Automatisierte Visualisierung von statischen Systemanforderungen zur Qualitätssteigerung in Softwareprojekten

Kai Gebhardt

Lehrstuhl für Softwaretechnik
Friedrich Schiller Universität Jena
Ernst-Abbe-Platz 3
07743 Jena
Kai.gebhardt@uni-jena.de

Betreuer der Arbeit: Prof. Dr. Wilhelm R. Rossak

Abstract: Die Entwicklung von Individualsoftware zur besseren Unterstützung der unternehmensspezifischen Arbeitsabläufe ist heute keine Seltenheit mehr. Dabei soll die Software möglichst gut an den Menschen als Endanwender und die Organisationstruktur angepasst sein. Werden in einem solchen Projekt Anforderungen zu Beginn nicht richtig verstanden, kann als Folge das gesamte Projekt scheitern, da ein Produkt entworfen wird, das für die Endanwender und deren Arbeitsabläufe keinen Nutzen bringt. Eine Lösung besteht darin, die potentiellen Endanwender in die Anforderungsermittlung einzubinden. Dazu ist eine gute Kommunikation zwischen ihnen als Domänenexperten und dem Anforderungsanalysten nötig. Dieses Paper schlägt eine Methodik vor, die bereits ermittelte Informationen aus einer Darstellung in einer kontrollierten Sprache automatisch in eine übersichtliche visuelle Form überführt, um nach einem Interview als Diskussionsgrundlage zu dienen. Dabei können fehlerhaft notierte Informationen entdeckt und korrigiert werden, was die Qualität der Anforderungen erhöht. Vertiefend wird in diesem Paper eine neue Darstellungsform für statische Domänenmodelle vorgestellt, welche auf Methoden aus dem Bereich der Informationsvisualisierung aufbaut. Die hier entworfene Visualisierung kann in ein UML-Klassendiagramm abgeleitet werden, womit das Verfahren zu einem modellgetrieben Softwareentwurf erweitert werden könnte.

1 Motivation

Viele Softwareprojekte scheitern, weil Anforderungen unvollständig oder fehlerhaft aufgenommen und umgesetzt werden. Dieses Problem tritt vor allem bei der Entwicklung von Individualsoftware auf, welche speziell für einen Kunden entwickelt wird. Dazu ist es nötig, dass im Vorfeld die Anforderungen an die Softwarelösung genau ermittelt werden, um auf die individuellen Bedürfnisse der Endnutzer und Arbeitsabläufe eingehen zu können. Es exisiteren eine Vielzahl an Studien (zum Beispiel [KKLK05] und [Das07]), die sich damit beschäftigen, dass eine frühzeitige Nutzereinbindung die Qualität der ermittel-

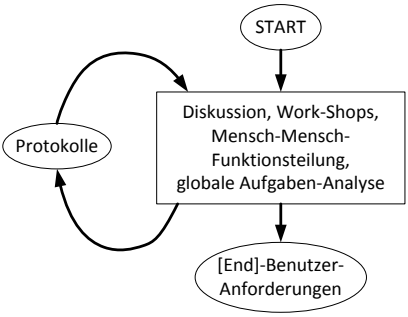

Abbildung 1: Teil eines partizipativen Softwareentwicklungsmodells aus [Rau91]

ten Anforderungen und damit den Projekterfolg erhöht. Ein Problem bei der Zusammenarbeit von interdisziplinären Teams ist deren unterschiedliche Bildung und Erfahrung, was die Kommunikation erschwert. Es ist daher noch immer eine herausfordernde Aufgabe, wie Stakeholder ohne IT-Hintergrund am besten in die Anforderungsermittlung einbezogen werden sollten. Durch Probleme in der Kommunikation werden Informationen oft fehlerhaft aufgenommen.

Rauterberg [Rau91] schlägt ein partizipatives Softwareentwicklungsmodell vor, welches das Problem der fehlerhaft aufgenommenen Informationen adressiert. Das Vorgehensmodell beruht auf einem Optimierungszyklus in jeder der Projektphasen Anforderungsermittlung, Design, Implementierung und Test. Für diese Arbeit ist der in Abbildung 1 zu sehende Ausschnitt des Modells von Interesse, der den lokalen Optimierungszyklus während der Anforderungsermittlung zeigt. Anforderungen werden in einem Interview, einer Diskussion oder einem Work-Shop gesammelt und durch Protokolle festgehalten, welche von allen Beteiligten so lange unter Begutachtung stehen, bis keine Inkonsistenzen mehr entdeckt werden. Durch dieses Vorgehen wird die Anforderungsqualität iterativ verbessert.

Auf eine hohe Projektbeteiligung von Endanwendern setzen auch agile Softwareentwicklungsmethoden [CLC03], mit dem Ziel flexibler und schlanker als klassische Vorgehensmodelle zu sein. Ein Teilgebiet ist die Agile Modellierung, bei der es darum geht, das Analysemodell über mehrere Iterationen hinweg mit starker Endnutzereinbindung schrittweise zu entwickeln. Zwischen den Iterationen sind optionale Reviews denkbar.

