehe Abbildung 19). Dieser Vorgang wird für alle Felder wiederholt, die für die Ähnlichkeitsberechnung verwendet werden sollen (siehe Abbildung 20). Felder, welche die Lösung des Falls abbilden, müssen dabei nicht verändert werden beziehungsweise können aus den Feldern, die das „Field"-Modul von Drupal zur Verfügung stellt, gebildet werden.

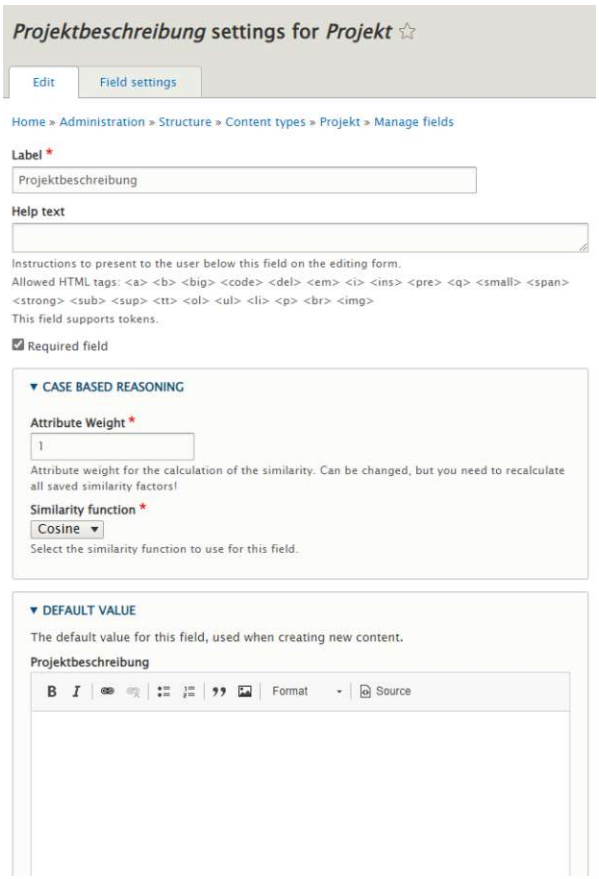

Abbildung 19: Gewicht und Funktion zur Berechnung der Ähnlichkeit können in den Feldeinstellungen konfiguriert werden. Eigene Aufnahme.

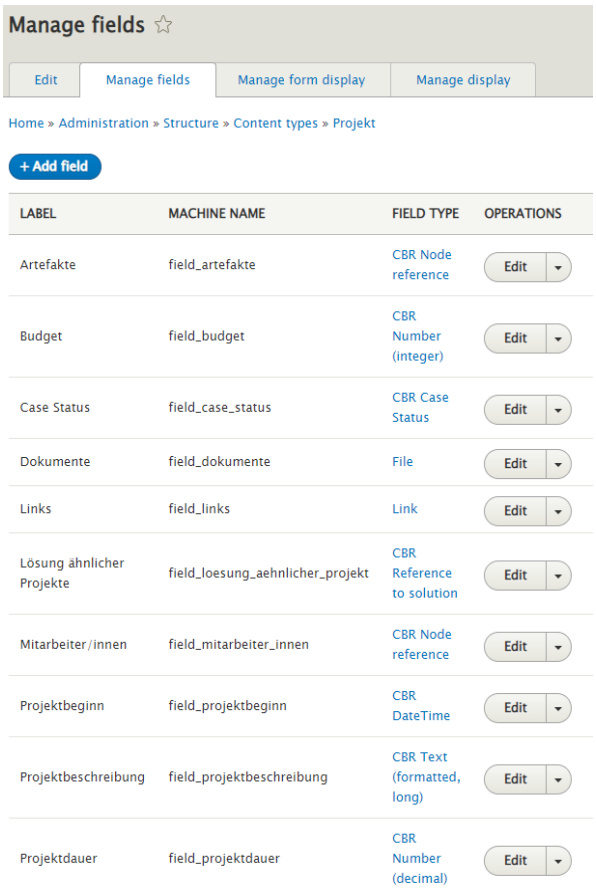

Abbildung 20: Modellierung der Wissensbasis mit Feldern aus dem CBR-Modul und aus Drupal. Eigene Aufnahme.

Zudem wird dem Inhaltstyp „Projekt" ein Feld vom Typ „CBR Case Status" hinzugefügt. Damit kann der Fallstatus („New", „Testing solution", „Solution accepted" und „Solution rejected") gespeichert werden. Ein Feld dieses Typs muss zu jedem Inhaltstyp hinzugefügt werden, der als CBR-Fall verwendet werden soll. Ebenso wird dem Inhaltstyp „Projekt" ein Feld vom Typ „CBR Reference to solution" hinzugefügt.

Für die Inhaltstypen „Mitarbeiter/innen" und „Artefakte" wird analog verfahren. Dem Inhaltstyp „Mitarbeiter/innen" wird ein Feld vom Typ „CBR Case Status" hinzugefügt. Da beim Inhaltstyp „Mitarbeiter/innen" der CBR Cycle nicht durchlaufen wird, wird der Standardwert vom Feld „CBR

Case Status“ auf „Accepted solution“ gestellt. Damit können zu jedem/jeder Mitarbeiter/in jene Kolleg/innen angezeigt werden, welche die ähnlichsten Fähigkeiten und Kompetenzen haben. Dies kann beispielsweise hilfreich sein, wenn Ersatz für eine/n Mitarbeiter/in gesucht wird oder ein Team vergrößert werden soll. Die Möglichkeit, Inhalte über ihre gesamte Struktur hinweg zu vergleichen und ähnliche Inhalte zu beliebigen Inhalten zu finden, erweitert die Einsatzmöglichkeiten eines CMS enorm. Dem Inhaltstyp „Artefakte“ werden zwar CBR-Felder hinzugefügt, jedoch kein Feld vom Typ „CBR Case Status“. Damit findet beim Speichern eines neuen Artefakts keine Ähnlichkeitsberechnung statt. Da Projekte jedoch auf Mitarbeiter/innen und Artefakte verweisen, werden die CBR-Felder in diesen beiden Inhaltstypen bei der Berechnung der Ähnlichkeit zwischen zwei Projekten verwendet.

Um die Ähnlichkeit zwischen vorgegebenen Begriffen zu definieren, können Taxonomien verwendet werden. Im Szenario kann für jedes Artefakt angegeben werden, mit welchem Framework es umgesetzt wird. Dazu wird die Taxonomie „Framework“ angelegt und die Terme werden in einer Baumstruktur angeordnet (siehe Abbildung 21). Dem Inhaltstyp „Artefakte“ wird anschließend ein Feld vom Typ „CBR Taxonomy Referenz“ hinzugefügt. Wird nun ein Artefakt mit dem Framework „Angular“ mit einem Artefakt mit dem Framework „Vue.js“ verglichen, so besitzen diese eine Ähnlichkeit von 0,33 und damit eine höhere Ähnlichkeit als beispielsweise „Angular“ und „Laravel“ mit einer Ähnlichkeit von 0,2.

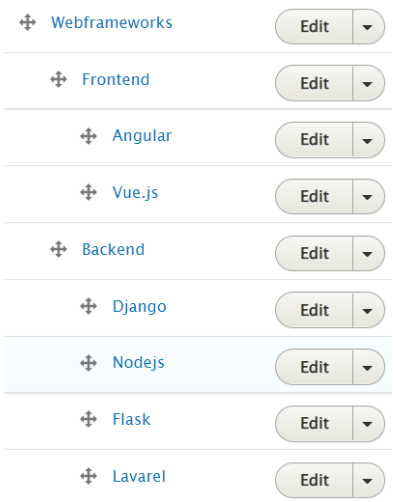

Abbildung 21: Taxonomien erlauben die Modellierung der Ähnlichkeit von unterschiedlichen Termen per Drag and Drop. Eigene Aufnahme.

Wird das Szenario nicht von bestehenden Daten aus modelliert, so kann der im CBR-Modul enthaltene Inhaltstyp „Sample Case“ angepasst werden. Bei diesem Inhaltstyp sind alle für CBR erforderlichen Felder bereits konfiguriert, es können jedoch weitere Felder hinzugefügt und bestehende Felder geändert werden.

4.2 Einsatz des Szenarios und Vergleich mit dem CBR Cycle

Das wie oben beschrieben konfigurierte CBR-System kann anschließend genutzt werden. Dazu wird im Content-Management-System ein neuer Inhalt vom Typ „Projekt“ erstellt (siehe Abbildung 22). Felder, welche die Lösung beschreiben, werden leer gelassen. Der Ablauf ist bisher völlig ident wie beim Erstellen eines Inhalts ohne CBR. Wird das Formular abgesendet, wird der

Fallstatus auf „New“ gesetzt und das Projekt wird mit allen anderen Projekten im System verglichen und die Ähnlichkeiten werden mit jedem anderen im System gespeicherten Projekt berechnet. Das kann ein paar Sekunden bis Minuten dauern. Anschließend werden die fünf ähnlichsten Fälle in einer Tabelle angezeigt. Dies entspricht dem „Retrieve“-Schritt des CBR Cycles.

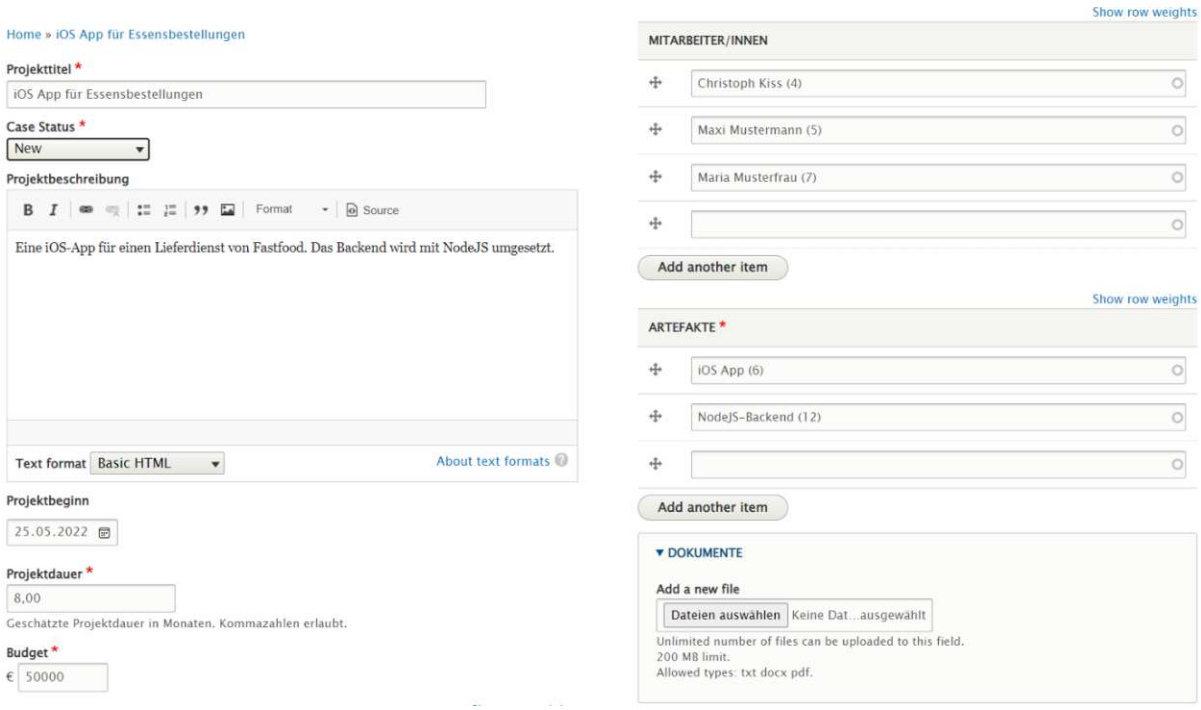

Abbildung 22: Formular zum Erstellen eines neuen Projektes. Eigene Aufnahme.

