Eine vergleichende Fallstudie mit CASE-Werkzeugen für formale und semi-formale Beschreibungstechniken

Erik Kamsties[1], Antje von Knethen[2], Jan Philipps[3], Bernhard Schätz[3]

[1]Fraunhofer Institute for Experimental Software Engineering,
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
[2]Department of Computer Science, University of Kaiserslautern,
D-67653 Kaiserslautern, Germany
[3]Department of Computer Science, Technische Universität München,
Arcisstraße 21, D-80290 München, Germany

Abstract

Viele Beschreibungstechniken wurden vorgeschlagen, um die Qualität von Spezifikationen und damit auch von Software zu verbessern. Dieser Bericht beschreibt den Ablauf und die Resultate einer Fallstudie mit acht semi-formalen und formalen Beschreibungstechniken (Focus, Petrinetze, OCTOPUS, ROOM, SCR, SDL, Statecharts, UML) und zugehörigen CASE-Werkzeugen. Das wesentliche Resultat betrifft die Mängel in informellen Anforderungen, die bei der Formalisierung aufdeckt werden; Mehrdeutigkeiten werden im Vergleich zu Unvollständigkeiten und Widersprüchen öfter übersehen. Mehrdeutigkeiten werden durch die Formalisierung eindeutig gemacht, diese unbewußte Lösung führt jedoch vergleichsweise häufiger zu Fehlern. Weiterhin wurden einige Unterschiede zwischen den Beschreibungstechniken selbst identifiziert, was ihre Effizienz hinsichtlich Einarbeitung, Erstellung und Überprüfung einer Spezifikation betrifft.

1 Einleitung

Software wird zunehmend in Produkten des Alltags wie Videorecordern und Waschmaschinen eingesetzt, aber auch im Automobilbereich, in der Luft- und Raumfahrt, in der Telekommunikation und in Medizintechnik. Mehr und mehr Funktionalität solcher Systeme wird nicht mehr durch Hardware, sondern durch Software realisiert.

Korrektheit und Zuverlässigkeit sind die entscheidenden Qualitätsmerkmale solcher Systeme. Ein Fehlverhalten der Software eines Flugzeugs kann zu lebensbedrohenden Situationen für die Passagiere führen; Fehlverhalten in Telekommunikationssoftware kann zu Qualitätsverlusten bei oder Ausfall von Diensten führen.

Formale Beschreibungstechniken ermöglichen es, Systeme präzise zu beschreiben und bieten damit die Grundlage für mathematische Verfahren zum Nachweis von Korrektheit und Zuverlässigkeit von Software. Semi-formale Beschreibungstechniken fehlt die formale Semantik, sie sind dafür oft leichter erlernbar. Beide Arten von Beschreibungstechniken offerieren einen Symbolvorrat mit eindeutiger Semantik, und helfen somit Mehrdeutigkeiten der natürlichen Sprache ganz zu vermeiden bzw. einzuschränken. Für den industriellen Einsatz solcher Beschreibungstechniken sind CASE-Werkzeuge unerläßlich. Das herausragendes Merkmal dieser Werkzeuge ist die Unterstützung von Aktivitäten, die ohne Werkzeug nicht in dem Umfang möglich wären, z.B. Konsistenzüberprüfung, oder Simulation. In einigen Fällen können sogar Systemeigenschaften formal verifiziert werden. Außerdem bieten Werkzeuge oft partielle oder vollständige Kodegenerierung, was Entwicklungszyklen verkürzt, und unterstützen die Verwaltung von Spezifikationen.

Ein breites Spektrum von Beschreibungstechniken steht dem Entwickler zur Verfügung. Dieser Bericht beschreibt eine Fallstudie mit acht Beschreibungstechniken und den dazugehörigen CASE-Werkzeugen. Konkret wurden als formale Beschreibungstechniken Focus [BDD+92],

Petrinetze [Rei85], SCR [HJL96], und SDL [ITU93], sowie als semi-formale Beschreibungstechniken OCTOPUS [AKZ96], ROOM [SGW94], Statecharts [Har87], und UML [Rat97] gewählt. Es wurden die folgenden CASE-Werkzeuge benutzt: AutoFocus [HMR+98], PEP [Gra97], SCR* [HBGL95], ObjectGEODE (Verilog), Software through Pictures/OMT (Aonix), ObjecTime Developer (ObjecTime Systems), Statemate MAGNUM und Rhapsody (beide I-Logix).
Ziel der Fallstudie war es, den Grad der Unterstützung von Aktivitäten der Anforderungsspezifikationsphase vergleichend[1] zu bewerten. Die beiden betrachten Aktivitäten waren die Erstellung einer Spezifikation und deren Überprüfung; die Anforderungen selbst waren bereits vorgegeben. Außerdem wurde auch noch der Aufwand für die Einarbeitung in die Beschreibungstechnik und das Werkzeug bewertet.
Die Fallstudie wurde im Rahmen eines Seminars durchgeführt, daß zeitgleich an der Universität Kaiserslautern und der Technischen Universität München stattfand. Insgesamt nahmen 22 Studenten an der Fallstudie teil, die über 13 Wochen lief. Als Spezifikationsaufgabe wurden textuelle Anforderungen an ein Tamagotchi vorgegeben (das Tamagotchi ist ein eingebettetes System, daß ein virtuelles Lebewesen simuliert).

