Universität Kaiserslautern
Fachbereich Informatik

Seminararbeit zum Thema

Software Reuse Repositories

Andreas Kohlmaier

Im Rahmen des Seminars

Datenbanksysteme: Anwendungsentwicklung im WS 1999 / 2000

AG Datenbanken / Informationssysteme Prof. Härder

Betreuung:
Dipl. Inform. Wolfgang Mahnke

1. Abstract

Dieses Seminar beschäftigt sich mit Repository Technologien, die zur Wiederverwendung von Software eingesetzt werden. Um sich mit diesem Gebiet auseinanderzusetzen, ist es zunächst nötig, den Begriff des *Software Reuse* zu klären. Dabei ist insbesondere interessant, welche Artefakte wiederverwendet werden (z.B. Codefragmente, Testtechniken, Entwürfe) und wie diese Wiederverwendung in den Softwareentwicklungsprozess eingeht. Kapitel 2 beschäftigt sich mit diesen Fragen und stellt das *Quality Improvement Paradigm* und die daraus resultierende *Experience Base* als zentrales Repository vor. Kapitel 3 erläutert und präzisiert den etwas allgemeineren Begriff des *Software Reuse Repositories* und stellt Eigenschaften und Maße zur Beurteilung solcher Repositories vor. Kapitel 4 nutzt diese Eigenschaften zur Einteilung bestehender Systeme in sechs Klassen. Diese Klassen werden an Hand von Beispielen erläutert. Kapitel 5 stellt zwei konkrete Systeme, *CodeFinder* und die *SFB 501 Experience Base,* im Detail vor.

2. Software Reuse

Ein wichtiges Ziel bei der Softwareentwicklung ist es, bereits entwickelte Komponenten in geeigneter Weise wiederzuverwenden, um Zeit und Kosten zu sparen. Dieses einleuchtende Ziel ist in der Praxis weitaus schwieriger zu erreichen, als man zunächst annehmen möchte. So ist ein erstes Hindernis beispielsweise, dass in einem großen Unternehmen gar nicht überall bekannt ist, was bereits entwickelt wurde. Aber selbst wenn bekannt ist, das eine Komponente bereits entwickelt wurde, so ist diese unter Umständen nicht zur Wiederverwendung geeignet. Entweder wegen mangelnder Dokumentation, zu geringer Modularität oder einfach dadurch, dass sie auf einem unpassenden oder veralteten System entwickelt wurde.

Um diese Hindernisse erfolgreich zu überwinden stellt Basili et al. einen eigenen Prozess, das *Quality Improvement Paradigm* vor. Dieser Prozess unterstützt die Wiederverwendung bei der Softwareentwicklung entscheidend. Ein Kernelement ist dabei die sogenannte *Experience Base.* Die Experience Base ist eine zentrale Einheit, die sich um die Wiederverwendung bemüht. Darunter fällt einerseits die nötige Technologie und andererseits ein nicht zu unterschätzender personeller Aufwand. Als Technologie ist ein unternehmensweites Repository zur Speicherung aller im Softwareentwicklungsprozess entstandener Artefakte nötig, das eine geeignete Suchfunktion anbietet. Der personelle Aufwand ist nötig, um die im Softwareentwicklungsprozess entstandenen Artefakte in geeigneter Weise aufzubereiten und zu dokumentieren, so dass sie in anderen Projekten wiederverwendet werden können.

2.1. Das Quality Improvement Paradigm

Die grundlegende Idee des QIP nach [BCR94] ist, dass alle im Softwareentwicklungsprozess entstandenen Artefakte einen Reifeprozess durchlaufen. So können z.B. die zur Durchführung eines Projekts benötigten und neu entwickelten Tools in einem anderen Projekt wiederverwendet und weiterentwickelt werden. Die zentrale Einheit zur Speicherung dieser Artefakte ist die Experience Base. Dabei werden zu Beginn des Prozesses verschiedene Artefakte aus der Experience Base entnommen, und nach Beendigung des Softwareentwicklungsprozesses werden die veränderten Artefakte wieder in die Experience Base zurückgeschrieben. Abbildung 1 listet die einzelnen Phasen im QIP und zeigt, welche Artefakte aus der Experience Base entnommen, bzw. in die Experience Base aufgenommen werden:

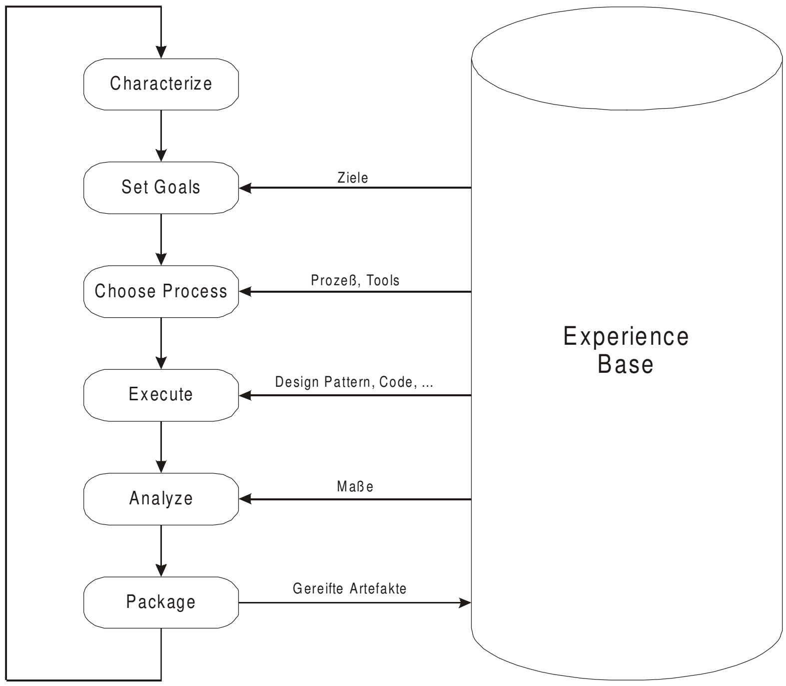

Abbildung 1: Das Quality Improvement Paradigm

Characterize: Dieser Schritt definiert die Umgebung des Softwareprozesses.

Set Goals: An Hand der festgelegten Umgebung werden konkrete Ziele festgelegt, die erfüllt sein müssen, damit der Prozess erfolgreich ist.

Choose Process: Auf Grund der Umgebung und der festgelegten Ziele wird ein geeigneter Prozess mit dazugehörigen Tools aus der Experience Base ausgewählt.

Execute: In dieser Phase wird der Softwareprozess durchgeführt. Dabei wird der Prozess durch Daten aus der Experience Base unterstützt und gesteuert.

Analyze: Diese Phase tritt nach Beendigung der Software Entwicklung in Kraft. Die während der Entwicklung angefallen Daten werden analysiert, um im nächsten Schritt als *lessons learned* in die Experience Base aufgenommen werden zu können.

Package: In dieser Phase werden alle während des Prozesses weiterentwickelten oder neu entstandenen Artefakte für eine spätere Wiederverwendung aufbereitet und in die Experience Base aufgenommen.

Die enge Zusammenarbeit mit der Experience Base stellt starke Anforderungen an das zu Grunde liegende Datenbanksystem: es müssen verschiedene Arten von Artefakten speicherbar sein, die Artefakte müssen in verschiedenen Versionen vorliegen, der Zusammenhang der Artefakte zueinander muss modellierbar sein und es muss eine intelligente Suchfunktion vorhanden sein, die zu jedem Schritt geeignete Artefakte auswählt.

Viele der bisher existierenden Systeme schrecken vor diesen Anforderungen zurück und implementieren nur eine Teillösung, indem sie zum Beispiel nur bestimmte Artefakte zulassen. Das folgende Kapitel untersucht detailliert die Eigenschaften, die ein *Software Reuse System* klassifizieren und stellt Maße vor, die helfen, die Leistungsfähigkeit der Systeme zu beurteilen.

3. Möglichkeiten zur Beurteilung von Software Reuse Systemen

Das Ziel eines Software Reuse Systems ist es nach [MMM98], bestimmte Artefakte des Softwareentwicklungsprozesses so zu speichern, dass sie später in geeigneter Weise wiederverwendet werden können. Diese Definition ist sicherlich korrekt, aber so allgemein, dass es zusätzlicher Klärung bedarf. Zunächst stellt sich die Frage: was bedeutet Wiederverwendung in geeigneter Weise? Man unterscheidet zwei unterschiedliche Arten: Wiederverwendung durch *Modifikation* und Wiederverwendung durch *Komposition.* Modifikation bedeutet, es wird ein Artefakt gesucht, das mit möglichst wenig Aufwand so verändert werden kann, dass es den Bedürfnissen entspricht. Bei Komposition versucht man, mehrere kleinere Komponenten möglichst unverändert zu lassen und zu einen neuen System zu kombinieren. Je nachdem, welche Strategie verfolgt wird, sind verschiedene Artefakte geeignet.

