Die SIKOSA-Methodik - Unterstützung der industriellen Softwareproduktion durch methodisch integrierte Softwareentwicklungsprozesse

The SIKOSA-Method – Support of the industrial Software Production by methodically integrated Software Engineering Processes

Dipl.-Kfm. techn. Daniel Weiß[1], Dipl.-Kfm. Jörn Kaack[1], Prof. Dr. Stefan Kirn[1],
Dipl.-Inf. Maike Gilliot[2], Dipl.-Inf. (FH) Lutz Lowis[2], Prof. Dr. Günter Müller[2],
Dr. Andrea Herrmann[3], MSc Carsten Binnig[3], Dipl.-Inf. Timea Illes[3], Prof. Dr. Barbara Paech[3]
Prof. Dr. Donald Kossmann[4]

[1] **Forschungszentrum Innovation und Dienstleistung (FZID)**
Universität Hohenheim
Schwerzstr. 35
70599 Stuttgart

[2] **Universität Freiburg**
Institut für Informatik und Gesellschaft, Abtl. Telematik
Friedrichstr. 50
79098 Freiburg

[3] **Universität Heidelberg**
Institut für Informatik, Lehrstuhl Software Engineering
Im Neuenheimer Feld 326
69120 Heidelberg

[4] **ETH Zürich**
Institut für Informationssysteme
Universitätstr. 6
8092 Zürich

Die SIKOSA-Methodik - Unterstützung der industriellen Softwareproduktion durch methodisch integrierte Softwareentwicklungsprozesse

Stichworte

Methodenintegration, Softwareentwicklungsprozesse, Geschäftsprozesse, Echtzeitsicherheit und Softwaretest-Automatisierung, Qualität

Keywords

Method Integration, Software Engineering Processes, Business Processes, Real Time Security, Software Test Automation, Quality

Abstract

Die SIKOSA-Methodik (Sichere und kollaborative Softwareentwicklung und Anwendung) adressiert die fehlende Durchgängigkeit von Qualitätssicherungsmethoden in Softwareentwicklungsprozessen. Zur Unterstützung einer industriellen Produktion von Unternehmenssoftware (USW) wurde eine durchgängige, qualitätsorientierte Methodik entwickelt. Ausgehend von den zu unterstützenden Geschäftsprozessen werden (nicht-)funktionale Anforderungen an die USW spezifiziert und in automatisierte Testfälle und -daten überführt.

Abstract

The SIKOSA method adresses the consistency of quality assuring methods in software engineering processes. To support an industrial business software production a new consistent and quality-oriented method was developed. Regarding business processes, which have to be supported, (non-)functional requirements of business software are specified and transferred into automated test cases and test data.

Kernpunkte

- Die SIKOSA-Methodik beschreibt einen durchgängigen Lösungsansatz zur Softwareentwicklung.
- Die Methodik überführt (nicht-)funktionale Prozessanforderungen in (nicht-)funktionale und testbare Unternehmenssoftware-Anforderungen und erhöht die Validität.

1 Motivation

Die Entwicklung von Unternehmenssoftware (USW) unterliegt einem grundlegenden Wandel - von einer personalintensiven, der Werkstattfertigung ähnlichen Vorgehensweise, hin zu einer industriellen Form der Fertigung mit geringen Fertigungstiefen, standardisierten Prozessen und Methoden. Als Hauptgrund lassen sich die offenbar nicht zu beseitigenden Qualitätsprobleme und damit einhergehend Produktivitätsdefizite bei der "werkstattorientierten" Softwareerstellung identifizieren: So geben US-Unternehmen jährlich über 250 Mrd. US$ für Softwareentwicklung (SE) aus; 31% dieser Projekte scheitern aufgrund von Qualitätsproblemen. Das entspricht einem Betrag von über 81 Mrd. US$ [GrSh03]. Dieser Befund gilt, obwohl die SE in allen ihren Phasen schon lange über ein umfangreiches Methodenrepertoire verfügt. Als Schwachpunkt erweist sich regelmäßig die unzureichende Integration der Einzelmethoden und spiegelt sich in der Folge in den für die SE verwendeten Werkzeugen (Eclipse, etc.) wider. Dieses Defizit wirkt sich bis zum Softwaretest hin aus. Dieser kann bei einer nur unzureichenden Methodenintegration über die vorhergehenden Phasen oft nur eingeschränkte Ergebnisse liefern. Im Wesentlichen lassen sich drei Teilprobleme identifizieren:

(1) Unzureichende Methodenintegration von der Geschäftsprozessanalyse bis zur Anforderungsspezifikation an USW, woraus sich Validierungsprobleme und Defizite bei automatisierten Softwaretests aufgrund der fehlenden Berücksichtigung der Geschäftsprozesssemantik ergeben.

(2) Die Testbarkeit von Software-Neuentwicklungen, insbesondere bei Änderungen betrieblich bereits genutzter Software und damit auch bei der Automatisierbarkeit des Softwaretests.
(3) Die überprüfbare Durchsetzung der erhobenen Sicherheitsanforderungen.

Der vorliegende Beitrag adressiert das Problem der Methodenintegration aus der spezifischen Perspektive der Automatisierbarkeit des Softwaretests in solchen Fällen, in denen mehrere Dienstleister in der Softwareerstellungsprozesskette zusammenarbeiten (kollaborative SE) und jeweils separate Methoden einsetzen. Forschungsmethodisch [HMPR04] wird an der Designwissenschaft Anlehnung genommen und dabei insbesondere das Instrument der Metamodellierung [Stra96] zur Anwendung gebracht. Ziel ist es, ein IT-Artefakt zu schaffen, mit Hilfe dessen nicht nur die Automatisierbarkeit des Softwaretests mittelfristig substantiell vorangebracht, sondern auf diese Weise auch zugleich die Qualität, Geschwindigkeit und Wirtschaftlichkeit der gesamten SE und deren Anpassungen deutlich verbessert werden kann. Bei der Entwicklung des Lösungsansatzes wird auf die bei den Forschungspartnern vorhandenen Methoden ProQAM, TORE und MOQARE zurückgegriffen. Diese unabhängigen Methoden decken den Bereich von der Prozessanalyse bis zur Ableitung testbarer Anforderungen als Input der Testfallerzeugung ab. Am Beispielszenario eines industriellen Beschaffungsprozesses wird die Anwendbarkeit bzw. Wirkungsweise der SIKOSA-Methodik aufgezeigt.

Kapitel zwei präsentiert die verwandten Arbeiten. Kapitel drei und vier beinhalten Lösungsansätze und Anwendungsbeispiele für die identifizierten Problemstellungen. Kapitel fünf behandelt zwei Lösungsansätze zur Überprüfung der Sicherheitsanforderungen: Der (automatisierte) Systemtest überprüft die Einhaltung der funktionalen Anforderungen. Der Sicherheitsmonitor zielt auf die Erkennung von Sicherheitsverletzungen während der Laufzeit ab. Kapitel sechs fasst die Ergebnisse zusammen und zeigt weiteren Forschungsbedarf auf.

2 Verwandte Arbeiten in der Literatur

Zur Entwicklung von USW bedarf es einer integrierten Analyse der Prozess- und USW-Anforderungen. Diese erfolgt zumeist top-down. Dabei lassen sich auf allen Granularitätsebenen funktionale Anforderungen (FA) und nicht-funktionalen Anforderungen (NFA) unterscheiden. Obwohl es nahe liegend erscheint, USW-Anforderungen aus Geschäftsprozessanforderungen abzuleiten (Geschäfts- und Prozessmodelle sowie immanentes Wissen bei der USW-Spezifikation sollen vollständig weiterverwendet werden), gibt es hierzu nur wenige Lösungsvorschläge [BrEn01; KoLi06a]. Als zu überwindende Hürden erweisen sich regelmäßig die unterschiedlichen Notation und Semantiken von Konstrukten der Geschäftsprozessspezifikation sowie die unterschiedlichen Qualitätsmodelle (incl. Metriken) zur Beschreibung der Qualität von Geschäftsprozessen und USW. Auch eine semantikerhaltende Überführung der Prozess- in IT-Attribute erweist sich als schwierig. Soll beispielsweise durch den Geschäftsprozess eine geforderte Wertschöpfung erbracht werden, leiten sich daraus u.a. Anforderungen bzgl. des Antwortzeitverhaltens an die USW oder auch Vorgaben für die akzeptierbaren Kosten der IT-Lösung ab. Ansätze zur Integration der Prozess- und USW-Anforderungsanalyse stellen bspw. [ScHa00; Nora04; KoLi06b] vor. Neuere Arbeiten weisen allerdings auf weiter bestehende Schwächen bzgl. der Methodenintegration hin [BrEn01; KoLi06a]. Auch konzentrieren sich die bisher verfügbaren Arbeiten zumeist auf FA. NFA werden in der Praxis gegenüber den FA hingegen vernachlässigt, obwohl sie im Hinblick auf gestiegene Kundenansprüche eine hohe Relevanz besitzen. Die hohe Bedeutsamkeit für Softwareanbieter ergibt sich aus der Tatsache, dass hoch-standardisierte USW zunehmend austauschbar wird, wodurch die Nutzer an Flexibilität hinzugewinnen. Dies bedeutet, dass der Kundenwettbewerb nicht mehr allein durch den Funktionsumfang der Softwarelösung gewonnen werden kann, sondern vielmehr durch deren Qualität. Daher fokussiert die SIKOSA-Methodik insbesondere den Bereich der NFA.

