Vorschlag Hypermodelling: Data Warehousing für Quelltext

Tim Frey
OvG University Magdeburg, Germany
tim.frey@tim-frey.com

ABSTRACT
This paper explains the idea to load source code into a Data Warehouse. First, separation of concerns is explained. Following, the motivation to load source code in a Data Warehouse is briefly presented. Afterwards, the multi-dimensionality of software is discussed. Also, a first model for software in a Data Warehouse is shown. Nearby, the challenge that multiple cubes will be needed in order to load software in a Data Warehouse is elucidated. Thereafter, related work is shown and its relation to the revealed idea is explained. Finally, conclusions are done and future work paths are described.

Categories and Subject Descriptors
D.2.3 [Software Engineering]: Coding Tools and Techniques; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

General Terms
Human Factors

Keywords
Separation of Concerns, OLAP

1. EINLEITUNG
Bei der Softwareentwicklung ist das Prinzip der Separation of Concerns (deutsch: Trennung der Belange, kurz SOC) etabliert. Dieses Prinzip entspricht der Handlungsweise, ein Softwaresystem aus verschiedenen Blickwinkeln zu betrachten und die Codierung für jeden Blickwinkel, in einzelnen Modulen zu erstellen [1]. Diese Blickwinkel werden oftmals auch Belang oder Concern genannt. Verschiedene Programmiersprachen stellen dabei verschieden mächtige Mechanismen zur Verfügung, um Module für die einzelnen Blickwinkel zu erstellen. Durch die Modularisierung wird eine erhöhte Wiederverwendbarkeit und Verständlichkeit erreicht. Trotz der Anwendung von SOC ist die Erstellung und die Untersuchung von Software eine große Herausforderung. Ferner ist es nicht einfach möglich, einzelne Concerns in Module zu kapseln, weil eine "absolute" Modularität mit gängigen eingesetzten Programmierparadigmen nicht möglich ist [2,4]. So ist es zum Beispiel bei der Kodierung von Problemstellungen durch das objektorientierte Paradigma nicht möglich, alle Concerns in einzelne Module zu verpacken. Ein Modul ist oftmals für verschiedene Zuständigkeiten zugleich programmiert [2, 3, 4]. Folglich sind verschiedene Concerns in Modulen vermischt und werden nicht in einzelnen Modulen, wie es eigentlich die Idee von SOC ist, kodiert. Um die Analyse von Software im Zusammenhang mit Concerns zu ermöglichen, muss folglich ein Analyseverfahren den Umstand der Vermischung von Concerns in Modulen beachten. Der Beitrag dieses Papiers ist daher die Idee, mehrdimensionale Analyseverfahren, die solche Umstände beachten können, aus dem betriebswirtschaftlichen Kontext für Quelltext einzusetzen. Es werden dabei erste Überlegungen zu deren Einsatz präsentiert. Das Fernziel des Einsatzes dieser Analyseverfahren ist es, Data Warehouse Technologie als Werkzeug zur Quelltextanalyse nutzen zu können.[1]
Im Folgenden wird zuerst die Motivation und Vision, Quelltext in ein Data Warehouse zu laden, beschrieben. Danach wird die Mehrdimensionalität von Software erläutert, um den Bezug zu mehrdimensionalen Daten im Data Warehouse zu geben. Im Anschluss werden erste relationale Schemata präsentiert und diskutiert, die Quelltext in einem Data Warehouse abbilden können. Durch deren Darstellungen werden weitere Anforderungen für zukünftige Schemata aufgedeckt. Nachfolgend werden verwandte Arbeiten, im Vergleich zu der in diesem Papier vorgeschlagenen Idee, erläutert. Am Ende werden Rückschlüsse und weitere Arbeitspfade beschrieben.

GvD Workshop'11, 31.05.-03.06.2011, Obergurgl,Österreich.

2. MOTIVATION UND VISION
Im betriebswirtschaftlichen Bereich existiert eine Vielzahl von Systemen, deren Daten in einem Data Warehouse zusammengefasst und homogenisiert werden. Dies ermöglicht es, diese zu aggregieren, Mining zu betreiben und strategische Entscheidungen zu treffen [5]. Des Weiteren werden Data Warehouses zur integrierten Unternehmensplanung verwendet. Dabei werden Planziele in einem Data Warehouse hinterlegt [6].[2]

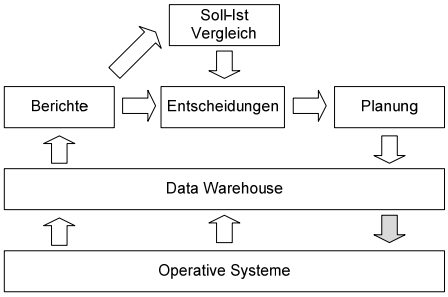

Abbildung 1: Prozesse mit einem Data Warehouse

Dargestellt ist dies in Abbildung 1. Daten werden aus operativen Systemen in ein Data Warehouse geladen. Diese Daten

[1] Aufgrund des Bezuges des Papieres auf SOC und Data Warehouse Technologien wird im weiteren Verlauf von einer Vertrautheit mit SOC [1] und den hierbei auftretenden Problemen [2,4], Frameworks [8,9], aspekt-orientierter Programmierung [12], domänen-spezifischen Sprachen [11], Annotationen [10] und Grundkenntnissen im Data Warehouse Bereich [5,7] ausgegangen.