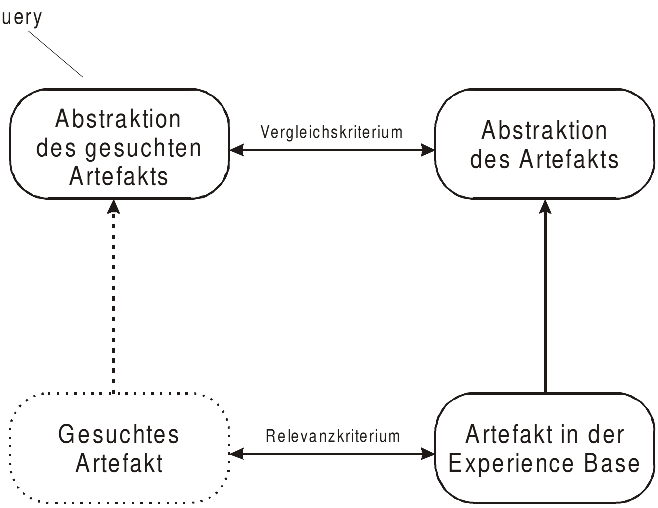

Abbildung 2: Relevanz- und Vergleichskriterium

Abbildung 2 illustriert die prinzipielle Arbeitsweise eines Software Reuse Systems. Der Benutzer erwartet als Ergebnis ein bestimmtes Artefakt oder auch eine Menge von Artefakten, die er zur Wiederverwendung verwenden möchte. Da dieses gesuchte Artefakt dem Benutzer natürlich nicht vorliegt, muss eine geeignete Abstraktion des gesuchten Artefakts erstellt werden, die dessen wichtigste Eigenschaften beschreibt. Diese Abstraktion dient als Query für das Software Reuse System. Zu allen in der Experience Base gespeicherten Artefakten existiert eine analoge Abstraktion. Ein Software Reuse System vergleicht nun diese Abstraktionen der gespeicherten Artefakte mit der Abstraktion des Zielartefakts, um zu bestimmen, welche Artefakte zurückgeliefert werden. Formal betrachtet existieren zwei unterschiedliche Kriterien: das *Vergleichskriterium* und das *Relevanzkriterium.* Das Relevanzkriterium bestimmt, ob ein Artefakt zur Wiederverwendung geeignet ist. D.h. erwünscht sind Systeme, die nur Artefakte zurückliefern, für die das Relevanzkriterium erfüllt ist. Allerdings lässt sich das Relevanzkriterium oftmals nicht berechnen. Wesentlich einfacher auszurechnen ist das Vergleichskriterium. Das Vergleichskriterium ist eine Funktion zwischen den Abstraktionen der gespeicherten Artefakte und der Abstraktion des gesuchten Artefakts. Da die Abstraktionen normalerweise wesentlich einfacher als die Artefakte sind, lässt sich auf ihnen besser rechnen. Man versucht nun das Vergleichskriterium so zu wählen, dass es das Relevanzkriterium impliziert. D.h. immer wenn die Abstraktion eines Artefaktes das Vergleichskriterium erfüllt, erfüllt das Artefakt selbst das Relevanzkriterium. Es genügt also, die Abstraktionen der Artefakte zu betrachten und das leichter bestimmbare Vergleichskriterium auszurechnen, um die relevanten Artefakte bestimmen zu können. Leider ist es sehr schwierig eine geeignete Abstraktion und ein geeignetes Vergleichkriterium zu finden. Deswegen sind meistens nur Näherungslösungen möglich, die nicht alle relevanten Artefakten finden oder einige Artefakte anzeigen, die nicht relevant sind.

Um eine Beurteilung und Klassifizierung der Systeme zu ermöglichen, werden im Folgenden verschiedene Eigenschaften eines Software Reuse Systems definiert und erläutert. An Hand dieser Eigenschaften lassen sich alle Systeme in sechs unterschiedliche Klassen einteilen. Zusätzlich dazu werden verschiedene Maße vorgestellt, die die Qualität eines Systems beurteilen. Dabei unterscheiden sich die einzelnen Systeme einer bestimmten Klasse in diesen Maßen zumeist nur gering. Deswegen ist es möglich, eine Gesamtbewertung einer ganzen Klasse von Systemen anzugeben. Im Folgenden werden alle sechs Klassen vorgestellt, indem zu jeder Klasse ein Beispielsysteme angegeben wird und eine Bewertung der Klasse an Hand der vorgestellten Maße vorgenommen wird.

3.1. Eigenschaften

Nach [MMM98] lassen sich die folgenden Eigenschaften eines Software Reuse Systems definieren:

Relevanzkriterium: Das Relevanzkriterium definiert, wann ein Artefakt geeignet zur Wiederverwendung ist. Auch wenn sich dieses Kriterium nicht durch eine geeignete Funktion ausdrücken lässt, ist es dennoch wichtig die Anforderungen an ein geeignetes Artefakt zu definieren. Möglich wäre zum Beispiel: ein Artefakt ist genau dann relevant, wenn es ohne Veränderung direkt wiederverwendet werden kann. Oder: ein Artefakt ist genau dann relevant, wenn es mit möglichst wenig Aufwand so verändert werden kann, dass es wiederverwendbar ist. Oder: ein Artefakt ist genau dann relevant, wenn es mit anderen Artefakten so kombiniert werden kann, dass eine Wiederverwendung möglich ist. Nur wenn das Relevanzkriterium eindeutig definiert ist, lässt sich die Qualität eines Systems beurteilen.

Vergleichskriterium: Das Vergleichskriterium ist eine berechenbare Funktion, die bestimmt, welche Artefakte das System zurückliefert. Das Vergleichskriterium arbeitet normalerweise nicht direkt auf auf den Artefakten sondern auf geeigneten Abstraktionen. Idealerweise impliziert das Vergleichskriterium das Relevanzkriterium. Normalerweise wird man das Vergleichskriterium aber schwächer wählen, so dass auch einige nicht relevante Artefakte gefunden werden.

Art der Artefakte: Das ist die wohl wichtigste Eingeschaft eines Software Reuse Systems. Damit wird definiert, was sich in dem Systems speichern läßt. Das kann z.B. ausführbarer Code, ein Designdokument oder auch ein Softwareprozess sein. Die meisten Systeme speichern nur Artefakte einer bestimmten Art, aber möglich und wünschenswert ist es natürlich, verschiedene Artefakte in einem gemeinsamen System zu speichern.

Abstraktion der Artefakte: Diese Eigenschaft beschreibt, welche Abstraktion verwendet wird, um die Artefakte zu beschreiben. Das kann z.B. ein Klassifizierungsvektor mit bestimmten Eigenschaften sein oder auch eine formale Spezifikation. Diese Eigenschaft bestimmt auch, in welcher Form Anfragen an das System zu stellen sind.

Struktur der Daten: Diese Eigenschaft beschreibt die interne Struktur der Experience Base. Darunter fällt das Speichermedium (z.B. relationale Datenbank), die Anordnung der Artefakte (z.B. sortiert nach einem bestimmten Index) und die Verknüpfungen zwischen den Artefakten (z.B. Verbindung durch semantische Links).

3.2. Maße

Zur besseren Beurteilung der Systeme ist es nötig, Maße zu definieren, die die Systeme beschreiben. Oftmals ist eine quantitative Aussage nicht möglich, dann wird ein qualitatives Maß von „sehr gering“ bis „sehr hoch“ verwendet. Es sind zwei Gruppen von Maßen erkennbar: technische und verwaltende. Die technischen Maße treffen Aussagen über die Qualität der gefundenen Artefakte, sowie die Geschwindigkeit des Suchvorgangs. Die verwaltenden Maße geben an, wie viel Aufwand nötig ist, die Bibliothek in Betrieb zu nehmen und zu warten und wie gut die Bibliothek zu benutzen ist.

3.2.1. Technische Maße

Präzision: Präzision ist definiert als relevante Artefakte pro insgesamt gefundener Artefakte. Eine Präzision von 1.0 bedeutet, dass alle gefundenen Artefakte relevant sind. Dabei ist allerdings zu beachten, dass eine Präzision von 1.0 auch erreicht werden kann, wenn gar keine Artefakte gefunden werden.

Recall: Recall ist definiert als Quotient der gefundenen relevanten Artefakte pro insgesamt vorhandenen relevanten Artefakte. Ein Recall von 1.0 bedeutet, dass alle relevanten Artefakte gefunden werden. Natürlich kann ein Recall von 1.0 einfach dadurch erreicht werden, dass alle Artefakte der Bibliothek zurückgeliefert werden. Nützlich ist also ein gleichermaßen hoher Wert von Präzision und Recall.

Abdeckung: Die Abdeckung bezeichnet, welcher Bruchteil der insgesamt vorhandenen Artefakte bei einer Abfrage angefasst wird. Bei einer einfachen linearen Suche werden alle Artefakte betrachtet und es ergibt sich eine sehr hohe Abdeckung. Bei einer strukturierteren Anordnung der Daten sind intelligentere Suchalgorithmen möglich, mit denen eine geringere Abdeckung erreicht wird.