2 Aufbau der Fallstudie

Ziel der Fallstudie war es, den Grad der Unterstützung von Aktivitäten der Anforderungsspezifikationsphase vergleichend zu bewerten. In der Fallstudie wurden dazu drei Aspekte untersucht, die Einarbeitung, die Unterstützung bei der Erstellung und bei der Überprüfung der Spezifikation. Die folgenden Fragestellungen dabei konkret untersucht.

- Einarbeitung
  - Wie leicht kann sich ein Entwickler in die Beschreibungstechnik und das Werkzeug einarbeiten? Hierzu wurde der Aufwand und Qualität der Dokumentation bewertet.
- Erstellung einer Spezifikation
  - Wie gut wird Teamarbeit durch Beschreibungstechnik und Werkzeug unterstützt?
  - Wieviele und welche Arten von Mängeln in informellen Anforderungen werden durch die Erstellung einer Spezifikation aufgedeckt?
  - Wie groß ist die entstandene Spezifikation?
  - Wie hoch ist der Aufwand und was sind Probleme bei der Erstellung einer Spezifikation?
- Überprüfung einer Spezifikation
  - Wie umfassend ist die automatische Überprüfung einer Spezifikation?
  - Wie ausführlich ist die Simulation bzw. wie umfangreich ist die Kodegenerierung?
  - Wie umfassend ist die Generierung von Dokumentation für Reviews?

Weitere interessante Fragestellungen würden sich im Zusammenhang mit der weiteren Verwendung der Spezifikationen für Implementierung und als Basis für Test, Verifikation und Wartung ergeben. Diese Aktivitäten lagen aber außerhalb des zeitlichen Rahmens der Studie.
Der Aspekt der Benutzerfreundlichkeit (Useability) der Werkzeuge wurde bewußt ausgelassen, da dies schwer ist auch nur annähernd objektiv zu bewerten.
Es wurden Beschreibungstechniken ausgewählt, die sich zur Spezifikation reaktiver und verteilter Systeme eignen. Die Auswahl von konkreten Techniken sollte ein möglichst weites Spektrum umfassen, war aber durch die Verfügbarkeit von Werkzeugen und Know-how eingeschränkt. Jedem Team sollte ein Experte zur Verfügen stehen, um über Anfangsschwierigkeiten

---

1. Vergleichend heißt, daß keine absolute quantitative Bewertung (z.B. “sehr gut” bis “mangelhaft”), sondern eine Bewertung der Beschreibungstechniken und Werkzeuge relativ zueinander in qualitativer Weise erfolgt.

mit der Technik und insbesondere dem Werkzeug zu helfen, denn sonst hätte der enge zeitliche Rahmen nicht eingehalten werden können. Die Wahl fiel aus diesen Gründen auf Focus [BDD+92], Petrinetze [Rei85], SCR [HJL96], und SDL [ITU93] als formale Beschreibungstechniken, sowie OCTOPUS [AKZ96], ROOM [SGW94], Statemate [Har87], und UML [Rat97] als semi-formale Beschreibungstechniken.

Die Studenten erhielten zu Beginn der Fallstudie eine Problembeschreibung (4 Seiten) und Anforderungsdokument (5 Seiten) für das Tamagotchi. Das Anforderungsdokument enthielt 42 funktionalen Anforderungen und war in natürlicher Sprache formuliert (deutsch). Jede Anforderung umfaßte durchschnittlich 2 bis 3 Sätze. Die Anforderungen wurden zum Teil einem Buch über das Tamagotchi entnommen [Ban97]. Einige Fehler (Mehrdeutigkeiten, Unvollständigkeiten und Widersprüche) waren bewußt in das Dokument eingebaut worden, der Rest war auf natürliche Weise in das Dokument gelangt und trat im Laufe der Fallstudie zutage.

Das Tamagotchi ist ein reaktives System, daß das Leben eines Kükens auf einem LCD-Display simuliert. Das Küken hat eine Reihe von Entwicklungsphasen (Ei, Baby, Kleinkind, Jugendlicher und Erwachsener). Der Benutzer muß sich um die Pflege und Aufzucht des Kükens kümmern, d.h. muß es z.B. füttern, oder mit ihm spielen. Das Verhalten und das maximal erreichbare Lebensalter des Kükens hängt von der Art der Pflege ab, z.B. entwickelt sich ein vernachlässigtes Kleinkind zu einem aufmerksamkeitssüchtigen Erwachsenen mit geringer Lebenserwartung.

Die Fallstudie lief in drei Schritten ab: Einarbeitung, Spezifikation, Review und Überarbeitung.