[2] Eine integrierte Unternehmensplanung ist die Hinterlegung von Planzielen in einem Data Warehouse. In diesem werden hierbei Merkmalkombinationen in Verbindung mit Kennzahlen gespeichert, wobei zumindest ein Merkmal ein zukünftiges Zeitdatum darstellt. Der Zusammenhang zwischen strategischen Entscheidungen und Planung ist, dass strategische Entscheidungen zu Planzielen führen können.

werden dann genutzt, um Berichte zu erzeugen. Aufgrund dieser Berichte werden Entscheidungen getroffen. Diese führen zu einer Planung für die Zukunft. Der Plan wird dann als Plandaten im Data Warehouse gespeichert. Dabei können die Plandaten auch dazu führen, dass automatisiert in die operativen Systeme geschrieben wird (grauer Pfeil). Durch Planung und deren Umsetzung entstehen neue Daten in den operativen Systemen. Diese werden dann wieder in das Data Warehouse geladen und können mit den Plandaten verglichen werden. Das führt zu Entscheidungen und gegebenenfalls zu einem neuen Plan [7].
Bei der Softwareentwicklung fallen ebenfalls Artefakte in einer Vielzahl von Systemen an, und bei vielen Projekten ist die Quelltextbasis zu groß, um einfach überblickt oder analysiert zu werden. Die Vision ist daher eine integrierte, homogenisierte Sicht auf Software als multidimensionales Gebilde, durch die die verschiedenen Artefakte, die im Produktlebenszyklus auftreten, abgebildet werden können. Der Kern der Idee ist, dass Software nicht nur aus der Applikationslogik besteht, sondern aus einer Vielzahl weiterer Artefakte, wie zum Beispiel Tests. Eine multidimensionale Anordnung im Data Warehouse könnte eine integrative Komponente sein, die es erlaubt, die Software aus verschiedenen Blickwinkeln zu betrachten. Desweiteren würde es der Einsatz dieser Technologie zulassen, außer der Analyse weitere Anwendungen möglich zu machen. Der Planungsprozess in Data Warehouses, der zur integrierten Planung verwendet wird, könnte dazu dienen, Qualitätskriterien zu definieren, oder auch Weiterentwicklungen von Software zu planen und diese dann mit der erfolgten Realität zu vergleichen.
Ziel dieser Arbeit ist es daher, erste Möglichkeiten des Speicherns von Quelltext in einem Data Warehouse zu untersuchen. Diese erste Evaluation soll Hinweise über die Realisierbarkeit des Einsatzes von Data Warehouses zur Quelltextuntersuchung liefern.

3. MEHRDIMENSIONALE SOFTWARE

Software wird normalerweise unter der Nutzung von Frameworks, die vorgefertigte Funktionalität bereitstellen, erstellt. [8,9]. Software kann als ähnlich zu einem n-dimensionalen Hyperwürfel betrachtet werden. Die Ecken bzw. Kanten des Würfels sind Frameworkfunktionen. Durch diese Anschauung ist es nötig, Software nicht nur aus einer Position zu betrachten. Die Ansicht ist ähnlich zu einem Hyperwürfel und kann gedreht werden. Eine Drehung kann hierbei verschiedene Dimensionen in den Vordergrund verschieben und andere im Hintergrund verschwinden lassen. Dieser Vergleich zeigt, dass der Mensch nicht dazu fähig ist Software, die ähnlich einem Hyperwürfel ist, direkt und vollständig zu erfassen. Darstellungen sind vielmehr Projektionen in den Verständnisraum des Menschen.
Die Analogie zu einem Hyperwürfel kann auf dessen Konstruktion reduziert werden. Dabei kann ein Würfel gesehen werden, der durch weitere Dimensionen erweitert wird. Im Falle von Software wird ein Würfel durch Quelltext "befüllt", der sich im vorgegebenen Rahmen eingliedert. Dies bedeutet, dass Quelltextmodule aufgrund der Nutzung verschiedener Frameworkfunktionen verschiedenen Dimensionen zugehörig sind. Diese Zugehörigkeit ergibt sich daraus, dass die einzelnen Concerns nicht in Modulen getrennt sind, und somit ein Modul durch die Nutzung verschiedener Funktionen verschiedenen Dimensionen gleichzeitig zugehörig sein kann.
Beim Erstellen von Software werden Funktionen, die im „aktuell erstellten“ Quelltext selbst sind und nicht aus Frameworks stammen, auch genutzt. Der Programmierer erzeugt sich ein eigenes Framework für seine spezielle Aufgabe. Dieses setzt auf bestehende Funktionen von Frameworks auf oder definiert vollkommen neue Funktionen. Somit wird ein Teil des Würfelinhaltes zu neuen Dimensionen. Diese befinden sich in dem Würfelrahmen und erzeugen hierin spezialisierte Unterräume oder sie spannen Dimensionen auf, die orthogonal zu den bisherigen Dimensionen stehen. Dabei rücken diese neuen Dimensionen in den Vordergrund und bisherige Dimensionen werden verdeckt, was einer Drehung des Würfels ähnlich ist.
In Abbildung 2 wird Quelltext, der selbst in eine Frameworkdimension aufsteigt, grafisch visualisiert. Dabei sind zuerst zwei Frameworks A und B zu sehen. Der Quelltext konsumiert die Fähigkeiten eines Frameworks A und eines anderen B, womit eine Komposition von beiden Frameworks erreicht wird. Ein solches Konsumieren kann z.B. ein Funktionsaufruf oder jedes andere Nutzen der Fähigkeiten eines Frameworks sein. Die Pfeile zeigen das Konsumieren von Fähigkeiten an. Aufgrund dessen, dass neuer Quelltext den bestehenden Quelltext nutzt, steigt der zuvor erzeugte Quelltext im unteren Teil der Grafik, selbst zur eigenständigen Dimension auf. Diese wird dann von neuem Quelltext konsumiert. Dabei kann es auch sein, dass der New Code nicht mehr das originale Framework B nutzt, sondern nur den zuvor erzeugten Code und Framework A, was durch die Schraffierung angedeutet ist.
Die Idee ist, dass Frameworkfunktionen oftmals Concerns repräsentieren. Somit können die verschiedenen Concerns der Software mit Dimensionen gleichgesetzt werden.