Zeitkomplexität pro Vergleich: Damit wird gemessen, wie aufwändig der Vergleich eines einzelnen Artefakts mit der Suchanfrage ist. Aus dem Produkt der Abdeckung und der Zeit pro Vergleich lässt sich die Zeit pro Suchanfrage abschätzen.

3.2.2. Verwaltende Maße

Aufwand bei der Einführung: Damit wird abgeschätzt, wieviel Aufwand (und damit welche Kosten) bei der Einführung der Bibliothek nötig sind. Dabei geht man davon aus, dass bereits eine gewisse Zahl an Artefakten vorhanden ist, die zu Beginn in das System eingespeist werden müssen. Manche Systeme setzen z.B. eine aufwändige manuelle Klassifizierung der Artefakte voraus, was einen hohen initialen Aufwand bewirkt.

Laufende Kosten: Die laufenden Kosten bezeichnen den relativen Aufwand in Bezug zur Bibliotheksgröße der nötig ist, um die Bibliothek in Betrieb zu halten und neue Artefakte aufzunehmen.

Benutzbarkeit: Dieser Faktor stellt fest, wie gut man mit den vorhandenen Produkten arbeiten kann. Dabei fließt sowohl die Benutzeroberfläche ein, als auch die Transparenz, d.h. wieviel Einblick in die interne Struktur des Systems nötig ist, um damit arbeiten zu können.

Diese verwaltendes Maße spielen ein große Rolle, wenn es darum geht, ein System für einen bestimmten Zweck auszuwählen. Dennoch tragen sie nur wenig zum prinzipiellen Verständnis der verschiedenen Klassen von Software Reuse Systemen bei. Deswegen werden wir uns bei der Diskussion der einzelnen Klassen auf die technischen Maße beschränken.

4. Die sechs Klassen von Software Reuse Systemen

4.1. Information Retrieval

Die meisten Artefakte eines Softwareprozesses, können im Grunde auch als Textdokumente betrachtet werden. Deshalb ist ein Software Reuse System im Prinzip nur eine spezialisierte Variante eines Information Retrieval Systems. Die wohl bekannteste Variante von Information Retrieval Systemen sind Suchmaschinen im Internet. Dokumente werden anhand der enthaltenen Schlüsselworte klassifiziert und eine Anfrage besteht aus einem Suchbegriff, oder einem Booleschen Ausdruck. Diese Methode lässt sich auch auf Software Komponenten anwenden.

4.1.1. Beispiel

Eine relativ einfache Anwendung von Information Retrieval Techniken wird von Frakes und Nejmeh in [FN87] angewendet. Das von ihnen entworfene System speichert C-Quellcode. An Hand der Header der Quelltextdateien werden die charakteristischen Attribute wie Autor, Erstellungsdatum und eine kurze Beschreibung in natürlicher Sprache extrahiert. Dies setzt voraus, dass diese Informationen in jedem Header vorkommen. Eine Suchanfrage besteht aus Worten, die in den Attributen vorkommen. Zum Beispiel: Autor = "Hans Meier", Funktion = "Dateidialog".

4.1.2. Beschreibung:

Art der Artefakte: Ein Information Retrieval System kann mit allem umgehen, was aus natürlicher Sprache besteht, oder zu dem eine Beschreibung in natürlicher Sprache existiert. Da diese Methoden nicht auf das Gebiet der Softwareentwicklung spezialisiert sind, können alle Artefakte gespeichert werden, die in textueller Form vorliegen.

Abstraktion der Artefakte: Eine explizite Abstraktion ist nicht nötig. Es wird direkt auf dem Artefakt, oder auf den Teilen des Artefakts, die aus natürlicher Sprache bestehen, gearbeitet. Eine Anfrage besteht aus bestimmten Schlüsselworten, die im Artefakt gesucht werden.

Relevanz- und Vergleichskriterium: Da keine explizite Abstraktion der Artefakte existiert, fallen Relevanz- und Vergleichskriterium zusammen. Ein Artefakt wird genau dann ausgewählt, wenn es alle in der Anfrage auftretenden Schlüsselworte enthält.

Struktur der Daten: Im einfachsten Fall ist keinerlei Struktur vorhanden. Oftmals existieren allerdings Verknüpfungen zwischen den Artefakten (*implemtiert, ruftAuf, ...*) die ausgenutzt werden können (Vergleiche Links bei Hypertext Dokumenten).

4.1.3. Beurteilung:

Präzision und Recall: Durch die relativ schwache Strukturierung, lässt sich keine große Genauigkeit erreichen. Die Präzision wird mit *mäßig* bewertet. Deshalb geben die meisten Information Retrieval Systeme eine relativ hohe Zahl an nicht relevanten Artefakten aus, um einen *hohen* Recall zu erreichen.

Abdeckung: Die meisten weiterentwickelten Systeme durchlaufen die Artefakte an Hand der angegeben Verknüpfungen. Deshalb muss nur ein kleiner Teil der gespeicherten Artefakte überprüft werden. Die Abdeckung wird mit *gering* bewertet.

Aufwand pro Vergleich: Ein Vergleich zwischen einem Artefakt und einer Anfrage ist trivial. Es muss überprüft werden, ob das Artefakt die erforderlichen Schlüsselworte enthält. Deshalb der Aufwand pro Vergleich *gering*.

4.2. Deskriptive Methoden

Deskriptive Methoden beschreiben eine etwas andere Form des Information Retrieval. Der Hauptunterschied zu den zuvor beschriebenen Methoden ist, dass zu jedem Artefakt eine explizite, normalerweise manuell erstellte Beschreibung existiert. Diese Beschreibung besteht aus einer Menge von Schlüsselworten, die das Artefakt hoffentlich umfassend und weitestgehend ohne Redundanz beschreiben. Eine Anfrage basiert auf Schlüsselworten, die mit der Beschreibung der Artefakte verglichen werden.

4.2.1. Beispiel

Ein Beispiel für deskriptive Systeme ist der von Prieto-Diaz in [PD85] entwickelte Facettenansatz. Dabei wird zu jedem Artefakt eine Beschreibung in verschiedenen Facetten erstellt. In jeder dieser Facetten ist eine vorher definierte Menge an Schlüsselworten gültig. Dadurch wird ein mehrdimensionaler Suchraum definiert, der in der Mächtigkeit weit über die unstrukturierte Schlüsselwortsuche hinausgeht. Mögliche Facetten für ein Codefragment sind z.B. Programmiersprache, Autor, Funktionalität und Betriebssystem. Das kontrollierte Vokabular in den einzelnen Facetten garantiert eine einheitliche Beschreibung.

4.2.2. Beschreibung

Speicherbare Artefakte: Deskriptive Methoden sind am unfassendsten einsetzbar. Jedes beliebige Artefakt lässt sich mit Hilfe von Schlüsselworten beschreiben, und kann deshalb im System verwendet werden. Außerdem ist eine Mischform von verschiedensten Artefakten im selben System problemlos denkbar.

Art der Abstraktion: Zur Beschreibung der Artefakte dient meist eine Vektor von Schlüsselworten. Dabei stammen die Schlüsselworte im Allgemeinem aus einem zuvor definierten Wortschatz.

Relevanz- und Vergleichskriterium: Ein Artfakt ist relevant, wenn es sich mit demselben Thema wie die Anfrage befasst. Das Thema wird durch den Vektor von Schlüsselworten beschrieben. Deswegen erfüllt ein Artefakt das Vergleichskriterium genau dann, wenn alle Schlüsselworte der Anfrage in der Beschreibung des Artefakts vorkommen.

Struktur der Daten: Struktur spielt bei deskriptiven Methoden meist eine untergeordnete Rolle. Normalerweise liegen die Artefakte ungeordnet und ohne modellierte Beziehung zueinander in der Datenbank.

4.2.3. Beurteilung

Präzision und Recall: Ausschlaggebend für gute Präzision und Recall ist, wie gut die Schlüsselworte die Artefakte tatsächlich beschreiben und wie einheitlich die Beschreibung durch Schlüsselworte erfolgt ist. Durch eine kontrollierte Beschreibung, die genau festlegt, welche Facetten und welches Vokabular zur Beschreibung der Artefakte verwendet werden, lässt sich dieses Problem gut in den Griff bekommen. Deshalb werden Präzision und Recall gleichermaßen mit *hoch* bewertet.