1. Einarbeitung in Modellierungstechnik und Werkzeug (4 Wochen)
   Die Studenten erhielten eine kurze Anforderungsbeschreibung einer Notabschaltung eines Reaktors als Trainingsbeispiel und Literatur zur Technik und Handbücher zum Werkzeug. Die Aufgabe bestand darin, die Literatur durchzuarbeiten und eine Spezifikation des Trainingsbeispiels mit Hilfe des Werkzeugs zu erstellen.
2. Spezifikation und Simulation des Tamagotchis (5 Wochen)
   Die Studenten erhielten das Anforderungsdokument für das Tamagotchi. Zunächst wurden die Anforderungen unter den Teammitgliedern aufgeteilt und ein Arbeitsplan für die Erstellung der Spezifikation aufgestellt. Beides wurde dann mit dem Betreuer abgestimmt. Anschließend wurde die Spezifikation geschrieben. Schritthaltend mit dem Spezifizieren wurde auch simuliert, sofern das Werkzeug dies zulies, um frühzeitig Fehler zu eliminieren.
3. Review und Rework (3 Wochen)
   Die Teams tauschten nach der vorläufigen Fertigstellung der Spezifikation diese paarweise untereinander aus. Die Spezifikation wurden zunächst inspiziert und anschließend in einem Meeting diskutiert. Außerdem wurden im Meeting die Spezifikationen auf Basis von Fragen stichprobenartig simuliert, die in der Inspektion aufgetreten waren.

3 Resultate

Die folgenden Ergebnisse wurden aus Fragebögen, den schriftlichen Ausarbeitungen und Vorträgen der Teams und abschließenden Interviews entnommen. Die folgende Präsentation der Ergebnisse ist im Aufbau an den Fragestellungen aus Kapitel 2 orientiert.

3. 1 Einarbeitung

Der Aufwand für die Einarbeitungsphase betrug zwischen 20 und 70 Stunden. Tabelle 1 zeigt die Aufwandsverteilung über die verschiedenen Beschreibungstechniken. Im Aufwand enthalten ist das Lesen der Literatur und Manuals, sowie die Erstellung der Trainingsspezifikation. Der Aufwand ist über die Teammitglieder summiert. Die Aufwandsdifferenzen entstanden im wesentlichen durch den Umfang der Literatur bzw. Handbücher, die bei den kommerziellen

Werkzeugen (ObjectGEODE, ObjecTime Developer, Rhapsody, Statemate MAGNUM, StP/ OMT) natürlich umfangreicher ausfällt als bei Forschungsprototypen (AutoFocus, PEP, SCR). Petrinetze und SCR wurden aufgrund ihrer wenigen Sprachkonstrukte als leicht erlernbar empfunden. Der Einarbeitungsaufwand des Statemate-Teams ist deswegen so gering, weil zur Modellierung der Reaktornotabschaltung ein State-chart ausreichend war. Die komplette Statemate-Vorgehensweise mit Activity- und Module-charts wurde erst in der Spezifikationsphase gelesen und verstanden. Der niedrige Aufwand für Rhapsody erklärt sich durch Vorkenntnisse eines Teammitglieds (hatte an Rhapsody-Schulung bei Berner&Mattner teilgenommen).

Focus AutoFocus	Petrinetze PEP	Octopus StP/omt	ROOM ObjecTime	SCR SCR*	SDL ObjectGEODE	Statecharts Statemate	UML Rhapsody
20	70	60	50	40	70	30	40

Tabelle 1: Aufwand für Einarbeitungsphase (Stunden)

Die Qualität der Dokumentation ist erwartungsgemäß bei den kommerziellen Werkzeugen sehr gut. Die Dokumentation zum Statemate-Ansatz beispielsweise ist zwar umfangreich (ca. 300 Seiten), ließ aber keine Fragen offen.

3. 2 Spezifikation

Teamarbeit. Es konnten nur vier der acht Teams tatsächlich arbeitsteilig arbeiten. Für ObjecTime, ObjectGEODE und Rhapsody stand nur eine Lizenz bzw. ein Rechnerarbeitsplatz zur Verfügung, sodas diese Teams das Tamagotchi gemeinsam modelliert haben. Zu diesen Beschreibungstechniken und Werkzeugen können wir daher keine Aussagen zur Teamarbeitfähigkeit treffen.
Beschreibungstechniken bieten oft Konstrukte an, die sich für Arbeitsteilung nutzen lassen. Im folgenden sind die Erfahrungen mit diesen Konstrukten beschrieben. AutoFocus unterstützt die Strukturierung der Spezifikationen mittels Hierarchisierung auf allen wesentlichen Arten seiner Beschreibungstechniken, also sowohl bei der Aufspaltung des Systems in Komponenten als auch bei der Beschreibung des Verhaltens (vom System oder einer Komponente) durch Beschreibung eines Zustands durch einen ganzen Automaten; ähnliches gilt für die Beschreibung der Szenarien. PEP erlaubt ebenfalls die Strukturierung der Spezifikation mittels hierarchischer Elemente; durch die Ausblendung bestimmter Bezüge bei der Verwendung hierarchischer Elemente geht jedoch auf Darstellungsebene ein Teil der Spezifikationsinformation verloren. OCTOPUS bietet das Konzept der Subsysteme, die die Aufteilung eines Systems in seine wesentlichen strukturellen Bestandteile beschreiben. Bei der geringen Größe des Tamagotchis konnte nur aber ein sinnvolles Subsystem identifiziert werden. Das Verhalten eines Subsystems wird -im Gegensatz zu UML/Rhapsody- unabhängig von dessen Klassenstruktur durch Statecharts beschrieben. Arbeitsteilung ist bei OCTOPUS daher primär auf der Subsystemebene möglich. Innerhalb eines Subsystems gibt es keine Möglichkeit mehr die weitere Zerlegung in Statecharts und deren Interaktionen zu beschreiben. Daher wurde bei OCTOPUS auf Arbeitsteilung verzichtet. Der SCR Notation fehlt ein Konzept zur Kapselung von Spezifikationsteilen, was die Teamarbeit stark unterstützt hätte. Statemate bietet das Konzept der Activities mit dessen Hilfe das Tamagotchi in unabhängig bearbeitbare Teile aufgeteilt wurde.
Teamarbeit muß auch durch das verwendete CASE-Werkzeug unterstützt werden. AutoFocus unterstützt durch die Verwendung eines Versionsverwaltungssystems mit Mehrbenutzerprinzip die Entwicklung im Team im wesentlichen sehr gut (im Serverbetrieb); in der aktuellen Version