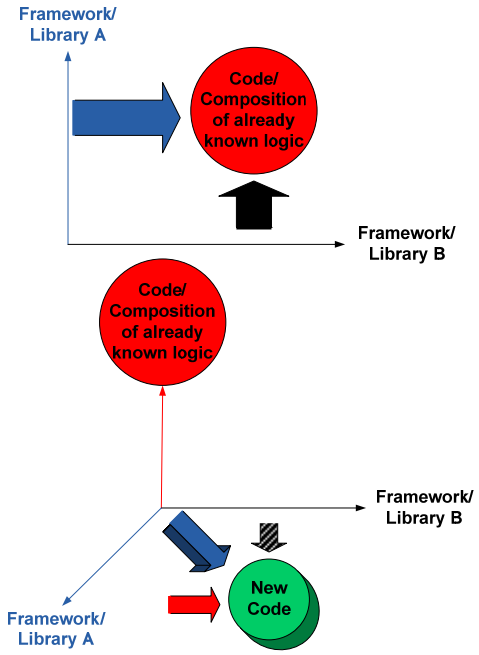

Abbildung 2: Quelltext, der zur Dimension aufsteigt

Daraus folgt eine abstrahierte Darstellung des Sachverhaltes von Abbildung 2 in Abbildung 3. Hierbei wurden die Dimensionen des Quelltextes mit C für Concern gekennzeichnet. Die Komposition durch Quelltext von verschiedenen Concerns ist durch den Verbindungsvektor VC1 dargestellt. In Abbildung 3b wird diese Dimension, beziehungsweise dieser Concern, selbst wiederum konsumiert (VC2). Der gestrichelte VC1 Vektor und dessen Parallelverschiebung zeigt, dass die neue Dimension komplett gleichberechtigt zu den „normalen“ C Dimensionen ist.

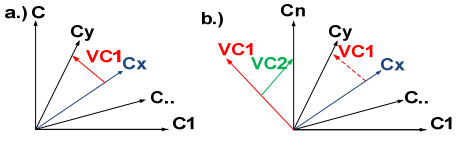

Abbildung 3: Quelltext der zur Dimension aufsteigt

Ein weiterer Umstand, den es zu beachten gilt, ist die Anordnung der Concerns im Falle von Quelltext. Dieser kann, wie beschrieben mehreren Concerns zugehörig sein. Diese können

sich zudem in verschiedenen Hierarchien befinden. Beispielsweise könnte eine Hierarchie VC2 zu VC1 und Cn sein. VC1 wiederum ist die Wurzel der Cx, Cy Hierarchie. Abstrakt dargestellt ist dies in Abbildung 4a. Dort wird die Hierarchie visualisiert, in der sich VC2 befindet. VC2 bildet dabei den Ursprung von dem aus die Hierarchie aufgespannt wird. In Abbildung 4b wird der Sachverhalt verallgemeinert gezeigt, dass Dimensionen/Concerns auch hierarchisch angeordnet sein können. Verschiedene Tiefenebenen sind als Dimensionsebenen (DE1- DEn) beschriftet.[3] Die Wurzel der Hierarchien bildet dabei das Fragment, in dem die einzelnen Concerns komponiert wurden. Ein praktisches Beispiel für Hierarchien sind Methodenaufrufe oder Vererbung.

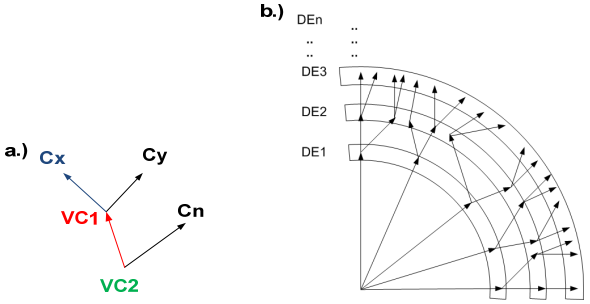

Abbildung 4: Dimensionshierarchien

Da in modernem Quelltext oftmals eine hohe Anzahl von Frameworks verwendet wird, stellt sich die Herausforderung, trotz der Analogie zwischen Frameworkfunktionen und Dimensionen, diese bei einer Umsetzung in einem Schema nicht vollständig gleichzusetzen. Dies hat den Grund, dass oftmals eine Vielzahl von Frameworks in Quelltext verwendet wird und somit eine hohe Anzahl verschiedener Dimensionen wahrscheinlich die Skalierbarkeit der Umsetzung beeinflussen würde. Folglich sollten Frameworkfunktionen als eine Dimension mit verschiedenen Elementen, den eigentlichen Funktionen der Frameworks, modelliert werden.
Ziel ist ein Modell, welches die zuvor genannten Sachverhalte in einem Data Warehouse abbildet. Dies kann genutzt werden, um Abfragen auf Quelltext zu ermöglichen. Aufgrund der Mehrdimensionalität des Ansatzes und dem Hintergrund, dass bei der Programmierung Modelle die Realität abbilden, ist der Name der Kombination, zwischen Data Warehouse und Softwarequelltext, Hypermodelling.

4. MODELLIERUNG

Data Warehouses nutzen im Normalfall eine relationale Datenquelle, um OLAP-Würfel zu füllen. Daher ist der erste Schritt ein relationales Schema, das Quelltext in einem Data Warehouse abbildet. In Data Warehouses werden zu der Abbildung von Daten in relationalen Tabellen im Normalfall Stern- oder Schneeflockenschemata verwendet [31]. Nachfolgend werden zwei relationale Modelle, inspiriert von diesen, gezeigt. Zuletzt wird die OLAP Darstellung angerissen und Problematiken des entwickelten Schemas aufgezeigt.

