Diplomarbeit

Collaborative Maintenance in einem FBS System

Alexandre Hanft

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik, Künstliche Intelligenz

Betreuung: Prof. Dr. Hans-Dieter Burkhard
Dipl.-Inform. Mirjam Minor

Berlin, den 25.11.2004

Danksagung

Hiermit möchte ich mich bei allen bedanken, welche mich in dieser Arbeit unterstüzt haben: vor allen Mirjam Minor und Prof. Dr. Hans-Dieter Burkhard, welche die Bertreuung dieser Diplomarbeit übernahmen.

Besonderer Dank gilt dabei Olga Schiemangk, Dagmar Monett Diaz, Kay Schröter und Diemo Urbig, welche sich für Durchführung der Evaluation bereit erklärten und somt die Überprüfung der Ergebnisse dieser Arbeit erst ermöglichten.

Darüber hinaus bin ich Olga Schiemangk für das Lösen unzähliger technischer Probleme, Kay Schröter für die vielen kritischen Fragen und Renate Zirkelbach für die langjährige organisatorische Unterstützung verbunden.

Uta Volgmann danke ich für das Korrekturlesen.

Inhaltsverzeichnis

Erläuterungen zur Typographie 6

1 Einleitung 7

1.1 Wissensmanagement, Erfahrung und Fallbasiertes Schließen 7

1.2 Motivation 8

1.3 Assistenzsysteme am Lehrstuhl für Künstliche Intelligenz 9

1.4 Problemstellung 10

2 Grundlegende Begriffe und Verfahren 11

2.1 Wissen und Experience Management 11

2.2 Fallbasiertes Schließen 12

2.2.1 4-R-Zyklus von Amodt & Plaza 14

2.2.2 Case Retrieval Netz 15

2.2.2.1 Retrieval in einem CRN 16

2.3 Textuelles Fallbasiertes Schließen 17

2.3.1 Abbildung der Fälle im Textuellen FBS auf IEs 19

2.4 Wissenscontainer 19

2.4.1 Wissenscontainer im FBS 20

3 Literaturrecherche und Analyse 21

3.1 Maintenance 21

3.1.1 Maintenance im FBS 22

3.1.2 Maintenance von Wissen 23

3.2 Kategorisierung nach Leake und Wilson 23

3.2.1 Datensammlung 24

3.2.2 Auslösung der maintenance 25

3.2.3 Verfügbare Operationstypen 25

3.2.4 Ausführung 26

3.2.5 Typische Ansätze 27

3.2.6 Bewertung 28

3.2.7 Einordnung der maintenance Strategien vom ExperienceBook 29

3.2.8 Einordnung der maintenance Strategien vom TestManager und ExperienceBook II 29

3.3 Systematic Maintenance of Corporate Experience Repositories 30

3.3.1 Knowledge Life-Cycle Modell von Nick & Althoff 32

3.4 Sechs Schritte und die SIAM Methode 33

3.5 Maintenance schwach strukturierter Fälle 34

3.6 Collaborative Maintenance mit Ulysses 36

3.6.1.1 Einordnung von Ulysses und Abgrenzung zum TestManager 37

3.7 Anforderungen für ein eigenes Modell 38

4 Neue Konzepte ... 41
4.1 Modell des Life-Cycle eines Falls ... 41
4.1.1 Unterschiedliche Sichten zur Konsistenzwahrung im CRN trotz maintenance ... 42
4.2 Collaborative Maintenance für das Life-Cycle Modell ... 43
4.2.1 Vom Vier-Augen-Prinzip zur Autorisierung von Änderungen an Fällen ... 43
4.2.2 Überblick ... 43
4.2.3 Revision eines Wissensbausteins ... 44
4.2.3.1 Sichten ... 46
4.2.3.2 Konsistenzwahrung der Sicht für das Retrieval mit dem CRN ... 48
4.2.4 Der Review Prozess ... 48
4.2.4.1 Revisionsbewertung ... 49
4.2.4.2 Auswahl der Begutachter ... 52
4.2.4.3 Einladung zum Review (Review Posting) ... 53
4.2.4.4 Erweiterungsmöglichkeiten ... 53
4.2.5 Integration ... 53
4.2.6 Auslösung der Maintenance ... 54
4.3 Zusammenfassung des Modells ... 55
4.4 Aufwand vs. Nutzen – Wann lohnt sich der Aufwand? ... 56
4.5 Vergleich mit bestehenden Ansätzen ... 57
4.5.1 Software Engineering ... 57
4.5.2 ExpBook I + II ... 57
4.5.3 Einordnung in Kategorisierungsschema nach Leake & Wilson ... 58
4.5.4 Systematic maintenance am IESE ... 58
4.5.5 «Six steps» und SIAM ... 59
4.5.6 Enzyklopädie WIKIPEDIA ... 60
5 Implementierung ... 61
5.1 Objektmodell ... 61
5.2 DataManager Schnittstelle ... 62
5.2.1.1 Status in neuen Schnittstellen ... 62
5.2.1.2 Abwärtskompatibilität ... 62
5.2.2 Die Klasse Review ... 62
5.3 Datenbank ... 63
5.3.1 Entwurf des Datenbankschemas ... 63
5.3.2 Löschen und Wiederherstellen ... 63
5.3.3 Implementierung des Prozessmodells ... 64
5.3.3.1 Ändern von Zuständen durch Reviews ... 64
5.3.3.2 Integration ... 64
5.3.3.3 Auslösung der Maintenance ... 64
5.3.3.4 Konsistenzwahrung der Sicht für das Retrieval ... 64
5.4 Web-Oberfläche ... 65
5.4.1 Das Sichtenkonzept auf der Oberfläche ... 65
5.5 Behandlung unterschiedlicher Fallformate ... 68
5.5.1 Konfiguration des Fallformats in der Ini-Datei ... 68
5.5.2 Einlesen der Ini-Datei ... 69
5.5.3 Verwendung im Objektmodell ... 70
5.5.4 Kommunikation mit den CGI Clients ... 70

6 Evaluation 71
6.1 Vorbereitung 71
6.2 Ergebnisse 71
6.2.1 Handhabung der Oberfläche 72
6.3 Fazit 73
7 Zusammenfassung und Ausblick 75
7.1 Zusammenfassung 75
7.2 Ausblick 76
8 Anhang 77
8.1 Abbildungsverzeichnis 77
8.2 Definitionen 78
8.3 Stichwortverzeichnis 80
9 Literatur 83
Selbständigkeitserklärung 88
Einverständniserklärung 89

Erläuterungen zur Typographie

Hervorhebungen erfolgen gesperrt kursiv.
Verweise auf andere Abschnitte erfolgen ebenso kursiv.
In `Courier New` werden Benutzereingaben, Code und Dateinamen dargestellt.
Besondere Termini und aufgeführte Dialoge wie «case authoring» werden in französische Anführungszeichen eingeklammert.
Literaturverweise werden in [AutorenJahr] eingefasst und optional mit einer Seitenangabe versehen: [Richter2003, S. 412].

Beispiele werden mit der Schriftart Verdana formatiert und zusätzlich eingerückt.

1 Einleitung

1.1 Wissensmanagement, Erfahrung und Fallbasiertes Schließen

Wissensmanagement

Knowledge Management, zu deutsch Wissensmanagement, war eines der Schlagworte Ende der Neunziger Jahre. Als man in der Wirtschaft Wissensmanagement für sich entdeckte und nutzen wollte, wurden hohe Erwartungen geweckt, die nicht erfüllt werden konnten. Nach dem Boom ist Wissensmanagement aber eine etablierte Disziplin in Wirtschaft und Wissenschaft. Je mehr man darüber erforscht, um so mehr stellt man fest dass man noch viel zuwenig darüber weiß. Dass jenes, was früher als Wissensmanagement verkauft wurde, nur Daten- und Informationsmanagement war. Mehrere Modelle zur Erklärung und Prozessmodellierung liegen zur Auswahl bereit. Aber wie genau wird nun das Wissen kreiert und weiterverarbeitet? Wissensmanagement ist eine junge Wissenschaft, erschöpfende Modelle sucht man noch.

Experience Management

Vor diesem Hintergrund entstand 2000 das Gebiet Experience Management. Selten wird die deutsche Form Erfahrungsmanagement erwähnt. Es beschränkt sich im großen Gebiet des Wissensmanagements darauf, Wissen aus Erfahrung zu extrahieren, zu managen und wieder nutzbar zu machen.

Man stellt fest, dass Menschen z. B. erstaunlich clever und kreativ im Problemlösen sind, und wie schwer es doch ist, Maschinen das automatische Problemlösen beizubringen. Auch wenn Software uns schon in beschränkten Teilgebieten überholt – Warum also nicht die Art und Weise des Problemlösens beim Menschen abschauen?

Fallbasiertes Schließen

Das Fallbasiertes Schließen z. B. ist eine solche Methode zum automatischen Problemlösen. Durch die Anwendung früher gemachter Erfahrungen ist es auch eine Methode, welche Experience Management technisch unterstützt.
Das Fallbasierte Schließen (FBS) arbeitet sehr ähnlich zu einer Arbeitsweise, wie Experten bei der Problemlösung vorgehen: Sie vergleichen das anstehende Problem mit ihren schon gemachten Erfahrungen, ob sie ein ähnliches Problem schon einmal hatten und versuchen, dessen frühere Lösung gegebenenfalls anzupassen und dann auf das aktuelle Problem anzuwenden. Nun sammelt ein Experte in seinem Leben soviel Erfahrungen, dass er sich bei schwierigen Problemen Notizen anlegen muss, weil er nicht alles behält. Konnte der Experte auf keine nützlichen Erfahrungen zurückgreifen, bat er früher eine Kollegin um Rat, schilderte ihr das Problem. Sie kramte in ihrem Erfahrungsschatz, bot ihre Lösung an, wenn sie etwas Adäquates fand oder verwies ihn an einen anderen Experten, der dazu mehr wissen könnte. Und wenn die Kollegin gerade keine Zeit hat?

Heute bedient sich der Experte eines fallbasierten Experience Management Systems, in das er seine eigenen Erfahrungen als Fälle einpflegt. Wenn seine Kollegin dies auch tut, kann er beim nächsten Problem mit einer „intelligenten“ Suchtechnik auf seine eigenen und die Erfahrungen seiner vielen Kollegen zurückgreifen. Ein noch größerer Vorteil ergibt sich für denjenigen, der noch gar kein Experte ist, aber wenn er ein ähnliches Problem hat, die Erfahrungen der Experten nutzen kann.

1.2 Motivation

Maintenance

Verändernde Umwelt

Bisherige Interaktion berücksichtigen

Auch wenn sich Experience Management Software und das in ihr eingebettete Wissen durch ihren Gebrauch nicht abnutzt, egal wie oft man es verwendet, so bleibt der Vorteil daraus nicht konstant. Vielmehr verändert sich die Umgebung der Nutzer und des Systems. Neue Erkenntnisse außerhalb des Systems werden gewonnen und Benutzer erwarten von einem Wissensbasierten System, dass es immer das neueste verfügbare Wissen berücksichtigt. Daher veraltet das Wissen innerhalb des Systems. Benutzer wünschen abgesehen davon dass, ein System in gewissen Sinne intelligent bzw. lernfähig ist. Dazu zählt in der Kommunikation mit einem Menschen, dass sich das Gegenüber an mit ihm geteilte Ereignisse erinnert und diese berücksichtigt. Also z. B. in der gleichen Situation nicht zweimal den gleichen Vorschlag macht, wenn er beim ersten Mal abgelehnt wurde. Das System sollte also aus der Interaktion „lernen“ und sein Verhalten anpassen. Aus diesen beiden Gründen ist es erforderlich, das Wissen zu überarbeiten. Dafür wird im allgemeinen Sprachgebrauch der Begriff Wartung benutzt. In dieser Arbeit wird aber die englische Form «maintenance» verwendet, da sich dies in der wissenschaftlichen Fachliteratur schon eingebürgert hat.

Review notwendig

Für eine von einem Autor berichtete Erfahrung kann nicht von herein davon ausgegangen werden, dass sie inhaltlich fehlerfrei ist bzw. eine ausreichende Qualität hat. Dies gilt insbesondere, wenn neues Fallwissen durch die Benutzer selbst eingegeben wird und dabei keine Voraussetzungen an deren fachliche Kompetenz gestellt werden. Aber selbst Experten können sich irren oder Flüchtigkeitsfehler begehen.
Eine Aufgabe eines Experience Management Systems besteht in der Kommunikation von Erfahrung(-swissen), also auch der Umwandlung von implizitem in explizites Wissen. Der Autor erstellt den Fall als ein Stück externes Wissen vor dem Hintergrund seines ganzen impliziten Wissens. Dabei kann selbst inhaltliche Richtigkeit im Erfahrungsbericht nicht gewährleisten, dass alle zum Verständnis des Berichts notwendigen Informationen enthalten sind. Ein gewisses Vorwissen wird immer notwendig sein, aber dieses Niveau muss sich an der Benutzergruppe orientieren. Nur ein anderer Gutachter oder die zukünftigen Benutzer dieser Erfahrung selbst können überprüfen, ob der Erfahrungsbericht verständlich ist – er muss begutachtet werden.

Versionsverwaltung

Arbeiten mehr als zwei Autoren verteilt an einem Dokument oder einer sonstigen Form von Wissen, ist ein System zur Verfolgung und Konsistenzwahrung der Änderungen notwendig, damit Änderungen nicht gegenseitig überschrieben werden. Beispiele sind weltweit verteilte Entwickler-

teams, welche eine spezielle Art von Wissen – Software – erstellen. Das Datenbankmanagementsystem MySQL[1] wie viele andere Open Source Software entsteht so. Prinzipiell kann jeder an der Kodierung des Wissens teilnehmen, nur liegen die impliziten Anforderungen, den Code zu verstehen und zu verbessern, hoch.
Ein zugänglicheres Beispiel ist die freie Enzyklopädie WIKIPEDIA[2]. Bei ihr kann jeder Benutzer neue Lexikoneinträge anlegen und vorhandene ändern. WIKIPEDIA arbeitet mit einem Versionskontrollsystem, man kann sich jede einzelne Version eines Artikels anzeigen und mit anderen vergleichen lassen. Von einem kleinen Projekt hat es sich mit über 160.000 Einträgen zu einer ernsthaften Konkurrenz für kommerzielle elektronische Universallexika entwickelt. Es kann jetzt sogar auf CD bestellt werden.

1.3 Assistenzsysteme am Lehrstuhl für Künstliche Intelligenz

ExperienceBook

Das ExperienceBook von Mirjam Minor [Kunze1998] ist zur Unterstützung der Systemadministration konzipiert worden und wird heute noch für die administrativen und technischen Belange täglicher Arbeit jenseits der Forschungsarbeit am Lehrstuhl eingesetzt. Charakteristisch für das ExpBook ist, dass neue Fälle als Vorschläge an der Web-Oberfläche eingegeben werden. Der ExpBook-Administrator erhält diese per Email und pflegt diese nach Überprüfung von Hand in die Fallbasis ein.

TestManager

Aufbauend auf dem bestehenden System ExperienceBook wurde im Rahmen eines Forschungsprojekts das System TestManager daraus weiterentwickelt. Es beschäftigt sich mit der Unterstützung bei der Erstellung und Verwaltung von Testfällen für den Test einer Software aus der Erdgas-Domäne. Die Benutzer des TestManager sind gleichberechtigte Softwaretester bzw. Entwickler, welche die Fälle ohne die Mithilfe eines Administrators einpflegen und ändern können. Dazu wurde eine partielle Ähnlichkeitsfunktion, eine flache Revisionsverwaltung und zwei verschiedene Sichten auf die Fallbasis eingeführt. Änderungen werden unmittelbar danach auch für die anderen Benutzer in der «authoring view» sichtbar, nach einem Neustart auch in der «retrieval view».

ExperienceBook II

Daraus wurde das ExperienceBook II abgeleitet, welches den Austausch von Erfahrungswissen der Informatik Studentinnen und Studenten am Institut zum Ziel hat. Es verfügt über eine andere Oberfläche. Technisch gesehen sind beide Geschwister, so dass die Ausführungen zum TestManager auch für das ExpBook II gelten.

[1] http://www.mysql.com
[2] http://de.wikipedia.org/wiki/Hauptseite

1.4 Problemstellung

Beim TestManager erfolgt die Integration der neuen Fälle ohne eine Überprüfung, was für eine kleine oder in der Kompetenz sehr homogene Gruppe auch adäquat sein kann.
In einer offenen, größeren Gruppe, wo die Benutzer über ein unterschiedliches Wissensniveau verfügen, kann eine Kontrolle notwendig sein. Beim ExpBook I übernimmt der Administrator die redaktionelle Kontrolle.

Kommunikation

Beim ExpBook II können die Benutzer einen Kommentar zu einem Fall abgeben, der direkt an den Falltext angehängt wird.
Um die Kommunikation beim Erstellen und Verändern der Fälle zu verbessern, scheint mitunter ein eher redaktioneller Kommentar notwendig, z. B. eine Frage „Wurde bei der Problemlösung dieser Aspekt betrachtet?“ auf die dann der Autor durch Änderung seines Falls reagiert. Diese Trennung des Kommentars vom Falltext ist vor allem dann sinnvoll, wenn die Fragestellende oder der Begutachtende sich auf dem Problemgebiet dieses Falles nicht so kompetent fühlt, um den Fall selbst zu ändern.

leichtgewichtiger Maintenance Prozess

Dafür soll ein leichtgewichtiger Maintenance Prozess für Erfahrungen in Textform entwickelt und implementiert werden, welcher ohne einen Administrator auskommt. Die maintenance beruht dabei auf der gegenseitigen Kontrolle durch die Benutzer selbst. Dadurch soll die Qualität des geänderten Wissens sicher gestellt und die Nützlichkeit des Erfahrungsmanagement-Systems erhalten werden.

Das Projekt wird in Anlehnung an bestehende Systeme *mtBook* genannt, wobei *mt* für maintenance steht.

Das folgende Kapitel 2 definiert *Grundlegende Begriffe und Verfahren*, die in dieser Arbeit verwendet werden, *ab Seite 11,* darunter zum Wissensmanagement und dem Fallbasiertem Schließen.
Danach wird in Kapitel *3 Literaturrecherche und Analyse ab Seite 21* der Stand der Forschung auf dem Gebiet der maintenance im FBS referiert. Dabei werden die in den Arbeiten verwendeten Methoden darauf hin analysiert, ob sie sich auf die Problemstellung übertragen lassen. Außerdem werden maintenance Prozesse der bisherigen Projekte anhand des Kategorisierungsschemas von Leake & Wilson eingeordnet.
Kapitel 4 stellt ab *Seite 41* die für die Zielstellung entwickelten *neuen Konzepte* des Collaborative Maintenance Modell für den Life-Cycle eines Wissensbausteins vor. Anschließend wird dieses Modell mit den Ansätzen aus Kapitel 3 verglichen und eingeordnet.
Kapitel 5 auf *Seite 61* skizziert den Entwurf und die *Implementierung* des Systems. Das Kapitel 6 stellt ab *Seite 71* die Ergebnisse der *Evaluation* des eingeführten maintenance Prozesses vor. Im Kapitel 7 folgt eine Zusammenfassung und der Ausblick.

2 Grundlegende Begriffe und Verfahren

2.1 Wissen und Experience Management

Das Wissensmanagement (WM, englisch Knowledge Management, KM) ist ein zur Zeit viel diskutiertes Thema im Übergang unserer heutigen Gesellschaft zu einer Wissens- und Dienstleistungsgesellschaft.
Unter den vielen verfügbaren Definitionen soll hier jene des Consulting-unternehmens KPMG aus dem Knowledge Management Report 1998 [KPMG1998] herausgegriffen werden:

Wissensmanagement

Definition 1: **Wissensmanagement**
Wissensmanagement stellt einen systematischen und organisierten Ansatz dar, um Wissen innerhalb einer Organisation dazu zu nutzen, die Leistung im Unternehmen zu steigern.

Das Erfahrungsmanagement ist eine Unterart des Wissensmanagements. In dieser Arbeit wird der äquivalente englische Begriff Experience Management verwendet, da er in der Fachliteratur der gebräuchlichere ist. Aus den verschiedenen sei hier eine Definition von [Bergmann2002] aufgeführt, wobei hier schon auf die Übersetzungen von [Bell2004] zurückgegriffen wird:

Experience Management

Definition 2: **Experience Management**
Das Experience Management ist eine spezielle Art des Wissensmanagements, welche sich darauf beschränkt, Erfahrungswissen zu managen, z. B. spezielles Wissen für einen bestimmten Problemlösungskontext.
Das Experience Management beschäftigt sich mit dem Sammeln, Strukturieren, Speichern, Wiederverwenden, Evaluieren und Pflegen von Erfahrungen.

Die eben erwähnten Begriffe sollen nicht offen bleiben, wobei wieder [Bergmann2002] gefolgt wird:

Definition 3: **Daten**
Daten sind syntaktische Entitäten. Sie können von Computern gespeichert und bearbeitet werden.

Definition 4: **Information**
Informationen sind interpretierte Daten, d. h. Daten mit Bedeutung. Daraus folgt: Informationen sind Daten in Verbindung mit einer Semantik.

Wissen

Definition 5: **Wissen**
Wissen ist eine Menge von inhaltlich zusammenhängenden Informationen mit Pragmatik. Wissen bringt Informationen in einen Kontext, der durch eine Aufgabe oder ein Ziel gegeben ist.

Definition 6: **spezielles Wissen**
Spezielles Wissen ist Wissen, welches Angaben über einen Teilaspekt in einer Domäne aufführt. Spezielles Wissen ist sehr ähnlich zu Faktenwissen.

Definition 7: **allgemeines Wissen**
Allgemeines Wissen ist Wissen, welches Angaben über verschiedene Teilaspekte einer Domäne aufführt. Es ist auch möglich mehrere Domänen einzubeziehen.

Erfahrung

Definition 8: **Erfahrung**
Erfahrung ist wertvolles, gespeichertes, spezielles Wissen, das von einem Akteur in einer früheren Situation erworben wurde.

Der Akteur kann dabei menschlicher Natur als auch ein Agent sein.

Aus dem Handbuch der Künstlichen Intelligenz ist folgende Definition [BorgeltEtAl2003, S. 291]:

Definition 9: **Wissensbasierte Systeme**
Wissensbasierte Systeme sind Programme, die auf der Grundlage von Wissen über einen bestimmten Anwendungsbereich Schlussfolgerungen ziehen können und so dem Benutzer helfen, ein Problem zu lösen oder eine Entscheidung zu treffen.

2.2 Fallbasiertes Schließen

Fallbasiertes Schließen

Die Methode des Fallbasierten Schließens (kurz FBS, engl Case-Based Reasoning, CBR) zählt zu den wissensbasierten Systemen auf dem Gebiet der Künstlichen Intelligenz. Es kann zur Unterstützung von Erfahrungsmanagement eingesetzt werden. Dabei wird zur Lösung eines Problems auf schon gemachte Erfahrungen bei der Lösung ähnlicher Probleme zurückgegriffen [Richter2003].

Insbesondere das Textuelle Fallbasierte Schließen, welches vornehmlich mit schwach strukturiertem Wissen in natürlichsprachlicher Textform arbeitet, wurde schon in vielen Projekten am Lehrstuhl für Künstliche Intelligenz des Instituts für Informatik untersucht [LenzEtal1998].

Die Unterstützung des automatischen Problemlösens geht auf den Ansatz des Dynamischen Gedächtnisses zurück [Schank1982]. Weitere Grundlagen stammen von [Kolodner1993] und [RiesbeckSchank1989].

Aus der Diplomarbeit von Mirjam Minor [Kunze1998] wurden folgende Definitionen übernommen:

Für die systeminterne Darstellung von Fällen und Anfragen wird folgendes Konzept der Informationseinheiten (IE, engl. Information Entity) verwendet (vergleiche auch [Bergmann2002, S.206]):

Informationseinheit

Definition 10: **Informationseinheit (IE)**
Eine Informationseinheit ist als atomares Element das kleinste modellierte Element zur Wissensrepräsentation. E ist die Menge aller (möglichen) Informationseinheiten in einer Domäne.

Dem allgemeinen Sprachgebrauch folgend wird IE als Kurzform in dieser Arbeit verwendet.

Fall vs. Fallrepräsentation

Der Begriff «Fall» ist nicht ganz eindeutig definiert, da er sowohl für Beschreibungen (z. B. Problem und Lösung), die unabhängig von dem konkreten fallbasierten System existieren, als auch für deren systeminterne Form innerhalb einer Fallbasis verwendet wird. Daher wird im weiteren Verlauf dieser Arbeit die systeminterne Form fortan in Abweichung zu [Kunze1998] als Fallrepräsentation bezeichnet, während im allgemeineren Kontext «Fall» verwendet wird:

Definition 11: **Fall**
Ein Fall ist eine gemachte Erfahrung aus einer bestimmten Domäne in der Form, in welcher der Akteur mit ihr arbeitet.

Definition 12: **Fallrepräsentation**
Eine Fallrepräsentation c spiegelt das Wissen eines Falls wider, sie besteht aus einer Menge von IEs: $c \subseteq E$.

Fallbasis

Definition 13: **Fallbasis**
Die Menge aller Fallrepräsentationen in der Fallbasis wird mit C bezeichnet, wobei gilt: $C \subseteq \wp(E)$.

Gleiches gilt für den Begriff «Anfrage» (engl. Query), welcher auf der einen Seite die Problembeschreibung, wie ein Benutzer des Systems sie formuliert, und auf der anderen Seite die systeminterne Darstellung dieser Anfrage bezeichnet. Daher wird zur Unterscheidung die zweite Bedeutung als Anfragerepräsentation bezeichnet.

Anfrage

Definition 14: **Anfrage**
Eine Anfrage ist eine Problemstellung an ein fallbasiertes System, in der Form wie sie der Benutzer definiert.

Das heißt nicht, dass die Anfrage immer völlig frei in ihrem Format ist. Sie kann z. B. bei Eingabe über ein Webformular in der Art strukturiert sein, dass sie nur bestimmte Attribute mit vorgegebenen Werten oder verschiedene Textfelder zulässt.

Definition 15: **Anfragerepräsentation**
Eine Anfragerepräsentation q spiegelt das Wissen einer Anfrage wider, dass zur Spezifizierung der Problemstellung an ein fallbasiertes System (vom Nutzer) angegeben wird und zu dessen Lösung verwendet werden kann.

Sie besteht aus einer Menge von IEs: $q \subseteq E$.
Die Menge aller Anfragerepräsentationen über der Fallbasis wird mit Q bezeichnet, wobei gilt: $Q \subseteq \wp(E)$.

Eine Anfrage wird häufig auch als unvollständiger Fall aufgefasst.

Definition 16: **lokale Ähnlichkeitsfunktion**
Eine lokale Ähnlichkeitsfunktion sim_{loc} bezeichnet den Grad der Ähnlichkeit eines IEs zu einem anderen IE:
sim_{loc}: $E \times E \rightarrow \Re$.