Ziel unserer Arbeit ist es eine Methode im Bereich des Requirements Engineering zu entwickeln, welche im angegeben Teilzyklus aus Rauterbergs Modell sowie in der Agilen Modellierung Anwendung finden kann, um den Endnutzern die aufgenommenen Informationen schnell und übersichtlich präsentieren zu können, was den Reviewprozess verbessert. Damit sollen die Ansätze für die Praxis noch attraktiver werden, da auf zusätzliche Zeit und Kosten durch Nachbearbeitungen der Ergebnisse und mehrmalige Diskussiontreffen verzichtet werden könnte.

Kernpunkt ist ein geeignetes Konzept zur Erstellung und Präsentation der zu begutachtenden Anforderungen. Werden reine Textdokumente als Protokolle genutzt, besteht bei

großen Projekten die Gefahr, dass diese nicht mehr leicht zu überblicken sind. Zudem besitzen Textdokumente gegebenenfalls Inkonsistenzen oder Doppeldeutigkeiten. Wird auf formalere Modelle zur Protokollierung des Interviews zurückgegriffen, besteht die Gefahr, dass die Domänenexperten diese nicht mehr verstehen. Zudem sind formale Modelle aufwändiger in der Erstellung, was eine Nutzung im direkten Meeting negativ beeinträchtigen könnte, da die zusätzliche Zeit nicht vorhanden ist. Aus den vorherigen Überlegungen ergeben sich aus unserer Sicht daher die folgenden Anforderungen an die zu verwendenden Protokolle:

- A1 Die Darstellung ist auch *für nicht Informatiker verständlich*.
- A2 Die gesammelten Informationen sind so aufbereitet, dass sie *schnell und einfach zu überblicken* sind.
- A3 Anforderungen sind *möglichst formal* erfassbar. Hierunter fällt eine weitestgehende Vermeidung von Redundanzen, Inkonsistenzen oder Doppeldeutigkeiten.
- A4 Protokolle sind in angemessener Zeit erstellbar (*Effizenz*).
- A5 Die aufgenommenen Informationen sind zur Weiterverarbeitung möglichst automatisch *in Modelle überführbar*. Aufgrund der großen Praxisverbreitung wird die Unified Modeling Language [Gro11] als Zielmedium gewählt.

Anforderung A5 ist für einen Praxiseinsatz relevant, da eine Modelltransformation für den späteren Systementwurf hilft und somit eine Effizenzsteigerung darstellt. Über Vorteile durch den Einsatz von Modellen berichtet [MW08].

Der in dieser Arbeit verfolgte Lösungsgedanke beruht darauf, dass Anforderungen von dem Analysten durch einen Texteditor aufgenommen werden, der eine kontrollierte Sprache unterstützt. Die Vorteile kontrollierter Sprachen in der Anforderungsermittlung sind Kapitel 2.5 zu entnehmen. Um die Informationen dann allen Beteiligten schnell und übersichtlich präsentieren zu können, wird eine automatische Transformation der aufgenommmenen Daten in eine geeignete Visualisierung angestrebt. Zusätzlich soll eine Abbildung der Daten auf die UML möglich sein, um mit dem weit verbreiteten objektorientierten Systementwurf kompatibel zu sein. Die funktionalen Anforderungen lassen sich im objektorientierten Systementwurf in eine statische, dynamische und funktionsorienterte Sicht unterteilen.

In diesem Paper konzentrieren wir uns zunächst auf den Aufbau der Visualisierung für statische Systemanforderungen. In Kapitel 2 wird gezeigt, dass es bisher keine Dokumentations- und Darstellungsform im Bereich des Requirements Engineering gibt, die alle gestellten Anforderungen erfüllt, um zur iterativen Qualitätsanalyse durch die Endnutzer sinnvoll einsetzbar zu sein. Im darauffolgenden Kapitel 3 stellen wir das von uns entwickelte Konzept zur endnutzertauglichen Darstellung statischer Anforderungen vor. Kapitel 4 gibt Auskunft über erste Ergebnisse und weitere geplante Forschungsarbeiten.

2 Stand der Technik im Requirements Engineering

Es exisitieren viele verschiedene Methoden um Anforderungen aufzunehmen. Um die bestehenden Ansätze bezüglich der in Kapitel 1 aufgestellten Anforderungen A1 bis A5 zu evaluieren, werden sie in Kategorien unterteilt und gemeinschaftlich verglichen. Hierbei werden die genannten Kriterien bezüglich der Aufnahme statischer Systemanforderungen in den Mittelpunkt gestellt, da dieses Paper für diese Anforderungsgruppe im Hauptteil eine Lösung anbietet.

2.1 Mathematische Methoden

Hierunter fallen Grammatiken, Logische Formalismen, Automaten, Petrinetze, Ereignis-Ausdrücke (CSP) sowie algebraisch-axiomatische Beschreibungen und ähnliche Ansätze. Eine gute Übersicht über diese Methoden im Kontext des Requirements Engineering stellt Partsch [Par10] auf.