4.1.1 Einfaches Schema

In Abbildung 5 ist exemplarisch eine Tabellenstruktur mit Fragmenten, die in Quelltext vorkommen, an ein Sternschema angelehnt, dargestellt. In der Abbildung ist das zentrale Element die Faktentabelle. Eine Zeile in dieser repräsentiert ein Quelltextfragment, wie ein Modul. Ein solches ist zum Beispiel eine Klasse oder Funktion. Eine Klasse kann Annotationen enthalten. Oftmals werden Annotationen in Frameworks definiert und daher ist die Annotationstabelle mit der Frameworktabelle verknüpft. Die Parameter Tabelle zeigt die Möglichkeit, dass Annotationen auch Parameter besitzen können. Zusätzlich sind oftmals externe domänenspezifische Sprachen (domain specific language (DSL)) mit Quelltext assoziiert. Durch diese können zu Quelltextfragmenten verschiedene Funktionalitäten hinzu konfiguriert werden [11]. Ein bekanntes Beispiel hierfür ist die Konfiguration persistenter Klassen über eine DSL. In modernem Quelltext treten auch Aspekte [12] auf. Aspekte sind Fragmente die es ermöglichen, Funktionalität, die nach dem Objektparadigma nicht an einer Stelle kodiert werden kann, an einem zentralen Punkt zu realisieren. In diesem wird dann angegeben welche Module von dem Aspektcode (Advice) betroffen sind. Die Konfiguration der betroffenen Module erfolgt über sogenannte pointcuts. Zuletzt kann ein Modul einer Komponente und Aufgaben zugordnet werden. Beispielsweise können solche Aufgaben das Einpflegen von Änderungen in der Funktionalität darstellen.

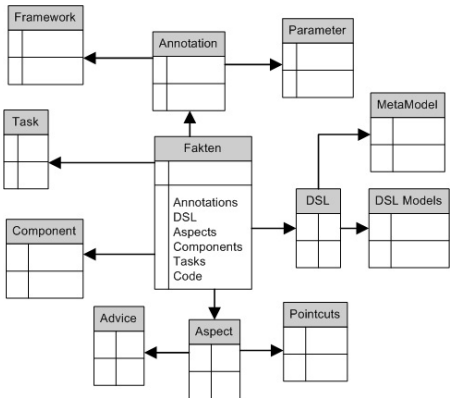

Abbildung 5: Grundlegendes relationales Quelltextschema

4.1.2 Schneeflockenschema

Das in Abschnitt 4.1.1 dargestellte Schema wurde in Abbildung 6, an ein Schneeflockenschema angelehnt, erweitert. Eine generische Zuordnung von Modulen zu Concerns wird besser ermöglicht und mehr Sachverhalte können dargestellt werden. Ebenfalls wird der Umstand beachtet, dass eine generische Darstellung der Beziehungen im Quelltext aus Abschnitt 3 dargestellt wird und verschiedene Frameworks nicht als eigenständige Tabelle realisiert sind. Vielmehr sind die Frameworkfunktionen in einer Tabelle zusammengefasst.
Wie zu sehen ist, verweist die Faktentabelle auf verschiedene Concerns, zu denen die Zugehörigkeit durch Cmembership in Prozent ausgedrückt werden kann. Als Modulgranularität wurde eine Funktion gewählt. Diese stellt eine kleine Einheit klarer Funktionalität dar und gliedert sich in eine Klasse ein. Folglich ist Aggregation von Fakten auf Klassenebene möglich. Eine Funktion kann andere Funktionen nutzen (Function_call). Hierbei kann spezifiziert werden, ob der Konsum ein Aufruf oder ein andere Nutzungsart (Usage type), wie zum Beispiel das Überschreiben einer Funktion eines Frameworks ist. Ebenfalls können Typen eines Frameworks, wie zum Beispiel durch Annotationen oder Erbschaftbeziehungen, konsumiert werden (Type_usage).
Rückverweise auf die Faktentabelle zeigen, dass Hierarchien von Funktionen möglich sind. Die Aufrufhierarchie wird durch die CallHierarchy Table dargestellt. Die EvolvedConcernCode Tabelle ermöglicht es, Funktionen/Fakten selbst als Concern zu führen. Durch diese Tabelle kann Quelltext, der sich zum Concern entwickelt (siehe Abschnitt 3), aber kein Teil eines Frameworks ist, dargestellt werden.

[3] Die Ebenen sind rein zur besseren Visualisierung dargestellt; nicht jede Dimension muss über die gleiche Anzahl von Hierarchien verfügen.

Besonders interessant ist die Issue Tabelle, die Bugs darstellt. Diese können auf zugehörige Tests verweisen. Dabei können auch Tasks oder deren Kontext mit solchen Issues verbunden sein. Ebenfalls wurde der Umstand ins Modell eingeflochten, dass Packages oftmals Layer zugeordnet werden können. Metrics, Profiling info und Author zeigen, dass weitere Informationen im Bezug auf Quelltext dargestellt werden können.