wurde jedoch eine geeignete Import/Export-Schnittstelle für den Offline-Betrieb vermißt. PEP bietet zwar prinzipiell Mechanismen zur Modularisierung von Spezifikationen, jedoch sind diese auch auf der werkzeugtechnischen Ebene nicht ausreichend realisiert. Während auf der einen Seite des verteilte Entwicklung und Simulieren prinzipiell gut möglich ist, stellte das Werkzeug bei der Zusammenführung der Spezifikationen einen wesentlichen Engpaß bei der Entwicklung im Team dar. Das SCR* Werkzeug sieht Teamarbeit nicht vor, es erlaubt weder die Aufteilung der Spezifikation in unabhängig bearbeitbare Teile noch die Zusammenführung getrennt entwickelter Spezifikationen. Statemate unterstützte die Teamarbeit vollständig. Die Spezifikation konnte arbeitsteilig entwickelt und simuliert werden, nachdem der Informationsaustausch zwischen den einzelnen Teilen mit Hilfe eines Activity-Charts definiert war.

Aufdeckung von Mängeln in informellen Anforderungen. Es stellen sich zwei Fragen im Zusammenhang mit den Mängeln, die in den informellen Anforderungen enthalten waren:

- Gibt es Unterschiede zwischen den *Beschreibungstechniken* hinsichtlich der *Anzahl von Mängeln insgesamt* (d.h. *über die Arten von Mängeln summiert*), die beim Spezifizieren aufgedeckt/übersehen werden?
- Gibt es Unterschiede zwischen den *Arten von Mängeln* hinsichtlich der *Anzahl entsprechender Mängel* (d.h. *über die Beschreibungstechniken summiert*), die beim Spezifizieren aufgedeckt/übersehen werden?

Wir haben die Mängel in den informellen Anforderungen durch den Spezifikationsprozeß in die Spezifikation nachverfolgt und dabei vier Fälle identifiziert, wie Mängel propagiert werden. Der Mangel

- lag außerhalb der Grenzen des Models, d.h. ein Aspekt des Systems wurde nicht modelliert, weil dieser nicht, oder nur aufwendig modellierbar war; der Mangel blieb unerkannt.
- wurde erkannt, gemeldet und anschließend in der Spezifikation korrigiert.
- wurde nicht erkannt und in die Spezifikation übernommen, z.B. blieb eine Unvollständigkeit in den informellen Anforderungen eine Unvollständigkeit in der Spezifikation.
- wurde nicht als solcher gesehen und daher bewußt oder unbewußt gelöst, z.B. wurde eine fehlende Information aus Analogien oder Domänenwissen abgeleitet; in der Spezifikation findet sich eine Lösung des Mangels.

Im letzten Fall haben wir noch zusätzlich untersucht, ob die bewußte/unbewußte Lösung des Mangels auch tatsächlich zu einem Fehler in der Spezifikation führte, oder nicht.Es ist nämlich anzunehmen, daß bedingt durch die subjektive Interpretation der natürlichen Sprache, auch Mängel nicht eindeutig sind, d.h. was für den einen Leser Fragen aufwirft ist für den anderen Leser eindeutig. Ein bewußt/unbewußt gelöster Mangel muß daher nicht notwendigerweise zu einem Fehler in der Spezifikation führen.