Allen hier genannten Methoden ist eine sehr hohe Formalität gemeinsam, wodurch eine Syntax- oder Semantikprüfung zur Vermeidung von Redundanzen, Inkonsistenzen oder Doppeldeutigkeiten größtenteils möglich wäre. Auch die Ableitung in Modelle der Unified Modeling Language ist denkbar, da durch die formale Struktur der Ansätze Transformationsregeln konstruierbar wären. Formale Modelle sind hingegen ohne Fachexpertenausbildung schwer verständlich und alles andere als übersichtlich. Das Erstellen der komplexen Beschreibungen kostet zudem viel Zeit.

2.2 Zielorientierte Ansätze

Mit diesem Ansatz wird ein System durch die zu erreichenden Ziele definiert. Das globale Hauptziel kann in kleinere Unterziele zerlegt werden und so weiter. Auf diese Weise entsteht eine Zielhierarchie deren Blätter elementare Anforderungen darstellen. Zu den Zielorientierten Ansätzen zählen Und-Oder-Bäume, KAOS oder i*.

Allen Ansätzen ist gemeinsam, dass sie sich vorwiegend zur Darstellung nicht-funktionaler Anforderungen oder reiner Funktionalität eignen. Es existieren Arbeiten, die Zielmodelle in UML-Use-Case-Diagramme überführen [SC02]. Mylopoulos et al. [MCY99] schlägt vor, Zielmodelle in frühen Phasen des Requirements Enginering einzusetzen, während in späteren Phasen objektorientierte Analysemodelle folgen sollten. Zusammenfassend lässt sich folgern, dass Zielmodelle nicht dafür vorgesehen sind, um existierende statische Beziehungen zwischen Systemelementen darzustellen. Für die Kriterien A1 bis A5 sind sie daher bezüglich statischer Anforderungen nicht geeignet. Zudem sind Zielmodelle nur beschränkt intuitiv verständlich, wie Chung et al. in ihren Studien aufzeigen: *'However, all but one of the interviewees did not initially understand the details of the goal graphs.'* [CN95].

2.3 Strukturierte Methoden

Im Mittelpunkt dieser Ansätze steht die Kommunikation von Prozessen über Daten. Vertreter strukturierter Methoden sind die Structured Analysis and System Specification, Structured System Analysis sowie Modern Structured Analysis (MSA).

Partsch [Par10] bietet einen guten Überblick über diese Ansätze. Die ersten beiden Methoden seien nahezu identisch und beschäftigen sich mit der Kommunikation von Prozessen über Daten. Die Datenbeschreibungen können durch aussagenlogische Formeln dargestellt werden. Die Modern Structured Analysis erweitert diese Ansätze um das Entity-Relationship-Model zur Darstellung statischer Beziehungen zwischen Systemelementen sowie um Zustandsdiagramme zur Verdeutlichung der Systemdynamik.

Bezüglich der Darstellung statischer Systemanforderungen ist daher nur der MSA-Ansatz relevant. Die verwendeten E/R-Modelle sind in ihrer Komplexität mit UML-Klassendiagrammen vergleichbar. Daher ist deren Eignung für einen direkten Einsatz im Kundengespräch als gering einzuschätzen (siehe Abschnitt 2.4).

2.4 Objektorientierte Methoden

Bei diesen Ansätzen geht es darum, ein System mittels Objekten und ihrer Interaktion darzustellen, womit eine Abbildung des realen Problembereiches möglich wird. Hauptvorteil der Objektorientierung ist damit die Beherrschung der Komplexität [Boo86], die vor allem Entwicklern bei der Programmierung großer Softwareanwendungen helfen sollte. Mit dem Aufkommen der Objektorientierten Analyse und Design wurden die Vorteile der Objektorientierten Programmierung bereits auf frühere Projektphasen ausgedehnt - zum defakto Standard in diesem Bereich wurde die Unified Modelling Language.

Dobing et al. [DP06] evaluieren den Nutzen der verschiedenen Diagrammarten der UML in der Praxis und betrachten dabei auch die Einbindung von Endnutzern. Die Ergebnisse zeigen eine große Nutzung des UML-Klassendiagramms, jedoch werden Endnutzer lediglich zu ca. 30% an deren Erstellung und zu ca. 50% an deren Review beteiligt. Als Ursache für diese geringen Werte werden Verständnisprobleme aufgeführt. Die Hälfte der Befragten gaben an, die Klassendiagramme nicht zu verwenden, da selbst die Anforderungsanalysten Verständnisprobleme hätten. Zur Endnutzertauglichen Darstellung statischer Anforderungen eignet sich die UML daher weniger gut, was auch Erfurth [Erf11] in ihrer Dissertation thematisiert.

2.5 Natürliche und kontrollierte Sprachen

Ein einfaches und schnelles Mittel zur Aufnahme von Anforderungen aller Art ist die natürliche Sprache. Der größte Vorteil dieser Methodik ist, dass sie einfach anzuwenden ist und sie jeder der beteiligten Stakeholder versteht. Aus den genannten Gründen verwen-

den viele Unternehmen aus der Praxis diesen Ansatz zum formulieren ihrer Anforderungen [Erf11]. Allerdings besitzt natürliche Sprache auch ihre Nachteile: Redundanzen, Inkonsistenzen oder Mehrdeutigkeiten können im Allgemeinen nicht vermieden werden. Um syntaktische Fehler zu erkennen, schlägt Stoyanova [Sto13] ein Modell zur automatischen Überprüfung von Text vor. Semantikfehler, die nur durch einen menschlichen Reviewer erkennbar sind, werden hierdurch nicht entdeckt. Ein anderer Ansatz zur Vermeidung der genannten Probleme ist der Entwurf kontrollierter Sprachen [Sch98] und Satzschablonen [Rup07]. Der Nachteil einer fehlenden Übersicht in umfangreichen Dokumenten kann aber auch damit nicht vermieden werden, was den Reviewprozess durch die Endnutzer erschwert.

