Querschnittsprojekt Q1

Durchführung eines domänen-spezifischen Requirements-Engineering Prozesses

Leiter

Prof. Dr. Gerhard Zimmermann (D1)

Mitarbeiter

Dipl. Inform. Raimund L. Feldmann (A1)

Dipl. Inform. Stefan Vorwieger (A1)

Dipl. Inform. Wolfgang Mahnke (A3)

Dipl. Inform. Jürgen Münch (B1)

Dipl. Inform. Christian Peper (B4)

Dipl. Inform. Martin Becker (B5)

Dipl. Inform. Martin Kronenburg (C1)

Dipl. Inform. Stefan Queins (D1)

Dipl. Ing. Jürgen Schäfer (D2)

Dr. Rolf Merz (D2)

1 Einleitung

In diesem Querschnittsprojekt wurden zwei Fallstudien mit einem Team von Mitarbeitern aus mehreren Teilprojekten durchgeführt, um Messungen zur Effizienz und Produktqualität an größeren Beispielen in einer Gruppe von Entwicklern und Experten durchführen zu können. Die Fallstudien (Case-Studies), die im weiteren CS3/1 und CS3/2 genannt werden, wurden im SE-Labor des SFB 501 mit Unterstützung der Teilprojekte A1 und B1 durchgeführt. Aufgabenstellung, Leitung und Prototyping wurde von dem Teilprojekt D1 übernommen, wobei ein Gebäudesimulator von Teilprojekt C1 als Ergebnis einer früheren Fallstudie zur Verfügung gestellt wurde.

Beide Fallstudien hatten das Ziel, ausgehend von einer Problembeschreibung der Domäne Gebäudeautomation, ausführbare System-Requirements zu erstellen und sie zu validieren und zu verifizieren. Dazu sollte nach einer in D1 entwickelten domänenspezifischen Analysemethode (siehe auch Ergebnisbericht D1, Arbeitsschwerpunkt E) vorgegangen werden und mit von Teilprojekt A1 definierten Messmethoden begleitet werden. Als Notationen wurden in beiden Fällen informelle Beschreibungen für die Problembeschreibung und die ersten Typbeschreibungen, UML für die Objektstruktur und SDL für die ausführbaren Modelle verwandt. CS3/1 wurde als Baseline angesehen, da in dieser Domäne keine Messergebnisse vergleichbarer Analysemethoden gefunden werden konnten. CS3/2 ist eine erste Iteration der verwandten Methode, wobei in CS3/1 erkannte Mängel abgestellt und zusätzliche Verbesserungen erprobt wurden. Dies bezieht sich vor allem auf die konsequente Nutzung von Wiederverwendung einzelner Artifakte und ein geändertes Vorgehensmodell. In CS3/2 musste außerdem die Aufgabenstellung modifiziert werden, da durch die Kenntnisse der Entwickler über den CS3/1-Entwurf die Messungen sonst erheblich verfälscht worden wären. Im Prinzip ist durch diese Maßnahme ein Vergleich der Messdaten zwischen den beiden Fallstudien möglich. Allgemeingültige Aussagen sind beim Vergleich von zwei Fallstudien natürlich nicht möglich, aber ein Trend der Daten konnte festgestellt werden, der sich z.B. bei der Wiederverwendung mit der Erfahrung anderer deckt. Dadurch konnte die Messmethode als richtig angesehen werden.

2 Fallstudie CS3/1

Die Aufgabenstellung von CS3/1 war es, eine Lichtsteuerung für ein Stockwerk in Gebäude 32 der Universität Kaiserslautern, in dem auch das Testfeld des Teilprojekts D1 liegt, zu entwerfen. Diese Aufgabenstellung wurde durch insgesamt 39 Needs notiert, die in drei Gruppen (User-, Facility-Manager-, Nonfunctional-Needs) eingeteilt wurden. Es wurde nicht vorausgesetzt, dass diese Liste vollständig und konsistent war, da dies auch von einem realen Kunden nicht verlangt werden kann. Im Querschnittsprojekt Q4 wurde eine Fallstudie durchgeführt, die diese Eigenschaften garantieren sollte. zusätzlich zu den Needs existierte eine Beschreibung des Gebäudes mit den bereits installierten Sensoren und Aktuatoren. Beides zusammen waren die Ausgangsdokumente der Fallstudie. Die Vorgehen war durch ein Modell beschrieben, das von den Entwicklern strikt eingehalten werden musste. Desweiteren war ein Mess-

und Testplan festgelegt, um Aufwands-, Kalenderzeit- und Fehlerdaten zu erfassen. Als Rechnerunterstützung wurde eine auf RCS basierende Umgebung bereitgestellt.

Insgesamt wurden 25 verschiedene Objekttypen modelliert, woraus sich 392 Instanzen ableiteten. Der Aufwand des gesamten Durchlaufs betrug 19000 Minuten (ca. 8 Personenwochen). Um einen Vergleich mit nachfolgenden Projekten zu ermöglichen, wurde eine Größe eingeführt, die die Komplexität eines Projekts beschreibt. Sie errechnet sich im wesentlichen aus den Aufgaben, die die einzelnen Objekttypen zu erfüllen haben. Diese Komplexität errechnete sich für CS3/1 zu 688. Der Quotient aus dem Aufwand und dieser Größe bestimmt die Produktivität. Hier beträgt sie 27,6.

Einzelheiten der Analysemethode, der Experimentierumgebung und der Auswertung sind in [1] dargestellt.

Als wesentliche Erfahrungen sollen nur die folgenden genannt werden:

- Ein Team der Größe von sieben Entwicklern kann eine in dieser Domäne typische Aufgabe in vernünftiger Zeit lösen.
- Inspektion und Prototyping haben sich gut ergänzt, um unterschiedliche Fehlertypen in unterschiedlichen Dokumenttypen zu finden.
- Durch eine von sehr großen Projekten übernommene sehr strikte Phaseneinteilung des Prozesses entstanden unnötige Wartezeiten und kleine Verbesserungen führten zu aufwendigen Iterationen.
- Fehler traten vor allem an den Schnittstellen zwischen Objekttypen auf, was auf eine inkonsequente Führung des Project-Dictionaries zurückzuführen war.
- Wiederverwendung war auf die Templates beschränkt und könnte erweitert werden.
- Prototyping und damit eine Validierung war erst am Ende der Analyse möglich.

Aus diesen Erfahrungen wurde als Konsequenz eine Reihe von Änderungen an dem Prozess eingeführt, die eine effizientere Durchführung gewährleisten sollten. Dies wurde in einer zweite Fallstudie (CS3/2) validiert.

3 Fallstudie CS3/2

In CS3/2 wurde die Problembeschreibung von CS3/1 um eine Temperatursteuerung ergänzt. Es kamen insgesamt 30 neue Needs hinzu. Um bessere Aussagen über die Wiederverwendbarkeit von Artefakten machen zu können, wurde festgelegt, dass der CS3/1 Entwurf nicht als Ganzes, sondern nur in Form kleinerer Einheiten wiederverwandt werden sollte. Dazu wurden einzelne Artefakte generalisiert und in einer Reuse-Library abgelegt.

Anstelle der strikten Phaseneinteilung wurde als Vorgehensmodell ein wesentlich flexiblerer Verfeinerungsansatz gewählt. Dadurch konnten alle Dokumente ständig verfeinert und korrigiert werden, ohne aufwendige Iterationen zu durchlaufen.

Weiterhin wurde das Project-Dictionary durch eine Datenbank (siehe [3]) realisiert und die ständige Wartung erzwungen. Dadurch verringerten sich die Schnittstellenfehler drastisch.

Bei jedem neu zu erstellenden Artefakt wurde die Suche in der Reuse-Library erzwungen. Gründe für die Nutzung bzw. Ablehnung der gefundenen Reuse-Artefakte und der Aufwand für die Anpassung wurden detailliert erfasst. Die berechnete Produktivitätssteigerung von 1,22 ist zwar typisch, ist aber jedoch aufgrund der relativ kleinen Anzahl von potenziellen Wiederverwendungen noch ungenau.

Durch die Aufteilung der Objektstruktur in kleinere Teilgraphen und die dynamische Zuweisung derselben zu verschiedenen Entwicklern entfielen die Wartezeiten. Durch die Nutzung von Stubs und Treibern konnten die verschiedenen Teile des Systems frühzeitig und getrennt voneinander getestet werden.

Die Anzahl der Objekttypen erhöhte sich um 12 auf 37 Objekttypen, die zu 947 Objekten instanziiert wurden. Die Gesamtkomplexität betrug 1826. Der Gesamtaufwand betrug 29000 Minuten (ca. 12 Personenwochen), woraus sich eine Produktivität von 15,9 ergibt.

Der vollständige Ergebnisbericht über CS3/2 findet sich in [2].

4 Zusammenfassung

Die Änderungen wurden von den Entwicklern subjektiv als Verbesserung bewertet und sie ergaben insgesamt eine Produktivitätssteigerung von 1,73. Die Anzahl der Fehler und nötigen Änderungen (relativ zu den Gesamtkomplexitäten) ging auf ca. die Hälfte zurück. Jedoch ist der durchschnittliche Aufwand zur Fehlerbeseitgung gestiegen, da in erster Linie die Fehlerquellen beseitigt wurden, die einen relativ einfachen Fehler verursachen.