Die Tabelle 2 zeigt die Propagierung der Mängel nach Art und Beschreibungstechnik aufgeschlüsselt. Die Tabelle ist zweigeteilt, da das OCTOPUS-, SCR-, Statechart- und UML-Team die bemerkten Mängel mit dem Betreuer diskutiert haben, das Focus-, Petrinetz-, ROOM- und SDL-Team hingegen diese eigenständig gelöst haben. Die Summen für die Meldungen von Fehlern und für die in die Spezifikation propagierten Fehler (vorletzte und letzte Spalte) sind die primär interessierenden Größen. Die Summe der Meldungen von Mängeln zeigen einen Unterschied zwischen OCTOPUS und UML auf der einen Seite, sowie Statecharts und SCR auf der anderen Seite. Da die Beschreibungstechniken große Ähnlichkeiten -abgesehen vom Paradigma- aufweisen, muß dies andere Ursachen haben. Die Aufschlüsselung für die einzelnen Arten von Fehlern zeigt, daß das OCTOPUS-Team und das UML-Team die Anforderungen öfter unbewußt bzw. selbständig interpretiert haben, während das SCR- und das Statemate-Team mehr nachgefragt haben. Die Summe der Mängel, die beim Spezifizieren propagiert wurden, d.h. auch in der Spezifikation enthalten sind, ist bei allen Teams ähnlich. Im Vergleich dazu enthalten die Focus-, Petrinetz-, ROOM- und SDL-Spezifikation durchschnittlich 2.25 propagierte Fehler mehr. Unter Berücksichtigung der Tatsache, daß diese Teams Mängel eigenständig gelöst haben, ist dies ist ein erstaunlich gutes Ergebnis, was darauf hindeutet, daß die Mängel in den textuellen Anforderungen oft trivial waren und sich aus ähnlichen Anforderungen, oder Allgemeinwissen ableiten ließen. Im Endeffekt sind keine signifikanten Differenzen zwischen den Beschreibungstechniken hinsichtlich der Progagierung von Mängeln bemerkbar gewesen.

Beschreibungs-technik	Unvollständigkeit (19)			Mehrdeutigkeit (13)			Widerspruch (5)			außerhalb des Modells	Summe Meldungen	Summe Fehler
	gem.	übern.	gelöst	gem.	übern.	gelöst	gem.	übern.	gelöst			
OCTOPUS	4	5	9 / 0	2	0	8 / 2	2	1	2 / 0	1	8	8
SCR	11	3	3 / 0	3	0	5 / 4	2	1	2 / 0	3	16	8
Statecharts	8	4	7 / 0	3	0	5 / 4	3	1	1 / 0	0	14	9
UML	3	3	8 / 4	4	0	6 / 3	1	1	2 / 1	0	8	12
Focus	-	2	8 / 5	-	0	8 / 4	-	1	2 / 2	5	-	14
Petrinetze	-	2	9 / 4	-	0	7 / 3	-	0	1 / 0	11	-	9
ROOM	-	3	12 / 2	-	0	7 / 5	-	1	3 / 1	3	-	12
SDL	-	4	12 / 3	-	0	10 / 3	-	1	4 / 0	0	-	11

Tabelle 2: Propagierung von Mängeln nach Beschreibungstechniken

Die zweite Frage ist, ob es Differenzen zwischen den Arten der Mängel hinsichtlich der Propagierung von Mängeln gibt. Die Tabelle 3 zeigt die Propagierung von Mängeln nach den unterschiedlichen Arten von Mängeln aufgeschlüsselt. Unter UKL-Techniken sind dort OCTOPUS, SCR, Statecharts und UML subsumiert, d.h. die Teams die Mängel gemeldet haben, und unter TUM-Techniken sind Focus, Petrinetze, ROOM und SDL subsumiert, bei denen diese Information fehlt. Die Prozentangaben beziehen sich daher auf jeweils 4 Teams, d.h. wenn 34% der Unvollständigkeiten gemeldet worden sind, bedeutet dies, daß die Mehrdeutigkeiten im Mittel von jeweils 1.36 der 4 Teams gemeldet worden sind. Die Prozentangaben summieren sich jeweils zu 100%.

Mangel	Unvollständigkeit (19)		Mehrdeutigkeit (13)		Widerspruch (5)	
	UKL	TUM	UKL	TUM	UKL	TUM
# nicht gemeldete Mängel	0	-	3	-	1	-
% außerhalb des Models	3	13	4	9	0	20
% Meldungen	34	-	23	-	40	-
% übernommene Mängel	21	15	0	0	20	15
% richtig gelöste Mängel	37	54	46	62	35	50
% falsch gelöste Mängel	5	18	27	29	5	15