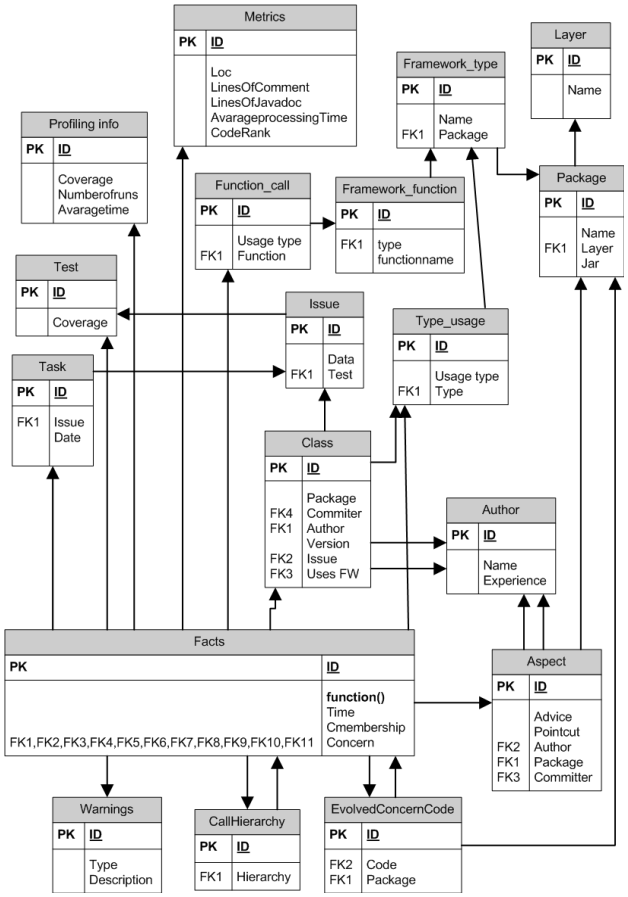

Abbildung 6: Quelltext, Schneeflockenschema

4.1.3 OLAP-Darstellung

Grundlegend zeigt das Schema aus Abschnitt 4.1.2 bei näherer Betrachtung die Einschränkung, dass eine direkte Umsetzung in einen OLAP-Würfel keine elegante Lösung darstellt. Insbesondere die rückverweisende CallHierarchy besitzt Eigenschafen einer Faktentabelle; CallHierarchy kann mehrmals gleiche Einträge besitzen, wenn eine Funktion mehrmals die gleiche Funktion aufruft. Zur Realisierung sollte somit mit der Sichtweise einer zentralen Faktentabelle gebrochen werden.

In Abbildung 7 ist aufgrund der zuvor genannten Einschränkungen daher die Idee, mehrere OLAP-Würfel zu erstellen, die einen Bezug zu den Daten eines konkreten Programmes, das analysiert werden soll, haben. Ziel ist es einen OLAP-Würfel, ähnlich A, zu erstellen, der Zuordnungen mit einem logischen Modell einer Programmiersprache enthält. Hierbei kann dieser OLAP-Würfel durch zusätzliche Daten, die im direkten Zusammenhang zu der Programmiersprachenlogik, stehen angereichert werden. Solche Daten könnten zum Beispiel Tests oder deren Ergebnisse sein. Dieser OLAP-Würfel stellt ein Abbild der Programmiersprachenstruktur dar, welches durch ein konkretes Program als Fakten „befüllt“ wird.

In einem weiteren OLAP-Würfel werden dann Fakten, die nicht der Programmiersprachenstruktur entsprechen, definiert. Jedoch besitzen die Daten zumindest einen Bezug zu dem Quelltextmodel. Im Vergleich zu OLAP-Würfel A ist OLAP-Würfel B somit keine Repräsentation der Programmstruktur, sondern vielmehr die Assoziation von Programmelementen miteinander. Ein Beispiel für die Assoziation von Programmelementen miteinander ist die zuvor beschriebene Aufrufhierarchie. Da somit beide OLAP-Würfel eine Assoziation zu dem Quelltext besitzen, können diese dann in einem Verknüpften OLAP-Würfel zusammengeführt werden. In diesem Linked Cube werden dann zusätzliche Fakten, mit denen des logischen Modells einer Programmiersprache zusammen geführt.

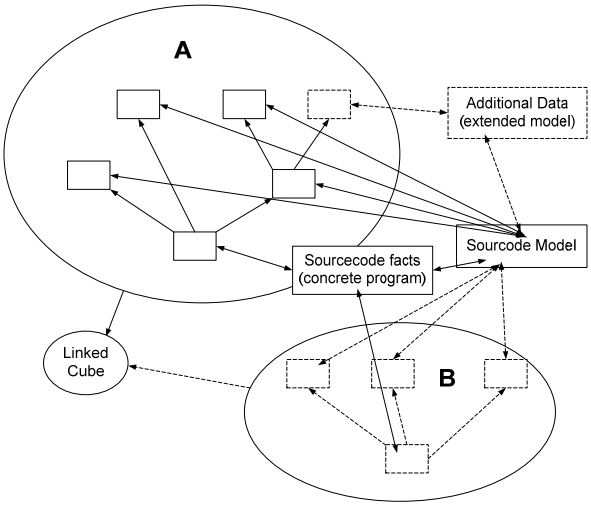

Abbildung 7: OLAP-Würfel, Zuordnung von Quelltext

Schlussendlich zeigt sich, dass für die Entwicklung eines relationalen Schemas, zuerst Fakten von der Struktur von Quelltext getrennt werden sollten. Danach können die verschiedenen Fakten in einem weiteren Modell zusammengeführt werden.

5. VERWANDTE ARBEITEN

Auf der Homepage[4] des Hypermodelling Projektes befinden sich weitere Informationen über die Idee, Quelltext in ein Data Warehouse zu laden und verwandte Forschung.