Beide Fallstudien haben sich als sehr wertvoll für die quantitative Erprobung der Analysemethode und der Messmethode herausgestellt und den Aufwand, der von den Mitarbeitern in die Teams eingebracht wurde, voll gerechtfertigt. Die für diese Fallstudien entwickelten Problembeschreibungen waren darüberhinaus auch Grundlage weiterer Fallstudien (siehe Querschnittsprojekt Q2). Weiterhin steht mit den entwickelten System-Requirements eine Eingabe für die nachfolgenden Entwicklungsphasen, speziell der Design-Phase, zur Verfügung.

5 Eigene Veröffentlichungen im Berichtszeitraum

SFB-Berichte

[1] R. Feldmann, J. Münch, S. Queins, S. Vorwieger, G. Zimmermann: *Baselining a Domain-Specific Software Development Process*, SFB 501 Bericht 02/99, Fachbereich Informatik, Universität Kaiserslautern, 1999

[2] S. Queins, G. Zimmermann: *A First Iteration of an Reuse-Driven Domain-Specific System Requirements Analysis Process*, SFB 501 Bericht 13/99, Fachbereich Informatik, Universität Kaiserslautern, 1999

[3] L. Baum, M. Becker, L. Geyer, G. Molter: *Zwei Unterstützungswerkzeuge auf dem Gebiet domänspezifischer Softwareentwicklung*, SFB 501 Bericht 14/99, Fachbereich Informatik, Universität Kaiserslautern, 1999

Zeitschriften / Monographien

[4] S. Queins, G. Zimmermann: *Reuse-Driven, Domain-Specific System Requirements Analysis Process*, erscheint in *Special Issue on RequirementsEngineering: The Light Control Case Study*, Journal of Universal Computer Science, 2000.

Ergebnisbericht Querschnittsprojekt Q2

Entwicklung eines Gebäudeautomationssystems

Leiter

Prof. Dr. Dieter Rombach

Mitarbeiter

Ergänzungsausstattung:

Dipl.-Inform. Raimund L. Feldmann
Dipl.-Inform. Jürgen Münch
Dipl.-Inform. Stefan Vorwieger

Grundausstattung:

Dipl.-Inform. Natalie Ardet
Dipl.-Inform. Antje von Knethen

Zusammenfassung

Im Rahmen des SFB 501 werden alle verfügbaren Techniken, Methoden und Werkzeuge regelmäßig auf ihre positive Wirkung in Bezug auf Qualität, Kosten und Zeit bei der Entwicklung umfassender Gebäudeautomationssysteme getestet. Dabei wird zum einen die Wirkung neu entwickelter Techniken, Methoden und Werkzeuge unter möglichst realistischen Bedingungen getestet, zum anderen werden Baseline-Erfahrungen erfasst, gegenüber denen Ergebnisse zukünftiger Forschungsarbeiten als Verbesserungen demonstriert werden müssen. Im Querschnittsprojekt Q2 wurde die erste umfassende Baseline (V1) erstellt. Dabei wurden bis Mitte 1999 verfügbare und isoliert erprobte Techniken und Werkzeuge eingesetzt. Im Rahmen eines Teamprojekts mit Informatikstudenten im Hauptstudium wurde ein komplettes Gebäudeautomationssystem entwickelt. Die dabei gewonnenen Erfahrungen, wie das verwendete Prozessmodell sowie Aufwands-, Zeit- und Qualitätsmodelle, wurden als Referenzen für zukünftige Baselines konserviert.

1 Bedeutung von Entwicklungsbaselines im SFB 501

Der positive Beitrag einer Technik oder eines Werkzeugs (z. B.: Reduktion des Entwicklungsaufwands, Verbesserung der Qualität) unterliegt in praktischen Entwicklungensprojekten sehr vielen Einflüssen (z. B.: Erfahrung der Entwickler, Einhaltung des Prozesses, Robustheit der Entwicklungsplattform). Darüber hinaus können sich unterschiedliche Techniken/Werkzeuge gegenseitig beeinflussen. Deshalb ist es nicht ausreichend, neu entwickelte Techniken und Werkzeuge isoliert empirisch zu untersuchen, sondern sie müssen in geeigneter Kombination (definiert über ein Prozessmodell) erprobt werden. Solche notwendigen Scale-Up-Experimente, sogenannte Baselining-Entwicklungen, definieren Baselines in Bezug auf den praktischen Effekt einer neuen Kombination einzelner Techniken/Werkzeuge. Baselining-Entwicklungen sind teuer und können nur in größeren Abständen wiederholt werden, da komplexe Systeme aus der Anwendungsdomäne des SFB vollständig entwickelt werden und somit eine Vielzahl von personellen Ressourcen (für die Planung, Entwicklung, Messdatenerfassung, Analyse etc.) über einen längeren Zeitraum gebunden ist. Aus diesem Grund wird an die eingesetzten Methoden, Techniken und Werkzeuge die Bedingung gestellt, dass sie bereits im Verantwortungsbereich der jeweiligen Teilprojekte im Rahmen von Technologie-Erprobungen bzw. Technologie-Experimenten erfolgreich erprobt wurden. Die Resultate von Baselining-Entwicklungen sowie die hierbei gemachten Erfahrungen sollen wiederum Änderungen oder Neuentwicklungen von Methoden, Techniken und Werkzeugen durch die Teilprojekte motivieren, die zu Verbesserungen führen.

2 Ziele von Q2

Ziel des Querschnittsprojekts Q2 war die Entwicklung einer kompletten Baseline von Modellen und Erfahrungen für den bis Mitte 1999 verfügbaren Satz von Beschrei-

bungstechniken, Entwicklungsmethoden und Werkzeugen. Diese Techniken, Methoden und Werkzeuge beinhalten die für OO-Entwicklung gängigen

- Beschreibungstechniken für die Erstellung der Anforderungen aus Benutzersicht (UML-Use-Case-Diagramme, Szenarienbeschreibungen etc. nach [14][17]) und die zugehörigen Analyse-Entwicklungsmethoden aus BOE [17] und dem Unified Process [15],
- Beschreibungstechniken für die Erstellung von Anforderungen aus Entwicklersicht (UML-Klassendiagramme, UML-Sequenzdiagramme etc. nach [14]) und die zugehörigen Analyse-Entwicklungsmethoden aus OMT [18] und dem Unified Process [15],
- Entwurfsbeschreibungstechniken (UML-Klassendiagramme, UML-Packagediagramme, UML-Zustandsdiagramme etc. nach [14][15]) und die Entwurfsentwicklungsmethoden aus OMT [18] und dem Unified Process [15],
- Kodierungstechniken- und Werkzeuge (C++, UNIX-Werkzeuge),
- Test-Techniken für die Komponenten- und System-Validation (Black-Box-Testen),

sowie bereits im SFB (weiter-)entwickelte

- Anforderungsbeschreibungstechniken (FOREST-Ansatz [1][5][7]; entwickelt in den Teilprojekten C1 und B4),
- Inspektionstechniken (Checklisten-basierte Inspektionen mit domänenspezifischen Checklisten [9]; entwickelt in Teilprojekt B1),
- Methoden und Werkzeuge zur Projekt- und Messplanung (Zielführende Projektplanung mit MVP-L [6], GQMPlanner [4]; entwickelt in Teilprojekt B1).

Diese Techniken, Verfahren und Werkzeuge wurden nach einem Wasserfall-ähnlichen Entwicklungsprozess zur Entwicklung eines kompletten Gebäudeautomationssystems eingesetzt. Als Baseline dokumentiert wurden

- das tatsächlich durchgeführte Prozessmodell [3],
- Aufwands-/Zeit-/Fehlermodelle (entsprechend des Prozessmodells) [2],
- Aufwands-/Zeit-/Fehlermodelle bezüglich einzelner Techniken, Methoden und Werkzeuge (nur rudimentär) [2].

3 Plan für Q2

Gemäß der Intention von Baselining-Entwicklungen im SFB war die Durchführung des gesamten Anwendungsentwicklungsprozesses von der OO-Analyse bis zum Systemtest vorgesehen. Ausgeschlossen blieb die Entwicklung von Kommunikations- und Betriebssystemschichten. Somit konnte das resultierende Anwendungssystem nur simulativ ausgeführt werden. Zugrundegelegt wurde ein Wasserfall-ähnlicher OO-Entwicklungsprozess, der sich am Stand der Technik orientiert und sich in ein für den

SFB standardisiertes Lebenszyklusmodell einfügt. Als wesentliche Entwicklungs- und Beschreibungstechniken wurden UML, C++, Checklisten-orientierte Anforderungs-, Entwurfs- und Kodeinspektionen sowie Black-Box-Testverfahren für die Komponenten- und System-Validation eingesetzt. Für die Planung wurden die Prozessmodellierungsansätze MVP-L und MILOS, der GQM-Messansatz sowie diese Ansätze unterstützende Planungswerkzeuge eingesetzt.