Tabelle 3: Propagierung von Mängeln nach Arten

Die Anzahl der gar nicht gemeldeten Mängel muß in Relation zur Gesamtanzahl (in Klammern in der Überschrift der Tabelle nochmals angegeben) interpretiert werden; Mehrdeutigkeiten wurden öfter völlig übersehen (d.h. von keinem Team gemeldet), als Widersprüche oder Unvollständigkeiten. Die Mehrdeutigkeiten wurden während des Spezifizierens auch prozentual gesehen am wenigsten gemeldet, die Widersprüche waren am auffälligsten. Unvollständigkeiten und Widersprüche wurden direkt in die Spezifikation übernommen, wenn sie übersehen wurden, Mehrdeutigkeiten hingegen gar nicht. Sie wurden *in allen Fällen* zu “Eindeutigkeiten”, d.h. die ursprüngliche Mehrdeutigkeit kann nicht mehr als solche auffallen. In Bezug auf die UKL-Teams wurden Mehrdeutigkeiten zwar zu 46% bewußt/unbewußt korrekt gelöst, aber auch zu 27% falsch. Im Vergleich wurden Unvollständigkeiten zu 37%/5% korrekt bzw. falsch gelöst und Widersprüche zu 35%/5%. Die prozentuale Verteilung ist bei den TUM-Teams verschoben, da hier Mängel nicht gemeldet wurden. Der Anteil der Mängel, die übernommen wurden, ist bei den UKL-/TUM-Teams trotzdem in etwa gleich. Der Anteil der falsch gelösten Mängel ist -wie zu erwarten war- höher, mit Ausnahme der Mehrdeutigkeiten, die keinen Unterschied aufweisen. Wir schließen daraus, daß die Möglichkeit, Fragen zu den Anforderungen zu stellen, hilft Fehler bei der Behebung von Unvollständigkeiten zu vermeiden - nicht jedoch im Fall von Mehrdeutigkeiten. Fallen letztere auf, dann ist ihre Lösung mit Hilfe von Kontextwissen scheinbar offensichtlich. Fallen sie jedoch nicht auf, dann ist die Gefahr von Fehlinterpretationen bei Mehrdeutigkeiten wesentlich größer, als bei Unvollständigkeiten oder Widersprüchen.

Größe der Spezifikation. Die Seitenanzahl der ausgedruckten Spezifikation gibt nur eine grobe Annäherung, da durch das automatisches Seitenlayout der Informationsgehalt von Seite zu Seite stark schwankt.Die Tabelle 4 zeigt die Größe der einzelnen Spezifikationen. Einige Werkzeuge erfordern die Beschreibung bestimmter Aspekte direkt in C++ Kode, in diesen Fällen ist zusätzlich die Zeilenanzahl des gesamten (einschließlich des generierten) Kodes angegeben.

Focus AutoFocus	Petrinetze PEP	Octopus StP/omt	ROOM ObjecTime	SCR SCR*	SDL ObjectGEODE	Statecharts Statemate	UML Rhapsody
25	20	20	84 (33)	36	136	40	(23)
-	-	-	5.5	-	-	-	16

Tabelle 4: Größe der Spezifikationen (Seitenanzahl, #KLOC)

Die Größe der Spezifikationen ist von der Beherrschung der Beschreibungstechnik und ihrer Darstellungseffizienz abhängig. Die Größe (Darstellungseffizienz) der Spezifikationen in Focus, Petrinetzen, Octopus, SCR, und Statecharts relativ einheitlich. Die ROOM und UML-Spezifikationen sind umfangreicher, da hier ein Teil graphisch (Seitenanzahl in Klammern gesetzt), und der andere Teil textuell als C++ Kode beschrieben wird. Die UML-Spezifikation ist von Kode her umfangreicher (16 KLOC), da hier noch ein graphisches Simulationsfrontend enthalten ist. Die Größe der SDL Spezifikation ist relativ hoch, was darauf zurückzuführen ist, daß Fallunterscheidungen, in SDL ausgedrückt, viel Platz benötigen und das automatische Seitenlayout nicht optimal ist.

Aufwand. Der Aufwand für die Spezikationsphase betrug zwischen 100 und 230 Stunden. Tabelle 5 zeigt die Aufwandsverteilung über die verschiedenen Beschreibungstechniken. Im Aufwand enthalten ist nur die Erstellung der Tamagotchi-Spezifikation mit dem Werkzeug. Der Aufwand ist über die Teammitglieder summiert. Die Aufwände für Focus, Petrinetzen, SCR, und Statecharts können nicht direkt mit den Aufwänden von OCTOPUS, ROOM, SDL und

UML verglichen werden. Bei den Teams, die die Spezifikation aufgeteilt haben, ist zusätzlicher Koordinierungsaufwand entstanden. Bei den Teams, die die Spezifikation gemeinsam erstellt haben, hätte im Grunde ein Student die Spezifikation auch alleine erstellen können, d.h. der Aufwand würde dann nur ein Drittel bzw. die Hälfte (je nach Teamgröße) betragen. Teilt man beispielsweise den Aufwand für UML durch die Teamgröße ergibt sich ein Aufwand von 80 Stunden; zieht man bei Statemate einen gewissen Koordinierungsaufwand ab, kommt man auf einen sehr ähnlichen Wert.

Focus AutoFocus	Petrinetze PEP	Octopus StP/omt	ROOM ObjecTime	SCR SCR*	SDL ObjectGEODE	Statecharts Statemate	UML Rhapsody
140	110	100*	110*	100	210 *	100	230 *

Tabelle 5: Aufwand für Spezifikationphase (Stunden, *=keine Arbeitsteilung)

Fazit ist, daß keine signifikanten Unterschiede im Aufwand zwischen den verschiedenen Beschreibungstechniken und Werkzeugen zu verzeichnen waren, mit Ausnahme von ROOM. ROOM hätte arbeitsteilig einen Aufwand von ca. 35 Stunden + Koordinierungsaufwand benötigt, was zum Teil daran liegt, daß große Teile der Funktionalität des Tamagotchi durch C++ - Kodefragemente in den Transitionen realisiert sind; im Umgang mit CASE-Werkzeugen unerfahrenen Studenten fällt dies leichter als alle Möglichkeiten der eigentlichen Modellierungssprachen auszuschöpfen. Beim Einsatz der Werkzeuge waren teilweise Probleme zu verzeichnen, die dazu geführt haben, daß der Aufwand nicht optimal war (AutoFocus, SCR).