Parallel dazu haben sich in den letzten Jahren automatisierte (Regressions-)Tests für die Qualitätssicherung von Softwareänderungen weitgehend durchgesetzt [GrFe99]. Insbesondere für datenbankbasierte USW stehen jedoch bisher keine vergleichbaren Lösungen zur Verfügung. Dieses stellt die SE vor nicht zu unterschätzende Herausforderungen, da der Großteil der USW gerade datenbankgestützt arbeitet.

Bisher müssen für den Test derartiger Systeme zuvor Testdatenbanken generiert werden, was in hochdynamischen Anwendungen kaum realitätsnah möglich ist. Zudem bestehen dabei auch weitere methodische Probleme, da oft nur das Datenbankschema adressiert wird [CDFD04] oder auf Basis einer statistischen Verteilung zufällige Daten generiert werden [BrCh05]. Letztere reichen jedoch nicht aus, um eine große Zahl kritischer Pfade der Anwendung abzudecken, was dazu führen kann, dass die Aussagekraft des Tests leidet und nicht zuletzt auch die Validität beeinträchtigt wird.

3 Von der Prozessanalyse zur Ableitung testbarer USW-Anforderungen: Die SIKOSA-Methodik

Die SIKOSA-Methodik umfasst Methoden für die Prozessanalyse und -spezifikation (ProQAM), das Requirements Engineering (TORE), die Echtzeitanalyse von Sicherheitsschwachstellen (Monitoring) und die Ableitung direkt testbarer Systemeigenschaften (MOQARE) sowie ein sich über diese Methoden erstreckendes integriertes Vorgehensmodell. ProQAM, TORE und MOQARE sind unabhängig voneinander entstanden und simulieren kollaborative SE für verschiedene SE-Phasen.

3.1 ProQAM - Process-oriented Questionnaire for Analyzing and Modeling Scenarios

Die Methode ProQAM unterstützt durch ein Fragebogenkonzept die empirische Analyse von Geschäftsprozessen und modelliert diese auf Basis von drei Metamodellen: *Aufbauorganisation, Prozesslandschaft und -qualität.* Zur Beschreibung der Prozesslandschaft dient das Metamodell der eEPK in Anlehnung an [Sche02] mit den Basiskonstrukten: *Ereignisse, Funktionen, Kontrollflusskanten* und *Verknüpfungsoperatoren.* Weitere Entitäten des Metamodells stellen die als *Prozessinput* oder *-output* modellierbaren *Umfelddaten* des Prozesses sowie die zur Ausführung des Prozesses notwendigen *Ressourcen, (Vor-)Leistungen* (in Form von Sach-, Dienst- und Informationsdienstleistungen, Finanzmitteln) und *Geschäftszielen* dar. Der *Organisationseinheit*, den *Rollen*, den *Stellen* und den *Stellentypen* kommt dabei entscheidende Bedeutung zu [BFOW06].

Ähnlich den bereits vorgestellten Metamodellen basiert das Qualitätsmetamodell ebenfalls auf Funktionen und Geschäftszielen (Sach- und Formalzielen). Auf der Basis von drei Klassen von *Qualitätsattributen* werden Qualitätsanforderungen an die Funktion bzw. den gesamten Prozess formuliert. Die Auswahl des Qualitätsattributes bzw. die Formulierung der NFA wird unmittelbar durch die Geschäftsziele beeinflusst. Die Nichterfüllung einer NFA führt zu einem *Qualitätsmangel* der Funktion bzw. des Prozesse und birgt damit ein *Unternehmensrisiko*. Das Prozessrisiko wird grundsätzlich aus dem potenziellen Unternehmensschaden und dessen Eintrittswahrscheinlichkeit ermittelt. Die drei Qualitätsattributklassen – strukturelle, formale und inhaltliche Attribute – beschreiben unterschiedliche Dimensionen der Prozessqualität. Während strukturelle Qualitätsattribute wie die Einhaltung regulatorischer Richtlinien (engl. Compliance), die Komplexität oder der Modularisierungsgrad sich nur indirekt auf die in den Prozessen verwendete USW auswirken, haben die formalen und inhaltlichen Qualitätsattribute direkten Einfluss auf die Auswahl. Sie leiten sich aus dem Zweck (Entwicklung, Beschaffung, usw.) der Prozesse ab und adressieren u.a. deren Effizienz, Flexibilität oder Integrität. Damit besitzen die formalen und inhaltlichen Qualitätsattribute unmittelbaren Einfluss auf die Sicherheit der verwendeten USW und deren Integrität, die sich bis in den automatisierten Test hinein durchschlägt.

3.2 TORE- Task Oriented Requirements Engineering

TORE stellt eine der wenigen Methoden des Requirements Engineering (RE) dar, die explizit Anwendungsanforderungen aus Prozessen herleiten [PaKo03] und kennt vier Ebenen der Analyse. TORE leitet durch Diskussion mit den Interessengruppen auf seiner Interaktionsebene die Anforderungen an die Benutzerschnittstelle aus den ProQAM-Prozessbeschreibungen und Systemverantwortlichkeiten her. Hieraus werden auf der Systemebene dann konkrete, funktionale Anforderungen an den Anwendungskern und die grafische Benutzeroberfläche erstellt. Das zentrale Konzept der Schnittstelle zur Analyse der NFA und zur Herleitung der Testfallspezifikationen stellen die Anwendungsfälle her.

3.3 MOQARE - Misuse Oriented Quality Requirements Engineering

MOQARE dient zur Konkretisierung von Qualitätszielen sowie zur Herleitung realisierbarer und testbarer Qualitätsanforderungen (NFA) bzw. Unterstützung der Qualitätsziele [HePa06]. Die MOQARE-Analyse beginnt mit den *Geschäftszielen,* die durch einen *Geschäftsschaden* bedroht werden können. Dieser beschreibt qualitativ den Schaden, während das *Unternehmensrisiko* ihn quantifiziert und definiert ist als Wahrscheinlichkeit des Geschäftsschadens mal der Schadenshöhe. Der Geschäftsschaden wird wiederum durch einen Qualitätsmangel der USW verursacht. Ein *Wert* ist jedes zu schützende Teil des Systems und beinhaltet alle durch ProQAM erfassten Elemente der Ablauf- und Aufbauorganisation. Für jeden Wert wird definiert, in Bezug auf welches *informationstechnologische Qualitätsattribut (IT-QA)* er geschützt werden soll. Daher betrachten wir NFA in der Form von *Qualitätszielen*, die jeweils eine Kombination aus einem Wert und seinem IT-QA darstellen. Erfüllt der Wert nicht das IT-QA, wird von einem Qualitätsmangel gesprochen. Zu beachten ist, dass die Qualitätsmängel und -ziele (auf IT bezogene) IT-QA verwenden, wie sie durch die ISO 9126 [ISO1991] beschrieben werden. Eine *Bedrohung* ist eine Aktion, die das Qualitätsziel bedroht und zugleich die Ursache des Qualitätsmangels darstellt. Verursachende *Fehlnutzer* können sein: Personen (Hacker, Benutzer, Administrator, usw.), andere Systeme oder Naturgewalten.
Sicherheit ist ein Spezialfall, bei dem der Fehlnutzer entweder (1) ein Eindringling ist oder (2) ein Benutzer, der eine für ihn nicht vorgesehene Funktion ausführt und dabei ein schädliches Ziel verfolgt. Ebenso kann (3) ein nachlässiger Benutzer die Sicherheitsregeln verletzen. Alle anderen IT-QA der ISO 9126 werden durch (4) normale Systembenutzer bedroht. Oft wird die Bedrohung durch eine *Schwachstelle* erleichtert, ermöglicht oder gar provoziert. Eine *Fehlnutzung* beschreibt dabei das gesamte Fehlnutzungs-Szenario inklusive Fehlnutzer, Schwachstelle, Bedrohung und dessen Folgen (Qualitätsmangel). Sie wird in Form und Granularität eines *Fehlnutzungsanwendungsfalls* dokumentiert. Um den Bedrohungen und Fehlnutzungen zu begegnen und die Qualitätsziele (bzw. die Geschäftsziele) zu schützen, definieren wir *Gegenmaßnahmen*, die meist neue Anforderungen darstellen. Gegenmaßnahmen können eine Fehlnutzung entdecken, verhindern oder abschwächen.
Solche Gegenmaßnahmen können (1) neue funktionale Anforderungen (z.B. Anwendungsfall) sein, (2) erweiterte oder neue Ausnahmeszenarien von Anwendungsfällen, (3) NFA an FA, (4) Architekturanforderungen oder (5) andere NFA. Die Fehlnutzungen können durchaus außerhalb der USW ihre Ursache haben. Sie führen zu Gegenmaßnahmen, die nicht durch die USW realisiert werden, sondern Anforderungen an den Betrieb, die Wartung oder den SE-Prozess darstellen. Das entstandene Metamodell enthält auch ausgewählte Entitäten aus der Sicherheits-Schwachstellenanalyse, da bei der Entwicklung von MOQARE die Fehlnutzungsanwendungsfälle [SiOp01; Alex03] berücksichtigt wurden. Sowohl TORE als auch MOQARE wurden bereits in Fallstudien evaluiert [HeRP06] und in dem Werkzeug „Sysiphus“ [Sysi06] implementiert.