2.6 Andere Ansätze zur Nutzereinbindung

Am Lehrstuhl für Softwaretechnik an der Friedrich-Schiller-Universität Jena wurden partizipative Methoden auf ihre Eignung in Softwareprojekten untersucht, um mit ihnen die Anforderungsermitttlung durchzuführen. Fachexperten sollten hierbei ihre Arbeitsabläufe mittels Kartenspielen wie Adaptionen von Card [Mul01] und CUTA [Laf96] beschreiben, welche dann auf die UML transformiert werden. Die Abbildung des im Forschungsprojekt UPEX entwickelten Card-Pal [MRE10] auf ein UML-Klassendiagramm erfolgt hierbei noch manuell, da nötige Formalisierungen fehlen. Kritikpunkt dieses Verfahrens ist der hohe zusätzliche Zeitaufwand während eines Kundeninterviews und der spielerische Charakter der eingesetzten Methoden.

Der in diesem Paper vorgestellte Ansatz grenzt sich inbesondere von UPEX dadurch ab, dass in UPEX die Endnutzer aktiv die Anforderungesmodelle durch eigenständiges Ausfüllen der Karten mit gestalten müssen, was von Kundenseite durchaus nicht unumstritten war in der Praxis. Wir verfolgen den Ansatz, dass der Anforderungsanalyst selbst das Modell entwirft und durch eine automatische Visualisierung ein Feedback der Endnutzer einholen kann, ob er deren Wünsche richtig umgesetzt hat. Das hier vorgestellte Verfahren ist vermutlich schneller und birgt weniger Akzeptanzrisiken, da der Anforderungsanaylst zunächst mit den Endnutzern ein seriöses Interview führt, womit er gleichzeitig das Vertrauen seiner Interviewpartner gewinnt und erst gegen Ende ohne großen Mehraufwand durch automatische Visualisierungen auf unkonventionelle Weise die bis dahin erreichten Ergebnisse verbessern kann.

Eine weitere Möglichkeit objektorientierte Ansätze verständlicher darzustellen bieten CRC Karten [BNT02]. Klassen werden hierbei mittels Karten dargestellt. Die Kommunikation zwischen Klassen kann durch Rollenspiele nachgestellt werden. Diese Methodik wurde jedoch dafür entworfen, unerfahrenen Programmierern die Objektorientierte Programmierung näher zu bringen. Für informatik-fremde Anwender besitzt sie eine sehr hohe Komplexität.

Den Ansatz nutzerverständlicher Diagramme durch die Einführung von Bildern bzw. Piktogrammen zu erhalten, wird in [BH12] verfolgt. Hier werden dynamische Systemaspekte mittels einer piktogrammbasierten Geschäftsprozessmodellierung beschrieben. Unter-

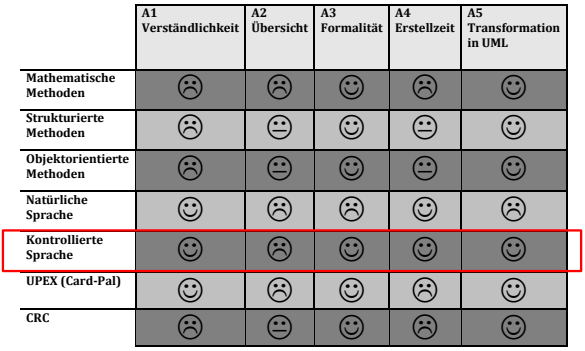

Abbildung 2: Evaluationsergebnisse bestehender Requirements Engineering Techniken

schiede zu der in diesem Paper vorgestellten Methode sind, dass Breitling et al. über die Dynamik zur Statik gelangen und die Endnutzer aktiv in die Diagrammerstellung einbinden, während wir den umgekehrten Weg von der Statik zur Dynamik verfolgen und den Endnutzer als Qualitätsprüfer statt aktiven Entwickler betrachten.

In Abbildung 2 sind die Ergebnisse der Evaluation noch einmal festgehalten. Zur Evaluation wurden Meinungen aus wissenschaftlichen Fachartikeln und Lehrbüchern zu Rate gezogen. Zudem wurde an einem kleinen Fallbeispiel dessen Darstellungen mit Vertretern aller Kategorien durchgeführt, um die Ansätze effektiv bewerten zu können. Die Zielorientierten Ansätze und die exemplarische Geschäftsprozessmodellierung fehlen in der Grafik, da diese Ansätze zur Darstellung von funktionsorientierten bzw. dynamischen aber nicht für statische Anforderungen geeignet sind. Es ist zu sehen, dass keine der exisitierenden Methodiken alle fünf Kriterien erfüllt. Ein möglichst gutes Ergebnis liefert der strukturierte Text, weshalb unsere Methodik auf diesem aufbaut und um eine automatische Visualisierung erweitert, um die geforderte bessere Übersicht von Anforderung A2 zu erfüllen. Diese Visualisierung wird im folgenden Kapitel vorgestellt.