Die Aufgabenstellung lautete, ausgehend von einer im Projektbereich D (Anwendungssystem Gebäude) erstellten Problembeschreibung, ein Gebäudeautomationssystem mit Licht- und Temperaturregelung zu erstellen. Der geplante zeitliche Rahmen betrug drei Monate. Als Entwickler waren Informatikstudenten mit Software-Engineering-Kenntnissen vorgesehen, die Leitung einzelner Entwicklungsgruppen sollten wissenschaftliche Mitarbeiter übernehmen. Um die Entwicklung in dem gegebenen zeitlichen und organisatorischen Rahmen realisieren zu können, wurde eine stark arbeitsteilige Vorgehensweise angestrebt. Hierzu war die Bildung von Teams vorgesehen, die beginnend mit der Analysesphase parallel an Subsystemen mit minimaler Kopplung arbeiten. Eine entsprechend geeignete Anforderungsstruktur wurde vorgegeben. Für die Entwicklung war die Verwendung folgender Werkzeuge vorgesehen: Zur Modellierung und (Teil-)Generierung der UML-Diagramme sollte Software through Pictures (StP) von Aonix in der UML-Version verwendet werden [13]. Die Dokumentation sollte weitgehend mit dem Textverarbeitungssystem FrameMaker erfolgen. Für die automatische Generierung großer Teile der Dokumentation wurde ein eigenentwickelter Generator [8] bereitgestellt, sodass ein einfaches Zusammenspiel von StP und FrameMaker möglich war. Zur Übersetzung von Kode und zur Systemintegration waren UNIX-übliche Werkzeuge vorgesehen. Die Baselining-Entwicklung sollte auf folgenden Ergebnissen des SFB aufbauen: Neben der bereits erwähnten und im Rahmen von Prototyping-Entwicklungen (siehe Querschnittsprojekt Q1) verfeinerten Problembeschreibung war eine natürlichsprachliche, gemäß des FOREST-Ansatzes erstellte Problembeschreibung als Referenzdokument vorgesehen [12] (siehe auch Querschnittsprojekt Q4). Diese von den Teilprojekten C1 und B4 erstellte Problembeschreibung sollte als detailliertes Nachschlagewerk zusätzlich zur Problembeschreibung verwendet werden [1][5][7]. Weiterhin wurde eine Systemdokumentation eines eingeschränkten Gebäudeautomationssystems (nur Klimaregelung) aus einem früheren Entwicklungsprojekt zur Verfügung gestellt, die in erster Linie als Orientierungshilfe bzw. Richtlinie für die Dokumentation dienen sollte. Die Planung (insbesondere gemachte Sollvorgaben) basierte auf Schätzungen für Aufwand, Fehler und Kalenderzeit aus früheren Entwicklungsprojekten [10][11].

4 Durchführung von Q2

Die Baselining-Entwicklung wurde von 17 Software-Entwicklern (Informatikstudenten mit Software-Engineering-Kenntnissen) an der Universität Kaiserslautern im Jahr 1999 durchgeführt. Die Baselining-Entwicklung erfolgte im Rahmen eines dreimonatigen Software-Engineering-Praktikums, das basierend auf einer vorhergehende Soft-

ware-Engineering-Vorlesung durchgeführt wurde. Die Entwickler wurden in fünf Teams zu je drei bzw. vier Personen eingeteilt. Wesentliche Aufgabe dieser Teams war die parallele Bearbeitung folgender Subsysteme: a) Temperaturregelung, b) Lichtregelung für Flure, c) Lichtregelung für Räume, d) Sensor- und Aktuatormanagement, e) Verwaltung von Systemmodi. Fünf Mitarbeiter/-innen der SFB-Teilprojekte A1 und B1 und der Arbeitsgruppe Software Engineerung betreuten die parallelen Teams unter Leitung von Prof. Rombach (Natalie Ardet, Raimund L. Feldmann, Antje von Knethen, Jürgen Münch, Stefan Vorwieger). Bei der Anforderungsanalyse sowie deren Inspektion wurde ein Mitarbeiter des SFB-Teilprojekts C1 involviert (Martin Kronenburg). Die Inspektion des Systementwurfs erfolgte unter Einbeziehung von Mitarbeitern der Teilprojekte B5 und B10 (Lothar Baum, Martin Becker, Lars Geyer und Georg Molter). Die Baselining-Entwicklung konnte entsprechend der Planung durchgeführt werden, jedoch konnten die Systemintegration und -validation nicht im Praktikum erfolgen und wurden im Anschluss von wissenschaftlichen Hilfskräften durchgeführt. Hierauf entfielen ca. 10% des Gesamtaufwandes. Die Komplettierung der Entwicklung durch wissenschaftliche Hilfskräfte war erforderlich, da der zeitliche Aufwand für Entwurfs- und Kodeinspektionen unterschätzt worden war.

5 Resultierende Modelle

Wesentliche Resultate der Baselining-Entwicklung sind ein Prozessmodell für die Entwicklung eingebetteter Software sowie eine Menge von Qualitätsmodellen:

- Das *Prozessmodell P501.V1*: Es handelt sich um eine Wasserfall-ähnliches Modell mit Iteration [3] mit folgenden Besonderheiten:
  - → Der Analyseschritt wurde zweistufig (getrennt nach Benutzer- und Entwickleranforderungen) mit Analysetechniken aus BOE [17], OMT [18] und dem Unified Process [15] durchgeführt.
  - → Der Entwurfsschritt wurde mit Entwurfstechniken aus OMT [18] und dem Unified Process [15] durchgeführt.
  - → Das Ende der Prozessschritte Analyse, Entwurf und Kodierung wurde durch einen Inspektionsschritt dokumentiert. Diese Inspektion wurde Checklisten-orientiert durchgeführt.
  - → Der Prozess legt einen Schwerpunkt auf die Beschreibung des Verhaltens des Systems. Dies spiegelt sich in der umfassenden Modellierung von Zustands-, Sequenz- und Kollaborationsdiagrammen wider.
  - → Der Prozess ist domänenspezifisch (für eingebettete Software) ausgelegt. Dies schlägt sich beispielsweise in modifizierten Notationen oder der Einbeziehung von Referenzmodellen nieder.

- Das *Aufwandsmodell A501.V1* zeigt eine (37%, 30%, 21%, 9%, 3%)-Verteilung auf die Phasen Analyse, Entwurf, Kodierung (mit integrierter Komponen-

ten-Validation), Integration und System-Validation [9]. Besonders fällt auf, dass

→ der Aufwand für die System-Validation außergewöhnlich gering (3%) ist. In industriellen Projekten geht man i. a. von bis zu 50% aus. Der geringe Anteil kann auf die frühzeitige Entdeckung vieler Fehler bei Inspektionen zurückgeführt werden. Allerdings ist bei der Beurteilung der verwendeten Testdaten zu berücksichtigen, dass diese aus den in der Analyse erstellten Szenarien abgeleitet sind und nicht auf statistisch fundierten Benutzungsprofilen fundieren.

→ der Aufwand in den frühen Prozessschritten (Analyse und Entwurf) verhältnismäßig hoch (67%) ist. Dies ist im wesentlichen auf zwei Analyseschritte und die umfangreiche Modellierung verschiedener UML-Sichten zurückzuführen. Die Produktivität bezogen auf den Gesamtaufwand liegt jedoch mit 8 LOC/h (Lines of Code/Stunde) weit über dem Durchschnitt industrieller Projekte (5 LOC/h für Organisationen mit CMM-Level 3).

- Das *Fehlerslippagemodell FS501.V1* zeigt eine Verteilung der Fehler auf Produkte gemäß Abb. 1 [9]. Es ist nach den Produkten, in denen die Fehler auftreten, gegliedert und gibt Auskunft über den jeweiligen Zeitpunkt der Entdeckung eines Fehlers. Das Fehlerslippagemodell ist beispielhaft im Anschluss an diese Auflistung hinsichtlich seiner Bedeutung und möglicher Schlussfolgerungen interpretiert. Besonders fällt auf, dass

  → die meisten der besonders teuren Anforderungsfehler (96%) bereits frühzeitig bei der Anforderungsinspektion entdeckt wurden.

- Das *Fehlerklassenmodell FK501.V1* zeigt für die Fehlerklassen “Unvollständigkeit”, “Inkonsistenz”, “Falsche Information”, “Überspezifikation” und “Vorwärtsreferenzen” folgende Gesamtfehlerverteilung: (13%, 11%, 8%, 3%, 4%) für das Analysedokument, (19%, 10%, 8%, 1%, 2%) für das Entwurfsdokument und (5%, 8%, 5%, 1%, 2%) für den Kode [9]. Besonders fällt auf, dass

  → in sämtlichen Fehlerklassen eine Abnahme der Fehleranzahl im Verlauf der Entwicklung festzustellen ist (mit einer Ausnahme beim Entwurf). Dies spricht für eine frühzeitige Erkennung unterschiedlichster Fehlerarten. Die hohe Zahl an Unvollständigkeiten und Inkonsistenzen bei Analyse und Entwurf lässt auf eine noch unzureichende Verfolgbarkeit zwischen den UML-Sichten schließen.

- Das *Kalenderzeitmodell K501.V1* zeigt eine (26%, 40%, 13%, 3%, 18%)-Verteilung auf die Phasen Analyse, Entwurf, Kodierung, Integration und System-Validation [9]. Die zeitliche Verteilung der Aktivitäten harmoniert weitgehend mit der Aufwandsverteilung. Abweichungen lassen sich auf die Einhaltung von Koordinationszeitpunkten zwischen Teams und damit verbundene zeitliche Verzögerungen zurückführen.