Abdeckung: Da die Daten in keiner Weise strukturiert sind, ist eine lineare Suche in der Datenbank nötig. Dabei ist auch keinerlei Ordnung, zwischen den Artefakten erkennbar. Deswegen werden bei einer Anfrage alle Artefakte besucht. Die Abdeckung ist also *sehr hoch.*

Aufwand pro Vergleich: Es werden Vektoren von Schlüsselworten miteinander verglichen. Dabei existieren diese beschreibenden Vektoren bereits, und müssen nicht wie bei den Methoden des allgemeinen Information Retrievals zur Laufzeit erzeugt werden. Deshalb ist der Aufwand pro Vergleich *sehr gering.*

4.3. Operationale Semantik

Die Methoden dieser Klasse beschränken sich auf ausführbare Artefakte. Also im Allgemeinen auf einzelne Funktionen. Eine Anfrage besteht aus einer Menge von Beispieleingaben mit den dazugehörigen erwünschten Ergebnissen. Während des Retrievals werden die Funktionen mit den übergebenen Eingabewerten ausgeführt, und diejenigen ausgewählt, die die gewünschten Ergebnisse liefern.

4.3.1. Beispiel

Eines der ersten auf operationaler Semantik basierenden Systeme wurde von Podgurski und Pierce in [PP92] entwickelt. Die Artefakte des Systems werden durch ihren ausführbaren Code definiert. Zusätzlich dazu enthält jedes Artefakt eine Beschreibung des Ein- und Ausgaberaums, sowie eine Wahrscheinlichkeitsverteilung über den Eingaberaum, die die Wahrscheinlichkeit des Auftretens eines bestimmten Eingabewerts während einer typischen Ausführung beschreibt. Eine Anfrage an das System besteht aus der Beschreibung des erwünschten Eingaberaums, sowie einer Orakelfunktion. Diese Orakelfunktion kann für je einen Ein- und Ausgabewert entscheiden, ob es sich um ein gültiges Tupel handelt. Im Retrievalprozess wählt das System basierend auf der Wahrscheinlichkeitsverteilung Werte aus dem Eingaberaum aus und schickt sie an die ausführbare Komponente des Artefakts. Der zurückgelieferte Ausgabewert wird zur Überprüfung an die Orakelfunktion gegeben. Zurückgeliefert werden alle Artefakte, die den gewünschten Eingaberaum haben und auf den zufällig ausgewählten Beispielen gültige Ausgabewerte geliefert haben.

4.3.2. Beschreibung

Speicherbare Artefakte: Da sich die Methoden der Operationalen Semantik die Ausführbarkeit der Artefakte explizit zu Nutzen machen, können nur ausführbare Artefakte gespeichert werden. Dabei ist die Komplexität der Artefakte nicht eingeschränkt. Es können also einzelne Methoden oder komplette Programmbibliotheken verwaltet werden, solange sich ein explizites Ein- Ausgabeverhalten feststellen lässt.

Beschreibung der Artefakte: Es gibt keine zusätzliche Beschreibung der Artefakte. Die Artefakte müssen in ausführbarer Form vorliegen, und vom System zum Retrieval mit den gewünschten Eingabewerten gestartet werden können. Deswegen sind die Programmteile normalerweise in Treiber eingebettet, die die Eingaben annehmen und die Ausgaben an das Software Reuse System zurückliefern.

Relevanz- und Vergleichskriterium: Ein Artefakt ist relevant, wenn es das erwünschte Ein-Ausgabeverhalten zeigt. Da dieses Verhalten nicht für beliebig viele Eingabedaten überprüft werden kann, wird eine Auswahl von Tupeln getroffen. Ein Artefakt erfüllt genau dann das Vergleichskriterium, wenn es für die ausgewählten Eingabewerte die korrekte Ausgabe liefert.

Struktur der Daten: Eine wichtige strukturelle Eigenschaft ist die Signatur der gespeicherten Funktionen. Also welche und wieviele Eingabeparameter erwartet werden, und von welchen Datentyp die Ausgabe ist. Nur Artefakte mit passender Signatur können mit der Anfrage verglichen werden.

4.3.3. Beurteilung

Präzision und Recall: Die Präzision hängt stark von der Anzahl der verwendeten Eingabewerte ab. Je mehr Werte überprüft werden, desto höher wird die Präzision. Da die Eingabewerte normalerweise zufällig ausgewählt werden, können beliebig viele Eingabewerte überprüft, und so eine *sehr hohe* Präzision erreicht werden. Wenn ein Artefakt das gewünschte Ein- Ausgabeverhalten hat, dann zeigt es das insbesondere auch in der ausgewählten Menge von Eingabedaten. D.h. alle relevanten Artefakte werden auch tatsächlich gefunden. In der Praxis ist das leider nicht so. Denn in einem ersten Schritt müssen zuerst die Artefakte mit passender Signatur ausgewählt werden. Da diese Signatur normalerweise sehr komplex ist, insbesondere bei stark typisierten oder objektorientierten Sprachen, kann es vorkommen, dass relevante Artefakte in diesem Schritt aussortiert werden. Deshalb ist der Recall nur *hoch.*

Abdeckung: Im ersten Schritt müssen alle Artefakte mit passender Signatur gefunden werden. Falls die Datenbank nicht nach der Signatur sortiert ist, müssen dazu alle Artefakte betrachtet werden. Da diese Sortierung nach der Signatur im Allgemeinen sehr komplex ist, ist dies fast nie der Fall. Deswegen werden zunächst alle Artefakte betrachtet. Der wesentlich zeitaufwändigere Prozess der Überprüfung des Ein- Ausgabeverhaltens wird nur auf die Artefakte mit passender Signatur angewandt, welche einen Bruchteil der insgesamt gespeicherten Artefakte ausmachen. Im Vergleich zu anderen Systemen ergibt sich eine *hohe* Abdeckung.

Aufwand pro Vergleich: Der erste Schritt, der Vergleich der Signaturen, ist bei typisierten Sprachen zwar nicht trivial aber auch nicht sehr komplex. Weitaus aufwändiger und kaum kontrollierbar ist die Überprüfung des Ein- Ausgabeverhaltens. Es kann insbesondere vorkommen, dass ein Artefakt für eine bestimmte Eingabe in eine Endlosschleife gerät. Da das Verhalten sehr stark von den gespeicherten Artefakten abhängt, wird der Aufwand im Mittel als *mäßig* eingestuft.

4.4. Denationale Semantik

Diese Klasse beschränkt sich auf Artefakte, deren Funktionalität vollständig beschrieben werden kann. Diese Beschreibung besteht aus einer *Signatur*, die die Syntax der Ein- und Ausgabewerte festlegt, sowie einem berechenbaren Prädikat, das die Beziehung zwischen Ein- und Ausgabewerten vollständig beschreibt. Eine Anfrage an dass System besteht aus der vollständigen Beschreibung der gewünschten Funktionalität bestehend aus Signatur und Prädikat. Es werden jetzt solche Artefakte ausgewählt, die die gewünschte Signatur haben, und deren Funktionalität das gewünschte Prädikat erfüllt. Das ist immer dann der Fall, wenn das Prädikat des Artefakts in das Prädikat der Anfrage transformiert werden kann. Diese Transformation ist im Allgemeinen nicht trivial und bedient sich den Methoden des automatischen Beweisens.

4.4.1. Beispiel

Das System von Moineau und Gaudel [MG91] beinhaltet eine Bibliothek aus Artefakten, die formale Datentypen implementieren. Diese Art von Artefakten eignet sich besonders gut für denationale Semantik. Zu jedem Artefakt ist die Signatur sowie die Menge von Axiomen gespeichert, die den Datentyp beschreiben. Eine Anfrage an das System besteht aus der gewünschten Signatur, sowie den geforderten Axiomen. Das System liefert dann alle Artefakte mit korrekter Signatur zurück, die die Axiome erfüllen.

4.4.2. Beschreibung

Speicherbare Artefakte: Wie bei den Methoden der operationalen Semantik, beschränkt sich die Anwendbarkeit hauptsächlich auf ausführbare Artefakte. Theoretisch sind auch nicht ausführbare Artefakte denkbar, deren Verhalten funktional beschreibbar ist.

Beschreibung der Artefakte: Zu jedem Artefakt wird die Signatur und ein Prädikat gespeichert, das die Funktionalität des Artefakts beschreibt. Das Prädikat ist dabei meistens durch eine Menge von Axiomen definiert.

Relevanz- und Vergleichskriterium: Ein Artefakt ist relevant bezüglich einer Query, wenn es eine geeignete Signatur besitzt und die in der Query geforderten Eigenschaften hat. Daraus ergibt sich das Vergleichskriterium: Ein Artefakt erfüllt genau dann das Vergleichskriterium, wenn die in der Query geforderten Aussagen mit Hilfe der Axiome des Artefakts bewiesen werden können.

4.4.3. Beurteilung

Präzision und Recall: Die Präzision ist theoretisch immer eins, da für jede gefundene Komponente bewiesen wurde, dass sie die gewünschte Funktionalität zeigt. In der Praxis sind gewisse Abstriche zu machen, dennoch ergibt sich eine *sehr hohe* Präzision. Analog zu den Methoden der operationalen Semantik können sich auch hier Probleme bei der Suche nach einer geeigneten Signatur ergeben und deshalb ist es möglich, dass relevante Artefakte nicht gefunden werden. Deshalb wird der Recall mit *hoch* bewertet.