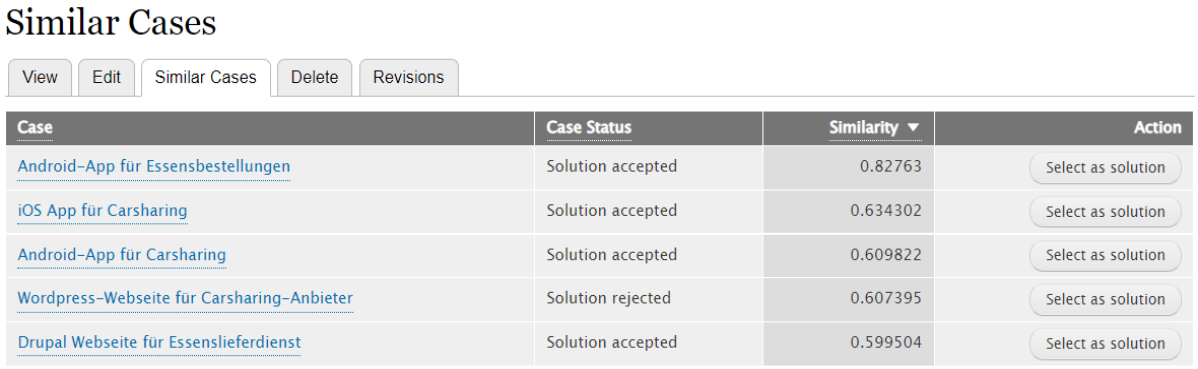

Abbildung 23: Anzeige der ähnlichsten fünf Fälle zu einem Fall. Aus diesen kann ein Fall als Lösung ausgewählt werden. Eigene Aufnahme.

Aus den fünf ähnlichsten Fällen (siehe Abbildung 23) kann nun der/die Anwender/in nun den am besten passenden Fall als Lösung auswählen. Das anschließende Formular bietet nun die Möglichkeit, die gewünschten Inhalte der Lösung des ausgewählten Falls in den aktuellen Fall zu übernehmen (siehe Abbildung 24). Wird das Formular abgeschickt, so wird der Status des Falls auf „Testing solution“ geändert, die Inhalte aller ausgewählten Felder werden übertragen und dem aktuellen Fall wird eine Referenz auf den als Lösung ausgewählten Inhalt hinzugefügt. Anschließend kann

der/die Nutzer/in die vorgeschlagene Lösung manuell anpassen. Damit ist der „Reuse"-Schritt des CBR Cycle abgeschlossen.

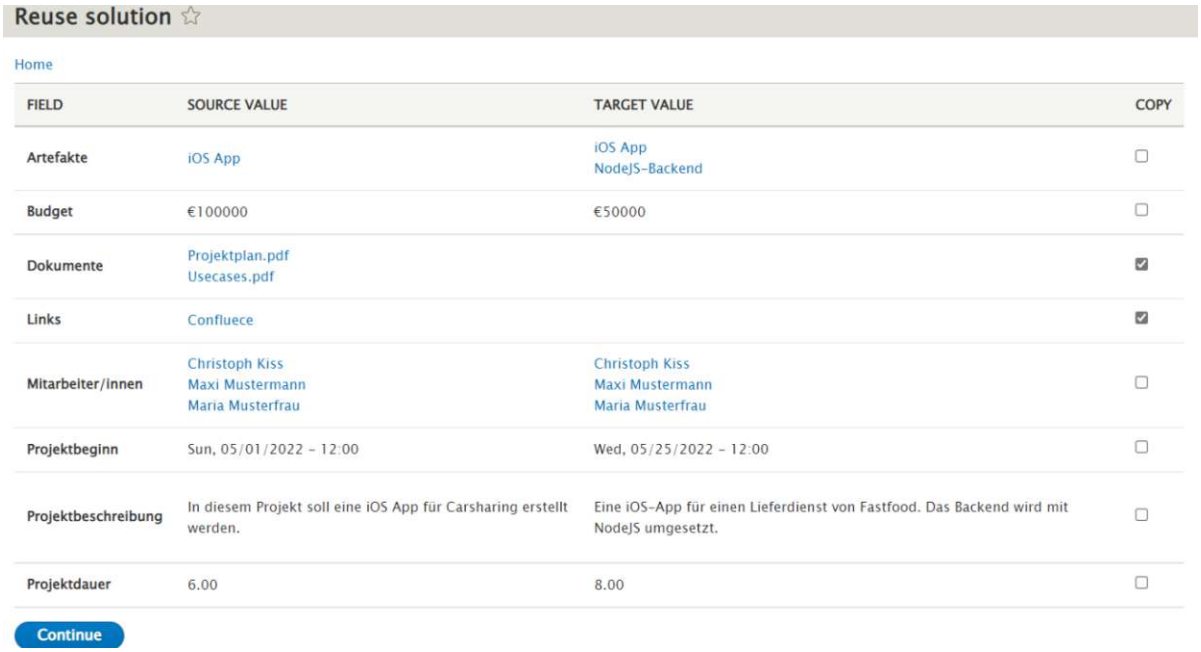

Abbildung 24: Formular zur Übernahme der Lösung in den aktuellen Fall. Felder, deren Inhalt aus dem als Lösung verwendeten Fall in den neuen Fall übernommen werden sollen, können ausgewählt werden. Eigene Aufnahme.

Die Lösung wird anschließend getestet. Im Fall des Szenarios bedeutet dies, dass das Team das Projekt durchführt und dabei auf Dokumente, Planungsunterlagen und Links des im System gespeicherten Falls zugreift. Wenn etwas nicht wie vorgeschlagen funktioniert, kann dies im Fall dokumentiert werden und die vorgeschlagene Lösung angepasst werden. Auf diese Weise kann vermieden werden, dass ähnliche Fehler in ähnlichen Projekten wiederholt werden. Ergebnis dieser im CBR Cycle als „Revise" bezeichneten Phase ist ein getesteter und gegebenenfalls reparierter Fall.

Wenn das Projekt abgeschlossen ist – erfolgreich oder auch nicht erfolgreich – wird in einem Textfeld im Fall beschrieben, warum das Projekt erfolgreich beziehungsweise nicht erfolgreich war und was davon bei ähnlichen Projekten berücksichtigt werden sollte. Auf diese Weise werden die „Lessons learned" des gesamten Projektteams gesammelt und im Fall hinterlegt. Anschließend wird der Status auf „Solution accepted" beziehungsweise „Solution rejected" gesetzt. Damit ist der Fall in die Wissensbasis aufgenommen und wird zukünftig bei der Suche nach ähnlichen Fällen berücksichtigt.

Beim Testen der Lösung kann es vorkommen, dass sich eine gewählte Lösung als nicht praktikabel erweist und daher eine andere Lösung ausprobiert werden muss. Hierfür kann der Fall geklont

werden, beispielsweise mit dem Modul „Quick Node Clone[28]“. Beim ursprünglichen Fall wird der Status auf „Solution rejected“ gesetzt und beschrieben, warum die vorgeschlagene Lösung nicht zum Ziel geführt hat. Dies ist wichtig, damit ähnliche Fehler in Zukunft vermieden werden. Wie in der Literatur erwähnt [5, S. 53f.], ist es wichtig, dass auch Lösungen, die nicht zum Ziel führen, im System verbleiben. Mit der Kopie des Falls kann beim Schritt „Revise” fortgefahren werden, nur dass nun eine andere Lösung gewählt wird.

Der CBR Cycle muss in dem umgesetzten Prototyp nicht streng eingehalten werden, was zu weiteren Möglichkeiten für den/die Anwender/in führt. Wie am Beispiel des Szenarios gut ersichtlich ist, kann es sein, dass die Lösung nicht aus einem Projekt, sondern aus mehreren Projekten übernommen werden soll. Das kann nützlich sein, wenn etwa Dokumente, Links und Planungen aus verschiedenen Projekten für das neue Projekt geeignet sind. Dazu kann der Schritt des „Reuse“ mehrmals durchlaufen werden: Nachdem ein Fall als Lösung ausgewählt wurde, kann der/die Nutzer/in erneut zum Tab „Similar Cases“ wechseln und dort einen weiteren Fall als Lösung auswählen. Wie zuvor wird der Inhalt der ausgewählten Felder in den ursprünglichen Fall kopiert und der Fall als weitere Lösung referenziert.