3.4 Metamodell der Geschäftsprozesse, der IT-Anforderungen und Testfallspezifikationen

Die Methoden ProQAM, TORE und MOQARE sowie die Sicherheitsanalyse und Testfallspezifikationen wurden integriert, um eine methodisch durchgängige Abbildung von der Prozessanalyse und -spezifikation bis zum Datenbanktest zu erreichen. Eine formalisierte Darstellung der Integration gibt Bild 1 wieder. Dort sind v.a. die Konzepte enthalten, welche die Schnittstellen zwischen den einzelnen Einzelmethoden darstellen. Die methodischen und technischen Details der Lösungsansätze sowie die Details der Integration werden in [KPMK06] beschrieben.

4 Anwendungsbeispiel

Zur weiteren Erläuterung verwenden wir das exemplarische Szenario eines idealtypischen industriellen Beschaffungsprozesses [Sche97]. Szenarien spielen bei der Anforderungsbeschreibung eine zentrale Rolle und beschreiben eine Anzahl potenzieller Ereignisse, deren Eintritt als realistisch erscheint. Sie beinhalten Änderungsziele und dienen als Stimulation und Dokumentation von relevanten Problemen, möglichen Vorfällen und damit verbundenen Annahmen, Aktionsmöglichkeiten und Risiken [Jark99]. Im RE sind sie für die Beschreibung von funktionalen USW-Anforderungen Stand der Technik [WPJH98] (z.B. als Anwendungsfall) [Cock01]) und ermöglichen zudem die Darstellung und Bewertung von NFA [LaRV99], der Architekturqualität [BoMo99] und die an sie gestellten Anforderun-

gen. Die Notwendigkeit, Szenarien als Grundlage für den Systemtest zu verwenden, wird in [WPJH98] hervorgehoben.

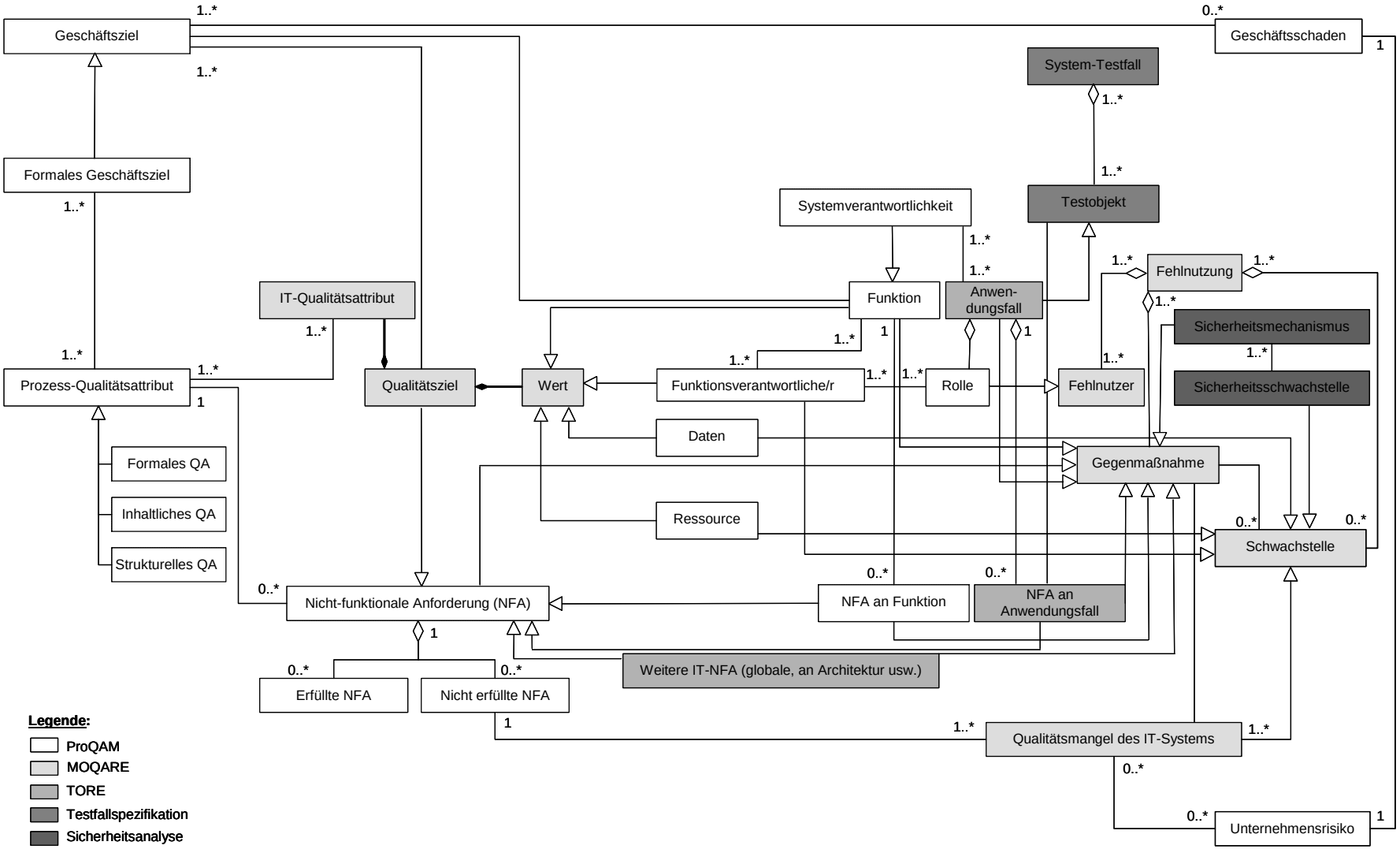

Bild 1 Integriertes Metamodell des Lösungsansatzes

4.1 Prozessanalyse - ProQAM

Die Prozessanalyse mit ProQAM beinhaltet die Analyse und Spezifikation der Geschäftsziele, -schäden und -prozesse (inklusive der Systemverantwortlichkeiten), die NFA an diese Prozesse sowie die Struktur der Aufbauorganisation. Die dafür notwendigen Informationen sind in den Unternehmen oftmals auf mehrere Rollen verteilt (Prozessingenieure, Manager). Beispielhaft zu erfüllende Qualitätsanforderungen an Geschäftsziele eines regulären Beschaffungsprozesses sind:

- Logische Integrität der Lieferantendaten > 0,90 (mehr als 90% der Informationsobjekte werden exklusiv von dem betrachteten Prozess genutzt).
- Wertschöpfung durch die Einführung der neuen USW von X Euro.

Die Wertschöpfung ergibt sich z.B. aus den drei Geschäftsunterzielen:

- Durchschnittliche Zeitersparnis bei der Prozessausführung > 10 Minuten/Prozessinstanz
- Ersparnis bei den laufenden Kosten von > 20.000 Euro/Monat
- Kosten durch schlechte Datenqualität (Datenbereinigungsmaßnahmen, ungünstiger Einkauf) sinken um 10.000 Euro/Monat

Jedes dieser Geschäftsziele wird durch mindestens einen Geschäftsschaden bedroht. Dieser kann einfach ein Gegenteil des Geschäfts(unter)ziels sein, z.B. die „Ersparnis bei den laufenden Kosten von > 20.000 Euro/Monat“ wird durch

- Wartungszeiten verringern sich nicht ausreichend,
- Ersatzteilkosten der Hardwareinfrastruktur zum Betrieb der USW sind höher als erwartet,
- Lizenzkosten sind höher als erwartet, bedroht.

Das Geschäftsunterziel „Durchschnittliche Zeitersparnis bei der Prozessausführung > 10 Minuten/Prozessinstanz“ wird durch den Geschäftsschaden „Zeitersparnis im gesamten Prozess ist 10 Minuten oder weniger“ bedroht, das Geschäftsunterziel „Kosten durch schlechte Datenqualität um 10.000 Euro/Monat senken“ durch den Geschäftsschaden „entsprechende Kosteneinsparungen sind geringer“.