Abdeckung: Analog zur operationalen Semantik werden zunächst alle Artefakte besucht, um diejenigen mit geeigneter Signatur zu finden, die dann vom automatischen Beweiser abgearbeitet werden. Es ergibt sich eine *hohe* Abdeckung.

Aufwand pro Vergleich: Der Aufwand ist immens. Zunächst müssen Artefakte mit passender Signatur gefunden werden, was bei Typhierarchien kein triviales Problem ist, und anschließend wird pro Vergleich ein kompletter Beweis geführt. Dabei ist es bei einer unglücklichen Kombination aus Zielen und Axiomen durchaus denkbar, dass der Beweiser nicht anhält. Der Aufwand pro Vergleich ist *sehr hoch.*

4.5. Topologische Methoden

Topologische Methoden versuchen immer den nächsten Nachbarn zu dem gewünschten Artefakt zu finden. Eine Anfrage besteht aus einer Beschreibung des gesuchten Artefakts. Es existiert ein *Distanzmaß*, das zu jedem Artefakt die Distanz zur Anfrage bestimmen kann. Ausgewählt werden entweder diejenigen Artefakte, die die Anfrage direkt erfüllen, also eine Distanz von Null haben, oder diejenigen Artefakte, deren Distanz zur Anfrage einen bestimmten Schwellwert unterschreiten. Der kritische Faktor ist also die Wahl eines passenden Distanzmaßes. Dabei unterscheidet man prinzipiell zwischen *funktionaler Distanz* und *struktureller Distanz.* Die funktionale Distanz beschreibt, wie ähnlich sich zwei Artefakte verhalten, die strukturelle Distanz beschreibt, wie ähnlich zwei Artefakte aussehen. Die funktionale Distanz ist wichtiger, wenn die Artefakte ohne Anpassung wiederverwendet werden sollen, die strukturelle Distanz ist wichtiger, wenn die Artefakte nach dem Retrieval angepasst werden sollen.

4.5.1. Beispiel

Ein gutes Beispiel ist das System von Spanoudakis und Constantopoulos [SC93]. Das System speichert objektorientierte Artefakte. Als Beschreibung dient eine spezielle Sprache, die die Eigenschaften von objektorientierten Systemen beschreibt. Diese Sprache kennt Konstrukte wie Klasse, Instanz, Methode und Attribut. Eine Anfrage besteht aus der gewünschten Struktur. Das Ähnlichkeitsmaß berechnet die strukturelle Distanz zwischen zwei Artefakten.

4.5.2. Beschreibung

Speicherbare Artefakte: Topologische Methoden unterliegen keinerlei Einschränkung bei der Auswahl der Artefakte.

Beschreibung der Artefakte: Um die Distanz zwischen zwei Artefakten bestimmen zu können, ist eine hinreichend einfache Beschreibung der Artefakte nötig. Das kann entweder ein Vektor beschreibender Schlüsselworte, oder eine objektorientierte Beschreibung sein. Dabei ist sowohl eine automatisch generierte, als auch eine manuell erstellte Beschreibung denkbar.

Relevanz und Vergleichskriterium: Das Relevanzkriterium versucht, das Artefakt auszuwählen, das der Anfrage am nächsten ist. Deshalb lässt sich für ein einzelnes Artefakt nicht entscheiden, ob es relevant ist oder nicht. Es müssen potentiell immer alle Artefakte betrachtet werden. Ein Artefakt ist am Ähnlichsten, wenn es die geringste Distanz aller Artefakte zur Anfrage hat. Daraus ergibt sich ein mögliches Vergleichskriterium: es werden alle Artefakte zurückgeliefert, deren Distanzmaß einen bestimmten Schwellenwert unterschreitet.

Struktur der Daten: Eigentlich spielt die Struktur der Daten keine große Rolle. Dennoch gibt es verschiedene Ansätze aus der KI, die ein Suchen nach dem ähnlichsten Artefakt beschleunigen.

4.5.3. Beurteilung

Präzision und Recall: Bei diesem nicht exakten Ansatz ist es schwierig von Präzision und Recall zu sprechen. Zu einer Anfrage werden garantiert die *n* Artefakte mit der ähnlichsten Beschreibung gefunden. Wie gut geeignet diese Artefakte sind, hängt entscheidend von der Wahl der Beschreibung und noch mehr von der Wahl des Distanzmaßes ab. Eine Möglichkeit ist beispielsweise einfach alle Artefakte sortiert nach ihrer Ähnlichkeit auszugeben. Das würde nach Definition eine Präzision von Null und einen Recall von eins bedingen. Dennoch ist das Ergebnis sinnvoll, da man die gefundenen Artefakte in der Reihenfolge ihrer Ähnlichkeit betrachten kann. Deshalb wird die Präzision mit *hoch* und der Recall mit *sehr hoch* bewertet.

Abdeckung: Wenn keine besondere Struktur in der Datenbasis vorhanden ist, müssen alle Artefakte besucht werden, um das Ähnlichste zu bestimmen. Durch geschickte Anordnung der Datenbasis, lässt sich die Anzahl der Vergleiche auf log n reduzieren. Die Abdeckung ist also mäßig.

Aufwand pro Vergleich: Das Distanzmaß ist meistens eine Akkumulationsfunktion, die die Ergebnisse der Teildistanzen zusammenfasst. Dabei sind diese Teilfunktionen zumeist einfache numerische Funktionen. Es ergibt sich also ein *mäßiger* Aufwand.

4.6. Strukturelle Methoden

Strukturelle Methoden beurteilen Artefakte allein auf Grund ihrer Struktur. Die funktionale Beschreibung bleibt völlig im Hintergrund. Oftmals werden in solchen Bibliotheken keine fertigen Programmteile, sondern nur Entwürfe (*Design Patterns*) gespeichert, die dann entsprechend instantiiert werden. Voraussetzung für diese Vorgehensweise ist, dass die Anfrage hinreichend abstrakt gestellt wird, und dass in der Bibliothek nur Artefakte abgelegt werden, die flexibel einsetzbar sind. Ein einfaches Beispiel ist eine Bibliothek von Design Patterns, wie in [GHJV95] beschrieben. Der Retrievalalgorithmus ist denkbar einfach: Design Patterns werden an Hand eines eindeutigen Namens identifiziert. Dabei sind diese Design Patterns sehr universell einsetzbar, aber noch sehr abstrakt. In einem elektronischen System ist eine Zwischenstufe denkbar, bei der schon eine gewisse Rahmenfunktionalität existiert.

4.6.1. Beispiel

Ein Beispiel ist das von Paul und Prakash in [PP94] erläuterte System. Das System speichert Quellcodedateiein, von denen automatisch eine strukturelle Beschreibung in einer abstrakten Sprache extrahiert wird. Eine Anfrage an das System definiert dann die gewünschte Struktur. Zurückgeliefert werden alle Systeme, deren Struktur in das gegebene Schema passt.

4.6.2. Beschreibung

Art der Artefakte: Die in einer solchen Bibliothek gespeicherten Artefakte sind entweder allgemein einsetzbare Programmteile, oder Designinformationen im Sinne von Design Patterns. Theoretisch können mit strukturellen Methoden beliebige Artefakte verarbeitet werden.

Art der Abstraktion: Die Beschreibung der Artefakte ist denkbar simpel, im extremen Fall einfach der Name des Entwurfsmusters. Andere Systeme, wie [PP94] extrahieren automatisch Strukturinformationen aus Quellcode.

Relevanz- und Vergleichskriterium: Als Anfrage wird eine bestimmte abstrakte Architektur eingegeben. Relevant sind diejenigen Artefakte, deren Architektur mit der der Anfrage übereinstimmen. Das Vergleichskriterium implementiert direkt das Relevanzkriterium oder liefert Artefakte mit korrektem Namen im Falle einer Bibliothek von Design Patterns.

Struktur der Daten: Bei strukturellen Methoden werden die Artefakte meist in ungeordneter Form gespeichert.

4.6.3. Beurteilung

Präzision und Recall: Im einfachsten Fall, dass Entwurfsmuster direkt über ihren eindeutigen Namen bestimmt werden, sind Präzision und Recall Eins. Das setzt allerdings voraus, dass der Name allgemein bekannt ist. Auch bei einer Suche über strukturelle Beschreibung lässt sich eine hohe Qualität erreichen. Es ergibt sich ein *sehr hoher* Recall und eine *sehr hohe* Präzision.

Abdeckung: Da die Daten in keiner weise strukturiert sind, ist eine *sehr hohe* Abdeckung der Datenbasis nötig.

Aufwand pro Vergleich: Der Vergleichsaufwand ist *gering*. Im trivialen Fall ist einfach das Entwurfsmuster mit dem richtigen Namen herauszufinden und auch der Vergleich zweier Strukturbeschreibungen ist wenig komplex.