Sämtliche Qualitätsmodelle sind unter [9] abgelegt und in [2] finden sich Interpretationen der Qualitätsmodelle. Das in Abb. 1 dargestellte Fehlerslippagemodell soll hier beispielhaft interpretiert werden. Es zeigt die Verteilung der Fehler über die Problemlösungsdokumentation (diese umfasst die Analyse), die Software-System-Dokumentation (diese umfasst den Entwurf), die Software-Komponenten-Dokumentation (diese umfasst die Komponenten-Entwürfe und den Kode) sowie das ausführbare System. Die Prozesse ‘System-Anforderungen bearbeiten’, ‘System-Entwurf bearbeiten’ und ‘System-Komponenten bearbeiten’ umfassen jeweils eine abschließende Inspektion der erstellten Dokumente. Die meisten Fehler, die während der Analyse gemacht wurden, konnten während der Anforderungsinspektion entdeckt werden. Nur 0,46% der Analysefehler wurden erst in der Entwurfsphase und nur 0,76% bei der System-Validation gefunden. Dies spricht für eine sehr effektive Anforderungsinspektion und einen hohen Reifegrad der verwendeten Checklisten. Entwurfsfehler konnten nicht in diesem Umfang bei der Entwurfsverifikation entdeckt werden: 5,23% wurden erst in späteren Phasen entdeckt. Eine Analyse ergab, dass vor allem auf Inkonsistenzen beruhende Fehler (24%) sowie Unvollständigkeitsfehler (48%) im Entwurfsdokument vorherrschten. Dies ist auf eine unzureichende horizontale Verfolgbarkeit zwischen verschiedenen UML-Entwurfssichten zurückzuführen. Eine entsprechende Entwicklungsunterstützung durch explizite Verfolgbarkeitsbeziehungen und hierauf abgestimmte Checklisten bei der Inspektion versprechen eine signifikante Verringerung der Entwurfsfehler. Bei der Komponentenentwicklung sind ca. 20% der Fehler aufgetreten, jedoch konnten die meisten (18,28%) bei der Kodeinspektion entdeckt und daraufhin behoben werden. Da das Qualitätsmodell die Fehler entsprechend ihres Ursprungsprodukts zuordnet, sind dem ausführbaren System keine Fehler zugeordnet. Bei der System-Validation entdeckte Fehlverhalten werden im Qualitätsmodell auf Fehler in den jeweiligen Dokumentationen abgebildet.

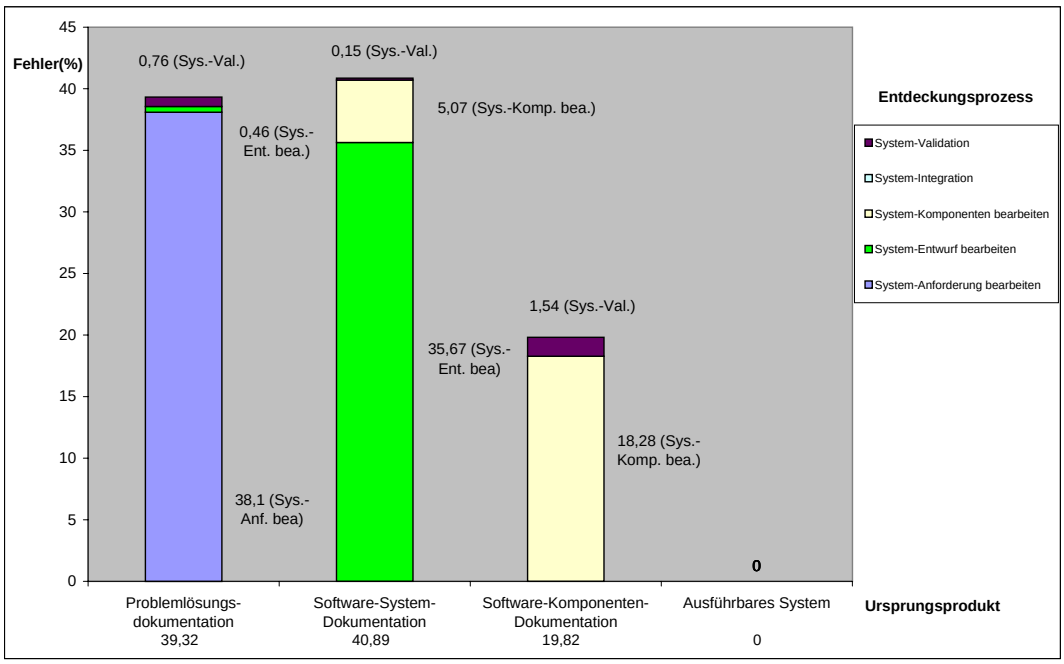

Abbildung 1: Slippagemodell für Fehler

6 Resultierende Erfahrungen

Bei der Durchführung der Baselining-Entwicklung wurden systematisch qualitative Erfahrungen in Form von Lessons Learnt dokumentiert und anschließend ausgewertet [2][9], sodass sie als Anhaltspunkte für zukünftige Verbesserungen der eingesetzten Methoden, Techniken und Werkzeuge verwendet werden können. Diese Erfahrungen sind vollständig in der SFB-Erfahrungsdatenbank abgelegt und in eine Zahl von Klassen eingeteilt, für die im Folgenden jeweils ein Beispiel angegeben ist: a) *Erfahrungen mit der Methodik:* Konflikte bei der Parallelmodellierung von Klassendiagrammen führten zur Zuordnung von Verantwortlichen für "Schnittstellenklassen"; b) *Erfahrungen mit den Entwicklungswerkzeugen:* Die werkzeuggestützte Generierung von Dokumenten führte zu einer hohen Zeitersparnis, sollte jedoch in Zukunft nur zu prozessgekoppelten Zeitpunkten erfolgen; c) *Erfahrungen mit der Planung/Management:* Der Aufwand für Inspektionen wurde unterschätzt, sodass mehr Zeit eingeplant werden sollte; d) *Erfahrungen mit Messen und Bewerten*: Die Wiederverwendung von Messerfahrungen konnte durch effektive Anpassung von existierenden GQM-Plänen erfolgen; e) *Erfahrungen mit der Wiederverwendung von Domänenwissen:* Die existierende Dokumentation reichte zur Anwendung der Steuerungsalgorithmen nicht aus und erforderte die Konsultation von Experten.

7 Offene Fragen

Die Analyse der quantitativen und qualitativen Erfahrungen führte zur Identifikation von Verbesserungspotentialen und lieferte damit Hinweise auf weitere Forschung im B- und C-Bereich. Als wesentliche offene Fragen und damit verbundene Verbesserungspotentiale wurden erkannt:

a) Wie lassen sich Unvollständigkeiten und Inkonsistenzen bei der Analyse und beim Entwurf reduzieren bzw. frühzeitiger entdecken? Der Einsatz von Werkzeugen zur Verbesserung der horizontalen Verfolgbarkeit, explizite Verfolgbarkeitsinformation zur Verbesserung der vertikalen Verfolgbarkeit sowie eine Änderung der Entwurfsinspektionen können diesbezüglich Verbesserungen bringen.

b) Wie lässt sich Domänen- und Anforderungswissen so beschreiben, dass eine weitgehend selbstständige Nutzung durch die Entwickler (ohne Expertenkonsultation) möglich ist? Der Einsatz von Mechanismen zur Generierung von Projektionen auf solches Wissen kann helfen, dieses Problem zu überwinden.

c) Wie lässt sich die Anleitung der Entwickler derart verbessern, dass die vorgegebenen Beschreibungstechniken korrekt angewendet werden, Abstimmungsprobleme bei der Erstellung verschiedener UML-Sichten minimiert werden sowie wenig erfahrene Entwickler effektiv unterstützt werden? Die Überarbeitung der Entwicklungsrichtlinien sowie die Dokumentation des Prozessmodells in Form eines Prozesshandbuchs versprechen diesbezügliche Verbesserungen.

8 Ausblick

Die Resultate der Baselining-Entwicklung sowie die identifizierten Verbesserungspotentiale sollen Änderungen oder Neuentwicklungen von Methoden, Techniken und Werkzeugen motivieren, die zunächst in den Teilprojekten isoliert erprobt und im Falle der positiven Erprobung wiederum in einer Baselining-Entwicklung in der dritten Förderungsperiode eingesetzt werden sollen. Des Weiteren ist vorgesehen, Baselining-Entwicklungen auf die Entwicklung von maßgeschneiderten Kommunikations- und Betriebssystemschichten auszuweiten (zusätzlich zur Anwendungssoftware). Mit Baselining-Experimenten wird im SFB 501 eine weitgehend realistische Entwicklungssituation angestrebt. Bei der Beurteilung der Übertragbarkeit der Resultate muss jedoch berücksichtigt werden, dass die Entwicklung mit Informatikstudenten im Hauptstudium in einem universitären Umfeld erfolgt. Die Durchführung von Baselining-Entwicklungen in industriellen Umgebungen mit professionellen, erfahrenen Entwicklern kann zu einer Steigerung der externen Validität (d. h., Übertragbarkeit) der Resultate führen. Dies soll im Kontext von Industrie-Kooperationen und Transferprojekten erfolgen.

9 Veröffentlichungen

9.1 Eigene Veröffentlichungen

[1] R. Gotzhein, M. Kronenburg, C. Peper: *Reuse in Requirements Engineering: Discovery and Application of a Real-Time Requirements Pattern*. 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT‘98), LNCS 1486, Springer, Lyngby, Dänemark, S. 65-74, 1998.

[2] Antje von Knethen, Raimund Feldmann, Jürgen Münch, Stefan Vorwieger: *A Baseline for the Development of Embedded Software.* Technischer Bericht Nr. 4/00, Sonderforschungsbereich 501, 67653 Kaiserslautern, 2000.

[3] Antje von Knethen, Jürgen Münch: *Entwicklung eingebetteter Software mit UML: Der Do-it-Prozess V 1.0*, Technischer Bericht Nr. 05/00, Sonderforschungsbereich 501, 67653 Kaiserslautern, 2000.

[4] Jürgen Kortgen: *Validierung der systematischen Entwicklung von GQM-Plänen.* Projektarbeit, Fachbereich Informatik, Universität Kaiserslautern, 2000.

[5] M. Kronenburg, C. Peper: *Definition and Instantiation of a Reference Model for Problem Specifications*. Proc. 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Deutschland, S. 332-336, 1999.