3. 3 Überprüfung

Automatische Überprüfung. Alle eingesetzten Werkzeuge bieten Konsistenzprüfung an, einige erlauben zusätzlich noch an Ankopplung eines Model Checkers. Der Konsistenzprüfer von StP/OMT konnte nicht benutzt werden, da sich OCTOPUS von OMT darin unterscheidet, daß Statecharts nicht Klassen sondern Subsystemen zugeordnet sind. Das SCR*-Werkzeug konnte der gegenseitige Ausschluß von Events (wichtig für Determinismus von Zustandsübergängen) nicht in allen Fällen überprüfen. Die Tamagotchi-Spezifikation hat eine kontinuierliche Eingangsgröße, nämlich die Zeit, sodas Modelchecking grundsätzlich nicht in Frage kam.

Simulation und Kodegenerierung. Mit Ausnahme von StP/OMT gestatten alle Werkzeuge die Simulation von Modellen. Die Simulation der Systemzeit (das Tamagotchi lebt maximal 30 Tage) gestaltete sich zum Teil aufwendig.
Mehrere Werkzeuge bieten außerdem Generatoren für graphische Simulationsfrontends, oder zumindest Schnittstellen für Frontends an. Nur das Statemate-Team hat Gebrauch vom Frontend-Generator gemacht; bemerkenswert war dabei der geringe Erstellungsaufwand.

Generierung von Dokumentation für Reviews. Die eingesetzten Werkzeuge erzeugen aus einer Spezifikation eine oder mehrere PostScript-Dateien, die in Spezifikationsdokumente eingebunden werden können. Einige Werkzeuge können darüber hinaus selbst Spezifikationsdokumente generieren, die mit gängigen Texteditoren (z.B. FrameMaker) weiterbearbeitet werden können.
Grundsätzlich sind semi-formale und formale Spezifikationsdokumente nicht leicht zu reviewen. Als besonders hinderlich erwies es sich, wenn Events, die Transitionen triggern, oder Ausgaben, die an Transitionen oder Zustände gebunden sind, nicht im ausgedruckten Zustandsdiagramm erscheinen. Wenn diese stattdessen in einem Data Dictionary erscheinen, war nur ständiges hin- und herblättern erforderlich. Zum Teil befand sich diese Informationen

aber auch nur im generierten Kode, d.h. dann mußte auch noch die betreffende Stelle im Kode ausfindig gemacht werden. Ein zweites Problem war das oft nur schwer nachvollziehbare Senden und Empfangen von Events zwischen Entitäten, z.B. Klassen (welche Entität produziert/ konsumiert dieses Event?).

4 Limitierungen der Fallstudie

Grundsätzlich stellen sich zwei Fragen zur Limitierung der Fallstudie, zum einen nach der Validität der Ergebnisse selbst (internal validity) und zum anderen nach der Übertragbarkeit der Ergebnisse auf andere Domänen bzw. professionelle Entwickler (external validity).
Die Validität der Ergebnisse sind unserer Ansicht nach aus mehreren Gründen hoch. Die Studenten zeigten eine kontinuierlich hohe Motivation, da die meisten Teams mit professionellen CASE-Werkzeugen arbeiten konnten, die sonst in Universitätsveranstaltungen noch wenig eingesetzt werden. Es stand für die Einarbeitung ausreichend Zeit zur Verfügung; für die Modellierung war die Zeit knapp. Die Studenten haben die Zeitknappheit durch erhöhten Arbeitseinsatz (ca. 8-10 Stunden pro Woche) weitgehend ausgeglichen. Unter der Zeitknappheit hat unserer Ansicht nach nur die Vollständigkeit der Spezifikationen gelitten. Die entstandenen Spezifikationen zeigen, daß die Studenten die Beschreibungstechniken und Werkzeuge gut bis sehr gut verstanden haben. Die Aufwandsdaten sind über die Anzahl der Mitglieder eines Teams summiert, sodas individuelle Leistungsunterschiede zum Teil ausgeglichen werden.
Die Übertragbarkeit der Ergebnisse auf Entwickler aus der Industrie und auf andere Domänen ist unterschiedlich. Die Aufwandsdaten sind auf professionelle Entwickler nicht übertragbar. Viele neu aufgesetzte Projekte müssen jedoch aufgrund der Personalknappheit heutzutage mit Berufseinsteigern und -umsteigern besetzt werden. Für diese Gruppe besitzen die Ergebnisse eine gewisse Gültigkeit. Die Größe der entstandenen Spezifikationen zeigen einen Trend, aber wie schon die Verkleinerung der SCR Spezifikation von 40 auf 25 Seiten zeigt, lassen sich vermutlich alle Spezifikationen ohne Einbußen an Informationsgehalt kürzer fassen. Die Anzahl der Mängel, die während des Spezifizierens aufgedeckt wurden, zeigt ebenfalls einen Trend für unerfahrende Entwickler. Die Größe der Spezifikation und damit auch der Erstellungsaufwand hängen von der jeweiligen Domäne ab. Das Tamagotchi ist ein einfaches eingebettetes System, für das alle Beschreibungstechniken gleichermaßen geeignet waren. Für spezifischere Domänen (z.B. Telekommunikation) differiert die Eignung der Beschreibungstechniken vermutlich stärker und damit auch die Größe der Spezifikationen und der Aufwand.