Ähnlichkeitsfunktion

Definition 17: **Ähnlichkeitsfunktion (globale)**
Eine Ähnlichkeitsfunktion *sim*: $Q \times C \rightarrow \Re$ berechnet den Grad der Ähnlichkeit einer Fallrepräsentation zu einer Anfragerepräsentation q, so dass gilt: $sim(q, c_i) > sim(q, c_j) \leftrightarrow c_i$ ist ähnlicher zu q als c_j.
Sie ist z. B. definiert als: $sim(Query, Case) = \sum_{e_k \in Query} \sum_{e_i \in Case} sim(e_k, e_i)$.

2.2.1 4-R-Zyklus von Amodt & Plaza

Die für das Fallbasierte Schließen wohl am meisten zitierte Charakterisierung ist der «4-R-Zyklus» von Aamodt & Plaza [AamodtPlaza1994]. Er beschreibt in Bild 1 das FBS als einen sich wiederholenden Kreis, der aus vier Prozessen besteht: Retrieve, Reuse, Revise und Retain.

4-R-Zyklus

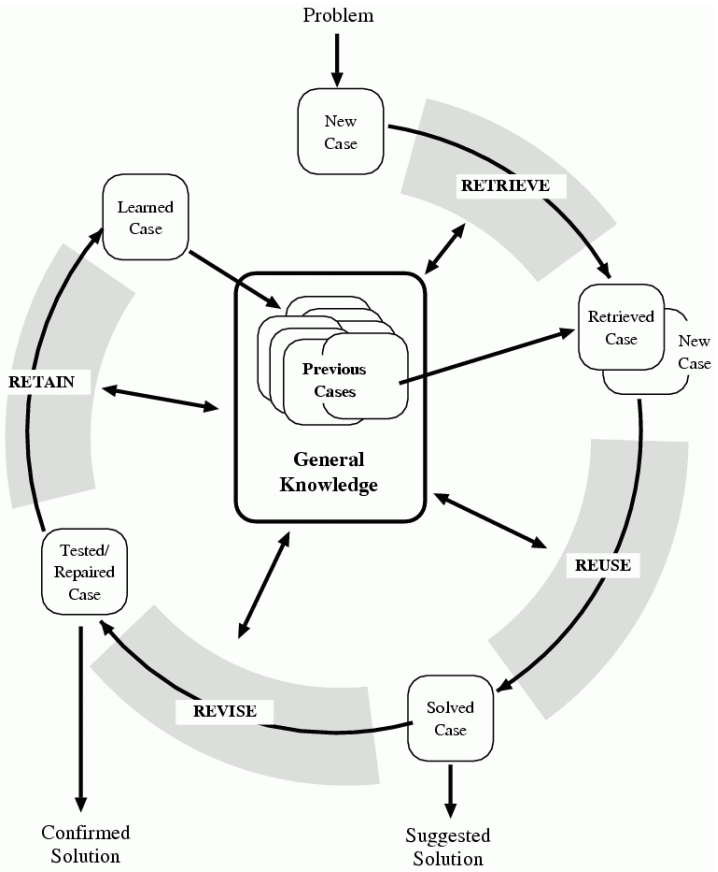


Bild 1: 4-R-Zyklus nach Aamodt & Plaza, entnommen aus [Lenz1999]

Retrieve besteht genau genommen aus zwei Teilen:

Retrieve

- dem Erzeugen eines unvollständigen Neuen Falls als Anfrage (-repräsentation) mit der aktuellen Problemstellung und dem
- Retrieval: Suche nach ähnlichen Fällen.

Reuse bezeichnet die

Reuse

- Benutzung des Wissens aus gefundenen Fällen für den Neuen Fall, (z.B. Übernahme der Lösung, Adaption durch Regeln).

Revise besteht aus dem

Revise

- Einsatz der vorgeschlagenen Lösung, der zu einer Bewertung des Neuen Falls führt. Evtl. muss die vorgeschlagene Lösung angepasst bzw. geändert werden.

Die anfängliche Problemstellung und die tatsächlich verwendete Lösung ergeben zusammen den kompletten neuen Fall.

Retain

Beim Retain wird

- der neue Fall in der Fallbasis abgespeichert, so dass er für das nächste Retrieval mit zur Verfügung steht.

Die Systeme ExpBook [Kunze1998] und TestManager [MinorHanft2000], [Hanft2004] vom hiesigen Lehrstuhl führen genau genommen davon nur die Prozesse Retrieve und Retain aus. Dabei werden dem Benutzer als Ergebnis des Retrievals eine Menge von Fällen vorgeschlagen, die verbleibenden Prozesse Reuse und Revise muss der Benutzer selbst erledigen. Zu Gründen siehe *2.3 Textuelles Fallbasiertes Schließen auf Seite 17*. Daher schlägt Richter in Konsequenz vor, es dass dann nicht Fallbasiertes Schließen, sondern Fallbasiertes Retrieval heißen müsste [Richter2003, S.407].

2.2.2 Case Retrieval Netz

Für das Retrieval im FBS gibt es verschiedene Speichertechniken, z.B. kd-Bäume [Wess1995], «fish and shrink» [Schaaf1996] oder Case-Retrieval-Netze (CRN). CRNs wurden am hiesigen Lehrstuhl für Künstliche Intelligenz entwickelt [Lenz1999] und sind eine Technik zur Strukturierung des Fallspeichers.

Case Retrieval Netz

Definition 18: **Case Retrieval Netz**
Ein Case Retrieval Netz (CRN) ist definiert als Netz
$N = [E, C, \sigma, \rho, \Pi]$ mit

E als endlicher Knotenmenge von Informationseinheiten (IEs),

C als endlicher Knotenmenge von Falldeskriptoren,

σ als Ähnlichkeitsfunktion $\sigma : E \times E \rightarrow \Re$,
welche die Ähnlichkeit $\sigma(e_i, e_k)$ zwischen den IEs e_i und e_k ausdrückt

ρ als Relevanzfunktion $\rho: E \times C \rightarrow \Re$
welche die Relevanz $\rho(e, c)$ zwischen IE e und Fall c ausdrückt und

Π als endlicher Menge von Propagierungsfunktionen π_n für jeden Knoten $n \in E \cup C$ mit π_n: $\Re^E \rightarrow \Re$.

Das Case Retrieval Netz ist ein gerichteter Graph mit zwei Arten von Knoten. IE-Knoten und Falldeskriptoren zur Identifikation von Fällen (auch Fallknoten genannt). Weiterhin besitzt der Graph zwei verschiedene, gewichtete Kantentypen. Bidirektionale Ähnlichkeitskanten drücken die je nach Typ (der IEs) gewichtete Ähnlichkeit zwischen zwei IEs aus. Relevanzkanten von IEs zu den Falldeskriptoren bestimmen die Relevanz dieser IEs für einen Fall, wie Bild 2 zeigt. Die Propagierungsfunktionen der IE- bzw. Fallknoten lauten π_{IEi} bzw. π_{Ci}.

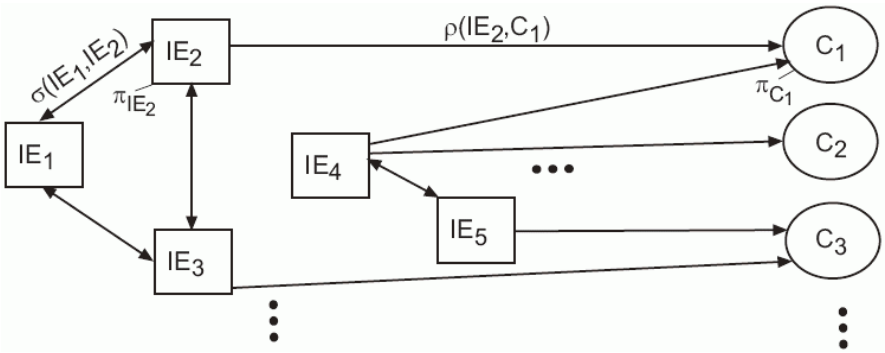


Bild 2: Case-Retrieval Net, entnommen aus [Bergmann2002, S.207]

Eine Fallrepräsentation besteht daher aus einem Teilgraph, mit einem Fallknoten und allen IE-Knoten, welche ihn repräsentieren.
Ähnlichkeiten zwischen den IEs als auch die IEs selbst reflektieren dabei die Eigenschaften der Domäne [Richter2003, S. 419].

2.2.2.1 Retrieval in einem CRN

Retrieval in einem CRN

Das Retrieval in einem CRN wird über einen spreading activation-Mechanismus ausgeführt. Die aus der Anfrage extrahierten IEs werden im CRN gesucht und aktiviert. Danach erfolgt eine Weiterleitung ihrer Aktivierung über die Ähnlichkeitskanten zu weiteren IEs und über Relevanzkanten zu den Falldeskriptoren. Zum Schluss werden die Aktivierungen der Falldeskriptoren eingesammelt und sortiert. So entsteht eine Halbordnung auf den Falldeskriptoren, die sich durch die Ähnlichkeit jedes Falles zum eingegebenen Fall herleitet. Der Falldeskriptor mit der höchsten Aktivierung zeigt auf den Fall, der am ähnlichsten zur Anfrage ist.

Durch Hinzufügen neuer Knoten und Kanten ist das CRN erweiterbar. Das Retrieval hat aufgrund der nichtlinearen Struktur des CRNs eine sehr kurze Laufzeit. CRNs eignen sich daher besonders gut, große Mengen von Falldaten (> 10.000) zu verwalten.

2.3 Textuelles Fallbasiertes Schließen

Textuelles Fallbasiertes Schließen

Das Textuelle Fallbasiertes Schließen (TFBS, engl. Textual CBR, TCBR) [LenzEtal1998] behandelt Fälle, welche aus vornehmlich schwach strukturiertem Wissen in natürlichsprachlicher Textform bestehen. Elekronisch verfügbare Textdokumente jeder Art können dafür verwendet werden. Dies wären beispielsweise Reparatur-Protokolle, Fallstudien, Dokumentationen über Produkte oder auch Beschreibungen im Internet zu Dienstleistungen wie z.B. Reisen.

> Definition 19: **Textuelles Fallbasiertes Schließen**
> Textuelles FBS ist FBS, welches Text benutzt, um den Prozess der Wissensverarbeitung zu ermöglichen oder zu verbessern.
> [WilsonBradshaw1999]

Für die Suche in Textdokumenten genügt es nicht, die gemeinsam vorkommenden Wörter zu zählen, um festzustellen ob zwei Texte (die Anfrage und Text aus einem Fall) inhaltlich etwas miteinander zu tun haben. Neben persönlichen Wortschätzen, Abkürzungen und vielen verschieden Möglichkeiten des Satzbaus gibt es daher viele Möglichkeiten ein und denselben Sachverhalt zu beschreiben. Dieses Problem unterschiedlicher Formulierungen wird auch «paraphrase problem» genannt [MinorLenz2001]: Der Nutzer findet keine Lösung zu seinem Problem, obwohl eine solche Lösung vorhanden ist – nur wurde sie mit anderen Worten beschrieben. Eine reiner Vergleich von Zeichenketten («string matching») liefert da unzureichende Ergebnisse. Textuelles Fallbasiertes Schließen vermeidet dieses Problem durch die Modellierung von Ähnlichkeiten, welche ja auch Beziehungen zwischen Synonymen definieren können.

Vorteile der Verwendung natürlichsprachlicher Texte

Der große Vorteil der direkten Verwendung der natürlichsprachlicher Texte ist, dass die Gewinnung von Fällen mit vergleichsweise geringem Aufwand möglich ist. Sind schon Dokumente vorhanden, entfällt der «case authoring» genannte Prozess der Fallerstellung, da die Dokumente (nach einer evtl. Aufbereitung) meist automatisch in Fälle überführt werden können.

Das heißt nicht, dass die Textdokumente völlig frei von einer Struktur sein müssen. Bei Protokollen aus dem Produkt-Support gibt es sehr oft eine vorgeschriebene Struktur der Dokumente, wie z. B. Textabschnitte zu Problembeschreibung des Kunden, Ergebnis der Analyse, Reparaturmaßnahmen usw. Diese Strukturierung kann dann im FBS System verwendet werden um beispielsweise die klassische Einteilung eines Falls in Problem und Lösung aufzustellen. Ebenso kann aber abhängig von der Aufgabenstellung diese Trennung auch vollständig aufgehoben werden, dann wird bei dem Retrieval das gesamte Textdokument berücksichtigt.

Fälle bestehen dann aus einer Identifikationsnummer, mehreren Attribut-Werte-Paaren (engl. Attribute-Value-Pair, AVP) und einem oder mehreren Textabschnitten. Bei den Attribut-Werte-Paaren werden die Attribute meist in zwei Mengen unterteilt, eine mit für das Retrieval relevanten

AVPs (z.B. Produktgruppe) und den anderen nur zur Information (z. B: Autor, Datum).

Jedes (Retrieval-) Attribut-Werte-Paar wird dabei auf eine Informationseinheit (IE) abgebildet. Einige Autoren sehen Texte auch als eine Form der Attribut-Werte-Paare an [Richter2003, 412]. Allerdings werden Texte (die länger als ein Wort sind) immer auf mehrere IEs abgebildet, so dass hier Textabschnitte immer extra geführt werden.

Folgende Definitionen und Erläuterungen sind wieder der Diplomarbeit von [Kunze1998] entnommen:

Indexterm

Definition 20: **Indexterm**
Ein Indexterm ist ein Begriff, der eine Menge von Strings repräsentiert, die dasselbe semantische Konzept beschreiben, z. B. grammatikalische Formen, Abkürzungen oder verschiedene Schreibweisen.

Indexvokabular

Definition 21: **Indexvokabular**
Das Indexvokabular eines fallbasierten Systems für Texte ist die Menge aller bekannten Strings, die auf Indexterme abgebildet werden können.

Beim fallbasierten Schließen für Texte werden zwei verschiedene Formen von IEs unterschieden:

Definition 22: **Text-IE**
Ein Text-IE ist ein IE, das einen Indexterm repräsentiert.

Jedem Index-Term ist genau ein IE zugeordnet.

Definition 23: **Attribut-IE**
Ein Attribut-IE ist ein IE, das ein Attribut-Werte-Paar repräsentiert.

Die Text-IEs sind in verschiedene Kategorien (z. B. „domänenspezifischer Begriff oder „allgemeiner Begriff“) eingeteilt:

IE- Kategorie

Definition 24: **IE- Kategorie**
Eine IE-Kategorie ist eine Markierung eines Text-IEs, welche die semantische Zugehörigkeit des IEs zu einer Klasse von Begriffen ausdrückt.

Indexlexikon

Definition 25: **Indexlexikon**
Ein Indexlexikon enthält die Menge aller Indexterme, die eindeutigen Abbildungsvorschriften des Indexvokabulars auf Indexterme und die eindeutige Zuordnung jedes Indextermes zu einer IE-Kategorie.

Zur im Folgenden erklärten Abbildung von Texten auf IEs werden fast immer mehrere Teillexika beteiligt. Eins existiert um allgemeinsprachliche Begriffe abzubilden. Hat die Anwendungsdomäne mit Computern zu tun, kommt ein Teil-Indexlexikon für Computerbegriffe hinzu. Ebenso werden

noch spezielle Lexika benötigt, wenn das FBS System eine spezielle Domäne behandelt. Wird nun ein FBS System für eine neue Anwendungsdomäne entwickelt, sind nur die Teil-Lexika für die neuen Fachbegriffe der Domäne neu zu erstellen und einzubinden.

Kein automatisches Reuse beim TCBR

Ein Nachteil des Textuellen Fallbasierten Schließens ist, dass der Reuse Prozess aus dem 4-R-Zyklus *(siehe Seite 14)* durch den Benutzer von Hand erledigt werden muss, da noch keine automatischen Verfahren zur Adaption der Falltexte existieren. Dafür wäre ein viel tieferes Verständnis mit einer vollständigen syntaktischen und semantischen Analyse der Texte Voraussetzung. Dies würde aber auf jeden Fall einen wesentlich höheren, wenn überhaupt realisierbaren Aufwand als bei der Erzeugung des Indexlexika und der Werte der lokalen Ähnlichkeitsfunktion bedeuten.

2.3.1 Abbildung der Fälle im Textuellen FBS auf IEs

Zum Aufbau der Fallbasis werden alle für den Retrieval-Prozess relevanten Textabschnitte der Fälle mit Hilfe des Indexlexikons auf Text-IEs abgebildet: Ein Parser liest jeden Text wortweise ein und sucht jeweils nach der längsten Folge von Wörtern, welche auf einen String des Indexvokabulars passt. Der Indexterm, auf den dieser String im Lexikon abgebildet ist, wird als Text-IE in die Menge der Text-IEs aufgenommen, welche den untersuchten Textabschnitt repräsentiert.
Die für den Retrieval-Prozess relevanten Attribut-Werte-Paare werden mit Hilfe einer Menge aller bekannten Attribut-Werte-Paare auf Attribut-IEs abgebildet.

Innerhalb der Fallbasis besteht der Fall aus einem Fallbezeichner, einer oder mehreren Mengen von Text-IEs und einer Menge von Attribut-IEs.

2.4 Wissenscontainer

Für die Einschätzung welches Wissen in einem FBS System benötigt wird, hat sich der Begriff des Wissenscontainers als geeignet erwiesen.

Wissenscontainer

> Definition 26: **Wissenscontainer (WC)**
> Wissenscontainer sind strukturelle Elemente, die das Wissen innerhalb eines wissensbasierten Systems enthalten.

Wissenscontainer (engl. knowledge container) lösen im Gegensatz zu Programmmodulen keine Aufgabe allein. Sie enthalten das Wissen, welches für viele Aufgaben innerhalb eines wissensbasierten Systems notwendig ist. Auch bei einfachen Aufgaben sind mehrere Wissenscontainer beteiligt. Wissenscontainer werden in dieser Arbeit mit WC abgekürzt.

2.4.1 Wissenscontainer im FBS

Richter identifiziert im FBS vier Wissenscontainer [Richter1998, S. 10]:

- Repräsentationssprache und Vokabular
- Ähnlichkeitsmaß
- Fallbasis
- Lösungstransformation

Dabei existiert keine strikte Trennung zwischen diesen vier, im Prinzip kann auch alles Wissen in einem einzigen Container stecken. Aber dies ist selten sinnvoll und so ist das Wissen gewöhnlich über alle vier Container verteilt, wobei die Verteilung sich mit der Zeit ändern kann. Die Fallbasis wird erst zur Laufzeit interpretiert wird und kann sich auch während dessen verändern, die anderen drei werden schon zur Übersetzungszeit benötigt. Die Fallbasis stellt also interpretiertes Wissen, die anderen drei kompiliertes Wissen dar. Das Vokabular stellt dabei eine Basis für die anderen drei dar, wie Bild 3 zeigt:

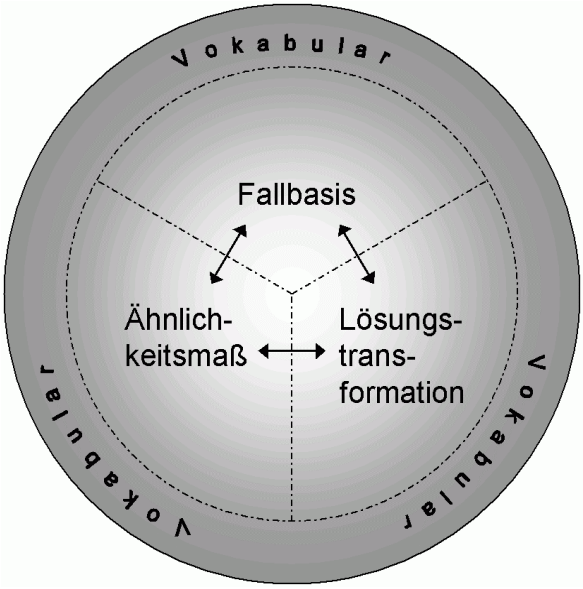


Bild 3: Knowledge Containter nach Richter

Da im Textuellen FBS Indexterme auf IEs abgebildet werden, besteht hier der erste Wissenscontainer nur aus dem Vokabular.
Bei TFBS-Systemen wird die Lösungstransformation vom Benutzer vorgenommen, da noch keine automatischen Verfahren für die Adaption von Texten existieren.

3 Literaturrecherche und Analyse

Im nächsten Unterkapitel wird ein Überblick über maintenance im Allgemeinen und speziell im FBS gegeben.
Daher werden einige der für das Gebiet der maintenance im FBS relevanten Arbeiten vorgestellt und eingeordnet. Dabei wird untersucht, ob sich diese Ideen für die hier gestellte Problemstellung eignen. Aus den vorgestellten Ansätzen und deren Eigenschaften werden die Anforderungen an das eigene Konzept abgeleitet.

Einen umfassenden Überblick über den Stand der Forschung auf dem Gebiet des maintenance für das FBS sowie eine Kategorisierung der verschiedenen Ansätze wird in *3.2 Kategorisierung nach Leake und Wilson auf Seite 23* vorgestellt. Dabei werden auch die am Lehrstuhl bestehenden Systeme klassifiziert.
Danach werden die Arbeiten eines systematischen maintenance Frameworks von Nick, Althoff und Tautz am Fraunhofer IESE in *3.3 Systematic Maintenance of Corporate Experience Repositories auf Seite 30* vorgestellt. Dabei wird auch ein explizites Knowledge Life-Cycle Modell verwendet.
Anschließend werden als Erweiterung des 4-R-Zyklus das 6-Schritte-Modell und die SIAM Methode von Th. Roth-Berghofer unter *3.4 Sechs Schritte und die SIAM Methode auf Seite 33* präsentiert.
Abschließend werden der Themenkomplex der Maintenance schwach strukturierter Fälle sowie die Arbeiten von Ferrario und Smyth zu Collaborative Maintenance vorgestellt.

3.1 Maintenance

In der Informatik wurde der Begriff maintenance zuerst im Software Engineering geprägt. Eine Definition von dorther liefert [Groenendijk1996]:

Maintenance

> Definition 27: **maintenance**
> Maintenance ist die Kombination aller technischen und damit verbundenen administrativen Tätigkeiten in der Absicht den Zustand eines Gegenstandes zu behalten oder wiederherzustellen, so dass er die erforderliche Funktionalität aufweist.

Nach [Swanson1976] gibt es in der Software maintenance drei Kategorien (nach dem Ziel):

- korrigierend (corrective),
- adaptiv (adaptive) und
- optimierend (perfective).

Während bei der ersten Kategorie auftretende Fehler korrigiert werden, befasst sich die zweite mit veränderten Anforderungen während des Ein-

satzes der Software. In der dritten Kategorie wird die Wartbarkeit oder Leistung verbessert, Ineffizienzen eliminiert.

3.1.1 Maintenance im FBS

Einer der ersten Artikel zu Maintenance in FBS Systemen ist jener von Smyth und Keane [SmythKeane1995], welcher sich mit Löschstrategien von überflüssigen oder schadhaften Fällen beschäftigt. Motivation war das utility problem, genauer ein Spezialfall davon, das swamping problem (zu deutsch überschwemmend):

utility Problem

Definition 28: **utility Problem**
Das utility Problem bezeichnet den Umstand, dass die Kosten (vornehmlich Zeit, Rechenleistung) für das Retrieval den Vorteil der Anwendung des Wissens (gefundener Fall) aufwiegen.

swamping Problem

Definition 29: **swamping Problem**
Das swamping Problem bezeichnet ansteigende Retrieval-Zeiten aufgrund immer weiter wachsender Fallbasen. Es ist ein Spezialfall des utility problems in der Domäne des FBS [FrancisRam1993].

Für das FBS definieren Leake & Wilson in [LeakeWilson1998]:

maintenance für FBS

Definition 30: **maintenance für FBS**
Case-based maintenance implementiert Strategien für die Überarbeitung der Organisation oder des Inhalts der Fallbasis, um zukünftige Schlussfolgerungsprozesse für einzelne (oder eine Menge von) Zielen hinsichtlich der Leistung zu verbessern.
Dies umfasst die Repräsentation, Inhalte der Domäne, Zugriffsinformationen und die Implementation der Fallbasis.

case-based maintenance

Die Autoren verwenden im Artikel dafür den Begriff «case-based maintenance» Fortan werden beide Begriffe äquivalent verwendet. Die Definition betrachtet die Informationen für die Indizierungsstrukturen als einen Teil der Fallbasis, deren maintenance ist also in dieser Definition miteingeschlossen.

Maintenance kann einzelne oder mehrere Fälle berühren sowie auf drei verschiedenen Ebenen erfolgen. Sie kann dabei die verwendete Fallrepräsentation, folglich welche Aussagen und Prädikate zur Beschreibung der Domäne-Merkmale verwendet werden, überarbeiten (Repräsentationsebene). Auf der zweiten Ebene wird die Implementierung der Repräsentation korrigiert, das können z.B. Merkmalsvektoren oder Attribut-Werte-Paare statt Listen oder Text sein. Oder die Informationen, wie oft auf einen Fall zugegriffen wird (Implementationsebene). Im dritten Fall wird Domänewissen verändert, was die Korrektur einzelner, fehlerhafter Merkmale eines Falls, das Löschen oder Hinzufügen eines Falls beinhalten kann (Wissensebene).

Zusammenfassend kann daher die Überarbeitung der Fallbasis auf der

- Repräsentationsebene,
- Implementationsebene oder
- Wissensebene

erfolgen, wie schon in [Dietterich1986].

3.1.2 Maintenance von Wissen

Fokus dieser Arbeit ist maintenance von Wissen bzw. Erfahrung. Th. Roth-Berghofer kombiniert in seiner Dissertation [Berghofer2003, S. 30] die allgemeine Definition von [Groenendijk1996] von *Seite 21* mit der von [LeakeWilson1998] zu maintenance für FBS auf *Seite 22*:

Knowledge Maintenance

> Definition 31: **knowledge maintenance**
> Knowledge maintenance eines FBS-Systems ist die Kombination aller technischen und verbundenen administrativen Tätigkeiten mit der Absicht den Zustand des Wissens eines FBS Sytems zu behalten oder wiederherzustellen, so dass es die erforderliche Funktionalität aufweist. Darin sind die Adaption an die Veränderungen der Anforderungen und Techniken ebenso wie das Feintuning eingeschlossen.

Während die Definition von [Groenendijk1996] nur korrigierende maintenance erfasst, schließt Berghofer alle drei Kategorien nach [Swanson1976] mit ein.

3.2 Kategorisierung nach Leake und Wilson

Kategorisierungs-schema: Wann und wie wird das CBR System überarbeitet?

Die Autoren Leake und Wilson [LeakeWilson1998], [Wilson2001] definieren ein Framework bzw. Kategorisierungsschema, welches die maintenance Ansätze anhand der verwendeten «maintenance policies» (zu deutsch etwa Wartungsstrategien) charakterisiert. Das Framework wird in den drei Prozessen

- Datensammlung (data collection),
- Auslösen der maintenance (triggering) und
- Ausführung (excecution)

sowie den

- Verfügbare Typen der maintenance Operationen (available operation types)

klassifiziert. Die folgenden Abschnitte 3.2.1 bis 3.2.4 beschreiben diese Dimensionen, vergleiche auch die Zusammenfassung in Bild 4 auf Seite 27. In den folgenden Definitionen sind in Klammern immer die englischen Bezeichnungen aus dem Original aufgeführt.