4.3 Analyse der Performance

Wichtig für die Akzeptanz eines CBR-Systems bei dem/der Benutzer/in ist die Wartezeit bei der Suche nach ähnlichen Fällen. Im umgesetzten Prototyp werden die Ähnlichkeiten zwischen allen Fällen zwischengespeichert und können daher jederzeit rasch aufgerufen werden. Allerdings müssen die Ähnlichkeiten nach dem Einfügen und Ändern eines Falls neu berechnet werden. Um die Dauer der Berechnung bei einer großen Anzahl von Fällen zu ermitteln, werden mit dem Drupal-Modul „Devel Generate[29]“ Fälle mit Dummy-Daten erzeugt. Da alle Felder des CBR-Moduls als Unterklassen von Feldern aus dem Drupal-Core implementiert wurden und die Felder in Drupal über eine Methode zur Generierung von Zufallsdaten verfügen, ist keine weitere Programmierung oder Konfiguration notwendig. Dabei werden so weit wie möglich gültige Datensätze erzeugt. Beispielsweise wird bei numerischen Werten das konfigurierte Minimum und Maximum berücksichtigt. Felder mit Referenzen verweisen auf schon zuvor angelegte Inhalte und Taxonomien. Im CBR-Modul wird die Dauer der Ähnlichkeitsberechnung ausgegeben. Dazu wird die Zeit zwischen dem Start des Batch-Prozesses und dessen Ende gemessen, es wird daher sowohl das Laden der Inhalte aus der Datenbank als auch die Berechnung selbst berücksichtigt. Nach der Generierung

[28] https://www.drupal.org/project/quick_node_clone, zuletzt abgerufen am 23.05.2022
[29] https://www.drupal.org/project/devel, abgerufen am 28.05.2022

der Daten wird für zehn zufällig ausgewählte Inhalte eine Berechnung der Ähnlichkeiten mit allen im System gespeicherten Fällen durchgeführt. Die Dauer der Berechnung ist in Tabelle 2 dargestellt. Dabei ist die Spanne zwischen dem Versuch mit der kürzesten und der längsten Dauer vermerkt.

Das umgesetzte Szenario geht von 50 Projekten pro Jahr und 1.000 Mitarbeiter/innen aus. Nach fünf Jahren wären also 250 Projekte in der Wissensbasis. Wenn für jedes Projekt ein Backend und ein Frontend implementiert wird, sind dies 500 Artefakte. Auf dem Testsystem dauert die Berechnung der Ähnlichkeiten für den Inhaltstyp „Mitarbeiter/innen“ zwischen 1,2 Sekunden und 1,6 Sekunden, für den Inhaltstyp „Artefakte“ zwischen 0,5 und 0,8 Sekunden und für den Inhaltstyp „Projekte“, der jedoch auf „Artefakte“ und „Mitarbeiter/innen“ verweist, zwischen 1,8 und 2,8 Sekunden. Natürlich hängt die genaue Dauer der Berechnung von der genauen Struktur und der verwendeten Hardware ab. Daher sind die absoluten Zahlen wenig aussagekräftig. In den Testläufen 2 und 3 wird daher die Anzahl der Inhalte jeweils vervierfacht. Es zeigt sich, dass die Berechnungsdauer im Wesentlichen proportional zur Anzahl der Inhalte in der Wissensbasis ansteigt. Bei genauerer Betrachtung zeigt sich, dass die Laufzeit leicht überproportional zur Anzahl der Inhalte ansteigt, so steigt die maximale Laufzeit zwischen Testlauf 2 und 3 für Projekte von 12 Sekunden auf 55 Sekunden, eine Vervierfachung wäre aber nur 48 Sekunden. Da die Messungen zu einem sehr großen Teil von den zufällig generierten Daten abhängen, nicht exakt durchführbar und vor allem nicht wiederholbar sind, wird im nächsten Schritt die maximale Laufzeit theoretisch abgeschätzt.

Test-Nr.	Anzahl Nodes			Dauer in Sekunden		
	Projekte	Artefakte	Mitarbeiter/innen	Projekte	Artefakte	Mitarbeiter/innen
1	250	500	1.000	1,8-2,8	0,5-0,8	1,2-1,6
2	1.000	2.000	4.000	7-12	2,5-3,2	4,8-6
3	4.000	8.000	16.000	30-55	9-14	21-28

Tabelle 2: Ergebnisse des Lasttests, durchgeführt am Szenario.

4.4 Theoretische Schätzung der Laufzeit zur Berechnung der Ähnlichkeit

Die Laufzeit der Ähnlichkeitsberechnung zwischen einem Fall ohne Referenzen auf andere Inhalte und allen anderen Fällen in der Fallbasis ist abhängig von:

- Anzahl der Attribute a
- Anzahl der Fälle in der Fallbasis n

Da die Anzahl der Attribute jedoch nach der Modellierung meist konstant bleibt, ist die Laufzeit in O-Notation:

$$f = O(n)$$

Die Laufzeit hängt also linear von der Anzahl der bereits in der Wissensbasis gespeicherten Fälle ab.

Verweist ein Inhalt auf andere Inhalte, so müssen auch diese Inhalte geladen, zusammengefasst und verglichen werden. Daher ist die Laufzeit der Ähnlichkeitsberechnung zwischen einem Fall mit Referenzen auf andere Inhalte und allen anderen Fällen in der Fallbasis abhängig von:

- Anzahl der Attribute mit Referenz *ar*
- Anzahl der referenzierten Fälle *nr*
- Anzahl der Attribute der referenzierten Felder *anr*
- Anzahl der Fälle in der Fallbasis *n*

Die Anzahl der Attribute *ar* und *anr* wird wieder als konstant betrachtet. Theoretisch kann jeder in der Wissensbasis gespeicherte Fall alle anderen gespeicherten Fälle referenzieren. Die Obergrenze der Laufzeit ist daher:

$$f = O(n^2)$$

In der Praxis dürfte die Laufzeit doch eher linear als quadratisch sein, da die Anzahl der referenzierten Inhalte (zum Beispiel alle Mitglieder eines Teams) eher konstant statt von *n* abhängig sein dürfte.

4.5 Architektur

Das CBR-Modul erweitert Drupal nicht nur um die Fähigkeit, CBR auf Drupal-Installationen auszuführen, sondern nutzt auch eine Reihe von Architekturpattern und Services von Drupal. Diese werden nun genauer untersucht.

Bis zur Version 7 verfolgte Drupal einen prozeduralen Ansatz. Um neue Funktionalität in einem Modul implementieren zu können, wurde stark auf sogenannte Hooks gesetzt. Das sind Methoden, die einer vorgegebenen Namensstruktur folgen müssen (beispielweise cbr_node_insert) und die dann von Drupal oder einem anderen Drupal-Modul zum passenden Zeitpunkt aufgerufen werden. Hooks können als *Decorator* gesehen werden, da beispielsweise die Funktionalität für alle Nodes mit obigem Hook geändert werden kann. Sie können ebenso als *Observer* gesehen werden, da sie beispielsweise bei der Änderung eines Nodes aufgerufen werden. [53] Obwohl Drupal seit

Version 8 vollständig objektorientiert implementiert ist, spielen Hooks noch immer eine wichtige Rolle. Im CBR-Modul sind folgende Hooks implementiert:

- `cbr_node_insert`
- `cbr_node_update`
- `cbr_node_delete`
- `cbr_views_data`
- `cbr_schema`

Die ersten drei Hooks reagieren auf die Änderung von Inhalten („Nodes“) und führen dann eine Berechnung der Ähnlichkeiten durch oder entfernen die errechneten Ergebnisse, wenn der Inhalt gelöscht wird.

Mittlerweile sind Hooks aber nur noch eine von mehreren Möglichkeiten, eigenen Code in Drupal zu integrieren. Drupal ist seit Version 8 zum überwiegenden Teil objektorientiert implementiert. Daraus ergeben sich auch neue Schnittstellen für Entwickler/innen. Drupal erlaubt es Modulen, mit der Plugin-API Funktionalität zur Verfügung zu stellen. Damit Drupal einen Code als Plugin erkennt und ausführt, müssen diese, relativ zum Verzeichnis des Moduls, im Ordner `src/Plugin/<Plugin-Typ>/<Mein-Plugin>.php` angelegt werden und der Namespace des Plugins muss `Drupal\<Modulname>\Plugin\<Plugin-Typ>\<Mein-Plugin>` sein. Damit Drupal das Plugin verarbeiten kann, muss die Klasse noch entsprechend annotiert werden (siehe Listing 5). Die Annotation enthält weitere Metadaten des Plugins. Im CBR-Modul werden Plugins der Typen „FieldFormatter“, „FieldType“ und „FieldWidget“ hinzugefügt, um Felder mit Funktionen zur Ähnlichkeitsberechnung auszustatten. Da Plugins objektorientiert sind, werden die CBR-Felder als Unterklassen der entsprechenden Felder im Drupal-Core angelegt und erben somit einen Großteil deren Funktionalität. Nur bestimmte Funktionen, wie etwa jene zur Erstellung des Formulars der Feldeinstellungen, werden überschrieben.

```
1.  /**
2.   * Defines the 'cbr_string' field type.
3.   *
4.   * @FieldType(
5.   *   id = "cbr_string",
6.   *   label = @Translation("CBR Text (plain)"),
7.   *   description = @Translation("A field containing a long string value."),
8.   *   category = @Translation("Case-Based Reasoning"),
9.   *   default_widget = "cbr_string_textfield",
10.  *   default_formatter = "cbr_basic_string",
11.  *   cardinality = 1
12.  * )
13.  */
14. class CBRString extends StringItem implements CBRFieldInterface { … }
```

Listing 5: Mittels Annotationen werden in Drupal Eigenschaften der Feldtypen definiert.

Eine weitere Möglichkeit, wie Module Funktionalität zur Verfügung stellen können, sind Services. Services können auf Ereignisse reagieren, wie etwa auf einen Seitenaufruf. Drupal verwaltet Service-Objekte selbstständig in einem Service-Container. Services können auf andere Services über Dependency Injection zugreifen. Damit Drupal Code als Service erkennen und verwalten kann, muss der Service in der service.yml im Root-Verzeichnis des Moduls konfiguriert werden.

Im CBR-Modul wird ein Service verwendet, um herauszufinden, ob ein/e Benutzer/in auf den Tab „Similar Cases“ zugreifen darf. Wenn kein Zugriff erlaubt ist, sperrt Drupal nicht nur den Pfad, sondern blendet auch den Tab aus. Zuerst muss der Service in der service.yml konfiguriert werden (siehe Listing 6). Soll ein anderer Service per Dependency Injection eingefügt werden, muss dies unter „arguments“ angegeben werden. Tags geben die Art des Services an sowie wie und wo der Service von Drupal verwendet wird.

```
1.   cbr.access_checker:
2.     class: Drupal\cbr\Access\SimilarCasesAccessCheck
3.     arguments:
4.       - '@current_route_match'
5.       - '@entity_type.manager'
6.       - '@logger.factory'
7.     tags:
8.       - { name: access_check, applies_to: _access_similar_cases_tab
```

Listing 6: Definition und Konfiguration des Services „cbr.access_checker“ in der service.yml des CBR-Modul.