3 Konzept zur Darstellung statischer Anforderungen

In diesem Kapitel wird eine Darstellung für Domänenmodelle entworfen. Wir verwenden sie in diesem Paper zur Präsentation statischer Systemanforderungen in einem Softwareprojekt. Die Anwendbarkeit der Modelle ist jedoch nicht darauf beschränkt. Die Darstellung soll für nicht-Informatiker verständlich sowie kompakt und übersichtlich sein. Als Zielsetzung unserer Visualisierungen streben wir an, mindestens die Hauptkonstrukte eines UML-Klassendiagrammes damit widerzuspiegeln. Das sind im Einzelnen:

- **Klassen:** Schablonen der eigentlichen im System vorhandenen Objekte. Sie besitzen Attribute und Methoden.
- **(un-)gerichtete unäre und binäre Assoziationen:** Die Beziehungen zwischen Systemelementen.
- **Spezialisierung/Generalisierung:** Eine Klasse erbt die Eigenschaften und Methoden einer anderen Klasse.
- **Aggregation und Komposition:** Eine Teil-Ganzes Beziehung zwischen Systemelementen. (Wird in diesem Paper nicht behandelt.)

Die Darstellungen werden so konzipiert, dass sie aus Text automatisch generierbar wären, um in das in Kapitel 1 beschriebene Gesamtkonzept integrierbar zu sein. Die entworfene Darstellung stützt sich auf Erkenntisse aus dem Forschungsgebiet der Informationsvisualisierung [PD10]. Die zuvor beschriebenen statischen Systemanforderungen werden durch zwei Sichten dargestellt, um eine Informationsüberfrachtung zu vermeiden: Die *Instanzmatrix* und die *Vogelperspektive*.

3.1 Die Instanzmatrix

Mit dieser Sicht werden vorhandene Systemklassen sowie gerichtete und ungerichtete Assoziation zwischen ihnen dargestellt.

Wie man sich leicht überlegen kann, ist es möglich ein UML-Klassendiagramm als einen gerichteten Graphen $G = (V, E, C)$ darzustellen. Eine Klasse wird dabei durch einen Knoten $n \in V$ repräsentiert und eine unäre Beziehung zwischen zwei Klassen wird durch eine gerichtete Kante $e \in E$ widergespiegelt. Durch ein Kantengewicht $c \in C$ kann die spezielle Art und Kardinalität der Beziehung codiert werden. In Abbildung 3 ist ein Klassendiagramm und sein stellvertretender Graph zu sehen. Eine binäre UML-Assoziation zerfällt wie zu sehen ist in zwei Kanten.

Für Graphen exisitieren bereits eine Reihe von Visualisierungstechniken, welche durch obige Überlegungen auch auf UML-Klassendiagramme anwendbar sind. Im Folgenden wird die von Bertin [Ber67] eingeführte Matrixvisualisierung betrachtet. Die Grundidee besteht darin einen kantengewichteten Graphen $G = (V, E, C)$ durch eine Adjazenzmatrix mit $a_{i,j} = \begin{cases} c_{i,j} & \text{falls } (i,j) \in E \\ 0 & \text{sonst} \end{cases}$ darzustellen. Es exisitieren einige Abwandlungen des Ansatzes von Bertin, welche statt des Kantengewichtes die ensprechenden Felder in der Matrix einfärben [HF06]. Ein hohes Gewicht würde beispielsweise mit einer dunklen und ein niedriges Gewicht mit einer hellen Farbe gekennzeichnet. Der in Abbildung 3 zu sehende Graph ist in Abbildung 4 als entsprechende Matrix dargestellt. Wie an diesem Beispiel erkennbar ist, bringt diese Form der Visualisierung noch keine eindeutigen Vorteile bezüglich Verständlichkeit. Eine farbliche Codierung der Beziehungen ist wenig aussagekräftig.

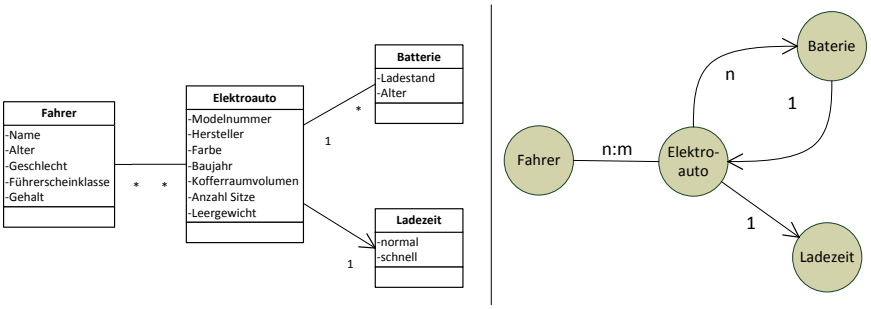

Abbildung 3: Ein UML-Klassendiagramm und ein äquivalenter Graph

	Fahrer	Elektro-auto	Batterie	Ladezeit
Fahrer		n:m		
Elektro-auto	m:n		1:n	gerichtet zu 1
Batterie		n:1		
Ladezeit				

Abbildung 4: Matrixdarstellung eines UML-Diagramms

Unsere Idee besteht darin, diesen Ansatz zu erweitern, um eine geeignete Visualisierung für objektorientierte Problemstellungen zu erhalten. Wie bereits in Abschnitt 2.4 geschildert, hat die Objektorientierung das Abbilden der realen Welt als Grundgedanken. Hierzu definieren wir eine neue Matrizenart.