Die Gruppe der Source Code Query Tools stellt im Wesentlichen Programme dar, die Quelltext aufgrund von Abfragen untersuchen können. Diese Werkzeuge können als verwandte Arbeiten zu der Idee von Hypermodelling gesehen werden, da diesen zu Grunde liegt, Quelltext, aufgrund von Abfragen zu untersuchen. JQuery ist ein Abfrage basierter Source Code Browser für Java. Realisiert ist er als Eclipse Plugin [13]. JQuery stellt eine deklarative Abfragesprache zur Verfügung und erlaubt es, durch diese verschiedene Ansichten auf Quelltext zu erzeugen [14]. Ferret ist ein weiteres Source Code Query Tool und wurde von Brian de Alwis im Rahmen seiner Dissertation entwickelt [15, 16]. Es erlaubt es ähnlich JQuery, Abfragen an Java Quelltext zu erstellen. Ferret ist hauptsächlich dafür gedacht, den Quelltext aufgrund seiner Aufrufstruktur und Vererbung zu untersuchen. Lost ist ein Query Tool für aspekt-orientierte Programmierung [17]. Grundlegend verfolgen alle Source Code Query Tools das gleiche Ziel und besitzen ähnliche Funktionalität. JQuery erscheint von allen Tools das am besten gepflegteste und mächtigste zu sein und kann somit als bestes Vergleichsprodukt zu Hypermodelling gesehen werden. Im Vergleich zu Hypermodelling unterscheiden sich die Source Code Query Tools darin, dass diese nicht das Ziel verfolgen, Data Warehouse Technologie zur Analyse von Quelltext zu nutzen. Ebenfalls orientiert sich ihre Funktionsweise oder die Abfragemöglichkeiten nicht an Data Warehouse Technologie.

[4] http://hypermodelling.com

Martin Robillard forscht auf dem Gebiet der Concernanalyse. Sein Werkzeug, ConcernMapper [18], soll helfen, Quelltext nach verschiedenen Concerns zu unterteilen und ist auf manuelle Zuordung von Concerns zu Quelltextfragmenten ausgelegt. Der Unterschied zu den Source Code Query Tools ist, dass der Concern Mapper keine Abfragesprache oder ähnliche Mittel nützt, sondern dieser auf die rein manuelle Zuordnung von Concerns ausgelegt ist. Der größte Nachteil im Vergleich zu Hypermodelling ist, dass das Anlegen und Zuordnen der Concerns eine Mehrarbeit des Entwicklers bedeutet und nicht das Ziel verfolgt wird, Abfragen aufgrund dem Entwickler bekannter Quelltextelemente zu ermöglichen.

In [19] wird ein Verfahren beschrieben, bei dem ein Metrics Warehouse eingesetzt wird. Dieses erinnert aufgrund des Warehouses entfernt an die Idee des Hypmermodelling. Dabei wird bei dem Verfahren beschrieben, wie Projekte anhand von Metriken, gesteuert werden können. Dies bezieht sich auf Metriken aus betriebswirtschaftlichen Systemen und nicht direkt auf Metriken, die durch eine Funktion aus Quelltext berechnet werden können. Diese Metriken werden dann in ein nicht näher erläutertes Metrics Warehouse geladen, das einem Data Warehouse ähnlich ist. Der genaue Aufbau des Metrics Warehouses, die Struktur der Daten, und der Inhalt des Warehouses wird nicht näher beschrieben.

Source Code Mining ist die Anwendung von Data Mining Algorithmen auf die Quelltext, Bug Datenbanken und Versionsverwaltungssysteme [20]. Somit stellt diese Technik auch die Zielsetzung auf, Quelltext zu analysieren und dadurch Verbesserungen zu erreichen, und kann somit als ähnlich erachtet werden. Die Hauptidee hierbei ist es, Bugs aus Issue Tracking Systemen, mit deren Lösungen aus Versionsverwaltungssystemen, aufgrund der entsprechenden Quelltexte zu verknüpfen und darauf Data Mining zur Analyse zu nutzen. Muster im Quelltext werden aufgedeckt, interpretiert und Handlungen abgeleitet. Solche Analysen können beispielsweise zu Feststellungen führen, dass Problemlösungen, die an einem bestimmten Wochentag gemacht wurden, mit einer hohen Wahrscheinlichkeit wieder zurückgenommen werden [21]. Das Source Code Mining besitzt den Nachteil, dass derzeit nur Zusammenhänge zwischen Fehlern und Beziehungen zwischen einzelnen Quelltextelementen im Source Code aufgedeckt werden können. Es wird dabei kein Verfahren offenbart, wie weitere Daten in die Wissensbasis integriert werden können. Des Weiteren verfolgt Source Code Mining im Vergleich zu Hypermodelling nicht das Ziel, eine Datenbasis in Form des Ladens von Daten in ein Data Warehouse zu erschaffen, auf welcher dann Analysen ausgeführt werden können.

In [22] wird Software Intelligence beschrieben. Software Intelligence wird hier als die Anwendung von Business Intelligence Mechanismen oder Technologien auf Software beschreiben. Dies ist der hier vorgestellten Idee ähnlich. Der Unterschied liegt darin, dass Software Intelligence nicht das Ziel verfolgt, Quelltext in OLAP-Würfel zu laden und sich auf Source Code Mining konzentriert.

Die Untersuchung von Quelltext unter der Zuhilfenahme einer relationalen Datenbank zur Analyse wird im Sourcerer Projekt durchgeführt [23]. Das relationale Modell umfasst vier Tabellen. Quelltext wird in Projekte, Dateien, Kommentare und Entitäten die mit Beziehungen verknüpft sind, aufgeteilt. Solche Entitäten sind zum Beispiel Klassen, Interfaces und Methoden [24]. Im Vergleich zu Hypermodelling zeigt der Umfang des relationalen Modells einen wesentlichen Unterschied; Hypermodelling visualisiert schon in den ersten vorgestellten Modellen die verschiedenen Beziehungen in unterschiedlichen Tabellen. Zudem ist das Ziel von Hypermodelling, Data Warehouse und insbesondere OLAP-Würfel zu verwenden. Dies ermöglicht Aggregationen von Fakten, was bei Sourcerer nicht erstrebt wird. Dies ist zudem auch darin zu sehen, dass Sourcerer keine Faktentabellen nutzt, um Beziehungen im Quelltext abzubilden.