5. Beispielsysteme

In diesem Kapitel werden zwei konkrete Software Reuse Systeme vorgestellt, die gänzlich unterschiedliche Ansätze verfolgen. Das erste System, CodeFinder, versucht mit einer minimalen Struktur des Repositories und einem geschickten Retrievalsystem, geeignete Ergebnisse zu finden. Die SFB 501 Experience Base setzt auf ein stark strukturiertes Repository mit einem topologischen Retrievalalgorithmus. Im Folgenden wird die Architektur der beiden Systeme vorgestellt, sowie eine Einordnung in die zuvor präsentierten Klassen vorgenommen. Außerdem werden die Vor- und Nachteile der beiden Systeme an Hand der eingeführten Maße diskutiert.

5.1. CodeFinder

Eines der Haupthindernisse zum erfolgreichen Einsatz eines Software Reuse Repositories ist nach [He97] der große anfängliche Aufwand, der nötig ist, um die Datenstrukturen aufzubauen. Doch ohne eine geeignete Datenstruktur sind in großen Bibliotheken bislang nur schlechte Ergebnisse erzielt worden. Deshalb verfolgt Henning den Ansatz, die Strukturen während der Benutzung des Systems in einem Lernprozess aufzubauen. So ist kein großer initialer Aufwand nötig, sondern der Aufwand wird in kleineren Einheiten über die gesamte Laufzeit der Bibliothek verteilt. Das implementierte Beispielsystem arbeitet mit Erweiterungsfunktionen für den EMACS, die in Lisp geschrieben sind. Dabei wird eine Wiederverwendung durch Komposition angestrebt, deswegen werden einzelne Funktionen, Konstanten und Variablen gespeichert. Insgesamt sind in dem System ca. 1800 Artefakte gespeichert.

5.1.1. Arbeitsweise

Um den großen initialen Aufwand zu vermeiden, werden neue Artefakte weitestgehend automatisch klassifiziert. Zu jedem Artefakt wird automatisch eine Abstraktion erstellt, deren wichtigste Attribute *Name, Beschreibung, aufgerufene Funktionen* und *Schlüsselworte* sind. Die Abstraktion wird aus dem Namen und der Beschreibung der Funktion erstellt. Dies setzt eine einheitliche Kommentierung der Funktionen voraus. Allerdings hat der Bibliothekar bei jedem Artefakt die Möglichkeit, die Abstraktion beliebig zu editieren oder um neue Schlüsselworte zu ergänzen.

Durch diese automatische Indizierung werden bei einer standardmäßigen Suche nur sehr schlechte Ergebnisse erzielt. Das liegt daran, dass die beschreibenden Begriffe oftmals keine guten Klassifikatoren sind (d.h. der Begriff beschreibt außer dem gesuchten Artefakt noch viele andere) und dass Synonyme des Suchbegriffs nicht berücksichtigt werden. Um dies zu kompensieren wird ein Netz aufgebaut, das durch "spreading activation" die passendsten Einträge bestimmt.

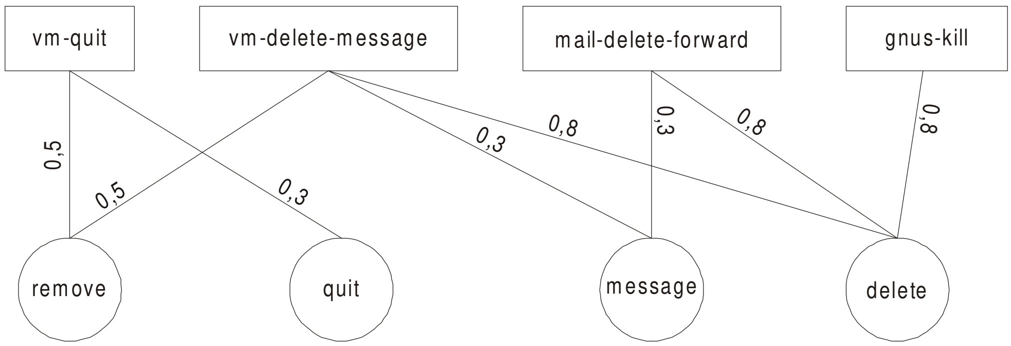

Abbildung 3: Netz mit spreading activation

Abbildung 3 zeigt ein mögliches Netz. Es existieren zwei unterschiedliche Arten von Knoten: Artefakte (Rechtecke in der Abbildung) und Schlüsselworte (Kreise in der Abbildung). An Hand der Beschreibung der Artefakte wird von jedem Artefakt eine Verbindung zu jedem Schlüsselwort gesetzt, das das Artefakt beschreibt. So kommt das Schlüsselwort *remove* beispielsweise in der Beschreibung der Funktionen *vm-quit* und *vm-delete-message* vor. Jeder dieser Verbindungen ist ein Gewicht zugeordnet, das angibt, wie gut die Aktivierung weitergegeben wird. Wenn nun in der Anfrage nach einem Schlüsselwort gesucht wird, so wird die Aktivierung des Knotens auf 1 gesetzt. Im nächsten Schritt verteilt sich diese Aktivierung auf alle Artefakte, die von diesem Schlüsselwort klassifiziert werden. Im Beispiel würde die Aktivierung also an die Artefakte *vm-quit* und *vm-delete-message* weitergegeben. Im darauffolgenden Schritt wird diese Aktivierung von den aktivierten Artefakten zurück an die beschreibenden Schlüsselworte gegeben, so dass auf diese Weise auch Synonyme zu dem Schlüsselwort gefunden werden. Im Beispiel breitet sich die Aktivierung zu den Schlüsselworten *quit, message und delete* aus. Dabei wird die Aktivierung am stärksten an *delete* weitergegeben. Auf diese Weise wird also automatisch das Synonym *delete* in die Suchanfrage aufgenommen. Die Gewichte der Verbindungen sind umso höher, je weniger Artefakte insgesamt durch das Schlüsselwort klassifiziert werden. Ein Schlüsselwort ist ein wesentlich besserer Klassifikator, wenn es genau einem Artefakt zugeordnet ist, als wenn es außerdem noch hundert andere beschreibt. Nach einer bestimmten Anzahl von Iterationen hat sich das Netz hinreichend stabilisiert, so dass man die Artefakte mit der höchsten Aktivierung als Ergebnis des Retrievals bestimmen kann. Auf diese Weise werden fast alle Nachteile der automatischen Indizierung überwunden.

Das System ist so gestaltet, dass der Benutzer nach jeder Anfrage Feedback über die tatsächliche Qualität des Suchergebnisses geben kann. Auf diese Art werden die Gewichte an den Verbindungen im Netz angepasst, so dass sich das System nach und nach an die Begriffe der Benutzer gewöhnt, und bessere Suchergebnisse liefert. Durch die geschickte Integration in die Benutzerführung, geht das Trainieren des neuronalen Netzes fast ohne Mehraufwand von Statten.

5.1.2. Architektur

Abbildung 4 zeigt die Architektur von Codefinder. Die Artefakte sind LISP Quellcodedateien und werden direkt im Filesystem abgelegt. Auf diesen Dateien arbeitet *PEEL*, ein Parser der die initialen Beschreibungen der Artefakte, die *Kandor Objekte,* generiert. Dabei enthält eine Quellcodedatei normalerweise viele Artefakte. Die Beschreibungen der Artefakte werden in einer unstrukturierten Datenbank abgelegt. Aus diesen Beschreibungen werden Termpaare extrahiert, aus denen sich das neuronale Netz anlegen lässt. Das Netz wird intern als Matrix repräsentiert. Beim Retrieval wird ein Eingabevektor, der die initiale Aktivierung der Suchbegriffe beschreibt an das Netz gegeben. Nach einer Stabilisierung des Netzes werden die Knoten mit höchster Aktivierung bestimmt und als Suchergebnis zurückgeliefert.