Definition 1: Aufbau der Instanzmatrix

Sei M = $\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$ eine Instanzmatrix. Dann gilt:

- Alle Felder a_{1j} mit $j = 2, ..., n$ und a_{i1} mit $i = 2, ..., m$ werden als Klassenfelder bezeichnet. Ein Klassenfeld enthält den Namen sowie eine bildliche Darstellung einer Instanz der Klasse. Zudem ist für jedes Klassenfeld eine Detailansicht verfügbar, welche die vorhandenen Attribute steckbriefartig aufzeigt. In Abgrenzung zu einer Adjazenzmatrix, welche nur die Beziehungen der Knoten darstellt, nehmen wir bewusst die Knoten in die Matrixdefinition mit auf, da diese wegen ihrer Zusatzinformationen einen hohen Stellenwert in unserem Visualisierungskonzept besitzen.

- Die übrigen Felder a_{ij} mit $i \neq 1 \wedge j \neq 1$ heißen Relationsfelder. Ein Relationsfeld a_{ij} gibt die Beziehung zwischen den Klassen aus den Feldern a_{i1} und a_{1j} an. Die Kardinalitäten der Beziehung werden durch die Anzahl der dargestellten Objekte widergespiegelt. Tabelle 5 zeigt die Transformationsregeln zwischen einer UML-Assoziation und den dargestellten Instanzen in der Instanzmatrix. Bei der Abbildung einer binären UML-Assoziation fassen wir zur besseren Übersicht die Informationen aus den Relationsfeldern a_{ij} und a_{ji} in dem Feld a_{ij} zusammen und trennen sie durch einen Querstrich. Das Feld a_{ji} bleibt leer. Damit bleiben alle Felder unterhalb der Matrixdiagonalen leer. Zur besseren Übersicht wird empfohlen immer nur eine Teilmatrix $k \times l$ mit $k \leq n$ und $l \leq m$ zu betrachten. Durch sinnvolle Zeilen und Spaltenumordnungen kann die Anzahl der zu betrachteten Teilmatrizen begrenzt werden, da Zeilen oder Spalten mit leeren Relationsfeldern uninteressant sind und daher gelöscht werden können. Erste Erfahrungswerte haben gezeigt, dass $k = l = 5$ ein sinnnvoller Wert zum Präsentieren ist.

Durch die bildliche Darstellung von Begriffen kann das visuelle System der menschlichen Betrachter besser ausgenutzt werden, womit die Grafiken schneller und einfacher zu überblicken sind. Die Umsetzung des UML-Diagrammes als Instanzmatrix ist in Abbildung 5 dargestellt. An dem Beispiel ist schön zu sehen, dass auf die beiden Zeilen 4 und 5 verzichtet werden könnte, da diese leer sind. Das bringt den Vorteil, dass dem Endnutzer eine kleinere Matrix präsentiert werden könnte. Falls das Gesamtsystem aus mehr wie vier Klassen besteht, könnten die Felder a_{41} und a_{51} durch andere Klassen ersetzt werden, womit gleich mehr Assoziationen auf einmal in der 5×5 - Matrix darstellbar wären.

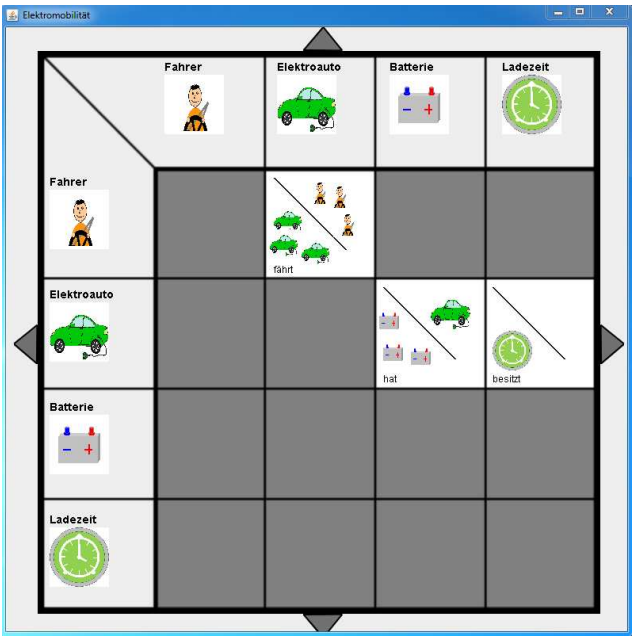

Abbildung 5: Beispiel einer Instanzmatrix (Screenshot unseres Prototypen)