4.2 Analyse der funktionalen (TORE) und nicht-funktionalen (MOQARE) USW-Anforderungen

Während der Analyse der USW-Anforderungen wird das Geschäftsprozessmodell durch Anforderungen an die USW ergänzt. Diese werden durch Anwendungsfälle [Cock01] erfasst und lassen sich aus den Systemverantwortlichkeiten, den beteiligten Rollen, den Daten usw. ableiten. Das Ergebnis einer MOQARE-Analyse wird - neben einer tabellarischen - auch in einer hierarchischen Darstellungsweise festgehalten - einem durch Fehlnutzung charakterisierten Fehlernutzungsbaum. Die Geschäftsziele und möglichen Geschäftsschäden sind bereits während der ProQAM-Analyse erfasst worden. Nicht alle denkbaren Bedrohungen und Gegenmaßnahmen werden hier dokumentiert, sondern nur diejenigen, die für das konkrete IS als besonders relevant angenommen werden. Zugrunde liegen Fehlnutzungen, die als sehr wahrscheinlich (schädlich) einzustufen sind, ebenso deren effektive Beseitigung. Bezogen auf die Methode ProQAM hängt das dort erwähnte Geschäftsunterziel „Durchschnittliche Zeitersparnis bei der Prozessausführung > 10 Minuten/Prozessinstanz“ bzw. der Geschäftsschaden „Zeitersparnis im gesamten Prozess ist 10 Minuten oder weniger“ von der Zeitersparnis innerhalb der verschiedenen Anwendungsfälle ab. Überträgt man dies auf den Anwendungsfall "Bestellung aufgeben", so lautet einer der Qualitätsmängel daher: „Zeitersparnis im Anwendungsfall "Bestellung aufgeben" ist 60 Sekunden oder weniger“ und das zugehörige Qualitätsziel Dauer des Anwendungsfalls "Bestellung aufgeben" unter normalen Bedingungen < 8 Minuten“. Wird beispielhaft ein Zweig aus einem Ausschnitt des hierarchischen Fehlernutzungsbaum herausgegriffen, bedeutet dies: Dem <<*Qualitätsziel: „Dauer des Anwendungsfalls "Bestellung aufgeben" unter normalen Bedingungen < 8 Minuten“*>> folgt eine <<*Fehlnutzung: In der Spezifikation geforderte Zeitersparnis zu gering*>> und eine <<*Ge-*

genmaßnahme: Spezifizierte Durchlaufzeit des Anwendungsfalls "Bestellung aufgeben" beträgt 8 Minuten>> (Bild 2).

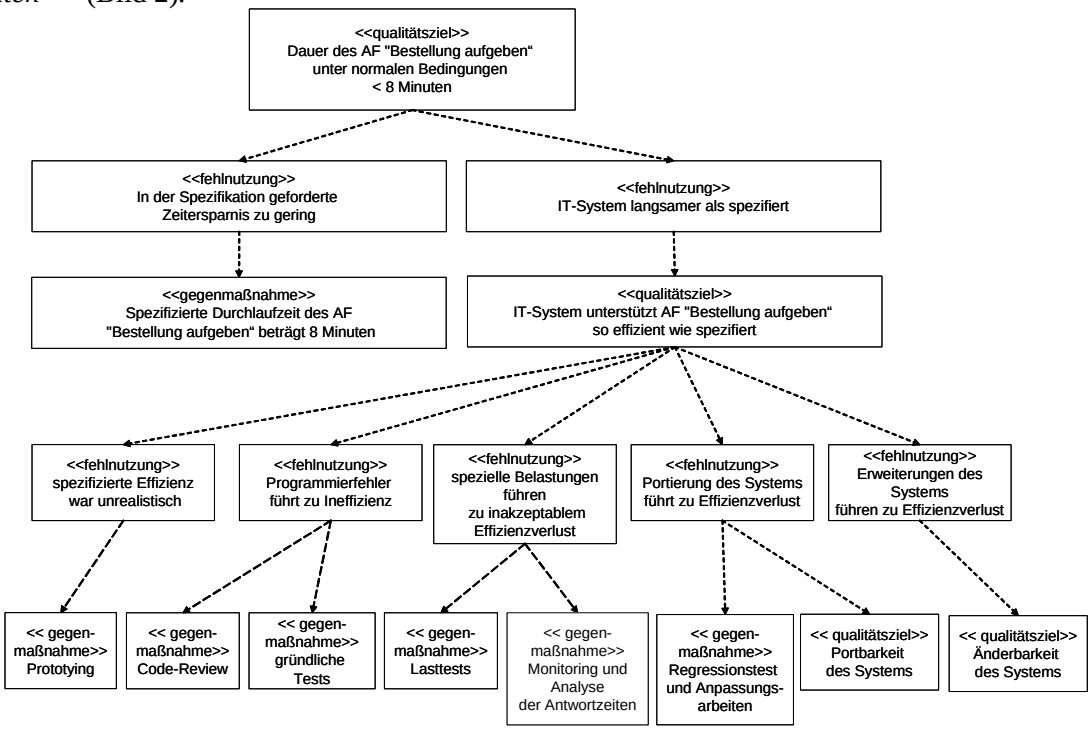

Bild 2 Fehlnutzungsbaum für das Qualitätsziel Dauer des Anwendungsfalls "Bestellung aufgeben"

5 Sicherheitsmonitoring und Systemtest

Durch Testen und Monitoring lässt sich die Einhaltung der mit MOQARE und TORE erhobenen Anforderungen überprüfen. Der automatisierte Systemtest dient der Überprüfung der funktionalen Anforderungen, der Sicherheitsmonitor der Erkennung von Sicherheitsverletzungen zur Laufzeit.

5.1 Sicherheitsmonitoring für USW

Führende Softwarehersteller wie SAP, IBM, Microsoft und Oracle bieten bereits auf SOA (Service-Orientierte Architektur) basierende USW an. Ein noch ungelöstes Problem stellt dabei jedoch die Durchsetzung von Sicherheitsanforderungen und vor allem die Konformität zu neuen Regularien dar. Dazu wird ein Monitor vorgeschlagen, der unter Berücksichtigung von Schwachstellen die Einhaltung von Sicherheitsanforderungen zur Laufzeit erzwingt.

Sicherheitsfehler in dynamischen Anwendungssystemen

SOA vereinfacht die flexible Einbindung von Kommunikationspartnern und Diensten, so dass zum Beispiel bei E-Procurement mit wechselnden Kommunikationspartnern spontan verhandelt werden kann. Dies führt zum einen zu Problemen bzgl. der Geheimhaltung (engl. privacy), ob der eingebundene Dienst mit den Daten und Rechten im Sinne der Richtlinie (engl. policy) umgeht [WoMü06]. Darüber hinaus erfordern diese Änderungen (Hinzunahmen von Diensten und Kommunikationspartnern) an der USW aber auch eine erneute Überprüfung (und gegebenenfalls die Anpassung) der Zugriffsregeln für das geänderte Softwaresystem. Derzeitige USW-Lösungen bieten jedoch keine ausreichenden Mechanismen für diese wiederholt notwendige Sicherheitsüberprüfung. Um dynamische, unternehmensübergreifende Kooperationen zu sichern, müssen die Sicherheitsbeziehungen flexibler gestaltet werden und die Sicherheitsüberprüfung effizient möglich sein. Sicherheitsschwachstellen lassen sich dabei nach dem Zeitpunkt, zu dem sie festgestellt werden, unterscheiden: Vor, während und nach der Abwicklung des Geschäftsprozesses. Der Fehlerzeitpunkt ist insofern von Bedeutung, als vom Zeitpunkt der Fehlerentdeckung die Reaktionsmöglichkeiten auf den Fehler abhängen.

- **Vor der Abwicklung:** Die häufigsten Methoden zur Sicherheitsüberprüfung vor dem Einsatz der Anwendung sind Testen und die formale Spezifikation und Verifikation der Software. Die Methoden der „Vorher-Sicherheit“ gehen davon aus, dass alle möglichen Verhaltensweisen der USW fest und bekannt sind. Sicherheitsrelevante Fehler, lassen sich zu diesem Zeitpunkt z.B. mittels geänderten Zugriffsregeln oder zusätzliche Sicherheitsmechanismen beheben.
- **Während der Abwicklung:** Für die Überprüfung von Sicherheitseigenschaften während der Abwicklung eignen sich Referenzmonitore oder auch um Sicherheitstypen ergänzte Programmiersprachen (z.B. C#). Dadurch werden Aktionen während ihrer Ausführung beobachtbar und beurteilungsfähig. Viele Sicherheitsschwachstellen werden erst während der Laufzeit der Anwendung deutlich und machen damit eine erneute Überprüfung der Sicherheitsanforderungen notwendig. Jedoch existiert derzeit kein Mechanismus, der zur Laufzeit auftretende Sicherheitsfehler betrachtet und Reaktionsmöglichkeiten auf diese Sicherheitsfehler anbietet. An diesem Punkt setzt der Sicherheitsmonitor an.
- **Nach der Abwicklung:** Durch das Auditieren (geloggter Informationen) kann nach der Abwicklung des Prozesses überprüft und auch gegenüber Dritten bewiesen werden, dass die Anforderungen bzgl. Sicherheit und Geheimhaltung eingehalten wurden [SaSA06]. Wurden Regeln nicht eingehalten, sind Verfahren aus der Informatik nur noch begrenzt möglich. Meist ist ein automatisiertes Zurücksetzen nicht möglich, da physikalische Veränderungen im Unternehmen vorgenommen wurden. Daher sind diese Methoden für USW nur in beschränktem Maß sinnvoll einsetzbar und werden nicht weiter betrachtet.