[6] Jürgen Münch: *Anpassung von Vorgehensmodellen im Rahmen ingenieurmäßiger Softwarequalitätssicherung*. Im Tagungsband des 6. Workshops der Fachgruppe 5.1.1 (GI), Kaiserslautern, 19.-20. April, 1999.

[7] C. Peper, R. Gotzhein, M. Kronenburg: *A Generic Approach to the Formal Specification of Requirements*. Proc. 1st IEEE International Conference on Formal Engineering Methods (ICFEM‘97), Hiroshima, Japan, S. 252-261, 1997.

[8] Wolfgang Wagenbichler: *Erstellung einer SW-Systemdokumentation mit StP.* Projektarbeit, Universität Kaiserslautern, 67653 Kaiserslautern, 1999.

9.2 Online-Dokumentationen

[9] AG Software Engineering: *Dokumentation Querschnittsprojekt Q2*. URL: http://sep1.informatik.uni-kl.de:18000/TEAM3/WWW/BaX3/baX3.html

[10] AG Software Engineering: *Software-Entwicklungsdokumentation zum SE-1-Praktikum 1998.* URL: file:/homes/edbdemo/EDB/INHALTE/SPEZIFISCH/FALLSTUDIEN/se-1-98_contents.html

[11] AG Software Engineering: *Software-Entwicklungsdokumentation zum SE-1I-Praktikum 1998/1999.* URL: file:/homes/edbdemo/EDB/INHALTE/SPEZIFISCH/FALLSTUDIEN/SE-2-99/

[12] C. Peper, R. Gotzhein, M. Kronenburg: *Problemspezifikation nach dem FOREST-Ansatz.* URL: http://www-avenhaus.informatik.uni-kl.de/forest/EXAMPLES/SE_PRAKTIKUM/SE_Prakti

9.3 Fremdveröffentlichungen

[13] Aonix: *Software through Pictures: User Manuals*. Aonix Release 2.4 und 3.4, 1998.

[14] Grady Booch, James Rumbaugh, Ivar Jacobson: *The Unified Modeling Language User Guide*. Addison-Wesley Longman, 1999.

[15] Ivar Jacobson and Grady Booch and James Rumbaugh: *The Unified Process.* IEEE Software, Vol. 16, Nr. 3, S. 96-102, Mai 1999.

[16] Ivar Jacobson, Grady Booch, James Rumbaugh: *Unified Software Development Process*. Addison Wesley Publishing Company, 1999.

[17] Bernd Oesterreich: *Objektorientierte Softwareentwicklung mit der Unified Modeling Language*. Oldenburg-Verlag, 3., aktualisierte Auflage 1997.

[18] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, Bill Lorensen: *Object-Oriented Modeling and Design.* Prentice Hall, 1990.

Querschnittsprojekt Q3

Modellierung der Domäne Gebäudeautomation

Mitarbeiter

Dipl. Inform. Lars Geyer (B5)

Dipl. Inform. Martin Kronenburg (C1)

Dr. Rolf Merz (D2)

Dr. Bernd Schürmann (D1)

1 Aufgabenstellung

Die initiale Zielsetzung dieses Querschnittsprojekts ist es, die Erfahrungen im SFB, insbesondere in den Teilprojekten D1 und D2, bezüglich der Domäne Gebäudeautomation in einem Domänenmodell zusammenzufassen. Ein Domänenmodell beschreibt das Wissen über ein bestimmtes Anwendungsfeld im Problemraum, d.h. es werden alle relevanten Anforderungen an mögliche Systeme in einer Domäne, sowie alle für das Verständnis des Problems notwendigen Informationen in einem Modell zusammengefasst.

Durch ein Domänenmodell werden im Wesentlichen zwei Ziele erreicht. Zum einen dient die Modellierung dazu, eine Domäne abzugrenzen. Ein Domänenmodell definiert die mögliche Anforderungsbandbreite, die bei der Erstellung eines Systems berücksichtigt werden muss. Zum anderen ist eine wesentliche Aufgabe eines Domänenmodells die Unterscheidung zwischen Gemeinsamkeiten und Variabilitäten. Gemeinsamkeiten sind Anforderungen, die an alle Systeme einer Domäne gestellt werden. Im Gegensatz dazu stehen variable Anforderungen, die nur in bestimmten Systemen einer Domäne eine Rolle spielen. Durch die explizite Modellierung von Gemeinsamkeiten erlaubt ein Domänenmodell eine Abschätzung bezüglich des Wiederverwendungspotentials in einer Domäne.

Als Grundlage dienten dem Querschnittsprojekt die von den Teilprojekten D1 und D2 während der zurückliegenden Förderungsperioden entwickelten Problembeschreibungen. Zudem stand durch die Teilnahme von Mitarbeitern dieser Teilprojekte Expertenwissen unmittelbar zur Verfügung.

Als initiale Vorgehensmethode wurde die *Commonality Analysis* aus dem *FAST*-Ansatz für *Product-Line Software Engineering* [7] verwendet, die in Abschnitt 2.2 vorgestellt wird. Die mit dieser Methode erzielten Ergebnisse waren jedoch nicht zufrieden stellend, so dass im Rahmen des Querschnittsprojekts die Methode weiterentwickelt wurde. Die resultierende Methode und das daraus abgeleitete Strukturierungskonzept für Domänenmodelle werden in den Abschnitten 2.2 und 2.3 beschrieben. Eine genauere Beschreibung der erzielten Ergebnisse ist in SFB-Bericht [1] zu finden.

2 Durchgeführte Arbeiten und Ergebnisse

2.1 Begriffserklärungen

Zunächst war es erforderlich, eine einheitliche Begriffswelt festzulegen. Dabei konnte auf die in [3] eingeführte Terminologie zurückgegriffen werden. Danach ist ein *System* ein Teil der realen Welt, der aus bestimmten Gründen als eine Gesamtheit angesehen wird. In einem System sind verschiedene Phänomene vereint und miteinander verbunden. Ein *Phänomen* ist ein Aspekt der realen Welt, der wesentlich für ein betrachtetes System ist. Phänomene sind zum Beispiel Zustände eines Systems wie

die Temperatur in einem Raum, Ereignisse wie das Einschalten einer Lampe, Objekte wie Sensoren oder aber auch Personen wie zum Beispiel der Hausmeister eines Gebäudes. Um über solche Phänomene und damit über ein System reden zu können, werden Bezeichner eingeführt. Ein *Bezeichner*, zum Beispiel ein Wort in natürlicher Sprache oder ein Prädikatssymbol in einer Logik, ist also ein Repräsentant eines bestimmten Phänomens eines Systems. Um Mehrdeutigkeiten zu vermeiden, fordern wir stets eine Eins-zu-eins-Beziehung zwischen Phänomenen und Bezeichnern. Außerdem verlangen wir, dass es zu jedem Bezeichner eine Erläuterung gibt, die beschreibt, welches Phänomen durch einen Bezeichner repräsentiert wird. In [6] wird die Wichtigkeit solcher Erläuterungen betont.

Eine *Aussage* drückt einen Zusammenhang zwischen einigen Phänomenen aus. Jede Aussage besteht im Wesentlichen aus den Bezeichnern, die die betrachteten Phänomene repräsentieren. Der Satz „Wenn ein Raum belegt ist, muss das Licht an sein“ oder die Formel „raumBelegt -> lichtAn“ sind Beispiele für Aussagen. Eine *Beschreibung* besteht aus einer Menge von Bezeichnern versehen mit Erläuterungen sowie einer Menge von Aussagen, in denen ausschließlich die angegebenen Bezeichner vorkommen. Zu beachten ist, dass die Angabe der Phänomene in den Erläuterungen der Bezeichner auch festlegt, aus welcher Sicht und auf welchem Abstraktionsgrad ein System betrachtet wird.

Eine *Domäne* ist schließlich eine Menge von Systemen, die etwas gemeinsam haben, sowie das Wissen über diese Systeme. Ein *Domänenmodell* ist eine Beschreibung einer Domäne. Zwischen einzelnen Domänen kann es eine große Anzahl von Beziehungen geben, zum Beispiel ist die Domäne Gebäudeautomation eine *Teildomäne* der Domäne *reaktive Systeme*. Weitere Relationen entstehen, wenn nur ein Ausschnitt oder eine andere Abstraktionsebene einer Domäne beschrieben wird.

2.2 Arbeitsbericht

Ausgangspunkt für die Erstellung des Domänenmodells war die Methode der *Commonality Analysis*, die im Rahmen des *FAST*-Ansatzes zur Domänenanalyse verwendet wird. Die Vorgehensweise in diesem Ansatz basiert auf einer Gesprächsrunde, in der sich Domänenexperten treffen, um über die Aspekte der betrachteten Domäne zu diskutieren. Ein Moderator führt durch die Diskussion. Zusätzlich sorgt der Moderator dafür, dass die Ergebnisse unmittelbar in einem Dokument erfasst werden, so dass die Mitglieder jederzeit Zugriff auf den aktuellen Stand der Diskussion haben.