Codegenie, dass auf Sourcerer aufsetzt, ermöglicht es, Komponenten aufgrund von Testfällen aufzudecken und in ein Programm zu integrieren [25, 26]. Ähnlich zu Codegenie zeigt [27] weitere Möglichkeiten der Komponentenaufdeckung, unter der Zuhilfenahme von Codesuchmaschinen und gibt einen Überblick und Vergleich über die Aufdeckung von Komponenten. Im Vergleich zu diesen Codesuchmaschinen liegt der Fokus von Hypermodelling auf der Analyse und Abfragen aufgrund von Concerns. Eine Verwendung der Hypermodelling Idee zur Codesuche könnte jedoch eine interessante Anwendung sein.

Orthographic Software Modeling (OSM) ist der Ansatz, mehrdimensionale Navigation durch Modelle bei der Softwareentwicklung einzusetzen [30]. Ziel ist es, Quelltext und die zugehörigen Modelle in einem zentralen Modell abzubilden um daraus die verschiedenen Modellansichten dynamisch erzeugen zu können [28]. Um die Navigation durch die verschiedenen Modellansichten zu ermöglichen, werden verschiedene Dimensionen und deren Ausprägungen definiert. Zum Beispiel eine Dimension die in ihren Ausprägungen die Abstraktionsebenen festlegt oder eine Projektionsdimension, deren Ausprägung die Struktur oder die Interaktion von Elementen beschreibt. Die Auswahl einer Kombination von Ausprägungen von Dimensionen bestimmt letztendlich dann, welches Modell gezeigt wird [29]. Die mehrdimensionale Navigation und die Sichtweise von Software als ein mehrdimensionales Konstrukt, das von verschiedenen Blickwinken aus betrachtet werden kann, ähnelt Hypermodelling. Jedoch verfolgt OSM nicht das Ziel, Data Warehouse ähnliche Mechanismen zu nutzen, um dadurch Quelltext untersuchen zu können. Ferner liegen die Ziele von Hypermodelling nicht direkt in der Modellierung von Software, sondern vielmehr darin, Abfragen unter der Nutzung verschiedener Concerns zu ermöglichen. In diesem Kontext könnte natürlich auch die dynamische Erstellung von Ansichten bei OSM als Abfragen verstanden werden, und es könnte interessant sein, eine Kombination zwischen OSM und Hypermodelling zu untersuchen.

Bezogen auf die Data Warehouse Schemata zeigen [5] und [31] einen umfassenden Überblick über das Thema Data Warehouse. Hier werden die verschiedenen Schemata besprochen und praktische Beispiele aufgezeigt. Business Intelligence, wie auch die integrierte strategische Unternehmensplanung im speziellen Anwendungsfall, wird in [7] beschreiben. Ein betriebswirtschaftlich- orientierter Überblick der integrierten Unternehmensplanung stellt [6] dar. Einen herstellerneutralen Überblick über die Planung mit Data Warehouse Systemen bietet [32].

6. ZUSAMMENFASSUNG UND AUSBLICK

In diesem Papier wurde die Idee, Quelltext in ein Data Warehouse zu laden, vorgestellt. Hierbei wurde beschrieben, dass Quelltext ein mehrdimensionales Gebilde aus Concerns ist. Dabei wurde insbesondere der Umstand hervorgehoben, dass Module oftmals mehreren Concerns zugeordnet werden können. Es wurde erläutert, dass in diesem mehrdimensionalen Gebilde neue Dimensionen durch Kompositionen erschaffen werden können. Dieser Umstand zeigt eine besondere Herausforderung, im Bezug auf das Laden von Quelltext in ein Data Warehouse. Unter Beachtung dieser Rahmenbedingung wurden relationale Modelle mit Faktentabellen präsentiert, in die Quelltext geladen werden kann. Durch deren Darstellung konnte die Problematik aufgedeckt und verdeutlicht werden: Eine einzige Faktentabelle ist zur Darstellung von Quelltext nicht ausreichend. Zuletzt wurden verwandte Arbeiten präsentiert, deren Analyse zeigt, dass die Idee, Quelltext in ein Data Warehouse zu laden, bisher noch nicht durchgeführt wurde. Dennoch deu-

ten die verwandten Arbeiten darauf hin, dass der primäre Einsatzort für Abfragewerkzeuge derzeit die integrierte Entwicklungsumgebung ist.

Eine wichtige Folgerung aus den Darstellungen dieses Papiers ist, dass bei der Erstellung eines weiteren Schemas, Quelltext in verschiedene Fakten aufgeteilt werden sollte. Dies wird benötigt, um ein relationales Modell zu erstellen. Somit stellt sich für zukünftige Arbeiten die Frage, welche Mittel einer Programmiersprache als strukturell und welche als assoziativ zueinander begriffen werden können. Eine derzeitige Überlegung ist daher Quelltext als Mechanismen von Kategorisierung (strukturell) und Komposition (assoziativ) zu betrachten. Kategorisierung ist hierbei alles, das einer logischen Zuordnung dient. Zum Beispiel Klassen zu Packages. Komposition stellen sämtliche Elemente dar, die eine direkte Auswirkung auf die Funktionalität ausüben, wie Methodenaufrufe. Dennoch sind hier Diskussionspunkte offen. So ist es zum Beispiel fragwürdig, ob Ableitungen eher eine Kategorisierung oder eine Komposition darstellen. Ziel dieser Überlegungen ist es Quelltext in OLAP-Würfel zu laden, um dadurch komplexe Analysen von Quelltext zu ermöglichen.