Der Service wird in der Datei SimilarCasesAccessCheck.php implementiert. Im Konstruktor werden die Services „`current_route_match`“, „`entity_type.manager`“ und „`logger.factory`” injiziert, sie können anschließend verwendet werden (siehe Listing 7).

```
1.  class SimilarCasesAccessCheck implements AccessInterface
2.  {
3.      protected RouteMatchInterface $routeMatch;
4.      protected EntityTypeManagerInterface $entityTypeManager;
5.      protected LoggerChannelInterface $logger;
6.
7.      public function __construct(
8.          RouteMatchInterface $routeMatch,
9.          EntityTypeManagerInterface $entityTypeManager,
10.         LoggerChannelFactoryInterface $loggerFactory
11.     ) {
12.         $this->routeMatch = $routeMatch;
13.         $this->entityTypeManager = $entityTypeManager;
14.         $this->logger = $loggerFactory->get('cbr');
15.     }
```

Listing 7: Beispiel einer Dependency Injection im Konstruktor in Drupal 9.

Aus einem statischen Kontext, wie beispielsweise aus Hooks, können Services über `\Drupal::service('service.name')` injiziert werden. Ein Vorteil der Dependency Injection gegenüber der direkten Initialisierung ist die Austauschbarkeit. Dies erleichtert beispielsweise das Schreiben von

Unittests, da eingebundene Services sehr einfach durch gemockte Versionen des Services ersetzt werden können.

Eine weitere Möglichkeit, wie Module den Funktionsumfang einer Drupal-Seite erweitern können, sind YAML-Dateien. Damit können Konfigurationen und Einstellungen von Drupal durch das CBR-Modul gesetzt werden. Im CBR-Modul wird beispielsweise bei der Installation des Moduls die View für die Anzeige ähnlicher Fälle angelegt. Diese und weitere Konfigurationsdateien befinden sich im Ordner `cbr/config/install`.

4.6 Erweiterbarkeit

Das Modul selbst ist, wo immer es möglich ist, objektorientiert umgesetzt. Dabei wird bei der Berechnung der Ähnlichkeit auf Polymorphie gesetzt. Jedes Feld implementiert das Interface „`CBRFieldInterface`“ mit den Methoden „`calculateSimilarity`“, „`getValueForSimilarityCalculation`” und „`summerize`“. Je nach Feldtyp sind diese sehr unterschiedlich implementiert. In der Methode zur Berechnung der globalen Ähnlichkeit werden diese dann aufgerufen, wodurch die Berechnung der lokalen Ähnlichkeit dem Feld überlassen wird. Dieser Ansatz ermöglicht nicht nur eine klare Struktur des Moduls selbst, sondern stellt auch sicher, dass andere Module ebenfalls Felder zur Verfügung stellen können, die CBR unterstützen. Dazu muss lediglich das „`CBRFieldInterface`“ implementiert werden und das CBR-Modul ruft dann bei der Berechnung der Ähnlichkeit die entsprechenden Methoden auf. Im Anhang befindet sich der Code für ein Modul, das CBR für Felder mit booleschen Werten implementiert.

Damit wird das CBR-Modul zur Basis für CBR-Projekte aller Art, da beliebige Feldtypen zur Berechnung der Ähnlichkeit verwendet werden können. Ebenso kann der Algorithmus zur Berechnung der lokalen Ähnlichkeit auf diese Weise überschrieben werden. Das CBR-Modul ermöglicht so weitere Entwicklungen und Forschungen zum Thema Case-Based Reasoning.

4.7 Weitere Einsatzmöglichkeiten

Bisher wurde in dieser Diplomarbeit hauptsächlich untersucht, wie mit Hilfe von Content-Management-Systemen bessere Systeme für Case-Based Reasoning realisiert werden können. Doch auch umgekehrt können Content-Management-Systeme und der Task des Content-Managements von den im CBR-Modul umgesetzten Algorithmen profitieren. Der Vergleich ganzer Inhalte über alle Felder hinweg ist ein bislang kaum beachteter Aspekt des Content-Managements. Eine kurze Recherche zeigt, dass es hierfür auf Drupal.org keine Module gibt. Zwar gibt es das Modul „Similar

Entries[30]“ für Drupal 7, dieses kann jedoch nur einzelne Textfelder vergleichen. Die Module „Similar By Terms[31]“ und „Taxonomy Similar[32]“ verwenden hingegen Taxonomien, können jedoch wiederum keine Felder anderer Art nutzen.

Das Auffinden möglichst ähnlicher Inhalte zu beliebigen Inhalten kann im Content-Management bei der Erstellung neuer Inhalte hilfreich sein, etwa wenn am Ende eines Artikels ähnliche Artikel verlinkt werden, wie es beispielsweise häufig bei Nachrichtenseiten der Fall ist. Auch in Onlineshops werden häufig ähnliche Produkte unter einem Produkt angezeigt, auch hier kann das CBR-Modul eingesetzt werden. Für die Suchmaschinenoptimierung kann es sinnvoll sein, die Ähnlichkeit zu anderen Inhalten abzufragen, um ähnliche oder gar doppelte Inhalte zu vermeiden.

Um das CBR-Modul für Anwendungsfälle außerhalb des CBR Cycles zu verwenden, ist keine Änderung am Modul nötig. Der Tab „Ähnliche Fälle“ kann über Drupal-Views beliebig angepasst werden. So kann etwa der Button „Select as solution“ ausgeblendet werden, wenn er nicht benötigt wird. Da das CBR-Modul Views vollständig unterstützt, kann ohne Programmierung beispielsweise für einen Onlineshop ein Block mit den ähnlichsten Produkten erstellt werden. Dieser wird dann automatisch mit den passenden Inhalten gefüllt.

4.8 Minimierung der Verzögerung bei der Berechnung ähnlicher Fälle

4.8.1 Berechnung der Ähnlichkeit via Cron

Wie zuvor erwähnt, kann das CBR-Modul auch so eingesetzt werden, dass es ausreichend ist, wenn die Ähnlichkeit zwischen dem aktuell bearbeiteten Fall und allen weiteren Fällen zu einem späteren Zeitpunkt erfolgt. Die Analyse des Prototyps hat darüber hinaus gezeigt, dass bei der Verwendung des CBR-Moduls im CBR Cycle nicht immer eine sofortige Ähnlichkeitsberechnung erfolgen muss. Beispielsweise werden im Schritt „Retain“ dem bereits abgeschlossenen Fall Attribute hinzugefügt, die den Fall besser beschreiben und somit leichter auffindbar machen. Wird der Fall gespeichert, ist es nicht zwingend notwendig, dass die Ähnlichkeiten zu anderen Fällen sofort berechnet werden müssen. Es sollte aber zeitnah erfolgen, damit zu jedem Fall die aktuell ähnlichsten Fälle angezeigt werden. Um dies umzusetzen, wird das Formular zum Erstellen und Bearbeiten von Inhalten um den Button „Save and calculate similarity“ (siehe Abbildung 25) ergänzt. Beim Klick auf diesen Button werden die Ähnlichkeiten zwischen diesem Fall und anderen Fällen sofort berechnet,

[30] https://www.drupal.org/project/similar, abgerufen am 06.06.2022
[31] https://www.drupal.org/project/similarterms, abgerufen am 06.06.2022
[32] https://www.drupal.org/project/taxonomy_similar, abgerufen am 06.06.2022

der/die Nutzer/in muss warten, kann danach aber sofort die ähnlichsten Fälle ansehen. Dieser Button erscheint nur bei Inhaltstypen, die CBR-Felder enthalten.

Abbildung 25: Buttons zum Speichern der Änderungen beim Bearbeiten eines Falls. Die Berechnung der Ähnlichkeiten zu allen anderen Fällen kann wahlweise sofort oder verzögert im Hintergrund erfolgen. Eigene Aufnahme.

Wird der Fall über den „Save"-Button gespeichert, so erfolgt keine sofortige Berechnung der Ähnlichkeiten. Der/die Nutzer/in wird damit nicht im Arbeitsfluss gestoppt, sondern kann sofort weiterarbeiten und etwa andere Inhalte editieren. Die ID dieses Inhalts wird jedoch in der Tabelle „cbr_calculation_queue" gespeichert, die vom CBR-Modul in der Datenbank des Content-Management-Systems angelegt wird.

Um Aufgaben regelmäßig im Hintergrund auszuführen, verwendet Drupal, wie andere Content-Management-Systeme auch, sogenannte Cronjobs. Cronjobs (von chronos, griechisch für „die Zeit") sind in Linux und anderen Unix-ähnlichen Systemen eine Möglichkeit, wiederkehrende Aufgaben zu einem bestimmten Zeitpunkt oder nach einer bestimmten Zeitspanne auszuführen. Im Wesentlichen wird in einer Tabelle, der sogenannten „Crontab", in einer bestimmten Syntax festgelegt, wann ein beliebiger Befehl ausgeführt werden soll. Dieser Befehl kann auch der Aufruf einer Webseite sein, in diesem Fall die cron.php einer Drupal-Installation. In Drupal werden eine Reihe von Aufgaben durchgeführt, welche nicht an einen bestimmten Seitenaufruf gebunden sind. Dazu gehören etwa das Aktualisieren des Suchindexes, das Prüfen auf Updates oder das Löschen von Logs. Sie alle werden beim Aufruf der cron.php ausgeführt.

Module wie das CBR-Modul können dieses System ebenfalls nutzen. Dazu implementiert das CBR-Modul den Hook „cron", welcher beim Aufruf der cron.php ausgeführt wird. In dieser Funktion werden aus der Tabelle „cbr_calculation_queue" nacheinander alle Fälle, welche auf eine neue Berechnung der Ähnlichkeiten warten, geladen und anschließend wird mit dem „cbr.similarity_calculator"-Service die Berechnung der Ähnlichkeiten vorgenommen. Die berechneten Ähnlichkeiten werden anschließend in der Datenbank gespeichert und können damit über den Tab „Similar Cases" eingesehen werden. Abschließend wird die ID des Inhalts, für den die Ähnlichkeiten berechnet wurden, aus der Tabelle „cbr_calculation_queue" entfernt. Da die Laufzeit der cron.php beschränkt ist, werden pro Aufruf die Ähnlichkeiten von maximal 100 Fällen zu allen anderen Fällen in der Wissensbasis berechnet. Je nach Anzahl der Fälle in der Wissensbasis, deren Struktur

und der verwendeten Hardware kann diese Zahl natürlich nach oben hin oder nach unten hin angepasst werden.

Damit ist es möglich, dass die Berechnung der Ähnlichkeiten vollständig unbemerkt von dem/der Benutzer/in erfolgt. Nachteilig ist, dass, je nach Konfiguration der Crontab, nicht immer aktuelle Ähnlichkeiten angezeigt werden. Für viele bestimmte Anwendungsfälle ist dies aber kein Problem. Wenn beispielsweise in einem Onlineshop für wenige Minuten nicht die ähnlichsten Produkte unter einem Produkt angezeigt werden, sondern Produkte, die vor der Änderung der Attribute dem Produkt am ähnlichsten waren, dann wird dies vermutlich kaum auffallen. Damit ist es möglich, das CBR-Modul ohne Wartezeiten und Einschränkungen im Workflow zu nutzen.