Ein Dokument nach dem *FAST*-Ansatz ist in mehrere Kapitel unterteilt. Es existiert jeweils ein Abschnitt für die Gemeinsamkeiten der betrachteten Systeme, deren Variabilitäten, die jeweiligen Bandbreiten der Variabilitäten sowie ein *Dictionary* der relevanten Begriffe. Dies wird ergänzt durch einen speziellen Abschnitt für offene Fragen. Die Phänomene der Domäne werden während der Analyse an Hand ihrer Bedeutung als Gemeinsamkeit oder als Variabilität klassifiziert und dementsprechend in den relevanten Abschnitt eingefügt, wobei die Abschnitte im Wesentlichen aus einer Liste der betrachteten Phänomene bestehen. Für variable Phänomene wird

zusätzlich noch die Bandbreite festgelegt, in der sie in der Domäne auftreten. Treten während eines Treffens offene Fragen auf, so werden diese in dem speziellen Abschnitt dokumentiert und als Hausaufgabe an Mitglieder der Gruppe verteilt. Diese sammeln die benötigten Informationen, so dass im darauf folgenden Treffen eine Entscheidung bezüglich dieser offenen Fragen getroffen werden kann.

Um die Abhängigkeiten zwischen den einzelnen Phänomenen zu dokumentieren, wird in den einzelnen Beschreibungen auf verwandte Phänomene verwiesen. Ebenso wird die Entscheidungsfindung bei offenen Fragen dokumentiert, indem eine Fragestellung mit den die Entscheidungen betreffenden Phänomenen verknüpft wird.

In einem ersten Schritt wurde im Querschnittsprojekt dieser Ansatz verwendet, um ein Modell der Domäne Gebäudeautomation zu erstellen. Im Laufe der Zeit wurde die Vorgehensweise jedoch an die Gegebenheiten der Gruppe angepasst. So war zum Beispiel auf Grund der Gruppenstärke von vier Personen kein Moderator zur Leitung der Diskussionen nötig. Alle vier Gruppenmitglieder waren gleichberechtigte Teilnehmer an den Diskussionen. Des Weiteren wurde das Dokument nicht unmittelbar während der Treffen erweitert. Statt dessen wurden Teile des Dokumentes von einzelnen Mitgliedern der Gruppe als Hausaufgabe zwischen den Treffen erstellt. In den Treffen wurden die neuen Teile dann systematisch durchgegangen und in der gesamten Gruppe diskutiert.

Eine wesentliche Änderung zur *Commonality Analysis* stellt die Form dar, in der das Domänenmodell dokumentiert wird. Bei der konkreten Anwendung zeigte sich, dass die starre Trennung zwischen Gemeinsamkeiten und Variabilitäten erzwungen erscheint. Des Weiteren schien auch die listenartige Aufzählung der Phänomene, wie dies in der *Commonality Analysis* normalerweise vollzogen wird, nicht adäquat. Daher entwickelte die Gruppe eine Technik zur strukturierten, objektorientierten Formulierung eines Domänenmodells. Da dieses Strukturierungskonzept ein wesentliches Ergebnis des Querschnittsprojektes darstellt, wird es im nächsten Abschnitt explizit beschrieben.

2.3 Das Strukturierungskonzept

Der strukturierte Aufbau des Domänenmodells basiert im Wesentlichen auf den Strukturierungskonzepten, die in der Kooperation der Teilprojekte B4 und C1, siehe auch Querschnittsprojekt Q4, zur objektorientierten Strukturierung von Problemspezifikationen entwickelt worden sind, vgl. [3], [4]. Um ein Domänenmodell, das eine Menge *großer* Systeme repräsentiert, adäquat darstellen zu können, verwenden wir die Konzepte Modularisierung, Vererbung, Aggregation und Parametrisierung.

Ein Domänenmodell wird als eine Folge von *Beschreibungsklassen* dargestellt. Eine Beschreibungsklasse enthält ihrerseits sowohl Informationen bezüglich der Vererbungs- und Aggregationsbeziehung zu anderen Beschreibungsklassen als auch eine Liste von Bezeichnern, zusammen mit Erklärungen hinsichtlich der repräsentierten Phänomene, sowie eine Liste von indikativen und optativen Eigenschaften. Durch

Beschreibungsklassen ist es möglich, einzelne Teile der Systeme, die zu einer Domäne gehören, getrennt und damit übersichtlicher zu beschreiben. Abbildung 1 zeigt einen Ausschnitt aus der graphischen Repräsentation der Aggregations- und Vererbungshierarchie des erarbeiteten Domänenmodells.

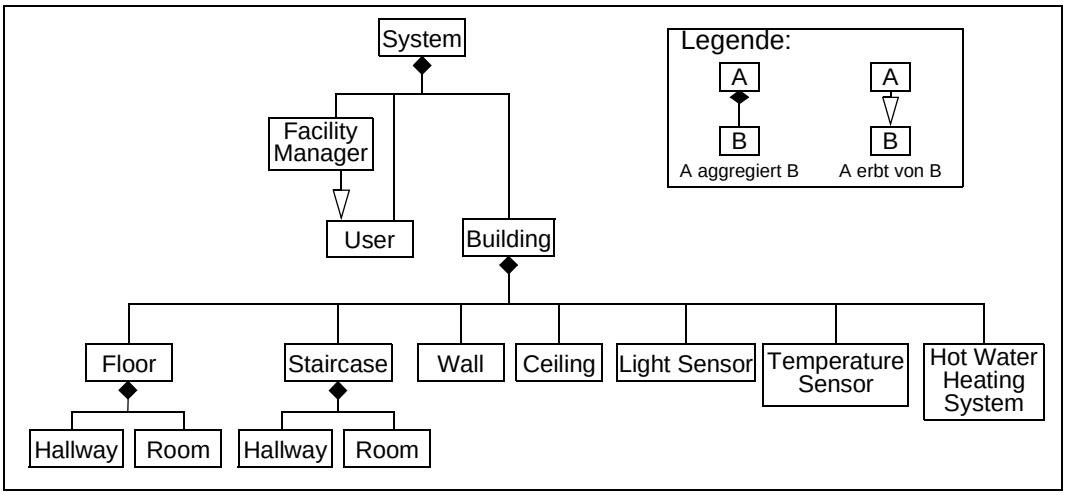

Abbildung 1: Ausschnitt aus der Aggregations- und Vererbungshierarchie

Eine Parametrisierung einer Beschreibungsklasse ist zum einen im Rahmen der Aggregation mit Hilfe von *Anzahlparametern* möglich. Dadurch kann die genaue Anzahl der aggregierten Objekte, zum Beispiel die der Räume auf einem Flur, offen gelassen werden. Zum anderen können im Eigenschaftsteil durch bedingte Aussagen weitere Variabilitäten ausgedrückt werden. Abbildung 2 zeigt einen Ausschnitt der Beschreibungsklasse Door.

2.4 Anwendung des Domänenmodells im SFB

Instanziierung des Domänenmodells

Nachdem ein initiales Domänenmodell vorliegt, muss untersucht werden, inwieweit das Modell als Eingabe für den SFB-Entwicklungsprozess herangezogen werden kann. Vor Durchführung eines entsprechenden ausführlichen Experiments sollte das Domänenmodell mit der im Querschnittsprojekt Q1 verwendeten Problembeschreibung verglichen werden. Ist bei der Instanziierung des Domänenmodells eine einfache Abbildung auf die verwendete Problembeschreibung gegeben, kann es in Zukunft im SFB eingesetzt werden, ohne dass größere Änderungen am Entwicklungsprozess notwendig wären.

Der aktuelle Stand des Domänenmodells und die im Querschnittsprojekt Q3 verwendete Problembeschreibung sind in den SFB-Berichten [1] bzw. [2] enthalten. Ein genauerer Vergleich der beiden Dokumente zeigt Unterschiede in drei wesentlichen Aspekten:

- Bei der Erstellung des Domänenmodells wurde darauf geachtet, dass dieses nur Anforderungen enthält. Die in Q3 verwendete Problembeschreibung enthält dagegen bereits Lösungsstrategien, wie sie für die Domäne typisch sind.

Class:	**Door** (category: building)
Description:	A door is an opening in a wall which users can use for moving between different building units or the environment. It does not store heat, transmits heat, and if it is not closed it also transmits light.
Derived Cl.:	
Inherited from:	*ShapeComponent*
Parts:	(0..1) window: *Window* (0..1) closedSensor: *DoorClosedSensor* (0..1) lockedSensor: *DoorLockedSensor.*
Items:	• *closed* : BOOLEAN This item represents whether a door is closed or not in the real world. It has not to be confused with the value of a possibly installed *closed sensor* measuring the value of *closed*. *closed* = 1 means that a door is closed, *closed* = 0 means that a door is not closed. • *locked* : BOOLEAN This item represents whether a door is locked or not in the real world. It has not to be confused with the value of a possibly installed *locked sensor* measuring the value of *locked*. *locked* = 1 means that a door is locked, *locked* = 0 means that a door is not locked.
Properties:	indicative: • A door transmits heat, i.e. *heatTransmission > 0*. • A door does not store heat, i.e. *heatCapacity = 0*. • A door is pervious for a user, i.e *userPervious = 1.* • If a door is *closed* it does not transmit light, i.e. *lightTransmission = 0*. • If a door is *not closed* it transmits light, i.e. *lightTransmission > 0*. • *A door* can only be opened or closed manually, i.e. no automatic doors are considered in the domain.

Abbildung 2: Ausschnitt aus der Beschreibungsklasse Door. Einige der benutzten Bezeichner (z.B. heatTransmission) werden von der Klasse ShapeComponent geerbt.

Konsequenz: Die Mitglieder des Querschnittsprojekts erachten die frühzeitige Bereitstellung allgemeiner Lösungsstrategien als wichtig und typisch für die Domäne. Auch wenn diese Lösungsansätze nicht verworfen werden sollten, sollten sie doch getrennt von den Anforderungen formuliert werden. Sie sollten Teil einer erweiterten Domänenmodellierung sein. Eine explizite Trennung von Anforderungen und Lösungsansätzen wird als wesentlich erachtet. Diese Thematik soll in der nächsten Förderungsperiode durch das Querschnittsprojekt vertieft betrachtet werden.