Anzahl Objekte der Klasse A im Relationsfeld	Anzahl Objekte der Klasse B im Relationsfeld	Kardinalität der Assoziation im Klassendiagramm gelesen von A nach B
1	1	1:1
1	3	1:n
3	3	n:m
keine	$c \in \{1, 3\}$	gerichtet mit Kardinalität abhängig von c

Tabelle 1: Mapping von Relationsfeldern und Assoziationen

3.2 Ausblick: Die Vogelperspektive zur Darstellung von Vererbungsbeziehungen

Bei der Vererbung handelt es sich um eine weitere wichtige statische Systemeigenschaft. Um die Instanzmatrix nicht zu überfrachten, wird diese in eine andere Sicht ausgelagert. Auch hier ist wieder unser Ziel durch eine ansprechende Visualisierung diese Eigenschaft den Endnutzern verständlich zu machen. Aufgrund der nötigen Fallbetrachtungen würde es den Umfang des Papers sprengen, diese Sicht vollständig vorzustellen. Es folgt daher nur ein erster Einblick.

In der Informationsvisualisierung gibt es eine Reihe von Hierarchievisualisierungen. Neben klassischen Baumvisualisierungen [RT81] exisitieren mit Cone Trees
[RMC91] sogar dreidimensionale Visualisierungsformen. Einen guten Überbick über andere Möglichkeiten der Hierarchievisualisierung bietet Robert [Rob07].

In Roberts Arbeit werden Venn-Diagramme und Cluster-Maps [FSVH03] zur Darstellung von Objekt-Attribut-Relationen verwendet. Diese Ansätze haben gemeinsam, dass sie Objekte in verschiedene Gruppen clustern, indem der betrachtete Weltausschnitt als Draufsicht dargestellt wird. In diesen Ansätzen werden die Objekte als Kreise dargestellt. In einem Venn-Diagramm besitzt ein Objekt alle Eigenschaften der es überdeckenden Regionen.

Auch bei dieser Sicht nutzen wir aus, dass wir die Objekte bildlich darstellen können. Grundgerüst ist ein Venn-Diagramm. Statt Kreise verwenden wir das Bild einer Instanz der Klasse. Damit findet sich ein Betrachter schneller zurecht, da sich die einzelnen Gruppen visuell stark unterscheiden und gut abgrenzbar sind. Unsere Sicht trägt den Namen *Vogelperspektive*, da wir die Objekte aus den vorherigen Gründen gut überblicken könnnen. Für jede Klasse im Klassendiagramm wird eine Menge angelegt. Ist eine Klasse A Spezialisierung der Klasse B, so ist $A \subset B$. Auch Mehrfachvererbung ist möglich, wird aber aus Platzgtründen hier nicht weiter behandelt. Abbildung 6 zeigt ein UML-Diagramm und die zugehörige Vogelperspektive. Wir plazieren nun in A ausschließlich Instanzbilder von A und in $B \backslash A$ ausschließlich Instanzen von B. Betrachtet der Beobachter eine Instanz, weiß er, dass diese Instanz alle Eigenschaften der umrandeten Regionen besitzt. Auf diese Weise lassen sich auch abstrakte Klassen darstellen. Wäre B eine abstrakte Klasse, darf in $B \backslash A$ kein Instanzbild von B eingefügt werden. Der Betrachter kann keine Instanzen von B vorfinden, da diese nicht exisiteren dürfen.

4 Evaluation und Ausblick

Bereits während der Entwicklung der verschiedenen Sichten wurden stets informatikfremde Probanden in deren Konzeption mit einbezogen, um möglichst geeignete Darstellungen zu entwerfen. Für die Instanzmatrix wurde bereits ein Prototyp in Java entwickelt und ersten Endnutzern mit positiven Feedback gezeigt. Dies ersetzt natürlich nicht größere Evaluationsstudien, welche in Planung sind. Auch müssen die Konzepte in komplexeren Projekten bezüglich ihrer Eignung evaluiert werden. Die Visualisierung für die Statik muss noch um eine Darstellung für die Teil-Ganzes-Beziehung ergänzt werden. Zu einem ganz-

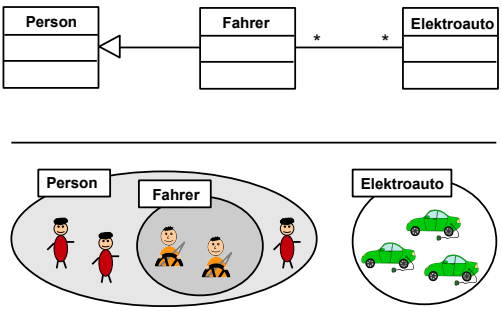

Abbildung 6: Klassendiagramm und abgeleitete Vogelperspektive

heitlichen Systementwurf werden zusätzlich Konzepte für die funktionsorientierten Anforderungen und Dynamik benötigt. Hier ist es geplant endnutzertaugliche Darstellungen mit Use-Case-Sytnax zu entwerfen. Für dynamische Aspekte wird evaluiert, ob sich eine Kopplung mit der eGPM [BH12] eignet. Abschließend muss für die automatische Generierung der Darstellungen eine Komponente entwickelt werden, die Fachbegriffen ein Bild zuordnet.