3.2.1 Datensammlung

Datensammlung

Der Prozess der Datensammlung sammelt die Daten aus der Fallbasis und über das Systemverhalten ein und bereitet sie auf. Dies können z. B. Informationen über konkrete Fälle sein, wie oft auf sie zugegriffen wurde, ob sie mit oder ohne Erfolg eingesetzt wurden. Oder Informationen über die Fallbasis als Ganzes, ihre Größe oder Eingabeprobleme, ungelöste Anfragen gehören dazu.
Datensammlung unterscheidet die Dimensionen *Typ der Datensammlung*, *Zeitsteuerung* und *Integration*.

Typ der Datensammlung

Dabei werden drei Arten, wann Daten eingesammelt werden, als *Typ der Datensammlung* unterschieden:

- keine (none),
- synchron (synchronic) oder
- entwicklungsgeschichtlich (diachronic).

Im einfachsten und ersten Fall werden keine Daten aus der aktuellen Fallbasis berücksichtigt und daher macht eine solche Strategie die Wartung unabhängig vom aktuellen Zustand der Fallbasis. Sie wird als *nicht-introspektiv* (non-introspective) bezeichnet. Anspruchsvoller ist die Strategie *synchron* (synchronic), wenn Bestandsaufnahmen der aktuellen Fallbasis (bzw. eines Teils) erstellt und mit einbezogen werden. Beim letzten und informativsten, dem *entwicklungsgeschichtlichen* Ansatz (diachronic) werden Daten bzgl. der Fallbasis über die Zeit gesammelt und ausgewertet. Ziel ist es dabei, Trends zu erkennen, wie sich der Inhalt und die Benutzung der Fallbasis verändern. Die letzten beiden Ansätze sind introspektiv (hineinschauend, engl. introspective).

Zeitsteuerung

Bei der *Zeitsteuerung* (Timing) wird in

- periodisch (periodic),
- konditional (conditional) und
- ad hoc

unterschieden. Eine *periodische* Zeitsteuerung bezeichnet maintenance Operationen, die sich in den FBS Zyklus einfügen, wobei es außerdem noch *kontinuierlich* (continuous) ist, wenn es in jedem FBS Zyklus passiert. *Konditional* bezeichnet Strategien, welche nicht periodisch, sondern infolge ganz bestimmter Bedingungen Daten sammeln. *Ad hoc* charakterisiert die zeitliche Steuerung einer Datensammlung welche nicht durch die Strategie bestimmt, sondern extern zum FBS System angestoßen wird, beispielsweise durch den Benutzer.

Integration

Die *Integration* kann entweder

- on-line oder
- off-line

erfolgen. Bei ersterem erfolgt die Datensammlung während eines aktiven Prozesses des FBS Systems (z. B. einer Anfrage), bei letzterem in einer Pause dazwischen. Dazu zählt auch, wenn das System währenddessen

abgeschaltet ist. Die Wahl zwischen beiden beeinflusst die verwendeten Resourcen und empfiehlt letztere bei zeitkritischen Prozessen.

3.2.2 Auslösung der maintenance

Auslösung der maintenance

Die Ergebnisse der Datensammlung werden als Eingabe weiter verwendet, um zu entscheiden, *wann* maintenance im FBS System notwendig ist. Dabei wird auch entschieden, welche der verfügbaren maintenance Operationen ausgeführt wird. Die Dimensionen *Zeitsteuerung* (Timing) und *Integration* werden wie bei der Datensammlung hier genauso verwendet, aber die *konditionale Zeitsteuerung* (conditional) wird noch weiter unterteilt.

Zeitsteuerung der Auslösung:

- periodisch,
- konditional
  - raumbasiert (space-based)
  - zeitbasiert (time-based)
  - ergebnisbasiert (result-based)
- ad hoc

Raumbasiert richtet das Auslösen nach der Größe (z. B. der Fallbasis) oder dem Platzbedarf aus, während eine zeitliche Grenze (z. B. fürs Retrieval) bei der *zeitbasierten* Auslösung nicht überschritten werden darf. An Ergebnissen wie Fehlern aufgrund ungelöster Fälle bzw. Retrieval ohne Ergebnissen orientiert sich die *ergebnisbasierte*, konditionale Auslösung.

Die *Integration* kann wieder entweder

- on-line oder
- off-line

sein.

3.2.3 Verfügbare Operationstypen

Verfügbare Operationstypen

Die verfügbaren Operationstypen werden nach dem Ziel der Operation, *Zieltyp* (Target Type), und der *Änderungsebene* (Revision level) unterteilt.

Der *Zieltyp* ist in diesem Kategorisierungsschema entweder

- Indizierungsstrukturen (indexing structures),
- Domäne-Inhalte (domain contents),
- Zugriffsinformationen (accounting information) oder
- Wartungsstrategien (maintenance policies).

Besonders den letzten Punkt, die Installation und Revision von maintenance Regeln nennen die Autoren «meta-maintenance» und widmen ihr ein eigenes Kapitel.

meta maintenance

Definition 32: **meta maintenance**
Meta maintenance bezeichnet die Installation und Revision von maintenance Strategien.

Die *Änderungsebene* differenziert nach drei verschiedenen Ebenen:

- Implementationsebene (implementation level),
- Repräsentationsebene (representation level) und
- Wissensebene (knowledge level),

auf denen die betreffende maintenance Strategie angewendet wird, wie schon in *3.1.1 Maintenance im FBS auf Seite 22* erwähnt.

Feinere Abstufungen sind denkbar und wurden auch schon von anderen Autoren z. B. in [HeisterWilke1998] und [ReinartzEtal2000] vorgeschlagen.

3.2.4 Ausführung

Ausführung

Die Ausführung der maintenance Operationen wird nach der schon in Datensammlung vorgestellten *Zeitsteuerung* und *Integration* sowie zusätzlich noch dem *Wartungsumfang* (scope of maintenance) differenziert.

Die *Zeitsteuerung* wird danach charakterisiert, ob die Ausführung zeitlich vom Zustand des FBS Systems abhängig ist oder es tangiert:

- periodisch (periodic),
- konditional (conditional) und
- ad hoc
- keine (none)

Das Aktualisieren von Fällen kann z. B. in einem Batch erfolgen, wenn eine genügende Anzahl zu ändernder Fälle ansteht – das wäre konditional. Die maintenance Strategie kann auch *keine Zeitsteuerung* haben, falls das System keine verändernden Operationen ausführt, sondern z. B. nur Informationen an den Benutzer ausgibt.

Die *Integration* in andere Systemprozesse des FBS geschieht entweder

- on-line oder
- off-line.

Des weiteren wird bei der Ausführung des *Wartungsumfanges* (scope of maintenance) nach

- begrenzt (narrow) oder
- weitläufig (broad)

differenziert, wobei ersteres zutrifft, wenn nur ein Fall oder eine kleine Teilmenge berührt werden. Im zweiten Fall sind dagegen eine großer Teil oder die gesamte Fallbasis betroffen.

Abschließend sei das Kategorisierungsschema im Bild 4 dargestellt:

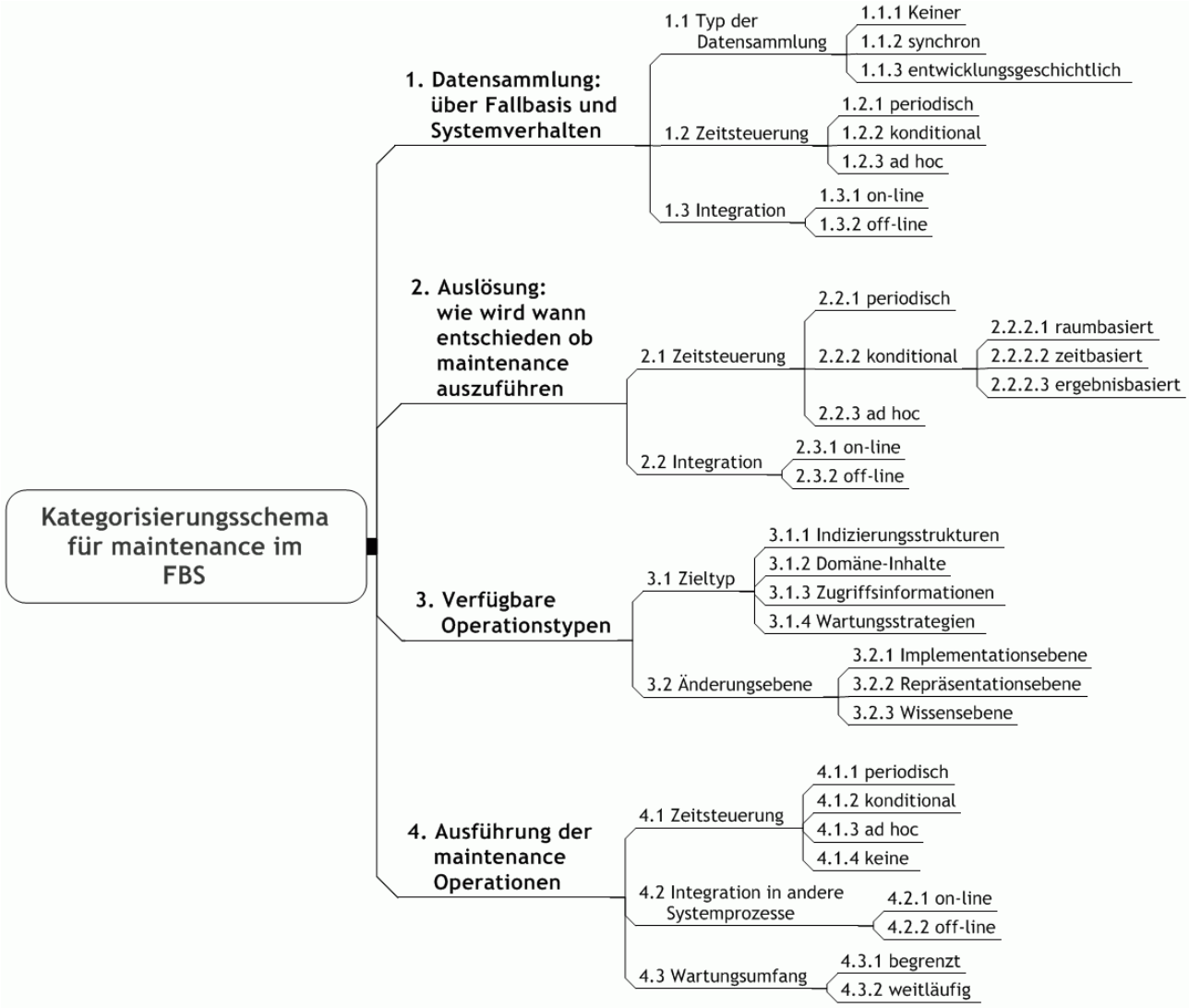


Bild 4: Kategorisierung von maintenance Ansätzen nach Leake & Wilson

3.2.5 Typische Ansätze

Beispiel CBR_1: Immer ein neuer Fall

Als Beispiele nennen die Autoren zwei übliche maintenance Ansätze CBR_1 und CBR_2, für welche sie die Datensammlung und die Ausführung erläutern, da diese am charakteristischsten sind. CBR_1 fügt bei nach jeder Retrieval-Anfrage einen neuen Fall zur Fallbasis. Dies entspricht auch dem 4-R-Zyklus aus [AamodtPlaza1994]. Der Fall wird ohne Betrachtung der Fallbasis eingefügt, keine Daten werden gesammelt, die *Datensammlung* ist daher *non-introspektiv*. Da das Lernen eines neuen Falls in jedem FBS-Zyklus erfolgt, ist die Zeitsteuerung *periodisch*, sogar *kontinuierlich* und der Wartungsumfang ist *begrenzt*.

Beispiel CBR_2: Neue Fälle nur durch Administrator

Eine andere verbreitete FBS maintenance Methode CBR_2 ist ein nicht automatisch lernendes System, welches durch einen FBS Administrator gewartet wird, der bei Bedarf einen oder mehrere Fälle hinzufügt. Dabei wird von Leake & Wilson angenommen, dass keine Analyse der Fallbasis

stattfindet, der Typ der Datensammlung also *nicht-introspektiv* ist. Das Ändern der Fallbasis erfolgt manuell und außerhalb der Systemprozesse, daher ist die Zeitsteuerung *ad hoc* und die Integration *off-line*. Der Wartungsumfang ist *begrenzt*, hängt aber von der Anzahl der veränderten Fälle ab, kann also auch *weitläufig* sein. Beide Beispiele adressieren mit ihren Operationen den *Domäneninhalt*. In seiner Dissertation führt David Wilson über ein Dutzend publizierter maintenance policies auf und ordnet sie in das Schema ein [Wilson2001, Seite 20ff].

3.2.6 Bewertung

Das Kategorisierungsschema von Leake & Wilson bietet einen detaillierten Ansatz zur Klassifizierung, welcher nicht nur für das FBS, sondern sicher auch für maintenance anderer prozesshafter und wissensbasierter Systeme geeignet ist. Die Autoren stellen dies mit der Einordnung vieler Systeme unter Beweis.
Auch wenn es meist das Ziel bei maintenance ist, diese zu automatisieren oder zumindest durch Software zu unterstützen, wird sie oft (zumindest zu einem Teil) von Hand durch den Benutzer erledigt. Dieser Prozess ist sehr wissensintensiv [NickAlthoff2001], da er vielfältige analytische Fähigkeiten verlangt und daher noch nicht immer komplett automatisiert werden kann.
Das Kategorisierungsschema eignet sich weniger für maintenance, die (teilweise) von Hand durch den Benutzer oder einen Administrator erfolgt. In diesem ad hoc Fall erscheint die Unterscheidung der Dimension Zeitsteuerung in jedem Prozess übermäßig, da Zeitsteuerung der Auslösung und Ausführung bei einer Person zeitlich zusammenfällt.
Auch lassen sich Abhängigkeiten im Prozess der maintenance zwischen System und einem oder mehreren Benutzern nicht klassifizieren, wie sie bei einer Autorisierung von Änderungen durch einen Administrator im ExperienceBook [Kunze1998] oder einem Review durch andere Nutzer in Ulysses [FerrarioSmyth2000] vorkommen.
Dass in CBR_2 überhaupt keine Analyse der Fallbasis stattfindet, kann nach Meinung des Autors dieser Arbeit für die meisten eingesetzten Systeme bezweifelt werden, da die Betreuer ihre Fallbasis sehr gut kennen und z. B. neue Fälle vor dem Hintergrund der bestehenden einfügen, um Wissenslücken zu schließen. Allerdings ist es richtig, dass dabei die Daten und die Analyse nicht zwingend maschinell und vom FBS System erfasst werden. Ob dabei auch der Verlauf der Fallbasis berücksichtigt wurde, liegt aber außerhalb der Festlegung einer formellen Strategie, der Typ der gesammelten Daten kann daher (statt keiner) synchron als auch entwicklungsgeschichtlich sein.

Kategorisierungs-schema statt Framework

Der hier von Leake & Wilson verwendete Begriff Framework ist nach Meinung des Autors im Deutschen ein wenig irreführend, da Framework als partiell vollständiges Software-System definiert wird, welches nur noch instanziiert und angepasst werden muss [BuschmannEtal1998, Seite 393]. Stattdessen schlägt der Autor vor, es im Deutschen Kategorisierung oder Kategorisierungsschema zu nennen, da es „nur“ eine gedankliche Strukturierung und Einordnung der maintenance Ansätze, aber keine prozesshafte Unterstützung bietet.

3.2.7 Einordnung der maintenance Strategien vom ExperienceBook

Anschließend werden die am hiesigen Lehrstuhl für Künstliche Intelligenz bestehenden Systeme klassifiziert, das ExperienceBook I und der Test-Manager/ ExpBook II.

Charakterisierung ExperienceBook

Beim ExperienceBook [Kunze1998] kann jeder Benutzer neue Fälle eingeben, welche dann an den Administrator geschickt werden. Der Administrator überprüft die Fälle und integriert sie danach in die Fallbasis wozu das System neu gestartet werden muss. Dies entspricht im Groben dem Typus CBR_2 oben, bis auf die fehlende Analyse der Fallbasis. Die Datensammlung ist also *synchron*, *ad hoc* und *off-line*. Bei der Auslösung wird die Zeitsteuerung als *konditional ergebnisbasiert* eingeschätzt, da der Benutzer eine neue Problemlösung eingibt, zu der er keine Lösung über das Retrieval in der Fallbasis fand. Diese Zuordnung ist aber nicht zweifelsfrei und könnte auch *ad hoc* sein, da die Eingabe eines neuen Falls nicht zwingend erfolgt.
Die Integration der Auslösung ist trotzdem *off-line*, da die Eingabe zeitlich nicht an einen Systemprozess wie das Retrieval gebunden ist.
Offensichtlich ist das Ziel der Veränderung beim ExpierenceBook wie auch in den vorherigen Beispielen nur der *Domäneninhalt* und in der Dimension der Änderungsebene nur die *Wissensebene*. Bei der Zeitsteuerung der Ausführung tritt der gleiche Interpretationskonflikt auf wie bei der Auslösung auf: Auf der einen Seite ist die Ausführung *konditional*, weil sie immer erfolgt, nachdem der Administrator eine Email mit einem neuen Fall oder einen Änderungswunsch bekommt, aber andererseits *ad hoc*, da dies nach frei wählbarer Zeit durch den Administrator und aus Sicht des Systems willkürlich erfolgt.
Der Wartungsumfang wird als *begrenzt* eingeschätzt, da immer nur ein Fall oder eine kleine Menge an Fällen neu hinzugefügt oder geändert werden.

3.2.8 Einordnung der maintenance Strategien vom TestManager und ExperienceBook II

Charakterisierung TestManager

Der bedeutendste Unterschied des TestManagers [Hanft2004] zum ExpBook I besteht u.a. darin, dass Fälle ohne einen Administrator in die Fallbasis integriert werden. Daher unterscheidet sich die Klassifizierung des TestManagers nur im Prozess Ausführung. Die Integration der Ausführung erfolgt *on-line*, da Fälle sofort übernommen werden. Dass dabei die Fälle erst nach einem erneuten Start des Systems für das Retrieval zu Verfügung stehen, wird nicht erfasst. Durch einen cron job, welcher den Server täglich zu festgelegter Uhrzeit herunterfährt und neu startet, ließe sich dies auch automatisieren, so dass die Integration offensichtlich online ist.
ExpBook II hat eine leicht andere Strategie: Neue Fälle werden unbesehen fürs Authoring zur Verfügung gestellt, aber vor der Freigabe für das Retrieval redaktionell bearbeitet. Die Ausführung ist also wie beim ExpBook I *off-line*.
Außerdem können nur beim TestManager auch IEs und die Ähnlichkeiten zwischen diesen durch den Benutzer verändert werden. Dies ist eine zusätzliche maintenance Strategie, welche mit ansonsten gleichen Charak-

teristika beim Zieltyp nicht den Domäneninhalt, sondern Indizierungsstrukturen adressiert.
Dies deckt sich auch mit der Einschätzung Wilsons selbst [Wilson2001, Seite 22], welcher sich dabei auf die Ausführungen zum Projekt in [MinorHanft2000b] bezieht.

3.3 Systematic Maintenance of Corporate Experience Repositories

continuous stream of experience

Die Autoren Nick, Althoff und Tautz stellen in [NickEtal2001] die Idee eines «continuous stream of experience» vor. Um das Lernen in einer Organisation zu ermöglichen, legt diese ein Experience Repository an. Weil das Lernen aus Erfahrung permanent erfolgt, benötigt das Experience Repository maintenance.

Experience Factory

Dafür wird das Konzept der sogenannten «Experience Factory» (EF) [BasilEtal1994] aus dem Software Engineering Bereich verwendet. Für diesen Begriff gibt es keine aädquate Entsprechung im Deutschen. EF ist eine logische oder physische Infrastruktur für das kontinuierliche Lernen aus Erfahrung in einer Organisation. Alles Wissen zu Projekten, Produkten usw. wird darin gespeichert. Angereichert wird dies durch explizit dokumentierte Erfahrungen: «lessons learned», die während der Anwendung des Wissens gewonnen wurden. Bei der EF heißen Fälle «experience packages» und werden in einer «experience base» (EB) gespeichert. Dabei wird explizit das menschliche Lernen aus Erfahrung nachmodelliert um infolgedessen das Lernen in einer Organisation zu unterstützen.

experience base

Als Beispiel einer EF wird das am IESE eingerichtete **Co**rporate **I**nformation **N**etwork (CoIN) vorgestellt, wo Erfahrungen aus der täglichen Arbeit gesammelt werden.

CoIN

Für die «experience base» (EB) genannte Fallbasis mit Erfahrungswissen identifizieren die Autoren zwei Phasen: die Erstellungsphase und den regulären Betrieb. In der Erstellungsphase werden die Objekte und die Zieldomäne der EB identifiziert, das konzeptuelle Domänenmodell entwickelt, der Prozess zur Erzeugung und Verbesserung der EB definiert und die aktuelle Infrastruktur der EB entwickelt. Dazu wird die Methodik DISER benutzt. DISER steht für **D**esign and **I**mplementation of **S**oftware **E**ngineering **R**epositories. Es ist eine Methode zu Entwurf und Implementierung eines Repositories für Erfahrungen (EbIS). EbIS bezeichnet ein erfahrungsbasiertes Informationssystem. Ein anderer Begriff dafür wäre erfahrungsbasiertes System in Analogie zum wissensbasierten System.

DISER

EbIS

Für den regulären Betrieb der EB werden

- aktuelle Fälle mit Erfahrung in der EB gespeichert,
- Erfolgskriterien überprüft und falls notwendig korrigierende Maßnahmen eingeleitet,
- strategische Entscheidungen verwirklicht, indem Teile aus der Erstellungsphase wieder geändert werden.

Danach entwerfen die Autoren ein ausführliches Schema für die maintenance, welches hier nur verkürzt wiedergegeben werden kann. Es ist

unabhängig von der Technik des EbIS (wie z. B. FBS) angelegt. Als erstes werden zwei Ebenen unterschieden:

- strategische Ebene für die Langzeit-Perspektive und
- operationale Ebene für die kurz- bis -mittelfristige Perspektive.

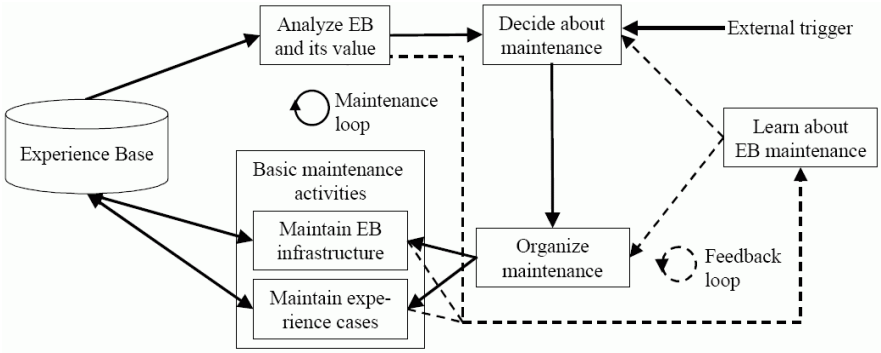
Im weiteren wird nur die operationale Ebene betrachtet, auf der zwei Schleifen implementiert werden:

- Wartungsschleife («maintenenance loop» für aktuell ausgführte maintenance) und
- Rückkopplungsschleife («feedback loop» für Verbesserung der maintenance aus Erfahrung)

Die Wartungsschleife besteht aus vier Schritten:

1. Analyse der EB und ihrer Benutzung,
2. Entscheidung über maintenance,
3. Organisation der maintenance und
4. maintenance der Fälle und der Infrastruktur der EB.

Eine Übersicht des Schemas zeigt Bild 5:



maintenance Schema im EMSIG Framework

Bild 5: Überblick das maintenance Schema EMSIG, entnommen aus [NickAlthoff2001]

Des weiteren stellen die Autoren Nick, Althoff und Tautz ein Satz von Rollen vor, die strategische bis zu operationalen Verantwortlichkeiten für die maintenance definieren. Daran anschließend wird die Implementierung des Konzeptuellen Modells einer FBS-basierten EB vorgestellt. Hier wird FBS auch für die maintenance der EB verwendet. Durch Messungen (Measurement Data) werden Richtlinien für die maintenance (Maintenance guidelines) aktiviert, welche eine oder mehrere Anforderung(en) zur Änderung der EB (Change Request) erzeugt. Aus diesem wie auch den Richtlinien werden einfache maintenance-Aktionen (Maintenance Primitive) angestoßen. Aus allen vier werden die Ergebnisse als neue Fälle (Experience Case) gespeichert. Grafisch wird dies in Bild 6 verdeutlicht:

Konzeptuelles Modell für die maintenance einer EB

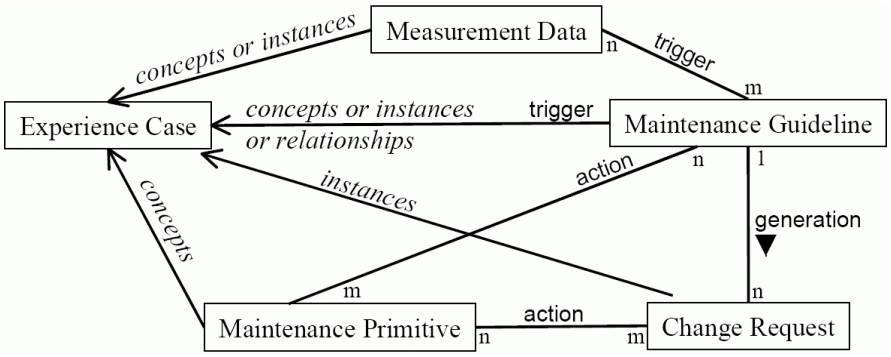


Bild 6: Konzeptuelles Modell für die maintenance einer EB, entnommen aus [NickAlthoff2001]

1,n und m verdeutlichen die Kardinaliäten 1:n und n:m der Toplevel-Konzepte zueinander. Das hier vorgestellte Konzept wird im EMSIG Framework implementiert, welches auf DISER aufbaut.

Einordnung:

Das Framework ermöglicht es maintenance als einen kontrollierten Prozess zu planen und durchzuführen. Da ad hoc ausgelöste maintenance keine adäquate Vorgehensweise darstellt, ist es wichtig Maintenance als systematischen, automatisch ausgeführten Prozess zu etablieren.
Das vorgestellte Framework ist sehr umfangreich und kann viel mehr als die in der Problemstellung geforderten maintenance Aufgaben umsetzen. Die Größe und Mächtigkeit des Frameworks erfordert aber auch viele Formalien, kann für Neulinge zu komplex und schwer verständlich sein. Daher wird im *mtBook* ein viel einfacheres Modell ohne Rollen entworfen, das dafür aber automatisch ohne Rollen ausfüllende Personen abläuft.
Vom Konzeptionellen Modell des EMSIG wird die nur die Idee übernommen, Maintenance über Messungen aufgrund einer Richtlinie auszulösen und daraufhin einzelne Aktionen auszuführen, welche dann die Maintenance darstellen.
Der hier verwendete Begriff des life-cycle einer EB ist wesentlich gröber und nicht mit dem *Modell des Life-Cycle eines Falls auf Seite 43* vergleichbar, da er sich auf die Entwicklung der kompletten Fallbasis bezieht.
Dieses Konzept arbeitet offenbar mit Strukturellem FBS, so dass es sich schwer auf die Zieldomäne mit Textuellen FBS übertragen lässt. EMSIG wird mit Hilfe kommerzieller Software umgesetzt, auf die in dieser Arbeit verzichtet wird.