5.1.1 Sichere Unternehmensinteraktion durch schwachstellensensitives Monitoring

Zum Umgang mit zur Laufzeit entdeckten Sicherheitsfehlern wird ein schwachstellensensitiver Sicherheitsmonitor „Vulnerability Evaluation Point“ (VEP) vorgeschlagen. Dieser betrachtet bei der Auswertung von Zugriffsanfragen nicht nur die Zugriffskontrollregeln, sondern auch die Schwachstellen der USW. Der VEP-Monitor ist eine Ergänzung zur regelbasierten Zugriffskontrolle, wie z.B. von XACML-basierten Zugriffsmechanismen[1], woraus sich ein zweistufiger Zugriffskontrollmechanismus ableitet: Im ersten Schritt wird eine eingehende Zugriffsanfrage gemäß der Zugriffskontrollregeln überprüft. Ist die Anfrage gemäß den Zugriffsregeln zulässig, überprüft der VEP-Monitor, ob der gewünschte Zugriff eine bekannte Schwachstelle ausnützen könnte. Dabei greift er auf eine Schwachstellendatenbank zurück, welche die bekannten Schwachstellen der Komponenten des Anwendungssystems enthält. Könnte der angefragte Zugriff eine Schwachstelle ausnutzen und unberechtigte Zugriffe oder Rechte zur Folge haben, lehnt der VEP-Monitor den Zugriff ab, obwohl die Anfrage gemäß den Zugriffsregeln zulässig ist. Somit lassen sich sicherheitskritische Anfragen, die sich während der Abwicklung des Geschäftsprozesses ergeben, entdecken und verbieten. Bild 3 beschreibt den Aufbau des **schwachstellensensitiven Sicherheitsmonitors**:

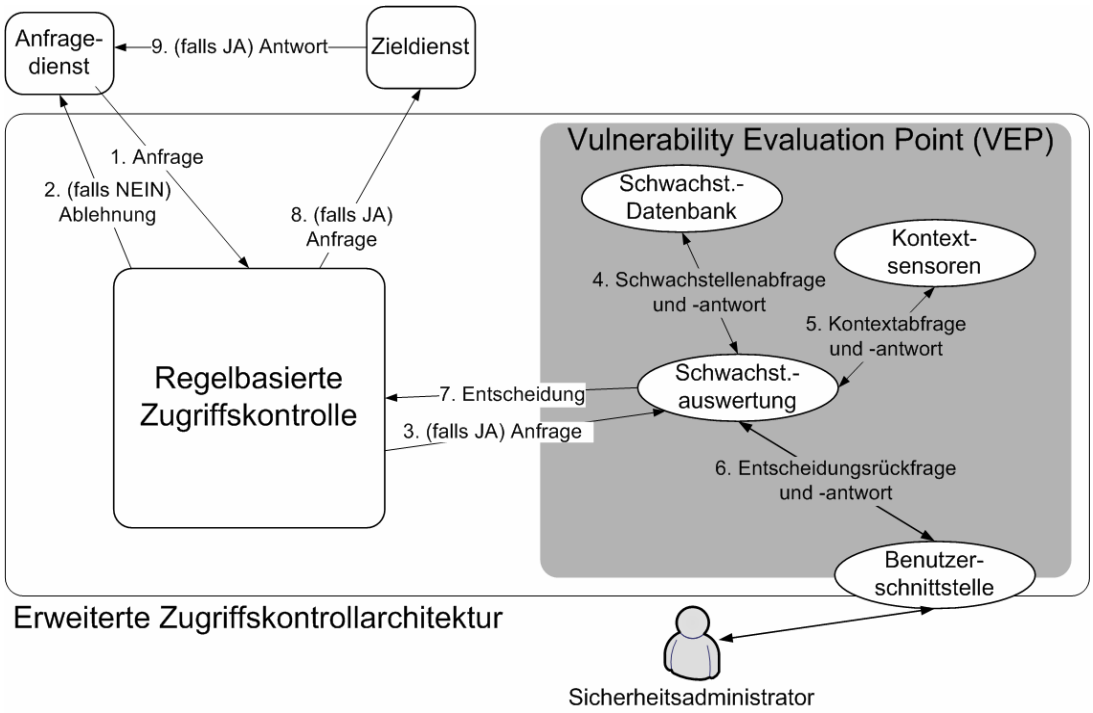

Bild 3 Aufbau des Sicherheitsmonitors VEP

In **Schritt 1** wird die Anfrage nach den Zugriffskontrollregeln bewertet. Lehnt der regelbasierte Zugriffskontrollmechanismus die Zugriffsanfrage ab, wird der anfragende Dienst über die Ablehnung informiert (**Schritt 2**). Der VEP-Monitor wird in diesem Fall nicht genutzt. Gestattet der regelbasierte Zugriffskontrollmechanismus den Zugriff jedoch, dann nimmt in **Schritt 3** die Auswertungseinheit die Anfrage entgegen. Über Rückfragen an die Schwachstellendatenbank (**Schritt 4**) und die Kontextsensoren (**Schritt 5**) erhält die Auswertungseinheit alle zur Bewertung der Dienstanfrage möglicherweise nötigen Zusatzinformationen wie Versionsnummern der beteiligten Dienste, parallel laufende Anwendungen und Informationen über Schwachstellen. Der VEP-Monitor ist über die Benutzerschnittstelle mit dem Sicherheitsadministrator verbunden. Über die **Benutzerschnittstelle** sind die Aktionen des VEP-Monitor beobachtbar und, falls dies vom Administrator gewünscht wird, auch kontrollierbar. Falls der Onlinemonitor aufgrund seiner Informationen keine Entscheidung treffen kann, fragt er beim Administrator nach und dieser entscheidet dann über die Ablehnung oder Zulassung einzelner Anfragen manuell (**Schritt 6**). Das Ergebnis des VEP-Monitors wird in **Schritt 7** wieder an die regelbasierte Zugriffskontrollmechanismus zurückgegeben. Bei einer Ablehnung der Zugriffsanfrage wird dies dem anfragenden Dienst mitgeteilt (wie in Schritt 2). Wird der Zugriff gewährt, wird die Anfrage in **Schritt 8** an den angefragten Dienst weitergereicht. Erst zur Laufzeit bekanntes Schwachstellenwissen kann auf diese Weise in die Zugriffskontrollentscheidung miteinbezogen werden.

5.2 Lösungsansatz für Systemtests

Bild 4 gibt einen Überblick über die unterschiedlichen Abstraktionsebenen, bei denen Testentscheidungen gefällt werden. Entscheidungen auf niederer Ebene bedingen zunächst eine Entscheidung auf nächst höherer Ebene. Auf der **Anforderungsebene** werden testbare Anforderungen formuliert und entschieden, ob die Testbarkeit der Anforderungen vorliegt bzw. diese bei Bedarf herzustellen ist. Weiterhin werden auf dieser Ebene die zu testenden Entitäten identifiziert. Da Testaktivitäten 30 – 40% der SE-Aktivitäten darstellen [Spil00], ist eine sorgfältige Planung ein wichtiger Bestandteil des Testprozesses. Auf der **Testzielebene** werden zum einen die zu testenden Entitäten und zum anderen die Methoden für die systematische Definition von Testfällen, Testendekriterien, u.a festgelegt. Die **logische Testebene** detailliert die definierten Ziele. Hier werden die logische Umgebung (Entscheidung über das zu testende Zusammenspiel zwischen System und Fremdsystemen), die logischen Daten (Entscheidungen über Eingabedaten, Vor- und Nachbedingung eines Testfalls), die Schritte (Entschei-

dung über die auszuführenden Schritte eines Testfalls) sowie die Testsequenzen (Entscheidung über die Reihenfolge der auszuführenden Testfälle) definiert. Auf der **Ausführungsebene** werden alle Entscheidungen getroffen, die zur Ausführung von Testfällen notwendig sind. Die logischen Testfälle werden hier konkretisiert, wobei für einen logischen Testfall mehrere konkrete Testfälle definiert werden können. Entscheidungen auf dieser Ebene betreffen die konkreten Umgebungen, die getestet werden müssen, die konkreten Daten sowie die konkreten Testschritte und -sequenzen. Auf der **Auswertungsebene** werden alle Entscheidungen hinsichtlich des Ergebnisses des Testens getroffen und geprüft, ob ein Testfall erfolgreich war oder nicht, welche Fehler aufgetreten sind, oder ob das Testende erreicht wurde und die Testaktivitäten abgebrochen werden können.