4.8.2 Verwendung der Cache-API

Da sich die Ähnlichkeiten nicht immer im Hintergrund berechnen lassen, zum Beispiel, wenn der/die Nutzer/in sofort die ähnlichsten Fälle einsehen möchte, wurde aufbauend auf der Evaluierung noch eine Verbesserung bei der Berechnung der Ähnlichkeiten vorgenommen. Wie der Lasttest gezeigt hat, kann die Berechnung der Ähnlichkeiten bei Attributen, die auf andere Inhalte verweisen, deutlich länger dauern, da diese Inhalte ebenfalls aus der Datenbank geladen werden müssen. Wenn also ein Projekt auf 50 Teammitglieder verweist, dann müssen diese bei der Berechnung der Ähnlichkeit des Projektes ebenfalls geladen werden. Wie in Abschnitt 3.4 beschrieben, werden die Eigenschaften der Teammitglieder anschließend zusammengefasst. Wird nun ein neues Projekt angelegt und dieses mit dem bereits in der Datenbank vorhandenen Projekt verglichen, so ist es wahrscheinlich, dass sich an der Zusammensetzung des Projektteams des bereits gespeicherten Projektes nichts geändert hat und alle Felder der einzelnen Projektmitglieder unverändert geblieben sind. Damit sind die zusammengefassten Eigenschaften des Teams unverändert, müssen aber erneut gebildet werden. Um dies zu vermeiden, wird für die Speicherung der zusammengefassten Eigenschaften die Cache-API von Drupal verwendet. Diese erlaubt das Speichern von beliebigen Objekten unter einem frei wählbaren Cache-Key. Werden nun die zusammengefassten Werte zur Berechnung der Ähnlichkeit benötigt, so wird zunächst der Cache-Key gebildet. In diesem Fall setzt sich der Cache-Key aus den Identifikatoren der referenzierten Inhalte zusammen. Wenn nun im Cache ein Wert für diesen Cache-Key existiert, so wird dieser verwendet. Existiert kein Wert im Cache, so wird der bisherige Algorithmus ausgeführt. Es werden also alle referenzierten Inhalte geladen und zusammengefasst. Damit dies bei künftigen Berechnungen nicht erneut geschehen muss, wird der zusammengefasste Wert im Cache unter dem zuvor erzeugten Cache-Key gespeichert. Was aber, wenn sich die referenzierten Inhalte ändern? Dazu bietet die Cache-API von Drupal sogenannte Cache-Tags an. Für Nodes bestehen diese aus der Zeichenkette

„node:“, gefolgt vom Identifikator des Nodes. Wird nun der Inhalt des entsprechenden Nodes geändert, so wird der Cache-Key automatisch gelöscht. Es können beliebig viele Cache-Tags zu einem Cache-Key hinzugefügt werden. Im CBR-Modul werden alle referenzierten Inhalte als Tags hinzugefügt. Ändert sich einer dieser Inhalte, so wird der Cache-Key gelöscht, womit alle referenzierten Daten erneut aus der Datenbank geladen werden. Damit ist sichergestellt, dass die Ähnlichkeitsberechnung immer mit den aktuellen Daten erfolgt. Auch wenn diese Verbesserung nichts an der maximalen Laufzeit ändert, da hierfür von einem leeren Cache ausgegangen werden muss, so verkürzt sich die Dauer der Berechnung der Ähnlichkeiten, sofern der Cache schon befüllt ist, deutlich. Bei einer erneuten Durchführung des in Abschnitt 4.3 beschriebenen Stresstests mit 4.000 Projekten, 8.000 Artefakten und 16.000 Mitarbeiter/innen sinkt die Laufzeit beim Berechnen der Ähnlichkeit zwischen einem Projekt und allen anderen Projekten auf unter 10 Sekunden für alle getesteten Durchläufe. Beim Lasttest mit den gleichen Testdaten ohne Caching-API liegt die Laufzeit zwischen 30 Sekunden und 55 Sekunden (vgl. Tabelle 2).

5 Diskussion

In dieser Diplomarbeit wurde der Prototyp eines CBR-Systems, welcher auf dem Open-Source-Content-Management-System Drupal 9 basiert, umgesetzt. Die Vor- und Nachteile des gewählten Ansatzes im Vergleich zu bestehenden CBR-Systemen wie myCBR und JColibri können aus drei Perspektiven diskutiert werden: Aus der Sicht der/des Anwenderin/Anwenders, welche/r neue Fälle in das CBR-System einträgt, nach ähnlichen Fällen sucht und eine Lösung auswählt; aus der Sicht einer/eines Systemintegratorin/Systemintegrators, welche/r, aufbauend auf der bestehenden CBR-Software, ein neues CBR-System für einen bestimmten Anwendungsfall oder ein bestimmtes Unternehmen erstellt; sowie aus der Sicht einer/eines Softwareentwicklerin/Softwareentwicklers, welche/r das bestehende CBR-System weiterentwickelt, indem sie/er etwa neue Attributtypen hinzufügt oder Algorithmen zur Ermittlung der Ähnlichkeit implementiert.

Im Gegensatz zu myCBR und JColibri bietet der umgesetzte Prototyp eine Weboberfläche für alle Aufgaben, die bei der Nutzung des Systems anfallen. Damit können mehrere Benutzer/innen das System gleichzeitig bedienen und auf eine geteilte Wissensbasis zugreifen. myCBR und JColibri sind hingegen Java-Applikationen, die mit den Tools myCBR Workbench und Colibri Studio auf dem eigenen Rechner bedient werden können. Diese Tools richten sich jedoch mehr an Programmierer/innen oder mit CBR sehr erfahrene Anwender/innen und dienen mehr zum Prototyping eines CBR-Systems als zur direkten Verwendung. Um myCBR und JColibri produktiv einsetzen zu können, muss daher noch eine Benutzeroberfläche geschrieben werden. Dazu kann entweder direkt über das SDK auf die Schnittstellen zugegriffen werden oder bei myCBR eine REST-Schnittstelle verwendet werden [54]. Damit können auch diese CBR-Systeme etwa über eine Weboberfläche bedient werden. Diese muss jedoch erst programmiert werden und kann daher nicht evaluiert und verglichen werden. Umgekehrt bietet der Prototyp eine sofort verwendbare, webbasierte Benutzeroberfläche sowie darüber hinaus die Möglichkeit, mit dem Drupal REST-Modul[33], welches Teil des Drupal-Cores ist, ebenfalls eigene Benutzeroberflächen und weitere Tools einzubinden.

Die Weboberfläche von Drupal für die Erstellung von Fällen unterscheidet sich nicht von der, die für die Erstellung anderer Inhalte verwendet wird. Damit kann die Wissensbasis auch von Anwender/innen gepflegt und genutzt werden, die möglicherweise mit dem Thema CBR nicht vertraut sind. Die Lösung eines Falls kann unterschiedlichste Gestaltungsformen haben, von Texten, Dokumenten, Tabellen und Präsentationen bis hin zu Links auf externe Ressourcen.

[33] https://www.drupal.org/docs/8/core/modules/rest, abgerufen am 10.06.2022

Bei der Generierung einer Lösung für einen neuen Fall erlaubt der Prototyp eine hohe Flexibilität, indem die Lösung aus verschiedenen anderen Fällen übernommen werden kann. Wie bei myCBR und JColibri erfolgt jedoch keine automatische Anpassung der Lösung an den aktuellen Fall. Dies könnte jedoch in einem weiteren Projekt oder einer wissenschaftlichen Arbeit erfolgen. Sowohl erfolgreiche als auch nicht erfolgreiche Lösungen können im Prototyp entsprechend gespeichert werden. Der Prototyp unterstützt somit alle vier Schritte des CBR Cycles.

In dem umgesetzten Prototyp werden nach der Erstellung beziehungsweise der Änderung eines Falls die Ähnlichkeiten zu allen anderen Fällen berechnet. Diese Vorgehensweise unterscheidet sich von myCBR und JColibri, dort werden die Ähnlichkeiten erst dann berechnet, wenn der/die Anwender/in explizit nach ähnlichen Fällen sucht. Die Ähnlichkeiten werden in myCBR und JColibri nicht zwischengespeichert. Der im Prototyp gewählte Ansatz hat folgende Vorteile gegenüber den von myCBR und JColibri gewählten Ansätzen folgende Vorteile: Es kann davon ausgegangen werden, dass die Attribute eines Falls weniger häufig geändert werden, als dass ähnliche Fälle zu einem Fall angezeigt werden, womit die Berechnung im Prototyp seltener durchgeführt werden muss als bei myCBR und JColibri. Zudem kann, wie im Prototyp umgesetzt, die Berechnung der Ähnlichkeit im Hintergrund erfolgen, wenn nicht sofort ähnliche Fälle ermittelt werden müssen. Wird so vorgegangen, kommt es für den/die Benutzer/in zu keiner Verzögerung bei der Bedienung des CBR-Systems, weder bei der Speicherung eines neuen Inhalts, noch bei der Anzeige der ähnlichen Fälle. Nachteilig ist jedoch, dass für die vorberechneten Ähnlichkeiten eine eigene Tabelle benötigt wird und dass die gespeicherten Ähnlichkeiten für einen bestimmten Zeitraum möglicherweise nicht mit den aktuellen Fallattributen übereinstimmen. Dies kann passieren, wenn der/die Benutzer/in ein Attribut im Fall ändert und dann den Fall ohne Berechnung der Ähnlichkeiten speichert und noch kein Cron-Lauf durchgeführt wurde. Dieses Problem kann jedoch durch eine korrekte Bedienung und eine weitere Verbesserung des Prototyps vermieden werden. Dies könnte dadurch erreicht werden, dass zu jeder berechneten Ähnlichkeit ein Zeitstempel gespeichert wird, der mit dem Zeitstempel der letzten Änderung des Falls verglichen wird und gegebenenfalls eine Warnung angezeigt wird.