3.3.1 Knowledge Life-Cycle Modell von Nick & Althoff

knowledge/ experience Life-Cycle model

Die Autoren Nick und Althoff beschreiben in [NickAlthoff2001] und [AlthoffEtal2001] ein sogenanntes *knowledge Life-Cycle model* bzw. *experience Life-Cycle model*. Idee dieses Modells ist schon beim Design eines Erfahrungsmanagement-basierten Systems zu analysieren, wie das Wissen sich entwickelt und daher gewartet werden soll. Dazu werden alle wichtigsten Wissensarten grob definiert und deren Beziehungen bzgl. der Wissensweiterentwicklung und Reifegrad untersucht. Das *knowledge/ experience Life-Cycle model* bietet dann die Basis für weitere Verfeinerungen des maintenance Prozesses und Verbesserung des Wissens bzw. der Erfahrung im System. Dabei unterscheiden die Autoren

zwei Bereiche: *Life-Cycle von Fällen eines Typs* und *Life-Cycle von Wissen verschiedener Typen*. Bei ersteren geht es vornehmlich um Fragen der Gültigkeit und des Revisionsstatus bzw. Reifegrades. Gültigkeit meint hier wie generell die beschriebene Erfahrung und wie wahrscheinlich eine erfolgreiche Wiederanwendung ist. Die Differenzierung in *Life-Cycle von Wissen verschiedener Typen* von Fällen ist notwendig, da verschiedene Typen von Fälle existieren: Gemachte Erfahrungen, «lessons learned» und «best practices». Der *Life-Cycle von Wissen verschiedener Typen* definiert nun wann und wie die Fälle von einem in den anderen Typ transformiert werden. Z. B können Fälle vom Typ «lessons learned» in «best practices» umgewandelt werden. Diese sind kürzer, reifer und für ein breiteres Publikum bestimmt.
Aus diesem *knowledge/ experience Life-Cycle* werden dann maintenance policies abgeleitet, welche informal beschreiben wann, warum und wie in der Fallbasis für Erfahrungswissen maintenance ausgeführt wird. Dabei konzentriert man sich auf die Behandlung von der fehlgeschlagenen Wiederanwendung von Fällen sowie der oben beschriebenen Transformation zwischen den unterschiedlichen Falltypen.

Einordnung:

Die Herangehensweise, schon beim Design ein komplettes Modell der Entwicklung des Wissens zu erstellen, ist hervorragend geeignet, die maintenance des Wissens später sicher zu stellen. Durch die Transformationen wird das Wissen verbessert: von gemachter Erfahrung mit viel Kontext über «lessons learned» zu generalisierten «best practices». Das Konzept wird innerhalb des EMSIG Frameworks umgesetzt.
Das hier vorgestellte *knowledge/ experience Life-Cycle Modell* wird nicht verwendet, da im *mtBook* keine Wissensverdichtung modelliert werden soll.

3.4 Sechs Schritte und die SIAM Methode

In seiner Dissertation stellt Th. Roth-Berghofer [Berghofer2003] u.a. das Modell «6 steps in CBR» vor. Dazu analysiert er das vorherrschende Modell des 4-R-Zyklus von Aamodt & Plaza (siehe *2.2.1 4-R-Zyklus von Amodt & Plaza auf Seite 14*). Die bestehenden vier Prozesse werden in einzelne Aufgaben (tasks) zerlegt. Folgende Nachteile vor dem Hintergrund von maintenance treten zutage:

- Keine Unterscheidung, ob ein Schritt/Prozess das Wissen verändert oder nicht,
- Keine Selbstbeobachtung um Systemveränderungen festzustellen,
- Keine Daten zur Vorbereitung der Wartung gesammelt.

«six steps» im FBS

Deswegen werden dem Zyklus zwei Schritte Review und Restore hinzugefügt und die bestehenden Schritte verfeinert. In diesem «six steps» Modell werden die Schritte der Anwendungs- oder der maintenance-Phase zugeordnet. Review misst und überwacht den Zustand eines FBS-Systems. Dazu wird ein Fall mit der klassischen Aufteilung in Problem und Lösung um «Qualitätsinformationen» erweitert. Zu den Qualitätsinformationen zählen u.a. die Zeit der letzten Benutzung (beim Retrieval) und die Zeit

der letzten Änderung. Beim Restore erfolgen die Änderungen, um die gewünschte Funktionalität des Systems wiederherzustellen. Beide gehören zusammen mit dem „alten“ Retain gehören zur maintenance Phase.

Darauf aufbauend wird die SIAM Methode entwickelt, welche für **S**etup, **I**nitialisierung (**I**nitialization), Anwendung (**A**pplication) und **M**aintenance steht.

INRECA

Die INRECA Methode, Ergebnis des gleichnamigen Forschungsprojekts, ist ein Framework zur Analyse und Entwicklung von FBS-Systemen im industriellen Umfeld (siehe [BergmannEtAl1999]. Sie deckt die Phasen Setup, Initialization und Application schon ab. SIAM erweitert also INRECA um die maintenance Phase, wobei Wechselwirkungen mit der Anwendungsphase bestehen. Zur Spezifikation der maintenance Phase besteht das Prozessmodell von SIAM aus drei verschiedene Abstraktionsstufen (allgemein, spezifisch und Instanz) und definiert Mappings zum Übergang zwischen diesen. Die sich daraus ergebenden Aufgaben (tasks) müssen von verschiedenen Komponenten ausgeführt werden.
Dem Autor zufolge stellt SIAM eine weiter gefasste Perspektive auf das FBS dar, da es sich z. B. auch auf andere Prozessmodelle wie CRISP-DM aus dem Data-Mining anwenden lässt. SIAM deckt den kompletten Life-Cycle eines FBS Systems ab. In der Arbeit die Implementierung von SIAM mit Hilfe des Systems orenge der Firma empolis gezeigt.

Einordnung:

Die Einführung expliziter Prozesse für die maintenance ist sinnvoll. Die Unterteilung aller Schritte in die Anwendungs- und maintenance-Phase ist adäquat, um die fortlaufend notwendige maintenance in den normalen Zyklus bei der Benutzung des FBS-Systems mit einzubeziehen.
Der hier verwendete Begriff des Life-Cycle ist nicht mit dem *Modell des Life-Cycle eines Falls auf Seite 43* vergleichbar, da er sich auf die Entwicklung eines kompletten Systems bezieht und den Lebenszyklus einzelner Elemente nicht modelliert.
Ebenso wie SIAM hat *mtBook* einen explizit modellierten Review Prozess. SIAM wird mit Hilfe kommerzieller Software umgesetzt, auf die in dieser Arbeit verzichtet werden soll.

3.5 Maintenance schwach strukturierter Fälle

Die Autoren Racine & Yang stellen in [RacineYang1997] maintenance von explizit unstrukturierten, textuellen Fällen vor, welche durch einen Agent der Software CodeAdvisor[3] verrichtet wird. Sie unterstreichen die Wichtigkeit der maintenance von FBS-Systemen und nehmen aufgrund der Gleichartigkeit zur Software maintenance an, dass die maintenance von FBS-Systemen wie bei dieser einen ähnlichen Anteil von 50–70% der gesamten Entwicklungskosten beträgt.
Sie verweisen mit Recht darauf, dass die meiste Literatur zu maintenance stark strukturierte Fälle mit eingeschränkten, wohl definierten Merkmalen (also AVPs, im engl. Original feature values) behandelt. Diese Fälle sind

[3] CodeAdvisor™ ist eine eingetragene Handelsmarke. Unter der im Artikel angegebenen Webseite ist die Software nicht erreichbar, sondern unter unter http://www.cs.sfu.ca/~isa/ erreichbar und für Bildungszwecke herunterladbar.

zwar leichter zu ändern, aber die initiale Transformation der Dokumente in Fälle als auch die nachträgliche Einführung von Merkmalen ist aufwändig.

Für die zweite Art, die unstrukturierten Fälle, werden zwei Probleme angesprochen, Inkonsistenz und Redundanz.

Inkonsistenz

Inkonsistenz existiert in zwei Formen:

- Inkonsistenz mit dem Hintergrundwissen der Domäne und
- Inkonsistenz mit anderen Fällen,

wobei die erste einen Widerspruch mit dem Domänenwissen darstellt, der auch durch eine Falschschreibung hervorgerufen werden kann. Letztere Form tritt offensichtlich bei zwei sich widersprechenden Abschnitten verschiedener Fälle auf.

Konsistenzverletzung

Diese Verletzung der Konsistenz kann als

- Weiche Konsistenzverletzung (soft constraint violation) oder
- Harte Konsistenzverletzung (constraint violation)

auftreten. Dabei stellt die weiche Verletzung einen unüblichen Wert für ein Attribut dar, welcher als Warnung an den Benutzer zur Überprüfung ausgegeben wird. Harte Verletzungen sind echte logische Widersprüche.

Redundanz

Die Redundanz tritt auf, wenn ein Fall einem anderen gleicht oder ihn (inhaltlich) einschließt.
Durch Methoden des «Information Retrieval» werden die Fälle geparst, um Schlüsselwörter und Phrasen zu extrahieren. Diese werden als normalisierte Repräsentation des Falls bezeichnet, auf die dann die Module zur Redundanz- und Inkonsistenzerkennung angewendet werden. Für die Erkennung von Inkonsistenzen (mit dem Domänenwissen) werden sogenannte «string-based guidelines» eingesetzt, welche eine Folge von unmöglichen Kombinationen von Schlüsselwörtern definieren (z.B. „Farbband Laserdrucker“). Weiterhin können Inkonsistenzen im numerischen Bereich erkannt und dem Benutzer vorgelegt werden. Bei der Redundanz werden zur Erkennung der Equivalenz eine «nearest-neighbor similarity» Funktion sowie für das Enthaltensein drei Regeln verwendet, welche für unterschiedliche Variationen von zusätzlichen Werten im Problem- und Lösungsteil des anwendbar sind.

Einordnung:

Racine and Yang konzentrieren sich bei der maintenance schwach strukturierter Fälle auf die Erkennung von Inkonsistenzen und Redundanzen beim Einfügen neuer Fälle. Das Ändern von Fällen (durch unterschiedliche Nutzer) wird nicht betrachtet. Die Extraktion von Schlüsselwörtern und Phrasen ist mit der von IEs teilweise vergleichbar (wie in *2.3.1 Abbildung der Fälle im Textuellen FBS auf IEs auf Seite 19*), da bei den IEs auch eine Rückführung auf Stammformen durchgeführt wird. Allerdings erfolgt die Festlegung eines IEs aufgrund eines hinterlegten Wörterbuchs, während ein Schlüsselwort bei Racine and Yang statistisch aufgrund der Konzentration auf eine Teilmenge von Fällen entsteht. Auch existiert bei den IEs keine im Schlussfolgern verwendete, tiefere Hierarchie oder Gruppierung

wie bei den Phrasen. Vererbungs- und Spezialisierungsbeziehungen können zwischen IEs zwar modelliert werden, aber sie werden im CRN wie alle anderen Ähnlichkeiten auf nicht mehr unterscheidbare numerische Werte abgebildet.

Die Inkonsistenzerkennung mit Hilfe der «string-based guidelines» könnte als Erweiterung auch für die *mtBook* verwendeten IEs übernommen werden. Dies wäre dann eine zusätzliche Wartungsstrategie Die Übertragung der Regeln zur Redundanzerkennung auf IEs wird dagegen als zu ungenau und wenig erfolgversprechend eingeschätzt, da die Methoden des IR auf statistischen Maßen basieren und erst ab mehreren 10.000 Fällen funktionieren.

3.6 Collaborative Maintenance mit Ulysses

Erstes Framework mit collaborative maintenance: Ulysses

Maria Ferrario und Barry Smyth stellen in [FerrarioSmyth2000] ein Framework namens *Ulysses* für das collaborative maintenance vor. Die Domäne enthält (Fälle über) gastronomische Einrichtungen oder Ereignisse um die Stadt Dublin. Benutzer sind alle Freizeitinteressierten, die sich beim *Ulysses* registrieren. Das System Ulysses basiert auf Benutzer-Profilen und präsentiert jedem Benutzer eine individuelle Portalseite. Aufgrund ihrer Aktivitäten Artikel anlegen, kontrollieren oder nur lesen werden sie als subscriber (information suppplier), reviewer (information mediators) oder information consumers klassifiziert. Ein neu angelegter Fall (hier: Case Update) wird automatisch information mediators (reviewern) zum Review vorgelegt (Review Posting), welche aufgrund ihres Profils die Beschreibung der Lokalität begutachten können.

Ein neuer bzw. geänderter Fall wird übernommen, wenn von Seiten der Begutachter (Reviewer) eine gewisse Homogenität in der Bewertung dieses Falls besteht. Ein Fall besteht aus sechs Merkmalen (features), denen eine Menge von diskreten Werten zugeordnet ist. Der Review besteht aus den „korrigierten“ Werten. Daher ist es möglich, numerisch aus unterschiedlichen Fällen zu einem gleichen Subjekt die Ähnlichkeit zwischen dem neuen Fall und der Version jedes Reviewers zu berechnen (Review Analysis), den Review Score.

Für die Annahme existieren verschiedene Strategien: Einfache Strategie (simple strategy) and die Meinungsstrategie (judgement strategy). Bei der einfachsten wird der neue Fall nur verworfen, wenn die durchschnittliche Ähnlichkeit zwischen einem Review und dem Fall unter einem Schwellwert liegt, ansonsten akzeptiert. Der Autor (subscriber) wird als information mediator, als Quelle verlässlicher Information eingestuft, wobei sich dies aus einer der Mehrheit entsprechender Einschätzung herleitet. Dabei existiert auch eine Korrekturfunktion, in welcher der Merkmalswert (feature value) auf den Wert der Mehrheit eingestellt wird.

collaborative maintenance

Definition 33: **collaborative maintenance**
Die Wartung eines FBS Systems wird durch die Benutzer selbst durchgeführt. Ergänzungen und Änderungen werden dabei durch Zusammenarbeit mehrerer Benutzer gegenseitig abgestimmt, was vom System unterstützt wird und so die Qualität der Änderungen bzw. Erweiterungen sicherstellt.

Die Verfügbarkeit eines verteilten Frameworks für die Unterstützung beim Review von Veränderungen zur Verfügung ist der Schlüssel des collaborative maintenance.

3.6.1.1 Einordnung von Ulysses und Abgrenzung zum TestManager

Das Ulysses Framework zeigt eindrucksvoll, wie die Pflege der Fälle eines FBS Systems in einer sich verändernden Domäne nur durch die Benutzer durchgeführt werden kann. Collaborative Maintenance stellt dabei Korrektheit und Konsistenz sicher. Die Begrenzung der Fälle auf Attribute und deren diskreter Wertebereich hat für die Berechnung einer Übereinstimmung viele Vorteile. Insbesondere die automatische Abänderung einzelner Merkmals-Werte ermöglicht es erst, die Reviews in Form eines geänderten Falls durch eine Software zu vergleichen.
Die Idee, die Übereinstimmung von Reviews zu vergleichen, könnte man auch auf den TestManager übertragen. Dafür bieten sich zwei Möglichkeiten an:

1. Vergleich der Reviews, welche Textkommentare zum vorgestellten Falltext sind. Es werden Schwierigkeiten aufgrund zu unterschiedlicher Ausdrucksmöglichkeiten für gleiche Aussagen z. B. “interessant, muss aber noch überarbeitet werden” aus (wie Paraphrase Problem, [LenzEtal1998, S. 119]) prognostiziert.
2. Der Review stellt den verbesserten Falltext dar, sofern eine Korrektur notwendig war. Diese Idee scheint verwendbar, aber diese Ähnlichkeit ist nicht so stringent herleitbar. Auch erscheint eine maschinell errechnete Abweichung zwischen neuem Fall und Review anhand mit Hilfe des TFBS nicht exakt genug, wenn nur einzelne Wörter geändert werden.

Konkreter: bei diskreten Wertebereichen folgt aus höherer Ähnlichkeit mehr Übereinstimmung in den AVPs. Bei TCBR mit IEs könnte eine unterschiedliche Ähnlichkeit von zwei Reviewtexten (die sich z. B. bis auf ein Wort unterscheiden) zu einem Fall auch auftreten weil dieses Wort nicht im Indexvokabular vorkommt, obwohl die Texte sonst gleich ähnlich zum Fall sind.

Eine strengere Strukturierung der Fälle wie es bis jetzt nur bei einzelnen AVPs (Projekt TestManager: Kategorie und Themenbereich, ExpBook II: System und Gerät) vorliegt wäre denkbar. Dies ist nach Erachten des Autors aber nicht auf die Testfallbeschreibung und -instruktionen bzw. die Problembeschreibungen des ExpBook anwendbar. Eine Einschränkung des Erwarteten Resultats im TestManager auf eine Menge von Werten erscheint mit erhöhtem Aufwand realisierbar. Da alle drei genannten Abschnitte als blanker Text vorliegen, ist ein automatisches Ändern auf

einen Mittelwert wie bei den diskreten, eingeschränkten Wertebereichen nicht möglich.
Das Konzept des Reviews wird übernommen, auf den Vergleich von Texten in Reviews aber innerhalb des *mtBook* verzichtet.
Eine Übernahme der Idee des kompletten Ulysses Frameworks würde weitreichende Änderungen an der Oberfläche erfordern, da der TestManager und ExpBook kein individuelles Portal für jeden Nutzer bietet. Benutzer ließen sich anhand der AVPs oder der IEs eingegebener Fälle als Tester für bestimmte Module der Software identifizieren. Dann wäre eine Kontrolle durch Benutzer mit übereinstimmendem Profil bei neu angelegten Fällen sinnvoll realisierbar.
Die Übernahme wurde anhand des Projekts TestManager beschrieben, dies gilt aber für die Falldaten des ExpBook I in gleicher Weise.

3.7 Anforderungen für ein eigenes Modell

Die vorangegangene Recherche der Literatur hat gezeigt, dass kein Ansatz verfügbar ist, der ein einfaches Modell für das Collaborative Maintenance von Texten im Fallbasierten Schließen bietet.
Die Beispieldomäne von *mtBook* sind administrative und technische Probleme im Umgang mit der EDV an einem Lehrstuhl. Dafür steht kein Administrator oder „knowledge engineer“ extra zur Verfügung, der sich darum kümmern kann. Die maintenance und Eingabe neuen Wissens muss quasi nebenbei durch die Mitarbeiter erfolgen. Dem entsprechend soll *mtBook* für Nutzergruppen anwendbar sein, welche keinen Administrator bereitstellen können und wo die Benutzer nur sehr wenig Zeit haben. Daher muss das Modell flink in der Handhabung und ohne unnötigen Overhead sein. Da es keinen Administrator gibt, muss maintenance automatisch ausgelöst werden.

Administration „nebenbei“

Da die maintenance nicht durch eine Autorität wie einen Administrator erfolgt, sollen alle Versionen einer Erfahrung behalten und Löschungen notfalls wieder rückgängig gemacht werden können.

Mächtige Frameworks oder Methoden erfordern durch ihren hohen Grad der Detailliertheit eine umfassende Analyse der Prozesse in einer Organisation. Für non-profit Organisationen ist es zu kompliziert, ein Geschäftsprozessmodell zu erstellen. Selbst kleine und mittlere Unternehmen sind mit der Analyse ihrer Geschäftsprozesse oft zeitlich und personell überfordert [MühlbrFeggeler2003].

Collaborative Maintenance setzt nicht nur eine Kultur, Wissen zu teilen, sondern auch die Bereitschaft zur Teilnahme am Prozess der Wissenserstellung voraus. Diese Bereitschaft entspringt z. B. aus der Überzeugung, daraus einen Nutzen zu ziehen, ohne größere Einarbeitung in ein Prozessmodell auf sich zu nehmen. Eine leichte Verständlichkeit der Prozesse für Benutzer ist also gefordert.

Kurzum: das Verfahren soll praktikabel sein, nicht die laufende Arbeit durch zu viel Formalismen behindern und sie automatisch unterstützen, wo es die Arbeit tatsächlich erleichtert. Gesucht wird ein leichtgewichtiges Prozessmodell, welches folgenden Anforderungen genügt:

- Flinke Handhabung,
- Geringer Overhead,
- Ohne Administrator, Maintenance automatisch ausgelöst,
- Keine Geschäftsprozessanalyse und
- Leichte Verständlichkeit

In dieser Arbeit wird ein eigenes Prozessmodell ohne eines der vorgestellten Frameworks erstellt, welches Collaborative Maintenance nach der Definition von Ferrario & Smyth implementiert. Von dorther wird die Idee des Reviewprozesses übernommen, allerdings wird der Review ganz anders gestaltet und die Berechnung der Bewertung ebenfalls vereinfacht. In einfacherem Maßstab als EMSIG wird Maintenance über Messungen aufgrund einer Richtlinie ausgelöst und mündet in einzelnen Aktionen, welche dann die Maintenance darstellen. Für die Weiterentwicklung und maintenance von Wissen wird eigene Modell des Life Cycle ausgebaut und verallgemeinert.
Im Anschluss an die Vorstellung der neuen Konzepte im nächsten Kapitel werden diese in *4.5 Vergleich mit bestehenden Ansätzen auf Seite 57* nochmals neben den bestehenden Ansätzen eingeordnet.

4 Neue Konzepte

4.1 Modell des Life-Cycle eines Falls

Das Modell des Life-Cycle eines Falls wurde wie in [MinorHanft2000] beschrieben entworfen und bildet die Entwicklung eines unvollständigen Falls bis zum kompletten Fall ab. Der Begriff Life-Cycle wird in dieser Arbeit trotz adäquat vorhandener Übersetzung Lebenszyklus nicht übersetzt, da er in der Literatur verwandter Fachgebiete wie dem Software Engineering schon länger verwendet wird. Ein Beispiel für einen typischen Life-Cycle eines Testfalls zeigt Bild 7. Dabei wird ein Fall mit seiner Erzeugung „geboren“. Der Testfall kann zu diesem Zeitpunkt auch nur als vage Idee oder aus einer einzelnen Textzeile bestehen. Mit jeder Änderung und Ergänzung, z.B. dem Eintragen der Test-Instruktionen oder dem erwartetem Ergebnis reift der Fall. Nach jeder Durchführung des Tests wird das Resultat „erfolgreich“ oder „nicht erfolgreich“ vermerkt. Mit dem eventuellen Löschen eines Falls aus der Fallbasis stirbt ein Fall und dessen Life-Cycle endet.

Modell des Life-Cycle eines Testfalls

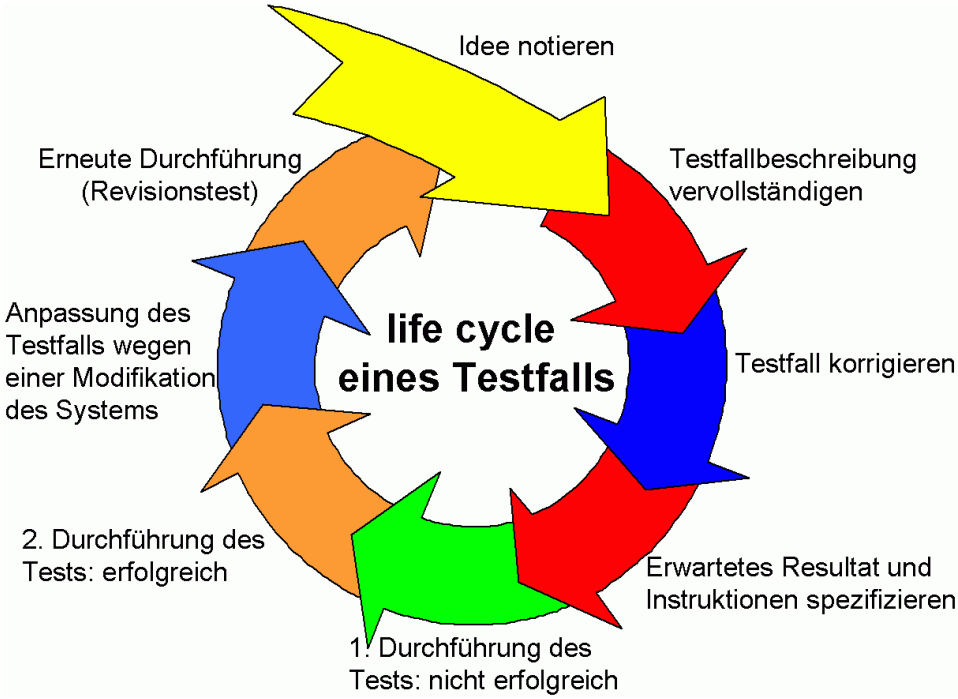


Bild 7: Ein typischer Life-Cycle eines (Test-)Falls

Jeder Fall erhält zur Identifizierung eine lebenslängliche, eineindeutige Nummer – die persistente Fall-Id.
Um die unterschiedlichen Stadien des Reifegrades eines Testfalls zu unterscheiden, wurde die Revisionsnummer eingeführt, die sich mit jeder (abgespeicherten) Änderung am Fall erhöht. Eine höhere Revisionsnummer zeigt also das Fortschreiten im Lebenszyklus bzw. den höheren Reifegrad an.

Grundlage für das Retrieval ist die Vergleichbarkeit fragmentarischer (unvollständiger) mit vollständigen Fällen, welches über eine dreischichtige Ähnlichkeitsfunktion erreicht wird (siehe dazu wieder [MinorHanft2000]). Dazu wird die Ähnlichkeitsfunktion nicht auf den kompletten Fall bzw. Anfrage, sondern immer jeweils abschnittsweise angewendet. Habe jeder Fall *m* Textabschnitte und sei S^l_{case} der *l*-te Textabschnitt eines Falls sowie S^l_{query} der korrespondierende Textabschnitt der Anfrage, so wird daraus die partielle Ähnlichkeit über

$$sim(S^l_{query}, S^l_{case}) = \sum_{e_k \in S^l_{query}} \sum_{e_i \in S^l_{case}} sim(e_k, e_i) \text{ berechnet.}$$

Sie ist per Definition 0, falls einer der beiden Abschnitte leer ist. Diese partiellen Ähnlichkeiten werden mit der Anzahl beidseitig gefüllter Abschnitte α normiert und ergeben aufsummiert die globale Ähnlichkeit zwischen Anfrage und Fall sim(Query, Case) mit

$$sim(Query, Case) = \sum_{l=1}^{m} \frac{1}{\alpha} sim(S^l_{query}, S^l_{case}).$$

Dadurch bleibt die Ähnlichkeit robust gegenüber der Vervollständigung von Fällen, d.h. vollständige Fälle sind nicht per se ähnlicher zur Anfrage als unvollständige.

Definition 34: **Life-Cycle eines Falls**
Der Life-Cycle eines Falls bildet die Entwicklung eines unvollständigen Falls bis zum kompletten Fall ab.
Der Life-Cycle eines Falls besteht aus

Bestandteile des Life-Cycle

- persistenter, eindeutiger Fall-Identifikation,
- Revisions-Nummer eines Falls und
- dreischichtiger Ähnlichkeitsfunktion zur Berechnung über unterschiedlich komplette Fälle.