Literatur

[Ber67] J. Bertin. *Sémiologie Graphique. Les diagrammes, les réseaux, les cartes*. Editions de l'Ecole des Hautes Etudes en Sciences, 1967.

[BH12] H. Breitling und S. Hofer. Schwerpunkt-beispielhaft gut modelliert: Exemplarische Geschäftsprozessmodellierung in der Praxis. *Objekt Spektrum*, (6):8, 2012.

[BNT02] R. Biddle, J. Noble und E. Tempero. Reflections on CRC cards and OO design. In *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, CRPIT '02, Seiten 201–205, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[Boo86] G. Booch. Object-oriented development. *Software Engineering, IEEE Transactions on*, SE-12(2):211–221, 1986.

[CLC03] D. Cohen, M. Lindvall und P. Costa. Agile software development. *Data & Analysis Center for Software (DACS), New York*, 2003.

[CN95] L. Chung und B. A. Nixon. Dealing with non-functional requirements: three experimental studies of a process-oriented approach. In *Proceedings of the 17th international conference on Software engineering*, ICSE '95, Seiten 25–37, New York, NY, USA, 1995. ACM.

[Das07] V.V. Das. Involvement of Users in Software Requirement Engineering. In *Information Technology, (ICIT 2007). 10th International Conference on*, Seiten 230–233, 2007.

[DP06] B. Dobing und J. Parsons. How UML is used. *Commun. ACM*, 49(5):109–113, Mai 2006.

[Erf11] I. Erfurth. *Eine Methode zur Anforderungsermittlung und Modellierung von Individualsoftware in Kooperation mit Fachexperten*. Dissertation, Friedrich-Schiller-Universität, 2011.

[FSVH03] C. Fluit, M. Sabou und F. Van Harmelen. Ontology-based information visualization. *Visualizing the semantic web*, Seiten 36–48, 2003.

[Gro11] Object Management Group. Unified Modeling Language (UML) 2.4.1, 2011. Zuletzt besucht am 10.04.2013.

[HF06] N. Henry und J. Fekete. MatrixExplorer: a Dual-Representation System to Explore Social Networks. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):677–684, 2006.

[KKLK05] S. Kujala, M. Kauppinen, L. Lehtola und T. Kojo. The Role of User Involvement in Requirements Quality and Project Success. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE '05, Seiten 75–84, Washington, DC, USA, 2005. IEEE Computer Society.

[Laf96] D. Lafrenière. CUTA: a simple, practical, low-cost approach to task analysis. *interactions*, 3(5):35–39, September 1996.

[MCY99] J. Mylopoulos, L. Chung und E. Yu. From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1):31–37, Januar 1999.

[MRE10] S. Möckel, W. Rossak und I. Erfurth. CARD-PAL - Ein kartenbasierter Ansatz zur Visualisierung und Formalisierung statischer Aspekte während der Anforderungsanalyse. In *Informatiktage 2010 - Fachwissenschaftlicher Informatik Kongress*, Gesellschaft für Informatik vol 9 2010, Seiten 121–124, 2010.

[Mul01] M. J. Muller. Layered participatory analysis: new developments in the CARD technique. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, Seiten 90–97, New York, NY, USA, 2001. ACM.

[MW08] J. Mindock und G. Watney. Integrating System and Software Engineering Through Modeling. In *Aerospace Conference, 2008 IEEE*, Seiten 1–12, 2008.

[Par10] H. Partsch. *Requirements-Engineering systematisch*. Springer Verlag, 2010.

[PD10] B. Preim und R. Dachselt. *Interaktive Systeme Band 1 Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. Springer-Verlag Heidelberg, 2010.

[Rau91] M. Rauterberg. Partizipative Konzepte, Methoden und Techniken zur Optimierung der Softwareentwicklung. *Softwaretechnik-Trends, Vol. 11(1991), No. 3*, Seiten 104–126, 1991.

[RMC91] G. G. Robertson, J. D. Mackinlay und S. K. Card. Cone Trees: animated 3D visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, Seiten 189–194, New York, NY, USA, 1991. ACM.

[Rob07] S. Robert. Information Visualization-Design for Interaction, 2007.

[RT81] E. M. Reingold und J.S. Tilford. Tidier Drawings of Trees. *Software Engineering, IEEE Transactions on*, SE-7(2):223–228, 1981.

[Rup07] C. Rupp. *Requirements-Engineering und Management*. Hanser, 2007.

[SC02] V. F A Santander und J.F.B. Castro. Deriving use cases from organizational modeling. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, Seiten 32–39, 2002.

[Sch98] R. Schwitter. *Kontrolliertes Englisch für Anforderungsspezifikationen*. Dissertation, Universität Zürich, 1998.

[Sto13] N. Stoyanova. Ein automatisiert überprüfbares Qualitätssicherungsmodell für textuelle Anforderungen. *Softwaretechnik-Trends*, 33(1):13–14, Februar 2013.