5 Zusammenfassung

Wir haben acht Beschreibungstechniken und zugehörige CASE-Werkzeuge im Rahmen einer Fallstudie miteinander hinsichtlich Einarbeitungsaufwand und Unterstützung bei der Erstellung und Überprüfung einer Spezifikation verglichen.
Das wesentliche Ergebnis ist, daß Mängel, die typischerweise in informellen Anforderungen enthalten sind, je nach Typ unterschiedlich gut bei der Formalisierung aufgedeckt werden. Mehrdeutigkeiten werden öfter übersehen und dann auch häufiger unbewußt *falsch* interpretiert, als im Vergleich Unvollständigkeiten oder Widersprüche.
Zwischen den Beschreibungstechniken und CASE-Werkzeugen konnten wir nur leichte Unterschiede feststellen. Petrinetze und SCR wurden subjektiv als leicht erlernbar empfunden, aber der Einarbeitungsaufwand für die Werkzeuge hat diesen leichten Vorteil wieder relativiert. Unterschiede konnten auch in der Unterstützung von Teamarbeit auf Seiten der Beschreibungstechnik wie auch der CASE-Werkzeuge festgestellt werden. Die kommerziellen Werkzeuge - wie z.B. Statemate- schnitten in diesem Punkt natürlich deutlich besser ab. Die deutlichsten Un-

terschiede zeigten sich in der Länge der entstandenen Spezifikationen, die zwischen 20 und 100 Seiten variierte - bei vergleichbarer Präzision und Vollständigkeit. Im Erstellungsaufwand schnitt ROOM etwas besser ab, was darauf zurückzuführen ist, daß größere Teile der Spezifikation direkt in C++ beschrieben wurden.

In der ersten Modellierungsfallstudie, die mit dem Tamagotchi jedoch ohne CASE-Werkzeuge durchgeführt wurde [KKR+98], zeigten sich größere Differenzen im Aufwand zwischen den Beschreibungstechniken, bedingt durch unterschiedlich reichhaltige Notationen (z.B. SCR vs. UML). Diese Differenzen wurden in dieser Fallstudie durch den Einsatz von CASE-Werkzeugen nivelliert. Der Aufwand stieg durch CASE-Werkzeuge nur um ca. 10-50%, wobei sich aber auch die Qualität der Spezifikationen stark verbesserte. Der Erzeugung leicht reviewbarer Dokumentation sollte bei der Weiterentwicklung von Werkzeugen mehr Aufmerksamkeit geschenkt werden.

6 Literatur

[AKZ96] Maher Awad, Juha Kuusela, and Jurgen Ziegler. *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion*. Prentice Hall, 1996.

[Ban97] Bandai. *Das Original Tamagotchi Buch*. TAMAGOTCHI & BANDAI, 1997.

[BDD+92] M. Broy, C. Dendorfer, F. Dederichs, M. Fuchs, T. Gritzner, and R. Weber. The design of distributed systems - An introduction to Focus. TUM-I 0192, Technical University of Munich, 1992.

[Gra97] Bernd Grahlmann. The PEP Tool. In *Tool Presentations of ATPN'97 (Application and Theory of Petri Nets)*, June 1997.

[Har87] David Harel. Startecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HBGL95] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: A toolset for specifying and analyzing requirements. In *Proceedings of the 10th Annual Conf. on Computer Assurance*, pages 109–122, Gaithersburg, MD, USA, June 1995.

[HJL96] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, July 1996.

[HMR+98] Franz Huber, Sascha Molterer, Andreas Rausch, Bernhard Schätz, Marc Sihling, and Oscar Slotosch. Tool supported specification and simulation of distributed systems. In Bernd Krämer, Naoshi Uchihira, Peter Croll, and Stefano Russo, editors, *Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.

[ITU93] *ITU-T. Recommendation Z.100, Specification and Description Language (SDL)*. ITU, 1993.

[KKR+98] Antje von Knethen, Erik Kamsties, Ralf Reussner, Christian Bunse, and Bin Shen. A comparative case study with industrial requirements engineering methods. In *Proceedings of the 11th International Conference on Software Engineering an its Applications*, Paris, France, December 8–10 1998.

[Rat97] Rational Software Corporation. *Unified Modeling Language*, 1997. Version 1.1, available at http://www.rational.com/uml.

[Rei85] Wolfgang Reisig. *Petri Nets - An Introduction*. Number 4 in EATCS Monograph. Springer Verlag, 1985.

[SGW94] B. Selic, G. Gullekson, and P.T. Ward. *Real–Time Object–Oriented Modeling*. John Wiley & Sons, New York, 1994.