4.1.1 Unterschiedliche Sichten zur Konsistenzwahrung im CRN trotz maintenance

Unterschiedliche Sichten zur Konsistenzwahrung notwendig

Neben einem jederzeit möglichen Retrieval soll die unmittelbare Integration von Änderungen an Fällen ermöglicht werden. Diese Verflechtung von Retrieval und «case authoring» wurde auch schon in [MinorHanft1999] beschrieben.
Nach bisherigem Stand des bestehenden FBS Systems muss für die Berücksichtigung neuer und geänderter Fälle beim Retrieval das Case Retrieval Netz (CRN) mit beträchtlichem Zeitaufwand neu aufgebaut werden. Eine Ergänzung um Fälle zur Laufzeit ist theoretisch möglich. Nach Änderungen an einem Fallen wird dieser evtl. durch andere IEs repräsentiert. Das hätte das Löschen und Hinzufügen von Relevanzkanten zur Folge. Während dieser Zeit wäre kein Retrieval möglich, was aber gefordert wird. Änderungen am Indexvokabular sind überhaupt nicht sinnvoll in ein bestehendes CRN integrierbar, da dazu alle Falltexte erneut geparst werden müssen. Daher wird die Integration neuer Fälle in ein

bestehendes CRN auch wegen des Implementierungsaufwandes verworfen.
Bei einem Retrieval sollen gefundene Fälle mit den für das Retrievalergebnis relevanten IEs im Text präsentiert werden. Durch mögliche Änderungen an diesem Fall ist aber nicht sichergestellt, dass der neue Falltext noch die gleichen repräsentierenden IEs enthält. Daher sind mehrere Revisionen eines Falls zu behandeln und zwei ständig verfügbar zu halten, eine Revision für das Retrieval in der «retrieval view» und die aktuellste Revision in der «authoring view», falls sie durch einen oder mehrere Benutzer geändert wurde.

4.2 Collaborative Maintenance für das Life-Cycle Modell

4.2.1 Vom Vier-Augen-Prinzip zur Autorisierung von Änderungen an Fällen

Im TestManager wird eine Änderung an einem Fall ohne weitere Kontrolle übernommen, abgesehen von der technischen Unterscheidung in «retrieval view» und «authoring view».

Vier-Augen-Prinzip

In vielen Bereichen, wo Dokumente rechtliche Konsequenzen oder finanzielle Auswirkungen haben oder bestimmten Qualitätsstandards genügen müssen, wird bei der Erstellung von Dokumenten das Vier-Augen-Prinzip verlangt. Dabei kann dies hierarchisch mit Kontrolle durch einen Vorgesetzten oder peer-to-peer durch einen anderen (kompetenten) Mitarbeiter unabhängig von der Hierarchie erfolgen.
Auch für offene Systeme im non-profit Bereich, „Communities“ kann dieses Prinzip sinnvoll sein, um unsinnige Änderungen „böswilliger“ oder unwissender Anwender zu unterbinden. Auch Flüchtigkeitsfehler können dadurch minimiert werden. In diesem Bereich findet die Kontrolle aber vornehmlich durch mehrere Benutzer statt, das Prinzip bleibt aber das gleiche. Genauer: Das Vier-Augen-Prinzip ein Spezialfall des Reviews durch mehrere Personen ist.

4.2.2 Überblick

Basierend auf dem *Wissenscontainer (Seite 19)* und dem *Modell des Life-Cycle eines Falls* (*Seite 41f)* wird hier ein allgemeineres Konzept entwickelt, welches Änderungen an Elementen der Wissenscontainer während der Laufzeit eines FBS Systems koordiniert und deren Qualität sicher stellt.
Es beruht auf dem *Wissensbaustein* genannten Element eines Wissenscontainers. Jede Änderung an ihm wird in einer eigenen Revision unterschieden. Jeder Revision wird ein Status (Zustand) und eine Bewertung zugeordnet.. Durch Spezifikation der Übergänge zwischen den Zuständen der Revision des Wissensbausteins wird Collaborative Maintenance für das Life-Cycle Modell als ein Prozess beschrieben, wie Bild 8 zeigt. Es stellt die unterschiedlichen Zustände *eingereicht*, *akzeptiert* und *freigegeben* einer Revision eines Wissensbausteins und die dafür notwendigen Prozesse dar:

Prozessmodell des Life-Cycle eines Wissensbausteins

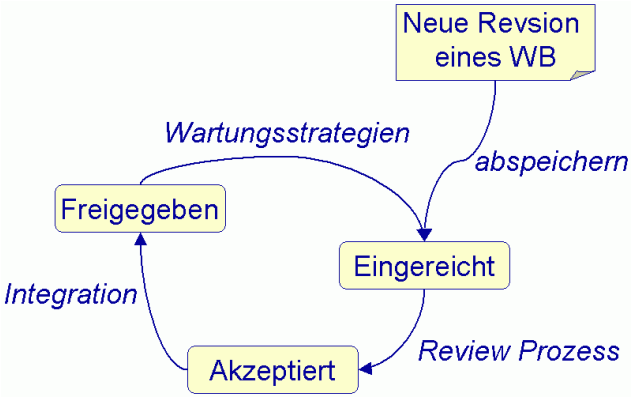


Bild 8: Prozessmodell des Life-Cycle eines Wissensbausteins mit den Zuständen

Beim Review Prozess lesen und bewerten Benutzer die Revision eines Wissensbausteins. Dadurch kann eine Revision vom Status *eingereicht* in den Status *akzeptiert* überführt werden.
Dies wird in *4.2.4 Der Review Prozess auf Seite 48* beschrieben. Die Integration der Revision, so dass sie zur Suche zur Verfügung steht, wird in *4.2.5 Integration auf Seite 53* erklärt. Auf die automatisch ausgelösten Wartungsstrategien wird in *4.2.6 Auslösung der Maintenance auf Seite 54* eingegangen.

4.2.3 Revision eines Wissensbausteins

Nach Richter enthält ein FBS System vier Wissenscontainer. Die einzelnen Elemente des Wissens innerhalb eines Containers werden von ihm nicht thematisiert aber hier benötigt.

> Definition 35: **Wissensbaustein (WB)**
> Der Wissensbaustein (WB) ist das kleinste modellierte Element (atomar) innerhalb eines Wissenscontainers (WC).
> Sei l ein Wissensbaustein und L die Menge aller möglichen Elemente in einem Wissenscontainer, so ist $l \in L$.

IEs sind Wissensbausteine des Wissenscontainers Vokabular, der Wert der Ähnlichkeit zwischen zwei IEs ist ein Element des Wissenscontainers Ähnlichkeitsmaß, während Fälle Elemente der Fallbasis sind.

In dieser Arbeit wird bei der Vergabe von Begriffen ein pragmatischer Ansatz gewählt, der tiefgehende, theoretische Diskussionen über den Wissensbegriff außen vor lässt.

In der englischsprachigen Literatur werden Wissensbausteine auch als «knowledge items» bezeichnet, z.B. in [Berghofer2003, S. 47].
Um auch kleinste Änderungen an Wissensbausteinen unterscheiden zu können, wird jeder WB mit einer eindeutigen ID und einer Revisions-Nummer versehen, welche bei jeder Änderung inkrementiert wird. Zusammen mit dem Autor kann dadurch jede Version jedes WB über den Deskriptor eines Wissensbausteins unterschieden werden:

Definition 36: **Deskriptor eines Wissensbausteins (WBD)**
Der Deskriptor eines Wissensbausteins (WBD) besteht aus einer ID, der Revision sowie dem Autor und identifiziert eine Revision eines Wissensbausteins eindeutig.

Die Einbeziehung des Autors geschah zur Vereinheitlichung, denn dann kann jeder Review zu einer WBR auch über einen WBD identifiziert werden.

Definition 37: **Revision eines Wissensbausteins (WBR)**
Die Revision eines Wissensbausteins *r* ist eine bestimmte Version eines Wissensbausteins (WB) und beschreibt ihren augenblicklichem Zustand. Sie besteht aus dem Wissensbaustein-Deskriptor (WBD), dessen Status und der aggregierten Bewertung.

Der Begriff Revision anstatt Version wurde schon im TestManager verwendet und daher beibehalten. Die aggregierte Bewertung wird in *4.2.4.1 Revisionsbewertung auf Seite 49* erläutert.

Der Status entscheidet welche Revision als „aktuell gültige“ Version des Wissensbausteins verwendet wird.

Definition 38: **Status (einer Revision)**
Der Status der Revision eines Wissensbausteins beschreibt den erreichten Reifegrad bezüglich des maintenance Prozesses.
Der Status hat folgenden diskreten Wertebereich:

freigegeben (release)

akzeptiert (accepted)

eingereicht (authoring)

Darüber ist folgende Präferenzordnung definiert:
freigegeben > *akzeptiert* > *eingereicht*.

Eingereicht (authoring) ist der initiale Status jeder Revision eines Wissensbausteins. Diesen erhält sie, wenn ihr Autor Änderungen an einem Wissensbaustein im System abspeichert. Diese Revision ist ungeprüft und steht jetzt für den Review genannten Begutachtungsprozess zur Verfügung, welcher in *4.2.4 Der Review Prozess auf Seite 48* erklärt wird. Als *akzeptiert* (accepted) gilt eine Revision, nachdem sie den Reviewprozess erfolgreich durchlaufen hat.
Den Status *freigegeben* (release) dokumentiert, dass diese Revision die jetzt offiziell gültige ist und im Retrieval verwendet wird. Dieser Zustandsübergang wird in *4.2.5 Integration auf Seite 53* beschrieben.
Auf den Übergang vom Status *freigegeben* in *eingereicht* wird in *4.2.6 Auslösung der Maintenance auf Seite 54* eingegangen.

Erweiterungsmöglichkeiten

Für mehrstufige Reviewprozesse ist eine Erweiterung des Wertebereichs denkbar. Dabei sind für jeden Status die Übergänge vom vorherigen Status zu definieren.

4.2.3.1 Sichten

Die Verwendung unterschiedlicher Sichten auf eine Fallbasis zur Unterstützung des Life-Cycle wurden schon in [MinorHanft2000] bzw. im Abschnitt 4.1.1 *Unterschiedliche Sichten zur Konsistenzwahrung im CRN trotz maintenance auf Seite 42* beschrieben.

Eine Sicht (View) legt fest, welche Revision jedes Wissensbausteins für einen bestimmten Anwendungsfall die aktuell „gültige“ ist und dafür verwendet wird.

Es existieren drei verschiedene Anwendungsfälle, in denen unterschiedliche Revisionen der Wissensbausteine verwendet werden. Für die Suche bzw. das Retrieval sind nur die Revisionen der WBs interessant, welche den Status *freigegeben* haben. Alle aktuellen Revisionen, denen als Ergebnis des Review Prozesses schon eine gute Qualität bescheinigt wurde, sind ebenfalls darzustellen. Die Revisionen hier haben also mindestens den Status *akzeptiert*. Soll ein WB geändert werden, ist als Grundlage die allerneueste Version zu verwenden, unabhängig davon welchen Status diese Revision hat. In der gleichen Sicht sind auch neu eingereichte Revisionen zu finden, ebenso wie jene die noch nicht positiv begutachtet wurden.

Die Zuordnung eines Status und die Sichtweise darauf ist auch vergleichbar mit Dateien im CVS System, wo nur jene Datei in die ausgelieferte Software übernommen werden, bei deren das Flag „stable“ gesetzt ist.

drei Sichten

Zusammenfassend gibt es drei Sichten, für die jeweils die Menge der sichtbaren WBRs einzuschränken ist:

- „offizielle“ Sicht für das Retrieval (Mindeststatus *freigegeben*)
- Sicht auf Ergebnis des Review-Prozesses (Mindeststatus *akzeptiert*)
- Sicht zum Editieren (implizit Mindeststatus *eingereicht*)

In jeder Sicht existiert pro WB immer nur eine „gültige“ Revision. Hat keine Revision eines bestimmten WBs den geforderten Mindeststatus, so taucht dieser WB in der Sicht nicht auf. Gelöschte WBs sind ebenfalls in diesen Sichten nicht verfügbar.

Definition Sicht (zu einem Status)

Definition 39: **Sicht (zu gegebenem Mindeststatus)**
Die Sicht auf einen Wissenscontainer schränkt die Menge aller WBRs dieses Wissenscontainers ein. Sie enthält von jedem WB nur die Revision mit der höchsten Revisionsnummer, deren Status größer oder gleich zum gegebenem Mindeststatus ist.

In eine Sicht werden also auch WBRs mit höheren Status als dem gegebenem eingeblendet, falls keine höhere Revision mit dem Mindeststatus existiert. Ein Wissensbaustein z. B. mit Status *freigegeben* ist auch in der Sicht *für das Ergebnis der Review Prozesses* enthalten, wenn kein WBR mit gleicher ID und höherer Revision den Status akzeptiert besitzt.

Beispiele für das Auftauchen von Revisionen in den Sichten

Als Beispiel seien hier verschiedene Revisionen einiger WBs aufgeführt und markiert in welcher Sicht sie auftauchen. Dabei wird eingereicht mit eing., akzeptiert mit akzp. und freigegeben mit frei. abgekürzt:

ID des WB	*11*				*12*			*13*	*14*	
Revision	*1*	*2*	*3*	*4*	*1*	*2*	*3*	*1*	*1*	*2*
Status	*frei.*	*akzp.*	*eing.*	*eing.*	*eing.*	*frei.*	*akzp.*	*eing.*	*frei.*	*eing.*
Offizielle Sicht für das Retrieval (Status ≥ freigegeben)	x					x			x	
Sicht auf Ergebnis der Reviews (Status ≥ akzeptiert)		x					x		x	
Sicht zum Editieren (Status ≥ eingereicht)				x			x	x		x

Vom WB 11 taucht in der Retrieval-Sicht die Revision 1 auf, in der Ergebnis-Sicht die Revision 2. Revision 3 erscheint nirgendwo, da mit gleichem Status schon eine höhere Revision 4 existiert. Gleiches gilt für Revision 1 des WB 12, diese Revision wurde aufgrund schlechter Reviews verbessert, die neue Revision 2 erhielt dann den Status freigegeben und ist in der Sicht für das Retrieval. Revision 3 wird in der Sicht auf das Review-Ergebnis als auch in der zum Bearbeiten verwendet. WB 13 ist weder in der Such-Sicht noch in der Ergebnis-Sicht enthalten, da keine Revision den entsprechenden Mindeststatus hat. Für den WB 14 ist die erste Revision in der obersten Sicht und wird auch in die Sicht auf das Ergebnis der Reviews „eingeblendet", da keine höhere Revision den Mindeststatus akzeptiert besitzt. Revision 2 wird nur zum Bearbeiten verwendet.

Konsistenzwahrung mit dem CRN

4.2.3.2 Konsistenzwahrung der Sicht für das Retrieval mit dem CRN

Wenn die für die Suche benutzte Speicher- und Indizierungstechnik nicht zu jedem Zeitpunkt an Änderungen der WBRs angepasst werden kann, müssen Änderungen an der Sicht für das Retrieval evtl. verzögert werden, um konsistent mit der Struktur für die Suche zu bleiben. Das hier verwendete CRN ändert sich während der Laufzeit des FBS-Systems (vom Start bis zum Herunterfahren des Systems) nicht. Änderungen sind also nur vor dem Aufbau des CRNs möglich.

Da die Sicht für das Retrieval die WBRs für das CRN bereithält, welches sich während der Laufzeit des FBS-Systems (vom Start bis zum Herunterfahren des Systems) nicht ändert, muss diese Sicht während der Laufzeit fixiert werden, um mit dem CRN konsistent zu bleiben. Dafür gibt es mehrere Möglichkeiten:

- Zustandsübergänge in und vom Status *freigegeben* verzögern,
- Momentaufnahme der Sicht beim Start des Systems erstellen.

Während der Laufzeit werden Zustandsübergänge von WBRs in und vom Status *freigegeben* unterbunden. Dadurch bleibt die Sicht für das Retrieval während der Laufzeit des Systems konstant. Dann könnten nur innerhalb der Integration und der Ausführung der Wartungsstrategien die Zustände der WBRs in und von *freigegeben* wechseln, so dass sich dann auch die Sicht auf das Retrieval ändert.

Sicht für das Retrieval als Snapshot fixieren

Bei der zweiten Möglichkeit wird bei jedem Start ein Snapshot der augenblicklichen die Sicht für das Retrieval erstellt und während der gesamten Laufzeit beibehalten. Durch schlechte Reviews ausgelöste Zustandsübergänge aus dem Zustand *freigegeben* (nach eingereicht) sind nur in den beiden anderen Sichten nachvollziehbar.
Im *mtBook* wird die zweite Möglichkeit des Snapshots verwendet. Diese Entscheidung ist implementierungsbedingt, zu Details siehe *5.3.3.4 Konsistenzwahrung der Sicht für das Retrieval auf Seite 64*.

4.2.4 Der Review Prozess

Eine neue Revision eines Wissensbausteins ist nicht in jedem Fall inhaltlich fehlerfrei und verständlich. Der Review Prozess verhindert die Übernahme „schlechten“ Wissens und verbessert die Kommunikation zwischen Autoren und Benutzern während der Erstellung und Überarbeitung von Wissen:

Review Prozess ist organisatorischer Rahmen für Bewertung

> Definition 40: **Review Prozess**
> Der Review Prozess ist der organisatorische Rahmen für die Bewertung von Wissensbausteinen (WB). Der Review Prozess stellt die Qualität von Änderungen an WBs sicher, da nur bei erfolgreichem Durchlaufen des Prozesses der Status einer WBR von *eingereicht* auf *akzeptiert* angehoben wird.

Er stellt den wichtigsten Schritt in diesem Life-Cycle Modell dar. Der Review Prozess beginnt mit dem Speichern einer neuen Wissensbausteins-Revision. Sie wird in eine Liste der «Ausstehenden Reviews» eingetragen.

Dort verbleibt sie solange, bis ihr Status aufgrund der Reviews auf *akzeptiert* angehoben wird.

> Definition 41: **Review**
> Der Review bezeichnet die Begutachtung einer Wissensbausteins-Revision durch eine andere Person als den Autor. Als Ergebnis enthält der Review eine Revisionsbewertung und einen optionalen Text-Kommentar des Begutachters.

Begutachter

Diese Person wird in dieser Arbeit Begutachter genannt. Der Begriff Gutachter wurde hier vermieden, da dieser eine Person mit besonderer Expertise kennzeichnet, während die Reviews hier durch gewöhnliche Benutzer erfolgen. In anderen Systemen kann der Review auch aus einem geändertem Fall bestehen [FerrarioSmyth2000].

Review ist kein Feedback!

Da für den Wissenscontainer Fallbasis der Review gleich im Anschluss an die Anzeige eines Falltextes möglich sein soll, darf dies nicht als Feedback des Retrievals missverstanden werden! Der Review dient nicht dazu, zu evaluieren ob der Fall für die Suchanfrage des Benutzers hilfreich war, da sonst eine gute Fallbeschreibung (z. B. aufgrund unzureichender Ähnlichkeitsmodellierung) als schlecht bewertet werden könnte. Auch können dabei die Bewertungen der Benutzer bei anderen Anfragen höchst unterschiedlich und zueinander widersprüchlich sein.

Auch zeigen die bisherigen Erfahrungen, dass die Feedbackquote, bei jedem Retrieval von den Benutzern gefordert, sehr gering ist (bei [Busch2001] 2%), so dass es dem Autor dieser Arbeit sinnvoller erschien, nur die Qualität der Wissensbausteins-Revision allgemein (und ausgeführt durch besonders motivierte Begutachter) zu bewerten.

4.2.4.1 Revisionsbewertung

Der Review kann wie in *3.6.1.1 Einordnung von Ulysses und Abgrenzung zum TestManager auf Seite 37* ausgeführt, nicht analog zu Ulysses in Textform erfolgen. Stattdessen wird eine Benotung eingeführt.

> Definition 42: **Revisionsbewertung**
> Die Revisionsbewertung n ist als Ergebnis eines Reviews zu einer WBR die Benotung mit 0 bis 5 Punkten, welche die vom Begutachter eingeschätzte Qualität und inhaltliche Korrektheit ausdrückt.
> $n \subseteq N_r$ sei die Menge aller Revisionsbewertungen zu einer Revision r eines Wissensbausteins. Es gilt $n \in \aleph$, wobei $0 \leq n \leq 5$.

Bei dieser Revisionsbewertung können folgende Punkte vergeben werden, hier in der ausführlicheren Fassung für den WB Fall:

5 Punkte exzellente Lösung, erfolgreich ausprobiert, sehr gut erklärt

4 Punkte gute Lösung, funktioniert, gut erklärt

3 Punkte Lösung ausreichend, aber umständlich, Erklärung knapp

2 Punkte nur teilweise Lösung/ enthält Fehler

1 Punkt unzureichend/ funktioniert nicht

0 Punkte Fall ist überflüssig, kann gelöscht werden

Die Wertung mit 0 Punkten ist speziell für veraltete Wissensbausteine vorgesehen, welche aus einem höheren Status noch einmal zwecks Durchsicht in den Status *eingereicht* herabgesetzt wurden. Damit wird die Löschung veranlasst. Der Life-Cycle dieses Wissensbausteins endet.

Bei jedem Hinzufügen (bzw. Ändern) eines Reviews wird aus allen Revisionsbewertungen zu dieser WBR mit Hilfe der Akzeptanzfunktion die Aggregierte Bewertung der Revision des Wissensbausteins (erneut) berechnet.

> Definition 43: **Akzeptanzfunktion**
> Die Akzeptanzfunktion a fasst alle zu einer WBR gehörenden Reviews zu einem Wert zusammen. a: $N_r \rightarrow \Re$, mit $0 \leq a \leq 5$.

Die Akzeptanzfunktion ist für jedes FBS System individuell definierbar. Sie ist im einfachsten Fall der Durchschnitt aller entsprechenden Revisionsbewertungen. Aber andere Bedingungen wie z. B. „mindestens drei Reviews mit Mindestnote 2“ sind definierbar.

> Definition 44: **Aggregierte Bewertung**
> Die Aggregierte Bewertung s einer Revision eines WB drückt die zusammengefasste Qualitätseinschätzung der Begutachter aus. Sie wird über die Akzeptanzfunktion $a()$ berechnet. Es gilt $s \in \Re$ wobei $0 \leq n \leq 5$.

Die Aggregierte Bewertung (auch Aggregierte Revisionsbewertung) wird mit einer definierten Akzeptanzschwelle verglichen.

> Definition 45: **Akzeptanzschwelle** (acceptance level)
> Die Akzeptanzschwelle ist die im System festgelegte Höhe der aggregierten Revisionsbewertung, ab der eine Wissensbausteins-Revision akzeptiert wird.

Der Standardwert der Akzeptanzschwelle beträgt 3 Punkte (ausreichend).

Das Akzeptieren der Revision bedeutet das (vorläufige) erfolgreiche Ende des Review Prozesses. Die WBR wechselt vom Status *eingereicht* in den Status *akzeptiert* und wird aus der Liste der «Ausstehenden Reviews» entfernt.

Review in jeder Sicht möglich

Das Modell schreibt nicht vor, dass nur WBRs mit dem Zustand *eingereicht* begutachtet werden dürfen. Benutzer können zu jeder WBR in jeder Sicht einen Review abgeben. Fällt dadurch deren Aggregierte Bewertung unter die Akzeptanzschwelle, wird diese WBR wieder in den Status *eingereicht* zurückgesetzt und damit der komplette Review Prozess erneut angestoßen. Dies kann man auch als manuelle oder vom Benutzer ausgelösten Maintenance bezeichnen, im Gegensatz zur automatischen *4.2.6 Auslösung der Maintenance auf Seite 54*.

Zwischen den Zuständen *eingereicht* und *akzeptiert* können WBRs aufgrund von Reviews ständig hin und her wechseln. Nur Wechsel von und in den Zustand *freigegeben* sind aufgrund der Konsistenzwahrung möglicherweise verzögert.

Zusammenfassend dargestellt ist der Review Prozess in einen Ablauf aus folgenden Schritten eingebettet:

1. Neue Revision eines Wissensbausteins wird eingegeben
2. Auswahl der Begutachter
3. Einladung zum Review (Review Posting)
4. Revision bewerten
5. Berechnung der aggregierten Bewertung der Revision
6. Integration/Verwerfen der Revision (Änderung des Status nach *akzeptiert*)

Bild 9 veranschaulicht grafisch den gesamten Prozess in einem Aktivitätsdiagramm:

Aktivitätsdiagramm des Review Prozess

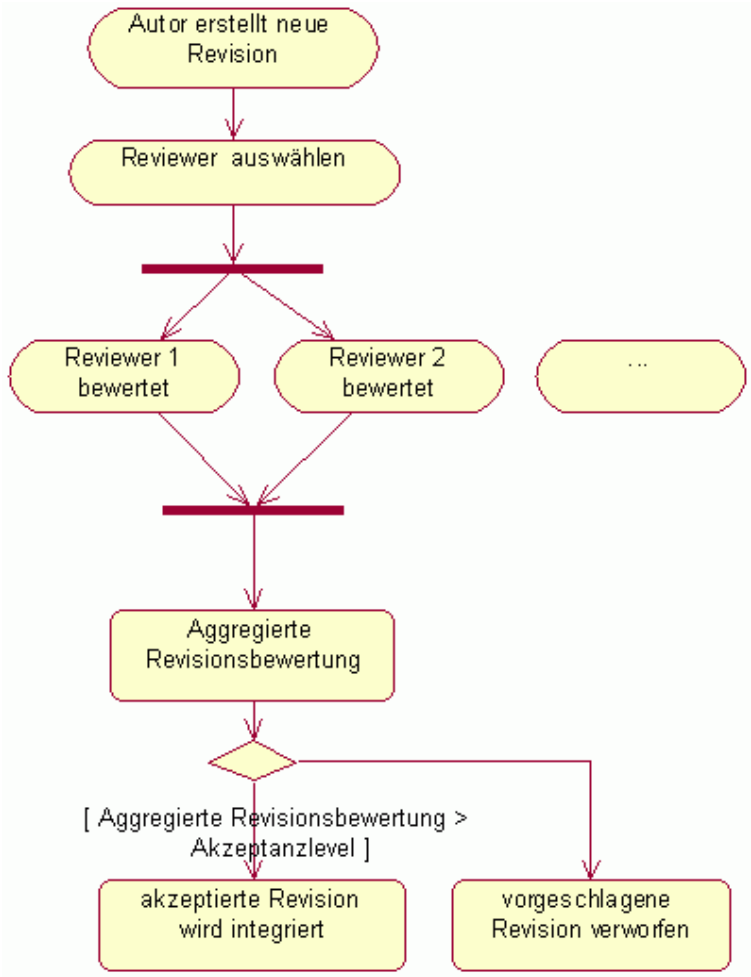


Bild 9: UML-Aktivitätsdiagramm des Review Prozesses

4.2.4.2 Auswahl der Begutachter

Der Auswahl der Begutachter für eine Wissensbausteins-Revision kommt umso größere Bedeutung zu, je mehr Änderungen an den Wissensbausteinen vorgenommen werden und je mehr Wissensbausteine dieser Wissenscontainer enthält. Die unterschiedliche Expertise der Begutachter fordert ebenso eine angepasste Auswahl, damit sie nicht ständig mit WBRs konfrontiert werden, zu denen sie kein Urteil abgeben können.
Der Autor ist als Begutachter ausgeschlossen. Im Folgenden werden vier unterschiedliche Auswahlmechanismen vorgestellt.