CBR-Systeme wie das im Prototyp vorgestellte System, aber auch myCBR und JColibri müssen für jeden Anwendungsfall entsprechend konfiguriert werden. Dazu gehören die Erstellung der Fallstruktur, das Hinzufügen von Attributen und die Konfiguration der Ähnlichkeitsberechnung. Dabei können etwa die Gewichtungen der Attribute angepasst oder aus verschiedenen Funktionen zur Berechnung der lokalen Ähnlichkeit eine geeignete ausgewählt werden. Diese Anpassung kann ein/e Systemintegrator/in übernehmen. Der Prototyp bietet hier eine webbasierte Oberfläche, in

der Attribute unterschiedlicher Datentypen zu einem Fall hinzugefügt und deren Gewichtung und Funktion zur Berechnung der Ähnlichkeit ausgewählt werden können. Die Erstellung einer Fallstruktur ähnelt der Erstellung neuer Inhaltstypen in Drupal 9, folgt also einem vielfach genutzten und getesteten Prinzip, mit zusätzlichen Einstellungsmöglichkeiten für Case-Based Reasoning. Die in den Fällen gespeicherten Daten werden dabei immer in der Datenbank der Drupal-Installation gespeichert. Drupal erstellt dabei alle Tabellen und Spalten, die zur Speicherung der Daten benötigt werden. Das so konfigurierte System kann anschließend entweder direkt verwendet werden oder über die Konfigurationssynchronisation von Drupal exportiert werden und in andere Drupal-Installationen importiert werden. Ebenso kann aus den so erzeugten YAML-Dateien einfach ein neues Drupal-Modul erstellt werden. Bei myCBR und JColibri müssen für die Systemkonfiguration die jeweiligen Entwicklungstools verwendet werden, bei denen es sich um Desktop-Anwendungen handelt. Beide Tools bieten eine grafische Oberfläche zum Erstellen einer Fallstruktur, zum Hinzufügen von Attributen und zum Auswählen von Funktionen zur Berechnung der lokalen Ähnlichkeit. Dabei legt myCBR jedoch nur eine Datenstruktur zur Verarbeitung der Daten an. Die Speicherung der Daten muss in einer eigenständigen CSV-Datei erfolgen. JColibri Studio generiert aus der konfigurierten Datenstruktur Java-Klassen. Wie die Daten geladen werden, muss der/die Systemintegrator/in selbst implementieren. Ebenso muss für beide CBR-Systeme eine Benutzeroberfläche implementiert werden, wenn das CBR-System unabhängig von den Entwicklungstools betrieben werden soll.

myCBR und JColibri bieten eine Vielzahl von Funktionen zur Berechnung der lokalen Ähnlichkeit sowie eine große Zahl an Konfigurationsmöglichkeiten. Dabei können etwa für Taxonomien die Ähnlichkeiten zwischen zwei Begriffen direkt in eine Tabelle eingetragen werden. Die im Prototyp umgesetzten Funktionen zur Berechnung der lokalen Ähnlichkeiten erlauben zwar das vollständige Umsetzen einer CBR-Lösung, die Texte, Zahlen und Taxonomien vergleichen kann, bieten aber nicht die Tiefe der Konfigurationsmöglichkeiten, die mit den beiden anderen CBR-Systemen möglich sind. Wird für einen konkreten Anwendungsfall eine weitere Funktion zur Berechnung der lokalen Ähnlichkeit oder eine weitere Einstellungsmöglichkeit benötigt, so erlaubt es der Prototyp, diese relativ einfach in einem neuen Drupal-Modul zu implementieren, wobei dieses dann auf die Schnittstellen des CBR-Moduls zugreifen kann.

Durch die Integration von CBR in ein CMS ist es sehr einfach möglich, das Design des Systems an etwa das Unternehmen, in dem das System eingesetzt wird, anzupassen. Da die Darstellung aller Eingabeelemente des CBR-Prototyps durch Drupal erfolgt, kann aus einer Vielzahl von Themes gewählt werden, die dann individuell angepasst werden können. Auch wenn dies in dieser Arbeit

nicht näher betrachtet wird, dürfte eine gewohnte Benutzeroberfläche für Anwender/innen zu einer höheren Akzeptanz des CBR-Systems führen.

Um die bestehenden Systeme weiterzuentwickeln oder Fehler zu beheben, können Softwareentwickler/innen bei allen drei Systemen auf bekannte Architekturen und Entwurfsmuster zurückgreifen. myCBR und JColibri sind als Java-Applikationen vollständig objektorientiert programmiert. Das CBR-Modul, welches in PHP geschrieben ist, einer Programmiersprache, die in der ersten Version noch gar keine objektorientierte Programmierung erlaubte, ist überwiegend objektorientiert umgesetzt. Dies ist möglich, da Drupal in der Version 8 zu einem großen Teil neu und objektorientiert implementiert wurde. Die objektorientierte Implementierung von Drupal und all seinen Modulen ermöglicht eine saubere Struktur mit einer „Separation of Concerns" in Klassen und Namespaces. Drupal setzt stark auf Services, die mittels Dependency Injection eingebunden werden. Das Prinzip ähnelt damit in etwa dem der Beans in Spring. Damit unterscheidet sich eine Programmierung für Drupal, welches das Framework „Symphony" verwendet, kaum mehr von einer Programmierung für andere Frameworks wie etwa Spring oder Vaadin. Während das Dependency Management von Java-Projekten mit Maven oder Gradle durchgeführt werden kann, nutzt Drupal Composer. Damit können Abhängigkeiten zu anderen Modulen und Bibliotheken einfach verwaltet werden, was für Softwareentwickler/innen eine große Zeitersparnis bedeutet.

Während myCBR und JColibri eigenständige Applikationen sind, muss das in dieser Arbeit entwickelte Modul stets an die aktuelle Drupal-Version angepasst werden. Dabei ist jedoch hilfreich, dass seit Drupal 8 Module, die keine als „deprecated" gekennzeichneten Schnittstellen verwenden, automatisch mit der nächsten Version von Drupal kompatibel sind. Das CBR-Modul sollte daher auch mit der im Dezember 2022 erscheinenden Drupal-Version 10[34] lauffähig sein.

Der Prototyp ist aufgrund seiner Abhängigkeit von Drupal in PHP geschrieben. Während alle Features umgesetzt werden konnten, ist anzumerken, dass PHP eine eher unübliche Wahl für ein KI-System ist. Für weitere Verbesserungen, etwa beim Textvergleich, stehen daher kaum fertige Bibliotheken zur Verfügung, wie dies etwa für Python der Fall ist. Ein interessanter Ansatz für ein zukünftiges Forschungsprojekt könnte daher der in dieser Diplomarbeit gewählte Ansatz für ein in Python geschriebenes Content-Management-System sein.

JColibri bietet mit dem JColibri-Studio eine auf Eclipse basierende Entwicklungsumgebung für die Programmierung von CBR-Anwendungen auf JColibri-Basis. Für myCBR und auch Drupal stehen

[34] https://www.drupal.org/about/10, abgerufen am 11.06.2022

aber ebenso umfangreiche Entwicklungsumgebungen zur Verfügung, wie etwa Intellij für myCBR und PhpStorm für Drupal. Insgesamt ist die Entwicklung für Drupal damit den beiden anderen Systemen sehr ähnlich, auch wenn sich die Programmiersprachen unterscheiden. Dies ist primär auf die neue objektorientierte Architektur von Drupal zurückzuführen.

Die Idee, Case-Based Reasoning und Content-Management zu kombinieren, ist nicht vollständig neu. Im Jahr 2016 veröffentlichte *Dorn* einen Artikel, in dem Projektwissen mit Hilfe eines mit Drupal umgesetzten CBR-Systems geteilt wird. Diese Diplomarbeit baut auf den Grundideen dieses Artikels auf, der Prototyp wurde jedoch komplett neu implementiert. In dem Artikel von *Dorn* wird Drupal 7 genutzt [55, S. 10], welches noch nicht objektorientiert ist. Damit aktualisiert diese Diplomarbeit das vorgestellte Konzept um eine moderne Implementierung und verbessert dieses. Im Artikel wird als mögliche weitere Untersuchung eine bessere Implementierung der Errechnung der Ähnlichkeit genannt, da diese bei vielen Fällen in der Wissensbasis langsam ist. Wie in dieser Arbeit gezeigt werden konnte, kann der implementierte Prototyp die Ähnlichkeit eines Falls zu tausenden von anderen Fällen in akzeptabler Zeit bestimmen. Für eine weitere Steigerung der Geschwindigkeit bei der Berechnung der Ähnlichkeiten sorgt die Nutzung der Caching-API von Drupal sowie die Möglichkeit, die Berechnung im Hintergrund erfolgen zu lassen.

Weitere Verbesserungs- und Forschungsmöglichkeiten dieses Prototyps liegen in der Berechnung der Ähnlichkeit der einzelnen Felder. Bei Texten wird etwa der gesamte Text ohne Aufarbeiten mit Hilfe des Jaccard-Koeffizienten und der Kosinus-Ähnlichkeit verglichen. Um hier das Ergebnis zu verbessern, könnte man Wörter vor dem Vergleich auf ihren Wortstamm zurückführen. Durch dieses als „Stamming“ bezeichnete Verfahren wird sichergestellt, dass etwa Wörter im Plural und im Singular auf einen einheitlichen Stamm zurückgeführt werden und damit vom CBR-System als gleich erkannt werden. Eine weitere Verbesserung könnte das Filtern des Textes von sogenannten „Stopwords“ sein. Damit sind Wörter wie „der“, „im“ und „ist“ gemeint, die für den Sinn des Textes nicht notwendig sind und die Ergebnisse der implementierten Ähnlichkeitsfunktionen verfälschen.

Auch wenn die Geschwindigkeit der Ähnlichkeitsberechnung in dieser Arbeit bereits stark optimiert wurde, kann diese durch den Einsatz der Cache-API weiter optimiert werden. So könnten die Daten der Felder, die zur Berechnung der Ähnlichkeit verwendet werden, von mehreren Fällen in einem Objekt zusammengefasst werden und im Cache gespeichert werden. Damit müsste nicht für jeden Fall eine Datenbankabfrage gemacht werden, ohne dass auf die Vorteile der Speicherung der Daten durch Drupal verzichtet werden muss.

6 Conclusio

Ähnliche Probleme mit ähnlichen Lösungen zu lösen [6, S. 3] ist der Grundgedanke von Case-Based Reasoning. Case-Based Reasoning ist ein maschinelles Lernverfahren, bei dem Lösungen aus einer Wissensbasis von bereits gelösten Fällen generiert werden. Zu den Pionieren zählen *Riesbeck* und *Schank*, die den Begriff „Case-Based Reasoning" prägten [5, S. 11], sowie *Aamodt* und *Plaza*, die 1994 den CBR Cycle vorstellten [8, S. 45]. Während sich die aktuelle Forschung zum Thema CBR meist mit neuen Einsatzmöglichkeiten von CBR beschäftigt [21, S. 64ff., 22, S. 232ff., 24, 1-9, 25, S. 94ff.], wird in dieser Arbeit Case-Based Reasoning mit Content-Management kombiniert und untersucht, wie diese beiden bislang kaum zusammen betrachteten Domänen der Informatik voneinander profitieren können.