Eine weitere Überlegung kommt aus der Betrachtung der verwandten Arbeiten, die Großteils für Eclipse umgesetzt wurden. Diese inspirieren dazu, OLAP ähnliche Abfragen direkt in der integrierten Entwicklungsumgebung zu ermöglichen. Dies kann ein besonders großer Vorteil für Entwickler sein, da diese mit solchen Hypermodelling Abfragen direkt in der Entwicklungsumgebung Quelltext untersuchen können.

7. LITERATUR

[1] E. W. Dijkstra. Selected Writings on Computing: A Personal Perspective. On the role of scientific thought. Springer.1982

[2] W.Harrison, H. Ossher. Subject-oriented programming: a critique of pure objects. OOPSLA '93. ACM. 1993

[3] Bernhard Lahres, Gregor Rayman. Objektorientierte Programmierung. Galileo Computing. 2009

[4] H. Ossher, P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In Proceedings of the Symposium on Software Architectures and Component Technology. Kluwer. 2001.

[5] W.H. Inmon. Building the Data Warehouse. 4th ed., J.Wiley & Sons, New York. 2005.

[6] M. C. Meier, W. Sinzig, P. Mertens. Enterprise Management with SAP SEM/Business Analytics. 2nd Edition, Springer. Berlin. 2005

[7] J. M. Gómez, C. Rautenstrauch, P. Cissek. Einführung in Business Intelligence mit SAP NetWeaver 7.0. Springer.2008

[8] S. H. Kaisler. Software paradigms. John Wiley and Sons.2005

[9] W. Pree. Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design. ECOOP 94. Springer. 1994

[10] J. A. Bloch. Metadata Facility for the Java Programming Language. 2004. http://www.jcp.org/en/jsr/detail?id=175

[11] T. Stahl, M. Völter, S. Efftinge, A. Haase. Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management. Dpunkt Verlag.2007

[12] R. E. Filman, T. Elrad, S. Clarke, M. Aksit Aspect-Oriented Software Development. Addison-Wesley Professional; 1 edition. 2004

[13] JQuery a query-based code browser. homepage. The University of British ColumbiaVancouver, Canada. http://jquery.cs.ubc.ca/index.htm.

[14] K. De Volder. JQuery: A Generic Code Browser with a Declarative Configuration Language. In Practical Aspects of Declarative Languages, 8th International Symposium. Springer. 2006.

[15] B. de Alwis, G. C Murphy. Ferret: A Tool Supporting Software Exploration. The University of British Columbia Vancouver, Canada. http://www.cs.ubc.ca/~bsd/research/ferret/

[16] B. de Alwis. Supporting Conceptual Queries Over Integrated Sources of Program Information . PHD Thesis. University of British Columbia, Vancouver Canada. 2008

[17] J.-H. Pfeiffer. Complex code querying and navigation for AspectJ. OOPSLA ’05. ACM. 2005

[18] M. P. Robillard and F.Weigand-Warr. ConcernMapper: simple view-based separation of scattered concerns. In Proceedings of the 2005 OOPSLA Workshop on Eclipse technology eXchange, ACM, 2005

[19] C. R. Pandian. Software metrics: a guide to planning, analysis and application. Auerbach Publications. 2003

[20] Mining Software Archives.Lehrstuhl für Softwaretechnik Universität des Saarlandes – Informatik. Prof. Zeller. http://www.st.cs.uni-saarland.de/softevo/.

[21] T. Zimmermann, A. Zeller. When do changes induce fixes?. ICSE 05, ACM. 2005

[22] A. E. Hassan, T. Xie.Software Intelligence: Future of Mining Software Engineering Data.In Proceedings of FSE/SDP Workshop on the Future of Software Engineering Research .Santa Fe. ACM. 2010

[23] Sushil Bajracharya, Trung Ngo, Erik Linstead et. al. Sourcerer: A Search Engine for Open Source Code Supporting Structure-Based Search, OOPSLA’06. USA. ACM. 2006

[24] Sushil K. Bajracharya, Joel Ossher, Cristina V. Lopes, Sourcerer - An Infrastructure for Large-scale Collection and Analysis of Open-source Code, Third International Workshop on Academic Software Development Tools and Techniques, Belgium, ACM, 2010

[25] Otavio Lemos, Sushil Bajracharya et al.. CodeGenie: Using Test-Cases to Search and ReuseSource Code, ASE’07, 2007, Atlanta, Georgia, USA. ACM

[26] Otávio Augusto Lazzarini Lemos , Sushil Bajracharya and Joel Ossher , CodeGenie: a Tool for Test-Driven Source Code Search, OOPSLA’07, Canada. ACM, 2007

[27] O. Hummel, “Semantic Component Retrieval in Software Engineering”,Ph.D. dissertation, Faculty of Mathematics and Computer Science, University of Mannheim, 2008.

[28] C. Atkinson and D. Stoll: "Orthographic Modelling Environment", in Proceedings of Fundamental Approaches to Software Engineering (FASE'08), Hungary, Spring, 2008

[29] C. Atkinson and D. Stoll: "An Environment for the Orthographic Modeling of Workflow Components", in Proceedings of the Prozessinnovationen mit Unternehmenssoftware (PRIMIUM), Germany, 2008

[30] C. Atkinson, D. Brenner, et al.. Modeling Components and Component-Based Systems in KobrA, in A. Rausch, R. Reussner et al. The Common Component Modeling Example: Comparing Software Component Models, Springer, 2008

[31] R. Kimball, M. Ross. The data warehouse toolkit: the complete guide to dimensional modeling. John Wiley and Sons. 2002

[32] F. Navrade. Strategische Planung mit Data-warehousesystemen. Gabler Verlag. 2008.