Manuelle Selektion

Die Auswahl der Begutachter wird von Hand durch den Begutachter erledigt. Neue bzw. noch nicht akzeptierte Revisionen werden in einer Liste «Ausstehende Reviews» präsentiert, aus der sich jeder Begutachter diejenigen WBRs heraussucht, zu denen er sich kompetent erachtet.

Autoren ähnlicher Revisionen von Wissensbausteinen

Diese Art der Selektion funktioniert nur bei den Wissenscontainern Fallbasis und Indexvokabular, da für die Selektion der Wissenscontainer Ähnlichkeitsmaß benutzt wird.
Als Begutachter werden diejenigen ausgewählt, welche ähnliche (WBRs) zu der zu begutachtenden WBR eingegeben haben.

> Beim WC Fallbasis wird die neue Fallrevision beispielsweise als Retrieval Anfrage an den Retrieval Server geschickt. Von den so ermittelten Fällen werden z. B. die ersten fünf Autoren ausgewählt, welche nicht mit dem Autor dieser neuen Fallrevision identisch sind. Für das Indexvokabular wird das gleiche Verfahren nur mit neuen IEs als Anfrage vorgeschlagen.

Zugeordnete Domäneexperten

Diese Art der Selektion eignet sich für alle Wissenscontainer, deren Elemente in Teilbereiche (Domänen) oder Kategorien unterteilt werden. Bei jedem gespeichertem Fall z. B. werden die Kategorien angegeben, aufgrund derer zugeordnete Domäneexperten als Begutachter bestimmt werden.

Für Autoren bzw. Domäneexperten, welche die Organisation verlassen, wird eine Ersatzregel definiert: Alle Fälle eines nicht mehr verfügbaren Autors werden an anderen Begutachter dirigiert

Weitere Möglichkeiten

Zuletzt könnte auch ein „Review Administrator“ die neuen WBRs manuell nach Ansicht bestimmten Begutachtern zuteilen. Damit funktioniert die Auswahl der Begutachter aber nicht mehr automatisch und ohne einen Administrator.

> Dieses Verfahren wird z. B. häufig für die Zuordnung der Gutachter zur Annahme von Konferenzartikeln eingesetzt.

Sollten die oben genannten Möglichkeiten nicht genug Begutachter erbringen, könnten noch einige Benutzer zufällig als Begutachter ausgewählt werden.

4.2.4.3 Einladung zum Review (Review Posting)

Nachdem die potentiellen Begutachter bestimmt sind, müssen sie nun informiert werden. Eine ziemlich einfach Lösung wäre, die Benutzer zu bitten die Webseite der «Ausstehenden Reviews» regelmäßig zu besuchen. Allerdings hängt dies davon ab ob der Benutzer als potentieller Begutachter sich regelmäßig daran erinnert. Besser ist dagegen eine Unterstützung durch das System. In der zweiten Variante bekommen die ausgewählten Begutachter eine Email mit einem Link auf die neue Revision des Wissensbausteins und der Bitte um einen Review. Bei sehr vielen neuen Revisionen bekommt der Begutachter sehr viele Emails, was sicher eine zusätzliche Belastung darstellt.
Eine adäquatere, dritte Lösung ist ein Internetportal, in dem der Begutachter sich wie jeder Benutzer einloggt. Dort erhält er eine personalisierte Webseite nur mit seinen «Ausstehenden Reviews».

So wurde dies beispielsweise in ULYSSES von Ferrario & Smyth [FerrarioSmyth2000] realisiert.

Die Implementierung eines Portals stellt allerdings einen erhöhten Aufwand dar, da in dem als Code-Basis verwendeten System ExpBook II weder eine Benutzerverwaltung noch eine Portalsoftware verwendet wird und liegt außerhalb des Fokus dieser Arbeit.

4.2.4.4 Erweiterungsmöglichkeiten

Vertrauensbonus

Statt der Punkteskala von 0 bis 5 kann der Begutachter beim Review eine pauschale Revisionsbewertung ohne Ansicht der WBR als Vertrauensbonus gegenüber dem Autor geben. Dafür wird anfangs ein Wert von 3 Punkten vorgeschlagen. Der Bonuswert könnte nach der durchschnittlichen Bewertung aller Reviews zu diesem Autor angepasst werden. Die pauschale Bewertung wäre aber nur für neu eingereichte WBRs adäquat, da veraltete Fälle nur durch Ansicht überprüft werden können. Das Bonussystem müsste noch einmal extra evaluiert werden.

Hierarchie bei den Autoren und Begutachtern

Hierbei wäre es für die Akzeptanz einer WBR nicht mehr gleichgültig welche Begutachter die Reviews durchführen. Dazu müsste allen Benutzern (potentiellen Autoren und Begutachtern) eine Hierarchiestufe zugeordnet werden. Die wäre z. B. denkbar wenn eine Benutzergruppe in Neulinge und Erfahrene Nutzer unterteilt werden kann. Die Akzeptanzfunktion könnte dann leicht so verändert werden, dass eine WBR nur akzeptiert wird, wenn mindestens ein Review (über der Akzeptanzschwelle) von einem Begutachter stammt, der in der Hierarchie höher als der Autor eingeordnet ist.

4.2.5 Integration

Akzeptierte Revisonen erhalten Status freigegeben

Durch die Integration werden bis zu diesem Zeitpunkt akzeptierte Revisionen von Wissensbausteinen für die Suche (Retrieval) zur Verfügung gestellt. Dazu werden alle WBRs mit dem Status *akzeptiert* in den Status *freigegeben* heraufgestuft. Sie befinden sich nun in der Sicht für das Retrieval und verdränden dort evtl. vorhandene ältere Revisionen des glei-

chen WBs. Dann kann die zur Suche benutzte Speicherstruktur aktualisiert werden, was bei einem FBS System durch den Neuaufbau des Fallspeichers (CRN) geschieht.
Wie schon in *4.1.1 Unterschiedliche Sichten zur Konsistenzwahrung im CRN trotz maintenance auf Seite 42* erläutert, erfolgt die Integration von neuen Revisionen nicht sofort. Durch eine regelmäßige Integration (z. B. täglich) verbleiben WBRs aber nicht länger als eine bestimmte Zeitspanne (1 Tag) im Status *akzeptiert*.

4.2.6 Auslösung der Maintenance

Da das Wissen in den Wissensbausteinen veraltet, werden diese regelmäßig einer maintenance unterzogen. Dazu wird die Revision des Wissensbaustein (aus der Sicht für das Retrieval) wieder in den Status *eingereicht* zurückgesetzt.

Dazu soll entschieden werden, wann die maintenance der Revision eines Wissensbausteins ausgelöst wird. Im folgenden werden drei Möglichkeiten der Auslösung vorgestellt:

- festgesetztes Wartungsintervall für Wissenscontainer,
- themenbasierte maintenance
- manuell ausgelöst

Wartungsintervall

Bei der ersten Möglichkeit ist die Zeitsteuerung der Auslösung zeitbasiert, während die letzten beiden Ansätze ad hoc sind (siehe *3.2.2 Auslösung der maintenance auf Seite 25*). Bei einem global festgesetzten Wartungsintervall wird die maintenance für alle WBRs ausgelöst, welche in der Sicht für das Retrieval erscheinen und älter als eine festgelegte Zeitspanne sind. Bei der Erstellung der Revision wäre auch eine vom Autor individuell festgelegte Lebensdauer denkbar.

> Ein Beispiel für ein global festgelegtes Wartungsintervall ist die Wartung eines Autos, alle zwei Jahre muss es zur Hauptuntersuchung. Erst nachdem diese Prüfung erfolgreich bestanden wurde, erhält das KFZ die sogenannte TÜV-Plakette und darf wieder im öffentlichen Verkehr gefahren werden.

Für den Review einer neu eingereichten WBR wurde der Autor als Begutachter ausgeschlossen. Bei erneuten Reviews aufgrund der zeitbasierten maintenance kann es sinnvoll sein, dass der Autor auch einen Review abgeben darf und damit dokumentiert, dass diese WBR noch in Ordnung ist.

themenbasiserte maintenance

Bei der themenbasiserten maintenance wird festgelegt, dass alle WBRs (aus der Sicht für das Retrieval) zu einem bestimmten Teilgebiet überarbeitet werden sollen. Für die Wissenscontainer Fallbasis und Vokabular kann das Retrieval für die Auswahl der betroffenen Wissensbausteine benutzt werden. Die Auswahl kann auch durch einen Benutzer erfolgen, allerdings empfiehlt es sich, damit einen Domäne-Experte zu betreuen, so dass dieser Ansatz hier nicht weiter verfolgt wird.

Die manuelle Auslösung erfolgt, wenn ein Benutzer zu einer WBR in der Retrieval Sicht oder der auf das Review Ergebnis einen Review abgibt, der die Revision unter der Akzeptanzschwelle bringt. Diese Revision wird dann wieder in den Status *eingereicht* zurückgesetzt.

Im *mtBook* wird die automatische, vom System aufgrund vorgegebener Wartungsstrategien ausgelöster Maintenance verfolgt.

Bei den erläuterten Möglichkeiten wird abgesehen von der manuellen Auslösung die bestehende WBR belassen und als neue Revision mit dem Status *eingereicht* gespeichert. Die Alternative, nur den Status der bisherigen Revision zu ändern, scheidet aus. Alle alten Reviews zu dieser Revision müssten sonst gelöscht werden, um nicht eine sofortige Akzeptanz hervorzurufen.

Andere Möglichkeiten

Zur Auslösung der maintenance gibt es noch weitere Möglichkeiten. Performance Messungen sind in ihrem Wartungsumfang eher weitläufig (siehe *3.2.4 Ausführung auf Seite 26*), da sie einen Wissenscontainer als ganzes betrachten. Dazu zählen z. B. Messungen der Retrievalzeiten, Registrieren unzureichender Retrievalergebnisse (kein Fall geliefert) oder die Größe der Fallbasis. Messungen der Benutzungshäufigkeit von Fällen könnten über die Analyse der Logfiles des Webservers erfolgen. Beide Ansätze werden aber im Rahmen dieser Arbeit nicht weiter verfolgt, da dies schon andere Autoren untersucht haben und der Implementierungsaufwand auf Basis des bestehenden Systems zu hoch eingeschätzt wird.

4.3 Zusammenfassung des Modells

Zusammenfassend lässt sich ein Maintenance Modell für den Life-Cycle eines Wissensbausteins definieren:

Maintenance Modell für den Life-Cycle

Definition 46: **Maintenance Modell für den Life-Cycle eines Wissensbausteins (MML)**
Das Maintenance Modell für den Life-Cycle eines Wissensbausteins besteht aus

- der Revision eines Wissensbausteins mit
  - ID, Revisionsnummer und Autor,
  - Status und
  - Bewertung

sowie

- den Prozessen, welche die Übergänge zwischen den Zuständen eingereicht, akzeptiert und freigegeben definieren:
  - Review Prozess
  - Integration
  - Maintenance Strategien.

Diese Definition ist hinsichtlich der maintenance bewusst allgemeiner gehalten.

Collaborative Maintenance Modell für den Life-Cycle

Definition 47: **Collaborative Maintenance Modell für den Life-Cycle eines Wissensbausteins (CMML)**
Das Collaborative Maintenance Modell für den Life-Cycle eines Wissensbausteins ist ein MML, bei dem

- der Review-Prozess «peer-to-peer» von den Benutzern und
- die Maintenance Strategien ohne die Mitwirkung eines Administrators durchgeführt werden können.

4.4 Aufwand vs. Nutzen – Wann lohnt sich der Aufwand?

Das Life-Cycle Modell inklusive des Review-Prozesses und der Verwaltung der unterschiedlichen Revisionen stellt einen beträchtlichen Mehraufwand dar. Er sollte daher nur für die Wissenscontainer betrieben werden, welche sich häufig ändern. Dabei ist der Aufwand gegen den Nutzen abzuwägen und rechtfertigt sich unter folgenden Umständen, wenn:

Qualitätskontrolle

- die (geforderte) Qualität der eingereichten WBRs nicht von vornherein gegeben ist,
- unterschiedliches Know-How der Benutzer vorhanden ist,
- häufige Fehlerquote / Datenverschmutzung zu kontrollieren ist,
- die WBRs (nicht erwünschte) subjektive Teile enthalten,
- beim Zusammenführen von Ontologien bzw. Wissenscontainern aus verschiedenen Quellen die Herkunft auch später berücksichtigt werden soll
- Rechtssicherheit gefordert ist, z. B. bzgl. Patentrechts- oder Copyright-Verletzungen, oder
- häufige Änderungen erfolgen

Beim Zusammenführen von unterschiedlichen Quellen kann die Note, à priori festgelegt, eine Bewertung der Wissensbaustein darstellen – ein Glossar verdient z. B. eine gute Bewertung, weil es innerhalb eines Softwareerstellungs-Vertrages auch von den technischen Experten und Geschäftsführern gebilligt wurde. Ebenso haben Bücher meist eine höhere fachliche Qualität als selbst ins Netz gestellte Webseiten, weil erstere von einem Lektorat geprüft werden.
Christina Bell verlinkt in ihrer Diplomarbeit [Bell2004] über ein fallbasiertes Weblexikon in der Domäne der RoboCup-Simulationsliga sehr unterchiedliche Quellen, u.a. das SoccerServer Manual und Emails.

Ebenso empfiehlt sich die Einrichtung eines solchen revisionsbasierten Systems wenn:

- Revisionssicherheit gefordert ist (welche Version war zu einem bestimmten Zeitpunkt aktuell)
- die Historie betrachtet werden soll, um z. B. Veränderungen zwischen unterschiedlichen Versionen aufzuzeigen

Eine Idee den Aufwand eines Life-Cycle Modells bei selten geänderten Wissensbausteinen zu reduzieren ist, bei einer Änderung an einem WB diesem temporär mit einer ID und Revisionsnummer auszustatten, um den Review Prozess bis zum Status *akzeptiert* zu ermöglichen. Mit dem Wechsel in den Status *freigegeben* wird das gesonderte Element in den Wissenscontainer integriert und ist wieder ein namenloses Element.

Im Weiteren wird Collaborative Maintenance für das Life-Cycle Modell nur für Fälle betrachtet und implementiert, da sich diese am häufigsten ändern. Eine WBR heißt dann einfach Fallrevision.

4.5 Vergleich mit bestehenden Ansätzen

Abschließend wird das entwickelte *mtBook* neben bestehende Ansätze eingeordnet. Teilweise wurden die Ansätze schon in Kapitel *3 Literaturrecherche und Analyse auf Seite 21* beschrieben, hier aber zum Vergleich noch einmal herangezogen, da die neuen Konzepte im *mtBook* umgesetzt sind. Zu Beginn werden Parallelen zum Software Engineering gezogen. Im Anschluss werden die Neuerungen gegenüber dem als Basis der Weiterentwicklung verwendeten ExpBook I + II aufgezeigt. Eine Einordnung der maintenance Strategien in die Kategorisierung nach Leake & Wilson folgt. Danach wird *mtBook* dem schon vorgestellten EMSIG Framework gegenübergestellt um anschließend die Gemeinsamkeiten und Unterschiede zum «Six steps» Modell bzw. zur SIAM Methode herauszuarbeiten. Zuletzt wird der hier verfolgte Ansatz des Collaborative Maintenance mit WIKIPEDIA, einer Software außerhalb des FBS, verglichen und diskutiert.

4.5.1 Software Engineering

Parallelen zum Änderungs-management

Im Konfigurations- und Änderungsmanagement werden Dokumente und Programmteile auch bei jeder Änderung mit einer neuen Version versehen. Sie haben dann den Status «in Bearbeitung» Nachdem die Qualitätssicherung diese Version positiv begutachtet hat, erhält sie den Status «akzeptiert» [Balzert1998, S. 241ff]. Dieses Dokument wird dann in eine neue Konfiguration eingebettet, was der Sicht für das Retrieval in *mtBook* entspricht. Änderungsmanagement wird auch im Deutschen mit der englischen Form Change Management bezeichnet.

4.5.2 ExpBook I + II

Beim ExpBook I musste ein Administrator die redaktionelle Kontrolle über neue Fälle und deren Integration übernehmen. Beim TestManager oder ExpBook II werden dagegen alle Falländerungen ohne Kontrolle sofort übernommen. Mit dem *mtBook* konnte ein System entwickelt

werden, welches 1. bei der Wartung ohne eine Administratorin auskommt und 2. gleichzeitig die Qualität von Änderungen als auch bestehenden Fällen durch gegenseitige Kontrolle sicherstellt.
Während im Retrievalergebnis des ExpBook II immer die neueste Revision des Falltextes angezeigt wird, präsentiert *mtBook* wirklich die Revision aus der Sicht für das Retrieval, welche im CRN indiziert ist. Der Vorteil vom *mtBook* ist dabei, dass hier nur durch einen Review überprüfte Fallrevisionen präsentiert werden. Im *mtBook* wurde Aktualität gegen Qualität zugunsten letzterer abgewogen. Der Nachteil des Verzichts auf Aktualität wird vermindert, indem bei Anzeige des Falltextes über einen Link immer die auch neueste Revision zugreifbar ist.

Review-Kommentar ist zurückhaltender

Im ExpBook II ein Kommentar zu einem Fall abgegeben, der direkt in den Falltext eingefügt wird. *mtBook* erlaubt mit den Review-Kommentaren, auch Fragen an den Autor eines Wissensbausteins bzw. Falls zu stellen. Diese Art des Kommentars ist auch zurückhaltender als gleich den Fall zu ändern. Dies ist angebracht wenn sich der Fragende nicht kompetent genug fühlt, den Fall selbst zu ändern oder aus Respekt vor dem Autor es ihm selbst überlassen möchte. Zusammengefasst wird also durch die Reviews die Kommunikation bei der Änderung des Wissens in einem FBS-System verbessert.

4.5.3 Einordnung in Kategorisierungsschema nach Leake & Wilson

Review für einjährige Fallrevisionen

Die maintenance Strategie, einjährige Fallrevisionen erneut begutachten zu lassen, hat eine synchrone Datensammlung, welche *periodisch* und *off-line* ist. Die einzelnen Dimensionen sind in *3.2 Kategorisierung nach Leake und Wilson auf Seite 23* erklärt. Die Zeitsteuerung ist offensichtlich zeitbasiert konditional, die Integration der Auslösung *off-line*, da dies beim Initialisieren des Systems geschieht. Ziel dieser Operation ist der *Domäneninhalt* und in der Dimension der Änderungsebene die *Wissensebene*. Die Zeitsteuerung der Ausführung ist *ad hoc,* da sie von der Eingabe der Reviews durch die Benutzer abhängt. Die Integration in andere Systemprozesse ist *off-line*, da dies parallel zum Retrieval erfolgt.
Der Wartungsumfang wird als *begrenzt* eingeschätzt, da immer nur ein Fall oder eine kleine Menge an Fällen zur Durchführung des Reviews aussortiert werden.

Review für geänderte Fallrevisionen

Von Autoren geänderte Fallrevisionen einem Review zu unterziehen, ist eine in der Datensammlung *nicht-introspektive* Strategie. Da die Änderung vom Benutzer abhängt, ist die Zeitsteuerung der Datensammlung und Auslösung *ad hoc*. Der Review Prozess ist der gleiche wie bei der ersten Strategie, für die weiteren Charakterisierungen gilt daher das eben gesagte.

4.5.4 Systematic maintenance am IESE

Das gleiche Ziel wie im *mtBook*, maintenance nicht ad hoc sondern automatisiert auszulösen, verfolgt der «systematic maintenance» Ansatz des EMSIG Frameworks der IESE, vorgestellt von Nick, Althoff und Tautz in [NickEtal2001], siehe auch *3.3* Systematic Maintenance of Corporate Experience Repositories *auf Seite 30*. Ebenso wie EMSIG verwendet

mtBook das FBS auch zur Unterstützung der Maintenance (für die Auswahl der Begutachter und themenbasierte Maintenance).

Mächtiges Konzeptuelles Modell

Das dort verwendete Konzeptuelles Modell läßt sich sehr gut mit dem hier vorgestellten Prozess des Collaborative Maintenance für das Life-Cycle Modell vergleichen, abgesehen davon dass EMSIG Collaborative Maintenance nicht behandelt. Siehe dazu auch die Erläuterungen und *Bild 6 auf Seite 32*. Angefangen von der Messung des Alters eines Falls (entspricht Measurement Data) wird die entsprechende Wartungsstrategie „Maintenance für Fälle älter als 2 Jahre“ für jeden betroffenen Fall ausgelöst, was mit der Aktivierung einer maintenance guideline über einen Trigger übereinstimmt. Als erste einfache maintenance-Aktion (Maintenance Primitive) wird der Status des Falls auf *eingereicht* zurückgesetzt. Darauf hin landet er in der Liste «Ausstehende Reviews», ein Eintrag hier entspricht einem Change Request. Darauf hin werden Reviews abgegeben. Obwohl ein Review bisher nicht als Erfahrung verstanden wurde, so kann er doch als eine Art Experience Case über die Wartung des Systems aufgefasst werden. Nach erstellten Reviews wird die aggregierte Revisionsbewertung neu berechnet, was ebenso wie der evtl. Anhebung in den Status *akzeptiert* je einer Maintenance Primitive gleichkommt.

Knowledge Life-Cycle

Außerdem gibt es noch eine weitere Parallele: Ein ähnlicher Ansatz zum Life-Cycle Modell eines Wissensbausteins ist das *knowledge Life-Cycle modell* in [NickAlthoff2001]. Das für *mtBook* verwendete Life-Cycle Modell entspricht der dort vorgestellten Kategorie *Life-Cycle von Fällen eines Typs* (siehe *3.3.1* Knowledge Life-Cycle Modell von Nick & Althoff *auf Seite 32*), da die Weiterentwicklung eines Wissensbausteins bzw. Falls im *mtBook* nicht explizit als verbessertes Wissen modelliert wird.

EMSIG unterscheidet sich vom *mtBook* dadurch, dass es die Definition und Ausführung von Maintenance umfassend durch ein Framework unterstützt. *mtBook* konzentriert sich auf Collaborative Maintenance und implementiert nur eine begrenzte Anzahl der Wartungsstrategien.

4.5.5 «Six steps» und SIAM

Qualitäts-informationen

Ein Prozessmodell zur maintenance wird auch schon von Th. Roth-Berghofer in seinem Modell der «six steps» im FBS eingeführt. Dabei hat es ebenso wie die breiter angelegte SIAM Methode mit *mtBook* gemeinsam, dass der Review Prozess Bestandteil des Modells ist und dabei in einzelne Aufgaben zergliedert wird. Die bei Berghofer zur Qualitätsinformationen eines Falls zählende Zeit der letzten Änderung wird auch im *mtBook* zur Auslösung der maintenance verwendet.

Life-Cycle

Obwohl auch SIAM mit dem Begriff des Life-Cycle gekennzeichnet wird, ist dort keine Modellierung eines Life-Cycle besonders von Wissensbausteinen o. ä. erläutert, sondern der Begriff wird nur zusammenfassend benutzt. SIAM modelliert kein Collaborative Maintenance.

Dafür ist das Modell allgemeiner gehalten und bei der Messung der Daten zur Maintenance wie auch bei der Strukturierung in Ebenen, viele Schritte und Komponenten detaillierter. Somit ist SIAM auf ein größeres Spektrum von maintenance Aufgaben anwendbar.

4.5.6 Enzyklopädie WIKIPEDIA

großes Collaborative Maintenance System

Die freie Enzyklopädie WIKIPEDIA[4]. Ist ein gutes Beispiel für ein großes Collaborative Maintenance System . Auch hier kann jeder Benutzer neue Lexikoneinträge erstellen oder ändern, aber dort werden Änderungen an sofort veröffentlicht.

WIKIPEDIA arbeitet auch mit einem Versionskontrollsystem, man kann sich jede einzelne Version anzeigen und mit anderen vergleichen lassen.

Heterogene Qualität der Artikel

Aufgrund der Heterogenität Kompetenz der Autoren schwankt die Qualität und Ausführlichkeit der Einträge, mit dem die Organisatoren umgehen müssen. Es gibt umfangreiche Hinweise und Richtlinien für das Verfassen von Artikel. Dies reicht aber nicht aus. Eine «1+ Qualitätsoffensive» wurde gestartet, welche abwechselnd auf besondere Themenkomplexe fokussiert. Dies zeigt folgendes Beispiel: Besucht man den Eintrag Wissensmanagement[5], erscheint über dem Beitrag ein Hinweis „*Dieser Artikel befindet sich derzeit im Reviewprozess. Hilf mit, ihn zu verbessern!*“…

Hinzu kommen rechtliche Gründe, welche gegen die sofortige Veröffentlichung von Artikeln sprechen: Einige Benutzer meinen, etwas gutes für WIKIPEDIA zu tun, in dem sie aus vorhandenen Webseiten oder anderem durch Copyright geschütztem Material Beiträge hineinkopieren. Administratoren müssen diese Artikel ausfindig machen und wieder entfernen um juristischer Verfolgung wegen Copyrightverletzung zu entgehen [Kurzidim2004].

[4] http://de.wikipedia.org/wiki/Hauptseite

[5] http://de.wikipedia.org/wiki/Wissensmanagement, besucht am 04.11.2004

5 Implementierung

Das System ist mit Hilfe einer Client-Server-Architektur realisiert. Es besteht aus mehreren CGI-Clients welche die Weboberfläche erzeugen. Die CGI-Clients kommunizieren mit dem Server über TCP/IP Sockets. Diese Kommunikation ist in Protokollen festgelegt. Näheres ist in [Kunze1998] und [Hanft2004] zu finden. Das System wurde vollständig in C++ realisiert.
Das bisherige ExpBook II bzw. TestManager System hielt die Fallbasis in Dateien vor, neue und geänderte Fälle wurden über eine Unterverzeichnisstruktur verwaltet. Zur Laufzeit besaß die Komponente DataManager eine Klasse zum Sortieren und Indexieren der Dateien, zu weiteren Details siehe wieder [Hanft2004].
Um den erweiterten Anforderungen mit dem zusätzlichen Status gerecht zu werden, wird die komplette Fallbasis jetzt über ein Datenbankmanagementsystem (DBMS) verwaltet.

Über Datenbank-Views lassen sich sehr einfach die unterschiedlichen Sichten des Modells realisieren. Zur Klärung von Begriffen zu Datenbanken (DB) wird auf [Vossen2000] verwiesen. Außerdem sind viele Anforderungen an ein verlässliches Softwaresystem wie Transaktionssicherheit, Persistenz, Wartbarkeit, Reaktionen auf Speicherung von Einträgen, Skalierbarkeit, Sortieren usw. mit einem DBMS adäquat zu lösen.
Als DBMS wurde die Open Source Datenbank PostgreSQL verwendet, da sie fortschrittliche Datenbankmerkmale wie z. B. Transaktionen, Stored Procedures und Trigger bietet.