- Fast alle Anforderungen der Problembeschreibung, die nicht durch das Domänenmodell abgedeckt sind, sind nicht-funktionaler Art. Diese wurden bei der bisherigen Erstellung des Domänenmodells unberücksichtigt gelassen.
  Konsequenz: Die Gruppe ist der Meinung, dass es noch weitergehender Untersuchungen bedarf, in welcher Form nicht-funktionale Anforderungen spezifiziert werden sollten. Eine Trennung von funktionalen und nicht-funktionalen Anforderungen erscheint sinnvoll.

Insgesamt kommen die Mitglieder dieses Querschnittsprojekts zu dem Schluss, dass eine relativ einfache Integration des Domänenmodells in den SFB-Entwicklungsprozess gegeben zu sein scheint. Eine endgültige Aussage ist allerdings erst nach Durch-

führung entsprechender Experimente möglich. Eine Fortführung des Querschnittsprojekts in der kommenden Förderungsperiode erscheint deshalb notwendig.

Teilprojekt B10

Für das Teilprojekt B10 gibt es zwei Ansatzmöglichkeiten, die Ergebnisse des Querschnittsprojekts in die Arbeiten einfließen zu lassen. Zum einen bildet ein Domänenmodell die Basis für ein generisches Anforderungsmodell, welches Wiederverwendung bereits auf der Ebene der Anforderungsanalyse ermöglicht. Dazu sind die spezifizierten Anforderungen in einer für diese Analyse geeigneten Technik zu modellieren, d.h. die Anforderungen des Domänenmodells müssen gegebenenfalls in eine für den Zweck der Analyse geeignetere Technik umformuliert werden.

Zum anderen dient ein Domänenmodell als Basis für die Erstellung eines vorkonfektionierten Systems. Das Domänenmodell spezifiziert die Anforderungen, die beim Einsatz eines vorkonfektionierten Systems bei der Anwendungsentwicklung zu erwarten sind. Im vorkonfektionierten System muss daher sichergestellt werden, dass die verwendete Architektur die Flexibilität des Domänenmodells unterstützt, um effektiv während einer Anwendungsentwicklung als Grundgerüst einsetzbar zu sein. Zusätzlich bieten gerade die als gemeinsam klassifizierten Anforderungen genug Wiederverwendungspotential, um sinnvoll im Voraus als wiederverwendbare Artefakte implementiert zu werden.

Teilprojekt B11

Teilprojekt B11 will verstärkt nicht-funktionale Eigenschaften betrachten. Eine wichtige Aufgabe von B11 wird daher die Erweiterung des bisher vorliegenden Domänenmodells um nicht-funktionale Anforderungen sein. Diese wurden in diesem Querschnittsprojekt bisher nicht betrachtet. Für die Bearbeitung nicht-funktionaler Eigenschaften ist domänenspezifisches Expertenwissen notwendig, weshalb eine Erweiterung des Domänenmodells um solches Expertenwissen gewünscht wird (vgl. auch Teilprojekt D1). Eine weitere wichtige Aufgabe von B11 wird die Betrachtung der Entwurfsphase sein, in der die meisten nicht-funktionalen Eigenschaften bearbeitet werden. In diesem Zusammenhang soll gemeinsam mit C1 die Erweiterung des Domänenmodells in Richtung Entwurfsphase untersucht werden.

Teilprojekt C1

Aus der Sicht von Teilprojekt C1 ergeben sich vor allem drei Fragen, die im Zusammenhang mit dem bisher erstellten Domänenmodell relevant sind. Erstens: Inwieweit können neben den bisher eingesetzten Strukturierungskonzepten auch die im Teilprojekt C1 entwickelten und verwendeten formalen Beschreibungstechniken eingesetzt werden, um das Domänenmodell auf eine präzisere semantische Grundlage zu stellen? Zweitens: Welche Erweiterungen müssen an dem bisher entwickelten Referenzmodell für Problemspezifikationen vorgenommen werden, um auch solche Domänenmodelle zu erfassen? Hier sind sicherlich weitere Klassifikationsmöglichkeiten sowohl der Bezeichner als auch der Eigenschaften erforderlich. Drittens: Wie

kann ein solches Domänenmodell für den Bereich der Anforderungen auf den Bereich der Entwürfe fortgesetzt werden? Insbesondere der Aspekt der Verfolgbarkeit ist hier von Interesse.

Teilprojekt D1

Auch für Teilprojekt D1 stellen sich drei Fragen im Zusammenhang mit dem Domänenmodell. Da das Domänenmodell die Bandbreite aller möglichen Systeme der Domäne definiert, kann aus ihm auch die Bandbreite der für das Testen und Prototyping benötigten Simulatoren und Testfelder abgeleitet werden. Ein weiterer Aspekt ist die Erweiterung des Domänenmodells um Expertenwissen. Dieses sollte im Wesentlichen in Form domänenspezifischer, generischer Lösungsansätze modelliert werden. Schließlich beschäftigt sich D1 mit der Entwicklung einer (Entwurfs-) Systematik, die vom Entwicklungsprozess beeinflusst wird. In diesem Zusammenhang ist zu untersuchen, wie der vorliegende und im Querschnittsprojekt Q1 erprobte SFB-Entwicklungsprozess zu erweitern ist, damit eine spezielle Problembeschreibung aus dem Domänenmodell in geeigneter Weise abgeleitet werden kann.

Teilprojekt D2

Eine interessante Anwendungsmöglichkeit des generischen Domänenmodells eröffnet sich aus der Sicht des Teilprojektes D2. Der Entwurf konkreter, instantiierter Steuerungs- und Regelungsalgorithmen benötigt detailliertes Wissen zum dynamischen Verhalten des zu beeinflussenden physikalisch, technischen Prozesses, der sogenannten Strecke. Dies leistet ein mathematisches Modell. Neben der Analyse der Wirkungszusammenhänge und Festlegung der Regelkreise, der Auswahl geeigneter Algorithmen und Bestimmung ihrer optimalen Parameter, ermöglicht es simulative Tests der prototypischen Algorithmen. Zusammen mit der Regelungsaufgabe entspricht es einer formalen Spezifikation regelungstechnisch relevanter Aspekte der Anforderungsbeschreibung [5]. Die Schwierigkeit ist seine Erzeugung, basierend auf informell spezifizierten Anforderungsdokumenten und Gebäudebeschreibungen. Im Teilprojekt D2 wurde eine generische Entwurfsmethode mathematischer Modelle zur Simulation thermischen Gebäudeverhaltens entwickelt. Die Methode beruht auf der Nutzung einer Modellbibliothek, bestehend aus modularen, objektorientiert entworfenen Komponentenmodellen mit klaren Schnittstellendefinitionen und Regeln zu ihrer Nutzung. Durch Selektion, Adaption und Komposition können instantiierte mathematische Simulationsmodelle für die konkrete Anwendung generiert werden. Die Struktur der objektorientierten, mathematischen Modellkomponenten spiegelt exakt die Vererbungs- und Aggregationshierarchie des hier entwickelten Domänenmodells wider. Eine Erweiterung des Domänenmodells um mathematische Beschreibungsformen und Modellkomponenten bietet sich an. Bei der Instantiierung eines konkreten Anforderungsdokumentes kann so simultan das mathematische Simulationsmodell als Vorlage für den Regelungstechniker generiert werden, um die Anforderungsbeschreibung bezüglich der für ihn notwendigen technisch, physikalischen Detaillierung zu erweitern.

3 Eigene Veröffentlichungen im Berichtszeitraum

SFB-Berichte

[1] L. Geyer, M. Kronenburg, R. Merz, B. Schürmann: *Domain Modeling in the SFB 501*, SFB 501 Bericht 07/00, Fachbereich Informatik, Universität Kaiserslautern, 2000

[2] R. Feldmann, J. Münch, S. Queins, S. Vorwieger, G. Zimmermann: *Baselining a Domain-Specific Software Development Process*, SFB 501 Bericht 02/99, Fachbereich Informatik, Universität Kaiserslautern, 1999

Tagungsbeiträge

[3] M. Kronenburg, C. Peper: *Definition and Instantiation of a Reference Model for Problem Specifications,* 11th International Conference on Software Engineering and Knowledge Engineering (SEKE‘99), Kaiserslautern, S. 332-336, 1999.

Zeitschriften / Monographien

[4] M. Kronenburg, C. Peper: *Application of the FOREST Approach to the Light Control Case Study,* erscheint in *Special Issue on RequirementsEngineering: The Light Control Case Study,* Journal of Universal Computer Science, 2000.

[5] R. Merz, L. Litz: *Objektorientierte mathematische Modellierung: Generische Methoden bei komplexen dynamischen Systemen*, Informatik Spektrum 2(2000), Springer Verlag Heidelberg, S. 90-99

4 Fremdveröffentlichungen

[6] M. Jackson: *A Discipline of Description (Keynote Talk)*, Requirements Engineering 3(2), S. 73-78, 1998.