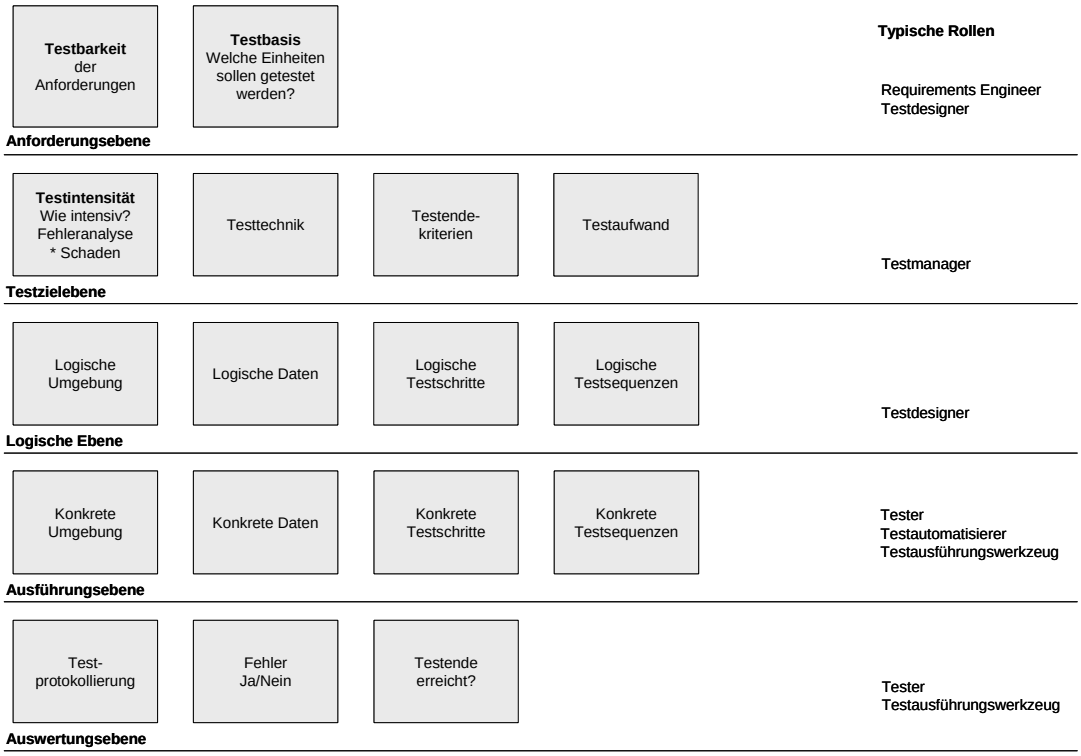

Bild 4 Ebenen des Testens

5.2.1 PAT³ – Musterbasierte Testfallermittlung aus Anwendungsfällen

Zur Ermittlung der Testfälle wird der Lösungsansatzes PAT³ (Process, Automation, Testability, Transformation, Traceability) verwendet, welcher seinerseits mit Mustern arbeitet. *Muster* stellen ein bewährtes Mittel zur Dokumentation, Wiederverwendung und Kommunikation von Erfahrungen dar und wurden im SE für Probleme im Software Design [GHJV95] oder im RE [WGRE04] erfolgreich eingesetzt. Ausgangspunkt aller Muster sind *Fehlerklassen*, die durch das Muster adressiert werden sollen. Da Testaktivitäten ausgeführt werden um Fehler zu finden, ist es unabdingbar zu wissen, welche Fehlerklassen durch die unterschiedlichen Muster adressiert werden. Bisherige Ansätze vernachlässigen diesen Aspekt, wobei das Wissen über den Erfolg einer bestimmten Technik (Fehler gefunden: Ja/Nein) sehr wichtig ist. Weiterhin decken bisherige Ansätze nicht alle Fehlerklassen ab, da nur spezielle Aspekte betrachtet werden [IlPa06a; IlPa06b]. PAT³ unterscheidet 5 *Musterkategorien*, welche die wichtigsten Probleme an der Schnittstelle zwischen Anforderungen und Test adressieren:

- **Prozessmuster** unterstützen die Optimierung der Prozessschnittstelle zwischen Anforderungen und Test. Beispielsweise gibt ein Muster eine mögliche Lösung zur Kopplung der Anforderungs- und Testprozesse durch Einbezug der Tester beim Review der Anforderungen an.

- **Automatisierungsmuster** unterstützen den Auswahl- und Anwendungsprozess von Testwerkzeugen. Beispielsweise wird der Auswahlprozess durch die Vorgabe von Selektionskriterien für eine Vorauswahl erleichtert.
- **Testbarkeitsmuster** unterstützen die Formulierung von testbaren Anforderungen. Ein Beispiel für ein Muster dieser Kategorie stellt die Bereitstellung von linguistischen Regeln zur genauen und damit testbaren Formulierung der Anforderungen dar.
- **Transformationsmuster** unterstützen die Ableitung von Testfällen aus Anforderungen. Transformationsmuster geben Schritt-für-Schritt Anleitungen zur Ableitung von Testfällen aus Anforderungen (Abbildungsmuster) vor. Weiterhin schlagen sie eine geeignete Darstellung für die Testfälle (Repräsentationsmuster) sowie mögliche Abdeckungskriterien (Abdeckungsmuster) vor. Beispielsweise gibt das Muster zum Aufdecken von Kontrollflussfehlern bei der Implementierung der Anforderungen schrittweise Anleitungen zur (1) Ableitung des in den Anforderungen spezifizierten Kontrollflusses (Abbildungsmuster) und (2) dessen Darstellung in Form eines Aktivitätsdiagramms (Repräsentationsmuster) sowie (3) mögliche Abdeckungskriterien, die angeben, wann Testaktivitäten eingestellt werden sollen, wie z.B. Entscheidungsüberdeckung. Entscheidungsüberdeckung fordert die Überdeckung jeder Entscheidung im Aktivitätsdiagramm durch mindestens einen Testfall.
- **Nachvollziehbarkeitsmuster**: Unterstützen die Nachvollziehbarkeit zwischen Anforderungen und Testfällen. Die Erstellung und Wartung von Nachvollziehbarkeitsverweisen ist sehr teuer. Um ein gutes Kosten-/Nutzenverhältnis zu gewährleisten, bietet z.B. ein erarbeitetes Muster Unterstützung für die Identifikation der zu erstellenden Links, die sich an deren Zielen und Abnehmern orientieren.

5.2.2 Generierung von Testdatenbanken

Da heutige USW praktisch ausschließlich datenbankgestützt arbeitet, kann nur getestet werden, wenn zuvor eine Testdatenbank generiert wurde (bei Neuentwicklungen stehen Realdaten nicht immer zu Verfügung oder unterliegen bspw. dem Datenschutz). Das bedeutet zumeist einen hohen manuellen Aufwand. Zwar existieren Werkzeuge, die aufbauend auf ein bestehendes Datenbankschema zufällige Daten generieren, doch zum Testen von USW sind diese Werkzeuge ungeeignet, da die erzeugten Daten nicht auf den spezifischen Bedarf der USW abgestimmt sind. Zur automatisierten Erzeugung von Testdatenbanken für USW dient das *Reverse Query Processing* (RQP) [BiKL07]. Dieses ermöglicht es, zusätzlich zum Datenbankschema Abhängigkeiten als logische Ausdrücke der Eingabe zu berücksichtigen und diese zur betriebswirtschaftlich *sinnvollen Testdatenerzeugung* zu verwenden. Solche logischen Ausdrücke können zum einen aus der logischen Testspezifikation gewonnen werden, zum anderen aber auch durch die Analyse des USW-Programmcodes. Beispiele hierzu und Informationen zum Prototypen können [BiKL06; BiKL07] entnommen werden. Technisch gesehen bekommt RQP als Eingabe eine Anfrage, eine Spezifikation von möglichen Anfrageergebnissen und ein Datenbankschema und liefert als Ausgabe eine Testdatenbank. Die Spezifikation der Anfrageergebnisse und der Anfrage dient dazu, für die USW und ihre Anwendung sinnvolle Testdatenbanken zu erzeugen. RQP basiert auf zwei Kerntechniken *Modellüberprüfung* und *Datenbankanfragebearbeitung:* Die Modellüberprüfung wird verwendet, um neue Daten zu generieren, die bestimmte Bedingungen erfüllen. Das klassische Modell zur Bearbeitung von Datenbankanfragen mittels Parsing, semantischer Analyse, logischer Optimierung und Ausführung durch eine Fließbandverarbeitung wird angewendet und erweitert, um sehr große Datenmengen (bis zu Terrabytes) mit akzeptablen Laufzeiten zu erzeugen. Details von RQP werden in [BiKL07] beschrieben. RQP arbeitet (vereinfacht) mit einem schrittweisen Vorgehen:

- *Schritt 1:* Die Eingabeanfrage wird geparst. Das Ergebnis ist ein Ausdruck der so genannten Reverse Relational Algebra (RRA), die auf der klassischen relationalen Algebra beruht.
- *Schritt 2:* Der Ausdruck der RRA wird annotiert mit logischen Ausdrücken (Bedingungen).
- *Schritt 3:* Der annotierte Ausdruck der RRA wird optimiert, in dem er in einen äquivalenten Ausdruck mit kleinerer Laufzeit transformiert wird. Das Ergebnis ist ein Ausführungsplan.