Da die meisten Änderungen am *mtBook* im Vergleich zum ExpBook II durch die Einführung der Datenbank und der Reviews erfolgten, werden diese etwas ausführlicher beschrieben. Des Weiteren wird noch auf das unterschiedliche Fallformat eingegangen.

5.1 Objektmodell

Die Maintenance wurde innerhalb der Komponente DataManager und der Datenbank realisiert. Das Objektmodell wurde um eine Klasse Review erweitert. Für den Datenbankzugriff wurde eine Klasse dbWrapper hinzugefügt, welche den kompletten Zugriff auf die Datenbank kapselt. Während der DataManager über seine Schnittstelle fertig konfektionierte Zugriffsfunktionen für das Laden und Speichern von Objekten wie Fällen und Reviews bietet, enthält dbWrapper nur unterstützende Funktionen zum Ausführen von SQL-Abfragen.

Die bestehende Technik, die Weboberfläche über in C++ programmierte CGI-Clients zu gestalten, wurde beibehalten. Es ist lediglich eine Hilfsklasse HTMLHelper hinzugekommen, welche häufig wiederkehrende Aufgaben bei der Erzeugung von HTML-Code auslagert und so den Code der CGI-Clients kürzer und übersichtlicher gestaltet.

Der Code wurde mit speziellen Kommentare versehen, aus denen mit Hilfe von DoxyGen eine Hypertext-Dokumentation der Schnittstellen aller Klassen erstellt wurde. Diese ist elektronisch verfügbar, deswegen wird hier auf die ausführliche Beschreibung der Schnittstellen verzichtet.

5.2 DataManager Schnittstelle

5.2.1.1 Status in neuen Schnittstellen

Da es einen neuen Status gibt, haben alle öffentlichen Funktionen statt dem newerFlag einen Status entsprechend *eingereicht*, *akzeptiert* und *freigegeben* zu übergeben.

5.2.1.2 Abwärtskompatibilität

Die alten Funktionen der öffentlichen Schnittstelle enthielten zur Unterscheidung, ob die Daten aus der «retrieval» oder «authoring» Sicht gefordert sind, eine Variable `newerFlag`. Zur Abwärtskompatibilität wurden die alten Funktionen behalten, welche dann die neuen Funktionen mit dem Status *eingereicht* (bei `newerFlag = false`) bzw. *freigegeben* (bei `newerFlag = true`) aufrufen.

5.2.2 Die Klasse Review

Für das Laden und Speichern eines Reviews wurden die Funktionen

```
void getReview( Review & completeReview )
int saveReview( Review & savedReview )
```

hinzugefügt. Erstere lädt einen über Fall-ID und Revision spezifizierten Review als Objekt zurück, die zweite speichert einen übergebenen Review ab.
Zur Übersicht über die Reviews wird die Funktion

```
long getCaseAndReviewsByState( ReviewList & someReviews,
const t_revStatus rstState )
```

angesprochen, welche alle Reviews aus einer bestimmten Sicht inkl. dem Autor und der aggregierten Gesamtbewertung des Falls in `someReviews` zurückliefert.

5.3 Datenbank

5.3.1 Entwurf des Datenbankschemas

Für die Speicherung der Fälle und der Reviews inklusive Autoren und Begutachter genügt ein einfaches Datenbankschema, da hier auf die Modellierung der Fallstruktur über separate Tabellen wegen des Aufwands verzichtet wird. Stattdessen wird der komplette Fall(-text) wie bisher in einer Datei jetzt komplett in einem einzigen Textfeld `content` abgespeichert. Über den Deskriptor eines Wissensbausteins (WBD) wird ja jede Revision eines Falls eindeutig identifiziert. Daher könnte der Fallinhalt auch in anderen Tabellen, als binäres Objekt oder an anderen Speicherorten aufbewahrt werden. Der Einfachheit halber wird hier aber der Fallinhalt und WBD in eine Tabelle `t_case_revision` integriert. Für die Umsetzung des Sichtenkonzepts genügt es nicht, beim gewünschten Löschen eines Falls einfach den Tabelleneintrag zu löschen. Auch ist dann ein Wiederherstellen nicht möglich. Daher wird beim Löschen eines Falls dieser in der Spalte `isdeleted` markiert. Der `score` in dieser Tabelle speichert die Aggregierte Gesamtbewertung einer Fallrevision, während `score` beim Review in `t_review` die Revisionsbewertung eines Begutachters mit dem Kommentar `comment` darstellt. Dies zeigt folgendes DB Schema in Bild 10:

DB-Schema

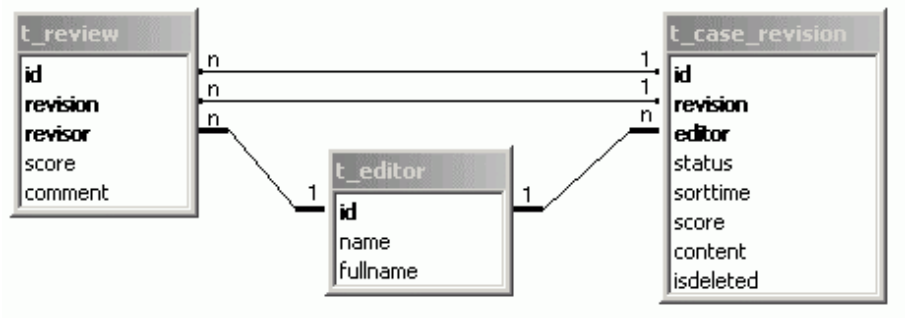


Bild 10: Datenbankschema für Fälle und Reviews

Hier sind auch die Kardinalitäten 1:n, die Fremdschlüsselbeziehungen als Linien und die Primärschlüssel(-kombinationen) als fettgedruckte Spaltennamen eingezeichnet. Daraus erkennt man zum Beispiel, dass Reviews nur zu bestehenden Fallrevisionen gespeichert werden können, wobei mehrere Reviews zur gleichen Fallrevision möglich sind. Ebenso kann es mehrere Reviews (zu unterschiedlichen Fallrevisionen) vom gleichen Begutachter geben.

5.3.2 Löschen und Wiederherstellen

Löschen einer Fallrevision

Das Löschen einer Fallrevision erfolgt über das Eintragen des Wertes `true` in der Spalte `isdeleted`. Dafür ist kein Benutzereingriff erforderlich, da dies automatisch nach Vergabe eines Reviews mit der Bewertung 0 erfolgt. Das Setzen auf `False` kann den Fall genauso einfach wiederherstellen.

Prozessmodell über Trigger und DB-Funktionen realisiert

5.3.3 Implementierung des Prozessmodells

Der Großteil des Prozessmodells wird über Trigger und Funktionen innerhalb der Datenbank umgesetzt.

5.3.3.1 Ändern von Zuständen durch Reviews

Akzeptanzfunktion f_set_case_acceptance

Ob eine Fallrevision akzeptiert wird, also deren Status eingereicht in den Zustand akzeptiert wechselt, wird durch eine einzige Akzeptanzfunktion `f_set_case_acceptance(int4, int4)` in der Datenbank gesteuert. Sie wird durch die Trigger `trf_review_trigger_update()` beim Einfügen und Ändern sowie `trf_review_trigger_delete()` beim Löschen von Reviews in `t_review` aufgerufen und schreibt die neu berechnete Aggregierte Revisionsbewertung als auch den Status in die Spalten in `score` und `status` des betroffenen Falls in der Tabelle `t_case_revision`.

5.3.3.2 Integration

Die Änderung des Status der Fallrevisionen von *akzeptiert* in *freigegeben* wird bei der Initialisierung des DataManagers und vor dem Aufbau des CRNs über die Ausführung der Funktion `f_set_update_casebase()` innerhalb der Datenbank erledigt.
Die Integration wurde in den Projekten TestManager und ExpBook II auch Update der Fallbasis genannt.

5.3.3.3 Auslösung der Maintenance

Der Integration folgend wird die Maintenance ausgelöst. Dazu wird ebenfalls eine Funktion der DB aufgerufen: `f_set_do_maintenance()`. Diese Funktion ist nur ein Container für mehrere Funktionen konkreter Wartungsstrategien, wie `f_set_mt_Older2Years()`. Diese implementiert die schon erläuterte Wartungsstrategie: Sie setzt alle Fallrevisionen mit dem Status *freigegeben*, welche älter als 2 Jahre sind, in den Status *eingereicht* zurück und legt davon eine Kopie als neue Revision an. Das Anlegen der Kopie als neue Revision ist notwendig, da sonst beim Verwendung der bisherigen Revision alle Reviews zu dieser gelöscht werden müssten. Die Wartungsstrategie kann einfach durch Editieren der Funktion in der Datenbank geändert werden. Weitere Wartungsstrategien können durch Erzeugen einer neuen DB-Funktion und Aufrufen dieser innerhalb von `f_set_do_maintenance()` hinzugefügt werden.
Dieser Vorgang findet ebenfalls bei der Initialisierung des DataManagers und vor dem Aufbau des CRNs statt. Danach kann das CRN aufgebaut werden.

5.3.3.4 Konsistenzwahrung der Sicht für das Retrieval

Zur Erstellung des Snapshots wird beim Initialisieren des DataManagers eine Funktion `f_set_RetrievalView()` aufgerufen, welche eine temporäre Tabelle `t_case_revision_retrievalview` anlegt. Beim Zugriff auf die Sicht für das Retrieval wird diese anstelle des Originals `t_case_revision` verwendet.

Die hier nicht umgesetzte Möglichkeit der Unterbindung der Zustandsübergänge wäre folgendermaßen zu realisieren:

1. Wird durch einen Review die Löschung einer Fallrevision mit dem Status *freigegeben* veranlasst, so wird nicht das isdeleted Flag der betreffenden Revision auf true gesetzt. Stattdessen wird eine neue Revision als Kopie angelegt und bei dieser der Status akzeptiert und das isdeleted auf true gesetzt. Aus den beiden unteren Sichten verschwindet der Fall sofort, während die ältere Revision in der Sicht für das Retrieval verbleibt.
2. Sollte ein Review eine Fallrevision mit dem Status *freigegeben* betreffen, wird innerhalb der oben erläuterten Funktion `f_set_case_acceptance()` der Status nicht geändert!
3. Innerhalb der globalen `f_set_do_maintenance()` wird `f_set_case_acceptance()` für alle Fallrevision mit dem Status *freigegeben* aufgerufen, um jetzt die Änderung des Status nachzuholen.

Eine Alternative zu dieser etwas kniffligen Wahrung der Konsistenz bei der Auslösung der maintenance besteht darin die Falldeskriptoren im CRN von einem Index auf die eindeutige Kombination aus Fall-ID und Revision umzustellen, so dass die Falldeskriptoren auch noch auf die richtigen Fälle verweisen, auch wenn sich deren Status schon längst geändert hat.

5.4 Web-Oberfläche

Von der Startseite existiert ein neuer Link zu den «Ausstehenden Reviews». Darauf werden entsprechend den Sichten drei verschiedene Reviewseiten angezeigt. Bild 11 zeigt die Weboberfläche der Seite «Ausstehende Reviews».

5.4.1 Das Sichtenkonzept auf der Oberfläche

Die meisten Clients können die geforderten Daten aus verschiedenen Sichten anzeigen. Dafür werden rechts oben unterhalb des Titels die Links auf die Sichten eingeblendet, welche gerade nicht angezeigt wird. So kann man schnell zwischen den verchiedenen Sichten wechseln. Zwei Clients erzeugen eine Übersichten: BrowseAllCasesCasesClient für die Fälle und BrowseAllReviewsClient für die Reviews. Bei ihnen wird nicht überprüft, ob Einträge in dieser Sicht vorhanden sind. Die Links werden immer angezeigt. Bei der Fall-Übersicht existiert noch ein zusätzlicher Link auf die gelöschten Fälle.

ExpBookI Übersicht Reviews - Mozilla Firefox

Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe

https://roy.informatik.hu-berlin.de/cgi-bin/ExpBookI/mtBookBrowseAllReviewsClient?View=Authoring

Ausstehende Reviews (Authoring-Sicht):

23 Fälle ohne Reviews oder mit Reviews, welche nicht ausreichend waren

Release-Sicht | Accepted-Sicht | neue Anfrage

Bitte klicken Sie auf die Revisionsnummer, wenn Sie einen Fall ansehen oder reviewen möchten.

Fall-ID	Revision	Fall-Autor	Gesamt-Score	Reviewer	Review-Score	Kommentar
3	1	kunze	2.000000	oschiem	2.000000	Man sollte die Beschreibung vielleicht einwenig erweitern.
				kschroet	1.000000	und 1000 andere ... hilft der Fall weiter?
				anonymous	3.000000	gibt es auch einen Editor, der von den Tastenkürzeln ähnlich wie unter Windows funktioniert?
23	1	hanft	2.500000	oschiem	3.000000	Die Erklärung ist eher zu lang.
				diaz	2.000000	andere Ursachen möglich, Titel näher spezifizieren
55	1	cbell				
56	1	hdb				
58	1	kschroet				
59	1	kschroet				
60	1	kschroet				
61	1	kramer				
62	1	kschroet				

Fertig roy.informatik.hu-berlin.de

Bild 11: Weboberfläche der Seite «Ausstehende Reviews»

Bei den Clients für den Falltext (CasetextClient, CasetextWithIesClient) hingegen wird innerhalb des Protokolls ein Bitset (als Zahl) vom Server geschickt, welche anzeigt in welcher Sicht es diesen Fall noch gibt. Nur dann wird der Link angezeigt.
Falls URLs im Falltext angegeben sind, werden diese unter den Textboxen zusätzlich als Links dargestellt. Auf der nächsten Seite wird im Bild 12 die Weboberfläche eines angezeigten Falltextes präsentiert. Für diesen Fall 76 mit der Revision 1 existiert noch keine Revision für die Suche. Der Link zur „Such-Revision“ (Sicht für das Retrieval) fehlt, darauf wird hingewiesen. Aber es existiert noch eine höhere Revision in der Sicht zum Editieren, zu der man über „neueste Revision“ gelangen kann. Unter dem Falltext ist der Abschnitt für den Review zu erkennen Der Review ist mit Name, Punktzahl und Kommentar schon fertig eingegeben, bereit zum Abschicken.

ExpBookI Text des Falls Nr. 76 - Mozilla Firefox

Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe

Text des Falls Nr. 76 (Revision 1, Accepted-Sicht):

Diese Revision steht erst nach einem Update der Fallbasis zur Verfügung..

neueste Revision | Texte editieren | Texte mit IEs | neue Anfrage

Titel:

Mail-Programm filter mit spamassassin fiel aus

Retrieval-Attribute:

Info-Attribute:

ersteltVon = Mirjam Minor
ersteltAm = 20.05.2003
MODE = Experte
geaendertAm = 20.05.2003
geaendertVon = Mirjam Minor
Revision = 1

Symptome:

Beim Filtern der Mailbox unter UNIX mit voreingestellten Mailadressen, die als Spamadressen bekannt sind (<A HREF="http://www.spamhaus.org">Spamhaus Blockliste</A> plus lokale Liste vom Postmaster), wurde mein .forward nicht mehr ausgefuehrt.

Spamhaus Blockliste

Schritte zur Lösung:

Ursache:

Wenn das Homeverzeichnis zum Schreiben fuer die Gruppe geoeffnet ist, ist filter misstrauisch und fuehrt .forward nicht mehr aus.

Lösung:

Die Zugriffsrechte des Homeverzeichnisses so einstellen dass nur der Nutzer selbst Schreibrechte hat.
Im Homeverzeichnis eine Datei namens .forward anlegen, darin steht:
"|/usr/local/bin/filter -F"
bzw.
"|usr/local/bin/filter -F -o ~/.elm/filtererrors", um eigene

Review (Bewertung der Fallrevision): Akzeptanzschwelle: 3

Begutachter: (login-Name reicht)

- 0 Punkte: Fall ist überflüssig, kann gelöscht werden
- 1 Punkt: unzureichend/funktioniert nicht
- 2 Punkte: nur teilweise Lösung/ enthält Fehler
- 3 Punkte: Lösung ausreichend, aber umständlich, Erklärung zu knapp
- 4 Punkte: gute Lösung, funktioniert, gut erklärt
- 5 Punkte: exzellente Lösung, erfolgreich ausprobiert, sehr gut erklärt

hanft

Kommentar des Reviews (an den Autor):

Könntest Du bitte noch den Befehl zur Vergabe der Dateirechte ergänzen?

Review absschicken!

Bild 12: Weboberfläche der Seite zum Falltext einschließlich der Eingabe des Reviews

5.5 Behandlung unterschiedlicher Fallformate

Als Codebasis diente ExpBook II. Das Fallformat des ExpBook I unterscheidet sich von diesem nicht nur syntaktisch <Tag> statt [Tag]. Das ExpBook I hat auch in Beschriftung und Anzahl andere Textabschnitte als ExpBook II. Ebenso unterscheiden sich spezielle Retrieval- und Infoattribute, welche beim Fall über Comboboxen eingegeben und angezeigt werden.
Die Namen der XML-Tags im Fallformat für die einzelnen Textabschnitte mussten zur Übersetzungszeit in der Headerdatei caseFormat.h vorliegen. Die Beschriftungen wurden hartverdrahtet in den CGI Clients im Code festgelegt. Möchte man ein neues FBS System aufsetzen, das sich nur in einem Textabschnitt unterscheidet, muss das komplette Projekt angepasst und neu übersetzt werden. Zur Verteidigung muss man natürlich anfügen, dass es sich dabei um einen Forschungsprototypen handelt.
Mit den Wissenscontainern aus ExpBook I soll gearbeitet werden. Um den Code für das Fallformat nicht wieder „zurück-“ zu entwickeln, wird ein verallgemeinertes Konzept für die Textabschnitte in Fällen vorgeschlagen, ohne den Aufwand eines kompletten (XML-) Parsers und der Übertragung ins Objektmodell auf sich zu nehmen. Es ist nicht völlig frei, erlaubt aber Modifikationen in einem Rahmen, der für alle bisherigen Projekte ausgereicht hätte:

Ein Fall hat

- 1 bis maximal 5 allgemeine Textabschnitte
- optional noch einen Textabschnitt als Fall-Titel
- 0 bis 2 Retrieval-AVPs
  (Attributname im Fallformat und Beschreibung)
- 0 bis 2 Info-AVPs
  (Attributname im Fallformat und Beschreibung)

Die Beschränkung auf augenblicklich 5 Abschnitte ist über die Konstante `coniTextSectionMax` einstellbar.

5.5.1 Konfiguration des Fallformats in der Ini-Datei

Alle Tags zu Textabschnitten, ihre Anzahl, jeweiliger Name des Tags im Fallformat und Beschreibung auf Weboberfläche stehen in der ini-Datei im Abschnitt [CaseFormat]. Leer gelassene Attribute bedeuten, dass dieser Abschnitt nicht im Fallformat existiert. TextSectionName*X* gibt die Beschriftung des Textabschnitts, TextSectionTag*X* das Tag im Falltext an. Kommentare sind mit einem Semikolon am Zeilenanfang markiert.
So sieht die Konfiguration für das ExpBook I Fallformat aus:

```
;;ExpBook I: Titel + 4 Textabschnitte
TextSectionTag0 = TITLE
TextSectionName0 = Titel:
TextSectionTag1= SYMPTOMS
TextSectionName1 = Symptome:
TextSectionTag2 = STEPS
TextSectionName2 = Schritte zur Lösung:
TextSectionTag3 = REASON
TextSectionName3 = Ursache:
TextSectionTag4 =SOLUTION
TextSectionName4 = Lösung:

;;Retrieval- und Info-AVP, dessen Wert Benutzer über Combobox auswählen kann
;;AVPAttr= Name des Attributs im Fallformat, AVPDes = Bezeichnung
;;2 Comboboxen für spezielle Retrieval Attribute
RetAVPAttribute1 = DEVICE
RetAVPDesc1 = Gerät
RetAVPAttribute2 = SOFTWARE_SYSTEM
RetAVPDesc2 = Software-System
;;ein InfoAttribut: Modus= Beide | Experten | Laie
InfAVPAttribute1 = MODE
InfAVPDesc1= Modus
```

Um den Unteschied zu zeigen, sei hier auch die Konfiguration für ExpBook II angeführt:

```
;;Format ExpBook II: 3 Textabschnitte ohne Titel
TextSectionTag0 =
TextSectionName0 =
TextSectionTag1= DESCRIPTION
TextSectionName1 = Problembeschreibung:
TextSectionTag2 = INSTRUCTION
TextSectionName2 = Lösung:
TextSectionTag3 = EXPECTED_RESULT
;Kommentare/alternative Lösung:
TextSectionName3 = Kommentare/alternative Lösungen(en):
TextSectionTag4 =
TextSectionName4 =
;;Retrieval- und Info-AVP, dessen Wert Benutzer über Combobox auswählen kann
;;AVPAttr= Name des Attributs im Fallformat, AVPDes = Bezeichnung
RetAttributeName1 = SUBDOMAIN
RetAttributeDes1 = Problemgebiet
;keine 2. Combobox
RetAttributeName2 =
;;keine zusaetzlichen Infoattribute
;InfAttributeName1 =
;InfAttributeName2 =
```

5.5.2 Einlesen der Ini-Datei

Eine statische Klassenfunktion `initializeCaseFormat()` der Klasse BasicTextualCase liest diese ini-Datei beim Hochfahren des FBS Systems aus und speichert die Namen, Beschreibung und Anzahl der Textabschnitte bzw. der speziellen Tags in Klassenvariablen, so dass sie global für alle TextualCaseObjekte zur Verfügung stehen. Auf die Tagnamen kann dann z. B. per `getTagName(const int index)` zugegriffen werden.

5.5.3 Verwendung im Objektmodell

Statt bei TextualCase über getDescription(), getInstruction()… wird jetzt über eine einzige Funktion

```
string getTextSection( const int index)
```

lesend als auch schreibend über

```
setTextSection(const int index,const String &newString)
```

zugegriffen. Der Index 0 ist für den Titelabschnitt. Als privates member besitzt jeder TextualCase dann anstatt theDescription, theExpected-Result… nur ein string Feld mit den 0 bis 5 Einträgen. Über

`static int getTextSectionCount()` bzw.

`static bool getHasTitle()`

kann herausgefunden werden wieviele Abschnitte und ob es einen Titel in diesem Fallformat gibt. Die Namen der Tags sind nicht öffentlich zugänglich, da sie nur in den Memberfunktionen `printCaseFormat()` und `scanCaseFormat()` gebraucht werden.

Die beiden Funktionen

```
static String getRetAVPAttr( const int iIndex )
```
und
```
static String getInfAVPAttr( const int iIndex )
```

liefern den Attributnamen des 0. oder 1. speziellen Retrieval- oder Info-AVPs. Die folgenden Funktionen

```
static String getRetAVPDesc( const int iIndex )
```
und
```
static String getInfAVPDesc( const int iIndex )
```

geben die Beschreibung dieser Attribute zurück, welche für die Comboboxen in der Weboberfläche verwendet wird.

5.5.4 Kommunikation mit den CGI Clients

Da die CGI Clients nichts über das Fallformat wissen, sieht das Protokoll vor, dass der Server jedes Mal bei der Anforderung eines Falltextes die Anzahl der Textabschnitte und ihre Beschreibung mitschickt.

6 Evaluation

6.1 Vorbereitung

Um die Evaluation an einer ausreichend großen Menge von Fällen durchzuführen, wurde die Fallbasis des ExpBook I verwendet. Dies waren 89 Fälle.
Alle Fälle älter als ein Jahr wurden zur Maintenance in den Status *eingereicht* zurückgesetzt. Dies waren 81 Fälle. Das Durchschnittsalter der Fälle betrug mehr als 3 Jahre und 7 Monate. Der älteste Fall war 6 Jahre und 8 Monate alt.
Es konnten fünf Mitarbeiterinnen und Mitarbeiter des Lehrstuhls zur Evaluation des mtBook Systems gewonnen werden. Ihnen wurde kurz der Review Prozess erklärt und die Oberfläche vorgeführt. Ziel der Evaluation war es zu überprüfen, ob der hier vorgestellte Prozess des Collaborative Maintenance ein geeignetes Verfahren ist, die maintenance an Fällen vorzunehmen.

6.2 Ergebnisse

Nach zwei Wochen wurde die Fallbasis mit den Reviews untersucht. Die Anzahl der Fallrevisionen war bis zu diesem Zeitpunkt auf 93 angewachsen. Die folgenden prozentualen Werte haben aufgrund der geringen Anzahl der Fälle und Reviews eine geringe statistische Relevanz. Es wurden 120 Reviews abgegeben, mit denen 89 Fallrevisionen (also 96%) aller Fallrevisionen begutachtet wurden. Von den 89 begutachteten Fallrevisionen wurden 12 aufgrund der Reviews gelöscht, 69 akzeptiert und 8 nicht akzeptiert. Die nicht akzeptierten Fälle müssen jetzt von den Autoren oder anderen Benutzern verbessert werden, damit eine neue Revision dieser Fälle auch akzeptiert wird. Durch diese maintenance wurden also 20 Fälle korrigierend behandelt (= 22%). Bis auf eine Ausnahme waren alle gelöschten Fälle älter als 4 Jahre. Es ergab sich keine Korrelation zwischen dem Alter der Fallrevisionen und ihrer Bewertung.
Die durchschnittliche aggrgierte Bewertung aller begutachteten Fallrevisionen inklusive der gelöschten betrug 2,9 Punkte. Zählt man nur die akzeptierten Fallrevisionen, so haben diese eine durchschnittliche Aggregierte Bewertung von 3,48 Punkten.
Von den Reviews aus betrachtet ergibt sich folgendes Bild: Jede Fallrevision wurde ein- bis dreimal, im Durchschnitt mit 1,35Reiews begutachtet. 46 von 120 Reviews wurden mit Kommentaren versehen, dies sind 38%. Die durchschnittliche in einem Review vergebene Punktzahl betrug 2,98.

Zur Übersicht sind noch einmal alle Zahlen in einer Tabelle aufgeführt:

Fallrevisionen	93	
Begutachtete Fallrevisionen	89	96%
Note = 0 (Gelöscht)	12	13%
Note < 3 (Nicht akzeptiert)	8	9%
Note >= 3 (akzeptiert)	69	78%
∅ Aggregierte Bewertung der Fallrevisionen	2,9	
∅ Aggr. Bewertung der akzeptierten Fallrevisionen	3,49	
Korrelation zwischen Alter und Bewertung	-0,07	
Reviews	120	
Reviews die zur Löschung führten	12	10%
Reviews mit Kommentaren	46	38%
∅ Anzahl Reviews je Fallrevision	1,35	
Durchschnittsnote ohne Löschen	3,31	
Durchschnittsnote mit Lösch-Reviews	2,98	
Zeitbedarf je Review	2-4 min	

Zusätzlich wurden die Tester kurz interviewt und nach dem Zeitaufwand befragt. Der angegebene Zeitaufwand für einen Review schwankte je nach Komplexität des Falls und dem Zeitbedarf für das Nachvollziehen der Lösung durchschnittlich von 2 min bis zu 3:36 min (50 Reviews in 3 Stunden).