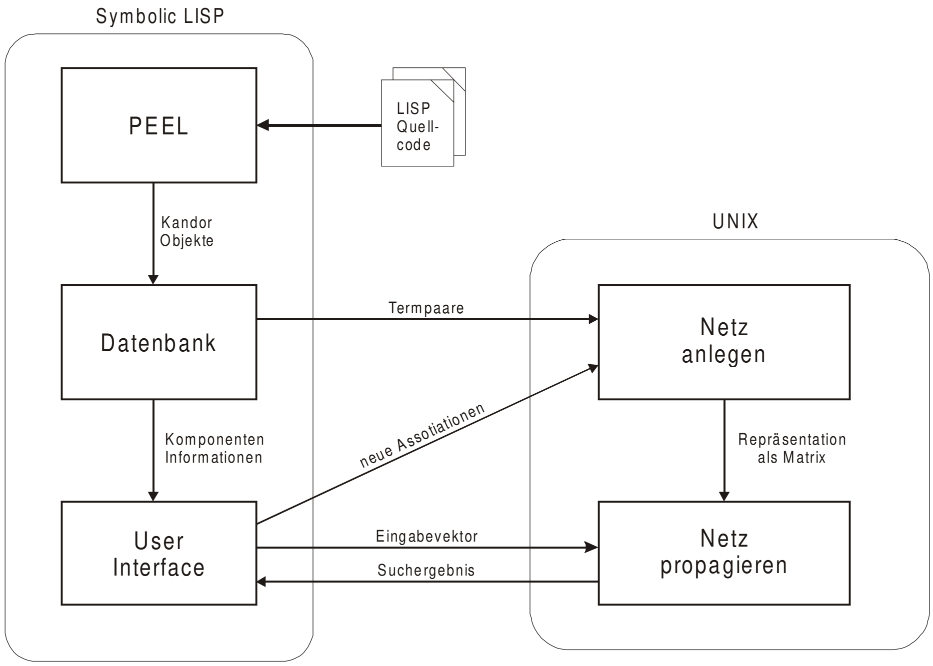

Abbildung 4: Architektur von Codefinder

5.1.3. Beurteilung

Das System lässt sich am besten in die Gruppe der deskriptiven Methoden einordnen. Der Hauptunterschied hierbei ist, dass die Beschreibungen zunächst automatisch generiert werden, und dass das System erst durch Benutzung nach und nach die Begriffe der Benutzer "erlernt". Da in diesem System ein immenser Aufwand betrieben wurde, um die Suchergebnisse zu verbessern, ergibt sich die folgende Bewertung:

Präzision und Recall: Durch das Prinzip der "spreading activation" werden schon von Anfang an relativ gute Werte für Präzision und Recall erreicht. Dabei ist das System so ausgelegt, dass einem höheren Recall auch auf Kosten einer schlechteren Präzision Vorrang gegeben wird. Während das System benutzt wird, übernimmt es die Begriffe der Anwender, so dass sich die Präzision kontinuierlich erhöht. Deshalb lässt sich die Präzision mit *hoch* und der Recall mit *sehr hoch* angeben.

Abdeckung: Das System arbeitet auf einem neuronalen Netz, das aus den Beschreibungen aller Artefakte generiert wurde. Deshalb werden zur Laufzeit gar keine Artefakte besucht. Das Netz wird durch jede Anfrage trainiert und erweitert, wenn neue Artefakte in die Datenbank eingefügt werden. Deshalb ist die Abdeckung *sehr gering.*

Aufwand pro Vergleich: Auch dieser Wert lässt sich nur schwer angeben, da im neuronalen Netz quasi alle Vergleiche parallel ausgeführt werden. Die Propagierung des Netzes ist relativ aufwändig, im Vergleich zu den Methoden der denotationalen Semantik jedoch nur *mäßig.*

Aufwand bei der Einführung: Da die initialen Beschreibungen weitestgehend automatisch generiert werden, ist kaum ein initialer Aufwand nötig. Das ist vor allen Dingen ein Vorteil, wenn bereits viele Artefakte vorhanden sind, die in das Repository aufgenommen werden sollen. Voraussetzung dafür ist allerdings eine einheitliche Kommentierung der Quellcode-Artefakte. Es ergibt sich also ein *geringer* initialer Aufwand.

Laufende Kosten: Um den Lerneffekt des Systems auszunutzen, ist nach jeder Anfrage ein Feedback nötig, das die Qualität der Suchergebnisse bewertet. Dies ist ein kleiner Zusatzaufwand. Allerdings kann dieses Feedback direkt vom Benutzer der Datenbank gegeben werden, so dass kein zusätzlicher Bibliotheksverwalter nötig ist. Die laufenden Kosten sind also alles in allem *mäßig.*

Benutzbarkeit: Um richtig mit dem System arbeiten zu können, und um ein geeignetes Feedback geben zu können, ist ein Verständnis der Arbeitsweise des neuronalen Netzes nötig. Dadurch, dass die Trainingsphase des Netzes direkt in das Retrieval eingebaut ist, wird die Bedienung erschwert. Die Benutzbarkeit ist *mäßig*.

5.2. Die SFB 501 Experience Base

Die im Rahmen des Sonderforschungsbereichs 501 zur Entwicklung großer Systeme mit generischen Methoden erstellte Experience Base [FGMRR99] ist ein System, das eine Wiederverwendung von Artefakten auf Basis des Quality Improvement Paradigms unterstützt. Dabei werden ergänzend zu der Experience Base nach [BCR94] noch zusätzliche während der Durchführung des Prozesses benötigte Artefakte in einem getrennten Bereich der Experience Base (der Experiment Spezifischen Sektion) gespeichert. Alle in dem Prozess anfallenden Artefakte werden in der Experience Base verwaltet.

5.2.1. Arbeitsweise

In der Experience Base werden sogenannte *Erfahrungselemente* abgelegt. Ein Erfahrungselement kann ein beliebiges Artefakt sein und in verschiedenen Formaten vorkommen. So kann z.B. das Erfahrungselement für ein SDL Pattern aus einer Sammlung von HTML Seiten, einem FrameMaker Dokument und einem speziellen Binärformat für ein Design Tool bestehen.

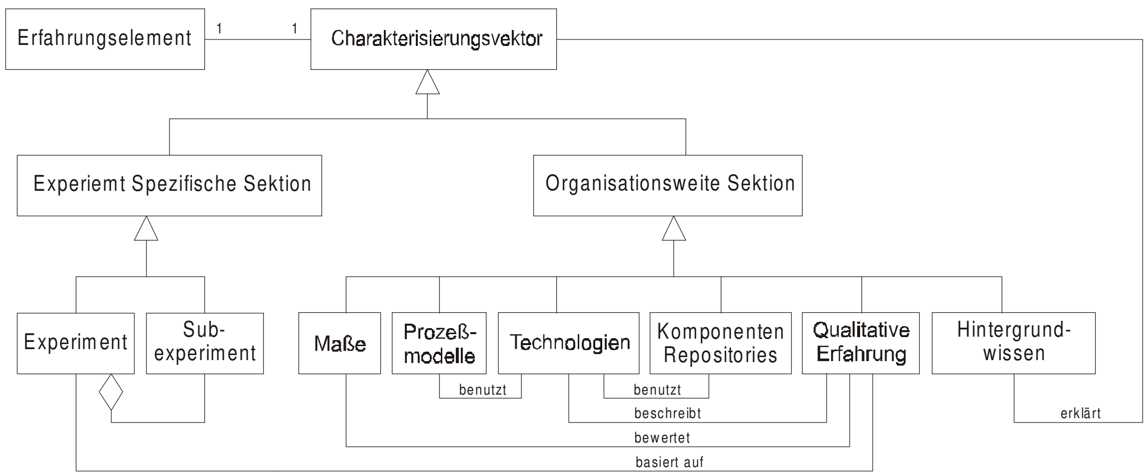

Abbildung 5: Hierarchie der Charakterisierungsvektoren

Abbildung 5 listet die Hierarchie der Charakterisierungsvektoren. Auch hier ist die Unterteilung in die Experiment Spezifische (*ESS*) und die Organisationsweite Sektion (*OWS*) erkennbar. In der ESS werden alle Informationen zum momentan laufenden Experiment gespeichert. Die OWS ist ein Experience Base nach [BCV94]. Momentan sind in der OWS sechs Klassen von Charakterisierungsvektoren modelliert, die die meisten im Softwareentwicklungsprozess auftretenden Artefakte beschreiben. Es können neue Klassen definiert, und von jeder Klasse noch weiter abgeleitet werden, so dass eine sehr detailliierte Hierarchie entsteht. Jede Klasse erbt die Eigenschaften ihrer Vorfahren. Außerdem existiert eine Reihe weiterer Assoziationen zwischen den Erfahrungselementen, die die Beziehung der Erfahrungselemente zueinander beschreiben. So ist z.B. modelliert in welchem Prozessmodell eine bestimmte Technologie verwendet wird. Diese Beziehungen gehen auch in die Experiment Spezifische Sektion, so dass eine sehr gute Verfolgbarkeit gegeben ist.

Die Benutzeroberfläche erlaubt eine Navigation der Experience Base an Hand der Beziehungen, oder eine ähnlichkeitsbasierte Suche nach bestimmten Artefakten. Bei der Suche wird zunächst eine Art *Supersuchmaske*, die alle Attribute aller Charakterisierungsvektoren enthält, präsentiert. Jetzt ist es entweder möglich, die Suche auf Erfahrungselemente einer bestimmten Klasse einzuschränken, oder in allen Klassen zu suchen, die die ausgefüllten Attribute beinhalten. Als Ergebnis werden die ähnlichsten Erfahrungselemente, gruppiert nach Klassen, zurückgeliefert. Die Suche kann jetzt weiter eingeschränkt werden, oder es kann an Hand der Links manuell durch die Erfahrungsdatenbank navigiert werden.

5.2.2. Architektur

Die gesamte Erfahrungsdatenbank basiert auf objektrelationaler Datenbanktechnologie. Die unterschiedlichen Formate der Erfahrungselemente werden direkt als BLOBs (*Binary Large Objekt)* in der Datenbank gespeichert. Die Hierarchie der Charakterisierungsvektoren ist direkt im Datenbankschema modelliert.