- *Schritt 4:* Der Ausführungsplan wird in einem Fließbandverarbeitungsmodell interpretiert. Hierbei wird, falls nötig, eine Modellüberprüfung aufgerufen, um Testdaten zu genieren.

5.2.3 Ausführung von Tests

Zur Ausführung von Testfällen werden in der Praxis überwiegend Capture&Replay - Werkzeuge verwendet. Danach werden in der Capture-Phase die Testfälle und zu erwarteten Ergebnisse aufgezeichnet. In der Replay-Phase werden die Testfälle abgespielt und die erzielten Ergebnisse mit den erwarteten Ergebnissen aus der Capture-Phase verglichen. Eine Deltafunktion wird angewendet, um relevante Differenzen zwischen den erzielten und erwarteten Ergebnissen zu ermitteln. Aktuelle Entwicklungen zur Unterstützung von Capture&Replay Werkzeugen für USW werden in [HaKL07] beschrieben:

- *Deltafunktionen*: Das neu entwickelte Rahmenwerk für Deltafunktionen erfüllt zwei Anforderungen, die für das Testen von USW besonders wichtig sind. Zum einen kann das Rahmenwerk gezielt Änderungen auf Webseiten erkennen und darstellen. Zum anderen haben die Deltafunktionen ein „Gedächtnis“, so dass Folgefehler und wiederkehrende Fehler erkannt und protokolliert werden. Sie vergleichen IST- mit SOLL-Werten und merken sich genau die Elemente, die sich dabei ändern dürfen.
- *Datenbankreset*: Bestehende Capture&Replay Werkzeuge haben keine Unterstützung zur Wiederherstellung des Testdatenbankzustands nach Ausführung eines Testlaufes. Diese Aufgabe muss manuell programmiert werden, was einen hohen Aufwand bedeutet, sehr fehleranfällig ist und dazu führen kann, dass einfache Tests eine Laufzeit von Monaten benötigen.
- *Parallelisierung:* Um eine große Abdeckung durch Testen zu erreichen, müssen Tests parallel ausgeführt werden, was in der heutigen Praxis wiederum nur manuell geschieht. Die entwickelten, allgemeinen Techniken wurden prototypisch in ein industrielles Capture&Replay Werkzeug namens HTTrace integriert und Industriepartnern zur Verfügung gestellt.

5.2.4 Anwendungsbeispiel (Fortführung)

Bild 5 führt das Beispiel aus Kapitel 4 weiter fort und zeigt eine logische Testspezifikation des o.g. Beschaffungsprozess zur Aufdeckung von Kontrollflussfehlern (vgl. logische Testebene). Zur Erstellung wurde das Transformationsmuster „Teste die korrekte Implementierung des Anwendungsfalls Kontrollfluss“ angewendet, welches als Testbasis einen Anwendungsfall benutzt. In einem ersten Schritt wird manuell das Aktivitätsdiagramm nach den im Muster vorgegebenen Schritten erstellt. Hierbei wird zuerst der „normale“ Ablauf modelliert, wobei Aktor- und Systemschritte (A, S) im Anwendungsfall auf Aktivitäten abgebildet werden. In einem Teilschritt werden mehrere Aktionen eines Schrittes auf mehrere Aktivitäten abgebildet. In einem Folgeschritt werden alternative Abläufe durch das Einfügen von Entscheidungsknoten modelliert. Schließlich erfolgt die Modellierung der Ausnahmen. Zusätzlich werden Vor- und Nachbedingungen sowie der Kontrollfluss spezifiziert. Aus solch einer logischen Testspezifikation werden (semi-automatisch) konkrete Testfälle, je nach Abdeckungsmuster erzeugt. Hierbei werden Pfade von Schritten durch das Aktivitätsdiagramm zu Testfällen kompiliert. Die Testfälle (Pfade im Aktivitätsdiagramm) werden mit Hilfe eines Capture&Replay Werkzeuges erstellt und ausgeführt. Die logischen Testdaten entstehen dadurch, dass die Bedingungen von Verzweigungen sowie Vor- und Nachbedingung im Aktivitätsdiagramm auf den Pfaden aufgesammelt werden, mit dem Ziel Testdaten und Testdatenbanken (mittels RQP) erzeugen zu können.

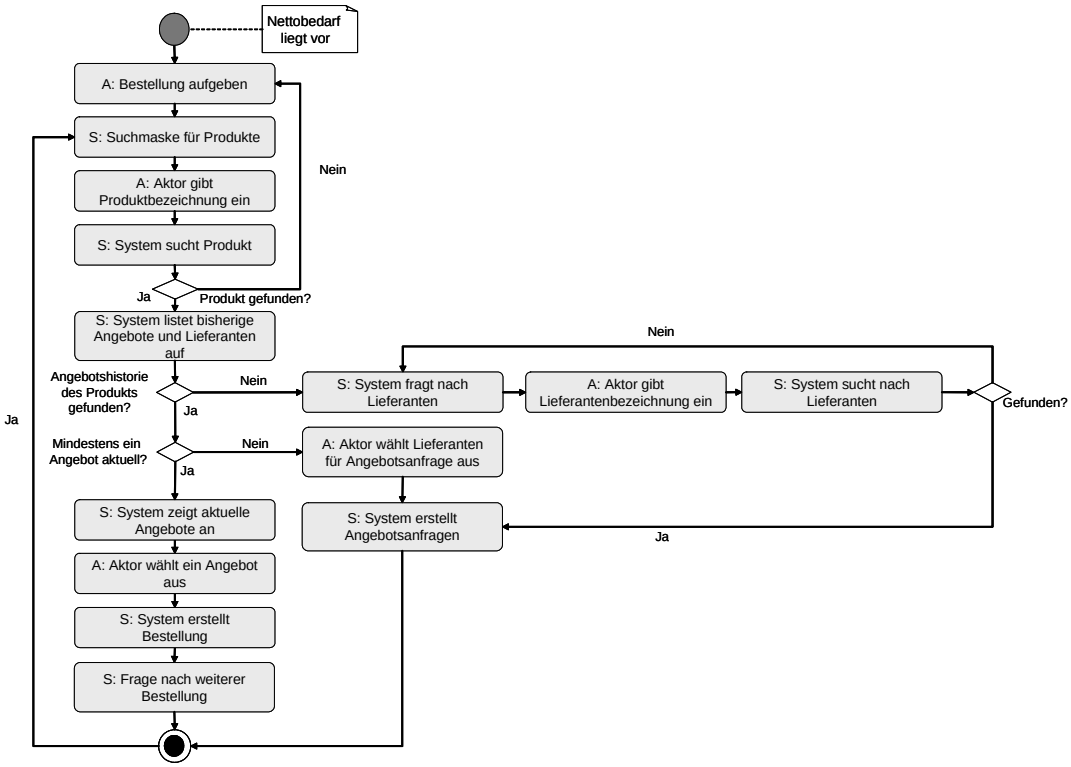

Bild 5 Logische Testspezifikation eines Beschaffungsprozesses

6 Zusammenfassung

Die SIKOSA-Methodik liefert einen Beitrag zur Industrialisierung der SE und fokussiert insbesondere die vertikale Integration der Prozess- und USW-Anforderungsanalyse. Sie verkettet diese mit dem logisch darauf aufbauenden Softwaretest. Als zentrales Element wurde ein gemeinsames Metamodell der Methoden ProQAM, TORE und MOQARE vorgestellt, das die Überführung von funktionalen und nicht-funktionalen Prozessanforderungen in funktionale und nicht-funktionale USW-Anforderungen ermöglicht. Insbesondere die Adressierung der nicht-funktionalen Anforderungen grenzt sie von [ScHa00; Nora04; KoLi06b] ab. Dadurch erhöht der Lösungsansatz die Validität der USW gegenüber den Benutzeranforderungen. Die Ergebnisse der SIKOSA-Methodik wurden prototypisch implementiert und können genau auf die Bedürfnisse angepasst werden. Die entwickelten Werkzeuge ergänzen bestehende industrielle Werkzeuge (konkret: UML Modellierungs- und Capture&Replay Werkzeuge) und werden in diese integriert.