[7] D. Weiss: *Software Product-Line Engineering: A Family-Based Software Development Process*, Addison Wesley, 1999

Querschnittsprojekt Q4

Anforderungsanalyse eines Gebäudeautomationssystems

Leiter

Prof. Dr. J. Avenhaus (C1)

Prof. Dr. R. Gotzhein (B4)

Mitarbeiter

Dipl. Inform. M. Kronenburg (C1)

Dipl. Inform. C. Peper (B4)

1 Aufgabenstellung

Aufbauend auf den Ergebnissen aus der Kooperation der Teilprojekte B4 und C1 in der ersten Förderperiode, die in den FOREST-Ansatz eingeflossen sind [12], sollten im Rahmen dieses Querschnittsprojekts mehrere Fallstudien durchgeführt werden. Dabei wurde stets eine gegebene natürlich-sprachliche Problembeschreibung aus dem Anwendungsfeld Gebäudeautomation in eine formale, objektorientierte Problemspezifikation überführt. Die Ziele dabei waren zum einen, allgemein die Eignung des entwickelten Ansatzes für dieses Anwendungsfeld zu überprüfen, und zum anderen, speziell die Möglichkeiten der Wiederverwendung von Teilen einer Problemspezifikation zu untersuchen.

2 Durchgeführte Arbeiten und Ergebnisse

Insgesamt wurden im Rahmen dieses Querschnittsprojekts drei Fallstudien aus dem Anwendungsfeld Gebäudeautomation bearbeitet. Erstens wurde als Unterstützung für die Arbeiten im Querschnittsprojekt Q2 eine Problemspezifikation erstellt, die den Studenten zur Verfügung gestellt wurde [3]. Zweitens nahmen die Mitglieder des Querschnittsprojekts an dem Dagstuhl-Seminar *Requirements Capture, Documentation, and Validation* teil und stellten dort Ausschnitte ihrer Problemspezifikation als vernetztes Geflecht von html-Dokumenten vor [4], [8], [13]. Drittens beteiligten sich die Mitarbeiter des Querschnittsprojekts an der internationalen Fallstudie *The Light Control Case Study* des Journal of Universal Computer Science (J.UCS) [5], [14].

Betrachtet wurde in diesen Fallstudien jeweils ein Stockwerk eines Universitätsgebäudes. Im Falle des Querschnittsprojekts Q2 wurden die Aspekte Licht und Temperatur betrachtet, in den beiden anderen Fallstudien jeweils der Aspekt Licht. Dass es sich hierbei stets um *große* Systeme gehandelt hat, soll anhand einiger Zahlen aus der J.UCS-Fallstudie belegt werden (die beiden anderen Fallstudien bewegen sich in derselben Größenordnung): In der Problemspezifikation zu der J.UCS-Fallstudie werden implizit ungefähr 10000 Phänomene betrachtet und implizit über 6000 Eigenschaften spezifiziert. Aufgrund der im FOREST-Ansatz verwendeten Strukturierungskonzepte mussten aber nur rund 160 Phänomene und rund 170 Eigenschaften explizit angegeben werden. *Explizit* bedeutet dabei zum Beispiel, dass in der Spezifikation der Beschreibungsklasse Room ein Bewegungsmelder nur einmal *explizit* angegeben wird. Durch die Aggregation von Instanzen der Beschreibungsklasse Room in der Beschreibungslasse Floor wird jedoch implizit für jeden aggregierten Raum ein solcher Bewegungsmelder *implizit* spezifiziert.

Die bei diesen Fallstudien gesammelten Erfahrungen geben berechtigten Anlass zu der Aussage, dass der FOREST-Ansatz geeignet ist, funktionale und zeitbehaftete Eigenschaften großer Systeme im Anwendungsfeld Gebäudeautomation präzise und verständlich zu erfassen. Dies wird im Folgenden genauer begründet.

Die von uns in der ersten Förderperiode mit maßgeschneiderten Operatoren angereicherte Temporallogik hat sich als ausdruckstark genug erwiesen, die erforderlichen funktionalen und zeitbehafteten Eigenschaften präzise zu spezifizieren. Wesentlich für das Verständnis einer Problemspezifikation ist die an mehreren Stellen vorgenommene Integration natürlicher Sprache. Vor allem die aus den temporallogischen Spezifikationen abgeleiteten natürlich-sprachlichen Übersetzungen und die für jede Signaturgröße, zum Beispiel Funktions- und Prädikatssymbole, verpflichtend vorgegebene Angabe einer Erläuterung des Phänomens, das durch eine Signaturgröße repräsentiert wird, tragen sehr zur Erhöhung der Verständlichkeit bei. Auf die Bedeutung solcher Erläuterungen wird auch in [15] hingewiesen.

Die in die Temporallogik integrierten Strukturierungskonzepte Modularisierung, Aggregation, Spezialisierung (in Verbindung mit Vererbung) und Parametrisierung haben sich als ausreichend mächtig erwiesen, Problemspezifikationen *großer* Systeme kompakt und damit übersichtlich darzustellen. Eine theoretische Fundierung dieser Konzepte, vor allem auch ihrer Semantik wird in [2] gegeben.
Jedoch hätten diese Fallstudien trotz der Strukturierungskonzepte nicht in dem Umfang durchgeführt werden können, wenn nicht auch ein entsprechendes Werkzeug [1] zur Verfügung gestanden hätte. Dieses Werkzeug erlaubt eine komfortable Eingabe und Verwaltung von Problemspezifikationen, garantiert deren syntaktische Konsistenz, unterstützt - teilweise automatisch - verschiedene Verfolgbarkeitsrelationen und erzeugt ansprechende Ausgaben, vor allem in Form einer Menge durch Hyperlinks verbundener html-Dokumente, siehe zum Beispiel [4], [5].

In den durchgeführten Fallstudien war, neben den durch die Strukturierungskonzepte gegebenen Wiederverwendungsmöglichkeiten innerhalb einer Problemspezifikation, eine Fallstudien übergreifende Wiederverwendung von Teilen einer Problemspezifikation auf drei Ebenen möglich:

- Auf der Ebene der Eigenschaften konnte eine große Anzahl von Eigenschaften durch Instanziierungen von *Requirement Patterns* spezifiziert werden.
- Des Weiteren war es möglich, einzelne Beschreibungsklassen, leicht abgeändert, in einer anderen Fallstudie zu verwenden.
- Schließlich hat sich gezeigt, dass die Zusammenfassung von mehreren Beschreibungsklassen zu Gruppen, zum Beispiel zur Gruppe aller Sensoren, hilfreich ist und in allen Fallstudien sehr ähnlich vorgenommen werden konnte.

3 Eigene Veröffentlichungen im Berichtszeitraum

Diplomarbeiten

[1] T. Schmidt-Samoa: *FOREST - Entwurf und Implementierung einer Umgebung zur Erstellung formaler System-Anforderungs-Beschreibungen*; Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern, 1999.

Dissertationen

[2] M. Kronenburg: *Ein Ansatz zur Erstellung präziser, verständlicher Problemspezifikationen großer reaktiver Systeme*; Dissertation, in Vorbereitung, 2000.

WWW-Dokumente

[3] http://www-avenhaus.informatik.uni-kl.de/forest/EXAMPLES/SE_PRAKTIKUM/SE_PraktikumEnglishStart.html

[4] http://www-avenhaus.informatik.uni-kl.de/forest/EXAMPLES/DAGSTUHL/DagstuhlEnglishStart.html

[5] http://www-avenhaus.informatik.uni-kl.de/forest/EXAMPLES/JUCS/JUCSStart.html

SFB-Berichte

[6] R. Gotzhein, M. Kronenburg, C. Peper: *Reuse in Requirements Engineering: Discovery and Application of a Real-Time Requirement Pattern*; Technischer Bericht SFB 501 08/98, Fachbereich Informatik, Universität Kaiserslautern, 1998.

Vorträge

[7] R. Gotzhein: *Pattern-based System Development*; Kolloquium, Ecole Polytechnique Federale a Lausanne, 9. Juli 1998.

[8] M. Kronenburg: *xforest: A Tool for the Creation of WWW-based Problem Specifications;* Dagstuhl-Seminar Requirements Capture, Documentation, and Validation, 13.-18. Juni 1999.

[9] C. Peper: *Wiederverwendung bei der Anforderungsspezifikation: Der FOREST-Ansatz,* Workshop “Software-Reuse” des SFB 501, Kaiserslautern, 22.-23. Februar 1999.

Tagungsbeiträge

(Der Vortragende ist unterstrichen.)

[10] C. Peper, R. Gotzhein, M. Kronenburg: *A Generic Approach to the Formal Specification of Requirements*; Proc. 1st IEEE International Conference on Formal Engineering Methods (ICFEM`97), Hiroshima, Japan, S. 252-261, 1997.

[11] R. Gotzhein, M. Kronenburg, C. Peper: *Reuse in Requirements Engineering: Discovery and Application of a Real-Time Requirements Pattern*; 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT`98), LNCS 1486, Springer, Lyngby, Dänemark, S. 65-74, 1998.

[12] M. Kronenburg, C. Peper: *Definition and Instantiation of a Reference Model for Problem Specifications*; 11th International Conference on Software Engineering and Knowledge Engineering (SEKE‘99), Kaiserslautern, Deutschland, S. 332-336, 1999.

[13] R. Gotzhein, M. Kronenburg, C. Peper: *Pattern-based Requirements Capture Applied: The SFB 501 Case Study*; in E. Börger, B. Hörger, D. Parnas, D. Rombach, Requirements Capture, Documentation, and Validation, Dagstuhl-Seminar-Report 242, 1999.

Zeitschriften / Monographien

[14] M. Kronenburg, C. Peper: *Application of the FOREST Approach to the Light Control Case Study;* erscheint in Special Issue on “Requirements Engineering: The Light Control Case Study”, Journal of Universal Computer Science (J.UCS), Springer, 2000.

4 Fremdveröffentlichungen

[15] M. Jackson: *A Discipline of Description (Keynote Talk)*; Requirements Engineering 3(2), S. 73-78, 1998.