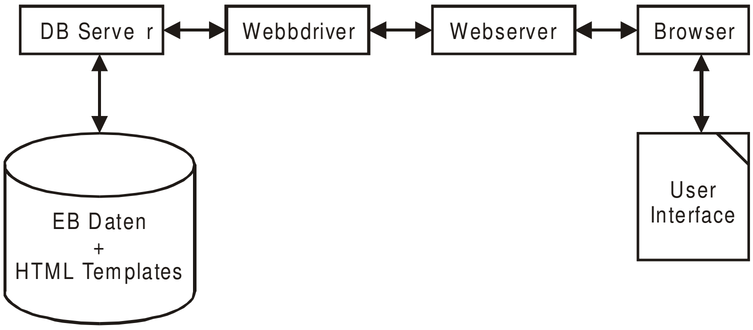

Abbildung 6: Architektur der Experience Base

Abbildung 6 zeigt den Aufbau der Experience Base. Wie erkennbar ist, wird ein webbasierter Ansatz verfolgt. Als Datenbankserver dient Informix [Inf99]. Alle Webseiten sind als Templates in der Datenbank gespeichert. Die Seiten werden direkt aus dem Metadatenkatalog der Datenbank erzeugt. Die Funktionen zur Ähnlichkeitsberechnung sind als *user-defined functions* gespeichert, und werden über in HTML Seiten eingebettete SQL Anweisungen angestoßen. Diese Vorgehensweise garantiert eine sehr hohe Flexibilität gegenüber Änderungen in der Struktur der Charakterisierungsvektoren oder der Berechnung der Ähnlichkeit.

5.2.3. Beurteilung

Die SFB 501 Experience Base unterscheidet sich ein wenig von den bisher beschriebenen Systemen. Es ist ein allgemein einsetzbares System, das alle im Softwareentwicklungsprozess anfallenden Artefakte verwaltet. Durch die Experiment Spezifische Sektion wird auch eine gewisse Unterstützung bei der Durchführung des Prozesses angeboten.

Das System lässt sich am besten in die Klasse der *deskriptiven Methoden* einordnet, da die Beschreibung der Artefakte durch manuell erstellte Charakterisierungsvektoren erfolgt. Eine Einordnung in die Klasse der *topologischen Methoden* ist auch möglich, da eine ähnlichkeitsbasierte Suche implementiert ist.

Präzision und Recall: Durch geschickte Wahl der Attribute der Charakterisierungsvektoren kann eine relativ hohe Präzision erreicht werden. In dem momentan eingesetzten Beispielsystem sind leider keine quantitativen Messungen vorgenommen worden. Durch die stark strukturierte Hierarchie der Charakterisierungsvektoren, ist sicherlich eine *hohe* Präzision zu erwarten. An Hand der gespeicherten Beziehungen zwischen den einzelnen Artefakten, lässt sich gut durch die Datenbank navigieren, so dass auch sehr leicht verwandte Artefakte gefunden werden können. Es ergibt sich ein *sehr hoher* Recall.

Abdeckung: Das System führt eine lineare Suche über alle Artefakte aus, deren Charakterisierungsvektoren die entsprechenden Attribute enthalten. Dazu müssen zunächst die Beschreibungen aller Artefakte besucht werden. Anschließend wird auf dieser Teilmenge ähnlichkeitsbasiert gesucht. Durch die Hierarchie der Charakterisierungsvektoren lässt sich so ein Teil der Artefakte von der Suche ausschließen, dennoch ergibt sich in der Praxis eine *hohe* Abdeckung.

Aufwand pro Vergleich: Das Ähnlichkeitsmaß zum Vergleich zweier Vektoren ist relativ einfach und wird als benutzerdefinierte Funktion direkt im Datenbankserver berechnet. Der Aufwand pro Vergleich ist deswegen *mäßig.*

Aufwand bei Einführung: Da das System darauf ausgelegt ist, möglichst viel an zusätzlicher semantischer Information zu speichern, ist es sehr aufwändig anzulegen und zu pflegen. Für alle bereits existierenden Artefakte müssen die verschiedenen Repräsentationen zusammengestellt werden, und ein entsprechender Charakterisierungsvektor angelegt werden. Zusätzlich dazu müssen alle Beziehungen zu bereits existierenden Erfahrungselementen angelegt werden. Dies ist mit *sehr hohem* manuellen Aufwand verbunden.

Laufende Kosten: Die während der Durchführung eines Prozesses entstandenen oder gereiften Artefakte müssen anschließend wieder manuell in die Erfahrungsdatenbank aufgenommen werden. Dabei wird zwar eine rudimentäre Unterstützung angeboten, die manche Attribute der Charakterisierungsvektoren ausfüllt und einige semantische Links anlegt. Dennoch muss die meiste Arbeit manuell geleistet werden. Allerdings ist das System sehr flexibel gegenüber strukturellen Änderungen, wie z.B. Erweiterung der Hierarchie der Charakterisierungsvektoren. Solche Änderungen sind mit minimalem Aufwand möglich und werden von den meisten bisher betrachteten Systemen gar nicht erst angeboten. Deshalb sind die laufenden Kosten zusammengenommen nur *mäßig.*

Benutzbarkeit: Durch die Supersuchmaske ist ein sofortiges Arbeiten mit der Experience Base möglich. Um die Ergebnisse nachvollziehen zu können, ist eine Kenntnis der Hierarchie der beschreibenden Vektoren hilfreich. Die Konfigurierung des Distanzmaßes erfordert eine tiefere Einarbeitung in das System. Insgesamt ist der Retrievalprozess leicht verständlich und die Navigation durch die Experience Base an Hand der Beziehungen einleuchtend. Die Benutzbarkeit des Systems ist *hoch.*

6. Ausblick

Wie wir gesehen haben, existieren die unterschiedlichsten Ansätze zur Implementierung eines Software Reuse Repositories. Und obwohl schon seit einigen Jahren intensiv an diesem Gebiet gearbeitet wurde, ist noch keine allumfassende Lösung gefunden worden. Systeme der Zukunft werden wahrscheinlich einen Schritt in Richtung prozessunterstützender Systeme gehen. Mit dem Ziel eines vollständig integrierten Systems vor Augen, dass Prozessmodellierung und Ausführung genauso unterstützt wie ein umfassendes Repository zur Speicherung aller anfallenden Daten. Erste Schritte in diese Richtung sind bereits unternommen worden, und die SFB 501 Experience Base ist einer davon. Bis jedoch solche Systeme ihre Marktreife erreicht haben, wird noch einige Zeit vergehen.

Literaturverzeichnis

[BCR94] V. Basili, G. Caldiera, H. Rombach. The Experience Factory. In J.J. Marciniak (ed) *Encyclopedia of Software Engineering. Volume 1. p 469-476, 1994.*

[FGMRR99] R. Feldmann, B. Geppert, W. Mahnke, N. Ritter, F. Rößler. The ORDBMS-based SFB 501 Experience Base, Technischer Bericht 98/99, Sonderforschungsbereich 501, Fachbereich Informatik, Universität Kaiserslautern, 1999.

[FN87] W.B. Frakes and B.A. Nejmeh. An Information System for Software Reuse. *Proceedings of the 10th Minnowbrook Workshop on Software Reuse, Syracuse University, Minnowbrook, NY, 1987.*

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Pattern. *Addison-Wesley 1995.*

[Inf99] Informix Universal Server Documentation. Informix Software Inc., http://www.informix.com/.

[He97] S. Henninger, An Evolutionary Approach to Constructing Effective Software Reuse Repositories. *ACM Transactions on Software Engeneering and Methodology Vol. 6, No. 2, pp. 111-140, 1997.*

[MMM98] A. Mili, R. Mili, R.T. Mittermeir. A survey of software reuse libraries. *Annual of Software Engineering 5, pp. 349-414, 1998.*

[MG91] T. Moineau, M.C: Gaudel. Software Reusability through Formal Specifications. *Proceedings on the First International Workshop On Software Reusability. SWT Memo Nr. 57, Universität Dortmund. pp 202-212, 1991.*

[PP94] S. Paul, A. Prakash. A Framework for Source Code Search Using Program Patterns. *IEEE Transactions on Software Engineering SE-20. pp. 462-475, 1994.*

[PP92] A. Podgurski and L. Pierce. Retrieving Reusable Software by Sampling Behaviour. *Proceedings of the 14th International Conference on Software Engineering. pp. 300-304, 1992.*

[PD85] R. Prieto-Diaz. A Software Classification Scheme. *PhD Thesis. Departement of Computer and Information Science. University of California at Irvine, 1985.*

[SC93] G. Spanoudakis, P. Castantopoulos. Measuring Similarity between Software Artifacts. *Proceeding of the 6th International Conference on Software Engineering and Knowledge Engineering. SEKE ´94, 1993.*