Weiterer Forschungsbedarf leitet sich insbesondere an der Schnittstelle zwischen nicht-funktionalen Prozess- und USW-Anforderungen sowie der Anforderungspriorisierung ab, da ein effektives und effizientes Vorgehen dort nicht nur wichtig für Projektmanagemententscheidungen (in Bezug auf Projektumfangsverhandlungen oder Priorisierung von Maßnahmen, etc.) ist, sondern dies auch die Auswahl der zu realisierenden Anforderungen, die Aufteilung von Testaufwenden, Entscheidungen beim Architekturdesign, Releaseplanung unmittelbar beeinflusst. Ansätze zur Auflösung von Priorisierungsproblemen lassen sich möglicherweise im Goal-oriented RE finden (GORE). Darüber hinaus stellt die Evaluation von Testfällen und Testdaten bei einer Änderung der Software ein ebenfalls noch ungelöstes Forschungsproblem dar.

Anmerkungen

[1] XACML ist eine vom OASIS-Konsortium vorgeschlagene Architektur für Zugriffskontrollmechanismen in Service-orientierten Systemen [Oasi03].

7 Literatur

[Alex03] *Alexander, I.:* Misuse Cases Help to Elict Non-Functional Requirements. In: Computing & Control Engineering Journal 14 (2003) 1, S. 40-45.

[BFOW06] *Bieser, T.; Fürstenau, H.; Otto, S.; Weiß, D.:* Requirements Engineering. In: *Kirn, S.; Herzog, O.; Lockemann, P.; Spaniol, O. (Hrsg.):* Multiagent Engineering. Theory and Applications in Enterprises. Heidelberg 2006, S. 359-381.

[BiKL06] *Binnig, C.; Kossmann, D.; Lo, R.:* Testing database applications. In: Proceedings of the 25th ACM SIGMOD International Conference on Management of Data / Principles of Database Systems. Chicago 2006, S. 739-741.

[BiKL07] *Binnig, C.; Kossmann, D.; Lo, R.:* Reverse Query Processing. In: Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE 2007). Istanbul 2007, im Erscheinen.

[BoMo99] *Bosch, J; Molin, P.:* Software Architecture Design: Evaluation and Transformation. In: Proceedings of the IEEE Conference and Workshop on Engineering of Computer-Based Systems. Potsdam 1999, S. 4-10.

[BrCh05] *Bruno, N.; Chaudhuri, S.:* Flexible database generators. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB). Trondheim 2005, S. 1097-1107.

[BrEn01] *Brücher, H.; Endl, R.:* Erweiterung von UML zur geschäftsregelorientierten Prozessmodellierung. Heidelberg 2002.

[CDFD04] *Chays, D.; Deng, Y.; Frankl, P. G.; Dan, S.; Vokolos, F. I.; Weyuker E. J.:* An AGENDA for testing relational database applications. In: Software Testing, verification and reliability 14 (2004) 1, S. 17-44.

[Cock01] *Cockburn, A.*: Writing effective use cases. Boston 2001.

[GHJV95] *Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.:* Design Patterns, Elements of Reusable Object-Oriented Software. Reading 1995.

[GrSh03] *Greenfield, J.; Short, K.:* Software Factories. Assembling Applications with Patterns, Models, Frameworks and Tools. In: Proceedings of the 18th Annual ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2003). Anaheim 2003, S. 16-27.

[GrFe99] *Graham, D.; Fewster, M.:* Software Test Automation. Effective Use of Test Execution Tools. Harlow 1999.

[HaKL07] *Haftmann, F.; Kossmann, D.; Lo, E.:* A Framework for Efficient Regression Tests of Database Application Systems. In: The VLDB Journal 16 (2007) 1, S. 145-164.

[HePa06] *Herrmann, A; Paech, B.:* MOQARE = Misuse-oriented Quality Requirements Engineering - Über den Nutzen von Bedrohungsszenarien beim RE von Qualitätsanforderungen. In: Softwaretechnik-Trends 26 (2006) 1, S. 13-14.

[HeRP06] *Herrmann, A.; Rückert, J.; Paech, B.*: Exploring the Interoperability of Web Services using MOQARE. In: Proceedings of the 1st International Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems. Bordeaux 2006, S. 199-211.

[HMPR04] *Hevner, A., R.; March, S.,M.; Park, J.; Ram, S.:* Design Science in Information Systems Research. In: MIS Quarterly 28 (2004) 1, S. 75-105.

[IlPa06a] *Illes, T.; Paech, B.*: From V to U or: How Can We Bridge the V-Gap Between Requirements and Test? In: Software & Systems Quality Conferences. Düsseldorf 2006.

[IlPa06b] *Illes, T.; Paech, B.:* An Analysis of Use Case Based Testing Approaches Based on a Defect Taxonomy. In: *IFIP International Federation for Information Processing:* Software Engineering Techniques: Design for Quality (SET2006), Boston 2006, S. 211-222.

[ISO1991] *International Organization for Standardization (ISO):* ISO/IEC 9126: Information technology - Software product evaluation - Quality characteristics and guidelines for their use. Genf 1991.

[Jark99] *Jarke, M.:* Scenarios for Modelling. In: Communications of the ACM 42 (1999) 1, S. 47-48.

[KoLi06a] *Korherr, B.; List, B.:* "Aligning Business Processes and Software Connecting the UML 2 Profile for Event Driven Process Chains with Use Cases and Components". In: Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE´06). Luxembourg 2006, S 19-22.

[KoLi06b] *Korherr, B.; List, B.:* A UML 2 Profile for Event Driven Process Chains. A UML 2 Profile for Business Process Modelling. In: Proceedings of the 1st International Workshop on Best Practices of UML at the 24th International Conference on Conceptual Modeling (ER 2005), Klagenfurt 2005.

[KPMK06] *Kirn, S.; Paech, B.; Müller, G.; Kossmann, D.:* Die SIKOSA-Methodik. Arbeitsbericht. Hohenheim, Heidelberg, Freiburg, Zürich 2006.

[LaRV99] *Lassing, N.; Rijsenbrij, D.; van Vliet, H.:* Towards a Broader View on Software Architecture Analysis of Flexibility. In: Proceedings of the 6th Asia-Pacific Software Engineering Conference (APSEC ´99). Takamatsu 1999, S. 238-245.

[Nora04] *Noran, O.S.:* Business Modelling: UML vs. IDEF. http://www.cit.gu.edu.au/~noran/Docs/UMLvsIDEF.pdf, Abruf am 2006-10-12.

[Oasi03] *Organization for the Advancement of Structured Information Standards (OASIS)*: eXtensible Access Control Markup Language (XACML) Version 1.0. http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf, Abruf am 2007-02-20.

[PaKo03] *Paech, B.; Kohler, K.:* Task-driven Requirements in object-oriented Development. In: *Leite, J.; Doorn, J. (Hrsg.):* Perspectives on Requirements Engineering. Norwell 2003, S. 45-67.

[SaSA06] *Sackmann, S., Strüker, J., Accorsi, R.:* Personalization in privacy-aware highly dynamic systems. Communications of the ACM 49 (2006) 9, S. 32-38.

[ScHa00] *Scheer, A.; Habermann, F.:* Making ERP a success. In: Communications of the ACM 43 (2000) 3, S.57-61.

[Sche97] *Scheer, A.-W.:* Wirtschaftsinformatik. Referenzmodelle für industrielle Geschäftsprozesse. 7. Aufl., Berlin 1997.

[Sche02] *Scheer, A.-W.:* ARIS - Vom Geschäftsprozess zum Anwendungssystem. 4. Aufl., Berlin 2002.

[SiOp01] *Sindre, G.; Opdahl, A.L.:* Templates for Misuse Case Description. In: Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2001). Interlaken 2001, S. 125-136.

[Spil00] *Spillner, A.:* From V-Model to W-Model – Establishing the Whole Test Process. In: Proceedings of the 4th Conference on Quality Engineering in Software Technology (Conquest 2000). Nürnberg 2000, S. 222-231.

[Stra96] *Strahringer, S.:* Metamodellierung als Instrument des Methodenvergleichs. Eine Evaluierung am Beispiel objektorientierter Analysemethoden. Aachen 1996.

[Sysi06] *Sysiphus.* http://sysiphus.informatik.tu-muenchen.de, Abruf am 2006-12-08.

[WGRE04] *Working Group on Requirements Engineering Patterns (WGREP)*: Final Report, Requirements Engineering Patterns - An Approach to Capturing and Exchanging Requirements Engineering Experience. http://repare.desy.de/Repare/pdf/Report%20WG%20RE%20Pattern.pdf, Abruf am 2007-02-22.

[WoMü06] *Wohlgemuth, S.; Müller, G.:* Privacy with Delegation of Rights by Identity Management. In: Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS 2006), Freiburg 2006, S. 175-199.

[WPJH98] *Weidenhaupt, K.; Pohl, K.; Jarke, M.; Haumer, P.:* Scenario Usage in System Development: A Report on Current Practice. In: IEEE Software 15 (1998) 2, S. 34-45.