6.2.1 Handhabung der Oberfläche

Die zum Review gewählte Menge der Fallrevision ist gerade noch von einem Begutachter überschaubar, wenn die Auswahl der Reviews durch die Begutachter selbst erfolgen soll.
Für den Prototypen ergab sich, dass die Zustandsübergänge deutlicher zu kennzeichnen sind, damit die Benutzer wissen, was wann mit ihrem Review passiert und wohin die begutachtete Fallrevision „verschwindet“. Bis jetzt wird nur im Text nach dem Speichern eines Reviews auf den neuen Zustand der Fallrevision hingewiesen.

Das Sortieren nach den Spalten der Tabellen und Filtern (z.B. nach Autor) innerhalb der Übersichten würde die Handhabung verbessern. Es wurde noch die Anzeige verschiedener Daten gewünscht, wie z. B. die Angabe des Autors in der Fallübersicht und der Titel in der Review-Übersicht.

Insgesamt gibt es sieben verschiedene Übersichtsseiten (Sichten). Dies ist für Benutzer mitunter zu viel des Guten, da einigen Benutzern nicht immer sofort die Unterschiede zwischen den Sichten auf die Fallbasis und die Reviews klar waren.

Der entwickelte Prototyp verfügt über eine akzeptable Benutzerfreundlichkeit und läuft stabil. Daher kann das bestehende, alte ExpBook I abgeschaltet und durch das neue System ersetzt werden.

6.3 Fazit

Die Evaluation hat gezeigt, dass das Modell des hier vorgestellten Review Prozesses geeignet ist, die maintenance an einem FBS System durchzuführen. Die häufige Verwendung des Textkommentars beim Review zeigt eindrucksvoll die Nützlichkeit dieses Instruments. Dadurch wird die Kommunikation bei der Erstellung und Änderung von Fallrevisionen verbessert. Die kurzen Zeiten für die Abgabe eines Reviews bestätigen, dass das vorgestellte Modell mit geringem Zeitaufwand zu warten ist.
22% gelöschte und nicht akzeptierte Fallrevisionen sowie eine unter der Akzeptanzschwelle liegende Durchschnittsnote von 2,9 zeigen, dass die Fallbasis eine maintenance nötig hatte. Die Qualität der akzeptierten Fallrevisionen ist mit 3,49 aber gut.
Äußerst wichtig für die Annahme eines solchen Collaborative Maintenance Systems durch die Benutzer ist die Erklärung der dahinterliegenden Prozesse. Daraus folgt, dass die Prozesse bei Bedarf an die Gegebenheiten der Organisation anzupassen sind.

Für größere Mengen von Falländerungen und Reviews muss die Zuordnung der Reviews zu den Begutachtern automatisiert werden, da die Menge der überschaubaren Fallrevisionen mit 80 Stück erreicht ist.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde das System *mtBook* für die Benutzung und Maintenance von Erfahrungen entwickelt. Im nächsten Abschnitt wird das Erreichte zusammengefasst. Im Rahmen einer Diplomarbeit können nicht alle Aspekte ausführlichst behandelt werden, auf mögliche Weiterentwicklungen geht daher der Ausblick ein.

7.1 Zusammenfassung

Das *mtBook* ist ein Experience Management System, welches Collaborative Maintenance zur Wartung seines Wissens einsetzt.
Die hier vorgestellte Beispieldomäne sind administrative und technische Probleme im Umgang mit der EDV an einem Lehrstuhl, die Benutzer sind Mitarbeiter dieses Lehrstuhls. Das System lässt sich aber leicht auch in anderen Domänen verwenden, da das hier vorgestellte Konzept der Maintenance allgemein gültig ist. Nur die Anpassung des Ähnlichkeitsmodells für das FBS ist notwendig.
Im *mtBook* können die Benutzer neues Wissen in Form von Fällen hinzufügen als auch die Fälle der anderen bewerten. Durch diese gegenseitige Kontrolle wird 1. regelmäßig das existierende Wissen überprüft und 2. die Qualität neu eingetragenen Erfahrungswissens sichergestellt.
Das dafür entworfene Collaborative Maintenance Modell für den Life-Cycle eines Wissensbausteins basiert auf der Revision eines Wissensbausteins, dessen Status und Bewertung. Durch die Definition der Übergänge zwischen den Zuständen wird das vorgestellte Modell als ein vollständiger Prozess definiert. Für die Behandlung der unterschiedlichen Zustände und Revisionen neben einem ständig möglichen Retrieval wurde das Sichtenkonzept eingeführt.
Das *mtBook* ist das erste System, welches Collaborative Maintenance für die maintenance in einem FBS-System für Texte anwendet.

mtBook[6] verfügt über eine Weboberfläche mit der die Benutzer ohne großen Einarbeitungsaufwand arbeiten können. Die Evaluation hat die erfolgreiche Anwendbarkeit dieses Konzepts am Beispiel der Wartung von Fällen gezeigt.
Da es keinen Administrator benötigt und die maintenance automatisch auslöst, eignet es sich hervorragend für Personengruppen, welche die Wartung und Benutzung nebenbei erledigen müssen. Leichte Verständlichkeit der dahinterliegenden Prozesse, schnelle Handhabung und geringer Overhead zeichnen diese leichtgewichtige Prozessmodell aus.
Die einzigen Voraussetzungen für Collaborative Maintenance sind eine Kultur unter den Benutzern, Wissen zu teilen, und die Bereitschaft zur Teilnahme am Prozess der Wissenserstellung.

[6] Das System ist für einen eingeschränkten Kreis von Benutzern unter https://roy.informatik.hu-berlin.de/ExpBookI verfügbar.

7.2 Ausblick

Die erfolgreiche Anwendung des Collaborative Maintenance für die Wartung eines Fallbasierten Systems bietet noch viele Ansatzpunkte für Erweiterungen.

Die verwendeten Wartungsstrategien können ausgebaut werden, z. B. die Auslösung der Reviews in Abhängigkeit von der Benutzungshäufigkeit. Ebenso sind mehrstufige Reviewprozesse und Bonussysteme denkbar, wie schon in *4.2.4.4 Erweiterungsmöglichkeiten auf Seite 53* vorgestellt.

Die Möglichkeit bei WIKIPEDIA jede einzelne Version eines Artikels mit einer anderen zu vergleichen ist eine gute Idee, die man für Fälle auch ins *mtBook* übernehmen könnte. Anders herum könnte man auch das in dieser Arbeit vorgestellte Maintenance Modell auf WIKIPEDIA übertragen. Dann würden die Nutzer nur die allerneueste Version sehen, wenn sie schon von einem anderen Nutzer positiv begutachtet wurde.

Erfahrung über die Wartung des Systems wie im EMSIG Framework wird bis jetzt nicht im *mtBook* ausgewertet. Dies wäre aber eine Möglichkeit weiterer Arbeiten. In gewisser Weise geht das als Option vorgeschlagene Bonussystem in diese Richtung.

Denkbar wäre auch die Idee Revisionen aus allen Sichten ins CRN zu integrieren und eine unterschiedliche Gewichtung anhand des Status vorzunehmen. Dazu würden beim Laden die Fallrevisionen aus allen drei Sichten in die Fallbasis geladen. Das Retrieval könnte wie gehabt ausgeführt werden. Nach dem die Liste der ähnlichen Fälle ermittelt wurde, werden die Fälle anhand ihres Status noch einmal gewichtet und die Liste erneut sortiert um sie anschließend dem Benutzer zu präsentieren.

Für eine weitere Qualitätsverbesserung der Fälle wäre die Implementierung einer Inkonsistenz- und Redundanzerkennung, wie schon in *3.5* Maintenance schwach strukturierter Fälle *auf Seite 34* erläutert.

Auch wenn die während der Evaluation geäußerten Verbesserungsvorschläge zum Teil auch in den bestehenden Clients implementiert werden können, so sollte doch zur Entsprechung aller Anforderungen und zukünftiger Erweiterbarkeit auf ein moderneres System zur Programmierung von Webseiten umgestiegen werden. Dafür empfiehlt sich ein Content Managementsystem oder eine Portalsoftware. Damit können auch weitere Aufgaben wie Anmeldung, personalisierte Seiten und Usertracking adäquat und mit vertretbarem Aufwand realisieren werden.
Mithilfe dynamischer Webseiten könnten die Übersichtsseiten zu den Reviews mit den Fallübersichten vereinigt werden, wo nur bei Bedarf die zugehörigen Reviews eingeblendet werden.

Abschließend hofft der Autor mit dem leichgewichtigen Prozessmodell Experience Management in Umgebungen zu ermöglichen, in denen bisherige mächtige und kommerzielle Frameworks einfach überdimensioniert sind.

8 Anhang

8.1 Abbildungsverzeichnis

Bild 1: 4-R-Zyklus nach Aamodt & Plaza, entnommen aus [Lenz1999] 14

Bild 2: Case-Retrieval Net, entnommen aus [Bergmann2002, S.207] 16

Bild 3: Knowledge Containter nach Richter 20

Bild 4: Kategorisierung von maintenance Ansätzen nach Leake & Wilson 27

Bild 5: Überblick das maintenance Schema EMSIG, entnommen aus [NickAlthoff2001] 31

Bild 6: Konzeptuelles Modell für die maintenance einer EB, entnommen aus [NickAlthoff2001] 32

Bild 7: Ein typischer Life-Cycle eines (Test-)Falls 41

Bild 8: Prozessmodell des Life-Cycle eines Wissensbausteins mit den Zuständen 44

Bild 9: UML-Aktivitätsdiagramm des Review Prozesses 51

Bild 10: Datenbankschema für Fälle und Reviews 63

Bild 11: Weboberfläche der Seite «Ausstehende Reviews» 66

Bild 12: Weboberfläche der Seite zum Falltext einschließlich der Eingabe des Reviews 67

8.2 Definitionen

Definition 1: **Wissensmanagement** 11
Definition 2: **Experience Management** 11
Definition 3: **Daten** 11
Definition 4: **Information** 11
Definition 5: **Wissen** 12
Definition 6: **spezielles Wissen** 12
Definition 7: **allgemeines Wissen** 12
Definition 8: **Erfahrung** 12
Definition 9: **Wissensbasierte Systeme** 12
Definition 10: **Informationseinheit (IE)** 13
Definition 11: **Fall** 13
Definition 12: **Fallrepräsentation** 13
Definition 13: **Fallbasis** 13
Definition 14: **Anfrage** 13
Definition 15: **Anfragerepräsentation** 13
Definition 16: **lokale Ähnlichkeitsfunktion** 14
Definition 17: **Ähnlichkeitsfunktion (globale)** 14
Definition 18: **Case Retrieval Netz** 15
Definition 19: **Textuelles Fallbasiertes Schließen** 17
Definition 20: **Indexterm** 18
Definition 21: **Indexvokabular** 18
Definition 22: **Text-IE** 18
Definition 23: **Attribut-IE** 18
Definition 24: **IE- Kategorie** 18
Definition 25: **Indexlexikon** 18
Definition 26: **Wissenscontainer (WC)** 19
Definition 27: **maintenance** 21
Definition 28: **utility Problem** 22
Definition 29: **swamping Problem** 22
Definition 30: **maintenance für FBS** 22
Definition 31: **knowledge maintenance** 23
Definition 32: **meta maintenance** 26
Definition 33: **collaborative maintenance** 37
Definition 34: **Life-Cycle eines Falls** 42

Definition 35: **Wissensbaustein (WB)** 44
Definition 36: **Deskriptor eines Wissensbausteins (WBD)** 45
Definition 37: **Revision eines Wissensbausteins (WBR)** 45
Definition 38: **Status (einer Revision)** 45
Definition 39: **Sicht (zu gegebenem Mindeststatus)** 47
Definition 40: **Review Prozess** 48
Definition 41: **Review** 49
Definition 42: **Revisionsbewertung** 49
Definition 43: **Akzeptanzfunktion** 50
Definition 44: **Aggregierte Bewertung** 50
Definition 45: **Akzeptanzschwelle** (acceptance level) 50
Definition 46: **Maintenance Modell für den Life-Cycle eines Wissensbausteins (MML)** 55
Definition 47: **Collaborative Maintenance Modell für den Life-Cycle eines Wissensbausteins (CMML)** 56

8.3 Stichwortverzeichnis

4-R-Zyklus 14
Abstraktionsstufe 34
Aggregierte Bewertung 50
Ähnlichkeit
  globale Ähnlichkeit 42
  partielle Ähnlichkeit 42
Ähnlichkeitsmaß 19, 44, 52
Akzeptanzfunktion 50, 53, 64
Akzeptanzschwelle 50, 53, 55
Anfrage 13
Anwendungsphase 33
Case ..... *Siehe* Fall
Collaborative Maintenance 36, 37, 43, 60
continuous stream of experience 30
Copyrightverletzung 56, 60
Datenverschmutzung 56
DISER 30
EbIS 30
Erfahrung 7, 12, 30
  continuous stream of experience 30
ExpBook I 58
ExpBook II9, 37, 53, 58, 61, 68, 69
Experience Factory 30
experience package 30
Experience Repository 30
ExperienceBook *Siehe* ExpBook I
ExperienceBook II ..... *Siehe* ExpBook II
Fall 13
Fallbasis 19
Fallrevision 52, 57
Fehlerquote 56
globale Ähnlichkeit 42
häufige Änderungen 56
Historie 57
Indexterm 20
Information Retrieval 35
Inkonsistenz 35
INRECA 34
Know-How 56
knowledge items ..... *Siehe* Wissensbaustein
Knowledge Life-Cycle Modell 21, 32
Knowledge Management .... *Siehe* Wissensmanagement
Kompetenz 60
  heterogene 60
Konsistenzverletzung 35
Lebenszyklus.... *Siehe* Life-Cycle
Life-Cycle 41, 43, 55
  Collaborative Maintenance Modell für den 56
  Knowledge Life-Cycle Modell 21, 32
  Maintenance Modell für den55
Lösungstransformation 19
Maintenance **21**, 55
  Auslösung der 55
  Knowledge ~ 23
  sytematische 30
Maintenance-Phase 33
Mapping 34
Ontologie 56
partielle Ähnlichkeit 42
Portal 36, 38, 53, 76
Qualität 56, 60
  heterogene 60
Query ..... *Siehe* Anfrage
Rechtssicherheit 56
Reifegrad ..... *Siehe* Status
Repräsentationssprache 19
Restore 33
Retain 15
Retrieve 15
Reuse 15
Review28, 33, 36, 43, 44, 46, 48, 49, 55, 59, 62, 63
Review Prozess 48, 51
  Aufwand vs. Nutzen 56
Revise 15
Revision 45
Revisionsbewertung 49
Revisionssicherheit 57
Sicht **46**, **47**, 50, 61, 62, 65, 66
  «case authoring» 17, 42
  «retrieval» 43
  auf Ergebnis des Review-Prozesses 46

für das Retrieval.46, 48, 53, 64
zum Editieren.....46
Status 43, **45**
akzeptiert 48, 50
eingereicht 48, 50, 71
freigegeben 48
Zustandsübergang...43, 48, 65, 72
Systematic Maintenance 30
TestManager9, 10, 15, 29, 37, 38, 43, 45, 49, 61
Vier-Augen-Prinzip 43
View .....*Siehe* Sicht
Vokabular 19
WIKIPEDIA 9, 60
Wissensbaustein 43, 44
Qualität 49
Revision eines ~.....45
Wissenscontainer.....19, 44
Wissensmanagement 11
Zusammenführen von Ontologien 56

9 Literatur

[AamodtPlaza1994] Aamodt A., Plaza E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches, S. 39–59 In AICom – Artificial Intelligence Communications, Vol. 7: 1, IOS Press, 1994

[AlthoffEtal2001] Althoff K.-D., Decker B., Hartkopf S., Jedlitschka A., Nick M. und Rech J.: Experience Management: The Fraunhofer IESE Experience Factory. In P. Perner (ed.), Proc. Industrial Conference Data Mining, Institut für Bildverarbeitung und angewandte Informatik, Leipzig 2001

[Balzert1998] Balzert Helmut: Lehrbuch der Software-Technik, Software-Mangement, Software-Qualitätssicherung, Unternehmensmodellierung (Band 2), Spektrum Akademischer Verlag, Heidelberg 1998

[BasilEtal1994] Basili V.R., Caldiera G., Rombach D.: Experience Factory, S. 469–476 In Marciniak J.J. (ed.), Encyclopedia of Software Engineering, volume 1, John Wiley & Sons Inc, New York 1994.

[Bell2004] Bell Christina: SIMLEX – Experience Management für die Simulationsliga des RoboCups, Diplomarbeit, Humboldt-Universität zu Berlin, Berlin 2004

[Berghofer2003] Roth-Berghofer Thomas R.: Knowledge Maintenance of Case-Based Reasoning Systems – The SIAM Methodology, Dissertation an der Universität Kaiserslautern, verfügbar als DISKI 262, Akademische Verlagsgesellschaft Aka GmbH, Berlin 2003

[Berghofer2003a] Roth-Berghofer Thomas R.: Developing maintainable CBR Systems: Applying SIAM to empolis orenge, S. 305-306 In Proceedings of GWEM 2003 in Reimer Ulrich, Abecker Andreas, Staab Steffen, Stumme Gerd (Eds.): WM 2003: Professionelles Wissensmanagement - Erfahrungen und Visionen, Beiträge der 2. Konferenz Professionelles Wissensmanagement, 2.-4. April 2003 in Luzern, LNI P-28 GI 2003, Bonn 2003

[Bergmann2002] Bergmann Ralph: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, LNAI 2432, Springer-Verlag, Berlin 2002

[BergmannEtAl1999] Bergmann Ralph, Breen Sean, Göker Mehmet, Manago Michel, and Wess Stefan: Developing Industrial Case–Based Reasoning Applications: The INRECA Methodology. State-of-the-Art-Survey, LNAI 1612 Springer-Verlag, Berlin 1999

[BorgeltEtAl2003] Borgelt Christian, Timm Heiko, Kruse Rudolf: Unsicheres und vages Wissen, S. 291–347 in Görz G., Rollinger C.-R., Schneeberger J. (Hrsg): Handbuch der Künstlichen Intelligenz, 4. Auflage, Oldenburg Wissenschaftsverlag, München 2003

[Busch2001] Busch Karl-Heinz, Siemens A & D: Was ist Qualität? Wie mißt man Qualität?, Vortrag auf dem Workshop 5 auf der WM2001: http://demolab.iese/fhg.de:8080/AK-Praktisches-Wissensmanagement/WM2001-WS5

[BuschmannEtal1998] Buschmann Frank, et al: Pattern-orientierte Software-Architektur, Ein Pattern-System, Addison Wesley Longmann, Bonn 1998.

[Dietterich1986] Dietterich Thomas G.: Learning at the knowledge level, S. 287-316 in Readings in Machine Learning Volume 1, Morgan Kaufmann Publishers Inc., San Mateo 1986

[FerrarioSmyth2000] Ferrario Maria Angela and Smyth Barry. A user-driven, distributed maintenance strategy for large-scale case-based reasoning systems, S. 55–63 In Mirjam Minor (editor), ECAI Workshop Notes, Humboldt-Universität zu Berlin, 2000.

[FrancisRam1993] Francis A. G. and Ram A.. The Utility Problem in Case-Based Reasoning. Technical Report (ER-93-08). Georgia Institute of Technology, USA, 1993.

[Groenendijk1996] Groenendijk Wim: Maintenance management system: Structure, interfaces and implementation. In Süleyman Özekici, editor, Reliability and Maintenance of Complex Systems, NATO ASI Series, Computer and Systems Sciences SERS F, Volume 154, Springer-Verlag, Berlin 1996

[Hanft2004] Hanft Alexandre: Entwurf und Dokumentation eines Life Cycle Modells in einem FBS System, Studienarbeit, Humboldt-Universität zu Berlin, Berlin 2004

[HeisterWilke1998] Heister F. and Wilke W.. An architecture for maintaining case-based reasoning systems, in Cunningham P., Smyth B. and Keane M. (eds.): Proceedings of the 4th European Workshop on Case-Based Reasoning: Advances in Case-Based Reasoning, Springer- Verlag, Berlin 1998

[Kolodner1993] Kolodner J. L.: Case-Based Reasoning. Morgan Kaufmann Publishers, San Mateo 1993

[KPMG1998] KPMG: Knowledge Management Research Report 1998, http://www.kpmg.com

[Kunze1998] Kunze Mirjam: Das ExperienceBook – Dokumentation eines fallbasierten Systems zur Unterstützung der Systemadministration Diplomarbeit, Humboldt-Universität zu Berlin, 1998. http://www.informatik.hu-berlin.de/~minor/Publications/diplomarbeit.zip

[Kurzidim2004] Kurzidim Michael: Wissenswettstreit – Die kostenlose Wikipedia tritt gegen die Marktführer Encarta und Brockhaus an, S. 132–136 in c't 21/04, Heise Zeitschriften Verlag, Hannover 2004

[LeakeWilson1998] Leake D. B. and Wilson D. B.: Categorizing Case-Base Maintenance: Dimensions and Directions, S. 196-207 In Cunningham P., Smyth B. and Keane M. (eds.): Proceedings of the 4th European Workshop on Case-Based Reasoning: Advances in Case-Based Reasoning, Springer- Verlag, Berlin 1998

[Lenz1999] Lenz Mario: Case Retrieval Nets as a Model for Building Flexible Information Systems, Dissertation, Humboldt-Universität zu Berlin, Berlin 1999.

[LenzEtal1998] Lenz M., Hübner A., Kunze M.: Textual CBR, S. 115–137 In Lenz M., Burkhard H.-D., Bartsch-Spörl B., Wess S. (Eds.): Case-Based Reasoning Technology - From Foundations to Applications, , LNAI 1400, Springer-Verlag, Berlin 1998.

[LenzEtal1999] Lenz M., Busch K.-H., Hübner A. and Wess S.: The SIMATIC Knowledge Manager, S. 40–45 In Aha D. W. and Mũnoz-Avila H., Exploring Synergies of Knowledge Management and Case-Based Reasoning. Technical Report AIC-99-008, Naval Research Lab 1999.

[MinorHanft1999] Minor Mirjam und Hanft Alexandre: Cases with a Life-Cycle, in Schmitt S., Vollrath I. (Eds.): Challenges for Case-Based Reasoning – Proceedings of the ICCBR'99 Workshops, Universität Kaiserslautern, 1999.

[MinorHanft2000] Minor Mirjam und Hanft Alexandre: The Life Cycle of Test Cases in a CBR System, S. 455–466 in Proceedings of EWCBR-2000, LNAI 1898, Springer-Verlag, Berlin 2000.

[MinorHanft2000b] Mirjam Minor, Alexandre Hanft: Corporate Knowledge Editing with a Life Cycle Model. In Proceedings of the 8th German Workshop on Case-Based Reasoning, GWCBR-2000: Göker Mehmet (Ed): DaimlerChrysler Research and Technology FT3/KL, Ulm, März 2000.

[MinorLenz2001] Minor Mirjam und Lenz Mario: Textual CBR im E-Commerce, Zeitschrift Künstliche Intelligenz, Heft 01/ 2001, arendtap Desktop Publishing Agentur, Verlags- und Vertriebs GmbH, Bremen 2001

[MühlbrFeggeler2003] Mühlbradt Th. und Feggeler A.: Wissensorientierte Analyse, Gestaltung und Unterstützung von Geschäftsprozessen in KMUs, Tutorial 6 auf der WM 2003: http://wm2003.aifb.uni-karlsruhe.de/tutorial/t06/ (04.11.2004)

[NickAlthoff2001] Nick M. and Althoff K.-D. Acquiring and Using Maintenance Knowledge to Support Authoring for Experience Bases, In Proceeding 4th International Conference on Case-Based Reasoning (ICCBR'01) Workshop on Authoring Support Tools, Vancouver, Canada 2001

[NickEtal2001] Nick Markus, Althoff Klaus-Dieter and Tautz Carsten: Systematic maintenance of corporate experience repositories, S. 364–386 In Goebel Randy, Greiner Russell, and Lin Dekang (Eds.) Computational Intelligence – Special Issue on Maintaining CBR Systems, Volume 17(2), Blackwell Publishers, Cambridge, Massachusetts 2001

[RacineYang1997] Racine Kirsti and Yang Qiang: Maintaining Unstructured Case Bases, S. 553–564 in Leake David B., Plaza Enric (eds.): Case-Based Reasoning Research and Development, Proceedings of 2nd International Conference on CBR ICCBR-97, LNAI 1266, Springer-Verlag, Berlin 1997

[ReinartzEtal2000] Reinartz Th., Iglezakis I. and Roth-Berghofer Th.: On quality measures for case base maintenance, S. 247–259 in Blanzieri E. and Portinale L. (eds): Proceedings of the Fifth European Workshop on Case-Based Reasoning, Springer Verlag, Berlin 2000

[Richter1998] Richter Michael M.: Introduction, Kapitel 1 S. 1–15 in Lenz M., Burkhard H.-D., Bartsch-Spörl B., Wess S. (Eds.): In Case-Based Reasoning Technology - From Foundations to Applications, LNAI 1400, Springer-Verlag, Berlin 1998.

[Richter2003] Richter Michael M.: Fallbasiertes Schließen, S. 407–430 in Görz G., Rollinger C.-R., Schneeberger, J. (Hrsg): Handbuch der Künstlichen Intelligenz, 4. Auflage, Oldenburg Wissenschaftsverlag, München 2003

[Schaaf1996] Schaaf J. W.: Fish and Shrink: a next step towards efficient case retrieval in large scale case bases. In Smith I. and Faltings B.: Advances in Case-Based Reasoning, LNAI 1186 Springer-Verlag, Berlin 1996

[Schank1982] Schank R.: Dynamic Memory: A Theory of Learning in Computers and People. Cambridge University Press, New York, 1982.

[Swanson1976] Swanson E. Burton : The dimensions of maintenance. In Proceedings of 2nd International Conference on Software Engineering, S. 492–497, IEEE Computer Society Press Los Alamitos, CA, USA San Francisco, California, United States 1976

[RiesbeckSchank1989] Riesbeck C. K. and Schank R. C.: Inside Case-Based Reasoning, Lawrence Erlbaum Associates 1989

[SmythKeane1995] Smyth B., Keane M.T.: Remembering to Forget: A Competence Preserving Deletion Policy for Case-Based Reasoning Systems, S. 377–382 in Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan-Kaufmann, San Mateo 1995

[Vossen2000] Vossen Gottfried: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage, Oldenbourg Wissenschaftsverlag, München 2000

[Wess1995] Wess Stefan: Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik, Dissertation an der Universität Kaiserslautern, verfügbar als DISKI 126, infix Verlag, Berlin 1995

[Wilson2001] Wilson David C.: case-based maintenance: the husbandry of experience, PhD thesis, Indiana University, http://www.cs.indiana.edu/~leake/papers/d-01-01.pdf, 2001

[WilsonBradshaw1999] Wilson David C. and Bradshaw Shannon. CBR Textuality. In Proceedings of the Fourth UK Case-Based Reasoning Workshop. pp. 67-80. AI-CBR, University of Salford. 1999

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 25.11.2004

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 25.11.2004