Content-Management wird durch Content-Management-Systeme unterstützt. Diese erlauben das Organisieren von Informationen aller Art [32, S. 10, 34, S. 10, 35, S. 1]. Mit Hilfe von Case-Based Reasoning kann das in diesen Systemen gespeicherte Wissen zur Lösung neuer Problemstellungen genutzt werden. Der umgesetzte Prototyp kann in eine bestehende Installation eines CMS integriert werden und die gespeicherten Daten nutzen.

Forschungsfrage 1 kann dazu wie folgt beantwortet werden: Während Fälle in CBR-Systemen aus Attributen zusammengesetzt sind, speichern Content-Management-Systeme Daten in Datenstrukturen aus Feldern ab. Diese Felder werden im Prototyp um Funktionen zur Berechnung der Ähnlichkeit erweitert und werden dadurch zu Attributen eines Falls. Unterstützt wird dies durch die objektorientierte Umsetzung des Content-Management-Systems und dessen Schnittstellen für Plugins und Module.

Daraus ergeben sich folgende Vorteile: Vorhandene Daten können einfach in das CBR-System integriert werden. Es ist keine doppelte Datenhaltung erforderlich und die Daten, mit denen das CBR-System arbeitet, sind immer aktuell. Die Erstellung eines Falls unterscheidet sich nicht von der Erstellung anderer Inhalte im CMS. Damit kann das System ohne Einschulung von allen bisherigen Benutzer/innen weiterverwendet werden, welche so die Wissensbasis aufbauen und aktualisieren. Das CBR-System profitiert von folgenden Features des CMS: Speicherung und Anzeige von Fällen, Attribute können über eine vertraute Benutzeroberfläche zu Fällen hinzugefügt und geändert werden, die Darstellung ähnlicher Fälle zu einem Fall erfolgt durch das CMS und ist ohne Programmierung anpassbar, das CMS übernimmt die Verwaltung von Benutzer/innen, Rechten und Rollen und die Konfiguration des Systems kann exportiert und importiert werden. Im Vergleich zu Systemen wie myCBR und JColibri ist der große Vorteil, dass das CMS und damit auch

der Prototyp eine Benutzeroberfläche mitbringen, die einerseits einfach an das einsetzende Unternehmen oder die einsetzende Organisation angepasst werden kann, andererseits aber auch ohne Änderungen sofort verwendet werden kann. Entwickler/innen profitieren von der objektorientierten Architektur und einer großen Community des CMS. Der Funktionsumfang des Systems lässt sich zudem durch Module von Dritten erweitern, etwa kann der Prototyp ohne Programmierung um eine REST-Schnittstelle erweitert werden. Umgekehrt profitiert das Content-Management-System von der Möglichkeit, Inhalte über ihre gesamte Struktur hinweg mit anderen Inhalten im System zu vergleichen. Damit lässt sich zum Beispiel eine Duplikaterkennung durchführen oder die Anzeige ähnlicher Produkte in einem Webshop zum aktuell betrachteten Produkt.

Nachteilig ist, dass bei jeder Änderung eines Inhalts die Ähnlichkeiten zu allen anderen Inhalten erneut berechnet werden müssen. Dies kann für den/die Benutzer/in zu Wartezeiten und Unterbrechungen im Workflow führen. Um die Auswirkungen zu reduzieren, kann in Anwendungsfällen, in denen die Berechnung der Ähnlichkeiten nicht unmittelbar nach dem Speichern erforderlich ist, die Berechnung der Ähnlichkeiten im Hintergrund erfolgen. Ein weiterer Nachteil ist, dass für alle Felder, für die eine Ähnlichkeitsberechnung durchgeführt werden soll, der Feldtyp geändert werden muss. Hierfür wurde ein Modul entwickelt, das die Daten aus dem bestehenden Feld in das neue Feld kopieren kann. Weiters kann als Nachteil gesehen werden, dass das im Prototyp verwendete Content-Management-System in der Programmiersprache PHP implementiert ist, wodurch auch alle Module in PHP programmiert werden müssen. Im Bereich der künstlichen Intelligenz stehen jedoch für andere Programmiersprachen, wie etwa Python, eine größere Anzahl an Bibliotheken zur Verfügung.

Inhalte können in Content-Management-Systemen auf andere Inhalte verweisen. Diese können ebenfalls zur Berechnung der Ähnlichkeit verwendet werden. Dazu werden diese Inhalte für jedes Feld zusammengefasst und anschließend wird der zusammengefasste Wert zur Berechnung der Ähnlichkeit verwendet. Forschungsfrage 2 lässt sich damit wie folgt beantworten: Da referenzierte Inhalte zur Berechnung der Ähnlichkeit ebenfalls geladen werden müssen, erhöht sich die theoretische, maximale Laufzeit von $O(n)$ auf $O(n^2)$. In der Praxis kann die Laufzeit jedoch durch Caching der zusammengefassten Werte deutlich reduziert werden. Dies konnte am Prototyp mit einer großen Anzahl zufällig generierter Testdaten gezeigt werden.

Der umgesetzte Prototyp wurde mit dem Szenario eines Unternehmens, das Case-Based Reasoning im Projektmanagement einsetzt, getestet. Dabei konnte gezeigt werden, wie bei Beginn eines neuen Projektes auf Wissen aus bereits durchgeführten Projekten zugegriffen werden kann und wie das

bei der Durchführung des aktuellen Projektes gewonnene Wissen für weitere Projekte im System gespeichert werden kann.

In Summe lässt sich festhalten, dass Case-Based Reasoning und Content-Management-Systeme auf den ersten Blick wenig gemeinsam haben. Content-Management-Systeme bieten jedoch mit ihren vielfältigen Möglichkeiten zur Organisation von Inhalten jedoch eine leistungsfähige und flexible Plattform zur Realisierung eines vollständig funktionsfähigen Case-Based-Reasoning-Systems. Content-Management-Systeme erhalten durch Case-Based Reasoning eine mächtige Möglichkeit zum Vergleichen von Inhalten. Moderne objektorientierte Web-Content-Management-Systeme erlauben das Umsetzen von Erweiterungen mit aus anderen Frameworks wie Spring bekannten Konzepten und sind damit eine hervorragende Plattform für zeitgemäße Softwareentwicklung.

LITERATURVERZEICHNIS

[1] C. Eunjin, H. Jongdae und H. Hyuksoo, „Risk Identification Using Case Based Reasoning in Software Project“, *Journal of Software*, Jg. 12, Nr. 9, S. 744–750, 2017.

[2] P. Gomes, J. Cordeiro, P. Gandola und N. Seco, „Software Development Knowledge Management Using Case-Based Reasoning“ in *Soft Computing Applications in Industry*, B. Prasad, Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 271–291, doi: 10.1007/978-3-540-77465-5_14.

[3] R. Friedrich, I. Iglezakis, W. Klein und S. Pregizer, „Experience-based decision support for project management with Case-Based Reasoning“ in *German Workshop on Experience Management*, 2002, S. 139–150.

[4] G. Kadoda, M. Cartwright und M. Shepperd, „Issues on the Effective Use of CBR Technology for Software Project Prediction“ in *Lecture Notes in Computer Science, Case-Based Reasoning Research and Development*, G. Goos, J. Hartmanis, J. van Leeuwen, D. W. Aha und I. Watson, Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 276–290, doi: 10.1007/3-540-44593-5_20.

[5] C. K. Riesbeck und R. C. Schank, *Inside case-based reasoning*. New York: Psychology Press, 1989.

[6] D. B. Leake, „CBR in context: The present and future“, *Case-based reasoning: Experiences, lessons, and future directions*, S. 3–30, 1996.

[7] M. M. Richter, „Fallbasiertes Schließen“, *Informatik-Spektrum*, Jg. 26, Nr. 3, S. 180–190, 2003, doi: 10.1007/s00287-003-0305-5.

[8] A. Aamodt und E. Plaza, „Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches“, *AI Communications*, Jg. 7, Nr. 1, S. 39–59, 1994, doi: 10.3233/AIC-1994-7104.

[9] V. Eisenstadt, C. Langenhan und K.-D. Althoff, „FLEA-CBR – A Flexible Alternative to the Classic 4R Cycle of Case-Based Reasoning“ in *Case-Based Reasoning Research and Development*, 2019, S. 49–63.

[10] E. Espenakk, M. J. Knalstad und A. Kofod-Petersen, „Lazy learned screening for efficient recruitment“ in *International Conference on Case-Based Reasoning*, 2019, S. 64–78.

[11] M. T. Keane und B. Smyth, „Good Counterfactuals and Where to Find Them: A Case-Based Technique for Generating Counterfactuals for Explainable AI (XAI)“ in *Case-Based Reasoning Research and Development: 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8–12, 2020, Proceedings*, 2020, S. 163–178, doi: 10.1007/978-3-030-58342-2_11.

[12] B. Díaz-Agudo, G. Jimenez-Diaz und J. L. Jorro-Aragoneses, „User Evaluation to Measure the Perception of Similarity Measures in Artworks“ in *Case-Based Reasoning Research and Development*, 2021, S. 48–63.

[13] D. Leake und X. Ye, „Learning to Improve Efficiency for Adaptation Paths“ in *International Conference on Case-Based Reasoning*, 2020, S. 325–340.

[14] A. Mazzucchelli und F. Sartori, „CCIS 478 - String Similarity in CBR Platforms: A Preliminary Study“.

[15] A. Jaiswal und K. Bach, „A data-driven approach for determining weights in global similarity functions“ in *International Conference on Case-Based Reasoning*, 2019, S. 125–139.

[16] H. Núñez *et al.*, „A comparative study on the use of similarity measures in case-based reasoning to improve the classification of environmental system situations“, *Environmental Modelling & Software*, Jg. 19, Nr. 9, S. 809–819, 2004, doi: 10.1016/j.envsoft.2003.03.003.

[17] M. M. Richter und S. Wess, „Similarity, Uncertainty and Case-Based Reasoning in Patdex“ in *Automated Reasoning: Essays in Honor of Woody Bledsoe*, R. S. Boyer, Hg., Dordrecht: Springer Netherlands, 1991, S. 249–265, doi: 10.1007/978-94-011-3488-0_12.

[18] D. Mcsherry, „Explanation in Recommender Systems“, *Artif Intell Rev*, Jg. 24, Nr. 2, S. 179–197, 2005, doi: 10.1007/s10462-005-4612-x.

[19] S. Ramachandran und C. Belardi, „Case-Based Reasoning for System Anomaly Detection and Management“ in *2021 IEEE Aerospace Conference*, Big Sky, MT, USA, 2021, S. 1–9, doi: 10.1109/AERO50100.2021.9438252.

[20] X. Ye, D. Leake, V. Jalali und D. J. Crandall, „Learning Adaptations for Case-Based Classification: A Neural Network Approach“ in *Case-Based Reasoning Research and Development*, 2021, S. 279–293.

[21] R. Dolphin, B. Smyth, Y. Xu und R. Dong, „Measuring Financial Time Series Similarity with a View to Identifying Profitable Stock Market Opportunities“ in *Case-Based Reasoning Research and Development*, 2021, S. 64–78.

[22] A. Upadhyay, S. Massie, R. K. Singh, G. Gupta und M. Ojha, „A Case-Based Approach to Data-to-Text Generation“ in *Case-Based Reasoning Research and Development*, 2021, S. 232–247.

[23] W. Yu, X. Guo, F. Chen, T. Chang, M. Wang und X. Wang, „Similar Questions Correspond to Similar SQL Queries: A Case-Based Reasoning Approach for Text-to-SQL Translation“ in *Case-Based Reasoning Research and Development*, 2021, S. 294–308.

[24] F. Abbas, N. Najjar und D. Wilson, „The Bites Eclectic: Critique-Based Conversational Recommendation for Diversity-Focused Meal Planning“ in *Case-Based Reasoning Research and Development*, 2021, S. 1–16, doi: 10.1007/978-3-030-86957-1_1.

[25] E. L. Flogard, O. J. Mengshoel und K. Bach, „Bayesian Feature Construction for Case-Based Reasoning: Generating Good Checklists“ in *Case-Based Reasoning Research and Development*, 2021, S. 94–109.

[26] I. Nkisi-Orji, N. Wiratunga, C. Palihawadana, J. A. Recio-García und D. Corsar, „Clood CBR: Towards Microservices Oriented Case-Based Reasoning“ in *Lecture Notes in Computer Science, Case-Based Reasoning Research and Development*, I. Watson und R. Weber, Hg., Cham: Springer International Publishing, 2020, S. 129–143, doi: 10.1007/978-3-030-58342-2_9.

[27] O. Berg, P. Reuss, R. Stram und K.-D. Althoff, „Comparing Similarity Learning with Taxonomies and One-Mode Projection in Context of the FEATURE-TAK Framework“ in *Lecture Notes in Computer Science, Case-Based Reasoning Research and Development*, K. Bach und C. Marling, Hg., Cham: Springer International Publishing, 2019, S. 1–16, doi: 10.1007/978-3-030-29249-2_1.

[28] B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García und A. A. Sánchez-Ruiz-Granados, „Building CBR systems with jcolibri“, *Science of Computer Programming*, Jg. 69, 1-3, S. 68–75, 2007, doi: 10.1016/j.scico.2007.02.004.

[29] Ł. Przysucha, „Content Management Systems Based on GNU GPL License as a Support of Knowledge Management in Organizations and Business“ in *3rd IFIP International Workshop on Artificial Intelligence for Knowledge Management (AI4KM)*, 2015, S. 51–65, doi: 10.1007/978-3-319-55970-4_4.

[30] S. K. Patel, V. R. Rathod und S. Parikh, „Joomla, Drupal and WordPress - a statistical comparison of open source CMS“ in *Computing (TISC)*, Chennai, India, 2011, S. 182–187, doi: 10.1109/TISC.2011.6169111.

[31] V. S. Bhirud, „Review Paper on an Open Source Content Management System: Joomla CMS“, *International Journal of Recent Research in Mathematics Computer Science and Information Technology*, Jg. 1, Nr. 2, S. 95–98, 2015.

[32] D. Barker, *Web Content Management: Systems, Features, and Best Practices*. O'Reilly Media, 2016.

[33] J. Cabot, „WordPress: A Content Management System to Democratize Publishing“, *IEEE Softw.*, Jg. 35, Nr. 3, S. 89–92, 2018, doi: 10.1109/MS.2018.2141016.

[34] J. Vivekavardhan, „Open Source Content Management System for Content Development: A Study on Wordpress, Joomla and Drupal“, *Library Waves*, Jg. 2, Nr. 1, S. 6–14, 2016.

[35] T. Tomlinson, *Beginning Backdrop CMS*. Berkeley, CA: Apress L. P, 2016.

[36] A. Mauthe und P. Thomas, *Professional Content Management Systems: Handling Digital Media Assets*. Wiley, 2005.

[37] J. E. Scott, „User Perceptions of an Enterprise Content Management System“ in *2011 44th Hawaii International Conference on System Sciences (HICSS 2011)*, Kauai, HI, 2011, S. 1–9, doi: 10.1109/HICSS.2011.473.

[38] M. Grossniklaus und M. C. Norrie, „Information concepts for content management“ in *Third International Conference on Web Information Systems Engineering (Workshops), 2002*, Singapore, 2002, S. 150–159, doi: 10.1109/WISEW.2002.1177858.

[39] *GNU General Public License 3*. [Online]. Verfügbar unter: https://www.gnu.org/licenses/gpl-3.0 (Zugriff am: 15. Februar 2022).

[40] S. Wan, D. Li und J. Gao, „Exploring the Advantages of Content Management Systems for Managing Engineering Knowledge in Product-service Systems“, *Procedia CIRP*, Jg. 56, S. 446–450, 2016, doi: 10.1016/j.procir.2016.10.087.

[41] K. R. Grahlmann, R. W. Helms, C. Hilhorst, S. Brinkkemper und S. van Amerongen, „Reviewing enterprise content management: A functional framework“, *European Journal of Information Systems*, Jg. 21, Nr. 3, S. 268–286, 2012.

[42] *w3techs.com*. [Online]. Verfügbar unter: https://w3techs.com/technologies/overview/content_management (Zugriff am: 22. Februar 2022).

[43] *builtwith.com*. [Online]. Verfügbar unter: https://trends.builtwith.com/cms (Zugriff am: 22. Februar 2022).

[44] *Licensing*. [Online]. Verfügbar unter: https://www.drupal.org/about/licensing (Zugriff am: 12. März 2022).

[45] T. Tomlinson und M. Edlefsen, *Enterprise Drupal 8 Development: For Advanced Projects and Large Development Teams*. Berkeley, CA: Apress L. P, 2017.

[46] T. Tomlinson, *Beginning Drupal 8*. Berkeley, CA: Apress L. P, 2015.

[47] *Continuous upgrades between major versions*. [Online]. Verfügbar unter: https://www.drupal.org/about/core/policies/core-change-policies/continuous-upgrades-between-major-versions (Zugriff am: 12. März 2022).

[48] D. Buytaert, *Why the big architectural changes in Drupal 8*. [Online]. Verfügbar unter: https://dri.es/why-the-big-architectural-changes-in-drupal-8 (Zugriff am: 16. März 2022).

[49] *Drupal 8.0.0 released*. [Online]. Verfügbar unter: https://www.drupal.org/blog/drupal-800-released (Zugriff am: 16. März 2022).

[50] A. Krouska, C. Troussas und M. Virvou, „Comparing LMS and CMS platforms supporting social e-learning in higher education“ in *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*, Larnaca, 2017, S. 1–6, doi: 10.1109/IISA.2017.8316408.

[51] *Absolute Basics of How a Component Functions.* [Online]. Verfügbar unter: https://docs.joomla.org/Absolute_Basics_of_How_a_Component_Functions (Zugriff am: 16. März 2022).

[52] *Extension types (general definitions).* [Online]. Verfügbar unter: https://docs.joomla.org/Extension_types_(general_definitions) (Zugriff am: 16. März 2022).

[53] *Drupal 6/7 programming from an object-oriented perspective.* [Online]. Verfügbar unter: https://www.drupal.org/docs/7/creating-custom-modules/drupal-67-programming-from-an-object-oriented-perspective (Zugriff am: 28. Mai 2022).

[54] K. Bach, B. M. Mathisen und A. Jaiswal, „Demonstrating the myCBR Rest API“ in *ICCBR Workshops*, 2019, S. 144–155.

[55] J. Dorn, „Sharing Project Experience through Case-based Reasoning“, *Procedia Computer Science*, Jg. 99, S. 4–14, 2016, doi: 10.1016/j.procs.2016.09.097.

```php
namespace Drupal\cbr\Plugin\Field\FieldType;

use Drupal\Core\Field\Plugin\Field\FieldType\BooleanItem;
use Drupal\Core\Form\FormStateInterface;
use Drupal\field\Entity\FieldConfig;

/**
 * Plugin implementation of the 'field_cbr_boolean' field type.
 *
 * @FieldType(
 *   id = "field_cbr_boolean",
 *   label = @Translation("CBR Boolean"),
 *   module = "cbr_boolean_field",
 *   category = @Translation("Case Based Reasoning"),
 *   description = @Translation("An entity field containing a boolean value."),
 *   default_widget = "boolean_checkbox",
 *   default_formatter = "boolean",
 *   cardinality = 1
 * )
 */
class CBRBoolean extends BooleanItem implements CBRFieldInterface
{
  /**
   * {@inheritdoc}
   */
  public function fieldSettingsForm(array $form, FormStateInterface $form_state): array
  {
    return CBRFieldHelper::cbrFieldSettingsForm($form, $form_state) + parent::fieldSettings-
    Form($form, $form_state);
  }

  /**
   * Similarity for two boolean values is 1, if they are equal, 0 otherwise.
   * @param $value1 boolean value 1
   * @param $value2 boolean value 2
   * @param FieldConfig $field_config The field config
   * @return float The similarity between the two values
   */
  public function calculateSimilarity($value1, $value2, $field_config): float
  {
    return $value1 == $value2 ? 1 : 0;
  }

  /**
   * Summerize multiple boolean values. This is used when cbr_reference_field references multiple
    entities.
   * @param array $values The values to summerize
   * @return float 1, if more of the values are true, 0 otherwise
   */
  public function summerize(array $fields): float
  {
    return array_sum($fields) > 0.5 ? 1 : 0;
  }

  /**
   * Get the value for similarity calculation.
   * @param FieldConfig $field_config The field config
   * @return boolean The value for similarity calculation
   */
  public function getValueForSimilarityCalculation(FieldConfig $field_config): float
  {
    return $this->value;
  }
}
```

CBRBoolean.php des Moduls cbr_boolean_field, welches das CBR-System um einen Feldtyp für boolesche Werte erweitert.